МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
**ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**
Институт компьютерных технологий и информационной безопасности
Кафедра интеллектуальных и многопроцессорных систем

ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ
**НИЦ СУПЕР-ЭВМ И НЕЙРОКОМПЬЮТЕРОВ**

---

**На правах рукописи**

**ДОРДОПУЛО АЛЕКСЕЙ ИГОРЕВИЧ**

# ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ТЕХНОЛОГИИ РЕСУРСОНЕЗАВИСИМОГО ПРОГРАММИРОВАНИЯ ГИБРИДНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Специальность: 2.3.5 - Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей, технические науки

**ДИССЕРТАЦИЯ НА СОИСКАНИЕ УЧЕНОЙ СТЕПЕНИ ДОКТОРА ТЕХНИЧЕСКИХ НАУК**

Научный консультант:
доктор технических наук,
профессор **И.И. Левин**

Таганрог – 2023 г.

## СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

| | |
|---|---|
| CLB | Configurable logic block (конфигурируемый логический блок) |
| CPU | Central processing unit (центральный процессор) |
| DSP | Digital signal processing (обработка сигналов) |
| FPGA | Field programmable gate array (ПЛИС) |
| GPGPU | General-purpose computing for graphics processing unit (неспециализированные вычисления на графических процессорах) |
| GPU | Graphics processing unit (графический ускоритель) |
| HDL | Hardware definition language (язык описания аппаратуры) |
| HLS | High-level synthesis (средства высокоуровневого синтеза) |
| LUT | Look-up table (таблица преобразования) |
| АПКС | Абсолютно параллельная кадровая структура |
| БО | Базовая операция |
| БП | Базовый подграф |
| БПФ | Быстрое преобразование Фурье |
| БС | Блок согласования |
| БСИ | Блок согласования интервала |
| ВД | Выбор данных |
| ВМ | Вычислительный модуль |
| ГВС | Гибридная вычислительная система |
| ДК | Динамическая коммутация |
| ИГЗ | Информационный граф задачи |
| КРП | Контроллер распределенной памяти |
| МБП | Минимальный базовый подграф |
| МВС | Многопроцессорная вычислительная система |
| МинКС | Минимальная кадровая структура |
| ОБПФ | Обратное преобразование Фурье |
| ПВМ | Плата вычислительного модуля |
| ПД | Подготовка данных |
| ПЛИС | Программируемая логическая интегральная схема |
| РВБ | Реконфигурируемый вычислительный блок |
| РВС | Реконфигурируемая вычислительная система |
| СЛАУ | Система линейных алгебраических уравнений |
| СПОВ | Структурно-процедурная организация вычислений |
| ТИП | Технология индуктивных программ |
| ТРНП | Технология ресурсонезависимого программирования |
| ФП | Функциональный подграф |

СОДЕРЖАНИЕ

Стр.

ВВЕДЕНИЕ ……. 7

1. ВЫЧИСЛИТЕЛЬНЫЕ АРХИТЕКТУРЫ И ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ СОВРЕМЕННЫХ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ . 37

1.1. Тенденции развития современных универсальных процессоров, графических ускорителей и сопроцессоров …….. 37

1.2. Тенденции развития современных реконфигурируемых вычислителей …….. 50

1.3. Анализ вычислительных архитектур для построения гибридных вычислительных систем …… 57

1.4. Технологии программирования и автоматического распараллеливания мультипроцедурных вычислений …… 62

1.5. Технологии программирования графических процессоров ……. 73

1.6. Технологии программирования ПЛИС ……. 80

1.7. Технология структурно-процедурных вычислений для многокристальных РВС и их программирование на языке высокого уровня COLAMO …… 84

1.8. Анализ рассмотренных технологий и разработка принципов ресурсонезависимого программирования гибридных вычислительных систем ……..… 90

1.9. Выводы к главе 1 …….. 97

2. ПРЕДСТАВЛЕНИЕ ВЫЧИСЛЕНИЙ ДЛЯ ГИБРИДНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ……. 101

2.1. Единая информационная модель вычислений для гибридных вычислительных систем …… 102

2.2. Преобразование информационной модели вычислений в абсолютно параллельную кадровую структуру …….... 108
2.3. Ресурсонезависимое представление абсолютно параллельной кадровой структуры для гибридных вычислительных систем с помощью параметризации доступным вычислительным ресурсом … 117
2.4. Преобразование кадровой структуры к доступному вычислительному ресурсу гибридной вычислительной системы …….. 124
2.5. Правила преобразования кадровых структур к архитектуре и конфигурации гибридной вычислительной системы в пространстве возможных реализаций …….. 133
2.6. Изменение направления движения в точках бифуркации по значению критического ресурса кадровой структуры …… 139
2.7. Принципы выбора преобразований кадровой структуры к архитектуре и конфигурации гибридной вычислительной системы …. 145
2.8. Выводы к главе 2 …….. 148

3. МЕТОДЫ И МЕХАНИЗМЫ ПРЕОБРАЗОВАНИЯ ПАРАМЕТРИЗУЕМОЙ РЕСУРСОМ КАДРОВОЙ СТРУКТУРЫ К ЦЕЛЕВЫМ АРХИТЕКТУРАМ ГИБРИДНОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ 152
3.1 Механизмы преобразования кадровых структур к целевой архитектуре гибридной вычислительной системы …….. 152
3.1.1. Редукция по числу подграфов $R^N$ …… 158
3.1.2. Редукция числу устройств $R^{Op}$ …… 159
3.1.3. Редукция кадровых структур к микро-кадрам …….. 164
3.1.4. Редукция по разрядности $R^{\rho}$ ……... 175
3.1.5. Редукции по интервалу обработки данных $R^I$ и частоте $R^{Freq}$ ... 176
3.2. Теоретические основы применения редукций для преобразования кадровой структуры …….. 179

3.3. Правила применения редукций для преобразования кадровой структуры к рациональным для целевых архитектур ГВС вариантам организации параллельных вычислений …….. 185
3.4. Методика преобразования кадровой структуры к рациональному варианту организации параллельных вычислений …….. 190
3.5. Алгоритмы преобразования кадровой структуры к рациональному варианту организации параллельных вычислений на доступном аппаратном ресурсе гибридной вычислительной системы 199
3.5.1 Алгоритм преобразования кадровой структуры к структурной реализации вычислений ……. 199
3.5.2 Алгоритм преобразования кадровой структуры к процедурной реализации вычислений ……. 202
3.6. Выводы к главе 3 …….. 203

4. МЕТОДЫ И АЛГОРИТМЫ ПРЕОБРАЗОВАНИЯ ПРИКЛАДНЫХ ЗАДАЧ С ИНФОРМАЦИОННО- ЗАВИСИМЫМИ ПОДЗАДАЧАМИ 206
4.1. Принципы преобразования информационно-зависимых подзадач 206
4.2. Метод и алгоритм преобразования информационно-зависимых подзадач к рациональному варианту организации параллельных вычислений…….. 219
4.3. Метод преобразования «флагмана» и множества информационно-зависимых «катеров» к рациональному варианту организации параллельных вычислений …… 233
4.4. Методы преобразования подзадач для особых случаев информационной зависимости …… 243
4.4.1. Информационная зависимость «флагман»-«катер»-«софлагман» 244
4.4.2. Информационная зависимость «взаимодействующие равные» ... 250
4.5. Выводы к главе 4 …….. 253

5. СРЕДСТВА ТЕХНОЛОГИИ РЕСУРСОНЕЗАВИСИМОГО ПРОГРАММИРОВАНИЯ ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ 256
5.1. Структура программных средств технологии ресурсонезависимого программирования ……... 256
5.2. Описание программы «Прокруст» …… 260
5.3. Описание программы «Щелкунчик» …… 269
5.4. Преобразование прикладных задач с к целевым архитектурам ГВС 278
5.4.1 Задачи с единственным информационно-независимым базовым подграфом …… 284
5.4.2 Задачи с единственным информационно-зависимым базовым подграфом ……. 288
5.4.3 Пример задачи с зависимостью «флагман»-«катер» …… 291
5.4.4 Пример задачи с зависимостью «флагман»-{катер} …… 295
5.4.5 Пример задачи с зависимостью «флагман»-«катер»-«софлагман» 300
5.4.6 Пример задачи с зависимостью «взаимодействующие равные» …. 303
5.5. Возможные направления дальнейших исследований …… 306
5.6. Выводы к главе 5 ……... 313
ЗАКЛЮЧЕНИЕ……. 316
СПИСОК БИБЛИОГРАФИЧЕСКИХ ИСТОЧНИКОВ ……... 322
ПРИЛОЖЕНИЕ 1. Редукционные преобразования типовых кадровых структур ……... 349
ПРИЛОЖЕНИЕ 2. Акты внедрения результатов работы …… 385
ПРИЛОЖЕНИЕ 3. Результаты портации прикладных задач программными средствами технологии ресурсонезависимого программирования …… 399

# ВВЕДЕНИЕ

Современные исследования в прикладных областях химии и медицины, гео- и астрофизики, баллистики и космонавтики, экономики и социологии тесно связаны с моделированием и высокопроизводительными вычислениями [1] (high performance computing, HPC), которые входят в число приоритетных направлений развития науки и техники большинства ведущих стран мира [2]. Более 30 лет рост производительности вычислительных систем обеспечивался достижениями микроэлектроники в соответствии с эмпирическим законом Мура[3] об удвоении интеграции микросхем, что позволяло каждые 1,5 года повышать быстродействие устройств улучшением технологии их производства и не замечать существенное различие между теоретическим и достигаемым на практике ускорением при решении прикладных задач [2]. В настоящее время микроэлектронные технологии практически достигли физических пределов миниатюризации транзисторов, что существенно замедлило привычные темпы роста производительности [4]. Исчерпание технологических возможностей повышения производительности наряду с известными с конца 70-х годов 20 века проблемами фон-Неймановской процессорной архитектуры привели к кризису HPC и вывели долго игнорируемую проблему эффективности реализации вычислений на первый план.

Эффективность реализации задачи в вычислительной системе можно определить как отношение *реальной*, достигаемой при ее решении, производительности к теоретически возможной, *пиковой.* Эффективность, выраженная в процентах, характеризует соответствие архитектуры вычислительной системы структуре решаемой задачи и, для большинства многопроцессорных вычислительных систем (МВС), не превышает 10-15% для реальных прикладных задач. С превышением числом вычислительных устройств в системе некоторого порога, эффективность реализации задач в МВС снижается еще больше – до нескольких процентов, а для ряда задач – даже до долей процента.

Одним из способов преодоления кризиса высокопроизводительных вычислений и повышения эффективности реализации задач стала адаптация МВС к структуре задачи с помощью специализированных вычислительных архитектур [5], таких как графические ускорители и программируемые логические интегральные схемы [6] (ПЛИС). Объединение вычислительных узлов различных архитектур в гибридную вычислительную систему (ГВС) позволило добиться роста реальной производительности и увеличения эффективности реализации вычислений. Именно с созданием гибридных [5, 7] или конвергентных [8, 9] вычислительных архитектур, содержащих процессоры и/или графические ускорители и/или ПЛИС, связывают преодоление кризиса HPC как ведущие ученые в области параллельного программирования (T.Sterling, J. Dongarra и др.), так и крупнейшие производители элементной базы (Intel, AMD, Nvidia, Xilinx). Подтверждением правильности выбранного пути служат последние редакции списка самых производительных мировых суперкомпьютеров TOP500[10], в котором первые позиции уже несколько лет принадлежат гибридным и гетерогенным вычислительным системам, содержащим миллионы узлов, например, Summit[11], Fugaku[12], TaihuLight[13] и др.

В то же время простое объединение или совмещение узлов различных архитектур в гибридную (гетерогенную) вычислительную систему само по себе не обеспечивает теоретически ожидаемого роста эффективности, т.к. вычисления организуются по «федеративному» [14-17] принципу и выполняются только той частью системы, структура узлов которой лучше всего соответствует решаемой задаче, причем простаивающие узлы не используются для расчетов, снижая эффективность и реальную производительность системы. Для повышения эффективности реализации вычислений в ГВС при решении прикладной задачи узлы разных архитектур должны применяться совместно.

Модели представления вычислений, языки и технологии программирования специфичны для каждого типа узла гибридной вычислительной системы, поэтому их совместное использование для решения

задачи требует от программиста обширных и глубоких знаний не только особенностей архитектур и технологий их программирования, но и экспертных навыков согласования фрагментов вычислений, выполняющихся на узлах разных архитектур. Согласование фрагментов вычислений в узлах разных типов необходимо не только при создании, но и при каждой корректировке программы в связи с изменением конфигурации гибридной вычислительной системы. Изменение конфигурации доступного вычислительного ресурса в связи с недоступностью одного или группы узлов из-за отказа оборудования и/или коммутационной системы – обычная процедура при эксплуатации любой вычислительной системы, приводящая к трансформации параллельной программы в связи с масштабированием числа доступных узлов. Корректировка программы для каждой изменившейся конфигурации (портация) требует от программиста усилий для согласования вычислений в разных узлах ГВС и может занимать длительное время, в ряде случаев сравнимое с временем разработки новой программы. Портация (от англ. **porting**, портирование или перенос) понимается как адаптация некоторой программы или её части, с тем чтобы она работала в другой среде с максимальным сохранением её пользовательских свойств. Портируемость или переносимость является свойством программного кода или исполняемого модуля программы. В первом случае портируемость обеспечивается с помощью общего (как язык программирования C) или промежуточного (как аппаратно-независимый байт-код Java) представления кода для несовместимых аппаратных платформ. Во втором случае используются специальные программы, преобразующие аппаратные инструкции одной архитектуры в инструкции другой архитектуры преобразованием исполняемого модуля или при исполнении «на лету». Для вычислительных задач портацию будем понимать как преобразование вычислений с созданием исполняемого модуля выбранных архитектур ГВС. Трудоемкость разработки и портации программы зависит от количества различных вычислительных архитектур в ГВС, структуры задачи и числа используемых технологий программирования, поскольку значительные различия между ними не

позволяют использовать одну единую технологию для эффективного программирования всех узлов ГВС.

Значительные различия в технологиях программирования различных вычислительных архитектур, проявляющиеся уже на уровне моделей вычислений, не позволяют использовать какую-либо одну из них для эффективного программирования всех вычислительных архитектур ГВС, поэтому необходима разработка теоретических основ организации вычислений в едином контуре, учитывающих как информационные зависимости и внутренний параллелизм задач, так и особенности различных вычислительных архитектур конвергентной мультипроблемно-ориентированной вычислительной системы. С учетом многообразия существующих и потенциально возможных вычислительных архитектур, разрабатываемые теоретические основы должны обеспечивать адаптацию вычислений к различному по составу и конфигурации аппаратному ресурсу гибридной вычислительной системы. Поэтому целесообразно, чтобы аппаратный ресурс ГВС был параметром представления задачи, с изменением которого параллельные вычисления с помощью научно-обоснованных методов могут быть преобразованы (портированы) к заданной архитектуре и конфигурации гибридной вычислительной системы. Параметризация представления задачи аппаратным ресурсом позволит создавать независимые от архитектуры и конфигурации ГВС или *ресурсонезависимые* параллельные программы, рационально использующие особенности вычислительных архитектур. Теоретические основы организации вычислений в гибридных вычислительных системах как совокупность научных понятий, выявленных закономерностей и вытекающих из них принципов, достаточных для совместного использования узлов различных типов при решении прикладной задачи являются фундаментом для разработки методов и инструментальных средств программирования гибридных вычислительных систем, совокупность которых можно объединить в технологию программирования[18, 19], обеспечивающую, с одной стороны, рациональную (эффективную) реализацию вычислений на различных узлах и любом доступном ресурсе ГВС,

а с другой – предоставляющую программисту достаточный уровень удобства разработки и отладки прикладных программ.

Принимая во внимание многообразие известных и разрабатываемых вычислительных архитектур и моделей вычислений, наиболее перспективным способом создания ресурсонезависимых программ представляется создание единой для различных вычислительных архитектур модели и формы представления вычислений, неотъемлемым параметром которой является доступный аппаратный ресурс и разработка методов, алгоритмов и инструментальных средств для преобразования (переноса или портации) прикладных программ с сохранением их информационных и семантических свойств к заданной текущей конфигурации гибридной вычислительной системы.

Поэтому актуальна разработка теоретических основ *ресурсонезависимого* программирования гибридных вычислительных систем – новых научных понятий, методологических принципов, положений и выявленных закономерностей, позволяющих описать единую для различных архитектур модель вычислений и преобразовать ее с помощью формализованных методов и алгоритмов в модель реализации для гетерогенного вычислительного ресурса, содержащего вычислительные узлы на основе процессоров, графических ускорителей и ПЛИС. Совокупность теоретических основ ресурсонезависимого программирования, методов реализации вычислений и инструментальных программных средств позволит создать новую технологию программирования гибридных вычислительных систем, обеспечивающую сбалансированное решение задачи для заранее неопределенного аппаратного ресурса ГВС из различного числа узлов разных архитектур, объединенных в единый вычислительный ресурс. Формализация методов и приемов создаваемой технологии ресурсонезависимого программирования позволит автоматизировать и сократить время портации (преобразования, трансформации или переносимости) прикладных задач для всего многообразия архитектур и конфигураций ГВС при заданном уровне эффективности (реальной производительности).

**Объектом исследования** являются технологии программирования высокопроизводительных вычислительных систем.

**Целью** диссертационной работы является сокращение времени портации параллельных программ для гибридных вычислительных систем.

**Предметом исследования** являются методы и средства технологии ресурсонезависимого программирования гибридных высокопроизводительных вычислительных систем.

Актуальная **научная проблема,** решаемая в диссертации, – разработка теоретических основ, методов и инструментальных программных средств ресурсонезависимого программирования, обеспечивающих сокращение времени портации прикладных задач для различных архитектур и конфигураций гибридных вычислительных систем при заданном уровне реальной производительности.

Прикладным результатом решения поставленной научной проблемы является существенное сокращение времени и стоимости портации параллельных программ на различные гибридные вычислительные системы для выполнения фундаментальных и прикладных исследований в науке и технике.

В общей сформулированной **научной проблеме** можно выделить несколько частных задач, для решения которых необходимо:

1 Провести анализ современных вычислительных архитектур и тенденций их развития, определить их ограничения при эффективной реализации прикладных задач различных классов для высокопроизводительных гетерогенных вычислительных систем.

2 Проанализировать известные методы и технологии программирования высокопроизводительных вычислительных систем, сформулировать требования и принципы программирования конвергентных архитектур гибридных вычислительных систем.

3 Разработать теоретические основы ресурсонезависимого программирования: единую форму вычислений, параметром которой является

доступный аппаратный ресурс и методы ее преобразования к различным архитектурам гибридной вычислительной системы.

4 Разработать методику портации задач к рациональной реализации на заранее неопределенном вычислительном ресурсе гибридной вычислительной системы.

5 Разработать методы и алгоритмы портации прикладных задач различных классов, содержащих связанные информационной зависимостью подзадачи с различной степенью параллелизма.

6 Разработать инструментальные программные средства технологии ресурсонезависимого программирования, реализующие разработанные принципы, методы и алгоритмы портации прикладных задач на архитектуру и конфигурацию гибридной вычислительной системы при заданном уровне реальной производительности.

7 Провести апробацию технологии ресурсонезависимого программирования при решении прикладных задач различных предметных областей и сравнить результаты с известными решениями.

**Степень разработанности темы**

Исследованиями, в той или иной мере связанными с тематикой поставленных в диссертации задач, занимались такие известные ученые, как Э. Дейкстра, Ч. Хоар, Р. Хокни, В. Хейдлер, Дж. Донгарра, Т.Стерлинг (за рубежом) и В.С. Бурцев, В.Г. Хорошевский, В.В. и Вл.В. Воеводин, А.И. Легалов, А.О. Лацис, В.А. Крюков, В.А. Вальковский, Б.Я. Штейнберг, А.В. Каляев , И.А. Каляев, И.И. Левин и многие другие исследователи, внесшие значительный вклад в теорию и практику программирования высокопроизводительных вычислительных систем.

**Методы исследований.** При проведении исследований были использованы: теория многопроцессорных вычислительных систем, комплексов и сетей, элементы теории графов, элементы теории множеств и мультимножеств. Теоретические исследования подтверждены экспериментами на ряде действующих образцов вычислительных систем, а

также реализацией программных средств ресурсонезависимого программирования прикладных задач в интересах ряда предприятий и организаций.

В работе получены следующие **основные новые научные результаты**:

1. Модель параллельных вычислений для гибридных вычислительных систем на основе информационного графа задачи, **отличающаяся** от известных единой для различных вычислительных архитектур формой описания вычислений в виде кадровой структуры.

2. **Новое** представление кадровой структуры с использованием вычислительного ресурса как параметра параллельно-конвейерной реализации задачи на заранее неопределенном ресурсе гибридной вычислительной системы.

3. **Новые** принципы преобразования параметризованной аппаратным ресурсом кадровой структуры к архитектуре и конфигурации гибридной вычислительной системы с помощью редукции производительности.

4. Теоремы о применении методов редукции производительности для преобразования кадровой структуры к архитектуре и конфигурации гибридной вычислительной системы.

5. Метод преобразования кадровой структуры и сокращения аппаратных затрат с помощью редукции производительности, **отличающийся** использованием доступного аппаратного ресурса гибридной вычислительной системы как параметра портации.

6. **Новый** метод преобразования и сокращения аппаратных затрат кадровой структуры, отличающийся синтезом новой формы организации вычислений – микро-кадров, выполняющих вычисления на ограниченном доступном аппаратном ресурсе с меньшей степенью параллелизма.

7. Метод преобразования задач, содержащих связанные информационной зависимостью подзадачи с разной степенью параллелизма, **отличающийся** от известных согласованной редукцией производительности

различных по вычислительной трудоемкости кадровых структур с синтезом сбалансированного по интервалу обработки данных решения.

8. Теоретические основы технологии ресурсонезависимого программирования гибридных вычислительных систем, объединяющие совокупность предложенных научных понятий и положений, выявленных закономерностей и принципов, разработанных методов и алгоритмов портации в достаточную для синтеза рационального решения задачи методику.

9. **Новые** алгоритмы функционирования инструментальных программных средств технологии ресурсонезависимого программирования гибридных вычислительных систем: алгоритм преобразования параметризованной ресурсом кадровой структуры прикладной задачи к целевой архитектуре ГВС процессором «Прокруст» и алгоритм редукции производительности выделенного фрагмента задачи в заданное коэффициентом редукции число раз при нехватке аппаратного ресурса процессором «Щелкунчик».

Новизна полученных результатов подтверждается отсутствием аналогов, опубликованных в открытых источниках.

**Теоретическая значимость научных результатов.** Совокупность предложенных научных понятий и положений, выявленных закономерностей и вытекающих из них принципов, сформулированных и доказанных теорем, составляющих теоретические основы, позволили разработать методы, алгоритмы и инструментальные средства, объединенные в технологию ресурсонезависимого программирования гибридных вычислительных систем, обеспечивающую сокращение времени портации прикладных задач при заданном уровне реальной производительности. Разработанные теоретические основы технологии ресурсонезависимого программирования гибридных вычислительных систем можно квалифицировать как новое научное достижение, а разработанные методы и средства – как новые научно обоснованные технические решения, внедрение которых вносит значительный вклад в развитие страны.

**Практическая значимость работы.** Все разработанные методы преобразования кадровой структуры формализованы и адаптированы для автоматического применения. Совокупность предложенных теоретических основ, разработанных методов, алгоритмов и программных средств позволила создать технологию ресурсонезависимого программирования гибридных вычислительных систем, апробированную при решении ряда прикладных задач, содержащих связанные информационной зависимостью подзадачи с разной степенью параллелизма. Применение разработанных инструментальных программных средств технологии ресурсонезависимого программирования для портации прикладных задач математической физики, символьной и цифровой обработки сигналов с различными видами информационной зависимости позволяет значительно (в 20-45 раз) сократить время портации ресурсонезависимых программ на различные архитектуры и конфигурации гибридных вычислительных систем с обеспечением реальной производительности не ниже заданного уровня. Практическая значимость работы и полученные результаты эффективности портации ресурсонезависимых программ подтверждаются актами внедрения результатов в академических, научно-исследовательских и промышленных организациях. Внедрение разработанной технологии ресурсонезависимого программирования гибридных вычислительных систем открывает возможность создания автоматического распараллеливающего компилятора для систем с распределенной памятью, позволяющего сократить время портации программ на процедурных языках программирования в ресурсонезависимые параллельно-конвейерные решения для гибридных вычислительных систем.

**Положения, выдвигаемые для защиты:**

1. Современные технологии программирования высокопроизводительных вычислительных систем не обеспечивают единое представление параллельных вычислений для процессоров, графических ускорителей и ПЛИС из-за различий моделей вычислений и архитектурной специфики, что исключает автоматическую портацию прикладных программ

на различные архитектуры и конфигурации гибридных вычислительных систем.

2. Единое представление параллельных вычислений для различных вычислительных архитектур может быть построено на основе информационного графа задачи, отображаемого в максимально параллельную кадровую структуру, которая в зависимости от доступного аппаратного ресурса ГВС может быть преобразована в структуру с меньшей степенью параллелизма.

3. Преобразование параметризованной аппаратным ресурсом кадровой структуры к заданной архитектуре и конфигурации гибридной вычислительной системы реализуемо формальными редукционными преобразованиями сокращения производительности и аппаратных затрат, число вариантов которых существенно меньше числа вариантов, анализируемых распараллеливающим компилятором для систем с распределенной памятью.

4. Совокупность разработанных формальных методов редукции производительности и аппаратных затрат, алгоритмов и инструментальных программных средств преобразования параметризованной аппаратным ресурсом кадровой структуры, составляющих основу технологии ресурсонезависимого программирования, сокращает время портации параллельно-конвейерных программ на заданную архитектуру и конфигурацию гибридной вычислительной системы, в том числе для задач, содержащих информационно-зависимые подзадачи с различной степенью параллелизма.

**Результаты, выдвигаемые для защиты:**

1. Теоремы о применении редукции производительности для преобразования кадровой структуры к архитектуре и конфигурации гибридной вычислительной системы.

2. Метод преобразования параметризуемой аппаратным ресурсом кадровой структуры к архитектуре и конфигурации гибридной

вычислительной системы на основе редукции производительности, сокращающий ее аппаратные затраты.

3. Метод преобразования кадровой структуры к новой форме организации вычислений – микро-кадрам, выполняющим вычисления с меньшей степенью параллелизма на ограниченном подмножестве функциональных устройств и информационных каналов.

4. Метод преобразования задач, содержащих связанные информационной зависимостью подзадачи с разной степенью параллелизма, на основе согласованной редукции производительности различных по вычислительной трудоемкости кадровых структур с синтезом сбалансированного по интервалу обработки данных решения.

**Степень достоверности и апробация результатов.**

**Достоверность и обоснованность** научных исследований подтверждается теоретическим обоснованием, основанным на использовании классического математического аппарата, и практически полным совпадением теоретических результатов с результатами экспериментальных исследований разработанных методов и алгоритмов созданными инструментальными программными средствами при портации прикладных задач различных предметных областей. Корректность разработанных методов преобразования кадровой структуры и функциональность инструментальных программных средств технологии ресурсонезависимого программирования подтверждены успешной портацией задач с наиболее распространенными видами информационной зависимости.

**Реализация результатов работы**

Основные результаты диссертационных исследований были использованы более чем в 25 научно-исследовательских и опытно-конструкторских работ, среди которых можно отметить следующие:

1. ОКР «Разработка технологии создания высокопроизводительных модульно-наращиваемых многопроцессорных вычислительных систем с программируемой архитектурой на основе реконфигурируемой элементной

базы» (НИИ МВС ТРТУ, г/к № 02.447.11.1007 от 06 июля 2005 г., шифр “Медведь”, № гос. рег. 0122.0510630, 2005-2006).

2. НИР «Создание модульно-наращиваемых многопроцессорных систем (МНМС) на основе реконфигурируемой элементной базы и программных средств поддержки масштабируемых программ для решения задач обработки информации и управления в реальном времени на различных конфигурациях МНМС, в том числе при деградации вычислительного ресурса» (НИИ МВС ТРТУ, шифр “Триада”, № ГР 01.2.00611470, 2006-2008).

3. ОКР «Создание семейства высокопроизводительных многопроцессорных вычислительных систем с динамически перестраиваемой архитектурой на основе реконфигурируемой элементной базы и их математического обеспечения для решения вычислительно трудоемких задач» (НИИ МВС ЮФУ, шифр “Большая Медведица”, № ГР 01.2007 05707, 2007-2008).

4. НИР «Разработка теоретических основ построения гетерогенных многопроцессорных вычислительных систем с программируемой архитектурой» (НИИ МВС ЮФУ, шифр “Карнавал”, № ГР 01.2007 02728, 2007-2010).

5. НИР «Разработка и исследование принципов построения и методов программирования реконфигурируемых гетерогенных вычислительных систем с открытой масштабируемой архитектурой» (НИИ МВС ЮФУ, шифр НШ-3289.2010.9, № ГР 01201155173, 2011).

6. ОКР «Разработка реконфигурируемой вычислительной системы РВС-7 и организация на ее основе производства реконфигурируемых вычислительных систем с производительностью до $10^{15}$ операций в секунду в одностоечном конструктиве 47U» (НИИ МВС ЮФУ, шифр 2011-2.7-527-008-001 «Плеяда», № ГР И111102120521, 2011-2013).

7. НИР «Разработка и исследование технологии создания ресурсонезависимого прикладного программного обеспечения высокопроизводительных вычислительных систем гибридного типа» (НИИ МВС ЮФУ, шифр «Русалка», № ГР 114101540025, 2014-2016).

8. НИОКР «Разработка нового поколения программного обеспечения РВС для решения ресурсоемких научно-технических задач различных предметных областей», (НИЦ СЭ и НК, шифр «Поколение», 2016-2022).

**Апробация результатов работы**

Основные результаты диссертационной работы были представлены на ряде международных и всероссийских тематических конференциях, среди которых можно выделить следующие:

**в России:**

- серия международных научных школ «Высокопроизводительные вычислительные системы» (ВПВС), 2004, 2005, 2007, 2011 гг.;
- серия Всероссийских научных конференций «Научный сервис в сети Интернет», г. Новороссийск, 2006-2014 гг.;
- серия Международных научных конференций «Параллельные вычислительные технологии» (ПАВТ), 2011-2021 гг.;
- серия международных научно-практических конференций «Искусственный Интеллект. Интеллектуальные и многопроцессорные системы» (ИИ), 2004, 2006, 2008, 2010 гг.;
- международные научно-технические конференции «Многопроцессорные вычислительные и управляющие системы» (МВУС), 2007 г., 2009 г.;
- серия международных научных конференций «Суперкомпьютерные системы и их применение» (SSA), 2008, 2010, 2012 гг.;
- международные научно-технические конференции «Суперкомпьютерные технологии» (СКТ), 2010, 2012, 2014, 2016, 2018, 2020 гг.;
- международные конференции «Parallel Computing Technologies» (PaCT), 2011, 2015, 2019, 2021 гг.;
- международные конференции «Суперкомпьютерные дни в России», 2015-2021 гг.;

- Всероссийские мультиконференции по проблемам управления (МКПУ), 2013, 2015, 2017, 2019, 2021 гг.;
- национальные Суперкомпьютерные Форумы (НСКФ), 2012-2019 гг.,

**за рубежом:**

- научно-практическая конференция молодых ученых и специалистов «Технологии высокопроизводительных вычислений и компьютерного моделирования», University of Amsterdam, г. Амстердам, Нидерланды, 2012 г.;
- Science and Information Conference (SAI), 2013, United Kingdom;
- IFAC Programmable Devices and Embedded Systems (PDES), 2013, Czech Republic;
- 18th International Conference on Circuits, Systems, Communications and Computers (CSCC 2014), 2014, Greece;
- 6th European Conference of COMPUTER SCIENCE (ECCS '15), 2015, Italy;
- 5th International Conference on Informatics, Electronics & Vision (ICIEV), 2016, Bangladesh;
- 14th IFAC International Conference on Programmable Devices and Embedded Systems (PDES), 2016, Czech Republic;
- 5th International Young Scientist Conference on Computational Science «Procedia Computer Science», 2016, Poland.

**Наиболее важными публикациями по теме работы являются:**

По теме работы опубликовано более 170 работ, из которых 29 индексируются в международных базах наукометрических данных Web of Science и Scopus, 36 работ из списка ВАК РФ, получено 15 свидетельств об официальной регистрации программ для ЭВМ.. Наиболее важными публикациями, посвященными тематике диссертационного исследования, являются:

1. Левин, И. И. К вопросу об автоматическом создании параллельных прикладных программ для реконфигурируемых вычислительных систем / И. И. Левин, А. И. Дордопуло // Вычислительные технологии. – 2020. – Т. 25, № 1. – С. 66-81. – DOI 10.25743/ICT.2020.25.1.005. (**ВАК РФ**).

2. Дордопуло, А. И. Применение методов редукции производительности для сокращения числа анализируемых вариантов параллельной программы / А. И. Дордопуло // Вестник компьютерных и информационных технологий. – 2019. – № 9(183). – С. 43-49. – DOI 10.14489/vkit.2019.09.pp.043-049. (**ВАК РФ**).

3. Архитектурно-независимая программа быстрого преобразования Фурье на языке программирования SET@L / И. И. Левин, А. И. Дордопуло, И. В. Писаренко, А. К. Мельников // Вестник Рязанского государственного радиотехнического университета. – 2019. – № 68. – С. 28-36. – DOI 10.21667/1995-4565-2019-68-2-28-36. (**ВАК РФ**).

4. Hybrid computer system programming technology with adaptation and scaling of calculations / A. A. Gulenok, A. I. Dordopulo, I. I. Levin, V. A. Gudkov // Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. – 2017. – Vol. 6, No. 1. – P. 73-86. – DOI 10.14529/cmse170105. (**ВАК РФ**).

5. Программирование вычислительных систем гибридного типа на языке программирования COLAMO / А. И. Дордопуло, И. И. Левин, И. А. Каляев [и др.] // Известия ЮФУ. Технические науки. – 2016. – № 11(184). – С. 39-54. – DOI 10.18522/2311-3103-2016-11-3954. (**ВАК РФ**).

6. Параллельно-конвейерная форма программы для программирования вычислительных систем гибридного типа / А. И. Дордопуло, И. И. Левин, И. А. Каляев [и др.] // Вестник Уфимского государственного авиационного технического университета. – 2016. – Т. 20, № 3(73). – С. 122-128. (**ВАК РФ**).

7. Технология программирования вычислительных систем гибридного типа / И. А. Каляев, А. И. Дордопуло, И. И. Левин [и др.] // Вычислительные технологии. – 2016. – Т. 21, № 3. – С. 33-44. (**ВАК РФ**).

8. Решение задач с существенно-переменной интенсивностью потоков данных на реконфигурируемых вычислительных системах / И. И. Левин, Д. А. Сорокин, А. К. Мельников, А. И. Дордопуло // Вестник компьютерных и информационных технологий. – 2012. – № 2(92). – С. 49-56. (**ВАК РФ**).

9. Высокопроизводительные многопроцессорные системы с реконфигурируемой архитектурой для цифровой обработки сигналов / И. А. Каляев, И. И. Левин, Е. А. Семерников, А. И. Дордопуло // Вестник Концерна ПВО Алмаз-Антей. – 2011. – № 2(6). – С. 88-104. (**ВАК РФ**).

10. Левин, И. И. Ресурсонезависимое программирование многопроцессорных систем / И. И. Левин, А. И. Дордопуло, В. А. Гудков // Известия ТРТУ. – 2006. – № 9-1(64). – С. 174-178. (**ВАК РФ**).

11. High-Level Synthesis of Scalable Solutions from C-Programs for Reconfigurable Computer Systems / A. I. Dordopulo, I. I. Levin, V. A. Gudkov, A. A. Gulenok // Lecture Notes in Computer Science. – 2021. – Vol. 12942 LNCS. – P. 88-102. – DOI 10.1007/978-3-030-86359-3_7. (**Scopus**)

12. Dordopulo, A. I. Performance Reduction for Automatic Development of Parallel Applications for Reconfigurable Computer Systems / A. I. Dordopulo, I. I. Levin // Supercomputing Frontiers and Innovations. – 2020. – Vol. 7, No. 2. – P. 4-23. – DOI 10.14529/jsfi200201. (**Scopus**)

13. Software Development Tools for FPGA-Based Reconfigurable Systems Programming / I. Levin, V. Gudkov, G. Yevstafiyev [et al.] // Communications in Computer and Information Science. – 2019. – Vol. 1129. – P. 625-640. – DOI 10.1007/978-3-030-36592-9_51. (**Scopus**)

14. High-performance reconfigurable computer systems with immersion cooling / I. Levin, A. Dordopulo, A. Fedorov, Y. Doronchenko // Communications

in Computer and Information Science. – 2018. – Vol. 910. – P. 62-76. – DOI 10.1007/978-3-319-99673-8_5. (**Scopus**)

15. Развитие отечественных многокристальных реконфигурируемых вычислительных систем: от воздушного к жидкостному охлаждению / И. А. Каляев, А. И. Дордопуло, И. И. Левин, А. М. Федоров // Труды СПИИРАН. – 2017. – № 1(50). – С. 5-31. – DOI 10.15622/sp.50.1. (**Scopus**)

16. Immersion liquid cooling FPGA-based reconfigurable computer system / I. I. Levin, Y. I. Doronchenko, M. K. Raskladkin [et al.] // IFAC-PapersOnLine. – 2016. – Vol. 49, No. 25. – P. 366-371. – DOI 10.1016/j.ifacol.2016.12.070. (**Scopus**)

17. Porting of parallel applications to reconfigurable computer systems with various architectures and configurations / A. I. Dordopulo, V. B. Kovalenko, V. A. Gudkov, L. M. Slasten // 2016 5th International Conference on Informatics, Electronics and Vision, ICIEV 2016. – 2016. – P. 1122-1127. – DOI 10.1109/ICIEV.2016.7760174. (**Scopus**)

18. Parallel Application Porting by Means of Soft-Architectures / A. I. Dordopulo, V. A. Gudkov, A. A. Gulenok [et al.] // IFAC-PapersOnLine. – 2016. – Vol. 49, No. 25. – P. 372-377. – DOI 10.1016/j.ifacol.2016.12.071. (**Scopus**)

19. FPGA-based reconfigurable computer systems for digital image processing / I. I. Levin, I. A. Kalyaev, V. A. Gudkov [et al.] // International Journal of Circuits, Systems and Signal Processing. – 2015. – Vol. 9. – P. 27-32.- Режим доступа: http://www.naun.org/main/NAUN/circuitssystemssignal/2015/a082005-028.pdf (дата обращения 24.04.2023). (**Scopus**)

20. FPGA-based reconfigurable computer systems / I. A. Kalyaev, I. I. Levin, A. I. Dordopulo, L. M. Slasten // Proceedings of 2013 Science and Information Conference, SAI 2013. – 2013. – P. 148-155.- Режим доступа: https://ieeexplore.ieee.org/document/6661730 (дата обращения 24.04.2023). (**Scopus**).

**Структура и краткое содержание диссертации.**

Диссертация состоит из введения, пяти глав, в которых изложены решения поставленных частных научных задач, заключения,

библиографического списка и приложений. Содержание работы изложено на 355 страницах, работа содержит 87 рисунков и 17 таблиц, список использованной литературы из 248 наименований.

**Во введении** обоснована актуальность работы, дана ее общая характеристика, сформулированы цель и задачи исследования.

В первой главе рассмотрены вычислительные архитектуры и технологии программирования современных высокопроизводительных систем. Проанализированы тенденции развития современных процессоров, графических ускорителей, специализированных вычислителей и программируемых логических интегральных схем. Показано, что процессоры фон-неймановской архитектуры находятся в глубоком системном кризисе, вызванном стенами памяти, энергетики, параллелизма на уровне команд и ментальности. Графические ускорители и специализированные ускорители на основе процессорной архитектуры во многом унаследовали известные проблемы процессоров, что существенно ограничивает класс решаемых задач слабосвязанными задачами и неизбежно приведет к аналогу системного кризиса процессоров через 4-5 лет. Реконфигурируемые вычислители на основе ПЛИС за счет адаптации архитектуры вычислительного устройства к структуре решаемой задачи позволяют при 10-кратно меньших частотах получать 100-1000-кратный выигрыш в скорости решения широкого спектра задач, но сложность их программирования и низкая (менее 50%) эффективность решения задач, содержащих большое число вложенных условных операторов, рекурсивных вызовов и структур с обратной связью, не позволяют считать их единственной перспективной вычислительной архитектурой. Поскольку ни одна из массовых вычислительных архитектур (процессоры, графические ускорители, ПЛИС) в полной мере не отвечает современным требованиям по эффективности реализации различных классов прикладных задач, наиболее перспективной высокопроизводительной вычислительной архитектурой является гибридная (гетерогенная или конвергентная), содержащая ПЛИС, процессоры и графические ускорители. При этом, наряду с требованиями эффективного использования всех

преимуществ разных вычислительных архитектур, необходимо обеспечить программиста привычным для процессорных систем уровнем удобства разработки и отладки прикладных программ, что требует адаптации известных технологий программирования или создания новых высокоуровневых инструментальных средств программирования гибридных вычислительных систем.

Проведен обзор наиболее известных существующих технологий и средств программирования высокопроизводительных вычислительных систем. Рассмотрены методы организации мультипроцедурных вычислений на традиционных процессорах (MPI, OpenMP, HOPMA, DVM, PVM), технологии программирования графических ускорителей (CUDA, OpenACC и др.), технологии программирования ПЛИС (MitrionC, COLAMO) и ГВС (OpenCL). Показано, что на сегодняшний день ни одна из рассмотренных технологий программирования в полной мере не удовлетворяет требованиям эффективной реализации прикладных задач на гетерогенных вычислительных узлах ГВС, поскольку уже на уровне модели вычислений все технологии зависят от архитектуры вычислительных узлов. Поэтому для программирования гибридных вычислительных систем различных конфигураций, содержащих заранее неопределенное число вычислительных узлов различных архитектур необходимы новые теоретические основы (принципы, положения и методы), позволяющие описать единую для различных архитектур модель вычислений, которую с помощью формализованных методов и алгоритмов можно преобразовать в модель реализации для гетерогенного вычислительного ресурса, содержащего вычислительные узлы на основе процессоров, графических ускорителей и ПЛИС. Разрабатываемые теоретические основы не должны зависеть от аппаратного ресурса и архитектуры вычислительных узлов и обеспечивать возможность плавного перехода от характерной для процессоров и графических ускорителей процедурной организации вычислений к свойственной ПЛИС структурно-процедурной организации вычислений с

синтезом сбалансированной вычислительной структуры в зависимости от доступного аппаратного ресурса ГВС.

Для разграничения понятий архитектуры ГВС как вычислительной системы и вычислительной архитектуры ее компонентов (процессоров, графических ускорителей и ПЛИС) как принятого способа организации вычислений, введен термин «целевая архитектура», означающий вычислительную архитектуру компонента гибридной вычислительной системы. На основе проведенного анализа технологий программирования, наиболее перспективным способом создания программ для гибридных вычислительных систем различных конфигураций, содержащих заранее неопределенное число вычислительных узлов различных архитектур, представляется разработка единой модели и формы представления вычислений, параметром которой является доступный аппаратный ресурс, и разработка методов, алгоритмов и инструментальных средств для преобразования (портирования или портации) единой формы к целевым архитектурам текущей конфигурации ГВС. Сформулированы основные принципы для ресурсонезависимого программирования гибридных вычислительных систем:

– вычисления прикладной задачи должны быть представлены в единой для различных целевых архитектур форме, обеспечивающей возможность реализации любого фрагмента задачи на любой из целевых архитектур гибридной вычислительной системы;

– преобразование представленных в единой форме одновременных вычислений к целевым архитектурам ГВС выполняется на основе параметра – архитектуры и конфигурации аппаратного ресурса;

– методы преобразования вычислений к целевым архитектурам ГВС для каждого фрагмента задачи должны обеспечивать возможность изменения способа организации вычислений, частоты, числа устройств и других характеристик аппаратной реализации;

– выбор наиболее рационального варианта реализации прикладной задачи или ее фрагмента на каждой из целевых архитектур ГВС должен

осуществляться с учетом характеристик решения – латентности, интервала обработки данных, разрядности, числа устройств и т.д.;

– методы преобразования вычислений должны учитывать информационную структуру прикладной задачи для подзадач с разной степенью параллелизма, чтобы синтезируемое для параметрически заданного аппаратного ресурса решение было согласованно между вычислительными узлами разных целевых архитектур;

– методы преобразования вычислений должны обеспечивать сбалансированное рациональное решение как при увеличении, так и при сокращении доступного аппаратного ресурса гибридной вычислительной системы.

Предложенные принципы позволяют разработать теоретические основы – новые научные понятия, методологические принципы и положения – ресурсонезависимого программирования гибридных вычислительных систем.

Вторая глава описывает способы представления вычислений, принципы и правила их преобразования к целевым архитектурам гибридной вычислительной системы. Для представления одновременных вычислений для различных целевых архитектур предложена единая форма на основе информационного графа задачи, которая не зависит от архитектуры и доступного вычислительного ресурса гибридной вычислительной системы. Информационный граф содержит все операции обработки данных задачи и, в соответствии с первым принципом ресурсонезависимого программирования, позволяет реализовать любой фрагмент задачи в любой целевой архитектуре гибридной вычислительной системы. Информационный граф инвариантен по отношению к различным способам организации вычислений с точностью до топологических перестановок, что позволяет минимизировать число вариантов параллельного представления, различающихся только порядком выполнения операций. Доказано, что информационный граф задачи может служить единой моделью для представления параллельных вычислений в гибридной вычислительной системе.

Для преобразования представленных в единой форме одновременных вычислений к любой из целевых архитектур ГВС предложен переход от информационного графа к максимально параллельной кадровой структуре, отражающей характеристики реализации фрагмента вычислений на узле целевой архитектуры (латентность, интервал обработки данных, разрядность выполняемых операций и др.). Показано, что при параметризации вычислительным ресурсом максимально параллельная кадровая структура представима в ресурсонезависимой форме. Показано, что при подстановке абсолютных значений доступного ресурса в параметры архитектуры и конфигурации гибридной вычислительной системы кадровая структура может быть преобразована в информационно-эквивалентную, но с меньшей степенью параллелизма, реализуемую на доступном аппаратном ресурсе.

Для преобразования максимально параллельной кадровой структуры к архитектуре и конфигурации гибридной вычислительной системы предложено использование секвентирования – сокращение параметров ее параллелизма. Показано, что параметры параллелизма кадровой структуры представимы пространством возможных реализаций кадровых структур, измерениями которого являются информационно-независимые подграфы (слои), информационно-зависимые подграфы (итерации), команды (устройства), разрядность, интервал обработки данных и время решения задачи. Предложено представление преобразования максимально параллельной кадровой структуры к архитектуре и конфигурации гибридной вычислительной системы в виде движения в пространстве возможных реализаций, траекторией которого является кусочно-непрерывный линейный сплайн. Показано, что каждый отрезок траектории формируется с помощью сокращения одной из характеристик параллелизма кадровой структуры, приводящего к сокращению аппаратных затрат. Предложена двухэтапная схема преобразования максимально параллельной кадровой структуры к рациональному для целевых архитектур гибридной вычислительной системы способу организации параллельных вычислений, на первом из которых кадровая структура достигает области доступного аппаратного ресурса, а на

втором в этой области ищется наиболее рациональная по времени решения задачи кадровая структура. Сформулированы *локальная* цель преобразования как достижение кадровой структурой области параметрически заданного доступного аппаратного ресурса и *глобальная* как поиск рациональной по времени решения задачи кадровой структуры в достигнутой области. Показано, что эти преобразования могут быть формализованы и автоматизированы.

Показано, что требуемый для реализации максимально параллельной кадровой структуры аппаратный ресурс, как правило, превышает доступный ресурс целевой архитектуры сразу по нескольким компонентам. Сформулированы правила движения в пространстве возможных реализаций, минимизирующие число преобразований кадровой структуры для достижения локальной цели. В качестве основного правила движения предложена минимизация критического ресурса – компонента целевой архитектуры с наибольшим удалением от области доступного аппаратного ресурса гибридной вычислительной системы. Рассмотрены возможные варианты изменения критического ресурса кадровой структуры при движении в пространстве возможных реализаций. Показано, что смена направления движения в пространстве возможных реализаций происходит после сокращения параметров параллелизма кадровой структуры. Для описания смены направления движения предложен термин *точка выбора альтернатив*, в которой происходит перерасчет и возможное качественное изменение критического ресурса кадровой структуры. Показано, что характер изменения критического ресурса в точке выбора альтернатив определяется текущими значениями параметров параллелизма кадровой структуры, доступного аппаратного ресурса гибридной вычислительной системы, выбранного преобразования и коэффициента сокращения аппаратных затрат. Показано, что число точек выбора альтернатив определяется суммарным числом компонент всех целевых архитектур гибридной вычислительной системы. Получена оценка числа точек выбора альтернатив для достижения локальной цели для целевой архитектуры ПЛИС. Предложены критерии выбора

предпочтительных направлений и параметров движения в пространстве возможных реализаций кадровых структур. Для преобразования параметров параллелизма кадровой структуры при достижении локальной цели в пространстве возможных реализаций предложен механизм редукции производительности и аппаратных затрат.

В третьей главе описываются методы и механизмы преобразования параметризуемой ресурсом кадровой структуры к целевым архитектурам гибридной вычислительной системы. В качестве возможных механизмов преобразования кадровых структур к целевой архитектуре гибридной вычислительной системы предложены используемые инженерами-схемотехниками для ручного согласования фрагментов задач частные эвристики, сокращающие параметры параллелизма. Показано, что известные частные преобразования – редукция по числу подграфов, редукция по числу устройств, редукция по разрядности, редукции по интервалу обработки данных и частоте – обеспечивают информационную эквивалентность результатов вычислений. Каждое из рассматриваемых редукционных преобразований формализовано и обобщено для сокращения параметров параллелизма параметризуемой ресурсом, а не конкретной, кадровой структуры. Для редукций по числу устройств и разрядности предложена новая форма организации вычислений – микро-кадр, существенным отличием которой от кадровой структуры является совмещение нескольких операций в одном вычислительном устройстве, приводящее к увеличению интервала обработки данных, но обеспечивающее единый и сбалансированный темп обработки данных без использования дополнительной памяти для хранения промежуточных результатов обработки входного потока данных. Предложены 5 различных вариантов синтеза микро-кадров для кадровой структуры. Приведены теоретические оценки целесообразности синтеза микро-кадров и применения редукции по числу устройств в зависимости от значения коэффициента редукции. На основе полученных теоретических оценок определен диапазон эффективности микро-кадров для целевой архитектуры ПЛИС на примере семейства Xilinx Virtex UltraScale.

Показано, что для минимизации числа преобразований кадровой структуры при достижении локальной цели в пространстве возможных реализаций, редукционные преобразования необходимо выполнять в комплексе, а не по отдельности. Для комплексного применения редукций сформулированы и доказаны теоремы о влиянии последовательности их применения на характеристики параллелизма и требуемый аппаратный ресурс при преобразовании кадровой структуры. Сформулированы следствия из доказанных теорем, позволяющие минимизировать необходимое для достижения локальной цели число редукционных преобразований кадровой структуры, что обеспечивает сокращение времени портации. На базе доказанных теорем и следствий из них разработаны правила применения редукций для преобразования кадровой структуры к рациональным для целевых архитектур ГВС вариантам организации параллельных вычислений.

На основе предложенных правил разработана методика автоматизированного преобразования кадровой структуры к рациональному варианту организации параллельных вычислений для целевых архитектур ГВС. В рамках предложенной методики разработаны алгоритмы преобразования кадровой структуры к структурной и процедурной формам организации вычислений для синтеза рационального варианта организации параллельных вычислений на доступном аппаратном ресурсе гибридной вычислительной системы.

Проведена оценка максимального числа необходимых редукционных преобразований кадровой структуры. Показано, что полученное оценочное значение существенно меньше числа вариантов, анализируемого распараллеливающим компилятором для систем с распределенной памятью. Доказано, что комплексное применение методов редукции производительности и аппаратных затрат позволяет существенно сократить число анализируемых вариантов по сравнению с распараллеливающим компилятором для систем с распределенной памятью

В четвертой главе рассмотрены методы и алгоритмы преобразования прикладных задач с информационно- зависимыми подзадачами. Показано, что

существует обширный класс задач, содержащих не одну, а несколько подзадач, выделенных на логическом уровне и связанных между собой информационной зависимостью. Предложено описание кадровой структуры таких задач в виде совокупности кадровых подструктур – со своим «базовым подграфом», числом слоев и итераций, числом устройств, разрядностью и интервалом обработки данных. Показано, что для преобразования максимально параллельной кадровой структуры такой задачи к рациональному варианту организации параллельных вычислений необходимо согласование степени параллелизма и интервалов обработки данных между подзадачами. Для описания таких подзадач, по аналогии с базовым подграфом, введено понятие функционального подграфа информационного графа задачи со своим «базовым подграфом», выделенными функциями отображения по слоям и итерациям, числом слоев и итераций, числом устройств, разрядностью и интервалом обработки данных. Показано, что вычислительную трудоемкость и занимаемый каждой подзадачей аппаратный ресурс можно оценить с помощью числа вычислительных устройств в максимально параллельной кадровой структуре. Предложено использование этой характеристики для ранжирования информационно-связанных подзадач по объему требуемого для их реализации аппаратного ресурса. Для различения подзадач по объему требуемого для их реализации аппаратного ресурса на качественном уровне введены термины «флагман» для наибольшей по числу устройств подзадачи и «катер» для подзадачи с существенно, как минимум на 1 десятичный порядок, меньшим занимаемым максимально параллельной кадровой структурой ресурсом.

Разработаны метод и алгоритм согласованного преобразования информационно-зависимых подзадач, содержащих «флагман» и «катер», к рациональному варианту организации параллельных вычислений. Отдельно рассмотрены часто встречающиеся особые случаи информационной зависимости: «флагман» и множество информационно-зависимых «катеров», «флагман»-«катер»-«софлагман» и «взаимодействующие равные». Для каждого из рассмотренных случаев разработаны метод и алгоритм

преобразования подзадач к рациональному варианту организации параллельных вычислений. Сформулированы и доказаны теоремы о критериях совмещения нескольких «катеров» на одном процедурном узле для рациональной реализации вычислений в целевых архитектурах гибридной вычислительной системы.

В пятой главе представлены разработанные инструментальные программные средства технологии ресурсонезависимого программирования, реализующие разработанные в диссертации методы и алгоритмы, а также приведены результаты их апробации при портации прикладных задач с различными типами информационной зависимости на различные архитектуры и конфигурации реконфигурируемых и гибридных вычислительных систем.

Описана структура инструментальных программных средств технологии ресурсонезависимого программирования, в состав которых входят:

– процессор «Прокруст», преобразующий ресурсонезависимую кадровую структуру прикладной задачи, к целевой архитектуре ГВС с помощью методов и алгоритмов, изложенных в 3-4 главах;

– процессор «Щелкунчик», осуществляющий редукцию производительности выделенного фрагмента в заданное коэффициентом редукции число раз при нехватке аппаратного ресурса в соответствии с методами и алгоритмами, изложенными в 3-4 главах.

Исходной формой представления задачи является ресурсонезависимое представление задачи в виде информационно-связанных параметризуемых ресурсом кадровых структур на языке программирования высокого уровня с неявным описанием параллелизма COLAMO. Подробно описаны все этапы преобразования кадровой структуры разработанными программными средствами технологии ресурсонезависимого программирования, в том числе преобразования кадровой структуры к м-кадрам процессором «Щелкунчик». Приведены примеры апробации разработанных теоретических основ технологии ресурсонезависимого программирования гибридных вычислительных систем и программных средств при решении прикладных

задач, содержащих связанные информационной зависимостью подзадачи с разной степенью параллелизма. Описаны результаты портации типовых задач с наиболее распространенными видами информационной зависимости: «флагман» -«катер», «флагман»-«множество катеров», «флагман» -«катер»-«софлагман», «взаимодействующие равные». Применение разработанных программных средств поддержки технологии ресурсонезависимого программирования для портации прикладных задач математической физики, символьной и цифровой обработки сигналов, содержащих связанные информационной зависимостью подзадачи с разной степенью параллелизма, позволяет существенно (в 20-45 раз) сократить время портации ресурсонезависимых программ на различные архитектуры и конфигурации гибридных вычислительных систем с обеспечением реальной и удельной производительности не ниже заданного уровня. Время портации на новую архитектуру гибридной вычислительной системы – преобразование ресурсонезависимой параллельно-конвейерной формы, масштабирование, согласование потоков данных и синтеза конфигурационных файлов – для рассмотренных прикладных задач с наиболее распространенными видами информационной зависимости не превышает 30 минут.

Полученные результаты проверки работоспособности разработанных в диссертации методов и алгоритмов преобразования для целого ряда задач символьной обработки, линейной алгебры, цифровой обработки сигналов подтверждают приведенные во 2-4 главах теоретические положения и выполненные на их основе расчеты, что позволяет сделать вывод об успешном решении поставленной научной задачи. Практическое использование полученных результатов и их внедрение в практику автоматической портации прикладных задач различных областей для гибридных вычислительных систем позволит сократить время создания рациональных решений с заданным уровнем реальной и удельной производительности до 30 минут.

В заключении диссертации приведены основные выводы, полученные при решении поставленной научной проблемы и ее частных задач.

Сформулирован вывод о достижении цели, а также рекомендации по возможным направлениям дальнейших исследований.

Таким образом, для сформулированной в диссертации актуальной научной проблемы, имеющей важное народно-хозяйственное значение, разработаны теоретические основы, предложены методы решения и созданы реализующие их программные средства, в результате чего разработана новая технология ресурсонезависимого программирования гибридных вычислительных систем. Успешная практическая апробация разработанных теоретических основ и реализующих их программных средств на ряде типовых задач с наиболее распространенными видами информационной зависимости подтверждает корректность разработанных методов технологии ресурсонезависимого программирования и позволяют сделать вывод об успешном решении поставленной научной проблемы.

Внедрение полученных в диссертации результатов позволит сократить сроки и стоимость портации прикладных задач на существующих и перспективных образцах гибридных вычислительных систем, что, помимо прямого экономического эффекта, имеет важное значение для развития критической технологии «Технологии и программное обеспечение распределенных и высокопроизводительных вычислительных систем», утвержденной Указом Президента Российской Федерации от 7 июля 2011 г. №899, и повышения обороноспособности страны.

# 1. ВЫЧИСЛИТЕЛЬНЫЕ АРХИТЕКТУРЫ И ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ СОВРЕМЕННЫХ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Основными компонентами высокопроизводительных систем, содержащих большое количество вычислительных узлов, уже более 40 лет являются процессоры (микропроцессоры). Именно на базе процессоров, в том числе многоядерных, создаются современные многопроцессорные вычислительные системы (МВС), содержащие десятки и сотни тысяч узлов.

## 1.1. Тенденции развития современных универсальных процессоров, графических ускорителей и сопроцессоров

Архитектура процессора как устройства обработки данных, выполняющего с помощью команд управления арифметические и логические операции, была предложена в 40-х годах 20 века для первых ламповых вычислительных машин ЭДВАК [20-22] и ЭНИАК [21]. В этих вычислительных системах обработка данных и команд выполнялась последовательно и циклически, что было вызвано техническими и технологическими ограничениями элементной базы того времени – электрических вакуумных ламп и электромеханических реле. Архитектура получила название по имени одного из разработчиков - Джона фон Неймана [20, 21], а ее дальнейшее развитие привело к созданию в конце 60-х годов 20 века процессоров, а в 70-х годах 20 века – микропроцессоров. Процессор с архитектурой фон-Неймана, структура которого приведена на рис. 1.1, содержит элементы, которые в том или ином виде используются и в современных процессорах: устройство управления, арифметико-логическое устройство, регистровую память и внутреннюю шину. Процессоры выполняли последовательную обработку данных и команд, которые могли храниться в одной памяти (архитектура фон-Неймана или принстонская) или в физически разных микросхемах (гарвардская архитектура) [21, 22, 23].

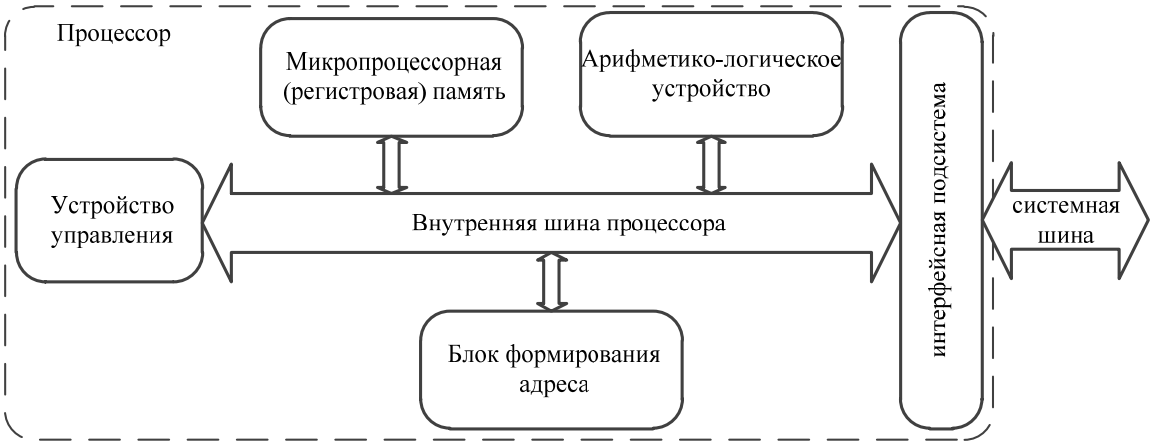



Рис. 1.1. Обобщенная структура процессора

Долгое время производительность процессоров ограничивалась низкой скоростью выполнения арифметических операций, занимавших не менее 20 тактов машинного времени для сложения и не менее 40 тактов для умножения. Зарождающейся мировой компьютерной индустрией были предприняты серьезные усилия, направленные на улучшение и модернизацию как структуры процессора, так и на технологические улучшения элементной базы. Результатом работ в области модернизации обобщенной структуры процессора явилось создание целого ряда усовершенствований: конвейерной [21, 24], суперскалярной [25], CISC- [20, 21, 26, 27], RISC- [21, 26, 27] и MISC-архитектур [21, 28-29], схем и алгоритмов кэширования [30-32], многоядерных процессоров [21, 33]. Для повышения производительности процессора увеличивалось число одновременно работающих элементарных устройств, но архитектура самого процессора принципиально не изменялась, т.е. общая последовательная организация вычислительного процесса сохранялась. Поэтому число устройств процессора, разделяющих один и тот же системный ресурс (например, внутреннюю шину процессора или шину данных), увеличилось, что стало приводить к конфликтам одновременного доступа нескольких его устройств к одному ресурсу (данным и командам в памяти). Эти конфликты выявили принципиальное ограничение архитектуры фон-Неймана, получившее название «бутылочное горло» [34-36] (или «узкое горло») – невозможность одновременного доступа к одному ресурсу. Для

сегментированной оперативной памяти этот конфликт проявился наиболее остро и получил название «стена памяти» (memory wall) [37], являясь существенным фактором, ограничивающим реальную производительность процессоров и МВС на их основе. На рис. 1.2 в виде графиков изменения [38] производительности процессоров и памяти представлена наглядная графическая иллюстрация «стены памяти».

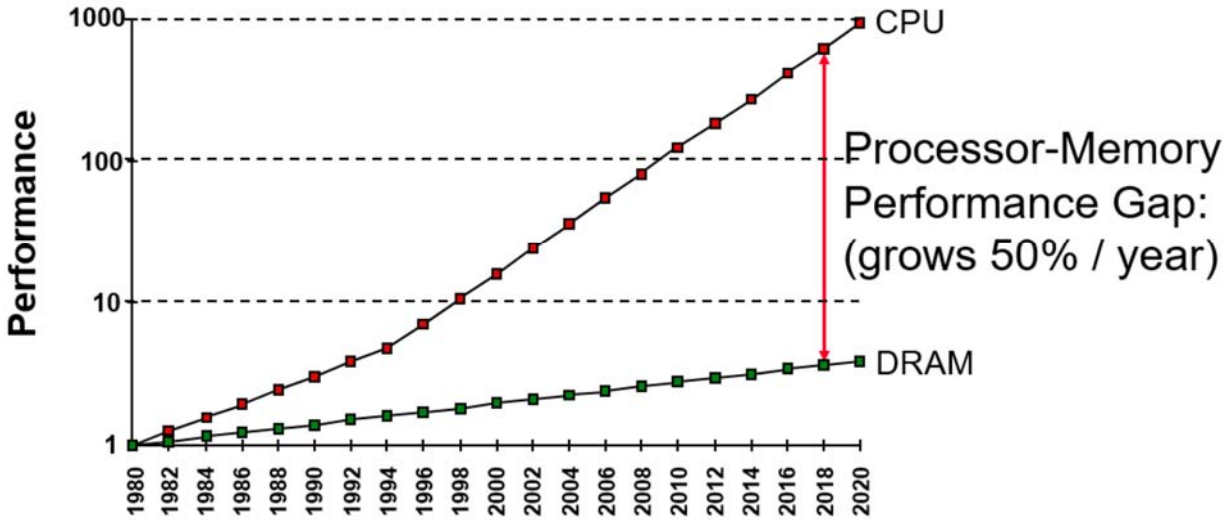



Рис. 1.2. Динамика роста производительности процессоров и памяти

До начала девяностых годов XX века реальная производительность прикладной программы определялась во многом именно скоростью выполнения операций с плавающей запятой, т.к. что оперативная память работала быстрее выполнения операций с плавающей запятой. После точки перегиба (см. рис. 1.2), благодаря конвейеризации данных и команд во внутренних устройствах процессора, скорость выполнения арифметических операций стала возрастать и в настоящее время большинство арифметических операций, в том числе и вспомогательные, выполняются за один такт. Темпы роста скорости доступа к памяти существенно (на несколько порядков) ниже темпов роста производительности процессора, поэтому сейчас реальная производительность во многом зависит от числа, характера обращений и расположения данных в памяти. Например, процессор Intel Core i7 с таковой частотой 3,2 ГГц может генерировать поток обращений к памяти со скоростью

передачи 25,6 миллиардов 64-разрядных данных в секунду, что соответствует полосе пропускания 409,6 Гбайт/с, а пиковая полоса пропускания динамической оперативной памяти составляет только 25 Гбайт/с, что составляет всего лишь 6% от возможностей процессора [39]. Такой существенный разрыв (на десятичный порядок) в пропускных способностях процессора и памяти нивелирует улучшения процессорной архитектуры и требует новых подходов для своего разрешения[40]. Появление многоядерных процессоров, увеличивших требования к полосе пропускания канала процессор-память, еще больше осложнило ситуацию и усугубило проблему «стены памяти».

В течение следующих двух десятилетий (1990 - 2010) повышение быстродействия процессоров обеспечивалось экстенсивным ростом частоты и мощности в соответствии с известным эмпирическим «законом» Гордона Мура, сформулированным им еще в 1965 году. Закон Мура [3, 41] (в разных редакциях) гласит, что количество транзисторов на интегральной схеме удваивается каждые 1,5-2 года (рис.1.3).

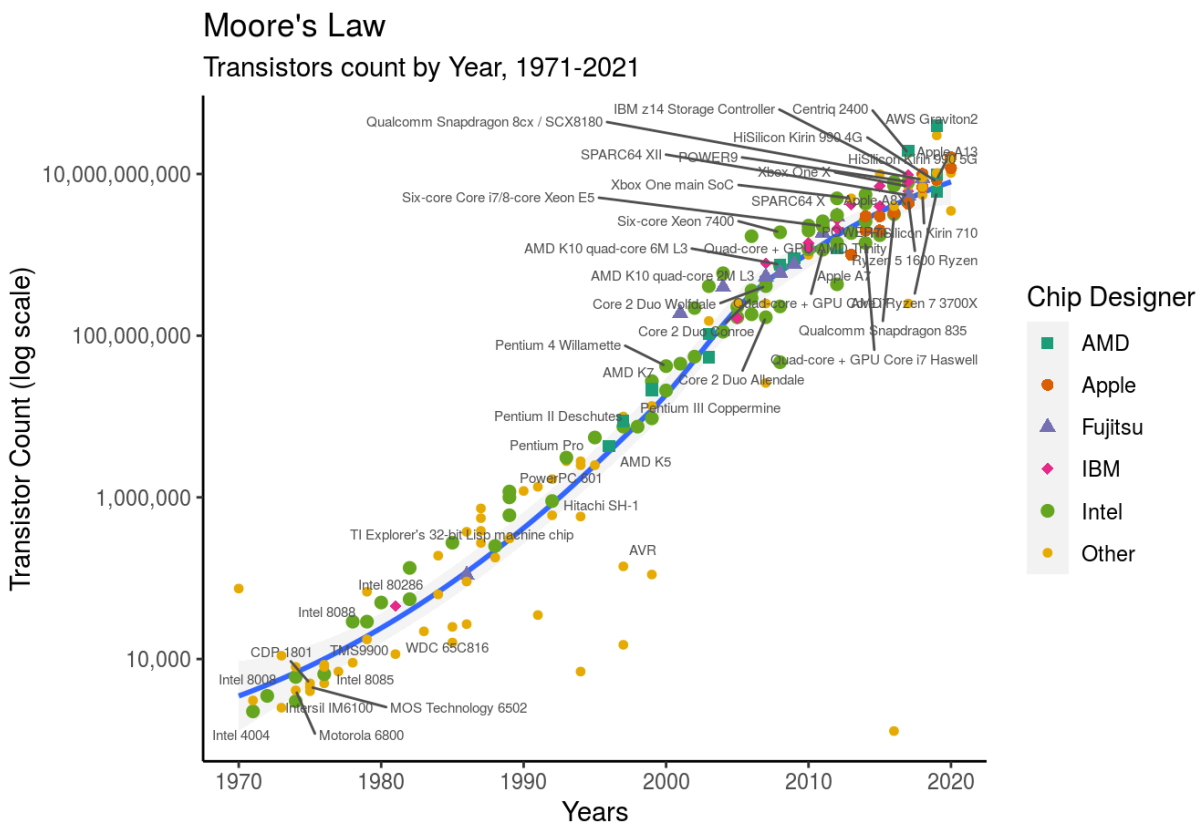



Рис. 1.3. Рост числа транзисторов на кристалле

Постоянное двукратное увеличение объема микросхем позволяло реализовывать большее количество более сложных устройств и ускорить выполнение арифметических операций. Поэтому зачастую увеличение числа транзисторов приравнивали к росту вычислительной мощности процессоров. Действие закона Мура продолжалось довольно длительное время (с 1971 по 2000 год), в течение которого будущее процессоров и вычислительной техники на их основе виделось в достаточно оптимистичном свете: даже несмотря на «стену памяти» производительность процессоров росла «сама собой» за счет перехода на новые технологические нормы производства и потому не требовала существенных архитектурных изменений. В начале 2000-х годов привычные темпы роста производительности замедлились и закон Мура перестал действовать, т.к. ускорение процессоров за счет технологических достижений исчерпало свои внутренние резервы, что подтвердил в 2003 году сам Мур [3]. Причина в том, что на достигнутом уровне технологий изготовления транзисторов эмпирический закон Мура вступил в противоречие с физическим принципом Ландауэра [42, 43], ограничивающим миниатюризацию электронных схем по энергетике. Это стало первым ощутимым свидетельством серьезного кризиса процессоров как вычислительной архитектуры, который невозможно преодолеть изменением его структуры [37].

Попытки разрешения конфликтов одновременного доступа за счет дублирования шин данных и памяти порождали новые, еще более сложные проблемы – например, необходимость синхронизации асинхронных, не связанных между собой вычислительных процессов, в процессоре. Известные способы устранения этих проблем (квитирование [44], схемы с арбитром [45] и др.) привели к существенному усложнению схем реализации доступа к ресурсу, что потребовало ввести в структуру процессора специализированный блок для управления и синхронизации – шинный интерфейс, еще больше усложнивший логику функционирования. Эти решения позволили лишь несколько улучшить ситуацию, вызванную ограничениями архитектуры процессора, но не решили проблему параллельного бесконфликтного доступа к одному ресурсу.

С ростом степени интеграции и частот работы процессоров все более определяющими стали энергетические ограничения, т.к. возможности по отводу тепла с единицы площади имеют физические пределы. «Энергетическая стена» [37], вызванная ростом на несколько десятичных порядков потребляемой процессором мощности при сохранении площади кристалла, требует от разработчиков массы усилий для отвода тепла и повышения энергоэффективности процессоров. В настоящее время «энергетическая стена» приобретает все большее значение для микропроцессоров, особенно с учетом их многоядерности, и еще сильнее углубляет кризис современных процессорных технологий.

Третьей фундаментальной проблемой процессорной архитектуры для высокопроизводительных вычислений является «стена параллелизма на уровне команд» [37], которая заключается в невозможности существенного повышения реальной производительности только за счет архитектурных усовершенствований процессора и качества компиляторов без учета присущего задаче естественного параллелизма. Первыми с этим ограничением столкнулись разработчики МВС при создании автоматических распараллеливающих компиляторов [46], а потом и разработчики процессоров констатировали невозможность создания эффективных компиляторов для казавшихся перспективными архитектур VLIW [20] и EPIC [21, 47]. В более простой формулировке «стена параллелизма на уровне команд» означает, что невозможно эффективно автоматически распараллелить изначально последовательную программу для архитектуры фон-Неймана, для этого необходима полная переработка программы с учетом естественного параллелизма задачи и архитектуры процессора.

Еще одной проблемой, существование которой отмечают многие специалисты [37], является необходимость кардинальным образом менять понимание одновременных вычислений, которые могут выполняться как информационно-независимо (параллельно), что и используется в большинстве современных МВС, так и информационно-зависимо (конвейерно), что используется только сравнительно небольшим числом специализированных

вычислительных устройств. Это явление получило наименование «стены ментальности».

Процессор как вычислительная архитектура оказался запертым в «четырех стенах»: стене памяти, энергетики, параллелизма на уровне команд и ментальности, что однозначно свидетельствуют о системном кризисе как архитектурных решений в области проектирования процессоров, так и кризисе технологических возможностей. Наложение (пересечение) этих кризисов обладает мультипликативным эффектом, и на настоящий момент проектирование процессоров находится в глубочайшем упадке из-за отсутствия новых идей. Возможности по усложнению внутренней структуры процессора исчерпаны, физические пределы технологических ограничений норм выпуска микросхем уже отчетливо видны, поэтому повышение производительности на реальных задачах, а не тестах, требует все более сложных и изощренных искусственных решений. Отсутствие предложений по разрешению рассмотренных проблем свидетельствуют о глубочайшем системном кризисе процессоров как вычислительной архитектуры, который выражается в невозможности обеспечения темпов роста реальной производительности высокопроизводительных вычислительных систем с процессорной архитектурой для широкого класса задач.

Раньше, ярче и острее всего кризис архитектуры процессоров проявился в области проектирования, создания и эксплуатации многопроцессорных вычислительных систем, поскольку именно в этой области число одновременно работающих при решении прикладной задачи процессоров раньше всего превысило оптимальное для процессорной архитектуры значение, наличие которого отмечали разные исследователи [48, 49]. Объявленная в начале 2000-х годов «точка перегиба для всей индустрии» - мультипроцессорность [50-52] многоядерных процессоров, объединивших несколько процессоров (процессорных ядер) с единым адресным пространством в одном корпусе для увеличения производительности, привела к тем же проблемам одновременного доступа нескольких процессоров к общему ресурсу, с которыми ранее уже сталкивались разработчики МВС. Более того, для многоядерных процессоров, как и для МВС, особое значение

приобрел сформулированный в 1967 году закон Амдала [53], устанавливающий верхнюю границу ускорения при распараллеливании вычислений. Поэтому реальная производительность определяется не только и не столько быстродействием элементной базы аппаратного обеспечения, а во многом – информационной структурой задачи и ее реализацией. Этим объясняется существенное снижение производительности многоядерных процессоров и МВС при большой доле последовательного кода. Существенный вычислительный ресурс, сосредоточенный в МВС, далеко не всегда обеспечивает пропорциональный числу узлов уровень производительности при решении прикладных задач. Разрыв между потенциально достижимой (пиковой) производительностью вычислительной системы и фактически достигаемой при решении задачи (реальной) производительностью характеризует эффективность ее параллельного решения. Разрыв между пиковой и реальной производительностью нарастает с увеличением числа процессоров в МВС из-за непропорционального роста накладных расходов, что сокращает эффективность решения задач [58, 59].

Стагнация в области повышения производительности новых процессоров наблюдается в течение последних 10-15 лет (с 2007-2008 годов): «косметические улучшения» архитектуры процессора (оптимизация блока предсказания ветвления, передней части конвейера, размеров буферов и числа исполнительных устройств процессора), которые стали называться микроархитектурой [20, 21], не могут обеспечить сколько-нибудь заметный прирост производительности, а переход на новый технологический уровень позволяет добавить еще несколько устройств или ядер процессора, но не увеличивает его реальную производительность из-за усложнения системы синхронизации и алгоритмов управления.

Большинство современных исследований в области параллельного программирования [52-57] сходятся во мнении, что необходима новая парадигма для эффективной организации вычислений. Сложившееся для многоядерных процессоров положение, когда распараллеливание изначально последовательного кода на несколько вычислительных ядер возложено на компилятор, который, помимо распараллеливания, должен синхронизировать

и результаты его выполнения, неэффективно и неприемлемо. Очевидно, что такая технология организации вычислительного процесса заведомо и очень существенно проигрывает исходно параллельному описанию задачи для выполнения на нескольких вычислительных устройствах, поскольку дублирует «бутылочное горло» сначала при распараллеливании, затем при синхронизации.

Поэтому на сегодняшний день можно с уверенностью констатировать завершение эры процессоров как основного направления развития вычислительной техники из-за наложения кризиса архитектуры процессора (структуру процессора усложнять далее бессмысленно и неэффективно) на технологический кризис (физические ограничения на размер транзисторов и выделяемое ими тепло). Фон-неймановская архитектура процессоров исчерпала все свои доступные ресурсы для модернизации, и дальнейшее развитие средств высокопроизводительных вычислений (после 2022 года) будет обеспечиваться главным образом за счет разработки и создания новых вычислительных архитектур, содержащих в едином вычислительном контуре процессор и специализированный вычислитель. Такие гибридные (или конвергентные) архитектуры, совмещающие в одном вычислительном устройстве процессор и один или несколько специализированных вычислителей (графический процессор или программируемую логическую интегральную схему) активно разрабатывают и серийно выпускают ведущие мировые производители: компании Intel и AMD[60].

Появление графических ускорителей (графических процессоров) как самостоятельных вычислительных устройств стало одной из первых удачных попыток реализации массовой архитектуры для высокопроизводительных вычислений, альтернативной традиционным процессорам. Первоначально вычисления на графических ускорителях появились как новая технология GPGPU (англ. General-purpose computing for graphics processing units) [54], которую воспринимали как новое направление в области ускорения вычислений традиционных процессоров с помощью дополнительных устройств. В качестве самостоятельного вычислительного устройства, способного успешно конкурировать с традиционными процессорами при

решении вычислительных задач ряда предметных областей, графические ускорители (GPU) стали восприниматься после того, как в июне 2014 года вторым по производительности суперкомпьютером Top-500 с производительностью в 17,59 Пфлопс был признан построенный компанией Cray с использованием 14-ядерных GPU NVIDIA Tesla K20x суперкомпьютер Titan [10]. Типовая структура современных графических ускорителей во многом соответствует представленной на рис. 1.4 структуре GPU NVIDIA G80 с конвейерной архитектурой и состоит из тысяч более простых (по архитектуре) и энергоэффективных ядер с меньшим набором команд, созданных для обработки нескольких задач одновременно. Базовый элемент графического ускорителя – ядро, называемое SP (Single Processing core), для выполнения большого количества математических операций из ограниченного набора без обращения к кэш-памяти.

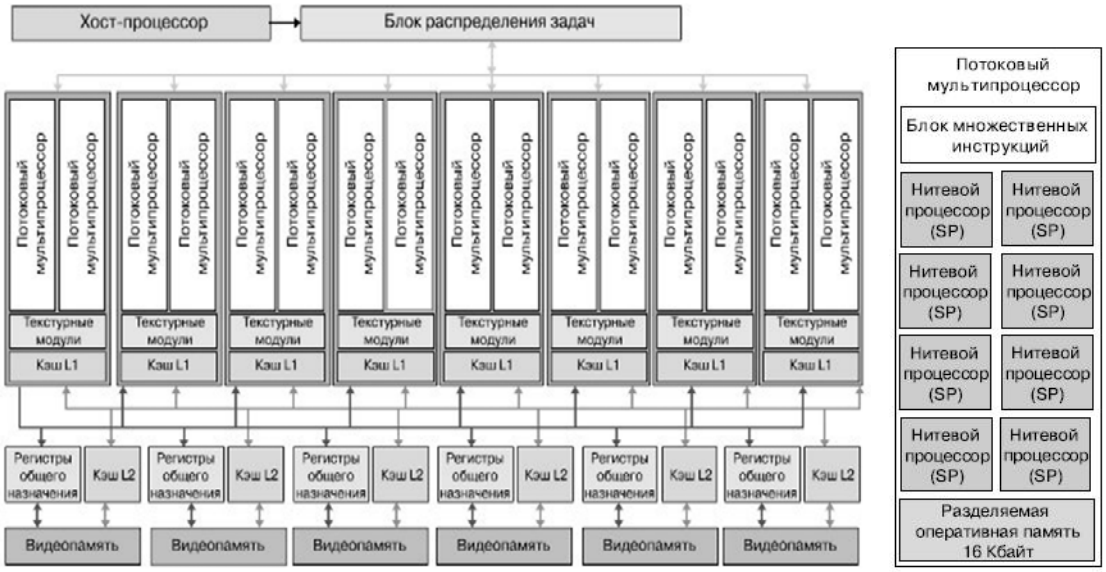


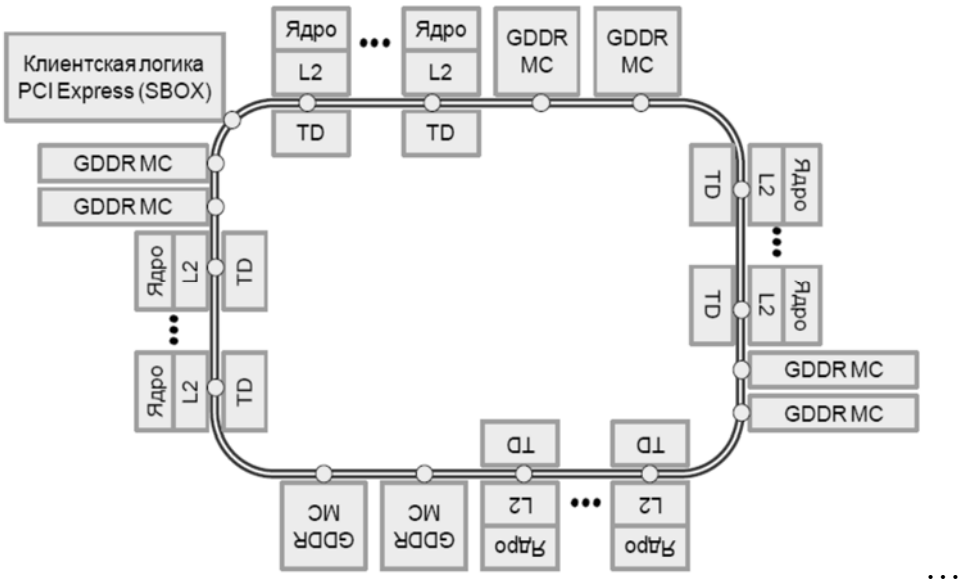

Рис. 1.4. Типовая структура графического ускорителя на примере NVIDIA G80, 2008 год

Каждый SP состоит из нескольких арифметико-логических устройств (АЛУ) и одного или нескольких математических сопроцессоров для операций с плавающей точкой (floating point unit, FPU). Ядра SP объединяются в массив со своим диспетчером исполнения, специальным процессором и кэш-памятью, а несколько массивов объединяются в текстурный процессор со своей кэш-памятью. Таким образом, графические ускорители представляют собой

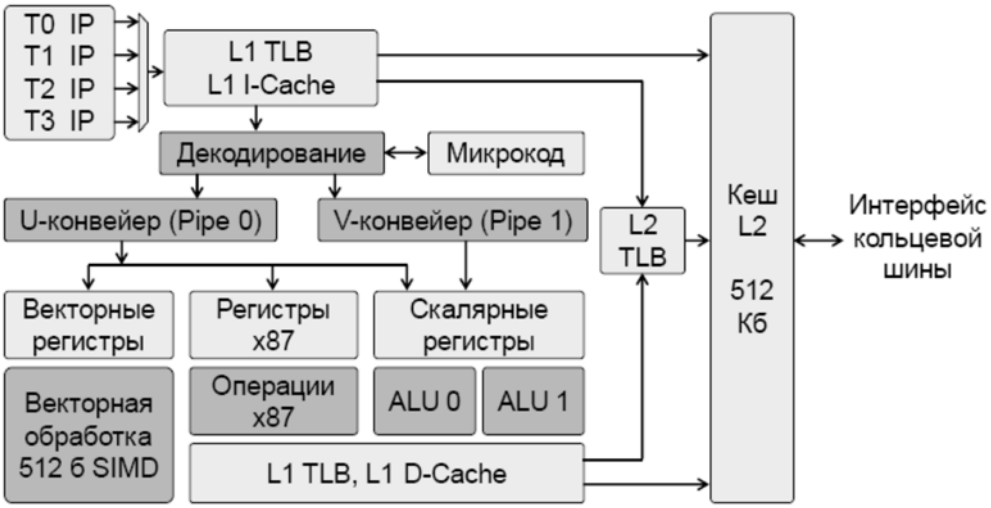
векторные многоядерные процессоры класса SIMD с несколькими сотнями достаточно простых потоковых процессоров, работающих с общей памятью. Наивысшую производительность вычислительные системы с графическими ускорителями показывают на задачах с локальными данными, которые целиком размещаются в памяти ускорителя. При этом рост производительности вычислительной системы обеспечивается, в первую очередь, не увеличением производительности отдельных ядер ускорителя, а значительным увеличением числа одновременно выполняемых операций для информационно-независимых или информационно-зависимых данных, размещенных во внутренней памяти GPU. Рост реальной производительности графических ускорителей на слабосвязанных задачах, в которых число информационных обменов невелико, может составлять несколько сотен раз по сравнению с кластерными системами, именно этим обстоятельством определяется довольно большое число систем с графическими ускорителями в верхних строках списка TOP 500 [10]. Достигнутая ими высокая производительность, пусть и при решении задач определенных классов, наглядно продемонстрировала преимущества конвейерной организации вычислений для решения прикладных задач. Организация вычислительного конвейера в графических ускорителях во многом схожа с реализацией вычислений на ПЛИС с той разницей, что GPU не позволяет изменять связи между своими элементами (ядрами, АЛУ и сопроцессорами) и поэтому не может гибко адаптироваться под информационную структуру решаемой задачи. Наибольшую эффективность графические ускорители демонстрируют для решения задач искусственного интеллекта (ИИ, Artificial Intellegence, AI), т.к. архитектура графического ускорителя, содержащего большое число потоковых процессоров, обрабатывающих данные малой разрядности (12 или 16 бит), наилучшим образом соответствует структуре искусственных нейронных сетей. Если информационные массивы не могут быть целиком размещены в памяти ускорителя, его реальная производительность снижается в 10÷100 раз по сравнению с пиковой из-за фиксированной топологии связей между ядрами и «бутылочным горлом» на шине обмена GPU с процессором. Для высокопроизводительных вычислений графические ускорители также

уперлись в «стену памяти», поэтому кризис архитектуры графических ускорителей для высокопроизводительных вычислений во многом схож с процессорным.

Специализированные вычислители процессорной архитектуры (сопроцессоры), наиболее известным из которых является Intel Xeon Phi [61, 62] (рис. 1.5), тоже подвержены влиянию перечисленных ограничений.




а)




б)

Рис. 1.5. Сопроцессор Intel Xeon Phi

а) структура сопроцессора, б) архитектура вычислительного ядра

Сопроцессор Intel Xeon Phi фактически представляет собой предназначенный для решения задач математической физики узкоспециализированный вычислитель на основе объединенных в кольцо мощных процессорных ядер и векторного сопроцессора. Intel Xeon Phi содержит 61 ядро, которые соединены высокопроизводительной встроенной двунаправленной кольцевой шиной. Для вычислений используется 60 ядер, каждое из которых является полнофункциональным и поддерживает выборку и декодирование инструкций из четырех потоков команд. Для повышения эффективности работы с памятью в сопроцессоре реализован распределенный каталог тегов кэша, позволяющий использовать более эффективный протокол для поддержания когерентности кэшей всех ядер. Ядра Intel Xeon Phi обеспечивают выполнение 32- и 64-битного кода, совместимого с архитектурой Intel64 без поддержки расширений MMX, AVX и SSE (всех версий). Блок векторных вычислений, содержащийся в каждом ядре, дополнительно реализует набор операций над 512-битными векторами. Контроллеры памяти обеспечивают теоретическую пропускную способность 352 гигабайта в секунду. Двунаправленная кольцевая шина обеспечивает передачу данных между компонентами сопроцессора.

Высокие характеристики быстродействия и скорости коммуникации с периферийными устройствами обусловлены аппаратной реализацией большинства используемых в задачах математической физики операций. Высокая стоимость Intel Xeon Phi, необходимость переработки прикладных программ под свою структуру и эффективность для достаточно узкого класса задач математической физики значительно сократили границы его применимости. Вместе с тем применение графических ускорителей или сопроцессоров, выполняющих основной объем специализированных операций с данными и ускоряющих процесс их обработки, действительно стало шагом вперед в области сверхвысокопроизводительных вычислений. При объединении в одну систему с традиционными процессорами графические ускорители и сопроцессоры позволяют резко поднять производительность такой системы на некоторых классах задач, не вызывая соответствующего скачка в потреблении энергии.

Кризис процессорной архитектуры и надвигающийся кризис графических ускорителей обусловил пристальное внимание к другим вычислительным архитектурам для ускорения вычислений, среди которых особое место занимают программируемые логические интегральные схемы (ПЛИС), позволяющие гибко изменять свою архитектуру под информационную структуру решаемой задачи.

## 1.2. Тенденции развития современных реконфигурируемых вычислителей

Программируемая логическая интегральная схема (ПЛИС, англ. programmable logic device, PLD) - электронный компонент для создания цифровых схем, логика работы которого задаётся посредством программирования (проектирования) на специальных языках описания аппаратуры (Hardware Definition Languages, HDL): Verilog, VHDL, AHDL и других. Архитектура ПЛИС появилась в конце 1970-х годов с программируемыми матрицами "И" и "ИЛИ" – архитектуры FPLA (Field Programmable Logic Array) и FPLS (Field Programmable Logic Sequensers) [65], дальнейшее развитие которых привело к современной архитектуре FPGA (Field Programmable Gate Array). Архитектура FPGA содержит множество конфигурируемых логических блоков (Configurable Logic Block, CLB), объединенных с помощью матрицы соединений. Каждый CLB состоит из одного или нескольких относительно простых логических элементов с 3-4-5-входовой таблицей преобразования (Look-Up Table, LUT), программируемый мультиплексор, D-триггер, цепи управления, а также элементы ускоренного переноса CARRY и триггеры FF. Кроме CLB, в структуре ПЛИС содержатся аппаратные блоки умножения-суммирования (Digital Signal Processing, DSP), блоки внутренней памяти BRAM, генераторы тактовых сигналов (PLL, MMCM), каналы ввода-вывода IO, аппаратные ядра стандартных интерфейсов (DDR, Ethernet, PCI Express, высокоскоростные приемопередатчики и т.д.) и матрицы соединений. Кристаллы ПЛИС используются для решения отдельных подзадач цифровой обработки сигналов, сетевых технологий и встраиваются в общую схему оборудования, а возможность

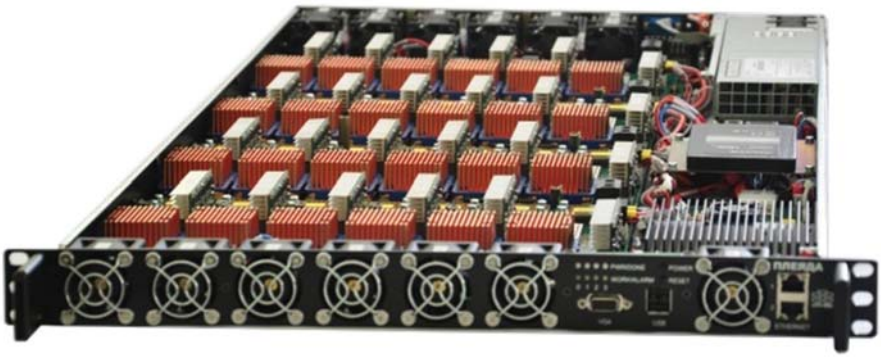
программирования архитектуры позволяет менять алгоритмы обработки данных в ПЛИС без изменения топологии устройства [66-68]. Возможность программирования архитектуры и создания специализированного устройства, аппаратно реализующего алгоритм обработки данных, открыло перспективы использования ПЛИС в качестве элементной базы для построения высокопроизводительных вычислительных систем (суперкомпьютеров).

Высокопроизводительные вычислительные системы на основе ПЛИС можно отнести к двум типам. К первому, наиболее массовому, типу относятся вычислительные системы с кластерной архитектурой, каждый узел которых содержит не только процессор, но и ускорители вычислений с 2-4 ПЛИС, выполняющими вспомогательную роль, обрабатывая отдельные трудоемкие участки алгоритмов. Примером таких кластерных суперкомпьютеров с ПЛИС являются Cray XT4 [69, 70], SiliconGraphics [71] RASC [72] на основе реконфигурируемых вычислителей Nallatech [73], Novo-G [74] и др. Такие гибридные вычислительные системы фактически представляют собой многомашинные комплексы, потому что они состоят из 2 подсистем: процессорной и специализированной. Программирование процессорной подсистемы осуществляется с помощью библиотек MPI, а реконфигурируемой – с помощью инструментария разработчиков кристаллов ПЛИС. ПЛИС в такой архитектуре используются для ускорения вычислительно-трудоемких фрагментов вычислений решаемой прикладной задачи, а решение задачи в целом возлагается на универсальные процессоры. Хотя использование ПЛИС-ускорителей в перечисленных выше гибридных системах позволяет существенно повысить их производительность, тем не менее все они обладают теми же недостатками, которые присущи обычным кластерным суперкомпьютерам, а именно – низким соотношением реальной производительности к пиковой, а также невысокими показателями удельной эффективности и энергоэффективности.

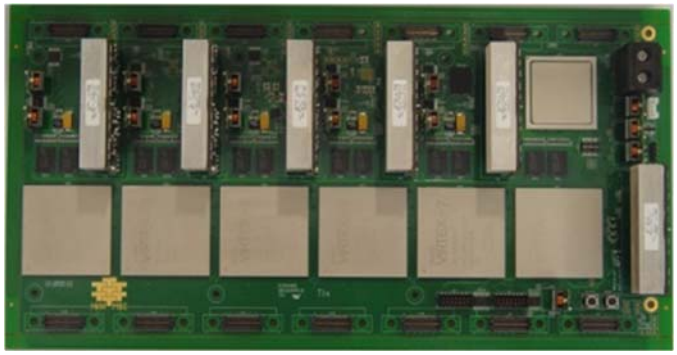
Второй тип вычислительных систем на основе ПЛИС представлен реконфигурируемыми вычислительными системами (РВС) [75], в которых ПЛИС используются в качестве основного вычислительного ресурса и объединяются в большие вычислительные поля с помощью пространственной

коммутационной системы. Процессоры общего назначения выполняют второстепенные функции сопровождения и используются для контроля функционирования реконфигурируемой части системы. Среди зарубежных компаний, занимающихся разработками подобных реконфигурируемых суперкомпьютеров, можно выделить такие компании как StarbridgeSystems [76], SciEngines [77], DiniGroup [78], XputerLab X [79], SGI [80], MathStar [81], вычислители COPACOBANA [82] и RIVYERA [83]. Отечественными производителями РВС этого типа являются ФГУП «НИИ «Квант», НИИ многопроцессорных вычислительных систем имени академика А.В. Каляева Южного федерального университета (НИИ МВС) и научно-исследовательский центр супер-ЭВМ и нейрокомпьютеров (НИЦ СЭ и НК, г. Таганрог). Спектр выпускаемых изделий различной степени интеграции достаточно широк: от полностью автономных малогабаритных реконфигурируемых ускорителей (вычислительных блоков «Мерак» и «Фекда») [84-91], реконфигурируемых вычислительных блоков (РВБ) в настольном или стоечном конструктивном исполнении (РВБ «Медведь» на Virtex-4, РВБ «Алиот» на Virtex-5, РВБ «Ригель» на Virtex-6, РВБ «Плеяда» на Virtex-7 и др.) [92-96] до реконфигурируемых вычислительных систем, состоящих из нескольких вычислительных шкафов, размещаемых в специально оборудованном машинном зале (РВС «Бенетнаш» [97-100], РВС «Плеяда» [97-100] и др). Отличительной особенностью всех изделий являлась сверхплотная компоновка ПЛИС в РВС, объединенных в единый вычислительный ресурс. Так, в 2009 году компания Silicon Graphics анонсировала рекордную, по ее расчетам, плотность компоновки с размещением 64 ПЛИС в одной вычислительной стойке конструктива 42U [71]. В том же 2009 году в НИИ МВС был создан РВБ «Орион» [101], содержащий 64 ПЛИС Xilinx Virtex-5 в конструктиве 1U, что в 42 раза превышало плотность компоновки Silicon Graphics и являлось неофициальным мировым рекордом. Одним из наиболее характерных образцов РВБ для настольного и стоечного исполнения является созданный в НИИ МВС по государственному контракту №14.527.12.0004 от 03.10.2011 вычислительный модуль 24V7-750 (ВМ «Плеяда»), представленный на

рисунке 1.6-а. ВМ 24V7-750 содержит четыре платы 6V7-180 (рис. 1.6.-б), управляющий модуль, подсистему питания, подсистему охлаждения и другие подсистемы. Производительность ВМ 24V7-750 составляет 2,58 Тфлопс при обработке 32-разрядных данных с плавающей запятой. Из 24 ВМ 24V7-750 была создана РВС «Плеяда», с общим вычислительным полем из 576 микросхем ПЛИС Virtex-7 XC7V585T-FFG1761 объемом 58 миллионов эквивалентных вентилей каждая, конструктивно объединенных в один вычислительный шкаф высотой 47U с пиковой производительностью $10^{15}$ операций с фиксированной запятой в секунду.



а)



б)

Рис. 1.6. Вычислительный модуль 24V7-750 (а - со снятой верхней крышкой, б – плата 6V7-180)

В НИЦ СЭ и НК, помимо РВС с воздушным охлаждением, разрабатываются и серийно выпускаются РВБ «Неккар» (рис. 1.7) и «Сегин» с иммерсионным жидкостным охлаждением для кристаллов ПЛИС, содержащих порядка 100 млн. эквивалентных вентилей с энергопотреблением

не менее 100 Вт: Virtex UltraScale, Virtex UltraScale+, Virtex UltraScale2 и др. Корпус РВБ с жидкостным охлаждением выполняется в конструктиве 3U 19” и состоит из вычислительной и теплообменной секций. Вычислительная секция представляет собой герметичный контейнер с хладагентом, в котором размещаются 12 плат вычислительных модулей, модули питания и управления, кросс-платы. Каждая плата вычислительного модуля РВБ «Неккар» содержит 8 ПЛИС семейства Kintex UltraScale XCKU095T и контроллер на ПЛИС Xilinx Kintex-7. Теплообменная секция с насосом и теплообменником примыкает к вычислительной секции. Насос обеспечивает циркуляцию хладагента в РВБ через теплообменник по замкнутому контуру. Теплообменник подключен к вторичному контуру охлаждения через штуцеры и предназначен для охлаждения хладагента, как правило, с помощью воды.

Рис. 1.7 Реконфигурируемый вычислительный блок «Неккар»

В высокопроизводительных вычислениях многокристальные РВС применяются для обработки и анализа больших потоков данных постоянным или редко меняющимся алгоритмом. Возможность программирования архитектуры вычислительной системы под структуру решаемой задачи

позволяет сократить накладные расходы на организацию вычислительного процесса и достигать высокой реальной производительности даже при существенно меньшей тактовой частоте работы устройства. Для решения задач в РВС создается специализированная вычислительная структура, где каждая операция задачи выполняется аппаратно реализованным устройством (сумматором, умножителем, делителем и др.). Все элементы вычислительной структуры соединены между собой прямыми связями в соответствии с информационными зависимостями задачи и могут использовать как параллелизм, так и конвейеризацию для реализации одновременных вычислений. Использование конвейерной обработки информации позволяет не только минимизировать время организации вычислительного процесса до времени заполнения конвейера, но и существенно сократить время решения до времени прохождения потока данных, обеспечивая высокую, не ниже 70%, реальную производительность. Снижение реальной производительности РВС на основе ПЛИС наблюдается при решении задач, алгоритм обработки информации которых зависит от входных данных (большое число условных операторов, рекуррентных структур и информационно-незначимых пересылок информации). Причиной этому является непродуктивные затраты ресурса на аппаратную реализацию всех возможных вариантов обработки, из которых будет выбран только один результат, соответствующий входным данным. Как правило, такая структура характерна для задач управления, которые с большей эффективностью могут быть решены на традиционных или специализированных сигнальных процессорах.

Важной особенностью ПЛИС является то, что в настоящее время их характеристики далеки от теоретического максимума и продолжают линейно расти. В работах [75, 102] показано, что ПЛИС обеспечивают стабильный, близкий к линейному, рост производительности. Темпы роста производительности ПЛИС существенно превышают темпы универсальных процессоров (рис. 1.8), причем не только для пиковой, но и для реальной производительности на широком классе задач, включая сильносвязанные. Стабильность темпов роста производительности ПЛИС для каждого следующего семейства обеспечиваются ростом частоты на 25% и удвоением

числа элементов [69-71]. Динамика роста этих показателей для ПЛИС фирмы Xilinx приведена на рис. 1.9.

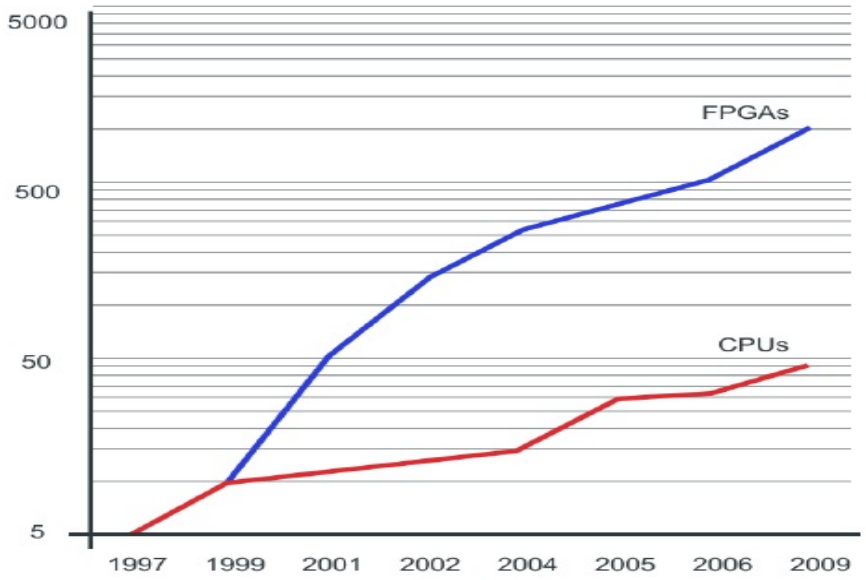



Рис. 1.8. Рост производительности ПЛИС (в GMACs) и универсальных процессоров (в GFLOPs)

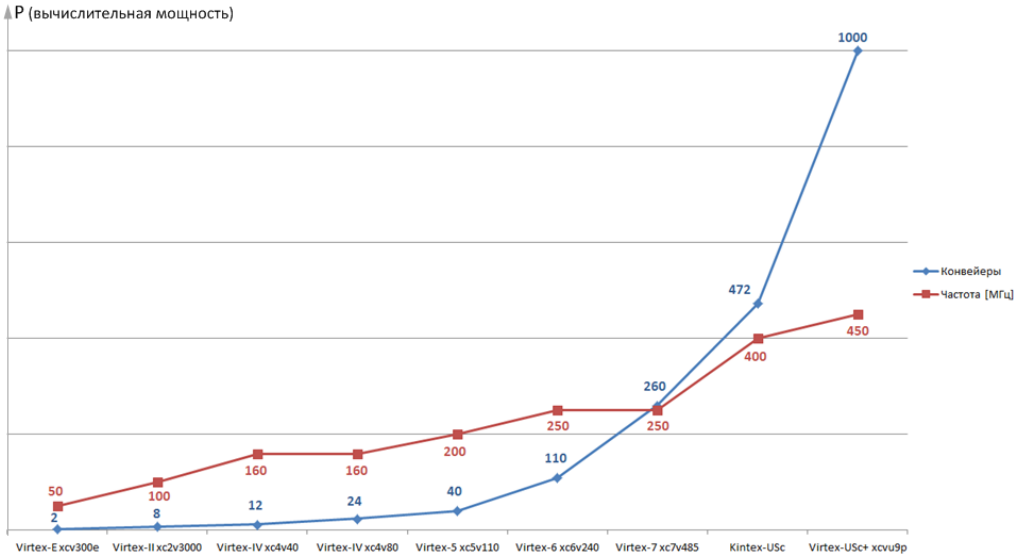



Рис. 1.9. Динамика роста вычислительной мощности ПЛИС фирмы Xilinx

Можно отметить, что РВС эффективны для решения потоковых задач обработки больших и сверхбольших объемов данных в соответствии с редко меняющимся алгоритмом. Характерная для РВС конвейерная организация вычислений обеспечивает максимальный уровень удельной

производительности для большинства задач, в том числе сильносвязанных. ПЛИС являются перспективной элементной базой для решения вычислительно-трудоемких задач, т.к. позволяют создавать вычислительную структуру, соответствующую решаемой задаче, что расширяет круг задач, обладают резервом по технологическому процессу производства (9 нм на сегодняшний день, достижение 5-7 нм - не раньше, чем через два года), по тепловыделению (по сравнению с процессорами рабочая температура ПЛИС ниже и гораздо меньше выделение тепла) и по обилию поддерживаемых интерфейсов доступа к памяти и синхронизации вычислений. Поэтому вычислительная архитектура ПЛИС является перспективной для построения гибридных вычислительных систем.

## 1.3. Анализ вычислительных архитектур для построения гибридных вычислительных систем

Идея построения гибридных вычислительных систем (ГВС), совмещающих в себе достоинства различных вычислительных архитектур, в том или ином виде формулировалась различными специалистами, начиная с 60-х годов 20 века. Под гибридной вычислительной системой здесь и далее будем понимать соединенное пространственной коммутационной системой с высокоскоростными интерфейсами множество узлов различных вычислительных архитектур. В настоящее время в качестве узлов ГВС используются процессоры, графические ускорители и/или ПЛИС. Создание высокопроизводительных гибридных вычислительных систем, содержащих узлы различных архитектур: процессоры, графические ускорители, ПЛИС, специализированные вычислители, стало попыткой преодоления кризиса вычислительных архитектур. Среди существующих высокопроизводительных вычислительных систем с гибридной или гетерогенной архитектурой можно отметить:

- суперкомпьютер Summit [7] Министерства энергетики США, содержащий 4608 взаимосвязанных серверных узлов IBM Power Systems

AC922 с 9216 процессоров IBM Power9 (22 ядра) и 27648 графических процессоров NVIDIA Tesla V100;

- суперкомпьютер с гибридной архитектурой Tianhe-2 [10] – 16 тысяч узлов с двумя 12-ядерными процессорами Intel Xeon E5-2692 с частотой 2,2 ГГц и тремя специализированными 57-ядерными сопроцессорами Intel Xeon Phi 31S1P с частотой 1,1 ГГц;

- суперкомпьютер с гибридной архитектурой Sunway TaihuLight [9] на основе многоядерных RISC-процессоров китайского производства SW26010 (64 бита, архитектура ShenWei, частота 1,45 ГГц) с 4 управляющими ядрами общего назначения и 256 вычислительными RISC-ядрами. Общее количество процессоров в суперкомпьютере – 40960, а общее количество ядер – свыше 10,5 миллионов;

- суперкомпьютер Fugaku [8] с 158 976 микропроцессоров Fujitsu A64FX;

- первый экзафлопсный суперкомпьютер Frontier (OLCF-5) [10], основанный на комбинации процессоров AMD Epyc и графических процессоров Radeon Instinct;

- суперкомпьютер Aurora [10] Аргоннской национальной лаборатории, содержащий более 9 тысяч вычислительных узлов мощностью 130 терафлопс, каждый с двумя процессорами Intel Xeon Sapphire Rapids, шестью графическими процессорами Intel XE и единой архитектурой памяти.

Многие ведущие исследователи на основе опыта создания ГВС предлагают создать конвергентную вычислительную архитектуру, объединяющую в одном кристалле процессоры и/или графические ускорители и/или ПЛИС. Решения в области конвергентных вычислительных архитектур разрабатывают большинство ведущих производителей – Intel, AMD, HP, Xilinx и др. Примером таких гибридных архитектур являются SapphireRapids (рис. 1.10) – основной процессор для экзафлопсной МВС Aurora и высокопроизводительный дискретный графический процессор общего назначения Ponte Vecchio от компании Intel, содержащие специализированные аппаратные блоки для высокоскоростной реализации часто используемых в различных предметных областях фрагментов вычислений.

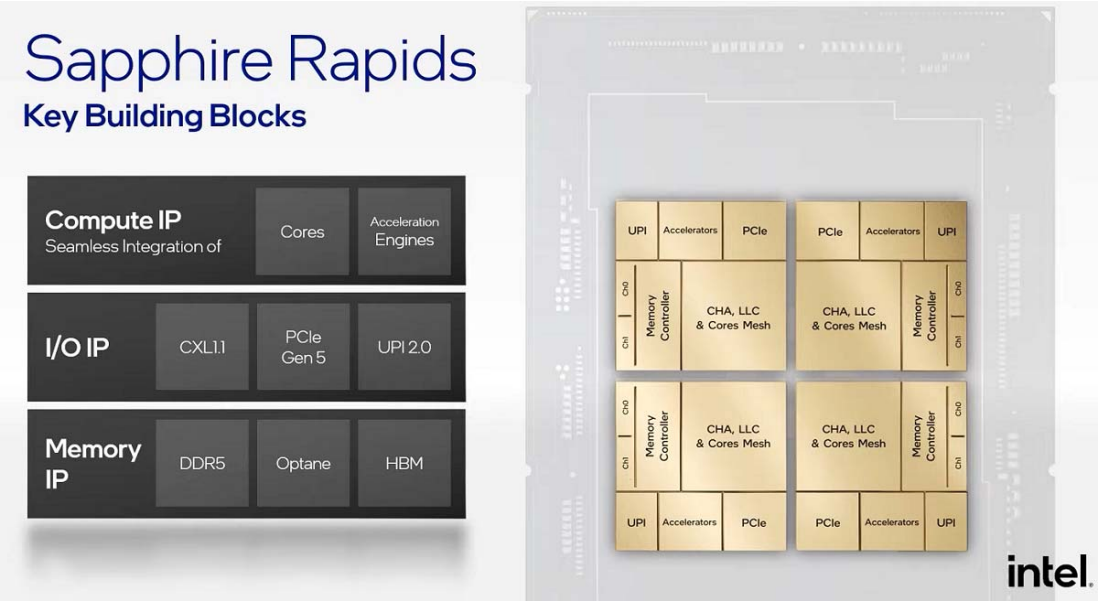



Рис. 1.10. Структурная схема процессора SapphireRapids

Характерным примером конвергентной архитектуры от производителей ПЛИС является разработанная компанией Xilinx настраиваемая платформа для ускорения вычислений ACAP (Adaptive Compute Acceleration Platform) Versal (рис. 1.11), ранее известная под кодовым названием Everest.

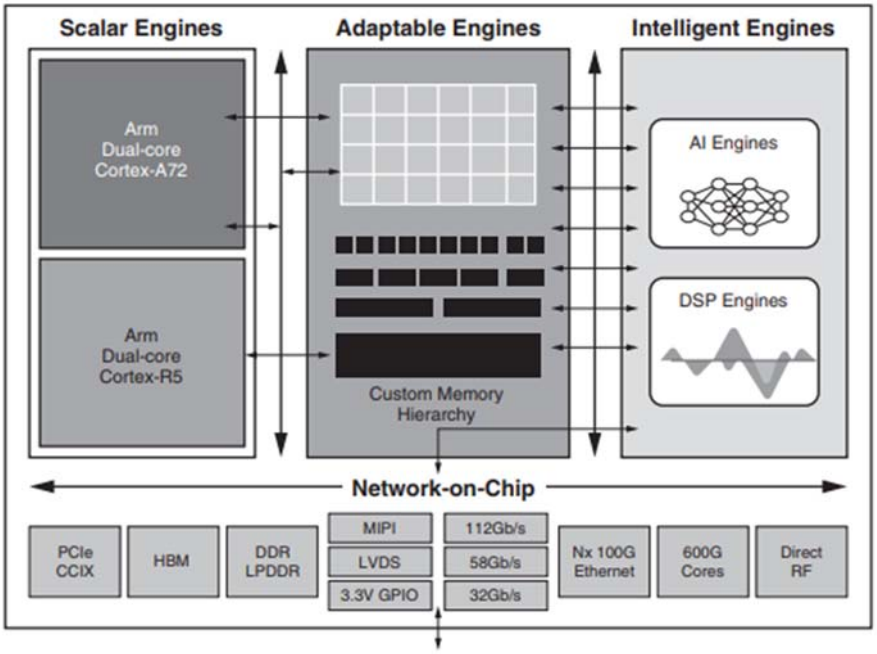



Рис. 1.11. Функциональная схема кристаллов Xilinx Versal

Новая архитектура ПЛИС ACAP семейства Versal представляет собой выполненные по 7 нм техпроцессу кристаллы, содержащие на одной подложке 3 разные вычислительные архитектуры – скалярные процессоры (2-ядерный Arm Cortex-R5 для работы в режиме реального времени и 2-ядерный Arm Cortex-A72), привычную реконфигурируемую логику с реконфигурацией «на лету», специализированные массивно-параллельные вычислители (DSP-блоки для выполнения вычислений с плавающей точкой и AI-блоки с векторными ядрами VLIW/SIMD частотой до 1 ГГц, объединенные с блоками встроенной памяти). Кристалл Versal ACAP содержит большое количество различных интерфейсов ввода-вывода, привычных для каждой из реализованных в кристалле архитектур – процессорной, ПЛИС и графических ускорителей. Versal ACAP содержит 4 вида программируемых процессоров (рис.1.12):

– прикладной процессор (Application Processing Unit, APU) в процессорной части (PS) для управляющих приложений, операционных систем, коммуникационных интерфейсов, простых или сложных вычислений;

– процессор реального времени (Real-time Processing Unit, RPU) в процессорной части (PS) для приложений, требующих повышенной безопасности и детерминизма;

– процессоры MicroBlaze в программируемой части (PL) для манипулирования данными и их транспортировкой, невекторных вычислений, сопряжения с процессорной частью (PS) и другими компонентами платформы ACAP;

– блоки искусственного интеллекта (AI) для векторных реализаций вычислительно-трудоемких функций.

Для программирования Versal ACAP на языках C, C++ и Python предназначена среда разработки Vitis, поддерживающая как фреймы (например, Caffe, TensorFlow) для задач AI, предварительно оптимизированные библиотечные элементы на C/C++/Python для процессоров или RTL-проекты для реконфигурируемой логики ПЛИС

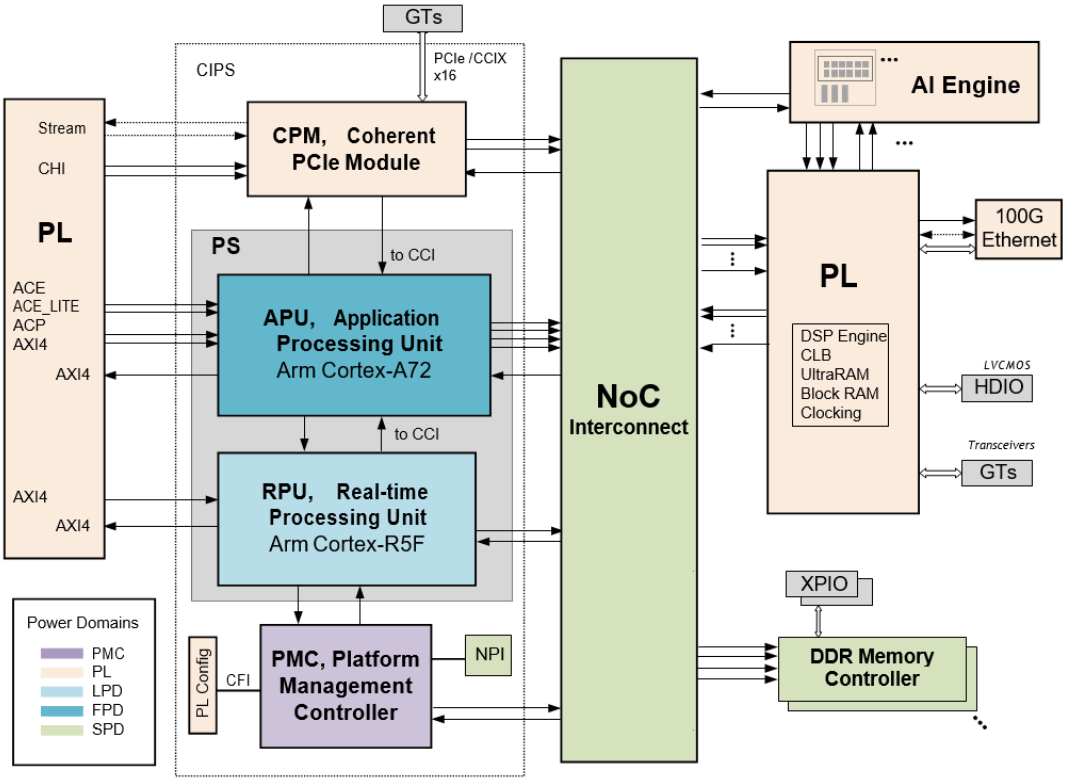



Рис. 1.12. Схема взаимодействия частей платформы Versal ACAP

Архитектура ACAP отражает стремления производителей аппаратуры к совмещению различных архитектур в одном вычислительном устройстве для его унификации при решении задач различных предметных областей. С позиций производителя это оправданная стратегия, но при решении задач такая гетерогенная вычислительная архитектура приводит к тому, что часть оборудования, не используемая для решения задачи, будет простаивать, что приводит к существенному снижению удельной производительности сродни характерному для многопроцессорных вычислительных систем кластерной архитектуры разрыву между декларируемой пиковой и реальной производительностью. Например, большинство существующих ГВС организованы по федеративному принципу [14-17] и представляют собой многомашинный комплекс, содержащий подсистемы – функционально обособленные части из узлов одной вычислительной архитектуры, а подсистемы соединены между собой коммутационной системой. Даже для мощнейших мировых суперкомпьютеров, представляющих собой ГВС, каждая их подсистема программируется в рамках своей технологии

программирования и на своем языке, а взаимодействие между ними осуществляется на уровне передачи файлов или требует высочайшей квалификации программиста, граничащей с искусством и экспертных знаний особенностей гетерогенных вычислительных архитектур. Вместе с этим, тенденция производителей к созданию гетерогенных вычислительных систем, совмещающих в одном устройстве различные вычислительные архитектуры, в ближайшее время, по всей видимости сохранится, если новые архитектуры, такие как квантовые [103] или фотонные [104] компьютеры, не смогут достичь уровня широкого повсеместного применения. ГВС уже сегодня позволяют использовать в полной мере все преимущества ПЛИС-технологий для достижения высокой реальной производительности задач различных классов за счет сокращения числа непродуктивных вычислительных операций, а за счет наличия процессорных элементов – эффективно решать задачи управления, содержащие большое число вложенных условных операторов и рекурсивных выражений, неэффективно реализуемых на ПЛИС. Но для создания вычислительной системы с гибридной архитектурой недостаточно механического размещения гетерогенных вычислительных узлов в одной системе и их соединения коммутационной системой. Использование же всех имеющихся в кристалле вычислительных архитектур для повышения удельной производительности при решении задачи требует разрешения ряда сложных и взаимосвязанных алгоритмических, программных и технологических проблем, что для существующих средств разработки не представляется возможным. Необходимы принципы и методы совместной организации вычислений, чтобы можно было использовать достоинства различных архитектур для повышения эффективности решения задач

## 1.4 Технологии программирования и автоматического распараллеливания мультипроцедурных вычислений

Программирование высокопроизводительных вычислительных систем, несмотря на обилие созданных технологий, методов и средств, остается исключительно сложной областью разработки программ, доступной узкому

кругу специалистов. Высокая сложность обусловлена не только организацией одновременных вычислений для множества параллельных и асинхронных процессов, но и необходимостью учета архитектурных особенностей узлов, коммутационной подсистемы [105] и других аппаратных ресурсов высокопроизводительной вычислительной системы. Поэтому эффективность и производительность параллельной программы не в последнюю очередь зависит от используемых технологий программирования [18, 19], созданных для облегчения разработки для каждого типа организации вычислений

Для кластерных МВС, построенных на основе узлов с процессорной архитектурой, характерно мультипроцедурное распараллеливание, при котором каждый узел выполняет информационно-независимые операции над своим пакетом данных. Исходной формой представления задачи является последовательный алгоритм, который распараллеливается по множеству локальных, информационно-независимых участков. Для описания последовательных алгоритмов наибольшее распространение получили графовые формы – алгоритмическая схема (также называемая блок-схемой алгоритма) и граф алгоритма. Алгоритмическая схема [1] представляется в виде ориентированного графа, вершинами которого являются действия (процессы), а дуги отражают передачу управления. Алгоритмическая схема описывает задачу как процесс *передачи управления* от одного процесса (или вычислительного устройства) к другому. Помимо алгоритмической схемы, отражающей порядок (последовательность) выполнения операций вычислительным устройством (или устройствами), существуют и другие графовые формы описания процессов передачи управления – граф потока управления [106], сети Петри [107] и их модернизации[108], управляющий граф[106], граф передачи управления, граф программы [1], но в настоящее время они используются существенно реже.

Для реализации алгоритмической схемы или графа алгоритма при процедурной и мультипроцедурной организации вычислений используются языки программирования высокого уровня, такие как C, C++, C#, Fortran, ABCL [109], Ада [110], ML [111], Occam [112, 113], Orca [114], Phantom [115], Sisal [116], SR [117] и другие. Наибольшую популярность приобрели язык

программирования C и его варианты – C++, C# и исходно первый язык высокого уровня для математических вычислений Fortran. Для организации параллельных вычислений используются различные расширения и библиотеки – MPI [118, 119], OpenMP [120, 121], HPF [122, 123], DVM [124], PVM [125, 126], HOPMA [127-129], Java FJTask [130] и другие. Поскольку идеология самих языков направлена на описание графа потока управления [1, 106], а не на описание собственно вычислений, для описания одновременных вычислений приходится прибегать к специальным библиотекам или технологиям, так как последовательность выполнения операторов в программе задается порядком их следования в тексте. Распараллеливание может выполняться программистом как вручную с помощью одной из перечисленных в настоящем параграфе технологий, так и автоматически – с помощью распараллеливающих компиляторов.

Одной из наиболее известных технологий для организации мультипроцедурных вычислений является библиотека MPI (Message Passing Interface) [118], реализующая взаимодействие параллельных процессов с помощью механизма передачи сообщений. Параллельная программа в терминах MPI представляет множество одновременно выполняемых процессов на разных вычислительных узлах (рис.1.13), которые представляют собой копии одного и того же программного кода [119].

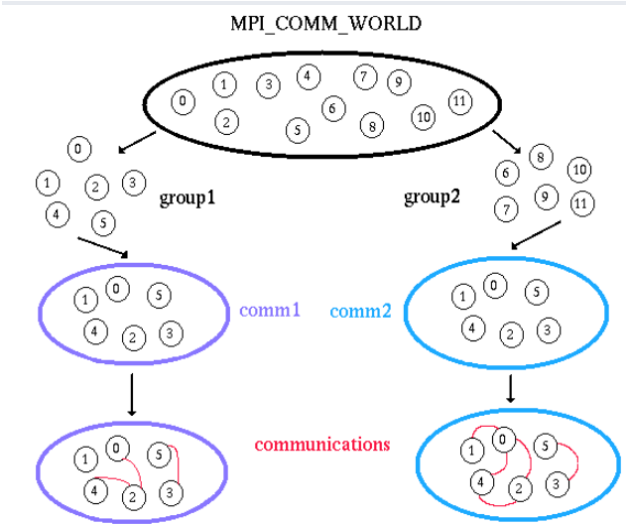
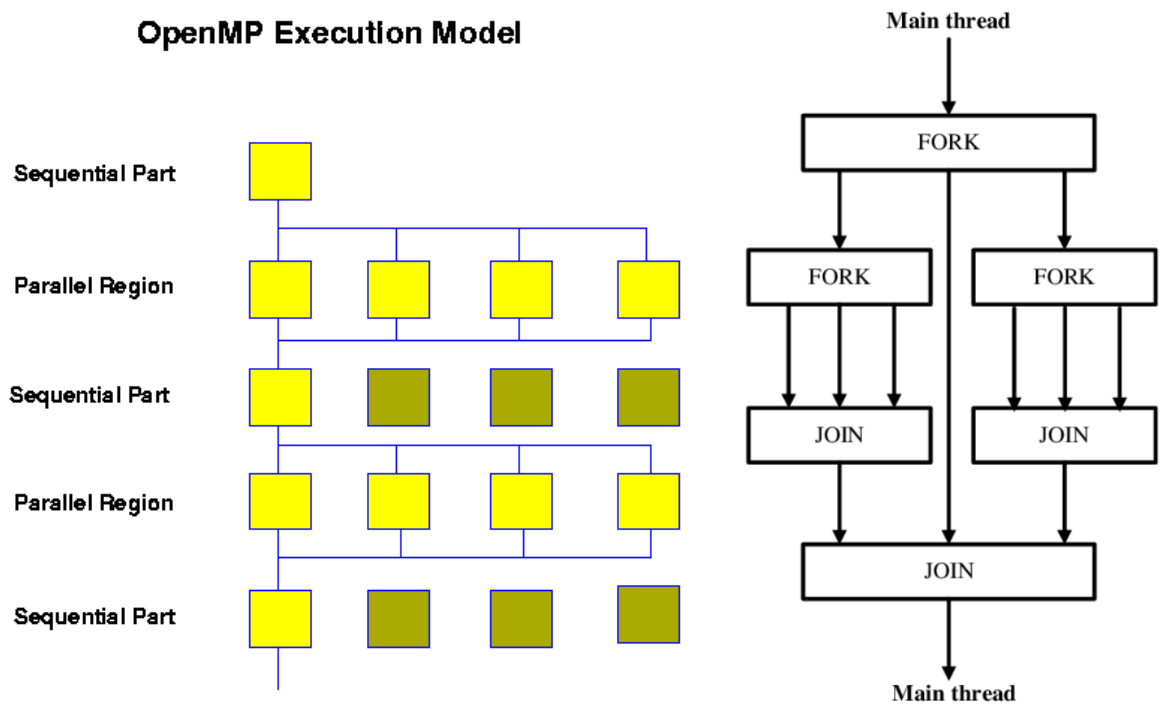



Рис. 1.13. Выполнение MPI-программы

Обмен данными между процессами осуществляется с помощью библиотеки из порядка 130 функций, выполняющих инициализацию процессов, парные (между двумя узлами) и коллективные (между группой узлов) коммуникации, формирование топологии групп и другие операции. Синхронизация вычислений осуществляется с помощью барьерной синхронизации – коллективной операции, которая должна быть вызвана всеми процессами группы для продолжения выполнения программы. Основными особенностями технологии программирования на основе MPI являются реализация только мультипроцедурных вычислений, необходимость детального управления распределением массивов и витков циклов между процессами, необходимость управления обменом сообщениями между процессами и ограничения на типы передаваемых данных. Зачастую программирование на MPI по сложности сравнивают с низкоуровневым программированием на ассемблере, а имеющиеся сравнения по эффективности [119] показывают невысокую продуктивность такого подхода, которая во многом зависит от скорости среды передачи данных и топологии вычислительного кластера. Высокая сложность разработки эффективных MPI-программ для произвольных размеров массивов и произвольного количества процессов делает практически невозможным повторное использование имеющихся программ. MPI не поддерживает методы и средства программирования графических ускорителей и, тем более, ПЛИС, поэтому использовать его для ресурсонезависимого программирования ГВС, содержащих не только процессорные узлы, но и графические ускорители и/или ПЛИС, невозможно.

Для многопроцессорных систем с общей памятью широко известна технология параллельного программирования OpenMP [120]. Технология OpenMP содержит набор директив компилятора, библиотеку функций и переменных окружения для поэтапного распараллеливания последовательной программы с помощью заданных пользователем директив. Вне параллельных областей OpenMP-программа [121] содержит обычный последовательный код, выполняющийся одним потоком (рис.1.14-а), а в отмеченных директивой распараллеливания областях происходит автоматическое создание дочерних

параллельных потоков(рис.1.14-б), которые уничтожаются по выходу из параллельной области в результате барьерной синхронизации.




а) б)

Рис. 1.14. Технология программирования OpenMP

а) модель выполнения OpenMP-программы; б) схема порождения процессов

Обмен данными между потоками осуществляется с помощью общих переменных. Для OpenMP характерны ограниченная область применения (мультипроцессоры и системы с общей памятью), необходимость низкоуровневого описания модели выполнения в терминах нитей и распараллеливание циклов с зависимостями по данным, что приводит к созданию плохо переносимых программ. Модель параллельного программирования OpenMP, как и модель MPI, обязывает программистов самостоятельно организовывать параллельные процессы и использовать низкоуровневые примитивы передачи сообщений и синхронизации. Если для программирования графических ускорителей есть примеры относительно успешного использования OpenMP, то для систем, содержащих ПЛИС в качестве основного вычислительного ресурса, она неприменима. Поэтому технология программирования OpenMP, так же как и MPI, не может быть использована для ресурсонезависимого программирования ГВС, содержащих

не только процессорные узлы и/или графические ускорители, но и ПЛИС, без существенных и фундаментальных доработок.

Исторически одна из первых технологий параллельного программирования с переносимостью программ – параллельная версия языка программирования Fortran – HPF (High Performance Fortran) [122] использовала директивы компилятора для изменения хода реализации программы, оператор FORALL, ряд библиотечных процедур и примитивных операций для параллельных вычислений при однопотоковой модели выполнения с использованием глобального адресного пространства. Из-за неэффективной технологии компиляции, отсутствия гибких распределений данных и требуемых для отладки параллельных приложений инструментальных средств, сложности взаимосвязей между программой и производительностью [123] HPF не обрел достаточной популярности и широкого распространения среди разработчиков параллельных программ, но привнес ряд важных идей в идеологию высокопроизводительных вычислений - автоматизацию распараллеливания на основе указаний пользователя наряду с совместимостью с последовательной программой. Как и предыдущие рассмотренные технологии, HPF ориентирован на программирование процессорных архитектур и не содержит средств организации вычислений для графических ускорителей и ПЛИС, поэтому непригоден для программирования ГВС.

Развитием идей MPI, OpenMP и HPF стала созданная в Институте прикладной математики им. М.В. Келдыша РАН технология программирования DVM (Distributed Virtual Memory и Distributed Virtual Machine) [124], которая использует единое адресное пространство распределенной виртуальной памяти виртуальных машин при двухступенчатой схеме отображения данных и вычислений на реальную МВС (рис. 1.15). Программирование осуществляется на языках C-DVMH и Fortran-DVMH [124] с директивами компилятору для распараллеливания циклов и использования групповых асинхронных взаимодействий процессоров в единой системе поддержки, отладки, оценки и предсказания производительности.

Рис. 1.15. Технология программирования DVM

При трансляции программы осуществляется ее перевод в машинно-независимую DVM-модель. Ограничением DVM-системы так же, как и моделей MPI-OpenMP, является необходимость явного указания распараллеливания циклов с зависимостями по данным, что требует высокой квалификации программиста и существенно ограничивает переносимость программ, даже несмотря на возможность повторного использования процедур за счет динамической настройки параметров приложения. Поскольку технология программирования DVM, как MPI и OpenMP, ориентирована на мультипроцедурные программы, требующие явного указания распараллеливания, использовать ее для ресурсонезависимого программирования ГВС без масштабных доработок невозможно.

Известной технологией параллельного использования взаимосвязанных компьютеров различных архитектур является общедоступный программный пакет PVM – Parallel Virtual Machine (параллельная виртуальная машина) [125, 126], который является своего рода композицией технологий MPI и OpenMP. PVM, более простой в освоении, чем MPI, и более гибкий, чем OpenMP, позволяет функционировать даже в гетерогенной вычислительной среде, но наследует и архитектурные недостатки своих предшественников: сложный

программный интерфейс, требующий от пользователя настройки коммуникации процессов при написании программы, необходимость установки пакета на все узлы МВС, падение производительности при межузловом взаимодействии. Отсутствие средств поддержки выполнения вычислений на графических ускорителях и ПЛИС ограничивают возможности использования технологии PVM, как и языка непроцедурных вычислений НОРМА [127-129], разработанного тем же коллективом, для программирования ГВС.

Параллельная среда Java FJTask [130] реализует концепцию параллелизма "разветвление-объединение" для языка программирования Java при распараллеливании сложных вычислений с помощью облегченных аналогов потоков Thread. Java FJTask реализует параллелизм на основе задач без блокировок, взаимодействия и произвольной синхронизации, поскольку не существует способа приостановки и продолжения отдельных задач во время их выполнения – поэтому задачи не должны содержать бесконечные циклы. Естественным недостатком такой технологии распараллеливания является ручное распределение подзадач для Java FJTask, что весьма трудоемко и может быть реализовано далеко не для всех задач, поэтому область применения этой технологии крайне узка, а переносимость программ крайне ограничена. Ручное распределение задач и отсутствие средств поддержки графических ускорителей и ПЛИС как целевых вычислительных архитектур не позволяют использовать Java FJTask для ресурсонезависимого программирования ГВС.

В 1995 году А.И. Легаловым [131] был предложен «**П**араллельный **И**нформационно-**Ф**ункциональный **А**л**ГОР**итмический» (ПИФАГОР) язык, предназначенный для разработки переносимых (архитектурно-независимых) параллельных программ. Функционально-потоковый язык программирования «ПИФАГОР» [132], построенный на основе принципа однократного присваивания, описывает вычисления в модели максимального параллелизма, без учета ресурсных ограничений реальных вычислительных систем [133], что упрощает программную реализацию алгоритмов и обеспечивает переносимость и архитектурную независимость разрабатываемых программ.

Архитектура вычислительной системы учитывается на этапе выполнения при сжатии параллелизма инструментальными средствами (транслятором, интерпретатором и генератором управляющего графа). К сожалению, язык «ПИФАГОР» не получил достаточно широкого практического распространения из-за отсутствия развитых инструментальных средств, обеспечивающих трансляцию для различных вычислительных архитектур, но имеет важно теоретическое значение как одна из первых теоретически обоснованных попыток разработки архитектурно-независимых параллельных программ. Отсутствие средств поддержки выполнения вычислений на графических ускорителях и ПЛИС ограничивают возможности использования языка «ПИФАГОР» для программирования ГВС.

Высокая реальная производительность систем с мультипроцедурной организацией вычислений достигается при условии, что время обработки информации существенно превышает время выполнения процедур обмена данными, что выполняется для множества отдельных задач либо слабосвязанной задачи. Поэтому реализация большинства параллельных алгоритмов и параллельных программ решения сложных задач с помощью существующих технологий параллельного программирования на практике существенно ограничена сверху по числу эффективно используемых процессоров, как правило, от 8 до 16 вычислительных узлов [2]. При дальнейшем увеличении доступного аппаратного ресурса МВС организация и обслуживание межпроцессорных обменов занимают гораздо больше времени, чем непосредственно вычисления, причем в ряде случаев даже для слабосвязанных задач [2].

Все рассмотренные технологии программирования мультипроцедурных вычислений реализуют только информационно-независимый (параллельный) способ одновременных вычислений, выполняемых на физически различных узлах вычислительной системы для информационно-независимых входных данных, и не содержат средств описания и реализации конвейерного способа одновременных вычислений. Параллельный способ организации одновременных вычислений эффективен только для несвязанных или слабосвязанных задач, в случае задач средней и сильной связности,

требующих интенсивных обменов данными между одновременно выполняемыми вычислениями, организация процедур межпроцессорных обменов требует больше времени, чем непосредственно сами вычисления. Используемые в большинстве технологий средства обмена данными и синхронизации (общие переменные и память, барьерная синхронизация) требуют ручного распределения параллельно выполняемых трудоемких фрагментов, высокой квалификации программиста для своей реализации, снижают производительность вычислительной системы из-за «узкого горла» и блокировок при доступе к одному ресурсу и крайне ограничивают возможности переноса (портации) программы на другую конфигурацию и, тем более, архитектуру вычислительной системы. Таким образом, рассмотренные технологии программирования мультипроцедурной организации вычислений не предназначены для описания конвейерных вычислений и не поддерживают вычислительные архитектуры графических ускорителей и ПЛИС, что не позволяет их использовать в качестве основы для ресурсонезависимого программирования.

Сложность создания параллельных программ и синхронизации вычислений программистом для мультипроцедурной организации вычислений при кажущейся простоте алгоритмизации этого процесса подвигла разработчиков на создание средств автоматизации – распараллеливающих компиляторов. Распараллеливающий компилятор из входной последовательной программы восстанавливает параллелизм исходного алгоритма, выявляет участки и фрагменты, которые можно выполнить одновременно (например, итерации циклов), и добавляет инструкции для их параллельного исполнения на узлах МВС. Ранее считалось, что автоматическое распараллеливание последовательных программ позволит справиться с растущей при увеличении числа узлов МВС сложностью их создания и синхронизации вычислений, выполняющихся на разных узлах. Однако автоматическое распараллеливание последовательных программ оказалось само по себе вычислительно трудоемкой задачей из-за крайне большого числа анализируемых вариантов.

Если для МВС с общей памятью компилятор, как правило, распределяет сосредоточенные в циклах вычисления на ядра процессоров, то для наиболее распространенных кластерных МВС с распределенной памятью по узлам распределяются не только вычисления, но и использующиеся в них данные для минимизации числа обращений к данным других процессоров [104]. При автоматическом распараллеливании последовательной программы для МВС с распределенной памятью компилятор равномерно распределяет все вычисления по используемым узлам МВС без разбиения на подзадачи. Анализ размещения данных по узлам выполняется для устранения возможной нелокальности данных, возникшей при механическом распределении вычислений без учета зависимостей (Read-After-Write, Write-After-Read, Write-After-Write, Read-After-Read) [134] исходной программы. Таким образом, распараллеливающий компилятор подбирает один параметр – коэффициент распараллеливания – в зависимости от числа используемых узлов МВС по критерию пространственной локальности (spatial locality) данных с учетом зависимостей исходной программы. Поэтому вычислительная сложность автоматического распараллеливания для МВС с распределенной памятью существенно возрастает, потому что для каждого варианта распараллеливания фрагментов последовательной программы компилятору необходимо оценивать и ранжировать варианты размещения данных по процессорам с учетом характеристик коммуникационной сети, что многократно расширяет пространство перебора. Для учета размещения данных и сокращения числа пересылок с помощью группировки обменов данными или их совмещения с вычислениями требуется анализировать индексные выражения между всеми циклами программы (а не одним, как для общей памяти). Для кластерной МВС, содержащей $n$ узлов, все многообразие вариантов распределения данных по узлам можно описать $n$-арным деревом (при условии, что каждый узел обрабатывает собственную локальную порцию данных). Согласно теореме Кэли, число вариантов размещения данных оценивается числом различных деревьев для $n$ вершин, которое составляет $n^{n-2}$. Уже для МВС масштаба университетской кафедры с 64 вычислительными узлами число вариантов возможных размещений данных

составит $64^{62} = 2^{372}$, а для кластерных вычислительных систем, использующихся в научных расчетах и промышленности с числом узлов не менее 512, число размещений составит не менее $512^{510} = 2^{4590}$. Анализ такого обширного числа вариантов полным перебором для выбора наилучшего на современных компьютерах в приемлемое время невозможен, особенно при отсутствии удобного критерия оценки эффективности синтезируемых версий параллельной программы.

Для сокращения пространства поиска задачу автоматического распараллеливания принято разделять на две подзадачи: анализ последовательной программы и ее преобразование в параллельную. Для каждой из подзадач разработаны свои формальные и эвристические методы, причем большинство исследований в области автоматического распараллеливания посвящены преимущественно эвристическим методам анализа для сокращения пространства поиска – например, анализ информационных зависимостей [135, 136], гнезд и витков циклов [136, 137], приватных и редукционных переменных[138], канонизация, раскрутки и развертки циклов [136-139] и др. Проведенные в 90-х годах 20 века неоднократные попытки различных исследователей [140] автоматизировать решение обеих подзадач автоматического распараллеливания для МВС с распределенной памятью показали [135, 139, 140], что для реальных, а не тестовых, задач это возможно только в крайне редких случаях. Низкая эффективность автоматически создаваемых распараллеливающим компилятором программ для МВС с распределенной памятью привела к практически полному прекращению исследований в этой области [135, 140]. Поэтому автоматическое распараллеливание также неприменимо для программирования ГВС.

## 1.5. Технологии программирования графических процессоров

Наибольшее распространение для программирования графических ускорителей и вычислительных систем на их основе получили технологии программирования CUDA, OpenAcc и OpenCL. Созданная компанией Nvidia

технология программирования CUDA (англ. *Compute Unified Device Architecture*) [141-144], является, пожалуй, наиболее известной и позволяет описывать выполняемые на графическом ускорителе алгоритмы с помощью подключения специальных библиотечных функций для упрощённого диалекта языка программирования C. В технологии CUDA графический ускоритель является массивно-параллельным сопроцессором к центральному процессору, поэтому для реализации вычислений в этой модели пишется единая для процессора и графического ускорителя программа на языке C, в рамках которой программист имеет доступ к ресурсам графического ускорителя с помощью вызова специальных библиотек. Использование широко распространенного и хорошо известного программистам языка программирования C с небольшим числом новых конструкций (спецификаторы типа, встроенные переменные и типы, директива запуска ядра) обусловило популярность этой технологии по сравнению с более ранними специализированными библиотеками, такими как DirectX [145] и OpenGL [146]. Типовая программа графического ускорителя по технологии CUDA содержит следующие этапы: выделение памяти на GPU, копирование данных из памяти процессора в выделенную память GPU, выполнение вычислений на GPU, копирование результатов вычислений обратно в память CPU и освобождение выделенной памяти GPU.

Вычисления на GPU выполняются в модели SIMT [142] (Single Instruction, Multiple Thread - по терминологии компании-разработчика NVidia), содержащей большое число (порядка нескольких тысяч) одновременно выполняющихся «легких» нитей (threads) с минимальным контекстом для минимизации накладных расходов на создание, управление и уничтожение нити. Все запущенные на выполнение нити организованы в иерархический массив - сетку (grid), разделенный на блоки одинакового размера (рис.1.16), каждый из которых представляет собой многомерный массив нитей, поделенных на группы (warp) по 32 нити с возможностью индексного доступа к отдельной нити. Нити в пределах группы (warp) выполняются физически одновременно, а управление выполнением групп осуществляет планировщик

Giga Thread, входящий в состав GPU, поэтому нити разных групп могут находиться на разных стадиях выполнения программы.

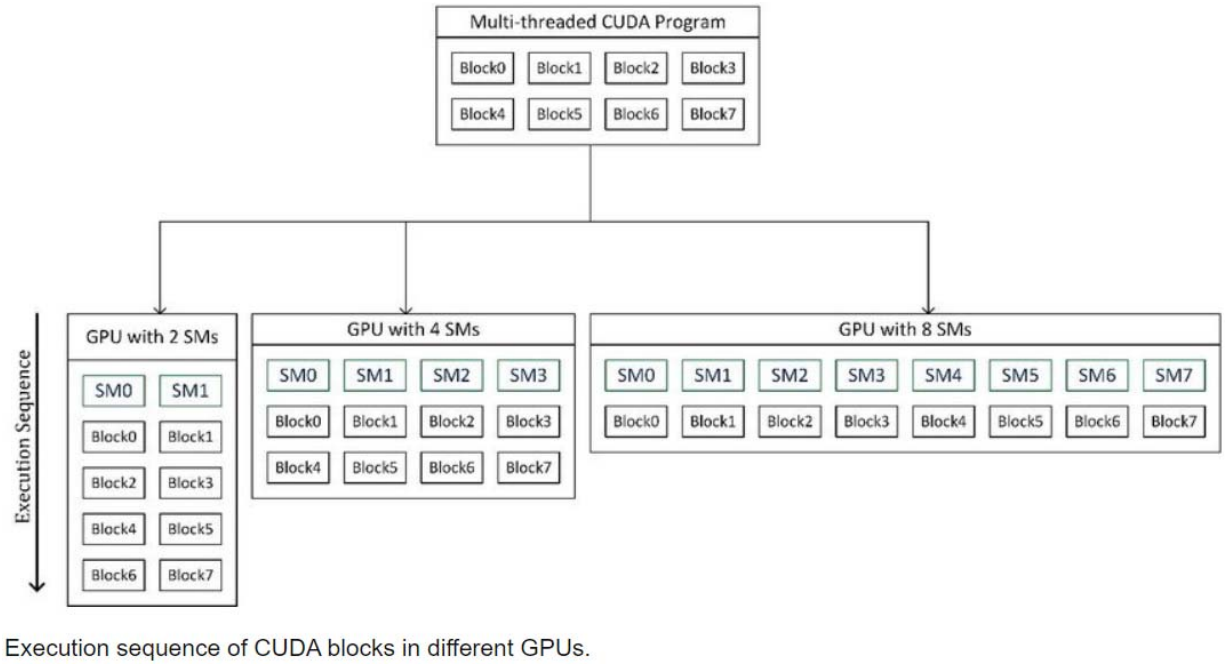



Рис. 1.16. Параллельные вычисления в технологии программирования CUDA

В программе на языке C для CUDA программист с помощью спецификаторов функций и переменных должен явно указать, где будет выполняться функция (на CPU или GPU), в какой памяти находятся ее операнды и результат и откуда функция может быть вызвана. Серьезным ограничением является возможность взаимодействия нитей только в пределах блока с помощью разделяемой памяти малого размера (16 Кб) и барьерной синхронизации. Память графического ускорителя очень ограничена как по объему, так и по возможности доступа к ней: глобальная, текстурная и константная память доступны только из CPU, а малый объем регистровой, локальной и разделяемой памяти, как правило, не позволяет организовать эффективную подкачку данных из CPU в GPU [143]. Декларируемыми [142-144] преимуществами технологии CUDA являются простота использования и перехода от традиционных процедурных вычислений к «расширенному» языку C, большой набор поддерживаемых решений и стандартных библиотек численного анализа, FFT (быстрое преобразование Фурье) и BLAS (линейная алгебра) и кроссплатформенность (поддержка 32- и 64-битных операционных систем Windows, Linux и MacOS).

Анализ программной модели CUDA показывает, что она, как и архитектура графического ускорителя, ориентирована на решение несвязанных или слабосвязанных прикладных задач, содержащих большое число вычислительных операций и не требующих для своего решения больших объемов памяти, т.к. объемы быстрой регистровой, локальной и разделяемой памяти очень малы (меньше объемов кэш-памяти современных процессоров), а латентность глобальной памяти очень велика (400-600 тактов). Используемый для описания мультипроцедурной в случае CUDA организации вычислений язык программирования C описывает граф потока управления и изначально не предназначен для описания одновременных конвейерных вычислений. Ресурсонезависимое программирование ГВС с помощью CUDA возможно только для графических ускорителей и процессоров, но не для ПЛИС. Поскольку компания Nvidia прекратила развитие технологии CUDA, ожидать появления средств поддержки вычислений на ПЛИС не приходится. Поэтому использование технологии CUDA и языка программирования C для программирования ГВС не позволит описывать все многообразие других форм организации вычислений и в полной мере использовать все преимущества архитектуры ПЛИС.

Известный открытый стандарт OpenACC [147, 148], являющийся в определенном смысле «переработкой» OpenMP для графических ускорителей, предоставляет высокоуровневую модель программирования, не зависящую от операционной системы и модели графического ускорителя. Технология OpenAcc автоматизирует выгрузку вычислительно-трудоемких участков последовательной программы на языках C/C++ или Fortran на графический ускоритель. Выгрузку кода и организацию его запуска реализует компилятор (рис.1.17) на основе сделанной программистом разметки текста программы с указанием типа параллелизма конструкции: группового (крупнозернистого), исполнительного (мелкозернистого) или векторного (мелкозернистого параллелизма на исполнительном уровне) [147]. Модель выполнения OpenACC [148] не требует явного указания барьерной синхронизации и берет на себя завершение реализации групп перед началом выполнения других групп так же, как и автоматическое порождение и завершение нитей на

графическом ускорителе. Как и в технологии CUDA, в OpenACC память процессора и память графического ускорителя физически разделены, что, наряду с автоматическим созданием контекста нитей, существенно снижает производительность графического ускорителя и ограничивает класс решаемых задач только слабосвязанными задачами.

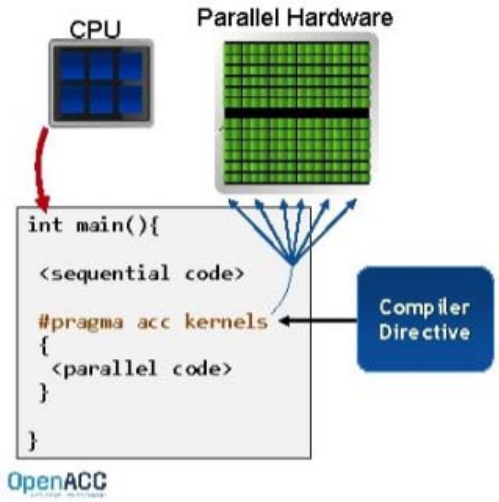



Рис. 1.17. Параллельные вычисления технологии программирования OpenAcc

В OpenACC пропускная способность памяти между процессором и графическим ускорителем определяет уровень вычислительной интенсивности для эффективного ускорения участка программы, поэтому ограниченный размер памяти графического ускорителя может препятствовать выгрузке участков программы, которые оперируют большими объемами данных. Перечисленные недостатки OpenAcc существенно сокращают его применимость для портации прикладных программ на разные архитектуры и конфигурации ГРВС.

Для задач, содержащих в своей структуре как конвейерные, так и процедурные фрагменты вычислений, реализуемых на вычислительных узлах различных архитектур (процессоры, графические ускорители, специализированные процессоры, ПЛИС), в 2008 году под эгидой компании Khronos Group создан переносимый индустриальный стандарт OpenCL [149, 150], предоставляющий программисту универсальный инструмент для использования всех вычислительных мощностей большинства современных гетерогенных вычислительных архитектур. OpenCL как универсальное

средство программирования разнородных вычислительных архитектур рассматривает прикладную задачу в терминах четырех моделей: модели платформы, модели выполнения, модели памяти и модели программирования, каждая из которых позволяет определить способ выполнения вычислений.

Модель платформы позволяет задать высокоуровневое описание гибридной (гетерогенной) системы, центром которой является управляющий модуль – хост (host), осуществляющий как взаимодействие с пользователем, так и управление процессом вычислений на устройствах (devices) системы, каждое из которых может содержать несколько вычислительных модулей (compute units) с набором обрабатывающих элементов (processing elements) [150].

Модель вычислений описывает абстрактное представление выполнения потоков инструкций в гибридной вычислительной системе, основанное на представлении функций на языке OpenCL C в виде набора ядер, каждое из которых выполняется на определенных контекстом устройствах системы. Контекст определяет среду выполнения ядер как устройства, сами ядра, программные объекты (program objects, исходный и выполняемый код будущих ядер), объекты памяти (memory objects). Запуск и управление вычислениями осуществляется хостом с помощью очереди команд для каждого устройства. Очередь команд допускает последовательное (in-order execution) или внеочередное (out-of-order execution) выполнение команд (выполнение ядер, управление памятью и синхронизация выполнения команд в очереди).

Модель памяти (memory model) описывает набор регионов памяти (memory objects), которые могут быть двух типов: буферные объекты (buffer objects) – доступные всем ядрам непрерывные участки памяти и объекты-изображения (image objects), которые могут располагаться в памяти хоста или устройства с различными механизмами доступа.

Модель программирования (programming model) задает два типа модели программирования: параллелизм по данным (data parallelism) и параллелизм по заданиям (task parallelism). Параллелизм по данным применим к структурам данных: каждый элемент определенной хостом структуры данных

обновляется одновременно (параллельно) с копиями одного и того же OpenCL-ядра. Организация такой структуры должна поддерживать возможность одновременного изменения различных её частей. Для взаимодействия нескольких ядер используется либо буферная память, либо барьерная синхронизация, а параллелизм выполнения ядер зависит от типа устройства.

Параллелизм по заданиям реализует масштабирование вычислений при возможности их параллельного выполнения на доступном вычислительном ресурсе, позволяя системе исполнения OpenCL самостоятельно планировать запуск множества различных заданий, в том числе связанных в граф исполнения с помощью OpenCL-событий. OpenCL-программы, выполняемые в среде исполнения OpenCL Runtime [149], выбирают подходящие ресурсы гетерогенной системы из списка доступных, создают последовательность инструкций, которые будут выполняться на ресурсах, готовят начальные данные для вычислений, выполняют OpenCL-инструкции обработки данных на каждом устройстве и собирают результаты вычислений. В целом, OpenCL является достаточно универсальным инструментом, но требует от программиста непосредственно управлять перемещением данных, определять места хранения переменных в иерархии памяти и вручную реализовывать параллелизм в коде. Серьезным недостатком стандарта OpenCL является отсутствие автоматической реализации фрагментов вычислений на нескольких ядрах (процессорах и/или графических ускорителях и/или ПЛИС). Возможно, именно эти ограничения не позволили технологии OpenCL стать стандартом программирования гетерогенных ГВС и завоевать всеобщую популярность среди разработчиков программ для высокопроизводительных вычислительных систем. В настоящее время технология OpenCL все реже упоминается как средство программирования ГВС.

Резюмируя рассмотрение существующих технологий программирования, можно утверждать, что на сегодняшний день ни одна из широко распространенных технологий программирования процессорных узлов (MPI, HPF, OpenMP, HOPMA, DVM и др.)[151], графических ускорителей (CUDA, OpenACC и др.) и/или гибридных вычислителей

(OpenCL) в полной мере не позволяет описать и реализовать прикладные задачи для гибридных вычислительных систем, содержащих разнородные вычислительные узлы.

## 1.6. Технологии программирования ПЛИС

Исходной формой представления задачи для ПЛИС является описание порядка обработки данных, а не передачи управления. Для этого также используют графовые формы [1] – граф алгоритма, граф зависимости, граф влияния, история реализации программы (решетчатый граф алгоритма) и информационный граф [1, 75, 102], которые представляются ориентированным графом, где дуги задают информационную зависимость – отношение между вершинами графа, при котором выходной аргумент одной вершины является входным аргументом другой (или входной вершиной).

В отличие от характерной для процессоров и графических ускорителей мультипроцедурной организации вычислений для реконфигурируемых вычислительных систем, программа состоит из двух компонентов: конфигурации вычислительной системы, реализующей логическую структуру специализированного вычислительного устройства (структурный компонент программы), и программы, управляющей потоками данных для этого устройства (потоковый компонент программы). Поэтому программированием РВС занимаются два специалиста: инженер-схемотехник разрабатывает и синтезирует конфигурацию вычислительного устройства, а прикладной программист разрабатывает программу управления потоками данных на традиционном языке высокого уровня C, C++, C#. Для программирования ПЛИС используются две технологии: схемотехническое программирование в графическом представлении всех операционных элементов цифровой схемы и/или высокоуровневое описание цифровой схемы на языке программирования аппаратуры HDL-группы (VHDL, AHDL, Verilog и др.). Для описания устройства в виде наглядных и привычных схемотехникам принципиальных схем и условных графических образов компонентов и функциональных блоков также используются схемотехнические редакторы,

входящие в состав большинства современных средств разработки Xilinx ISE, Xilinx Vivado, Intel Altera Quartus, Altium Designer и др.

Схемотехническое программирование являлось исторические первой технологией программирования ПЛИС, но в настоящее время все больше вытесняется текстовым описанием задачи на языках описания аппаратуры HDL или в средствах высокоуровневого синтеза HLS. Общим недостатком как текстового, так и графического способов представления логической структуры проектируемого решения является ограниченность одним кристаллом ПЛИС. Для многокристальных РВС или множества кристаллов ПЛИС, объединенных в единый вычислительный ресурс в составе ГВС, это требует от разработчика самостоятельной декомпозиции задачи на фрагменты, реализуемые в одном кристалле ПЛИС с учетом его внутренней архитектуры, и синхронизации вычислений между кристаллами с учетом топологии связей между микросхемами ПЛИС. Сроки разработки решения прикладной задачи для многокристальных РВС достаточно велики и составляют 4-12 месяцев в зависимости от сложности задачи.

Существующие высокоуровневые средства программирования ПЛИС, такие как Mitrionics [152], Celoxica [153], Impulse Accelerated Technologies [154] и др., основаны на специализированных подмножествах языка программирования C, имеют синтаксис, похожий на синтаксис языка C, но отличаются семантикой использования конструкций, ориентированных на параллельное программирование, разными подходами к программированию ПЛИС.

Одной из наиболее известных технологий высокоуровневого программирования ПЛИС является программный комплекс компании Mitrionics, который создает в ПЛИС один виртуальный процессор Mitrion VirtualProcessor (MVP). Программирование MVP осуществляется на языке высокого уровня Mitrion-C [155] с библиотеками функций для построения хост-программ MithalAPI, входящих в пакет разработки Mitrion SDK. Mitrion-C является специализированным языком высокого уровня семейства C, но работает с потоком данных, а не с потоком команд, реализуя параллелизм на уровне инструкций и циклов. На уровне циклов используется явный,

определяемый пользователем с помощью ключевого слова «Foreach», параллелизм, а параллелизм на уровне инструкций является неявным, основой которого служит выполняемый автоматически при компиляции программы анализ зависимостей данных. Порядок выполнения инструкций не определен и зависит от степени готовности данных. Из-за недетерминированности, вызванной неявным описанием параллелизма на уровне инструкций, используется правило однократного присваивания [6, 102, 131]. Правила доступа к данным в языке определяются на уровне описания массивов, которые делятся на три типа: список, поток и вектор. Доступ к элементам списка осуществляется только последовательно, а вектор позволяет обращаться к любому элементу. Использование векторов ограничивается ресурсами ПЛИС, поэтому на практике вектора большого размера используются очень редко. Поток представляет собой последовательность элементов, обработка которых осуществляется по принципу конвейера (пока одни данные считываются из памяти, вторые обрабатываются, третьи подаются на выход). Программа на языке Mitrion-C должна быть полностью реализована на одном виртуальном процессоре MVP. Поэтому Mitrion-C не позволяет программировать многокристальные РВС, а следовательно, существенно снижает эффективность использования поля ПЛИС. Для нескольких кристаллов ПЛИС в составе ГРВС программа на языке MitrionC вырождается в MPI-программу для кластера (множество процессоров MVP), реализованную в поле ПЛИС, частота работы которых на десятичный порядок ниже частоты работы процессора, что приводит к катастрофически низкой производительности. Возможно, это стало одной из причин прекращения исследований (сайт компании не обновлялся с 09.05.2007).

Еще одной известной технологией высокоуровневого программирования ПЛИС является разработанный компанией MentorGraphics язык Handel-C [155], представляющий программисту возможности явного указания параллельно исполняемых инструкций с помощью команды par, как в языке Occam, однако широкого распространения эта технология не получила из-за низких частот, удельной производительности и эффективности синтезируемого решения, но явилась основой для других разработок,

например CatapultC. Общим для этих средств программирования и наиболее существенным недостатком является попытка реализации аналога специализированного процессора в аппаратном поле ПЛИС. Такой подход является заведомо неэффективным с учетом разницы в частотах работы современных процессоров и ПЛИС на один десятичный порядок.

В настоящее время для программирования ускорителей на ПЛИС все большее распространение получают средства высокоуровневого синтеза - трансляторы программ на языке С в конфигурационные файлы ПЛИС [156-162], преобразующие последовательные программы в схемотехническую конфигурацию специализированных аппаратных средств на языках HDL-группы. Они получили название средств высокоуровневого синтеза или HLS-компиляторов. По типу входного языка программирования средства HLS могут быть разделены на две основные категории (рис.1.18) [156]: трансляторы проблемно-ориентированных языков (Domain Specific Language) и трансляторы языков общего назначения (General Purpose Language). На рис.1.18 приняты следующие условные обозначения: символ ► означает, что HLS-компилятор разрабатывается и поддерживается, || - проект приостановлен и нет данных о дальнейших планах развития, ■ – проект завершен и больше не поддерживается.

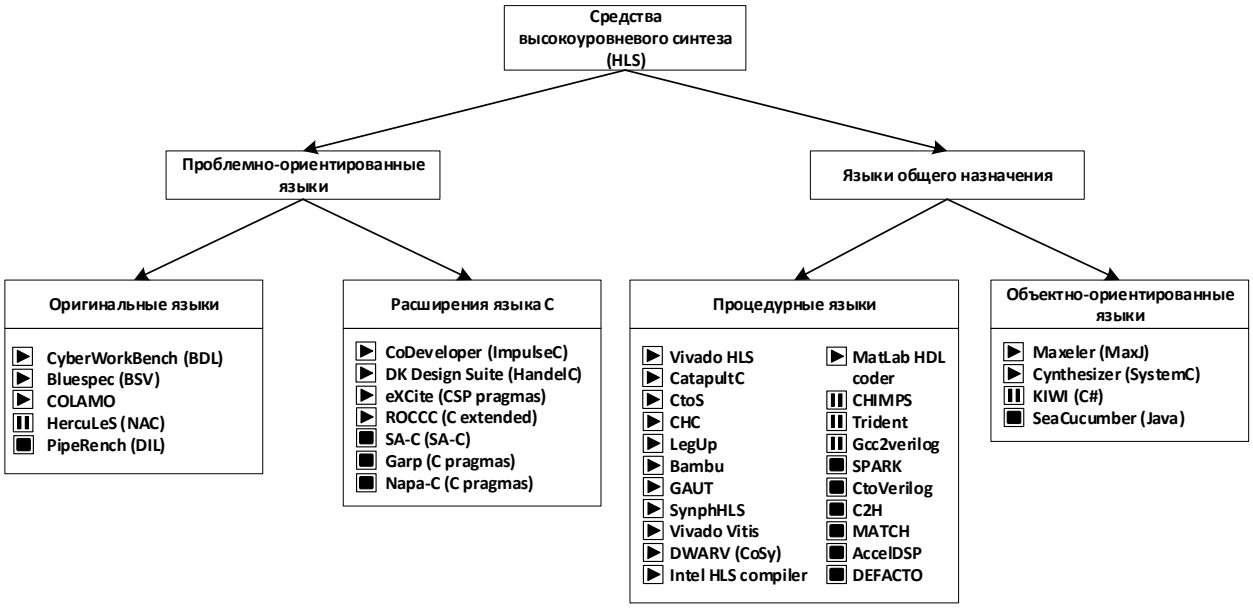



Рис.1.18. Классификация средств высокоуровневого синтеза

В настоящее время развиваются как академические (разработанный в Делфтском технологическом университете DWARV [157], разрабатываемый в Politecnico di Milano BAMBU [158] и разработанный в Университете Торонто LEGUP [159]), так и коммерческие (CatapultC, Intel HLS Compiler, Cadence Stratus HLS, Xilinx Vivado HLS [161] и Vivado Vitis [162]) комплексы, которые используются для проектирования высокопроизводительных и энергоэффективных решений. Подробный обзор и сравнительный анализ возможностей HLS-компиляторов, приведенных на рис. 1.18, сравнение характеристик, достоинств и недостатков приведен в [156, 160]. Подавляющее большинство приведенных в [156] HLS-компиляторов анализируют вычислительно трудоемкий фрагмент программы на языке C и преобразуют его в специализированный вычислитель (IP-ядро), синтезируемый на основе автоматной модели или процессорной парадигмы. Несмотря на существенный выигрыш в скорости вычислений по сравнению с процессорной реализацией (по данным обзора [156]), результатом их работы является занимающее сравнительно небольшой аппаратный ресурс IP-ядро, а организация потоков данных для него и масштабирование даже в пределах кристалла ПЛИС возлагается на пользователя. Поэтому поиск рационального решения всей задачи для доступного аппаратного ресурса является целиком и полностью задачей программиста, которая многократно усложняется при использовании нескольких ПЛИС или многокристалльных РВС, содержащих множество ПЛИС, связанных пространственной коммутационной системой [84-101]. Поэтому ни один из упомянутых HLS-компиляторов не позволяет описать и реализовать прикладные задачи для гибридных вычислительных систем с разнородными вычислительными узлами.

## 1.7. Технология структурно-процедурных вычислений для многокристальных РВС и их программирование на языке высокого уровня COLAMO

Рассмотренные в предыдущем параграфе технологии программирования ПЛИС ориентированы на использование ПЛИС в качестве ускорителя для вычислительно-трудоемкого фрагмента задачи, а не основного

вычислительного ресурса. Использование ПЛИС как элементной базы для многокристальных систем с программированием архитектуры рассматривается в теории структурно-процедурных вычислений. В теории структурно-процедурной организации вычислений [75, 102], разработанной А.В. Каляевым и И.И. Левиным для программирования вычислительных систем, построенных на основе реконфигурируемой элементной базы, задача представляется в максимально параллельной форме в виде информационного графа [75, 102] – ориентированного графа, вершины которого соответствуют операциям над данными, а дуги отражают информационную зависимость между ними. Каждая операция над элементами данных представляется операционной вершиной, поэтому общее число операционных вершин соответствует числу операций над данными задачи, а число входных вершин соответствует размерности всех обрабатываемых данных. Операционные вершины информационного графа задачи (ИГЗ) распределены по слоям и итерациям: слои содержат информационно-независимые вершины, поэтому связи между вершинами в слое отсутствуют, а итерации описывают информационную зависимость вершин разных слоев между собой. ИГЗ является ациклическим, т.е. не содержит обратных связей. Операционные вершины в слоях и итерациях ИГЗ обычно представляют подграфы из нескольких арифметическими простых операций, связанными информационными зависимостями, среди которых могут быть условные или коммутационные операции.

ИГЗ – это модель вычислений максимально параллельной формы задачи с наглядным представлением структуры информационных связей. Он не учитывает характеристики аппаратной реализации, такие как латентность, длительность такта обработки, производительность и др., в нем отсутствуют потоки и интервал обработки данных. Аппаратная реализация ИГЗ в виде вычислительной структуры обладает минимальной латентностью и интервалом обработки данных, наименьшим временем решения и максимальной производительностью задачи, но требует множество вычислительных устройств и каналов памяти для одновременной подачи всех

заданных параметрами задачи входных данных и выполнения всех операций задачи, что обычно недостижимо для реальных вычислительных систем.

Для реализации задачи в вычислительной системе [75, 102] ИГЗ преобразуется в кадровую структуру [75, 102], содержащую вычислительную структуру, аппаратно-реализованную на доступном ресурсе, и правила организации входных и выходных потоков (рис. 1.19).

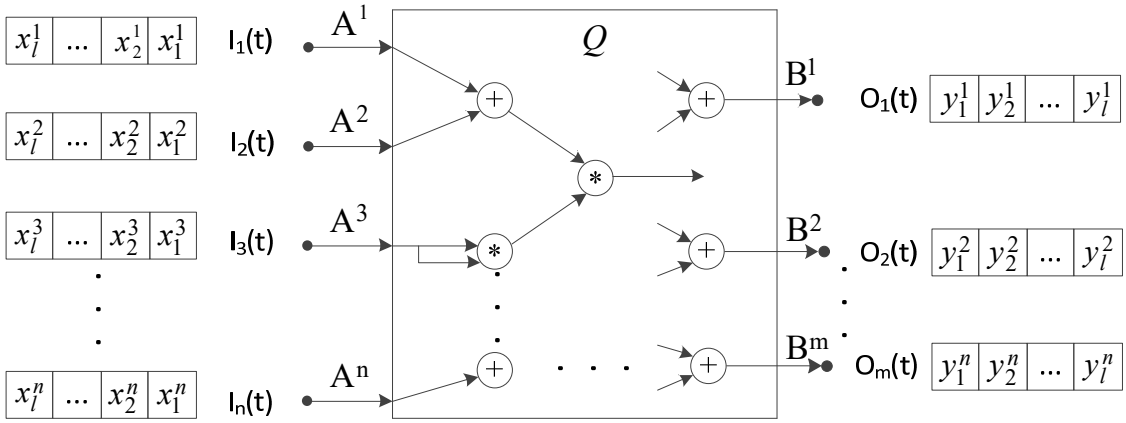



Рис. 1.19. Кадровая структура, реализующая вычисления ИГЗ

Вычислительная структура кадра представляет собой аппаратно-реализованный подграф ИГЗ $g_{jk}^{i}$, на вход которого последовательно подаются данные всех подграфов слоя. Таким образом, вычисления задачи реализуются линейным конвейером без обратных связей с доказанным [6, 75, 102] преимуществом в удельной производительности, по сравнению с процедурной организацией вычислений. Необходимым условием для выполнения такого преобразования является представление задачи в функционально-регулярной форме [75, 102], когда для расположенных в слоях и итерациях изоморфных подграфов $g_{jk}^{i}$ можно найти или построить регулярное функциональное соответствие (упорядочивание) слоевых $F_{ij}$ и итерационных $\Phi_i$ подграфов таким образом, чтобы представить ИГЗ $G$ функциональной зависимостью:

$$G = \Phi_i(F_{ij}(g_{jk}^{i})), \tag{1.1}$$

Условия, требования и методы приведения ИГЗ к функционально-регулярной форме описаны в [75, 102], а структура функционально-регулярного ИГЗ представлена на рис. 1.20, где $F_{ij}$ характеризует функцию упорядочивания информационно-независимых подграфов в вычислительном

слое $S_i, i = 1 \ldots N$ задачи, а $\Phi_i$ – функцию отображения информационно-зависимых вычислительных слоев $S_1, S_2, \ldots, S_N$ в реализации ИГЗ $G$ для изоморфных подграфов $G_{ij}$, воспринимаемых как минимальный неделимый фрагмент вычислений.

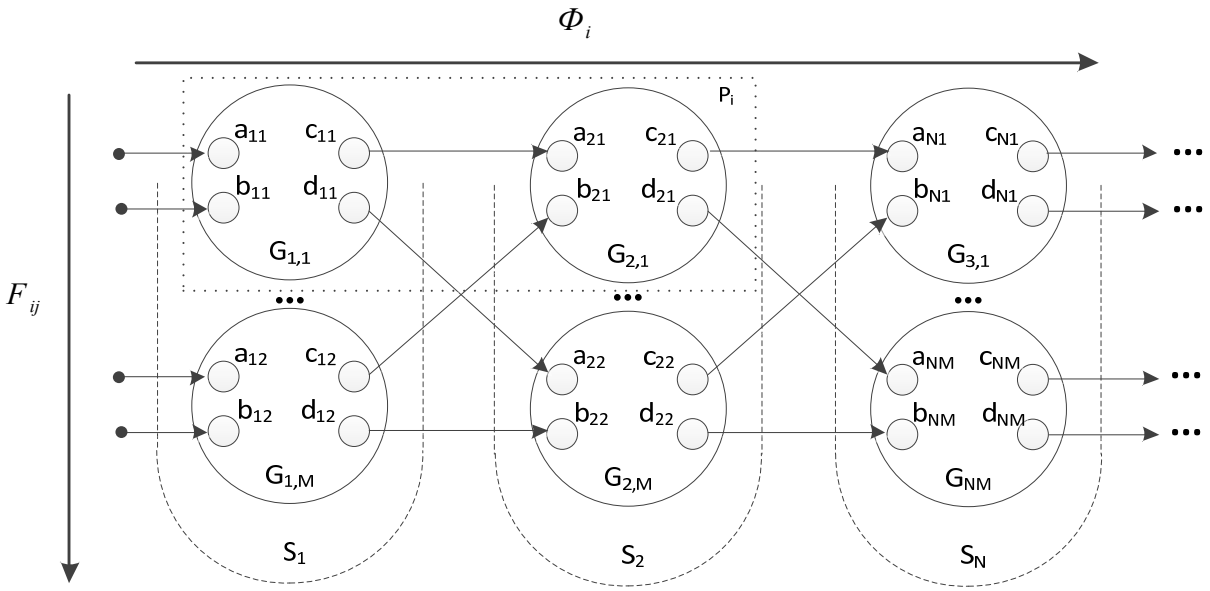



Рис. 1.20. Пример функционально-регулярного ИГЗ

Задачи с регулярной внутренней структурой легко представимы в функционально-регулярной форме, что позволяет синтезировать для них вычислительный конвейер с высоким быстродействием при сравнительно небольших аппаратных затратах. Примерами таких задач являются алгоритмы, содержащие в тексте программы циклы с явной индексацией элементов массивов и сравнительно небольшое число условных операторов, не превышающее 50% от общего числа операторов в теле цикла. Задачи, у которых сложно выделить внутреннюю структуру, описываются функционально-нерегулярным графом, который можно привести в функционально-регулярную форму с помощью дополнения слоев и векторизации [75, 102]. Основная сложность представления ИГЗ в функционально-регулярной форме заключается в выделении *базового* подграфа $g_{jk}^{i}$ в (1.1) или $G_{ij}$ на рис. 1.20, изоморфного остальным подграфам, поскольку критерии его формального выбора для прикладных задач в общем случае не определены. В работах [6, 75, 102] базовый подграф (БП) выделялся разработчиком вручную с учетом изоморфизм подграфа в структуре ИГЗ и

возможности его аппаратной реализации как вычислительной структуры кадра на доступном аппаратном ресурсе. Поэтому выбор БП в равной степени определяется структурой и информационными зависимостями ИГЗ, доступным аппаратным ресурсом и опытом разработчика. Для дальнейших рассуждений определим БП как реализуемый на доступном аппаратном ресурсе функционально-завершенный фрагмент ИГЗ, обеспечивающий решение прикладной задачи с помощью синтезированных для него функций чтения и записи (подачи входных и выходных потоков данных). Аппаратная реализация хотя бы одного БП в виде конвейерной вычислительной структуры в РВС является необходимым условием структурно-процедурной организации вычислений [75, 102]. Если на доступном аппаратном ресурсе возможно реализовать несколько БП, то с помощью технологии индуктивных программ [75, 102] возможно масштабирование реализации задачи, сокращающее время решения – построение мультиконвейерной вычислительной структуры или реализация нескольких информационно-зависимых итерационных ступеней $g_{jk}^{i}$. Масштабирование выполнялось высококвалифицированным инженером вручную, а способ – построение мультиконвейера или распараллеливание по итерациям – определялся аппаратным ресурсом $A_{RVS}$, который для РВС задается числом ПЛИС, числом ячеек LUT, MLUT, FF, числом блоков DSP и BRAM в каждой ПЛИС, числом и разрядностью каналов доступа к памяти, внутренних связей и т.д.

Таким образом, структурно-процедурная организация вычислений за счет учета информационных зависимостей и минимизациии непродуктивных операций пересылки позволяла выполнять вычисления прикладной задачи с достаточно высокой удельной производительностью, но решение задачи реализовывалось исключительно вручную и требовало высокой квалификации инженера. Отсутствие формальных критериев выделения БП и необходимость его ручной оптимальной схемотехнической реализации в виде конвейерной вычислительной структуры кадра ограничивают возможность применения СПОВ как технологии программирования ГВС.

Развитием идей СПОВ стала технология на основе языка программирования с неявным описанием параллелизма COLAMO, реализующая описание ИГЗ и кадровых структур синтаксическими конструкциями языка программирования. Язык COLAMO (Common Oriented Language for Architecture of Multi Objects), разработанный И.И. Левиным в 1987 году [6, 75, 102], предназначен для описания в архитектуре РВС кадровых структур с аппаратно реализованными фрагментами ИГЗ, последовательная смена которых позволяет выполнить все вычисления задачи. Конструкции языка позволяют описать как вычислительную структуру кадра из библиотечных элементов (сумматоров, умножителей, специальных функций, элементов синхронизации потоков данных и связей между ними), так и потоки чтения и записи данных, а также очередность смены кадров. Параллельная программа на COLAMO сочетает асинхронность выполнения операций вычислительной структуры кадра при обработке потока данных с детерминизмом выполнения фрагментов прикладной задачи в РВС.

В синтаксисе языка программирования высокого уровня COLAMO, описанном в [6], отсутствуют обязательные для MPI и OpenMP директивы распараллеливания и иные явные формы описания параллелизма. Распараллеливание описывается типом доступа к переменным и индексации элементов массивов. По типу доступа к элементам массивы в языке COLAMO могут разделяться на параллельный (векторный, заданный ключевым словом **Vector**) и потоковый (последовательный, заданный ключевым словом **Stream**) доступ к элементам массива. Комбинация параллельных и последовательных типов доступа к каждому измерению массива позволяет задавать варианты доступа к элементам массивов различной сложности. Тип доступа и информационные зависимости в индексации элементов массивов определяют параллельный или конвейерный способ реализации вычислительной структуры кадра, синтез которой осуществляет транслятор [75, 102]. Для разрешения коллизий при обращении к памяти на синтаксическом уровне, в языке COLAMO действуют правило однократного присваивания и правило единственной подстановки [75, 102] при доступе к переменным. Правило однократного присваивания позволяет для одной и той же переменной в теле

кадра выполнить однократную запись и неограниченное число процессов чтения. Правило единственной подстановки [75, 102] запрещает одновременные запись и чтение переменной несколькими процессами.

Язык программирования COLAMO позволяет в сжатой форме описывать различные сочетания параллельной и конвейерной форм организации одновременных вычислений задачи с помощью описания структур данных, масштабировать решение в соответствии с ТИП и изменять способ реализации вычислений. Однако, все параллельные программы на COLAMO создаются под определенный вычислительный ресурс РВС и не могут автоматически адаптироваться при его изменении, что осложняет портацию программ на РВС других конфигураций и архитектур. Также к недостаткам технологии программирования РВС на языке COLAMO можно отнести сложность гибкого автоматического изменения основных параметров вычислительных структур (числа вычислительных устройств, частоты работы, разрядности обрабатываемых данных), требующих ручной корректировки программы, и невозможность описания взаимодействия мультипроцедурных вычислений по аналогии с существующими (MPI, OpenMP, HOPMA и др.). Поэтому технология программирования на языке COLAMO не позволяет решить проблему взаимодействия различных вычислительных архитектур в едином контуре и, соответственно, не может быть использована как технология ресурсонезависимого программирования ГВС.

## 1.8. Анализ рассмотренных технологий и разработка принципов ресурсонезависимого программирования гибридных вычислительных систем

Для рассмотренных в этой главе технологий программирования процессоров (MPI, OpenMP, HOPMA, DVM, PVM), графических ускорителей (CUDA, OpenACC, OpenCL) и ПЛИС (MitrionC, HLS-компиляторы, COLAMO) характерны различные модели вычислений, отражающие архитектурные особенности аппаратного ресурса уже на уровне представления вычислений. Поэтому все перечисленные технологии позволяют реализовать прикладную задачу только для заранее заданной

вычислительной архитектуры узлов и фиксированной конфигурации аппаратного ресурса вычислительной системы. При изменении конфигурации и/или архитектуры вычислительной системы разработанную прикладную программу необходимо заново преобразовывать под изменившийся ресурс, что по трудозатратам и времени корректировки зачастую сравнимо с разработкой новой программы. Одной из главных причин этого является существенное различие моделей вычислений в рассмотренных технологиях программирования. Каждая из технологий программирования процессорных узлов, графических ускорителей и ПЛИС ограничена рамками своей вычислительной архитектуры и не содержит общей формы представления вычислений и методов их отображения на другие вычислительные архитектуры. Отсутствие в рассмотренных технологиях инструментальных средств, позволяющих менять архитектуру узлов и конфигурацию вычислительной системы при портации одной и той же прикладной программы в различные архитектуры, исключает возможность их использования для гибридных вычислительных систем с различным и зачастую не определенным на этапе разработки программы числом узлов и вычислительных архитектур. Для процедурных вычислений исходной формой представления задачи является последовательная программа, отображаемая на заданную архитектуру и конфигурацию МВС распараллеливающим компилятором либо программистом. Для средств высокоуровневого синтеза конфигурационных файлов ПЛИС та же последовательная программа рассматривается уже в виде композиции вычислительно-трудоемкого фрагмента для ПЛИС, требующего выделения специальными директивами, и остальных вычислений, реализуемых на процессоре. Границы вычислительно-трудоемкого фрагмента зависят от доступного аппаратного ресурса – количества ПЛИС и узлов других типов, т.е заданы нестрого и могут изменяться. Определение границ фрагмента каждый раз требует анализа программы, ручной переразметки кода директивами и портации программы. Поэтому последовательная программа и родственные ей формы представления вычислений (алгоритмическая схема, граф передачи управления и др.) не позволяют без изменения кода отразить возможные для

разных архитектур (процессоров и ПЛИС) способы организации вычислений. Для различных вычислительных архитектур необходима единая форма, позволяющая представлять вычисления прикладной задачи с возможностью их реализации на узле любой вычислительной архитектуры гибридной вычислительной системы без изменения кода. Это доказывает первое положение, выдвигаемое для защиты.

Определим понятие архитектуры гибридной вычислительной системы и терминологию для описания ее вычислительных узлов как основных компонентов архитектуры. Архитектура вычислительной системы – это совокупность ее основных свойств, существенных для пользователя – устройств, узлов и блоков, структура управляющих и информационных связей между ними, обеспечивающая выполнение заданных функций. С учетом этого определения архитектура гибридной вычислительной системы, как и любой вычислительной системы, характеризуется структурой информационных и управляющих связей между своими компонентами – процессорами, графическими ускорителями и ПЛИС, каждый из которых обладает собственной вычислительной архитектурой и специфическим способом организации вычислений: процедурным для процессоров, параллельно-конвейерным для графических ускорителей и параллельным для ПЛИС. Общепринятого термина для обозначения вычислительной архитектуры компонента гибридной вычислительной системы нет, а наиболее подходящие по смыслу «под-архитектура» или «суб-архитектура» (по аналогии с «системой» и «подсистемой») звучат очень непривычно. Поэтому для разграничения понятий архитектуры ГВС как вычислительной системы и вычислительной архитектуры ее компонентов как принятого способа организации вычислений, для описания последней будет использоваться термин «целевая архитектура». В ряде случаев будет использоваться «вычислительная архитектура узла», имеющая тот же смысл, что и целевая архитектура.

Для целевых архитектур ГВС, существенно различающихся в методах и технологиях организации вычислений, необходимы общие принципы, новые методы и средства эффективного их программирования, объединенные в

единую технологию. Параллельные программы, обладающие возможностью изменения характеристик аппаратной реализации вычислений каждого фрагмента задачи в зависимости от параметров целевой архитектуры ГВС, будем называть независимыми от архитектуры и конфигурации или *ресурсонезависимыми* параллельными программами. Ресурсонезависимость можно интерпретировать как независимость или инвариантность параллельных вычислений относительно доступного вычислительного ресурса. Технологию программирования, предназначенную для создания и трансляции таких программ для различных целевых архитектур ГВС здесь и далее будем называть технологией *ресурсонезависимого* программирования[163]. Технология *ресурсонезависимого* программирования обеспечивает программирование различных целевых архитектур в едином контуре с синтезом общего сбалансированного решения прикладной задачи.

На основе проведенного в настоящей главе анализа известных технологий программирования каждой из целевых архитектур, можно сделать вывод, что для эффективного использования заранее неопределенного аппаратного ресурса гибридной вычислительной системы необходимы теоретические основы ресурсонезависимого программирования, включающие:

- модель вычислений в единой для всех рассматриваемых целевых архитектур форме;
- принципы организации единого вычислительного контура для различных целевых архитектур;
- методы и средства преобразования вычислений в единой форме, адаптирующие их к целевой архитектуре и конфигурации ГВС;
- средства реализации задач на множестве разнородных вычислительных узлов ГВС, обеспечивающие рациональное решение в автоматическом режиме без участия программиста.

Совокупность перечисленных новых моделей представления вычислений, принципов и методов их преобразования, а также инструментальных средств программирования разнородных вычислительных архитектур в едином вычислительном контуре, образуют технологию

программирования [18, 19] гибридных вычислительных систем. Создаваемая технология программирования гибридных вычислительных систем должна обеспечивать разрешение научного противоречия, которое заключается в том, что, с одной стороны, необходимо обеспечить единую для всех целевых архитектур форму представления вычислений, а с другой – синтезировать рациональное решение задачи, учитывающее особенности каждой целевой архитектуры ГВС. Рациональность или эффективность реализации вычислений для заданного аппаратного ресурса оценивается временем решения или заданным уровнем производительности задачи, которое должно учитываться при поиске рационального варианта.

Поэтому **первый принцип** теоретических основ разрабатываемой технологии ресурсонезависимого программирования можно сформулировать следующим образом – *вычисления прикладной задачи должны быть представлены в единой для различных целевых архитектур форме, обеспечивающей возможность реализации каждого фрагмента задачи на любой из целевых архитектур гибридной вычислительной системы*.

Представленные в единой форме вычисления должны преобразовываться и адаптироваться к любой из целевых архитектур ГВС, независимо от ее конфигурации, поэтому архитектура и конфигурация аппаратного ресурса должны являться параметром синтеза программы. При подстановке фактических характеристик целевой архитектуры и конфигурации гибридной вычислительной системы единая форма должна автоматически, с помощью формальных преобразований и без семантического разрыва, преобразовываться в рациональное для выбранной конфигурации решение. Поэтому **второй принцип** теоретических основ разрабатываемой технологии программирования можно сформулировать следующим образом: *преобразование представленных в единой форме одновременных вычислений к целевым архитектурам ГВС выполняется с учетом параметра – архитектуры и конфигурации аппаратного ресурса*.

Число узлов целевых архитектур в составе ГВС заранее неизвестно, поэтому методы преобразования и адаптации единой формы к архитектуре вычислительной системы должны обеспечивать возможность гибкого

перехода между параллельной, параллельно-конвейерной, конвейерной и процедурной формами организации вычислений в фрагментах задачи и их различных сочетаний. Для выполнения этого требования методы преобразования и адаптации для каждого фрагмента задачи должны обеспечивать возможность изменения типа реализации, частоты работы, числа одновременно выполняемых операций и разрядности обрабатываемых данных. Это позволяет сформулировать **третий принцип** – *методы преобразования вычислений к целевым архитектурам ГВС должны обеспечивать возможность изменения характеристик аппаратной реализации: способа организации вычислений, частоты, числа устройств и др. для каждого фрагмента задачи.*

Поэтому при описании вычислений задачи в единой форме должны учитываться разрядность, латентность, интервал обработки данных вычислительных устройств и характеристики аппаратного ресурса ГВС. Рациональное для выбранной архитектуры решение должно синтезироваться при подстановке этих параметров. Тогда **четвертый принцип** разрабатываемой технологии *ресурсонезависимого* программирования можно сформулировать следующим образом*: выбор наиболее рационального варианта реализации прикладной задачи или ее фрагмента на каждой из целевых архитектур ГВС должен осуществляться с учетом характеристик решения – латентности, интервала обработки данных, разрядности, числа устройств и т.д.*

В модели вычисления прикладной задачи всегда сбалансированы, но ее фрагменты могут иметь разную степень параллелизма и собственные значения основных параметров – типа реализации, частоты работы и разрядности обрабатываемых данных, числа одновременно выполняемых операций. Для каждого вычислительного фрагмента задачи это требование означает согласованное с остальными изменение параметров фрагмента, итоговой целью которого является поиск наиболее рационального варианта, обеспечивающего минимизацию времени реализации прикладной задачи с учетом ее информационной структуры. Поэтому **пятый принцип** ресурсонезависимого программирования можно сформулировать следующим

образом: *методы преобразования вычислений должны учитывать информационную структуру прикладной задачи для подзадач с разной степенью параллелизма, чтобы синтезируемое с помощью параметрически заданного аппаратного ресурса решение было согласованно между вычислительными узлами разных целевых архитектур*.

Сбалансированность получаемого решения является одним из ключевых требований к методам преобразования единой формы для адаптации к архитектуре ГВС. Формализация и разделение методов представления вычислений, преобразования и синтеза решений для целевых архитектур ГВС позволит исключить их взаимную зависимость и создать параметризуемую ресурсонезависимую форму, которая может быть адаптирована (портирована) к архитектуре и конфигурации ГВС с помощью изменения параметров без существенной переработки исходного текста параллельной программы. Параметризуемая аппаратным ресурсом единая форма должна позволять преобразовывать вычисления подзадач с различной степенью параллелизма как при увеличении доступного вычислительного ресурса ГВС, так и при его сокращении вплоть до одного узла. Поэтому **шестой принцип** разрабатываемой технологии ресурсонезависимого программирования состоит в том, что *методы преобразования вычислений должны обеспечивать сбалансированное рациональное решение как при увеличении, так и при сокращении доступного аппаратного ресурса гибридной вычислительной системы*.

Совокупность средств описания, методов и алгоритмов преобразования вычислений к архитектуре и конфигурации ГВС, построенных на основе сформулированных принципов, позволит создать технологию ресурсонезависимого программирования гибридных вычислительных систем, позволяющую не только получить рациональное решение прикладной задачи для доступного аппаратного ресурса, но и найти его за наименьшее число преобразований.

## 1.9. Выводы к главе 1

1) Достижение микроэлектронными технологиями физических пределов миниатюризации транзисторов, завершение «гонки частот» и двукратное замедление закона Мура наряду с известными проблемами фон-Неймановской процессорной архитектуры явились основными причинами кризиса высокопроизводительных вычислений, что осложнило эффективную реализацию параллельных вычислений.

2) Графические ускорители и специализированные вычислители с процессорной архитектурой как способ повышения эффективности реализации вычислений за счет адаптации вычислительной системы к структуре задачи не позволяют разрешить сложившуюся в HPC кризисную ситуацию из-за схожих с процессорами архитектурных ограничений, поэтому необходимы новые решения в области разработки вычислительных архитектур.

3) Реконфигурируемые вычислители на основе ПЛИС в наибольшей мере позволяют адаптировать архитектуру вычислительного устройства к структуре решаемой задачи, что обеспечивает 100- и 1000-кратный выигрыш в скорости решения целого ряда как слабосвязанных, так и сильносвязанных задач. Однако эффективность реализации на ПЛИС задач управления с большим числом вложенных условных операторов, рекурсивных вызовов и структур с обратной связью невысока, но для их решения очень эффективны процессоры. Поэтому ПЛИС должны использоваться совместно с другими вычислительными архитектурами для решения всего спектра прикладных задач.

4) Наиболее перспективным способом повышения эффективности реализации HPC-вычислений по оценкам ведущих исследователей является создание гибридных или конвергентных вычислительных архитектур, объединяющих различные вычислительные узлы (процессоры и/или графические ускорители и/или ПЛИС) в гибридную вычислительную систему. Подтверждением служат последние редакции списка TOP500, в котором первые позиции уже несколько лет принадлежат гибридным и гетерогенным вычислительным системам.

5) Широкое распространение гибридных вычислительных систем с узлами различных архитектур сдерживается отсутствием единой универсальной технологии их программирования в общем вычислительном контуре.

6) Современные технологии программирования процессорных узлов (MPI, OpenMP, HOPMA, DVM, PVM), графических ускорителей (CUDA, OpenACC и др.), ПЛИС (MitrionC, COLAMO) или гибридных вычислительных систем (OpenCL) обладают архитектурной спецификой уже на уровне представления вычислений и позволяют описать задачу только для заранее заданной архитектуры и фиксированной конфигурации аппаратного ресурса вычислительной системы. Упомянутые технологии программирования высокопроизводительных вычислительных систем не позволяют без участия программиста преобразовывать (портировать) вычисления прикладной задачи к другой конфигурации и/или архитектуре вычислительной системы. Основной причиной является существенное различие моделей вычислений для разных вычислительных архитектур, которое не позволяет использовать одну из перечисленных технологий программирования для разнородных узлов ГВС и исключает автоматическую портацию прикладных задач, что доказывает первое положение, выдвигаемое для защиты.

7) Для преодоления технологических ограничений в программировании различных архитектур и выполнении портации прикладных задач необходим новый подход к созданию единой формы представления вычислений, которые могут транслироваться на любые вычислительные узлы ГВС.

8) В рамках данного похода для обозначения вычислительной архитектуры компонента гибридной вычислительной системы (процессоров, графических ускорителей, ПЛИС) как принятого способа организации вычислений предложен термин «целевая архитектура». Параллельные программы, обладающие возможностью изменения характеристик аппаратной реализации вычислений каждого своего фрагмента в зависимости от параметров целевой архитектуры ГВС, предложено называть независимыми от архитектуры и конфигурации или *ресурсонезависимыми*.

Ресурсонезависимость параллельных программ подразумевает инвариантность параллельных вычислений относительно доступного вычислительного ресурса.

9) Для ресурсонезависимого программирования гибридных вычислительных систем как единой конвергентной архитектуры необходимы новые принципы представления и организации вычислений, формальные методы их преобразования и инструментальные программные средства портации прикладных задач для заранее неизвестного аппаратного ресурса целевых архитектур ГВС. Предложены основные принципы ресурсонезависимого программирования ГВС:

– вычисления должны быть представлены в единой для различных целевых архитектур форме, обеспечивающей возможность реализации каждого фрагмента задачи на любой из целевых архитектур гибридной вычислительной системы.

– представленные в единой форме одновременные вычисления преобразуются к целевым архитектурам ГВС с учетом параметра – архитектуры и конфигурации аппаратного ресурса.

– методы преобразования вычислений к целевым архитектурам ГВС должны обеспечивать возможность изменения характеристик аппаратной реализации: способа организации вычислений, частоты, числа устройств и др. для каждого фрагмента задачи.

– выбор наиболее рационального варианта реализации прикладной задачи или ее фрагмента для каждой из целевых архитектур ГВС должен осуществляться с учетом характеристик решения – латентности, интервала обработки данных, разрядности, числа устройств и т.д.;

– методы преобразования вычислений должны учитывать информационную структуру прикладной задачи, в том числе для подзадач с разной степенью параллелизма, чтобы синтезируемое для параметрически заданного аппаратного ресурса решение было согласованно между вычислительными узлами разных целевых архитектур;

– методы преобразования вычислений должны обеспечивать сбалансированное рациональное решение как при увеличении, так и при

сокращении доступного аппаратного ресурса гибридной вычислительной системы.

10) Сформулированные принципы являются фундаментом теоретических основ ресурсонезависимого программирования – новых научных понятий, методологических принципов, положений и закономерностей, позволяющих описать единую для различных архитектур модель вычислений, которую с помощью формализованных методов и алгоритмов можно преобразовать в модель реализации для гетерогенного вычислительного ресурса, содержащего вычислительные узлы на основе процессоров, графических ускорителей и ПЛИС.

# 2. ПРЕДСТАВЛЕНИЕ ВЫЧИСЛЕНИЙ ДЛЯ ГИБРИДНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

На различных этапах реализации прикладная задача представляется следующими переходящими друг в друга моделями:

- математической моделью, заданной исходными расчетными формулами;
- моделью вычислений, описывающей связи в структуре задачи между исходными данными и результатами выполнения операций, а также определяющей последовательность их выполнения;
- моделью организации и реализации вычислений для заданного аппаратного ресурса на определенной вычислительной архитектуре с набором ее инструкций и операторов.

Математическая модель прикладной задачи, как правило, является исходной формой описания вычислений. Для представления взаимосвязи операций задачи используются модели вычислений, а их воплощение в вычислительной архитектуре выполняется на основе модели организации и реализации вычислений. Согласно первому принципу теоретических основ разрабатываемой технологии программирования (см. главу 1) все вычисления прикладной задачи должны быть представлены форме, единой для различных способов организации одновременных вычислений, и позволяющей реализовать каждый фрагмент задачи на любой из целевых архитектур ГВС. Для этого модель вычислений должна обеспечивать не только возможность представления параллелизма и информационных зависимостей задачи, но и ресурсонезависимость, т.е. независимость вычислений от доступного аппаратного ресурса и способность их реализации на любой целевой архитектуре ГВС.

Под ресурсонезависимым представлением вычислений будем понимать такое описание задачи или ее фрагмента, которое позволяет выполнить все операции на вычислительной системе независимо от ее архитектуры,

имеющей хотя бы одно устройство со структурной или процедурной организацией вычислений. Основополагающее свойство ресурсонезависимого представления вычислений – это описание операций задачи без привязки к аппаратному ресурсу, т.е. в единой для всех целевых архитектур форме, которую можно адаптировать к любой вычислительной архитектуре с вычислительными устройствами для реализации математических операций задачи. Уже на уровне модели вычислений, ресурсонезависимые вычисления должны быть представлены в форме[164, 165], отображаемой в различные целевые архитектуры как при увеличении, так и при сокращении доступного вычислительного ресурса. Поэтому ресурсонезависимость следует понимать не как отсутствие ресурса, а как параметризацию единой формы описания вычислений доступным ресурсом.

## 2.1. Единая модель вычислений для гибридных вычислительных систем

Среди моделей вычислений, которые используются для описания одновременных вычислений, наибольшее распространение получило представление задачи в виде ориентированного графа. Графовая форма представления задачи с помощью множеств входных, выходных и операционных вершин, соединенных направленными дугами, показывает структуру и связи между операциями, позволяя выявить информационные зависимости данных. В процессорных архитектурах для описания передачи управления наиболее известны алгоритмическая схема или блок-схема алгоритма (см. параграф 1.4), а для параллельно-конвейерных архитектур, графических ускорителей и ПЛИС – граф алгоритма [1] и информационный граф [6, 75,102], описывающие информационные зависимости при обработке данных.

Граф алгоритма представляет вычисления как совокупность операций, распределенных по подмножествам – ярусам. В наиболее известных параллельных формах графа алгоритма, таких как ярусно-параллельная [1, 6]

(рис. 2.1) и каноническая [1] формы, все вершины разбиты на перенумерованные подмножества ярусов, а направление исходящих из вершины дуг соответствует увеличению порядковых номеров ярусов. В пределах одного яруса все вершины информационно-независимы, поэтому между ними отсутствуют дуги. Как правило, вершины графа алгоритма – это простые операции, такие как сложение, умножение, деление и т.д. Сложные составные операции (макрооперации) обычно не используются, но допускаются [1]. Граф алгоритма отражает информационные зависимости и естественный параллелизм *тонкой структуры алгоритмов* [1] и позволяет получать количественные и качественные оценки алгоритмов применением формального аппарата теории графов. Например, минимальное время выполнения алгоритма можно оценить с помощью критического пути графа и др.

Представление структуры, информационных зависимостей и связей между операциями характерно и для информационного графа задачи (см. параграф 1.7), используемого в парадигме структурно-процедурной организации вычислений. В отличие от графа алгоритма, в слоях и итерациях информационного графа задачи (ИГЗ) обычно используются более сложные объекты – подграфы, содержащие несколько информационно связанных операций, в том числе условных и коммутационных [6, 75, 166]. Поэтому для таких простых алгоритмов, как алгоритм вычисления корней квадратного уравнения [167] (рис. 2.1), информационный граф будет совпадать с графом алгоритма, а для более сложных, таких, как информационный граф прямого хода решения системы линейных алгебраических уравнений (СЛАУ) методом Гаусса [168] (рис. 2.3), они будут различаться. Распределение информационно-независимых подграфов по слоям и информационная связь между итерациями ИГЗ отражают зависимости между крупными функционально-завершенными фрагментами задачи на структурно-топологическом уровне в модели вычислений максимально параллельной формы задачи. Поэтому ИГЗ достаточно просто формируется из

последовательной программы путем развертки циклов и построения информационных зависимостей.

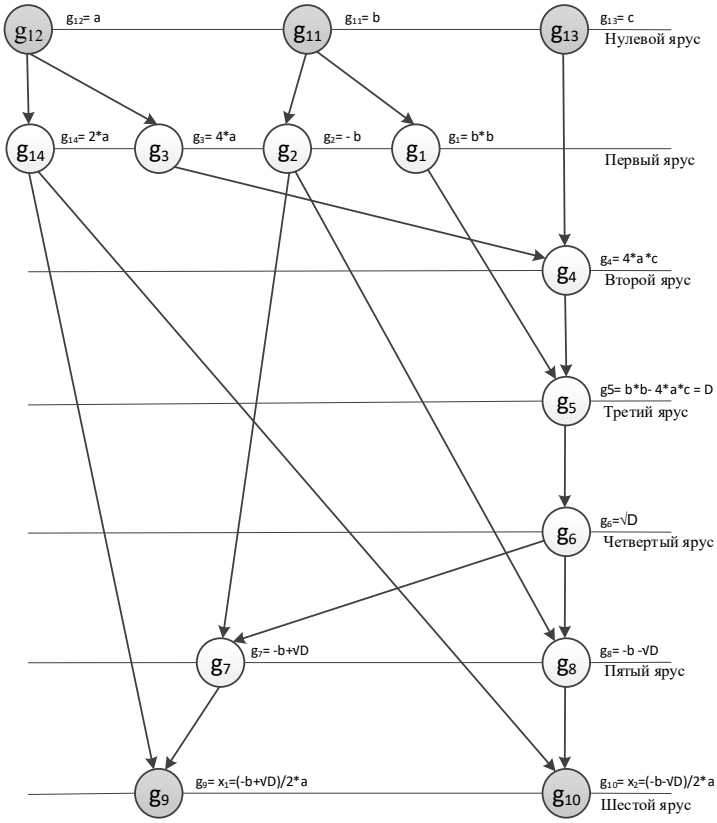



Рис. 2.1. Ярусно-параллельная форма графа алгоритма вычисления корней квадратного уравнения

На рис. 2.2 представлен фрагмент последовательной программы на языке программирования С для прямого хода СЛАУ методом Гаусса, а соответствующий ему информационный граф показан на рис. 2.3.

```
for (i = 0; i < N; i++) {
  for (j = i+1 ; j < N ; j++) {
    d = (m[j][i]/m[i][i]);
   for (k = i; k < N+1 ; k++)
      m[j][k] = m[j][k] - (m[i][k]*d);
   }
}
```

Рис. 2.2. Фрагмент программной реализации прямого хода решения СЛАУ методом Гаусса на языке С

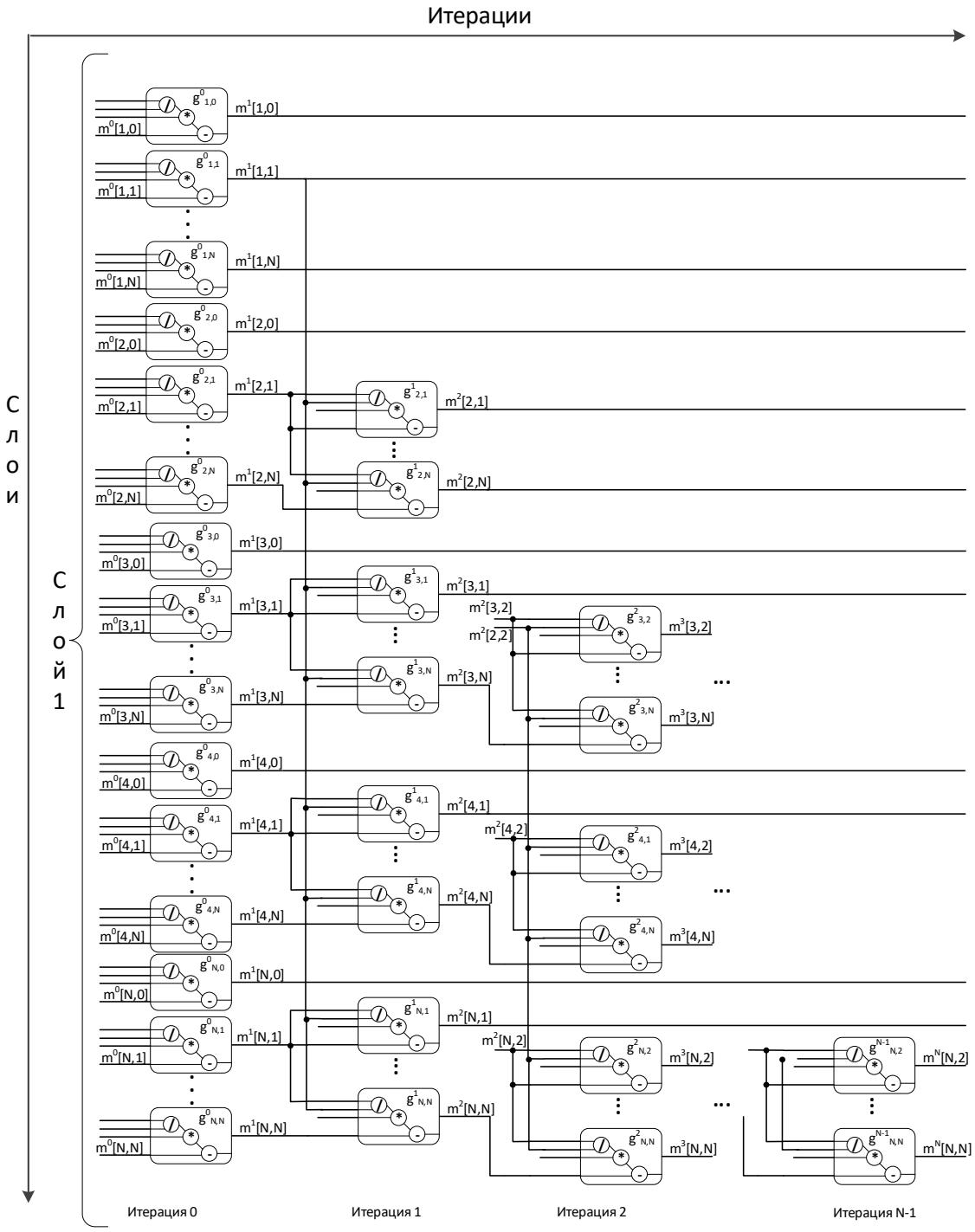



Рис. 2.3. Информационный граф прямого хода решения СЛАУ методом Гаусса

В слоях (рис. 2.3) расположены изоморфные информационно-независимые подграфы $g_{jk}^{i}$, каждый из которых содержит 3 операционные вершины: деление, умножение и вычитание. Информационная зависимость между итерациями задается связями между подграфами: выход $m^1[1,1]$ подграфа $g_{1,1}^{0}$ нулевой итерации является входом подграфов $g_{2..N,1..N}^{1}$ первой,

выход $m^2[3,2]$ подграфа $g_{3,2}^1$ является входом $g_{3..N,2..N}^2$ второй и т.д. Для решения СЛАУ методом Гаусса согласно циклам исходной программы (рис. 2.2) каждый следующий слой ИГЗ (рис. 2.3) содержит меньшее число подграфов $g_{jk}^i$.

Как уже отмечалось, для каждой целевой архитектуры ГВС характерны своя модель вычислений и базовая форма организации одновременных вычислений: последовательная - для процессоров (см. параграф 1.4), параллельно-конвейерная - для графических ускорителей (см. параграф 1.5) и параллельная - для ПЛИС (см. параграфы 1.6, 1.7). Для единой модели вычислений в гибридных вычислительных системах необходимо выбрать одну из рассмотренных форм или разработать новую, обеспечивающую ресурсонезависимое представление параллельных вычислений. Рассмотрим применимость каждой из этих моделей для представления параллельных вычислений всех целевых архитектур.

Алгоритмическая схема как модель последовательности действий применима для универсальных процессоров, но неэффективна для представления вычислений в параллельных и параллельно-конвейерных архитектурах, т.к. при ее отображении в параллельные архитектуры графических ускорителей и ПЛИС неизбежно возникает необходимость распределения вычислений по узлам, трудоемкость которого соответствует автоматическому распараллеливанию последовательных программ (см. параграф 1.4). Поэтому в качестве модели вычислений для ГВС более перспективно использовать модели для параллельных вычислений – граф алгоритма или информационный граф, т.к. они применимы не только для параллельных или параллельно-конвейерных, но и для последовательных процессорных архитектур. Для различных целевых архитектур ГВС более предпочтителен информационный граф, т.к. он описывает вычисления не только на уровне операций, но и на уровне подграфов, каждый из которых реализуем на отдельном узле ГВС любой архитектуры. Информационный граф, как и граф алгоритма, позволяет применять формальный аппарат теории

графов для расчета основных характеристик, таких как длина критического пути, высота и число ярусов и др. Информационный граф оперирует вершинами и дугами, не зависит от целевой архитектуры, и полностью соответствует первому принципу теоретических основ ресурсонезависимых вычислений, т.е. позволяет описывать задачу в единой для различных вычислительных узлов ГВС форме. Можно утверждать, что среди рассмотренных моделей вычислений ИГЗ является наиболее общей формой представления вычислений, позволяющей описывать не только выполняемые операции задачи, но и информационные связи между ними как на уровне элементарных операций, так и между подграфами задачи.

Еще одним, возможно, наиболее важным, свойством ИГЗ для теоретических основ ресурсонезависимого программирования является инвариантность (неизменность) его формы для различных способов описания одной и той же прикладной задачи. Поскольку ИГЗ – это максимально параллельная форма представления вычислений, отражающая информационные зависимости между функционально-завершенными фрагментами программы, то, с точностью до топологической перестановки подграфов в слое, ИГЗ будет неизменен для разных последовательных программ, описывающих одну и ту же прикладную задачу, при одинаковом базисе используемых вычислительных операций.

Например, при разбиении внешнего цикла по переменной ***i*** (рис. 2.2) на два цикла, получится одинаковая по семантике, но разная по форме, программа. При распараллеливании исходной и измененной, семантически одинаковых, программ автоматическим распараллеливающим компилятором будут получены разные параллельные программы. ИГЗ, построенные из исходной и измененной программ, будут одинаковыми, с точностью до топологической перестановки подграфов в слое. Поэтому ИГЗ можно считать не только единой, но и инвариантной формой представления параллельных вычислений для различных способов описания прикладной задачи. Инвариантность ИГЗ для представления параллельных вычислений позволяет

объединять топологически различные, но информационно-эквивалентные варианты описания вычислений в единой форме, что существенно сокращает число потенциальных анализируемых вариантов организации параллельных вычислений уже на уровне модели.

Поэтому информационный граф задачи может служить единой моделью для представления различных способов описания и организации одновременных вычислений, которая содержит все заданные параметрами задачи операции обработки данных, а также сохраняет информационные зависимости в ее структуре. Тогда именно информационный граф будем считать моделью вычислений для гибридных вычислительных систем, единой для всех ее целевых архитектур, что доказывает второе положение, выдвигаемое для защиты.

## 2.2. Преобразование модели вычислений в максимально параллельную кадровую структуру

Для отображения информационного графа как единой максимально параллельной формы задачи в архитектуру гибридной вычислительной системы (ГВС) необходимо разделить его на фрагменты вычислений для каждой целевой архитектуры и распределить их по доступным узлам ГВС. Критерием рационального решения задачи будет минимальное общее время, поэтому при распределении вычислений нужно учитывать как характеристики фрагмента вычислений (длину критического пути или латентность, число информационно-независимых и информационно-зависимых операций, их разрядность и др.), так и обобщенные характеристики вычислительных узлов целевых архитектур (длительность такта обработки, число одновременно выполняемых за такт операций, производительность и др.). Для этого модель организации и реализации вычислений в архитектуре гибридной вычислительной системы должна отражать эти характеристики для оценки времени выполнения как каждого фрагмента, так и всей задачи в целом.

Информационный граф, как форма описания максимально параллельных вычислений, единая для всех целевых архитектур ГВС, не содержит параметров, характеризующих реализацию операций задачи, таких как интервал обработки данных, длительность такта обработки, число одновременно выполняемых за такт операций, производительность и т.д. Поэтому описать преобразование максимально параллельных вычислений в архитектуру ГВС с помощью только графовой модели ИГЗ невозможно. Здесь требуется представление, отражающее не только информационные зависимости, но и характеристики времени выполнения вычислений. Для этого следует перейти от графовой модели со структурно-топологическим представлением вычислений вершинами и дугами к вычислительным устройствам и коммутационной структуре, которые имеют такие параметры, как быстродействие, интервал обработки данных, латентность и др.

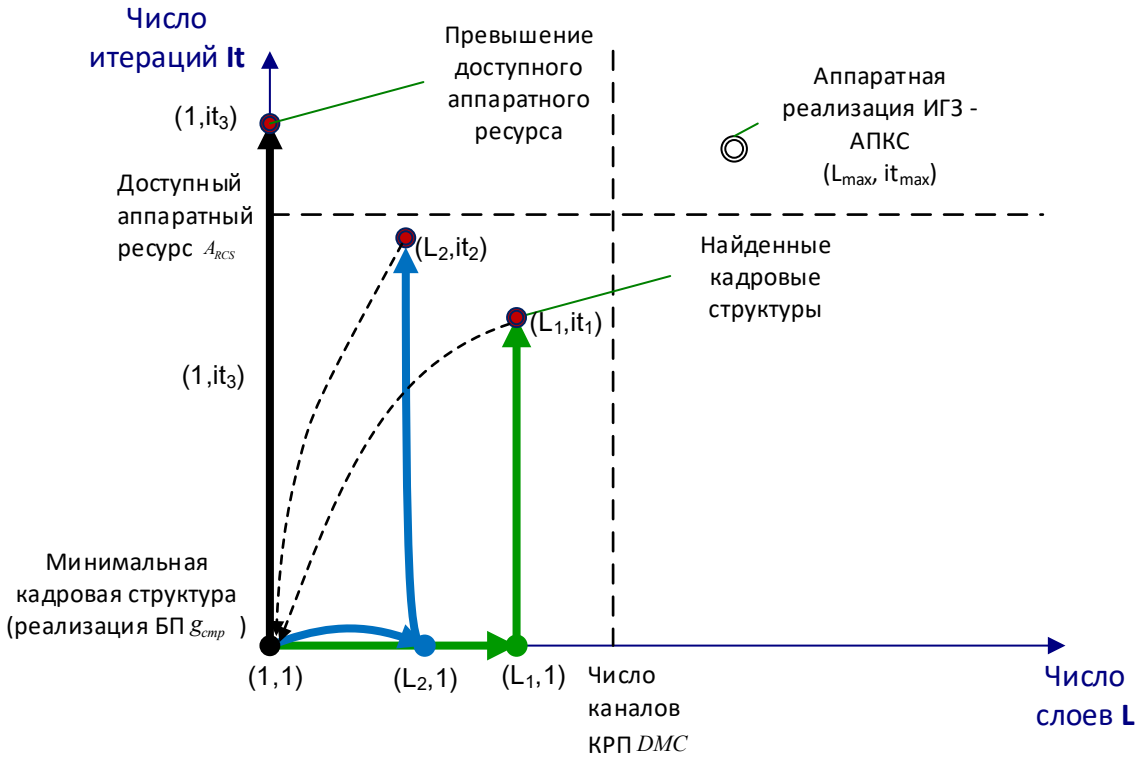
Для информационного графа такой формой представления является кадровая форма (структура) [6,75,102] (см. параграф 1.7). Кадровая структура учитывает основные параметры вычислительных устройств и характеристики решения задачи, такие как время выполнения, характеристики параллелизма фрагментов информационного графа и потоки данных. Переход от информационного графа к кадровой структуре выполняется назначением вычислительных устройств операционным вершинам графа и заменой дуг связями коммутационной системы. Полученная в результате такой замены максимально параллельная кадровая структура (МПКС), как и ИГЗ, содержит все устройства для обработки данных, заданные параметрами задачи и соединенные согласно информационным зависимостям. Максимально параллельная кадровая структура, как и информационный граф, выполняет вычисления для всех данных задачи, подаваемых одновременно на ее входы, поэтому она не содержит потоков данных в привычном понимании этого термина. Для одновременной обработки всех данных задачи в максимально параллельной кадровой структуре требуется реализация всех вычислительных устройств и коммутационной структуры, определяемых параметрами задачи,

что, в свою очередь, требует существенных аппаратных затрат. Но, в отличии от информационного графа, максимально параллельная кадровая структура позволяет оценить время решения задачи с учетом характеристик реальных вычислительных устройств. Кроме того, ее можно преобразовать в другие кадровые структуры, которые обеспечивают информационную эквивалентность результатов вычислений, но занимают меньший аппаратный ресурс.

Кадровая структура, совмещающая форму представления и модель реализации вычислений, применима для масштабирования задачи. Примером масштабирования кадровой структуры задачи является технология индуктивных программ [75], предложенная в 2003 году для структурно-процедурной организации вычислений. Масштабирование с помощью технологии индуктивных программ (ТИП) заключалось в индуктивном наращивании числа аппаратно реализованных кадровых структур, соответствующих базовому подграфу (см. параграф 1.7) информационного графа, которые далее будем называть минимальными кадровыми структурами. Минимальная кадровая структура (МинКС) считалась [75, 102] неделимой и атомарной единицей структурной организации вычислений и объектом масштабирования в технологии индуктивных программ. МинКС, как правило, разрабатывалась высококвалифицированным прикладным программистом в виде структурной конвейерной реализации $g_{стр}$ для заданных параметров прикладной задачи и доступного вычислительного ресурса. При реализации и оптимизации минимальной кадровой структуры $g_{стр}$ прикладной программист максимально сокращал число каналов распределенной памяти $DMC$ и интервал обработки данных $I$ МинКС. Поэтому индуктивное наращивание числа информационно-независимых оптимизированных МинКС позволяло легко построить эффективный мультиконвейер (см. Приложение 1). Фактически, масштабирование кадровых структур в технологии индуктивных программ представляет собой увеличение числа одновременно реализованных

слоевых подграфов ИГЗ, соответствующих $g_{стр}$, в кадровой структуре. Поток данных МинКС при масштабировании делится пропорционально числу реализованных кадровых структур, поэтому каждая из них обрабатывает меньшее число данных.

Масштабирование в технологии индуктивных программ представлено как перемещения на плоскости слоев $L$ (от англ. **L**ayer) и итераций $It$ (от англ. **It**eration) из исходной точки МинКС $g_{стр}$ с координатами (1, 1) по слоям и итерациям (рис. 2.4).




а)

Рис. 2.4. Масштабирование прикладной задачи в технологии индуктивных программ

Движение по осям слоев $L$ и итераций $It$ дискретно и с целочисленным шагом, зависящим от доступного ресурса РВС. Аппаратный ресурс РВС – это не скалярная, а интегральная, величина, задаваемая графом [75] с вершинами-объектами вычислительной системы: каналами контроллеров распределенной памяти (КРП), ячейками LUT и MLUT, регистрами и триггерами FF, блоками внутренней памяти BRAM, блоками аппаратной обработки DSP и внешними

связями кристалла. Доступный аппаратный ресурс в обобщенном представлении (рис. 2.4) изображен пунктирными линиями, параллельными осям слоев $L$ и итераций $It$. Число каналов распределенной памяти $DMC$ ограничивает количество аппаратно реализуемых информационно-независимых МинКС $g_{стр}$ по слоям, а обобщенный аппаратный ресурс объектов вычислительной системы $A_{RCS}$ ограничивает количество информационно-зависимых структур $g_{стр}$ по итерациям. Область, заданная началом координат, осями слоев $L$ и итераций $It$ и точкой пересечения пунктирных линий, характеризующих доступный аппаратный ресурс, определяет возможные для него реализации кадровых структур на основе МинКС. Каждая точка этой области с целочисленными координатами соответствует кадровой структуре с собственными значениями эффективности решения задачи и занимаемого аппаратного ресурса.

Максимально параллельная кадровая структура с координатами $(L_{\max}, It_{\max})$ как аппаратная реализация информационного графа является предельным вариантом масштабирования. В 2003 году, когда технология индуктивных программ была впервые предложена [75], возможность аппаратной реализации информационного графа для реальных, а не модельных, задач казалась фантастической, т.к. вычислительный ресурс реальных ПЛИС и РВС на их основе был невелик и всегда считался конечным и ограниченным. В начале 2000-х годов на ПЛИС удавалось реализовать, как правило, не более одного базового подграфа (МинКС) и вспомогательных структур прикладной задачи, поэтому масштабирование рассматривалось исключительно для слоевых информационно-независимых МинКС с построением мультиконвейера (см. Приложение 1). Эти варианты показаны (рис. 2.4) сплошными линиями из (1, 1) в точки $(L_1, 1)$ и $(L_2, 1)$ вдоль оси слоев $L$. Ограничением этой модели является число каналов распределенной памяти: наращивание числа $g_{стр}$ в МинКС по слоям возможно до исчерпания свободных каналов памяти $DMC$.

С ростом доступного ресурса в новых семействах ПЛИС и появлением возможностей для практической реализации идей технологии индуктивных программ [75], тиражирование МинКС в слое быстро достигло предела по числу каналов КРП, практически неизменному с начала 2000-х годов. Дальнейший рост ресурса ПЛИС привел к тому, что каналы КРП для построения мультиконвейера быстро заканчивались, а аппаратного ресурса ПЛИС было достаточно для реализации дополнительных МинКС. Поэтому для повышения эффективности использования аппаратного ресурса и удельной производительности после достижения предельного числа $g_{стр}$ по слоям, согласно технологии индуктивных программ масштабирование выполнялось по итерациям путем индуктивного наращивания информационно-зависимых МинКС $g_{стр}$. Такой метод масштабирования получил название «распараллеливание по итерациям», а соответствующий ему график показан (рис. 2.4) сплошными линиями вдоль оси итераций: из $(L_1, 1)$ в $(L_1, it_1)$ и из $(L_2, 1)$ в $(L_2, it_2)$. Конечно, возможность масштабирования по итерациям зависит не только от наличия свободного аппаратного ресурса, но и от наличия итераций в структуре информационного графа прикладной задачи. Например, для многих задач символьной обработки распараллеливание по итерациям неприменимо из-за отсутствия информационной зависимости между конвейерами $g_{стр}$. В то же время распараллеливание по итерациям успешно используется для решения задач линейной алгебры с итерационной структурой, таких как решение СЛАУ методом Гаусса, Якоби, Гаусса-Зейделя и др. [169]. При распараллеливании по итерациям задач линейной алгебры удавалось выигрывать 2-3 десятичных порядка по времени решения задачи по сравнению с реализацией на процессоре даже при 10-кратно меньших рабочих частотах РВС, составлявших 160-250 МГц против 3 ГГц [169].

Масштабирование в технологии индуктивных программ похоже на распараллеливание вычислений в традиционных МВС, когда фрагмент

вычислений, соответствующий МинКС, тиражируется (распараллеливается) по слоям и итерациям. В отличие от методов автоматического распараллеливания, для технологии индуктивных программ формально определен критерий эффективности масштабируемого решения, такой как общее время решения задачи. Поскольку рациональный минимум времени решения задачи – это основной критерий выбора рациональной кадровой структуры, то наилучшим вариантом будет минимальный интервал обработки данных **I=1**, как для МинКС, так и для масштабированного решения задачи в целом.

Распараллеливание по итерациям конвейерных ступеней из МинКС $g_{стр}$ имеет свои особенности (см. Приложение 1). Если аппаратного ресурса при масштабировании по итерациям недостаточно для реализации всех итерационных ступеней из МинКС (переход вдоль оси итераций из $(1,1)$ в $(1, it_3)$ на рис. 2.4), то при реализации меньшего числа МинКС в масштабированном решении интервал обработки данных и время решения задачи увеличиваются пропорционально общему числу итерационных ступеней (см. Приложение 1). Изначально технология индуктивных программ была предложена для задач символьной обработки, у которых при наращивании числа информационно-независимых $g_{стр}$ интервал обработки данных в результирующей кадровой структуре не менялся. Поэтому при построении мультиконвейера, т.е. при движении по слоям вдоль оси $L$, время решения задачи линейно сокращалось. Но при решении задач с итерационной структурой, т.е. при движении по итерациям вдоль оси $It$, время решения задачи возрастало, если не удавалось реализовать все итерационные ступени из МинКС. Распараллеливание по итерациям выявило существенный недостаток технологии индуктивных программ: **при масштабировании МинКС не учитывался интервал обработки данных**. Сокращение интервала обработки данных средствами технологии индуктивных программ было невозможно, поэтому для получения рационального решения задачи были разработаны сокращающие интервал обработки данных оптимизации,

такие как построение конвейера в конвейере для $g_{стр}$ или макроконвейера для процедурной реализации МинКС $g_{proc}$ (см. Приложение 1).

Еще один недостаток технологии индуктивных программ состоит в том, что эффективность получаемого результата зависит от выбора МинКС $g_{стр}$ и ее реализации прикладным программистом. Неформализованность критериев выделения базового подграфа для синтеза МинКС $g_{стр}$ позволяет для одной и той же задачи выделять разные базовые подграфы и синтезировать несколько МинКС с различными траекториями масштабирования (рис. 2.4). Время решения задачи зависит не только от числа реализованных в слоях и итерациях МинКС $g_{стр}$, но и от интервала обработки данных как в $g_{стр}$, так и в полученном при масштабировании решении.

Не умаляя заслуг А.В. Каляева и И.И. Левина как создателей теории структурно-процедурной организации вычислений и методов масштабирования технологии индуктивных программ, следует отметить основные ее недостатки для автоматического масштабирования задачи:

- неформализованность критериев выделения МинКС и необходимость ее оптимальной схемотехнической реализации;

- отсутствие учета интервала обработки данных, необходимое при масштабировании по итерациям;

- невозможность масштабирования задач, МинКС $g_{стр}$ которых превышает доступный аппаратный ресурс $A_{RCS}$.

Концепция софт-архитектур [170, 171, 172], предложенная как развитие технологии индуктивных программ, также не смогла устранить все отмеченные недостатки. Возможной причиной этого является отсутствие формального разделения информационного графа как модели и кадровой структуры на основе аппаратно реализуемого базового подграфа как модели реализации вычислений, что вносит неоднозначность в используемые понятия и затрудняет формализацию перехода от модели вычислений к их реализации.

Поэтому в разрабатываемой технологии ресурсонезависимого программирования [173, 174] для целевых архитектур гибридной вычислительной системы необходимо учесть и, если возможно, устранить недостатки, свойственные технологии индуктивных программ.

Таким образом, для максимально параллельной кадровой структуры можно отметить следующее:

- МПКС сравнительно просто синтезируется из ИГЗ заменой дуг связями коммутационной системы, а операционных вершин графа – вычислительными устройствами;

- МПКС, как и ИГЗ, содержит все заданные параметрами задачи устройства для обработки данных, соединенные согласно информационным зависимостям и не содержит потоков данных в привычном понимании этого термина (все данные задачи обрабатываются одновременно, т.е. поток данных единичной длины);

- в отличии от ИГЗ, МПКС позволяет оценить время решения задачи с учетом характеристик реальных вычислительных устройств;

- МПКС можно преобразовать в другие кадровые структуры, обеспечивающие информационную эквивалентность результатов вычислений, но более экономичные по занимаемому аппаратному ресурсу;

- для поиска рационального решения при масштабировании МПКС необходимо учитывать не только доступный аппаратный ресурс вычислительной системы, но и ее параметры параллелизма: информационно-независимые подграфы (слои), информационно-зависимые подграфы (итерации), интервал обработки данных и время решения задачи.

Преобразование максимально параллельной кадровой структуры к доступному аппаратному ресурсу вычислительной системы можно назвать секвентированием или «распоследованием», т.к. она преобразуется в информационно-эквивалентную кадровую структуру с меньшим параллелизмом. В отличии от технологии индуктивных программ, методы, разрабатываемые для технологии ресурсонезависимого программирования,

преобразуют максимально параллельную, а не минимальную, кадровую структуру к целевым архитектурам гибридной вычислительной системы с учетом доступного аппаратного ресурса, являющегося параметром преобразования.

## 2.3. Ресурсонезависимое представление максимально параллельной кадровой структуры для гибридных вычислительных систем путем параметризации доступным вычислительным ресурсом

Во всех рассмотренных в главе 1 технологиях программирования высокопроизводительных вычислительных систем реализация прикладной задачи **T** выполнялась преобразованием ее модели к командам вычислительной архитектуры системы **A** с заранее заданным и неизменяемым аппаратным ресурсом **R**

$$CS = \mathrm{Map}(T, A, R) \tag{2.1}$$

где $CS$ – реализация задачи, а $\mathrm{Map}(T, A, R)$ – функциональное преобразование, отображающее модель задачи **T** на заданную вычислительную архитектуру **A** и определенный аппаратный ресурс **R** вычислительной системы.

При изменении архитектуры **A** и/или конфигурации аппаратного ресурса **R** вычислительной системы преобразование $\mathrm{Map}(T, A, R)$ выполняется для новых значений **R** и **A**, т.е. ни **R,** ни **A** не являются параметрами $\mathrm{Map}(T, A, R)$. В (2.1) **R** и **A** являются, скорее, «граничными условиями», при изменении которых преобразование необходимо выполнить заново. Обычно преобразование $\mathrm{Map}(T, A, R)$ достаточно трудоемко, его сложно формализовать и автоматизировать. Кроме того, оно требует больших временных затрат, сравнимых с временем реализации задачи. Для всех рассмотренных технологий программирования универсальных процессоров, графических ускорителей и ПЛИС это преобразование выполняется, как правило, разработчиком, поэтому масштабирование и портация созданного решения прикладной задачи на другую конфигурацию затруднительны.

Для целевых архитектур гибридной вычислительной системы преобразование модели задачи (2.1) с учетом нескольких вычислительных архитектур принимает вид:

$$CS = \mathrm{Map}(T;\ A_1, A_2, \ldots A_N; R_1, R_2, \ldots R_N). \qquad (2.2)$$

Увеличение числа архитектур и конфигураций при адаптации прикладной задачи к архитектуре ГВС, помимо усложнения самого преобразования $\mathrm{Map}(T, A, R)$, увеличивает и число анализируемых вариантов для поиска рационального решения, т.к. для каждого фрагмента задачи требуется не только анализ эффективности его реализации на каждой архитектуре, но и оценка влияния на возможные варианты реализаций остальных фрагментов задачи. Поэтому число анализируемых вариантов преобразования прикладной задачи в архитектуру и конфигурацию ГВС (2.2) зависит как от числа функционально-обособленных фрагментов прикладной задачи, так и от конфигурации гибридной вычислительной системы. С увеличением числа вычислительных архитектур и различных конфигураций вычислительных узлов количество вариантов отображения существенно увеличивается и становится сопоставимо с числом вариантов, анализируемых распараллеливающим компилятором.

Поэтому, в соответствии со вторым принципом ресурсонезависимого программирования (см. главу 1), архитектура **A** и конфигурация **R** гибридной вычислительной системы должны быть параметром преобразования (2.2), подстановка значения которого синтезирует решение задачи для заданного вычислительного ресурса. Для этого необходимо проанализировать и формализовать методы преобразования кадровых структур и разработать методы адаптации максимально параллельной кадровой структуры к различным вычислительным архитектурам ГВС. Параметризация преобразования модели в архитектуру и конфигурацию гибридной вычислительной системы (2.2) позволит решить проблему портации и масштабирования созданного решения прикладной задачи. Формализация методов преобразования и автоматизация расчета характеристик модели

организации и реализации вычислений позволит существенно сократить время масштабирования и портации прикладной задачи на новые архитектуры и конфигурации гибридных вычислительных систем.

Прикладную задачу, параметризуемую ресурсом, можно представить в виде

$$CS(R) = \text{Map}(T(R = R^*), A^*, R = R^*), \tag{2.3}$$

где $CS(R)$ – определяемая доступным ресурсом ГВС реализация задачи, а $T(R^*)$ – результат преобразования ресурсонезависимого представления задачи под аппаратный ресурс **R**, $R^*$ - заданный объединенный ресурс (конфигурация) гибридной вычислительной системы, $A^*$ - архитектура ГВС как единой вычислительной системы. Рассмотрим способы параметризации кадровой структуры доступным аппаратным ресурсом.

Максимально параллельная кадровая структура как неразрывная совокупность вычислительной структуры, аппаратно-реализуемой на доступном ресурсе, и правил организации входных и выходных потоков данных синтезируется из функционально-регулярной формы [75] информационного графа (см. параграф 1.7):

$$G = \Phi_i(F_{ij}(g_{jk}^i)), \tag{2.4}$$

где $F_{ij}$ – функция упорядочивания информационно-независимых подграфов в вычислительном слое $S_i, i = 1 \ldots N$ задачи $T$, а $\Phi_i$ – функция отображения информационно-зависимых вычислительных слоев $S_1, S_2, \ldots, S_N$ для минимальных неделимых фрагментов вычислений, т.е. изоморфных подграфов $g_{jk}^i$, реализуемых в виде МинКС. Здесь и далее, если специально не оговорено, будем считать все задачи функционально-регулярными. Если задача имеет нерегулярную структуру, то ее можно привести в функционально-регулярную форму с помощью методов дополнения [75] и векторизации [75], разработанных в теории структурно-процедурной организации вычислений.

Как было показано в предыдущем параграфе при рассмотрении методов масштабирования технологии индуктивных программ, реализация функций отображения $\Phi_i$ и $F_{ij}$ в кадровой структуре зависит от доступного аппаратного ресурса вычислительной системы. Поэтому для кадровой структуры функции $\Phi_i$ и $F_{ij}$ в (2.4) можно представить как композицию $\overset{A_{RVS}}{\circ}$ параллельной и последовательной (процедурной) составляющих, реализация которых определяется аппаратным ресурсом $A_{RVS}$ вычислительной системы:

$$\Phi_i(A_{RVS}) = \Phi_{пар} \overset{A_{RVS}}{\circ} \Phi_{пос}, F_{ij}(A_{RVS}) = F_{пар} \overset{A_{RVS}}{\circ} F_{пос}. \qquad (2.5)$$

Тогда кадровую структуру задачи $T$ из (2.4) можно представить как

$$T(A_{RVS}) = \Phi_{пар} \overset{A_{RVS}}{\circ} \Phi_{пос}(F_{пар} \overset{A_{RVS}}{\circ} F_{пос}(g_{ijk})), \qquad (2.6)$$

где $\overset{A_{RVS}}{\circ}$ – композиция функций упорядочивания $\Phi_i$ и $F_{ij}$, зависящая от аппаратного ресурса $A_{RVS}$ вычислительной системы;

$\Phi_{пар}$ – параллельная, т.е. реализованная структурно в пространстве, кадровая структура из информационно-зависимых МинКС $g_{ijk}$ в виде ступеней конвейера, а в предельном случае – структурная реализация всех итераций максимально параллельной кадровой структуры;

$\Phi_{пос}$ – последовательная, т.е. реализованная процедурно во времени (с обратной связью), кадровая структура из информационно-зависимых МинКС $g_{ijk}$, а в предельном случае – обработка всех итераций одной МинКС;

$F_{пар}$ – параллельная, т.е. реализованная структурно (в пространстве), кадровая структура из информационно-независимых МинКС $g_{ijk}$, а в предельном случае – структурная реализация всех подграфов слоя в виде мультиконвейера (см. Приложение 1);

$F_{пос}$ – последовательная, т.е. реализованная процедурно (во времени), кадровая структура из информационно-независимых МинКС $g_{ijk}$, а в

предельном случае – обработка входных данных всех слоевых подграфов одной МинКС с кратным увеличением потока обрабатываемых данных.

Для функционально-регулярного информационного графа максимально параллельная кадровая структура, являющаяся исходной точкой адаптации единой формы задачи $T$ к архитектуре и конфигурации ГВС, будет определяться как

$$T_{МПКС} = \Phi_{пар}(F_{пар}(g_{стр})). \tag{2.7}$$

Аппаратная реализация МинКС как необходимое условие [75, 102] структурно-процедурной организации вычислений, представима в виде

$$T_{МинКС}(A_{RVS_{min}}) = \Phi_{пос}(F_{пос}(g_{ijk})) \ . \tag{2.8}$$

Если условие (2.8) не выполнялось, единственным способом решения всей задачи была процедурная (последовательная) организация вычислений на процессоре. С развитием теории структурно-процедурной организации вычислений была доказана [175] возможность дискретного перехода между двумя способами реализации МинКС $g_{ijk}$, которая, в зависимости от доступного аппаратного ресурса $A_{RVS}$, реализуема либо структурно как $g_{стр}$, либо процедурно как $g_{проц}$. Более того, в [175] показано, что для ряда задач, содержащих вложенные условные операторы, рекуррентные структуры и информационно-незначащие пересылки данных, можно сочетать структурные $g_{стр}$ и процедурные $g_{проц}$ фрагменты вычислений в едином вычислительном контуре. Поэтому кадровая структура задачи $T$ (2.6) представима как функция, зависящая от доступного аппаратного ресурса, для всех компонент в виде

$$T(A_{RVS}) = \Phi_{пар} \overset{A_{RVS}}{\circ} \Phi_{пос}(F_{пар} \overset{A_{RVS}}{\circ} F_{пос}(g_{стр} \overset{A_{RVS}}{\circ} g_{проц})). \tag{2.9}$$

В отличие от форм описания кадровых структур, известных и принятых в теории структурно-процедурной организации вычислений, представление кадровой структуры (2.9), параметризуемое аппаратным ресурсом, позволяет:

- явно параметризовать кадровую структуру значением доступного аппаратного ресурса, при подстановке которого она может меняться в диапазоне от минимальной до максимально параллельной;

- описывать задачи, МинКС $g_{стр}$ которых превышает доступный аппаратный ресурс $A_{RCS}$;

- получать результат преобразования (2.9) при подстановке значения аппаратного ресурса, без промежуточных возвратов к МинКС, как в технологии индуктивных программ.

Представление кадровой структуры задачи в форме (2.9) обеспечивает выполнение второго принципа теоретических основ ресурсонезависимого программирования, согласно которому для поиска рационального варианта масштабирования вычислений в ГВС анализируется не информационный граф, а соответствующая ему максимально параллельная кадровая структура, параметризуемая доступным аппаратным ресурсом $A_{RCS}$ и учитывающая характеристики параллелизма фрагментов ИГЗ и потоки данных при расчете времени решения задачи. Кроме этого, форма (2.9) позволяет реализовать и третий принцип теоретических основ: при подстановке аппаратного ресурса, являющегося параметром метода преобразования максимально параллельной кадровой структуры к архитектуре и конфигурации ГВС, единая форма должна эффективно отображаться и преобразовываться к любой из целевых архитектур.

С учетом того, что для реализации задачи кадровая структура может содержать несколько кадров $C_1, C_2, \ldots, C_N$, а ресурс является составным по числу используемых вычислительных архитектур, кадровая структура всей задачи для ГВС представляется в виде

$$CS(R) = \text{Map}(C_1(R = R^*), C_2(R = R^*), \ldots, C_N(R = R^*),$$
$$R^* = R_1 \cup R_2 \ldots \cup R_M, A^*). \quad (2.10)$$

Здесь каждую кадровую структуру с учетом (2.9) можно представить как функциональную зависимость от ресурса вида

$$C_i(R) = U\left(\Phi_i(R), F_{ij}(R), g_{ijk}(R)\right). \quad (2.11)$$

Оценим число параметров представлений (2.9-2.11), зависящих от доступного аппаратного ресурса и влияющих на преобразование кадровой

структуры к целевым архитектурам ГВС. К таким параметрам относятся определяемые аппаратным ресурсом композиции параллельной и последовательной составляющих функций $\Phi_i/F_{ij}$ и способ реализации МинКС $g_{ijk}$.

Таким образом, при преобразовании кадровой структуры для каждой целевой архитектуры ГВС нужно проанализировать и подобрать значения трех параметров: параметры композиции параллельной и последовательной составляющих функций $\Phi_i/F_{ij}$ и способ реализации МинКС $g_{ijk}$. Для трех рассматриваемых целевых архитектур гибридной вычислительной системы, общее число анализируемых значений не превышает 9, что существенно меньше числа вариантов, анализируемых распараллеливающим компилятором. Если в структуре задачи несколько кадровых структур, то для каждой из них нужно проанализировать не более 9 вариантов. Поскольку число кадровых структур в реальных задачах обычно не превышает 7-10, то число анализируемых вариантов остается сравнительно небольшим даже в этом случае, что позволит существенно сократить время преобразования и портации прикладных задач.

Согласно сформулированным принципам, результатом преобразования максимально параллельной кадровой структуры должна стать информационно-эквивалентная кадровая структура с меньшей степенью параллелизма, рационально реализуемая на доступном аппаратном ресурсе целевых архитектур ГВС. Получить рациональную кадровую структуру можно двумя способами – синтезировать оптимальную кадровую структуру для всех возможных целевых архитектур ГВС или синтезировать кадровую структуру для заданного вычислительного ресурса с последующей ее оптимизацией для текущей конфигурации ГВС. В первом случае методы синтеза существенно усложняются из-за необходимости учета особенностей конкретных целевых архитектур на каждом этапе преобразования. Второй вариант более предпочтителен для реализации принципов ресурсонезависимого программирования, т.к. он на логическом уровне

отделяет методы преобразования кадровых структур от методов их оптимизации, что позволит дополнять технологию как при появлении новых критериев синтеза кадровых структур, так и при добавлении новых целевых архитектур в составе ГВС. Поэтому, по аналогии с многопроходной компиляцией, задачу синтеза рационального варианта при преобразовании максимально параллельной кадровой структуры целесообразно решать в два этапа, т.е. сначала получить множество решений в области доступного аппаратного ресурса ГВС, а затем найти наиболее рациональный вариант решения в заданной области. Рассмотрим первый этап синтеза – поиск множества допустимых решений задачи.

## 2.4. Преобразование кадровой структуры к доступному вычислительному ресурсу гибридной вычислительной системы

Доступный аппаратный ресурс вычислительной системы в представлении кадровой структуры (2.9) определяет не только число слоевых и итерационных подграфов в функциях $\Phi_i$ и $F_{ij}$, но и способ реализации МинКС $g_{ijk}$. Поэтому двух измерений (слоев $\boldsymbol{L}$ и итераций $\boldsymbol{It}$), использовавшихся в технологии индуктивных программ (рис. 2.1), будет недостаточно для описания преобразования кадровой структуры в архитектуру вычислительной системы. Кроме этого, для гибридной вычислительной системы преобразование кадровой структуры должно учитывать особенности не только ПЛИС, но и других целевых архитектур: универсальных процессоров с исключительно последовательно выполняемыми инструкциями и графических ускорителей с параллельно-конвейерными устройствами сокращенной разрядности. Для учета особенностей всех целевых архитектур гибридной вычислительной системы при изменении способа реализации МинКС $g_{ijk}$ в (2.9) представление задачи (рис. 2.1) на основе слоев и итераций нужно дополнить еще одним измерением – числом команд минимальной кадровой структуры (ось «Команды С»,

рис. 2.2). Преобразование максимально параллельной кадровой структуры к архитектуре гибридной вычислительной системы в этом случае происходит не на плоскости (рис. 2.4), как в ТИП, а в трехмерном пространстве, которое будем называть пространством возможных реализаций кадровых структур (рис. 2.5).

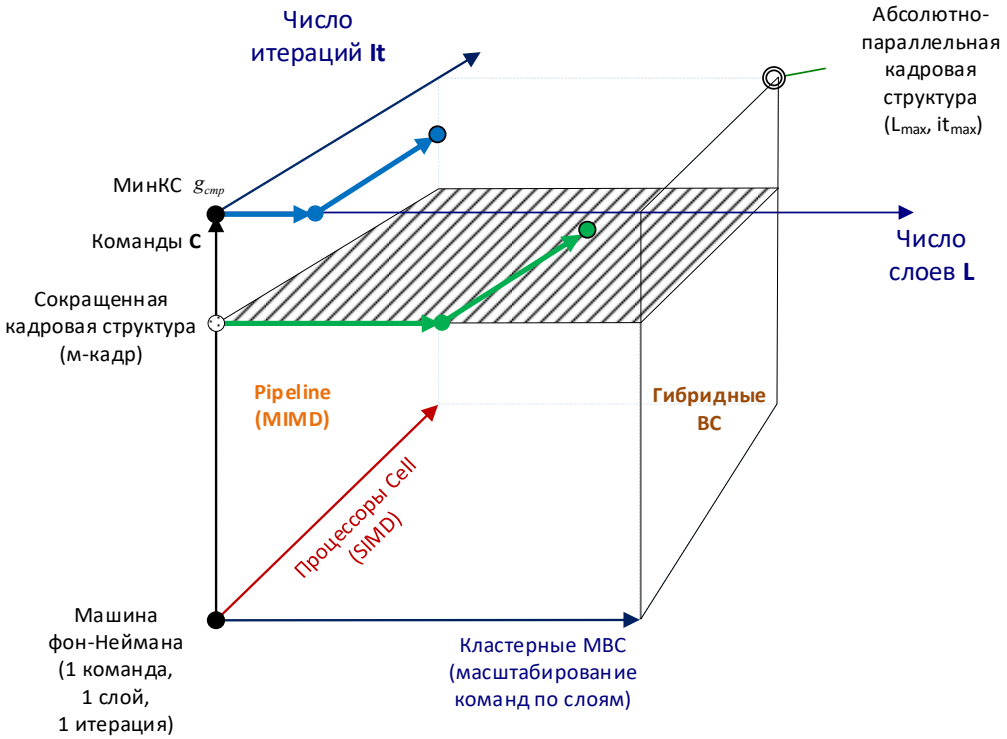



Рис. 2.5. Пространство реализации вычислений для целевых архитектур ГВС

Пространство возможных реализаций (рис. 2.5) в единой для всех целевых архитектур системе координат отражает основные характеристики кадровой структуры, влияющие на аппаратные затраты, производительность и реализацию вычислений в ГВС - последовательную процессорную реализацию, параллельно-конвейерную для GPU и максимально параллельную кадровую структуру для ПЛИС. Учет ресурса для единой формы представления вычислений в (2.9) и на рис. 2.5 соответствует второму принципу ресурсонезависимого программирования для всех целевых архитектур ГВС: универсальных процессоров, графических ускорителей и ПЛИС. Кроме этого, в пространстве реализации вычислений можно

представлять кадровые структуры, занимающие промежуточные между структурной $g_{стр}$ и процедурной $g_{проц}$ реализациями формы. Такие кадровые структуры с сокращенным числом устройств или разрядностью, характеризуются меньшим по сравнению с МинКС аппаратным ресурсом и находящейся в диапазоне от структурной $g_{стр}$ до процедурной реализации $g_{проц}$ производительностью.

Пространство возможных реализаций кадровой структуры позволяет представить организацию вычислений для большинства распространенных существующих архитектур и вычислительных систем. Так, процессорной архитектуре фон-Неймана, выполняющей одну команду слоевого или итерационного подграфа в один момент времени, соответствует минимальная с точки зрения параллелизма реализация вычислений с координатами (1 команда, 1 слой, 1 итерация) (рис. 2.5). Объединение универсальных процессоров в кластерную МВС приводит к увеличению числа команд, одновременно выполняемых разными процессорами, и может быть представлено движением от архитектуры фон-Неймана из точки (1, 1, 1) вдоль оси «Число слоев L», что соответствует увеличению числа информационно-независимых вычислительных устройств или числа слоев в терминологии структурно-процедурной организации вычислений. Для конвейерных (pipeline) архитектур и графических ускорителей, характерно увеличение числа одновременно выполняемых информационно-зависимых команд или итераций. Такие архитектуры могут быть представлены движением от архитектуры фон-Неймана из точки (1, 1, 1) вдоль оси «Число итераций It». Движение только по оси «Число итераций It» означает конвейеризацию вычислений в одном устройстве, как в процессоре Cell, а движение в плоскости «Число итераций It» – «Команды C», т.е. одновременное с конвейеризацией увеличение числа исполняемых информационно-независимых команд, характерно для конвейерных вычислительных архитектур, pipeline-систем с архитектурой MIMD и машин потока данных. В гибридных архитектурах и графических ускорителях одновременно могут

работать несколько параллельных (информационно-независимых) вычислителей (например, нити и варпы), поэтому начальной точкой является не фон-Неймановская машина (1, 1, 1), а (1, K, 1), где K – число информационно-независимых конвейеров архитектуры. Движение происходит в плоскости «Число итераций It» – «Команды C» для заданного архитектурой значения числа информационно-независимых конвейеров (слоев) K.

Минимальная кадровая структура $g_{стр}$ (рис. 2.5) расположена на плоскости слоев и итераций с координатами (g, 1, 1), где g характеризует число вычислительных устройств структурной реализации базового подграфа. Нетрудно заметить, что минимальная кадровая структура $g_{стр}$ (рис. 2.5) изначально имеет более высокую производительность по сравнению с кластерными МВС и графическими ускорителями за счет структурной реализации всех команд (операций) базового подграфа. В отличие от технологии индуктивных программ (рис. 2.4), представление пространства реализации вычислений для целевых архитектур ГВС (рис. 2.5) позволяет описывать и задавать плоскости, параллельные плоскости слоев и итераций ***L***-МинКС $g_{стр}$-***It***, использовавшейся для структурно-процедурной организации вычислений. Такие плоскости соответствуют сокращенной кадровой структуре, занимающей меньший ресурс, чем минимальная, что обеспечивает выполнение четвертого принципа ресурсонезависимых вычислений по изменению способа организации вычислений, частоты, числа устройств и других характеристик аппаратной реализации на любой из целевых архитектур ГВС. Кадровую структуру с сокращенным числом устройств и/или разрядностью, представляющую собой композицию $g_{стр} \overset{A_{RVS}}{\circ} g_{проц}$, будем называть микро-кадром или м-кадром. Принципы, методы синтеза и свойства м-кадров подробно рассмотрены в пункте 3.1.2 и Приложении 1.

Представление пространства реализации (рис. 2.5) позволяет описывать вычисления для всех целевых архитектур гибридной вычислительной

системы, но явным образом не учитывает время решения задачи, которое задано третьим принципом ресурсонезависимого программирования. Поскольку при поиске эффективной и рациональной кадровой структуры нужно учитывать разрядность, число операций, интервал обработки данных и общее время решения задачи, систему координат (рис. 2.5) следует дополнить измерением, характеризующим временную составляющую реализации кадровой структуры. Интервал обработки данных – это основная характеристика, практически линейно определяющая время решения задачи, поэтому именно его следует выбрать в качестве дополнительного измерения для пространства возможных реализаций кадровых структур (рис. 2.6).

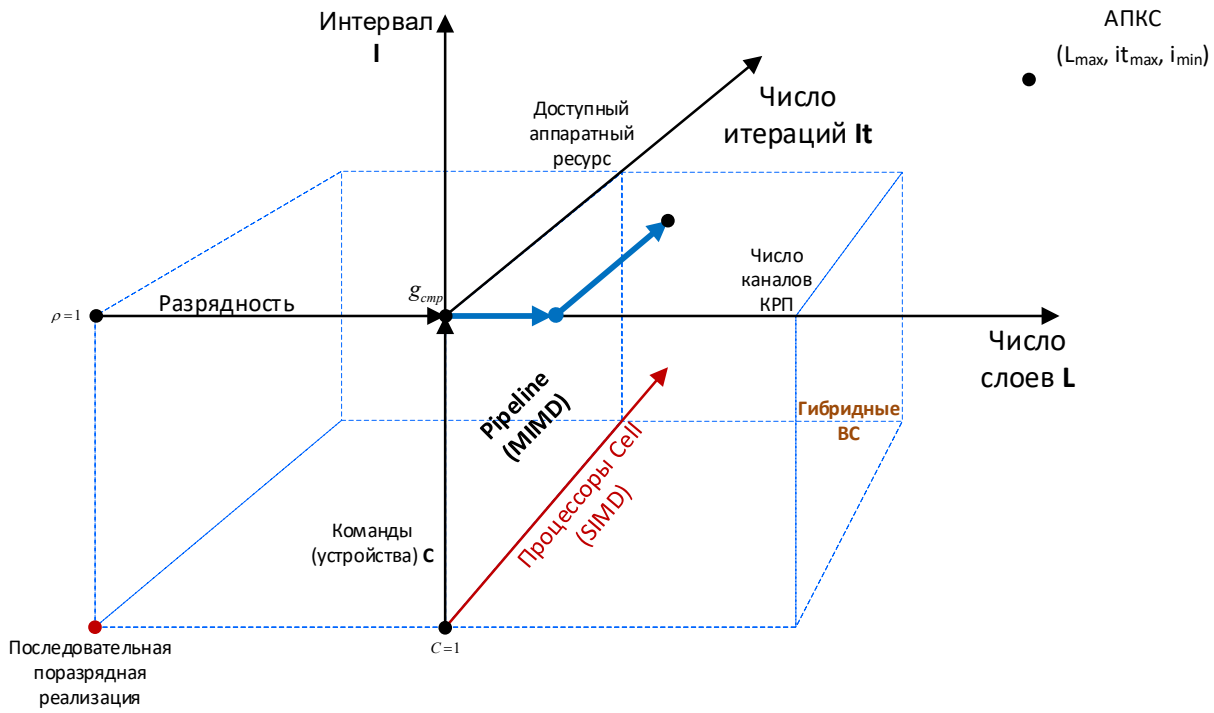



Рис. 2.6. Пространство возможных реализаций кадровых структур для целевых архитектур гибридной вычислительной системы

Ось команд «Команды С» (рис.2.5) на рис. 2.6 разделена на две ортогональные оси: ось «Команды (устройства) С» и ось разрядности. Разделение осей необходимо для представления возможных реализаций МинКС, находящихся между граничными значениями $g_{стр}$ и $g_{проц}$, в целевых архитектурах графических ускорителей и ПЛИС, способных изменять не

только число команд, но и разрядность обрабатываемых данных. Добавление интервала обработки данных в пространство возможных реализаций кадровых структур позволяет при преобразовании кадровой структуры оценить время решения задачи, которое является главным критерием эффективности реализации на заданном аппаратном ресурсе. С помощью оси интервала обработки данных «Интервал I» можно представлять увеличение интервала для преобразований кадровой структуры. Так, масштабирование итераций будет представляться пространственной ломаной (линейным сплайном), обозначенной пунктирной линией на рис. 2.7, отражающей увеличение интервала и времени решения и соответствующей действительности, а не прямой (сплошная линия), как в технологии индуктивных программ.

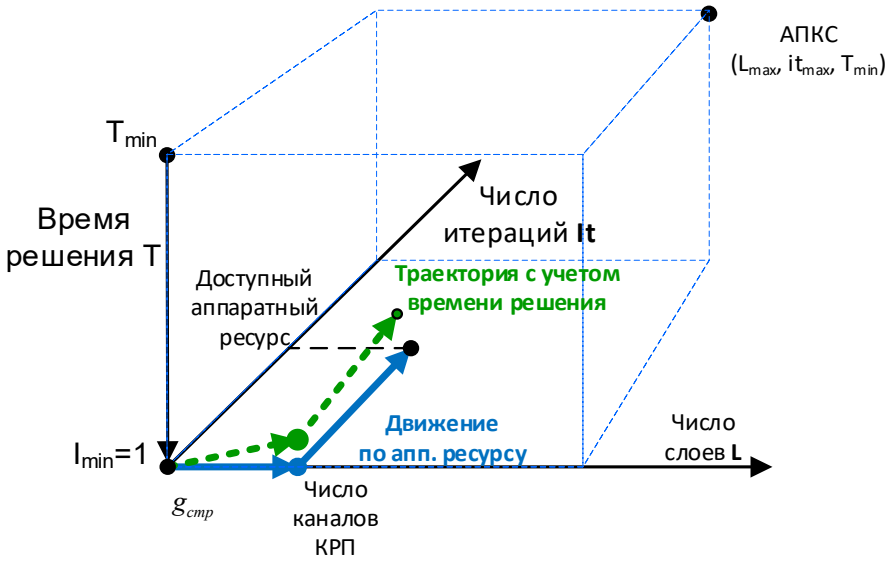



Рис. 2.7. Преобразование кадровой структуры в технологии индуктивных программ и в пространстве реализаций

Оси времени и интервала (рис. 2.7) совмещены таким образом, что максимально параллельная кадровая структура имеет минимальный интервал и минимальное время решения, которое увеличивается при движении от МПКС к МинКС $g_{стр}$ без изменения интервала обработки данных.

Рассмотрим преобразование кадровой структуры в пространстве реализаций (рис. 2.8) с учетом интервала обработки данных и времени

решения задачи для технологии индуктивных программ (точечный пунктир) и разрабатываемого метода (штрих-пунктир).

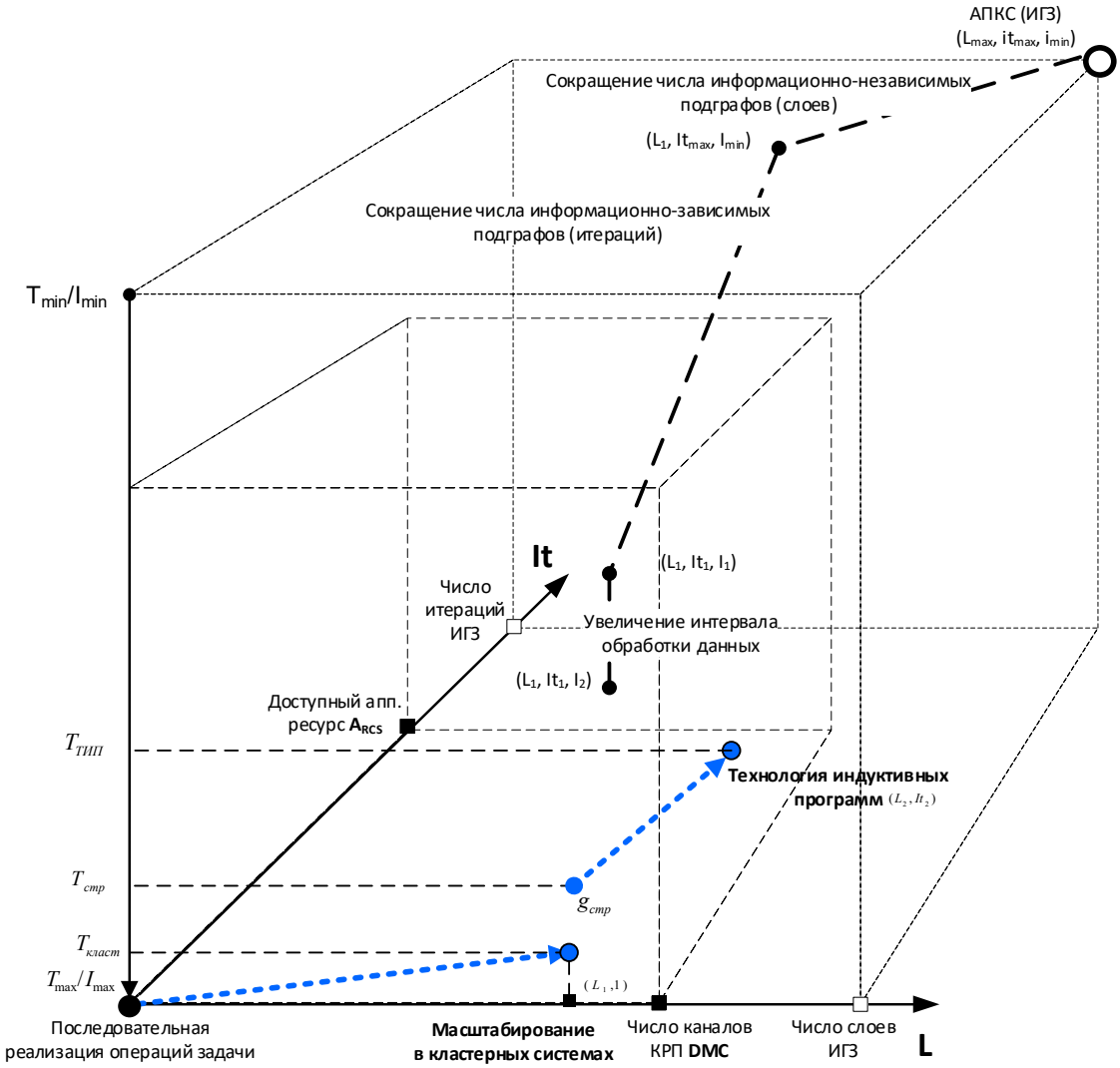



Рис. 2.8. Преобразование кадровых структур в пространстве возможных реализаций

Здесь ось времени решения $T$ совмещена с осью интервала обработки данных $I$ таким образом, что минимальное время решения $T_{min}$ соответствует максимально параллельной кадровой структуре с минимальным интервалом $I_{min}$. При движении вдоль совмещенной оси $T_{min}/I_{min}$ в направлении последовательной реализации операций задачи увеличивается и время решения, и интервал обработки данных до значений $T_{max}/I_{max}$. Внешнему кубу с контуром из мелкой пунктирной линии (рис. 2.5) соответствует аппаратный ресурс, требуемый для реализации максимально параллельной

кадровой структуры задачи. Внутренний куб с контуром из крупного пунктира соответствует доступному ресурсу вычислительной системы. Преобразование кадровой структуры в пространстве возможных реализаций под доступный вычислительный ресурс гибридной вычислительной системы заключается в переходе из точки МПКС с координатами $(L_{max}, It_{max}, T_{min})$ в область доступного вычислительного ресурса путем сокращения параметров параллелизма максимально параллельной кадровой структуры таким образом, чтобы обеспечить рациональную и эффективную для заданного ресурса организацию вычислений, минимизирующую время решения задачи.

Параметризуемое доступным аппаратным ресурсом преобразование максимально параллельной кадровой структуры к целевым архитектурам ГВС существенно отличается от методов технологии индуктивных программ, также зависящим от доступного ресурса. Во время движения происходит сокращение степени параллелизма, производительности и аппаратных затрат максимально параллельной кадровой структуры до информационно-эквивалентной структуры, которая находится в области доступного ресурса, ограниченной по слоям числом каналов КРП $DMC$, а по итерациям – доступным аппаратным ресурсом $A_{RCS}$.

Исходной точкой преобразования является максимально параллельная кадровая структура с координатами $(L_{max}, It_{max}, T_{min})$, а не минимальная $g_{стр}$. Начальным направлением движения выбирается сокращение числа каналов как типичного критического ресурса для большинства задач различных предметных областей. Сокращение числа каналов выполняется путем сокращения числа информационно-независимых МинКС (слоевых подграфов информационного графа) и завершается в точке с координатами $(L_1, It_{max}, \mathrm{I}_{min})$, соответствующей количеству каналов для доступного аппаратного ресурса. Критерием завершения движения по оси «L» является либо достижение числа доступных каналов КРП в точке $L_1$, либо сокращение числа слоев до минимального значения $L_1 = 1$. Положение точки

$(L_1, It_{max}, \mathrm{I}_{min})$ зависит от соотношения между числом каналов в $g_{стр}$ и числом каналов в доступном аппаратном ресурсе. Сокращение числа слоевых подграфов, как правило, не увеличивает интервал обработки данных, но увеличивает время решения по сравнению с МПКС. Траектория преобразования кадровой структуры при сокращении слоевых подграфов (рис. 2.5) отражает увеличение именно времени решения задачи, а не интервала.

После достижения кадровой структурой доступного числа каналов в точке $(L_1, It_{\max}, I_{\min})$, движение продолжается путем сокращения числа информационно-зависимых подграфов (итераций) в направлении области доступного ресурса. Движение по оси «It» завершается в точке $(L_1, It_1, I_1)$, которая соответствует аппаратному ресурсу для размещения $L_1 \cdot It_1$ итерационных ступеней МинКС $g_{стр}$. Значение $It_1$ соответствует доступному ресурсу либо предельному значению $It_1 = 1$. После того, как область доступного аппаратного ресурса достигнута, можно выполнять поиск наиболее рациональной кадровой структуры как структуры с минимальным интервалом обработки данных. Для этого сокращается интервал обработки данных движением по оси «Интервал I» с помощью оптимизационных преобразований (см. Приложение 1).

После перехода от единой формы представления задачи в виде информационного графа в максимально параллельную кадровую структуру, последнюю можно преобразовать к одной или нескольким целевым архитектурам гибридной вычислительной системы. Для достижения области доступного аппаратного ресурса ГВС максимально параллельная кадровая структура путем сокращения параметров преобразуется в информационно-эквивалентную структуру с меньшей степенью параллелизма, который понимается в самом широком смысле: параллелизм рассматривается не только для слоевых и итерационных подграфов, но и для числа одновременно работающих устройств и разрядности обрабатываемых данных. При движении

число слоев и/или итераций сокращается до ограничений доступного аппаратного ресурса, а не увеличивается, как в технологии индуктивных программ. При сокращении слоев и итераций в пространстве возможных реализаций кадровых структур учитывается увеличение интервала обработки данных, поэтому движение происходит в пространстве, а не на плоскости и без возвратов в исходную точку. Рассмотрим правила сокращения параметров параллелизма кадровой структуры при движении в пространстве возможных реализаций.

## 2.5. Правила преобразования кадровых структур к архитектуре и конфигурации гибридной вычислительной системы в пространстве возможных реализаций

При переходе от максимально параллельной кадровой структуры к области доступного аппаратного ресурса (рис. 2.5) можно использовать сокращение числа информационно-независимых (слоевых) и информационно-зависимых (итерационных) подграфов, числа устройств в аппаратной реализации, разрядности выполняемых операций, частоты и интервала обработки данных. Эти преобразования кадровой структуры известны и используются прикладными программистами в структурно-процедурной организации вычислений для решения задач с заданными параметрами под заранее определенный вычислительный ресурс. Каждое из этих преобразований в том или ином виде используется на этапе проектирования, но обычно они рассматриваются не как комплекс методов, а как отдельные инструменты для достижения частной цели – сокращения параллелизма и аппаратных затрат максимально параллельной кадровой структуры для реализации на заданном аппаратном ресурсе.

При преобразовании максимально параллельной кадровой структуры снижается уровень ее параллелизма, т.е. максимально параллельная реализация вычислений секвентируется, т.е. становится более

последовательной и переходит в параллельно-конвейерную. Входные данные информационно-независимых слоевых подграфов информационного графа упорядочиваются в кортеж, который задает порядок их обработки кадровой структурой. Предельным случаем преобразуемой кадровой структуры является МинКС, структурно реализующая базовый подграф информационного графа для конвейерной обработки кортежа входных данных. Как отмечалось в параграфе 2.3, структурная $g_{стр}$ и процедурная $g_{проц}$ реализации МинКС являются граничными, но не единственно возможными формами организации вычислений, между которыми существует целое семейство сокращенных (редуцированных) структур – м-кадров, требующих меньший по сравнению с $g_{стр}$ аппаратный ресурс. Переход от минимальной кадровой структуры $g_{стр}$ к м-кадрам выполняется путем сокращения параллелизма и разрядности, которое отображается по осям «Разрядность», «Команды (устройства) С» и «Интервал I/ Время Т» (рис. 2.9). Рассмотрим преобразование кадровой структуры задачи при сокращении аппаратных затрат МинКС $g_{стр}$, число слоев и итераций которой сокращено до единицы. Как отмечалось, для этого сокращаются ее параметры параллелизма, что представимо движением из МинКС $g_{стр}$ в направлении последовательной поразрядной реализации (рис. 2.9). Так же, как и при сокращении слоев и итераций в верхнем октанте, при сокращении параллелизма минимальной кадровой структуры $g_{стр}$ в нижнем октанте возможны несколько различных направлений движения: сокращение разрядности, сокращение числа устройств (команд) и одновременное сокращение разрядности и числа устройств. Возможные траектории движения обозначены сплошными линиями (рис. 2.9). Выбор наиболее рациональной из них определяется соотношением между абсолютными значениями параметров кадровой структуры и коэффициента сокращения аппаратных затрат.

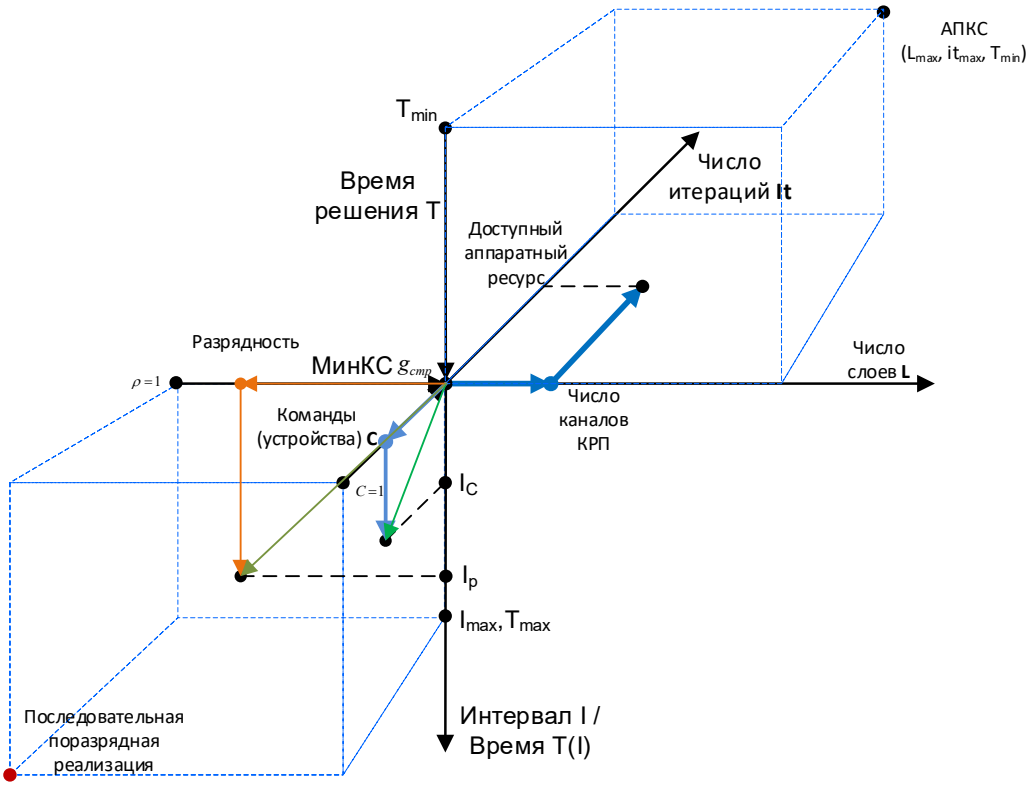



Рис. 2.9. Пространство возможных реализаций кадровых структур

Начальное направление движения определяется ресурсом, наиболее превышающим доступный аппаратный ресурс ГВС, который далее будем называть критическим ресурсом. Каждая из сокращаемых характеристик кадровой структуры по-своему влияет на аппаратный ресурс, поэтому одновременно с понижением параллелизма следует максимально сокращать и аппаратные затраты. Это означает, что среди всех возможных направлений, уменьшающих параллелизм на определенную величину, следует выбрать ту, которая обеспечивает наибольшее сокращение аппаратных затрат, максимально приближая текущую кадровую структуру к области доступного аппаратного ресурса.

Специфика движения в нижнем октанте состоит в неизбежном увеличении интервала обработки данных, поэтому при выборе любого преобразования (сокращении разрядности и/или числа устройств) перемещение происходит на плоскости, образованной осью «Интервал» и осью «Разрядность» и/или «Команды (устройства)». Для нижнего октанта пространства возможных реализаций, как и для верхнего, критерием

завершения движения служит достижение области доступного аппаратного ресурса, после которого полученную кадровую структуру можно оптимизировать для согласования и сокращения интервала обработки данных (см. Приложение 1).

В области доступного аппаратного ресурса (рис.2.9) как в верхнем, так и в нижнем октанте [176] существуют кадровые структуры с одинаковым временем решения и интервалом, находящиеся в плоскости, перпендикулярной оси «Интервал/Время», но с разными параметрами по числу слоев, итераций, разрядности и числу команд. Из этого можно сделать вывод о том, что для одной и той же кадровой структуры, в зависимости от параметров и очередности выполнения преобразований, можно получить различные результаты ее преобразования с одинаковой производительностью, но с разными параметрами по числу слоев, итераций, разрядности и числу команд. Возможно, в ряде случаев эффективные решения задачи можно получить при правильном выборе порядка применения преобразований, а в других случаях потребуются оптимизационные преобразования, увеличивающие число шагов для поиска рациональной кадровой структуры. Выбор параметров и последовательности применения преобразований, сокращающих аппаратный ресурс, нужно учесть при разработке методов преобразования кадровой структуры в архитектуру и конфигурацию ГВС.

В соответствии с поставленной целью работы – минимизацией времени портации, число преобразований кадровой структуры также требуется минимизировать, поэтому нужно сформулировать правила движения в пространстве возможных реализаций кадровых структур для минимизации числа перемещений:

- исходной точкой является максимально параллельная кадровая структура;

- для достижения области доступного аппаратного ресурса движение в верхнем октанте выполняется в направлении минимальной кадровой структуры сначала по оси «Число слоев», затем по оси «Число итераций»;

- если минимальная кадровая структура не принадлежит области доступного аппаратного ресурса, то происходит переход в нижний октант, движение в котором выполняется в направлении последовательной поразрядной реализации сначала по оси «Команды (устройства)», затем по оси «Разрядность»;

- критерием завершения движения является достижение кадровой структурой области доступного аппаратного ресурса.

Как отмечалось ранее, преобразование кадровой структуры выполняется в пяти направлениях: сокращением числа слоевых (слоев) и итерационных подграфов (итераций), разрядности, числа устройств и интервала обработки данных. Траектория преобразования максимально параллельной кадровой структуры к целевым архитектурам ГВС представляет собой непрерывную кусочно-линейную функцию (линейный сплайн), формируемую в результате сокращения ее параметров. Правила движения в пространстве возможных реализаций необходимы для быстрого достижения области доступного аппаратного ресурса с минимальным числом шагов и вершин сплайна. Рациональная кадровая структура – это точка в пространстве возможных кадровых структур, принадлежащая области доступного аппаратного ресурса.

В области доступного аппаратного ресурса (рис. 2.8) можно выделить границы минимального и максимального доступного ресурса, которые являются границами масштабирования. Область максимального доступного ресурса вычислительной системы ограничивает параллелизм кадровой структуры по числу каналов и аппаратному ресурсу сверху, а область минимального доступного ресурса позволяет завершить ее преобразование на уровне минимальной логической единицы гибридной вычислительной системы (например, одного процессора или кристалла ПЛИС), ниже которой сокращение аппаратных затрат непродуктивно. Введение понятий минимального и максимального доступного аппаратного ресурса позволяет уменьшить число выполняемых для кадровой структуры преобразований и определить критерий их завершения. Так, при сокращении числа каналов

максимально параллельной кадровой структуры с помощью числа слоев, необходимо достичь области максимального доступного ресурса по числу каналов уже на первом шаге при движении вдоль оси «Число слоев», а затем выбрать другое направление. При сокращении аппаратных затрат кадровой структуры меньше БП $g_{стр}$, область минимального доступного аппаратного ресурса определяет геометрическое место точек, за пределами которого проводить сокращение нецелесообразно, т.к. производительность полученной структуры будет ниже, а занимаемый в вычислительной системе ресурс не изменится – например, один кристалл ПЛИС, но с разными коэффициентами заполнения.

Области минимального и максимального ресурса могут располагаться друг относительно друга по-разному для различных задач, т.к. зависят не только от доступного ресурса РВС, но и от абсолютных значений характеристик минимальной кадровой структуры $g_{стр}$. Поэтому для одной и той же РВС области минимального и максимального ресурса могут располагаться как в верхнем, так и в нижнем октантах в зависимости от вида задачи и выбранного базового подграфа.

Для завершения преобразования кадровой структуры задачи и прекращения движения, по аналогии с теорией стратегического планирования [177], необходимо определить локальную и глобальную цели. Локальная цель – это цель подчиненного характера, достижение которой позволяет реализовать глобальную цель как представляемое или желаемое будущее состояние предмета. Локальной целью преобразования кадровой структуры при движении в пространстве возможных реализаций является достижение области доступного аппаратного ресурса по всем направлениям пространства, а глобальной – поиск в области доступного аппаратного ресурса рациональной или эффективной по критерию минимума времени решения задачи кадровой структуры. Для достижения глобальной цели необязательно искать самую оптимальную кадровую структуру, достаточно прийти к рациональному для

целевых архитектур гибридной вычислительной системы варианту организации параллельных вычислений.

Тогда сформулированные правила движения в пространстве возможных реализаций кадровых структур можно дополнить следующими:

- при движении в пространстве сначала достигается локальная цель, а потом – глобальная;

- сначала проверяем максимальный, а потом минимальный доступный ресурс.

На основе рассмотренных принципов движения в пространстве реализаций кадровых структур определим критерии выбора параметров и направления движения, шага используемых преобразований, положения и координат точек, в которых меняется направление движения, а также методов повышения быстродействия найденной кадровой структуры, которые необходимы для разработки метода и алгоритма преобразования прикладных задач в архитектуру и конфигурацию ГВС.

## 2.6. Изменение направления движения в точках выбора альтернатив по значению критического ресурса кадровой структуры

При описании принципов и правил движения в пространстве возможных реализаций аппаратный ресурс $A_{HCS}$ рассматривался как обобщенная характеристика области доступного аппаратного ресурса ГВС. Для гибридной вычислительной системы, объединяющей в себе процессоры, графические ускорители и ПЛИС, он представим в виде композиции ресурсов целевых архитектур

$$A_{HCS} = (A_{RCS}, A_{GPU}, A_{PROC}), \tag{2.12}$$

где $A_{RCS}$ – аппаратный ресурс ПЛИС реконфигурируемой части ГВС,

$A_{GPU}$ - аппаратный ресурс графических ускорителей ГВС,

$A_{PROC}$ - аппаратный ресурс процессоров ГВС.

Каждую из целевых архитектур, составляющих общий аппаратный ресурс ГВС $A_{HCS}$, в свою очередь, можно представить в виде композиции

своих составных частей. Рассмотрим аппаратный ресурс ПЛИС $A_{RCS}$ реконфигурируемой части ГВС как наиболее сложного компонента. Аппаратный ресурс $A_{RCS}$ объединяет в себе все элементы архитектуры ПЛИС, используемые для реализации задач (ячейки LUT и MLUT, регистры и триггеры FF, блоки внутренней памяти BRAM, блоки аппаратной обработки DSP, внешние связи кристалла), которые представимы в виде

$$A_{RCS} = \left(A_{RCS}^{LUT}, A_{RCS}^{MLUT}, A_{RCS}^{FF}, A_{RCS}^{DSP}, A_{RCS}^{BRAM}, A_{RCS}^{Ch}\right). \quad (2.13)$$

Определим *критический ресурс* $A_{cr}$ кадровой структуры как компонент реализации текущей кадровой структуры, максимально превышающий значение компонента доступного аппаратного ресурса целевой архитектуры ГВС. Для аппаратного ресурса ПЛИС $A_{RCS}$ критический ресурс определяется как

$$A_{Cr} = Max\left(\frac{A_T^{LUT}}{A_{RCS}^{LUT}}, \frac{A_T^{MLUT}}{A_{RCS}^{MLUT}}, \frac{A_T^{FF}}{A_{RCS}^{FF}}, \frac{A_T^{DSP}}{A_{RCS}^{DSP}}, \frac{A_T^{BRAM}}{A_{RCS}^{BRAM}}, \frac{A_T^{Ch}}{A_{RCS}^{Ch}}\right) > 1, \quad (2.14)$$

где $A_T^{LUT}$ характеризует число ячеек LUT, а $A_T^{MLUT}$ – число ячеек MLUT текущей кадровой структуры, $A_T^{FF}$ – число регистров и триггеров FF, $A_T^{BRAM}$ – число блоков внутренней памяти BRAM, $A_T^{DSP}$ – число блоков аппаратной обработки DSP, $A_T^{Ch}$ - число внешних связей кристалла ПЛИС, соответственно. Критический ресурс можно интерпретировать как элемент архитектуры ПЛИС, который быстрее всего расходуется при масштабировании задачи с помощью технологии индуктивных программ.

Для достижения кадровой структурой области доступного аппаратного ресурса параметры ее реализации (критический ресурс) не должны превышать значения соответствующих элементов архитектуры из (2.13). Для преобразования кадровой структуры в первую очередь необходимо сократить именно критический ресурс $A_{cr}$ как наиболее превышающий заданные архитектурой и конфигурацией аппаратные границы ГВС. Поэтому можно считать, что критический ресурс задает направления движения в пространстве возможных реализаций кадровых структур. Для достижения области доступного аппаратного ресурса кадровая структура по всем компонентам

архитектуры должна быть реализуема на текущей конфигурации ГВС, что означает выполнение условия $A_{Cr} \leq 1$ для каждого компонента архитектуры в (2.13). Последовательное приведение критического ресурса по каждому компоненту архитектуры к области доступного аппаратного ресурса позволяет получить кадровую структуру, реализуемую в гибридной вычислительной системе.

При преобразовании максимально параллельной кадровой структуры в архитектуру и конфигурацию ГВС, для кадровой структуры определяется текущий критический ресурс $A_{Cr} > 1$, выбирается направление движения (сокращаемый параметр), механизм движения (тип преобразования) и рассчитывается коэффициент сокращения аппаратных затрат. При выполнении преобразования кадровой структуры происходит перемещение в новую точку пространства возможных реализаций кадровых структур. В этой точке процесс повторяется: для текущей кадровой структуры определяется критический ресурс $A_{Cr} > 1$, направление и механизм движения и рассчитывается коэффициент сокращения аппаратных затрат. Критерием завершения движения является выполнение условия $A_{Cr} \leq 1$ в очередной точке. Вершины сплайна, в которых вычисляется критический ресурс и меняется направление движения в пространстве возможных реализаций кадровых структур, будем называть *точками выбора альтернатив.* В точке выбора альтернатив критический ресурс может качественно измениться или остаться прежним. Это зависит от абсолютных значений параметров кадровой структуры, доступного аппаратного ресурса РВС, выбранного преобразования и коэффициента сокращения аппаратных затрат. При движении между двумя точками выбора альтернатив критический ресурс $A_{cr}$ не меняется. Благодаря определению критического ресурса и точек выбора альтернатив, можно гибко расширять предложенную метрику и пересчитывать число анализируемых вариантов при появлении новых архитектурных компонентов как в архитектуре ПЛИС, так и в других архитектурах ГВС и при (возможном) добавлении новых вычислительных архитектур в структуру ГВС.

Для аппаратного ресурса ПЛИС число точек выбора альтернатив ограничено сверху количеством элементов архитектуры ПЛИС, составляющих аппаратный ресурс РВС $A_{RCS}$. Поэтому максимальное число шагов при движении в пространстве возможных реализаций кадровых структур для достижения кадровой структурой области доступного аппаратного ресурса для ПЛИС ГВС, согласно (2.13), не превышает 6. Если на очередном шаге критический ресурс сократится количественно, но не изменится качественно и останется больше единицы, то изменится вид преобразования и направление движения, но общее число шагов масштабирования не превысит число компонентов архитектуры ПЛИС. При преобразовании кадровой структуры в ряде случаев одновременно с сокращением текущего критического ресурса могут сократиться и другие компоненты архитектуры ПЛИС из (2.13), необходимые для ее реализации, поэтому число точек выбора альтернатив может быть меньше количества компонентов архитектуры.

Как правило, для задач большинства областей наиболее распространенный критический ресурс – это каналы памяти. Поэтому начальное направление движения при преобразовании максимально параллельной кадровой структуры в архитектуру и конфигурацию ГВС совпадает с осью «Число слоев L», т.е. число каналов памяти сокращается путем сокращения числа информационно-независимых слоевых подграфов. Одновременно с каналами памяти сокращаются и остальные компоненты ПЛИС, необходимые для реализации слоевых подграфов кадровой структуры, поэтому при достижении первой точки выбора альтернатив критический ресурс пересчитывается. Для полученного значения выбирается новое направление движения к области доступного аппаратного ресурса с учетом проблемной области, структуры задачи и типа организации вычислений. Для ряда задач символьной обработки критический ресурс – это ячейки LUT и MLUT, для других задач этой же области – блоки BRAM. Для задач линейной

алгебры критический ресурс – это ячейки LUT и блоки DSP, а для задач цифровой обработки сигналов – блоки DSP и блоки памяти BRAM.

На каждом шаге минимизации критического ресурса $A_{cr}$ кадровой структуры будет достигнута точка выбора альтернатив, в которой аппаратный ресурс, занимаемый текущей кадровой структурой, приблизится к области доступного аппаратного ресурса. При движении в верхнем октанте в области слоев и итераций структурная реализация минимальной кадровой структуры, соответствующая началу координат (рис. 2.6), будет достигнута не более, чем за два шага масштабирования. Если доступного аппаратного ресурса недостаточно для реализации минимальной кадровой структуры, достичь последовательной поразрядной обработки в нижнем октанте как теоретического минимума аппаратных затрат и времени выполнения можно также за два шага. Тогда число точек выбора альтернатив не превысит числа различных компонентов архитектуры ПЛИС. Учитывая, что в ряде случаев одновременно с сокращением критического ресурса будут сокращаться и другие компоненты архитектуры ПЛИС, необходимые для реализации прикладной задачи, число точек выбора альтернатив может уменьшиться, что сократит и общее число шагов.

Следует отметить, что для сокращения критического ресурса можно использовать и технологические приемы – например, использовать элементы, выполняющие одинаковые операции задачи, но реализованные с помощью разных архитектурных компонентов ПЛИС. Так, схемотехническая реализация сумматоров возможна как на логических ячейках LUT/MLUT, так и на основе блоков DSP. Сбалансированное использование обоих компонентов целевой архитектуры ПЛИС позволяет эффективнее использовать аппаратный ресурс для реализации задачи. При этом для кадровой структуры учитывается количество этих элементов каждого типа в используемом кристалле ПЛИС.

После сокращения критического ресурса по всем архитектурным компонентам ПЛИС и достижения области доступного аппаратного ресурса РВС выполняется поиск наиболее рационального и эффективного варианта

полученной кадровой структуры путем оптимизации интервала обработки данных, т.е. движение в вертикальной плоскости вдоль оси «Время/Интервал» до рационального значения с помощью оптимизирующих преобразований (минимизация интервала обработки данных, см. Приложение 1):

- конвейера в конвейере [178], уменьшающего интервал для мультиконвейерных кадровых структур с обратными связями;
- макроконвейера [179], уменьшающего интервал для процедурных кадровых структур за счет использования дополнительного аппаратного ресурса;
- автоподстановки [180], сокращающей интервал для кадровых структур с обратными связями за счет использования дополнительного аппаратного ресурса.

Для кадровой структуры выполняется только одна из оптимизаций, а не их совокупность, потому что условия их применения различны и ортогональны друг к другу. Минимизацию интервала обработки данных с помощью методов оптимизации при движении в пространстве возможных кадровых структур тоже можно считать точкой выбора альтернатив, в которой выбирается наиболее подходящее оптимизационное преобразование. Как правило, переход к более оптимальной кадровой структуре выполняется за один шаг. Общее число шагов масштабирования для синтеза рациональной кадровой структуры определяется суммой числа шагов для достижения доступного ресурса РВС и числа шагов оптимизационных преобразований. Для рассматриваемых различных компонентов архитектуры ПЛИС, число которых не превышает 7, общее число шагов масштабирования, с учетом оптимизации интервала обработки данных, очевидно, не превысит 8. Полученная оценка числа шагов масштабирования задачи в пространстве возможных кадровых структур справедлива в случаях, когда в каждой точке выбора альтернатив были правильно определены направление движения, шаг и метод преобразования, которые привели к сокращению критического ресурса кадровой структуры.

Зачастую один и тот же компонент архитектуры ПЛИС можно сократить одновременно несколькими преобразованиями. Например, число ячеек LUT и MLUT можно уменьшить как сокращением числа слоевых и/или итерационных подграфов, так и сокращением разрядности и/или числа устройств.

Критический ресурс определяет необходимость и коэффициент сокращения аппаратных затрат кадровой структуры, а направление и геометрическое место точек в пространстве реализаций, соответствующих выбранным параметрам движения, задает выбранное преобразование.

Каждое из преобразований, сокращающих аппаратные затраты (по числу слоевых и итерационных подграфов, числу устройств и разрядности) формирует звено сплайна в пространстве реализаций, которое можно рассматривать как прямую на плоскости, образованной соответствующей осью и осью интервала обработки данных.

Как правило, при этом характеристики кадровой структуры изменяются сразу в двух измерениях, одним из которых является интервал обработки данных. Увеличение интервала обработки данных, наблюдающееся при многих преобразованиях, увеличивает и время решения задачи, поэтому следует выбирать преобразования, которые не увеличивают интервал обработки данных.

## 2.7. Принципы выбора преобразований кадровой структуры к архитектуре и конфигурации гибридной вычислительной системы

При преобразовании кадровой структуры к области доступного аппаратного ресурса гибридной вычислительной системы путем минимизации критического ресурса следует предельно сократить число выполняемых преобразований или число точек выбора альтернатив. Для этого нужно достаточно точно рассчитывать шаг движения и координаты следующей точки пространства. Точность определения шага и координат непосредственно

влияет на число шагов для достижения очередной точки выбора альтернатив и, соответственно, на общее число преобразований. Абсолютное значение шага различно для всех преобразований и определяется параметрами кадровой структуры, доступного аппаратного ресурса и типом преобразования.

При преобразовании кадровой структуры на каждом шаге $it$ изменяются текущие значения ее параметров параллелизма: число ее слоев $L_i^{it}$ и итераций $It_i^{it}$, число устройств $Q_i^{it}$, разрядность $\rho_i^{it}$ или интервал $I_i^{it}$. Все значения целочисленные и представимые в виде произведения простых сомножителей для выбора общих с коэффициентом сокращения аппаратных затрат сомножителей, чтобы сократить производительность и аппаратные затраты наиболее рациональным способом. Если используемое для движения преобразование линейно влияет на характеристику кадровой структуры, то значение шага можно рассчитать аналитически достаточно точно, но в ряде случаев это невозможно. Как правило, при движении в верхнем октанте пространства (рис. 2.6), образованном осями «Число слоев L», «Число итераций It» и «Интервал I», значения шага рассчитываются достаточно точно с помощью коэффициента сокращения аппаратных затрат $R_T$ (см. параграф 2.4). Рассчитанное новое значение координаты по соответствующей оси обратно пропорционально коэффициенту сокращения аппаратных затрат с округлением в меньшую сторону.

При движении в нижнем октанте (рис. 2.6) по осям «Разрядность», «Число устройств» и «Интервал» определение эффективного шага преобразований затруднено как спецификой сокращаемой характеристики кадровой структуры, так и ее нелинейным влиянием на аппаратные затраты. Так, при движении по оси «Разрядность» сокращение аппаратных затрат нелинейно и определяется не только коэффициентом редукции $R_0$, но и форматом представления обрабатываемых данных (логический, целочисленный, фиксированная точка, плавающая точка) в выполняемой операции. Для логических операций («И», «ИЛИ» и «Сумма по модулю 2») и целочисленных арифметических операций сокращение аппаратных затрат

прямо пропорционально коэффициенту сокращения аппаратных затрат, для арифметических операций в формате с фиксированной точкой – близко по значению к линейному, а для операций в формате с плавающей точкой сокращение разрядности может привести даже к росту аппаратных затрат (см. Приложение 1).

Поэтому для данных с фиксированной точкой и логических данных эффективный шаг по оси «Разрядность» определяется значением коэффициента сокращения аппаратных затрат и наличием библиотечных элементов с сокращенной разрядностью. Для данных с плавающей точкой шаг для 32-разрядных операндов равен 2, для 64-разрядных – 2 или 4. Увеличение шага выше указанных значений для данных с плавающей точкой может быть оправданно только при существенном недостатке каналов либо при существенной нехватке аппаратного ресурса, вынуждающем разработчика переходить к последовательной поразрядной обработке, что в настоящее время с учетом постоянного роста числа вентилей в кристаллах ПЛИС встречается очень редко.

При движении по оси «Число устройств» происходит объединение одинаковых вычислительных операций в одном устройстве, что является элементом перехода от структурной организации вычислений к процедурной. Такое преобразование оправданно при минимизации вычислительной структуры м-кадра при нехватке аппаратного ресурса для одновременной реализации нескольких устройств. Шаг движения по оси «Число устройств» определяется операциями БП и информационными зависимостями между ними. Если БП содержит различные между собой операции, то путем сокращения числа устройств возможен переход только к процедурной организации вычислений. Теоретически возможные, но крайне редко применяемые на практике варианты совмещения в одном устройстве двух разных операций, таких, как сумматор и делитель, рассматривать не будем. При наличии в БП операций, реализуемых одинаковыми аппаратными

устройствами (сложение и вычитание, деление с умножением и обратная величина), возможно несколько вариантов их совмещения.

Преобразование максимально параллельной кадровой структуры, параметризуемой доступным вычислительным ресурсом, к целевым архитектурам гибридной вычислительной системы состоит в сокращении ее параллелизма и переходу к информационно-эквивалентной кадровой структуре, занимающей меньший аппаратный ресурс. Механизмом этих преобразований могут стать применявшиеся для решения частных задач на реконфигурируемых вычислительных системах эвристические редукционные преобразования, сокращавшие различные характеристики параллелизма кадровой структуры – число подграфов, число устройств и разрядность. Известные применения редукции [181, 182, 183] выполнялись для фиксированных параметров кадровой структуры задачи и заданного вычислительного ресурса, поэтому для применения как механизма движения в пространстве возможных реализаций их необходимо обобщить и дополнить обязательным параметром – доступным вычислительным ресурсом. Обобщение и формализация редукционных преобразований и критериев их применения для изменяемого и параметризуемого аппаратного ресурса позволит автоматизировать их использование в технологии ресурсонезависимого программирования гибридных вычислительных систем.

## 2.8. Выводы к главе 2

1) В качестве единой формы представления параллельных вычислений для различных целевых архитектур предложен информационный граф задачи, который содержит все операции обработки данных, заданные параметрами задачи. Информационный граф задачи с точностью до топологических перестановок инвариантен к различным способам организации вычислений, что позволяет минимизировать число различных вариантов параллельного представления вычислений, различающихся только порядком выполнения операций. Являясь графовой моделью,

информационный граф не зависит от архитектуры и доступного вычислительного ресурса ГВС, поэтому, согласно первому принципу ресурсонезависимого программирования, позволяет представлять вычисления для любой ее целевой архитектуры.

2) Для перехода от единой формы информационного графа со структурно-топологическим представлением вычислений и информационных зависимостей вершинами и дугами графа к устройствам и коммутационной структуре целевых архитектур предложена максимально параллельная кадровая структура, которая учитывает характеристики реализации фрагмента вычислений (латентность, интервал обработки данных, разрядность выполняемых операций и др.). В соответствии со вторым принципом ресурсонезависимого программирования максимально параллельная кадровая структура, в зависимости от архитектуры и конфигурации доступного ресурса вычислительной системы, может быть преобразована в информационно-эквивалентную структуру с меньшим параллелизмом и занимаемым аппаратным ресурсом.

3) Параметризация максимально параллельной кадровой структуры доступным вычислительным ресурсом целевых архитектур ГВС обеспечивает ресурсонезависимое представление задачи. Подстановка значений архитектуры и конфигурации в ресурсонезависимое представление задачи определяет степень параллелизма кадровой структуры и занимаемый ею аппаратный ресурс ГВС, что доказывает второе положение, выдвигаемое для защиты.

4) Предложено представление всего многообразия кадровых структур как пространства их возможных реализаций, измерениями которого являются характеристики параллелизма кадровой структуры: информационно-независимые подграфы (слои), информационно-зависимые подграфы (итерации), команды (устройства), разрядность, интервал обработки данных и время решения задачи. Сокращение параллелизма (секвентирование) максимально параллельной кадровой структуры может

быть представлено движением в пространстве возможных реализаций, траекторией которого является пространственная ломаная линия – кусочно-непрерывный линейный сплайн. Каждое звено сплайна отражает сокращение параллелизма кадровой структуры и/или аппаратных затрат на ее реализацию во введенных измерениях.

5) Портация как преобразование максимально параллельной кадровой структуры в информационно-эквивалентную структуру с меньшим параллелизмом, реализуемую на заданном аппаратном ресурсе гибридной вычислительной системы, может быть представлена движением в пространстве возможных реализаций от ресурсонезависимого представления МПКС к кадровым структурам, лежащим в области доступного аппаратного ресурса заданной конфигурации целевых архитектур ГВС. Достижение кадровой структурой области доступного аппаратного ресурса является локальной целью портации, а поиск в этой области рационального по времени решения варианта кадровой структуры – глобальной целью.

6) Введено понятие *критического* ресурса как параметра кадровой структуры, доля которого в ее общих аппаратных затратах в наибольшей мере превышает доступный ресурс целевых архитектур. В пространстве реализаций критический ресурс соответствует наиболее удаленному от области доступного аппаратного ресурса компоненту целевой архитектуры ГВС. Достижение области доступного аппаратного ресурса как локальной цели портации выполняется минимизацией критического ресурса, т.е. сокращением параллелизма кадровой структуры (слоев, итераций, команд/устройств, разрядности и интервала) во введенных измерениях, чтобы связанные с этими параметрами аппаратные затраты кадровой структуры находились в области доступного аппаратного ресурса. В результате минимизации критического ресурса на каждом участке движения в пространстве возможных реализаций кадровая структура приближается к области доступного аппаратного ресурса.

7) После достижения на текущем интервале движения критическим ресурсом кадровой структуры области доступного аппаратного ресурса

происходит его перерасчет, по результатам которого возможно изменение направления дальнейшего движения. Вершины сплайна, отражающего траекторию движения кадровой структуры в пространстве возможных реализаций, в которых меняются направление движения и/или критический ресурс, называются точками выбора альтернатив. Число точек выбора альтернатив определяет количество звеньев сплайна при движении кадровой структуры в пространстве возможных реализаций для достижения локальной и глобальной целей и количество преобразований кадровой структуры, а, следовательно, и время ее портации. Показано, что максимальное число точек выбора альтернатив зависит от суммарного числа компонент всех целевых архитектур ГВС, что определяет число преобразований кадровой структуры для достижения локальной цели портации – области доступного аппаратного ресурса в пространстве возможных реализаций. Число компонент целевых архитектур для рассматриваемых ГВС сравнительно невелико, поэтому количество преобразований кадровой структуры для достижения локальной цели портации существенно меньше числа вариантов, анализируемого распараллеливающим компилятором для систем с распределенной памятью. Получена оценка количества точек выбора альтернатив для достижения локальной цели для целевой архитектуры ПЛИС.

8) Предложены критерии выбора предпочтительных направлений и параметров движения в пространстве возможных реализаций кадровых структур. Для сокращения параллелизма кадровой структуры в пространстве возможных реализаций при портации предложен механизм достижения области доступного аппаратного ресурса – редукция производительности и аппаратных затрат.

# 3. МЕТОДЫ И МЕХАНИЗМЫ ПРЕОБРАЗОВАНИЯ ПАРАМЕТРИЗУЕМОЙ РЕСУРСОМ КАДРОВОЙ СТРУКТУРЫ К ЦЕЛЕВЫМ АРХИТЕКТУРАМ ГИБРИДНОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

Согласно третьему принципу теоретических основ ресурсонезависимого программирования (см. параграф 1.8) об использовании характеристик аппаратной реализации при выборе наиболее рационального варианта решения, правила движения в пространстве возможных реализаций кадровых структур (см. параграф 2.5) учитывают латентность, интервал обработки данных, разрядность, число операций и другие характеристики кадровой структуры при выборе наиболее рационального варианта реализации прикладной задачи или ее фрагмента на целевых архитектурах гибридной вычислительной системы. Для реализации четвертого принципа, т.е. возможности изменения способа организации вычислений, частоты, числа устройств и других характеристик для каждого фрагмента задачи, их нужно дополнить механизмами преобразования кадровой структуры задачи к рациональному для целевой архитектуры варианту организации параллельных вычислений.

## 3.1 Механизмы преобразования кадровых структур к целевой архитектуре гибридной вычислительной системы

В [181-187] описано сбалансированное решение задачи молекулярного докинга на РВС, при синтезе которого из-за нехватки аппаратного ресурса для структурной реализации базового подграфа применялись редукционные преобразования кадровой структуры. При проектировании структурно-процедурного решения задачи докинга прикладной программист преобразовывал максимально параллельную кадровую структуру в минимальную, вручную выполняя редукционные преобразования для каждого фрагмента задачи: подбирал число аппаратно реализуемых базовых подграфов, число устройств и разрядность обрабатываемых данных с учетом

заданного аппаратного ресурса, согласовывал фрагменты с разной степенью параллелизма с помощью интервала и частоты обработки данных. В результате длительной и трудоемкой разработки получалось структурно-процедурное решение задачи молекулярного докинга на РВС, оптимизированное под заданный аппаратный ресурс и параметры задачи.

Для обозначения комплекса частных преобразований, примененных прикладным программистом для фиксированных параметров задачи и заданного аппаратного ресурса, в [181] использовался термин «редукция производительности и аппаратных затрат». Преобразование кадровой структуры молекулярного докинга к заданному ресурсу РВС показало принципиальную возможность сокращения аппаратных затрат и производительности с помощью редукционных преобразований.

Отдельные редукционные преобразования из примененного комплекса использовались при решении задач из других предметных областей. В [182] описано применение редукции по числу устройств для реализации операции быстрого преобразования Фурье в задачах цифровой обработки сигналов, а в [183] редукция применялась для реализации алгоритма SRMP, алгоритмов сортировки и решения обратной кинематической задачи сейсморазведки, относящейся к задачам реального времени. Частные редукционные преобразования [181-183], предложенные для решения конкретных задач на реконфигурируемых вычислительных системах, можно расширить и обобщить для широкого круга предметных областей, чтобы использовать редукцию производительности и аппаратных затрат для преобразования кадровой структуры к рациональному варианту организации параллельных вычислений на доступном аппаратном ресурсе вычислительной системы. Формализация механизмов редукционных преобразований позволит использовать их для автоматического преобразования кадровых структур при движении в пространстве возможных реализаций не только для реконфигурируемых, но и для гибридных вычислительных систем.

Для реализации параллельных вычислений и построения общего метода сбалансированной редукции производительности и аппаратных затрат кадровой структуры необходимо от упомянутых частных эвристик [181-183] перейти к обобщенным преобразованиям, параметром которых является аппаратный ресурс [188-191]. Для построения формального метода сбалансированной редукции, параметризуемой аппаратным ресурсом, рассмотрим математическое представление редукционных преобразований и определим критерии, при которых сокращается не только производительность, но и аппаратные затраты.

Производительность вычислительной системы – это количественная характеристика скорости выполнения операций. Известны различные метрики определения производительности вычислительных систем, как, например, IPS(MIPS) [1, 5, 21], FLOPS и ее производные, сравнение производительности на основе времени решения фиксированного набора тестов (TOP-500 и др.). Из-за различий в архитектурах, системах команд и принципах организации вычислений для большинства вычислительных систем, определим производительность системы как число операций, выполненных за время решения прикладной задачи. Для исключения влияния архитектурных особенностей вычислительных систем при выполнении операций, разрядности исполнительных устройств и различных форматов представления данных, будем понимать производительность как общее число вычислительных операций над двоичными разрядами для получения результата. Это позволяет оценивать производительность задачи $P$ независимо от модели организации и реализации вычислений, а также сгладить различия в моделях целевых архитектур гибридной вычислительной системы. Для простоты рассуждений будем считать, что информационный граф задачи $P$ функционально-регулярный, число подграфов в слоях постоянно для всех итераций, количество устройств и разрядность обработки данных также постоянны. Примером такой задачи является решение СЛАУ методом Якоби.

Общее число вычислительных операций (**N**umber of **C**alculations) $NC_P$ максимально параллельной кадровой структуры прикладной задачи $P$ можно оценить как произведение числа слоевых $N_F$ и итерационных $N_\Phi$ подграфов, числа устройств $Op$ в МинКС и разрядности обрабатываемых данных $\rho$:

$$NC_P = N_F \cdot N_\Phi \cdot Op \cdot \rho. \quad (3.1)$$

Для функционально-регулярных задач с параметрами подграфов, изменяющимися в пределах слоя и от итерации к итерации, можно использовать приведение информационного графа к функционально-регулярной форме с помощью операции дополнения [75], после чего оценить производительность согласно (3.1). Для нерегулярных задач произведение числа подграфов $N_F \cdot N_\Phi$ в (3.1) нужно заменить на сумму числа слоевых подграфов в каждой итерации и сумму числа подграфов во всех итерациях.

Время решения задачи $P$ для максимально параллельной кадровой структуры будет пропорционально латентности этой структуры $lat$ и длительности такта обработки $\tau$:

$$t_P = lat \cdot \tau. \quad (3.2)$$

Тогда, с учетом введенных обозначений, производительность максимально параллельной кадровой структуры при решении прикладной задачи $P$ представима как

$$Perf_P = \frac{NC_P}{t_P} = \frac{N_\Phi \cdot N_F \cdot Op \cdot \rho}{lat \cdot \tau} = \frac{N_\Phi \cdot N_F \cdot Op \cdot \rho \cdot Fr}{lat}, \quad (3.3),$$

где $Fr = \frac{1}{\tau}$ – частота обработки данных.

Выражение (3.3) описывает максимальную производительность вычислительной системы при решении задачи $P$, т.е. производительность аппаратной реализации функционально-регулярного информационного графа как максимально параллельной кадровой структуры. Редукция производительности (3.3) с целочисленным коэффициентом $R$ – это сокращение производительности $Perf_P$ в целое число раз обратно пропорционально значению коэффициента редукции $R$

$$Perf_P(R) = \frac{Perf_P}{R} = \frac{N_\Phi \cdot N_F \cdot Op \cdot \rho \cdot Fr}{lat \cdot R}. \quad (3.4)$$

В математической постановке редукция производительности максимально параллельной кадровой структуры, сводится к сокращению числителя дроби (3.4) на значение коэффициента редукции $R$. Наилучший способ редукции – это сокращение числителя и знаменателя на общие множители. Редуцировать производительность (3.4) можно различными способами: сокращением числа аппаратно реализованных подграфов $N_{\Phi}$ или $N_F$ в МинКС задачи (традиционный способ для РВС [6, 75, 102]), сокращением числа устройств $Op$, реализующих МинКС, сокращением разрядности обрабатываемых данных $\rho$ либо сокращением частоты обработки $Fr$ [181, 184, 190, 191]. При сокращении общих множителей числителя и знаменателя в первых трех случаях можно сократить как производительность, так и аппаратные затраты на реализацию кадровой структуры, если полученное решение будет сбалансированным и не потребует ресурсов для дополнительной коммутации и синхронизации вычислений, превышающих сокращенные ресурсы. При сокращении частоты обработки данных $Fr$ сокращается только производительность, аппаратные затраты не меняются (незначительным сокращением затрат на синхронизацию можно пренебречь).

Наилучший случай для выполнения редукции – это пропорциональное сокращение дроби (3.4) на значение коэффициента редукции $R$ или, если это невозможно, то сокращение общих сомножителей коэффициента редукции $R$ и характеристик $N_{\Phi}, N_F$, $Op$, $\rho$ и $Fr$ кадровой структуры

$$Perf_P(R) = \frac{n_{\Phi 1} \cdot n_{\Phi 2} \cdot \ldots \cdot n_{\Phi c} \cdot n_{F1} \cdot n_{F2} \cdot \ldots \cdot n_{Fd} \cdot op_1 \cdot op_2 \cdot \ldots \cdot op_e \cdot \rho_1 \cdot \rho_2 \cdot \ldots \cdot \rho_F \cdot Fr_1 \cdot Fr_2 \cdot \ldots \cdot Fr_H}{lat \cdot r_1 \cdot r_2 \cdot \ldots \cdot r_k}. \quad (3.5)$$

Отличительная характеристика любой кадровой структуры – это одинаковый темп обработки данных во всех ее фрагментах, обеспечивающий сбалансированность ее конвейерной реализации. Поэтому применение редукционных преобразований, в отличие от механического сокращения параметров реализации кадровой структуры, также должно обеспечивать одинаковый темп обработки данных для редуцированной кадровой структуры. Отличием редукции производительности и аппаратных затрат как

преобразования структуры прикладной задачи от тривиального математического сокращения сомножителей дроби (3.5) является не только сокращение дроби, но и поиск наиболее рационального варианта кадровой структуры, реализуемого на доступном аппаратном ресурсе и минимизирующего необходимость дополнительного согласования интенсивности потоков данных между фрагментами задачи $P$. Под фрагментом здесь и далее будем подразумевать подзадачу задачи $P$, хотя понятие подзадачи шире и она может содержать несколько вычислительных фрагментов. Методы структурно-процедурной организации вычислений позволяют найти баланс между сокращением числа аппаратно реализованных базовых подграфов задачи $N_{\Phi}$ или $N_F$ и ее производительностью. Сокращение остальных параметров минимальной кадровой структуры не формализовано и выполнялось вручную прикладным программистом при оптимизации структурной реализации. Примеры и детали реализации каждого из перечисленных редукционных преобразований при решении задачи докинга на РВС подробно изложены в [181].

При комплексной редукции прикладной задачи, содержащей несколько вычислительных фрагментов, для каждого фрагмента возможны собственные уникальные варианты сокращения числителя на сомножители коэффициента редукции $R$. При сокращении различных параметров в разных фрагментах задачи $P$ может понадобиться согласование потоков данных между ними из-за различия в параметрах обработки: интервала, разрядности и др. Это требует дополнительных аппаратных затрат и увеличивает время решения задачи. Поэтому основная проблема комплексной редукции задачи, состоящей из нескольких вычислительных фрагментов, заключается в подборе рациональной комбинации сокращаемых в каждом фрагменте сомножителей, которая уменьшает аппаратные затраты и минимизирует накладные расходы на согласование информационных потоков между фрагментами с различающимися параметрами реализации вычислительных структур ($N_{\Phi}, N_F$, $Op$, $\rho$ и Fr). Формализация преобразований для совместного и комплексного

применения при сокращении аппаратных затрат путем сокращения сомножителей разных параметров ($N_{\Phi}, N_F$, $Op$, $\rho$ и Fr) числителя и коэффициента редукции $R$ согласно (3.5) позволит использовать редукцию для сбалансированного преобразования кадровой структуры при движении в пространстве возможных реализаций.

Для использования редукционных преобразований как методов преобразования кадровой структуры задачи к рациональному варианту организации параллельных вычислений на заданном аппаратном ресурсе необходимо разработать основные принципы их комплексного применения. Редукционные преобразования будем обозначать символом $R^T$, где верхний индекс – это тип используемого преобразования, т.е. $R^N$ для редукции по числу подграфов, $R^{Op}$ - для редукции по числу устройств, $R^{\rho}$- для редукции по разрядности, $R^{Freq}$ - для редукции по частоте и $R^I$ - для редукции интервала обработки данных. Рассмотрим основные принципы выполнения редукционных преобразований и их влияние на характеристики кадровой структуры.

### 3.1.1. Редукция по числу подграфов $R^N$

Редукция по числу подграфов $R^N$ сокращает число аппаратно реализованных подграфов исходного информационного графа в кадровой структуре. Редукция $R^N$ – это основной механизм движения по осям «Число слоев» и «Число итераций» пространства возможных кадровых структур (рис. 2.6). Редукцию по числу подграфов $R^N$ можно представить как композицию редукционных преобразований по слоям и итерациям, параметризуемую аппаратным ресурсом:

$$R^N = R^F \underset{\circ}{A_{HCS}} R^{\Phi}, \tag{3.6}$$

где $R^F$ - редукция по числу слоевых подграфов, заданных функцией $F_{ij}(A_{HCS})$, а $R^{\Phi}$ – редукция по числу итерационных подграфов, заданных функцией $\Phi_i(A_{HCS})$, и $A_{HCS}$ – доступный аппаратный ресурс гибридной вычислительной системы. Обычно при редукции по числу слоевых подграфов $R^F$ сокращается

занимаемый аппаратный ресурс и увеличивается длина потока данных для оставшихся подграфов, т.е. входные данные редуцированных подграфов упорядочиваются в кортеж, формируя поток входных данных для редуцированной кадровой структуры. Поэтому редукция по числу слоевых подграфов $R^F$ - это один из основных методов не только для структурно-процедурной, но и для мультипроцедурной организации вычислений: для кластерных вычислительных систем она соответствует пропорциональному перераспределению числа данных, обрабатываемых каждым процессором, при изменении общего числа задействованных в вычислениях процессоров.

Редукция по числу итерационных подграфов $R^{\Phi}$ так же, как и редукция по числу слоевых $R^F$, линейно сокращает аппаратный ресурс, и в ряде случаев сокращает число каналов, не меняя их разрядность. Число каналов может сократиться для итерационных зависимостей вида x[i]=f(x[i], a[i], b[i],..) (где a[i], b[i] – отдельные каналы памяти), а для итерационных зависимостей вида x[i]=f(x[i-t]) (не требующих дополнительных каналов памяти для каждой итерационной ступени) число каналов не изменится. При редукции по числу итерационных подграфов $R^{\Phi}$ существенно увеличивается интервал обработки данных для оставшихся подграфов из-за появления обратной связи, что необходимо учитывать при ее использовании. Сокращение интервала обработки данных требует специальных оптимизационных преобразований (см. Приложение 1).

Редукция по числу подграфов $R^N$ [181] позволяет с линейным коэффициентом сократить число реализуемых подграфов до единственного подграфа в МинКС, необходимого для решения задачи. Редукция $R^N$ – это одно из наиболее привычных преобразований, которое используется как при структурно-процедурной, так и при процедурной организации вычислений.

### 3.1.2. Редукция по числу устройств $R^{Op}$

Редукция по числу устройств $R^{Op}$ – это механизм движения по оси «Число команд» в нижнем октанте пространства возможных кадровых

структур (рис. 2.6). Редукция $R^{Op}$ сокращает число устройств в МинКС за счет совмещения и последовательного выполнения операций с одинаковыми типами данных в одном устройстве. В результате редукции по числу устройств $R^{Op}$ происходит переход от параллельной обработки входных данных к параллельно-последовательной. В результате увеличивается интервал обработки данных в редуцированной вычислительной структуре, а также общее время решения задачи. Предельный случай редукции по числу устройств $R^{Op}$ - это последовательная (процедурная) обработка данных, когда все операции МинКС совмещаются в одном вычислительном устройстве – процессоре, который в зависимости от управляющей команды выполняет различные операции (например, сложение, умножение, деление, вычисление тригонометрических функций и др).

При редукции по числу устройств $R^{Op}$ в кадровой структуре синтезируется коммутационная структура для синхронизации операндов, поэтому аппаратные затраты сокращаются нелинейно. При совмещении большого числа операций в одном устройстве рост накладных расходов на синхронизацию может сравняться с выигрышем аппаратного ресурса от сокращения устройств. Принцип действия редукции по числу устройств $R^{Op}$ рассмотрим на примере базовой операции (БО) быстрого преобразования Фурье (БПФ), которая для комплексных чисел на уровне математической модели задается следующими математическими формулами [182]:

$$\dot{A}_{t+1} = \dot{A}_t + \dot{B}_t,$$
$$\dot{B}_{t+1} = (\dot{A}_t - \dot{B}_t) \cdot \dot{W}_t,$$
$$\dot{W}_t = \dot{W}_B \cdot \dot{W}_H, \qquad (3.7)$$

где $\dot{A}_t$ и $\dot{B}_t$- комплексные данные, поступающие на входы алгоритма; $\dot{W}_B$ и $\dot{W}_H$ – базовые и вспомогательные комплексные весовые коэффициенты.

Так как комплексные данные представляются отдельно вещественной и мнимой частями, то формулы вычислительной модели примут вид:

$$
\begin{aligned}
&Re[\dot{A}_{t+1}] = Re[\dot{A}_t] + Re[\dot{B}_t],\\
&Im[\dot{A}_{t+1}] = Im[\dot{A}_t] + Im[\dot{B}_t],\\
&Re[\dot{B}_{t+1}] = \left(Re[\dot{A}_t] - Re[\dot{B}_t]\right) \cdot Re[\dot{W}_t] - \left(Im[\dot{A}_t] - Im[\dot{B}_t]\right),\\
&Im[\dot{B}_{t+1}] = \left(Re[\dot{A}_t] - Re[\dot{B}_t]\right) \cdot Im[\dot{W}_t] + \left(Im[\dot{A}_t] - Im[\dot{B}_t]\right), \qquad (3.8)\\
&Re[\dot{W}_t] = Re[\dot{W}_B] \cdot Re[\dot{W}_H] - Im[\dot{W}_B] \cdot Im[\dot{W}_H],\\
&Im[\dot{W}_t] = Re[\dot{W}_B] \cdot Im[\dot{W}_H] + Im[\dot{W}_B] \cdot Re[\dot{W}_H].
\end{aligned}
$$

Для БО БПФ граф алгоритма и информационный граф структурной реализации по (3.8) совпадают. На рис. 3.1 показан их вид для случая, когда значения весовых коэффициентов $\dot{W}_B$ и $\dot{W}_H$ рассчитаны заранее и хранятся в отдельных массивах. Информационный граф БО БПФ в 4 слоях содержит 16 операций (4 суммирования, 4 вычитания и 8 умножений), которые обрабатывают данные, поступающие по 8 входным каналам. Выходы всех устройств пронумерованы для удобного представления сути редукционных преобразований. Для одной пары комплексных чисел максимально параллельная кадровая структура БО БПФ будет аналогичной структуре (рис. 3.1), а для множества данных приведенная структура будет информационно-независимо мультиплицирована в МПКС.

При редукции по числу устройств $R^{Op}$ одинаковые операции информационного графа назначаются одному вычислительному устройству. Для информационного графа (рис. 3.1) можно совместить операции каждой группы (сложения, вычитания и умножения) в одном или нескольких устройствах[192, 193]. Учитывая, что операции сложения и вычитания при аппаратной реализации и представлении данных в дополнительном коде выполняются, как правило, одним устройством, в кадровой структуре можно сократить число групп до двух: 8 операций «сложение-вычитание» и 8 умножений. Число совмещаемых в одном устройстве операций каждой группы задается коэффициентом редукции $R^{Op}$, который определяет и интервал подачи данных.

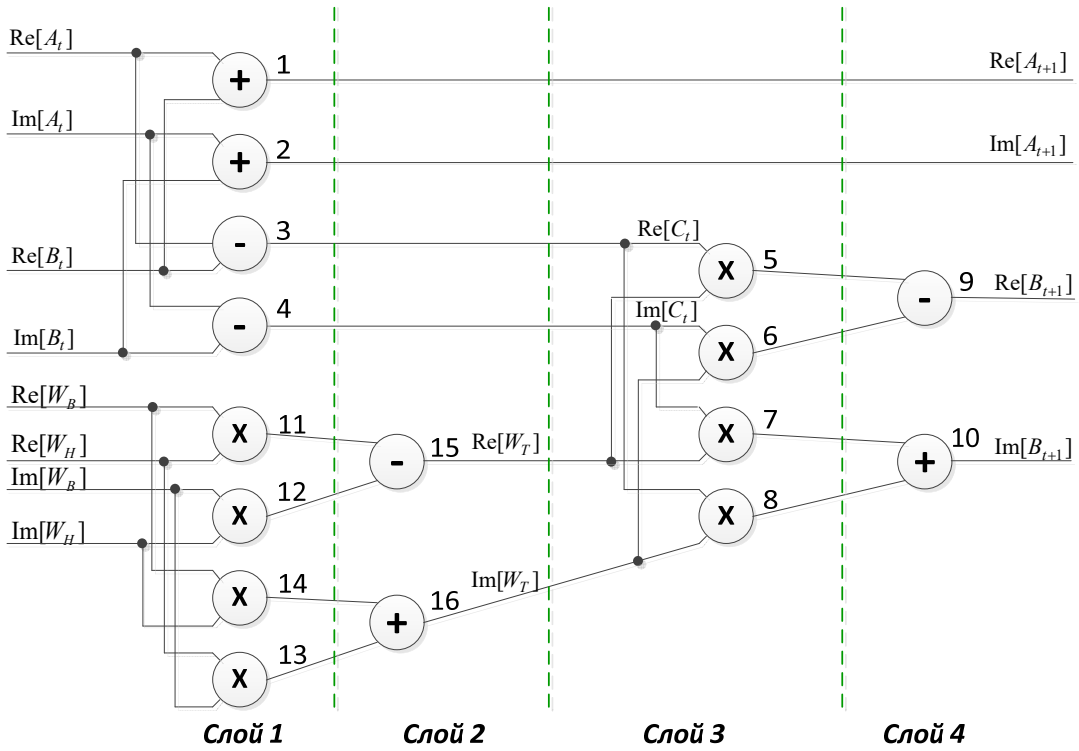
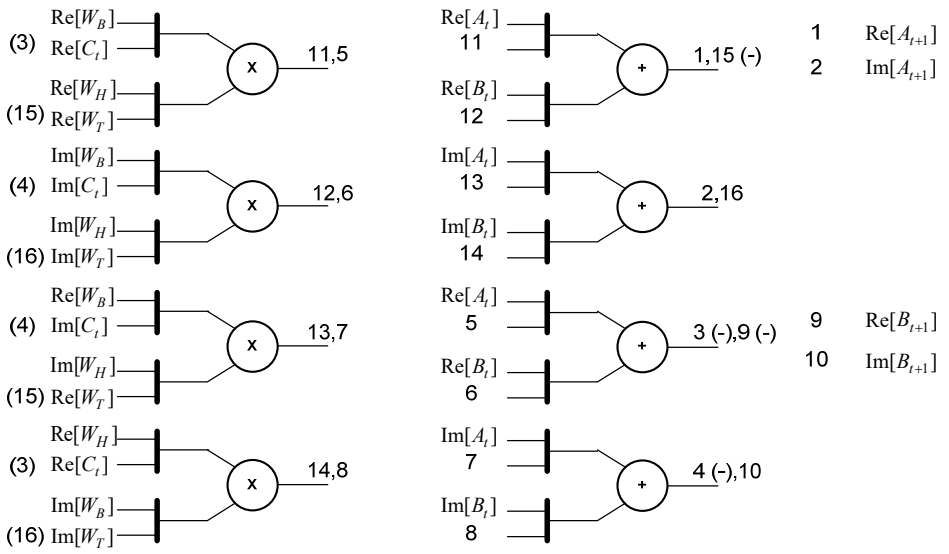
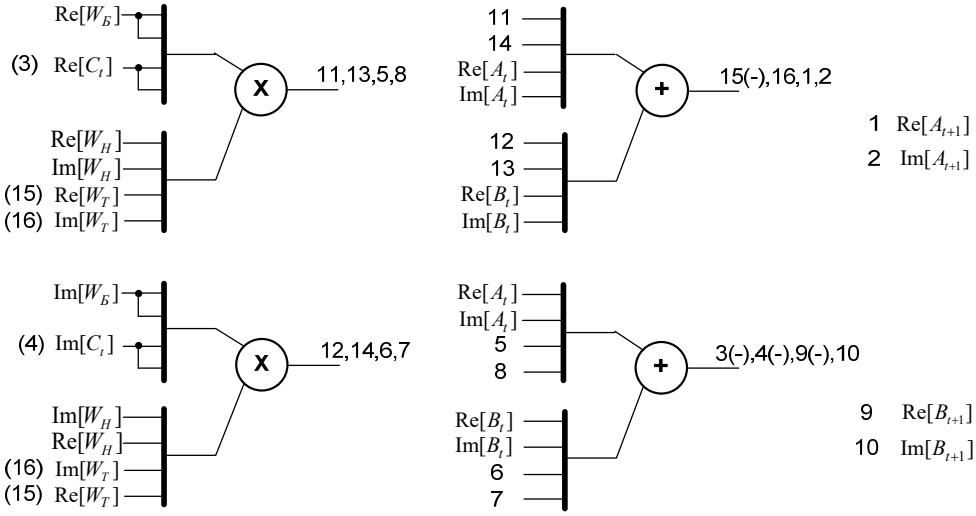



Рис. 3.1. Структурная реализация информационного графа БО БПФ

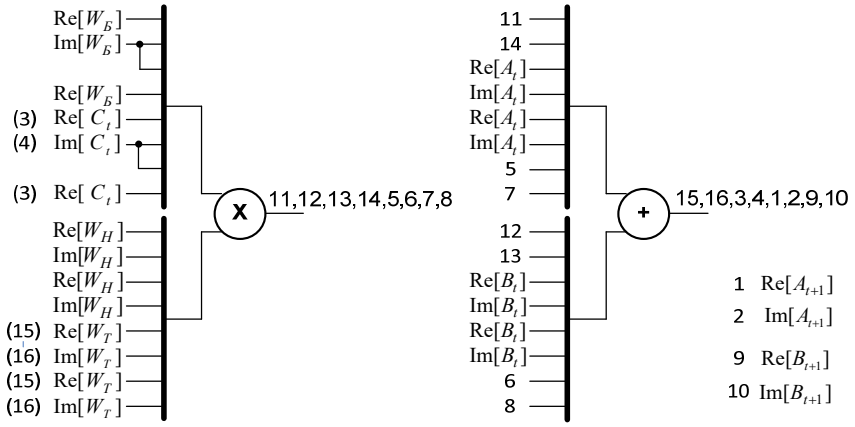
Так, при двукратной редукции по числу устройств $R^{Op} = 2$ число устройств в редуцированной кадровой структуре БО БПФ (рис. 3.2-а) сократится в два раза до 8 устройств: 4 устройства суммирования-вычитания и 4 устройства умножения с интервалом обработки данных $I = 2$. При $R^{Op} = 4$ число устройств в редуцированной структуре (рис. 3.2-б) сократится до четырех: 2 сумматора и 2 умножителя с интервалом обработки $I = 4$ такта, а для $R^{Op} = 8$ (рис. 3.2-в) редуцированная по числу устройств кадровая структура содержит два устройства: 1 сумматор и 1 умножитель с интервалом обработки $I = 8$ тактов). В приведенных примерах редукции по устройствам (рис. 3.2) число операций каждой группы кратно коэффициенту редукции. Если число операций в группе не делится нацело на коэффициент редукции, то результат деления (число устройств в редуцированной кадровой структуре) округляется вверх к ближайшему целому. Редуцированная кадровая структура будет менее эффективной, т.к. для синхронизации и выравнивания операндов потребуются дополнительные аппаратные затраты, а интервал обработки данных увеличится пропорционально коэффициенту редукции.




а) редукция БО БПФ по числу устройств с коэффициентом $R^{Op} = 2$




б) редукция БО БПФ по числу устройств с коэффициентом $R^{Op} = 4$




в) редукция БО БПФ по числу устройств с коэффициентом $R^{Op} = 8$

Рис. 3.2. Редукция кадровой структуры БО БПФ по устройствам $R^{Op}$

Например, при некратном увеличении коэффициента редукции до $R^{Op} = 5$ для БО БПФ, редуцированная кадровая структура будет содержать 2

сумматора и 2 умножителя, как и при $R^{Op} = 4$, но интервал подачи данных увеличится с 4 до 5, что в 1,25 раза увеличит время решения. Такая же несбалансированность будет наблюдаться для значений $R^{Op} = 6$ и $R^{Op} = 7$. Поэтому при редукции по числу устройств следует подбирать кратный коэффициент, учитывая число операций базового подграфа редуцируемой кадровой структуры.

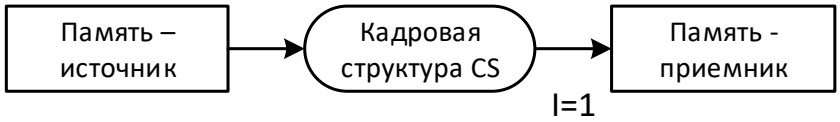
Для рассматриваемого примера БО БПФ предельным значением коэффициента редукции является $R^{Op} = 16$, при котором все операции совмещаются в одном устройстве (процессоре), выполняющем операции сложения, вычитания и умножения, и переходят в последовательную (процедурную) организацию вычислений. Увеличение коэффициента редукции для $R^{Op} > 16$ не позволит сократить аппаратные затраты, т.к. значение коэффициента превышает число исходных устройств, а сократить число устройств менее одного невозможно. Дальнейшее сокращение производительности достижимо только с помощью других механизмов, таких как редукция по разрядности для сокращения аппаратных затрат и/или редукция по интервалу обработки данных для сокращения производительности или темпа обработки данных.

### 3.1.3. Редукция кадровых структур к микро-кадрам

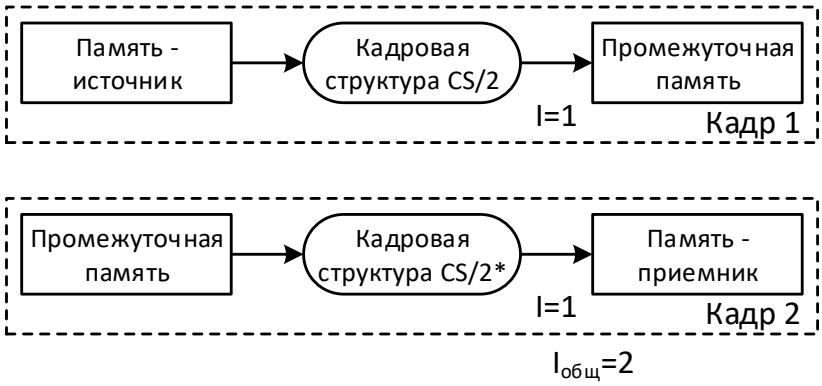
Для обозначения редуцированных по числу устройств кадровых структур определим новое понятие – микро-кадр. Микро-кадр или м-кадр, названный по аналогии с кадровой структурой в структурно-процедурной организации вычислений, имеет ряд принципиальных отличий. Основное отличие м-кадра от кадровой структуры – это совмещение нескольких операций в одном вычислительном устройстве, приводящее к увеличению интервала обработки данных, но обеспечивающее единый и сбалансированный темп обработки данных без использования

дополнительной памяти для хранения промежуточных результатов обработки входного потока данных (рис. 3.3).

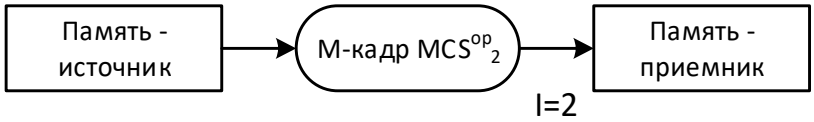
Если аппаратный ресурс вычислительной системы достаточен для структурной реализации кадровой структуры CS (рис. 3.3-а), то при ее реализации обеспечивается единый для всех ее фрагментов сбалансированный интервал обработки данных, проходящих потоком из каналов КРП памяти-источника в КРП памяти-приемника результатов.




а) структурная реализация исходной кадровой структуры CS




б) структурно-процедурная реализация кадровой структуры CS при нехватке аппаратного ресурса




в) реализация микро-кадра MCS как редукционного преобразования кадровой структуры CS при нехватке аппаратного ресурса

Рис. 3.3. Структурная схема способов реализации кадровой структуры

При нехватке аппаратного ресурса для структурной реализации, кадровая структура реализуется структурно-процедурно (рис. 3.3-б) несколькими кадрами: в первом кадре на доступном аппаратном ресурсе вычислительной системы реализуется фрагмент исходной структуры CS в виде кадровой структуры «CS/2», которая выполняет часть операций для всего

потока данных и записывает результат в промежуточную память, а в следующем кадре кадровая структура «CS/2» перенастраивается для выполнения операций следующего фрагмента вычислений и обрабатывает поток данных из промежуточной памяти. Это приводит либо к разрыву потока данных и кратно числу кадров сокращает производительность кадровой структуры, либо к сложным и не всегда реализуемым механизмам работы памяти в «старт-стопном» режиме (пакетной подачи данных с паузой). На рис. 3.3 представлен пример разделения исходной кадровой структуры на два кадра, но в общем случае число кадров может быть больше, что определяется соотношением между аппаратным ресурсом, необходимым для структурной реализации кадровой структуры CS, и доступным аппаратным ресурсом вычислительной системы. Каждый из кадров (рис. 3.3-б), как и исходная кадровая структура, работает с единичным интервалом обработки данных, но общий интервал, как и время решения задачи, увеличатся вдвое.

М-кадр MCS (рис. 3.3-в), в отличие от структурно-процедурной реализации «CS/2», обрабатывает входной поток данных с интервалом, равным 2, но при этом не требует дополнительной выделенной памяти для хранения промежуточных результатов обработки. Если нужно сохранить промежуточные результаты для обрабатываемого пакета данных, они хранятся в регистрах и элементах задержки м-кадра. Поскольку промежуточные результаты хранятся только для текущего пакета данных, а не для всего потока, объем используемой памяти невелик. Коммутационная структура «на лету» обеспечивает перенастройку операций, выполняемых устройствами м-кадра, и коммутацию входных данных. Устройства м-кадра обрабатывают непрерывный поток данных, подаваемых с некоторым интервалом (например, для (рис. 3.3-в) интервал равен 2).

Как видно, общий интервал и время обработки данных для структурно-процедурной реализации кадровой структуры «CS/2» и м-кадра MCS совпадают (временем перенастройки кадров пренебрежем). При этом аппаратные затраты при реализации м-кадра существенно (на величину

памяти, необходимой для хранения обрабатываемого потока данных) меньше, чем при структурно-процедурной организации вычислений. Более того, если в исходной кадровой структуре CS есть обратная связь, то ее структурно-процедурная реализация сильно усложняется и зачастую невозможна без специальных преобразований и оптимизаций, в то время как реализация в виде микро-кадра обеспечит обработку непрерывного потока данных с некоторым неединичным интервалом. Предельным случаем м-кадра является последовательная процедурная реализация вычислений. М-кадры для примера БО БПФ с другими коэффициентами приведены в Приложении 1. Там же представлены теоретические обоснования и методы выбора состава вычислительных устройств для м-кадров, примеры редукции задач и подзадач различных предметных областей.

Для редукции по числу устройств можно предложить различные варианты синтеза м-кадров[191] (Приложение 1), например:

1. М-кадр $\mu_1$ (минимальный), в котором все операции одного вида реализуются не более, чем на одном устройстве. Для рассматриваемого примера БО БПФ, содержащего две группы операций, м- кадр $\mu_1$ будет иметь два устройства – умножитель и сумматор (рис. 3.2-в).

2. М-кадр $\mu_2$ (кратный), в котором число устройств для всех групп операций сокращается в одинаковое целое число раз, равное коэффициенту редукции, что соответствует «вынесению общего множителя» для операций базового подграфа кадровой структуры. Синтез м-кадра $\mu_2$ определяется исходной кадровой структурой и не всегда возможен (например, если число операций базового подграфа не содержит общих делителей). Для БО БПФ м-кадр $\mu_2$ возможен, причем сразу в трех вариантах: 8 устройств (4 умножителя и 4 сумматора - рис. 3.2-а), 4 устройства (2 умножителя и 2 сумматора - рис. 3.2-б) и 2 устройства (1 умножитель и 1 сумматор - рис. 3.2-в).

3. М-кадр $\mu_3$, который синтезируется из максимального по числу операций яруса базового подграфа, для соблюдения информационной эквивалентности дополненного устройствами, реализующими остальные

операции кадровой структуры. Для рассматриваемого примера м-кадр $\mu_3$ будет содержать 8 устройств из первого яруса (4 умножителя и 4 сумматора - рис. 3.2-а). Для м-кадра $\mu_3$ ярус базового подграфа, содержащий наибольшее число операций, включается полностью и будет реализован за один такт, что может сократить затраты на синхронизацию вычислений.

4. М-кадр $\mu_4$, который содержит все устройства из наиболее разнородного по типу операций яруса, при необходимости дополненные устройствами для соблюдения информационной эквивалентности. Для рассматриваемого примера м-кадр $\mu_4$ содержит 8 устройств из первого яруса (4 умножителя и 4 сумматора).

5. М-кадр $\mu_5$ - это минимальный м-кадр $\mu_1$ с одним дополнительным устройством, реализующим наиболее часто встречающуюся в базовом подграфе операцию. Для примера БО БПФ м-кадр $\mu_5$ будет содержать 3 устройства. Здесь возможны два варианта: 2 умножителя и 1 сумматор или 1 умножитель и 2 сумматора.

Представленный список вариантов м-кадров сформирован на основе анализа подграфов задач цифровой обработки сигналов, символьной обработки, линейной алгебры и молекулярного докинга, успешно решаемых на РВС.

Для задач других предметных областей данный перечень можно дополнить новыми методами синтеза м-кадров, если их базовые подграфы являются более специфичными, чем рассмотренные. С учетом конечного числа предметных областей применения РВС общее число возможных м-кадров вряд ли превысит 10.

Все варианты м-кадров, кроме $\mu_2$, можно синтезировать для любых прикладных задач. М-кадр $\mu_1$ универсален для различных прикладных задач, а вариант редукции с помощью м-кадра $\mu_2$ наиболее эффективен, но не всегда возможен, т.к. зависит от исходной кадровой структуры. М-кадры $\mu_3, \mu_4$ и $\mu_5$ теоретически можно синтезировать для любых базовых подграфов, если они

допустимы с учетом значения коэффициента редукции, определяющим аппаратные затраты для редуцированной кадровой структуры.

Количественные оценки сокращения аппаратного ресурса вычислительных устройств м-кадра, реализующих основные математические операции для различных типов данных, представлены в табл. 3.1-3.3.

Таблица 3.1. Сокращение аппаратного ресурса при объединении в одном устройстве математических операций для 32 разрядных чисел с фиксированной точкой

| Вычислительное устройство (математическая операция) | | LUT | Flip-Flop | DSP |
|---|---|---|---|---|
| Операции сложения/вычитания с фиксированной точкой | | | | |
| Сумматор (+) | | 32 | 32 | 0 |
| Вычитатель (-) | | 32 | 32 | 0 |
| Сумматор-вычитатель (±) | | 32 | 32 | 0 |
| Операции умножения/деления с фиксированной точкой | | | | |
| Умножитель (*) | logic | 1094 | 1169 | 0 |
| | dsp | 17 | 66 | 4 |
| Делитель (/) | logic | 1260 | 2425 | 0 |
| | dsp | 764 | 1310 | 14 |
| Умножитель-делитель | logic | 1450 | 2800 | 0 |
| | dsp | 880 | 1508 | 16 |
| Операция сравнения на равенство с фиксированной точкой | | | | |
| Равенство (=) | | 11 | 1 | 0 |
| Неравенство (≠) | | 11 | 1 | 0 |
| Равенство -неравенство ($^{=}/_{\neq}$) | | 11 | 1 | 0 |
| Операция сравнения «больше»/«меньше» с фиксированной точкой | | | | |
| (≥) / (≤) | | 16 | 1 | 0 |
| Меньше (<)/Больше (>) | | 16 | 1 | 0 |
| Больше либо равно -меньше ($^{\geq}/_{<}$)/Меньше либо равно -больше ($^{\leq}/_{>}$) | | 16 | 1 | 0 |

Таблица 3.2. Сокращение аппаратного ресурса при объединении в одном устройстве математических операций для 32 разрядных чисел с плавающей точкой

| Вычислительное устройство (математическая операция) | | LUT | Flip-Flop | DSP |
|---|---|---|---|---|
| Операции сложения/вычитания с плавающей точкой | | | | |
| Сумматор (+) | logic | 576 | 1260 | 0 |
| | dsp | 526 | 1159 | 1 |
| Вычитатель (-) | logic | 577 | 1260 | 0 |
| | dsp | 526 | 1159 | 1 |
| Сумматор-вычитатель (±) | logic | 575 | 1261 | 0 |
| | dsp | 523 | 1160 | 1 |
| Операции умножения/деления с плавающей точкой | | | | |
| Умножитель (*) | logic | 570 | 670 | 0 |
| | dsp | 89 | 164 | 2 |
| Делитель (/) | logic | 1134 | 1960 | 0 |
| | dsp | 653 | 527 | 26 |
| Умножитель-делитель) | logic | 1305 | 2254 | 0 |
| | dsp | 753 | 607 | 30 |
| Операция сравнения на равенство с плавающей точкой | | | | |
| Равенство (=) | | 29 | 6 | 0 |
| Неравенство (≠) | | 47 | 9 | 0 |
| Равенство -неравенство ($^{=}/_{\neq}$) | | 29 | 6 | 0 |
| Операция сравнения «больше» или «меньше» с плавающей точкой | | | | |
| Больше либо равно (≥) | | 47 | 9 | 0 |
| Меньше (<) | | 47 | 8 | 0 |
| Больше либо равно -меньше ($^{\geq}/_{<}$) | | 47 | 9 | 0 |
| Операция сравнения «меньше» с плавающей точкой | | | | |
| Меньше либо равно (≤) | | 47 | 9 | 0 |
| Больше (>) | | 46 | 8 | 0 |
| Меньше либо равно -больше ($^{\leq}/_{>}$) | | 47 | 9 | 0 |

Таблица 3.3. Сокращение аппаратного ресурса при объединении в одном устройстве математических операций для 64 разрядных чисел с плавающей точкой

| Вычислительное устройство (математическая операция) | | LUT | Flip-Flop | DSP |
|---|---|---|---|---|
| Операции сложения/вычитания с плавающей точкой | | | | |
| Сумматор (+) | logic | 1216 | 2901 | 0 |
| | dsp | 1172 | 2788 | 4 |
| Вычитатель (-) | logic | 1331 | 2830 | 0 |
| | dsp | 1123 | 2788 | 4 |
| Сумматор-вычитатель (±) | logic | 1329 | 2829 | 0 |
| | dsp | 1121 | 2790 | 4 |
| Операции умножения/деления с плавающей точкой | | | | |
| Умножитель (*) | logic | 3963 | 3871 | 0 |
| | dsp | 251 | 982 | 10 |
| Делитель (/) | logic | 7399 | 11549 | 0 |
| | dsp | 775 | 7875 | 55 |
| Умножитель-делитель | logic | 8512 | 13282 | 0 |
| | dsp | 892 | 9057 | 64 |
| Операция сравнения на равенство с плавающей точкой | | | | |
| Равенство (=) | | 14 | 10 | 0 |
| Неравенство (≠) | | 14 | 10 | 0 |
| Равенство -неравенство ($^{=}/_{\neq}$) | | 14 | 10 | 0 |
| Операция сравнения «больше» с плавающей точкой | | | | |
| Больше либо равно (≥) или меньше либо равно (≤) | | 33 | 15 | 0 |
| Меньше (<), больше (>) | | 32 | 14 | 0 |
| Больше либо равно -меньше ($^{\geq}/_{<}$), Меньше либо равно -больше ($^{\leq}/_{>}$) | | 33 | 15 | 0 |

Данные в табл. 3.1-3.3 получены на основе схемотехнической реализации устройств для ПЛИС семейства Virtex UltraScale в среде

схемотехнического проектирования Vivado. Приведенные значения получены в результате схемотехнического моделирования реализации библиотечных элементов с помощью блоков DSP (для 32-разрядного делителя с фиксированной и плавающей точкой) и/или логических ячеек (для 64-разрядных вычитателя, сумматора-вычитателя и умножителя с плавающей точкой).

Реализация кадровой структуры в виде м-кадра возможна для целевой архитектуры ПЛИС гибридной вычислительной системы, а ее эффективность зависит от типа совмещаемых в одном устройстве операций и разрядности обрабатываемых данных. Эффективность преобразования кадровой структуры к м-кадру можно количественно оценить с помощью сокращения аппаратного ресурса, необходимого для его реализации. Согласно приведенным в табл. 3.1-3.3 данным, при объединении операций в одном устройстве аппаратные затраты сокращаются примерно в 1,3-1,8 раза в зависимости от формата представления данных и выполняемой математической операции. Сокращение аппаратных затрат зависит также от структуры компонентов архитектуры ПЛИС и для разных семейств (например, Xilinx Virtex-7, UltraScale, UltraScale+ и т.д) может отличаться.

Критерием целесообразности редукции по числу устройств и переходу к м-кадрам можно считать следующий: накладные расходы на коммутацию потоков данных не должны превышать аппаратных затрат на реализацию вычислений

$$A_{Com} \leq A_{CS}.$$

Обозначим $Q$ число функциональных устройств в исходной кадровой структуре, а $A_{ФУ-ср}$ – среднее для рассматриваемой кадровой структуры значение аппаратного ресурса, занимаемого одним устройством.

В результате редукции по числу устройств с коэффициентом $R$, число устройств в м-кадре сократится до величины $\left\lceil \frac{Q}{R} \right\rceil$, а аппаратный ресурс для реализации вычислений можно оценить как

$$A_{CS} = \left\lceil \frac{Q}{R} \right\rceil \cdot A_{ФУ-ср}.$$

При редукции по числу устройств на оба входа устройств м-кадра добавляются R-входовые коммутаторы (мультиплексоры), поэтому затрачиваемый на их реализацию ресурс для м-кадра можно оценить как

$$A_{Com} = 2 \cdot \left\lceil \frac{Q}{R} \right\rceil \cdot A_{R-Mx},$$

где $A_{R-Mx}$ – ресурс на реализацию одного R-входового мультиплексора, который, в свою очередь, можно оценить как

$$A_{R-Mx} = \frac{R}{2} \cdot A_{Mx},$$

где $A_{Mx}$ – аппаратный ресурс на реализацию 2-входового мультиплексора для обрабатываемой разрядности данных. Тогда

$$A_{Com} = 2 \cdot \left\lceil \frac{Q}{R} \right\rceil \cdot A_{R-Mx} = 2 \cdot \left\lceil \frac{Q}{R} \right\rceil \cdot \frac{R}{2} \cdot A_{Mx} = \left\lceil \frac{Q}{R} \right\rceil \cdot R \cdot A_{Mx}.$$

Подставляя полученные для $A_{CS}$ и $A_{Com}$ выражения в исходное неравенство, получим оценку для коэффициента редукции, при котором целесообразно применение м-кадров:

$$\left\lceil \frac{Q}{R} \right\rceil \cdot R \cdot A_{Mx} \leq \left\lceil \frac{Q}{R} \right\rceil \cdot A_{ФУ-ср}, \; R \leq \frac{A_{ФУ-ср}}{A_{Mx}} \quad . \tag{3.9}$$

Как правило, аппаратный ресурс мультиплексора зависит от разрядности коммутируемых данных и числа входов: прямо пропорционален разрядности и числу входов, деленному на 4:

$$A_{MX}(k) = \frac{\rho \cdot k}{4}, \tag{3.10}$$

где $\rho$ - разрядность коммутируемых данных на входах мультиплексора, $k$ - число входов мультиплексора.

Полученные формулы (3.9-3.10) задают теоретические границы целесообразности применения редукции по числу устройств с помощью м-кадров. Абсолютные значения для коэффициента редукции в формуле (3.9) зависят от семейства ПЛИС и аппаратных затрат на реализацию коммутационных устройств. Данные по занимаемому аппаратному ресурсу в ячейках LUT (Look-UP Table) для рассматриваемых в предыдущих примерах ПЛИС фирмы Xilinx семейств Virtex7 и UltraScale в среде проектирования

Xilinx ISE 14.7 представлены в табл. 3.4. Значения $A_{ФУ-ср}$ были представлены ранее в табл. 3.1-3.3, а значения $A_{Mx}$ приведены в табл. 3.4. Как видно, $A_{Mx}$ реализуется только на LUT, причем на 1 разряд обрабатываемых данных требуется 1 LUT. Многие функциональные элементы, ресурс которых представлен в табл. 3.1-3.3, можно реализовать как с помощью логических элементов LUT, так и с помощью других компонентов целевой архитектуры ПЛИС, например, блоков DSP. В этом случае число используемых для реализации элементов LUT сокращается, и граница применимости м-кадров сдвигается.

Таблица 3.4. Данные по занимаемому аппаратному ресурсу в ячейках LUT (Look-UP Table) для ПЛИС фирмы Xilinx семейств Virtex7 и UltraScale в среде проектирования Xilinx ISE 14.7

| Разрядность коммутируемых входов | Число входов мультиплексора | | | |
|---|---|---|---|---|
| | **2** | **4** | **8** | **16** |
| 8 | 4 | 8 | 16 | 32 |
| 16 | 8 | 16 | 32 | 64 |
| 32 | 16 | 32 | 64 | 128 |
| 64 | 32 | 64 | 128 | 256 |

Оценим некоторые абсолютные значения коэффициента редукции для м-кадров. Аппаратные затраты на реализацию 2-входового мультиплексора, коммутирующего 32-разрядные операнды, составляют 16 LUT. Аппаратные затраты для 32-разрядной операции умножения с фиксированной и плавающей точкой составляют 1094 и 570 LUT, поэтому коэффициент редукции не должен превышать 68 и 35 соответственно (табл. 3.1-3.2). Для более простых устройств, как, например, сумматоры – не должен превышать 2. Общим критерием эффективности редукции по числу устройств $R^{Op}$ можно считать сокращение аппаратных затрат не менее, чем в 1,5 раза, при котором дополнительные накладные расходы на коммутацию и синхронизацию

операндов в м-кадре не превысят 50% от затрат на реализацию вычислительных устройств с объединенными операциями.

Объединение операций в одном устройстве в ряде случаев позволяет достичь еще одной цели – перейти на другой уровень редукционных преобразований при изменении кратности вычислительных операций кадровой структуры, как при объединении операций сложения и вычитания в примере редукции БО БПФ (рис. 3.2). Кратность объединенной операции равна суммарной кратности объединяемых операций, при этом из-за изменения состава и кратности вычислительных устройств кадровой структуры может измениться состав м-кадров $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5$. Поэтому в некоторых случаях объединение операций в одном устройстве позволяет получить м-кадры, более эффективные по занимаемому аппаратному ресурсу и интервалу обработки данных и реализовать, например, м-кадр $\mu_2$.

Можно утверждать, что редукция по числу устройств $R^{Op}$, результатом которой являются м-кадры, нелинейно сокращает аппаратные затраты на реализацию кадровой структуры из-за дополнительных накладных расходов на коммутацию и синхронизацию (коммутационное оборудование и мультиплексоры - рис. 3.2) для управления входными потоками данных совмещенных устройств. Интервал обработки данных и, соответственно, время решения увеличиваются линейно, пропорционально коэффициенту редукции.

### 3.1.4. Редукция по разрядности $R^{\rho}$

Редукция по разрядности $R^{\rho}$ – это механизм движения по оси «Разрядность» в нижнем октанте пространства возможных кадровых структур (рис. 2.6). Редукция по разрядности $R^{\rho}$ сокращает число одновременно обрабатываемых разрядов за счет преобразования параллельной обработки разрядов каждого операнда в параллельно-последовательную (в предельном случае – в последовательную) обработку, что позволяет сократить аппаратные

затраты на реализацию вычислительного устройства. Редукция по разрядности $R^{\rho}$ для каждого информационного входа формирует кортежи элементов разрядности $\frac{\rho}{R^{\rho}}$, конкатенация которых образует входные данные. Интервал обработки данных в редуцированной по разрядности кадровой структуре увеличивается кратно значению коэффициента редукции $R^{\rho}$, а сокращение аппаратных затрат является нелинейной функцией, зависящей как от формата представления данных (логический, целочисленный, фиксированная точка, плавающая точка), так и от выполняемой операции. Редукция по разрядности $R^{\rho}$ для логических и целочисленных данных в формате с фиксированной точкой применима для всего диапазона возможных значений коэффициента редукции. Для данных в формате с плавающей точкой эффективным значением коэффициента редукции будет 2 для 32-разрядных и 2,4 для 64-разрядных данных из-за раздельной обработки разрядов мантиссы и порядка. Для стандартных арифметических операций следует использовать коэффициент редукции, кратный $2^q$, а для коэффициентов, некратных степени 2, необходимо использовать вычислительные блоки с одноразрядной обработкой, что может увеличить аппаратные затраты на коммутацию. Комплексный анализ метода редукции производительности и аппаратных затрат базового подграфа представлен в работах [185, 186], а применение при решении практических прикладных задач описано в работах [181-184].

Редукция по разрядности $R^{\rho}$ имеет важное свойство: одновременно с увеличением интервала обработки она позволяет увеличить число внешних логических каналов кратно значению коэффициента редукции и без изменения их физического числа, что в ряде случаев необходимо, особенно с учетом существенного недостатка числа внешних каналов ПЛИС (см. главу 1).

### 3.1.5. Редукции по интервалу обработки данных $R^{I}$ и частоте $R^{Freq}$

Редукция по интервалу обработки данных (скважности) $R^{I}$ – это механизм движения по оси «Время/Интервал» пространства возможных

кадровых структур (см. рис. 2.6). Редукция по интервалу обработки данных $R^I$ , в отличие от остальных видов редукционных преобразований, не сокращает, а кратно коэффициенту редукции увеличивает интервал обработки входных данных и время решения задачи. При редукции $R^I$ меняются не аппаратные затраты, а характеристики интенсивности потока данных. Особенностью увеличения интервала обработки данных в редуцированной кадровой структуре является распространение его влияния на все информационно-связанные фрагменты, что приводит к кратному увеличению времени решения. На первый взгляд, применение редукции по интервалу $R^I$ нецелесообразно, т.к. аппаратные затраты не сокращаются, а время решения увеличивается. Однако, редукция $R^I$ важна для согласования темпа обработки потоков данных между фрагментами кадровой структуры с различными интервалами обработки данных, полученными в результате применения других редукционных преобразований к кадровой структуре. Чтобы избежать применения редукции по интервалу обработки данных в ряде случаев можно сократить интервала обработки данных кадровой структурой с помощью оптимизационных преобразований (см. П1 Приложения 1) К числу таких преобразований относятся построение конвейера в конвейере [178], построение макроконвейера [179] и автоподстановка [180]. Каждое из этих преобразований подробно рассмотрено в параграфе П1 Приложения 1.

Построение конвейера в конвейере [178] объединяет несколько изоморфных информационно-независимых конвейерных структур с обратными связями в одну структуру с регистрами в цепи обратной связи. Число объединяемых конвейерных структур и число регистров в цепи обратной связи объединенной структуры совпадают и численно равны латентности $lat$ исходных конвейерных структур. В результате такого преобразования в $lat$ раз сократится число устройств и входных каналов, а данные на вход конвейера в конвейере (рис. П1.5-б) будут подаваться плотным потоком с интервалом $I = 1$. Время обработки данных не изменится, а

освободившийся в результате преобразования аппаратный ресурс и каналы можно использовать для рациональной реализации других фрагментов задачи.

Для структур с процедурной, а не конвейерной обработкой данных, преобразование к конвейеру в конвейере неприменимо, но для сокращения интервала обработки данных можно использовать макроконвейерную обработку данных, предложенную советским математиком и кибернетиком В.М. Глушковым[179]. Суть макроконвейера состоит в реализации нескольких процедурных устройств таким образом, чтобы при подаче входных данных плотным потоком с интервалом $I = 1$ всегда было хотя бы одно свободное устройство, способное обработать поступившие данные (рис. П1.6). Число реализуемых процедурных устройств, достаточное для обработки данных плотным потоком с интервалом $I = 1$, определяется максимальным количеством тактов работы процедурного устройства.

Автоподстановка [180] сокращает интервал обработки данных для конвейерных вычислительных структур с обратной связью за счет «раскрытия» или «развертки» обратной связи при наличии дополнительного аппаратного ресурса и последующей оптимизации полученной структуры для сокращения интервала и времени обработки данных.

Метод построения конвейера в конвейере характеризуется не только сокращением интервала обработки данных, но и сокращением аппаратных затрат, поэтому он наиболее целесообразен для оптимизации кадровой структуры. Построение макроконвейера для процедурных блоков и применение автоподстановки для конвейерных структур требуют дополнительного свободного аппаратного ресурса, что ограничивает их применимость.

Редукция по частоте $R^{Freq}$ сокращает частоту обработки входных данных и применима в тех же случаях, что и редукция по скважности $R^{I}$ для согласования темпа обработки данных в разных фрагментах кадровой структуры.

## 3.2. Теоретические основы применения редукций для преобразования кадровой структуры

Согласно пятому принципу теоретических основ ресурсонезависимого программирования для преобразования кадровой структуры к рациональному для целевых архитектур ГВС варианту организации параллельных вычислений с помощью механизма редукций необходимо разработать правила и методику совместного применения редукционных преобразований разных типов, учитывающие информационную структуру задач. Методы редукции производительности и аппаратных затрат, с помощью которых выполняется движение в пространстве возможных реализаций, являются механизмом преобразования кадровой структуры к реализуемому на целевых архитектурах ГВС варианту организации параллельных вычислений, корректное использование которого обеспечивает информационную эквивалентность исходной и редуцированной кадровых структур. Помимо информационной эквивалентности, согласно пятому и шестому принципам ресурсонезависимого программирования, методы преобразования задачи должны обеспечивать эффективное и согласованное решение задачи с учетом ее информационной структуры. Для этого необходимо разработать методику достижения общего коэффициента редукции кадровой структуры с помощью комплекса применяемых редукций, определить их взаимное влияние на эффективность получаемого результата и оценить необходимое число выполняемых преобразований.

Все рассмотренные редукционные преобразования влияют на различные характеристики кадровой структуры, поэтому их можно считать ортогональными друг к другу, т.е. редукция по разрядности не изменяет число слоев, итераций или устройств, как и редукция по числу итераций не влияет на разрядность или число слоев редуцируемой кадровой структуры. В результате рассмотрения редукционных преобразований, можно объединить

все виды редукций и их влияние на редуцируемую кадровую структуру в табл. 3.5.

Таблица 3.5. Влияние редукций разных типов с коэффициентом $R$ на характеристики кадровой структуры

| **Типы редукций** | **Число каналов** | **Аппаратный ресурс** | **Интервал обработки** | **Разрядность** | **Время решения** |
|---|---|---|---|---|---|
| по числу слоевых подграфов $R^N$ | Линейно сокращается $\propto R$ | Линейно сокращается $\propto R$ | Не меняется | Не меняется | Линейно растет $\propto R$ |
| по числу итерационных подграфов $R^N$ | Не меняется (иногда может сократиться) | Линейно сокращается $\propto R$ | Нелинейно растет $\propto R$ | Не меняется | Нелинейно растет $\propto R$ |
| по числу устройств $R^{Op}$ | Не меняется (сокращается при оптимизации) | Нелинейно сокращается $\propto R$ (с учетом комм. расходов) | Линейно растет $\propto R$ | Не меняется | Нелинейно растет $\propto R$ |
| по разрядности $R^{\rho}$ | Линейно сокращается $\propto R$ | Близко к линейному сокращается $\propto R$ | Линейно растет $\propto R$ | Линейно сокращается $\propto R$ | Линейно растет $\propto R$ |
| по интервалу $R^I$ | Не меняется | Не меняется | Линейно растет $\propto R$ | Не меняется | Линейно растет $\propto R$ |
| по частоте $R^{Freq}$ | Не меняется | Не меняется | Не меняется | Не меняется | Линейно растет $\propto R$ |

При рассмотрении комплексного применения редукционных преобразований к кадровой структуре последовательное их выполнение будем обозначать символом $\circ$. Например, последовательное применение редукций по числу подграфов и по числу устройств будет обозначаться как $R^N \circ R^{Op}$. Как уже отмечалось, при редукции производительности с целью сокращения аппаратных затрат сбалансированно сокращается число аппаратно реализованных подграфов и/или вычислительных устройств и/или разрядности обрабатываемых данных в заданное натуральным коэффициентом редукции число раз до целого значения, которое не может быть меньше единицы. Для описания такого сокращения будем использовать

операцию округления в меньшую сторону [194] до единичного значения, которую для натуральных чисел $a \geq 1, b \geq 1$ определим как

$$\left\lfloor \frac{a}{b} \right\rfloor_1 = \begin{cases} a \operatorname{div} b, a > b, \\ 1, a \leq b, \end{cases}, \tag{3.9}$$

где **div** – операция целочисленного деления, а обозначение $\lfloor\ \rfloor_1$, выбранное по аналогии с символом $\lfloor\ \rfloor$ («пол») из [194], указывает, что значение операции «пол» ограничено снизу единичным значением.

Результат операции округления в меньшую сторону (3.9), ограниченный снизу единичным значением, описывает предел редукции параметров кадровой структуры: числа подграфов, вычислительных устройств и разрядов данных, значение которых не может быть меньше единицы. Операция округления (3.9) имеет важное свойство [194]: для любых вещественных m, x и натурального числа n справедливо равенство

$$\left\lfloor \frac{\left\lfloor \frac{x}{m} \right\rfloor}{n} \right\rfloor = \left\lfloor \frac{x}{m \cdot n} \right\rfloor. \tag{3.10}$$

Поскольку натуральные числа – это подмножество вещественных чисел, а функция $\left\lfloor \frac{a}{b} \right\rfloor_1$ монотонна и непрерывна, то (3.10) справедливо и для нее. Тогда с учетом коммутативности получим

$$\left\lfloor \frac{\left\lfloor \frac{x}{m} \right\rfloor_1}{n} \right\rfloor_1 = \left\lfloor \frac{\left\lfloor \frac{x}{n} \right\rfloor_1}{m} \right\rfloor_1 = \left\lfloor \frac{x}{m \cdot n} \right\rfloor_1. \tag{3.11}$$

При рассмотрении правил движения в пространстве возможных реализаций кадровых структур (см. параграф 2.5) и выбора эффективного шага (см. параграф 2.7), одним из способов достижения границ области доступного аппаратного ресурса гибридной вычислительной системы было последовательное применение редукций одного типа для минимизации критического ресурса. Рассмотрим влияние последовательного применения редукционных преобразований одного типа на итоговое значение коэффициента редукции.

**Теорема 3.1** (о последовательном применении редукционных преобразований)

*Последовательно примененные редукции* $R_m^T$ *и* $R_n^T$ *одного типа* $T$ *с натуральными коэффициентами* $m > 1$ *и* $n > 1$ *эквивалентны редукции* $R_{n \cdot m}^T$ *того же типа с коэффициентом, равным их произведению* $(m \cdot n) > 1$*:*

$$R_n^T \circ R_m^T = R_{n \cdot m}^T.$$

Доказательство теоремы 3.1 приведено в Приложении 1.

Последовательные редукционные преобразования одного типа можно заменить одним преобразованием этого же типа с коэффициентом редукции, равным произведению исходных коэффициентов. Для практического использования последовательных редукционных преобразований важно следствие из теоремы 3.1:

**Следствие 3.1.1 из теоремы 3.1.** Представление коэффициента редукции производительности в виде произведения простых сомножителей позволяет сократить число преобразований для подбора рациональных значений коэффициентов последовательной редукции одного или разных типов. Разложение коэффициента редукции на простые сомножители позволяет выбрать наиболее рациональное значение коэффициента для редукции каждого типа и наоборот – для заданного значения коэффициента редукции возможно определить наиболее подходящий тип редукции с учетом характеристик задачи (3.10).

**Следствие 3.1.2 из теоремы 3.1.** Если коэффициент редукции производительности $R$ – это простое число больше 2, а значение $R$ недостижимо с помощью однократной редукции, то целесообразна редукция не в $R$, а в $(R + 1)$ раз для сокращения аппаратных затрат не менее, чем в $R$ раз. Поскольку в этом случае $(R + 1)$ будет четным составным числом, то возможно последовательное применение редукционных преобразований с коэффициентами редукции из разложения числа $(R + 1)$ на простые сомножители.

**Следствие 3.1.3 из теоремы 3.1.** При кратном увеличении коэффициента редукции при последовательном применении редукции одного типа нет необходимости возвращаться к исходной кадровой структуре. Действительно, если полученная в результате редукции кадровая структура нуждается в дополнительном кратном (не менее, чем вдвое) сокращении аппаратных затрат по критическому ресурсу и для данного типа редукции возможно кратное увеличение ее коэффициента, то согласно теореме 3.1 редукция, примененная последовательно и без возврата к исходной кадровой структуре, сократит число шагов для получения итоговой редуцированной структуры.

**Следствие 3.1.4 из теоремы 3.1.** Из редуцированной с коэффициентом $n$ кадровой структуры с помощью последовательно примененной редукции любого типа нельзя получить редуцированную с коэффициентом $n+1$ кадровую структуру. Согласно теореме 3.1 и следствиям 3.1.1-3.1.3, при последовательно примененных редукциях суммарный коэффициент равен произведению, а не алгебраической сумме коэффициентов редукций. Поэтому из редуцированной с коэффициентом $n$ вычислительной структуры с помощью последовательно примененной редукции любого типа нельзя получить редуцированную с коэффициентом $n+1$ вычислительную структуру, что бывает необходимо при появлении накладных расходов на коммутацию и синхронизацию (например, при редукции по числу устройств и по разрядности).

Это утверждение справедливо и для коэффициентов вида $(n+x)$, и его можно сформулировать в виде следующей теоремы:

**Теорема 3.2** (о неаддитивности последовательных редукционных преобразований)

*Для редуцированной с коэффициентом* $\boldsymbol{n}$ *кадровой структуры с помощью последовательного применения дополнительной редукции любого типа* $\boldsymbol{T}$ *с любым натуральным коэффициентом* $k>1$ *в общем случае*

*невозможно получить структуру с коэффициентом редукции* $(n + x)$ *для любого наперед заданного* $x$, $\forall x \geq 1$, *некратного* $\boldsymbol{n}$:

$$R_n^{T1} \circ R_k^{T2} \neq R_{n+x}^{T}.$$

Доказательство теоремы 3.2 приведено в Приложении 1, а на ее основании можно сформулировать следствие, важное для определения последовательности применения редукционных преобразований.

**Следствие 3.2.1 из теоремы 3.2.** Поскольку при последовательных редукциях любых типов невозможно увеличить коэффициент редукции на произвольную величину для уже редуцированной структуры, то при необходимости дополнительного сокращения аппаратных затрат с помощью редукции этого же типа необходимо возвращаться к исходному подграфу и выполнять редукционные преобразования с новым (увеличенным) коэффициентом редукции $R$. Возврат к исходной кадровой структуре увеличивает общее число преобразований для получения результата, поэтому важно точно рассчитать коэффициент редукции, чтобы минимизировать число возвратов.

Последовательность применения редукций разных типов, которая определяет траекторию движения в пространстве возможных реализаций, не влияет на характеристики редуцированной кадровой структуры и может быть любой.

**Теорема 3.3** (о коммутативности редукционных преобразований)

*Суперпозиция редукций различных типов (T1 с коэффициентом* $\boldsymbol{n}$ *и T2 с коэффициентом* $\boldsymbol{m}$*) коммутативна, т.е. изменение порядка выполнения редукций различных типов T1 и T2 в последовательных редукционных преобразованиях не изменяет редуцированную кадровую структуру.*

$$R_n^{T1} \circ R_m^{T2} = R_m^{T2} \circ R_n^{T1},$$

где $T_1, T_2$ - типы редукционных преобразований,

$n, m$ - коэффициенты редукции.

Доказательство теоремы 3.3 также приведено в Приложении 1. Основное следствие из теоремы 3.3: изменение порядка выполнения

редукционных преобразований разных типов не приводит к разным редуцированным структурам и не требует возврата к исходной кадровой структуре.

Теоремы 3.1-3.3 (и следствия из них) – это теоретическая основа преобразования кадровой структуры с помощью редукционных преобразований к выбранному варианту организации параллельных вычислений.

## 3.3. Правила применения редукций для преобразования кадровой структуры к рациональным для целевых архитектур ГВС вариантам организации параллельных вычислений

На основании доказанных теорем и следствий из них можно сформулировать основные правила преобразования максимально параллельной кадровой структуры к варианту организации параллельных вычислений, реализуемому на доступном аппаратном ресурсе гибридной вычислительной системы.

*1) Для сокращения числа выполняемых редукционных преобразований следует выбирать коэффициенты редукции для каждого типа из разложения общего коэффициента редукции на простые сомножители.*

Для выбора наиболее рационального варианта сочетания редукционных преобразований, обеспечивающих достижение общего коэффициента редукции, из всего многообразия возможных значений коэффициентов разных типов (по числу подграфов $R^N$, по числу устройств $R^{Op}$, по разрядности $R^{\rho}$ и $R^I$ для редукции интервала обработки данных), следует использовать целые сомножители общего коэффициента редукции. Сокращение каждого из параметров кадровой структуры в целое число раз, пропорциональное сомножителям коэффициента редукции $R$, позволяет получить сбалансированную редуцированную структуру без непродуктивных аппаратных затрат на коммутацию и синхронизацию потоков данных и сократить число передвижений в пространстве возможных реализаций.

*2) Если число подграфов в кадровой структуре больше 1, то первым шагом редукционных преобразований следует выполнять редукцию по числу подграфов $R^N$.*

Редукция кадровой структуры по числу подграфов $R^N$ линейно сокращает занимаемый аппаратный ресурс по числу логических ячеек ПЛИС и по числу каналов (для слоевых подграфов), а также линейно увеличивает время решения задачи (табл. 3.1). Она не требует дополнительных коммутационных расходов, как редукции по числу устройств $R^{Op}$ и разрядности $R^{\rho}$. Поэтому ее следует использовать для наибольшего сокращения аппаратных затрат без дополнительных накладных расходов.

*3) При выполнении редукции по числу вычислительных устройств и разрядности для сокращения числа редукционных преобразований следует проводить их до предельного значения, выбранного с учетом сомножителей коэффициента редукции, после чего переходить к редукции другого типа.*

Редукции по числу устройств $R^{Op}$ и разрядности $R^{\rho}$ могут приводить к дополнительным накладным аппаратным расходам на коммутационное оборудование, что увеличит общие аппаратные затраты на реализацию редуцированной кадровой структуры. В результате, они не будут удовлетворять заданному коэффициентом редукции значению и потребуют дополнительные преобразования по сокращению аппаратных затрат для достижения области доступного аппаратного ресурса в пространстве возможных реализаций. Поэтому следует выполнять эти редукции до наиболее «удобных» значений, обеспечивающих рациональное сочетание дополнительных аппаратных затрат и производительности вычислительной структуры.

*4) Для определения начального значения коэффициента редукции производительности $R$ кадровой структуры следует использовать коэффициент сокращения аппаратных затрат $R_T$, округленный к большему целому.*

Коэффициент необходимого сокращения аппаратных затрат $R_T$ определяется как отношение аппаратного ресурса $A_T$, требуемого для реализации кадровой структуры, к доступному ресурсу вычислительной системы $A_{RCS}$ по аналогии с (3.14) для расчета критического ресурса.

Начальное значение коэффициента редукции производительности $R_0$ для максимально параллельной кадровой структуры определяется как округление коэффициента $R_T$ к ближайшему целому вверх:$R_0 = \lceil R_T \rceil$. Для линейных вычислительных структур (например, задач символьной обработки и задач линейной алгебры для итерационной вычислительной структуры) значения коэффициента сокращения аппаратного ресурса $R_T$ и коэффициента редукции производительности $R_0$ могут совпадать. В большинстве случаев редукция производительности с коэффициентом, равным коэффициенту сокращения аппаратных затрат, может потребовать дополнительного увеличения коэффициента редукции производительности из-за непредвиденных коммутационных расходов.

Начальное значение коэффициента редукции производительности $R_0 > 1$ согласно основной теореме арифметики следует представить в виде произведения простых сомножителей:

$$R_0 = \prod_i r_{0i}. \tag{3.12}$$

С помощью полученного разложения на простые множители и с учетом параметров кадровой структуры (число подграфов и операций, разрядность обрабатываемых данных) коэффициент редукции производительности $R_0$ для кадровой структуры нужно представить в виде произведения трех сомножителей, т.е. коэффициентов редукции по числу подграфов $R_0^N$, по числу устройств $R_0^{Op}$ и по разрядности $R_0^{\rho}$:

$$R_0 = R_0^N \cdot R_0^{Op} \cdot R_0^{\rho}. \tag{3.13}$$

Если $R_0$ - простое число, то, согласно следствию 3.1.1, его значение увеличится на 1 и будет представлено в виде разложения на множители.

Оценим число редукционных преобразований для получения редуцированной кадровой структуры, реализуемой в области доступного аппаратного ресурса. Рассмотрим общий случай, когда для сокращения аппаратных затрат необходимо выполнить все виды рассмотренных редукционных преобразований, т.е. по числу подграфов, по числу устройств и по разрядности. В этом случае все коэффициенты редукции: $R_0^N$ (по числу подграфов), $R_0^{Op}$ (по числу вычислительных устройств), $R_0^{\rho}$ (по разрядности данных) должны быть больше 1. Согласно второму принципу

сформулированных правил, первым преобразованием будет редукция производительности по числу подграфов с коэффициентом $R_N^0$. Вторым преобразованием будет редукция по числу вычислительных устройств с коэффициентом $R_{Op}^0$. В общем случае при сокращении числа устройств можно синтезировать не менее пяти различных м-кадров (см. пункт 3.1.3) с различными аппаратными затратами и интервалом обработки данных, среди которых нужно выбрать наиболее эффективный по критерию интервала обработки данных (времени решения). На третьем этапе для каждого м-кадра, полученного после редукции по числу вычислительных устройств, будет выполнена редукция по разрядности с коэффициентом $R_\rho^0$. При этом число возможных вариантов редукции по разрядности для наиболее распространенных типов данных не превышает двух:

- при редукции по разрядности логических и целочисленных данных (с фиксированной точкой) сокращение аппаратных затрат линейно пропорционально коэффициенту редукции, поэтому редукция будет выполнена в заданное коэффициентом $R_0^\rho$ число раз, не превышающее разрядности обрабатываемых данных;

- при редукции данных в формате с плавающей точкой число возможных значений коэффициентов редукции для 32-разрядных данных равно одному (двукратная редукция по разрядности), а для 64-разрядных данных равно двум (двух- и четырехкратная редукция по разрядности), что обусловлено экспоненциальным ростом накладных расходов на обработку мантиссы и порядка для других значений коэффициентов редукции по разрядности.

Таким образом, после редукции по разрядности число вариантов м-кадров оценивается как $5 \cdot 2 = 10$, для каждого из которых необходимо проанализировать занимаемый аппаратный ресурс и интервал обработки данных, определяющий время решения прикладной задачи. В ряде случаев, если аппаратные затраты итоговой редуцированной кадровой структуры задачи $A_T^1$ превышают доступный аппаратный ресурс РВС $A_{RCS}$, может потребоваться дополнительный, четвертый этап преобразований. Такая ситуация возможна из-за появления дополнительных коммутационных расходов при редукциях по числу вычислительных устройств и по разрядности для данных в формате с плавающей точкой, т.к. сокращение

аппаратных затрат в этих случаях нелинейно. При редукции по числу вычислительных устройств не всегда возможно точно определить итоговое сокращение аппаратных затрат до выполнения преобразования, поэтому значения коэффициентов $R_0^{Op}$ и $R_0^{\rho}$ после редукции могут потребовать увеличения.

После выполнения всех редукционных преобразований оценивается сокращение аппаратных затрат, достигнутое для кадровой структуры. Возможны два варианта. Редуцированная кадровая структура достигла области доступного аппаратного ресурса и реализуема в архитектуре вычислительной системы, либо требуется дополнительное сокращение аппаратных затрат из-за непредвиденного роста накладных расходов. В первом случае редукция для преобразования максимально параллельной кадровой структуры к реализуемому в ГВС варианту организации параллельных вычислений завершилась не более, чем за 4 этапа с анализом не более 10 вариантов. Во втором случае необходимо вернуться к исходной максимально параллельной кадровой структуре (согласно теореме 3.2) и выполнить редукцию производительности (шаги 1-3) с увеличенным коэффициентом $R_1 = R_0 + 1$, либо, если это возможно, выполнить кратную редукцию в 2 или более раз по одному из параметров. Очевидно, что во втором случае при повторной редукции суммарное число проанализированных вариантов удвоится и не превысит 20. Даже с учетом наличия в структуре задачи нескольких фрагментов и необходимости их взаимного согласования с помощью редукций по частоте и интервалу обработки данных, общее число выполненных преобразований будет сравнительно невелико. На практике большинство прикладных задач содержат не более 3-5 различных фрагментов, поэтому общее число вариантов редукционных преобразований таких задач вряд ли превысит 60. Полученная оценка числа выполненных преобразований максимально параллельной кадровой структуры к реализуемому в ГВС варианту организации параллельных вычислений существенно меньше сопоставимого показателя, т.е. числа анализируемых вариантов распараллеливания последовательной программы для МВС с распределенной памятью, который для $n$ узлов, согласно теореме Кэли, оценивается числом различных деревьев для $n$ вершин, которое составляет $n^{n-2}$ (см. параграф 1.4).

Редукция производительности и аппаратных затрат прикладной задачи выполняется с общим для всех подзадач коэффициентом редукции, что обеспечивает сбалансированность редуцированной кадровой структуры. Для реконфигурируемых узлов гибридной вычислительной системы возможна редукция производительности кадровой структуры по таким параметрам, как число устройств, разрядность и интервал обработки данных, что недоступно для процессорных узлов и графических ускорителей. Поэтому, в отличие от других архитектур, для ПЛИС общий коэффициент редукции каждой подзадачи представим в виде уникального набора произведения редукционных коэффициентов (по числу подграфов, устройств, разрядности и интервалу). Возможность использования для каждой подзадачи собственной комбинации редукционных коэффициентов позволяет гибко учитывать параметры подзадач, выбирая наиболее рациональные коэффициенты редукционных преобразований, и сократить вариативность редуцированных вычислительных структур. Эта особенность ПЛИС существенно сокращает как число анализируемых вариантов, так и время преобразования кадровой структуры к рациональному варианту организации параллельных вычислений.

## 3.4. Методика преобразования кадровой структуры к рациональному варианту организации параллельных вычислений

На основе разработанных правил применения редукционных преобразований к кадровой структуре (см. параграф 3.3) можно создать методику преобразования кадровых структур к рациональному варианту организации параллельных вычислений.

Рассмотрим методику преобразования кадровой структуры $CS$ с аппаратными затратами $A_{CS} = (a_1^{CS}, a_2^{CS}, \dots, a_n^{CS})$ и параметрами быстродействия $P_{CS} = (p_1, p_2, \dots, p_n)$ к аппаратному ресурсу реконфигурируемой или гибридной вычислительной системы $A_{HCS} = (a_1^{HCS}, a_2^{HCS}, \dots, a_n^{HCS})$. Отметим, что аппаратные затраты $A_{CS}$ кадровой структуры нелинейно зависят от параметров быстродействия $P_{CS}$: $A_{CS} = \Psi(P_{CS})$, причем один параметр быстродействия $p_i$ может влиять одновременно на несколько характеристик аппаратного ресурса. Цель портации – найти параметры быстродействия $P'_{CS}$, которые будут являться решением системы

$$\begin{cases} A'_{CS} = \Psi(P'_{CS}) \leq A_{HCS} \\ Perf(P'_{CS}) \geq Perf_з \end{cases},$$

где $Perf(P'_{CS})$ – производительность кадровой структуры, а $Perf_з$ – заданный уровень реальной производительности.

Для этого необходимо выполнить следующие действия:

1. **Определение параметров редукции**

1.1. Определить критический ресурс и коэффициент сокращения аппаратных затрат $K_{cr} = max\left(\frac{a_i^{CS}}{a_i^{HCS}}\right) > 1$.

1.2. Определить коэффициент редукции $R = \lceil K_{cr} \rceil$.

1.3. Упорядочить параметры быстродействия $p_i$ в кортеж $< P_{CS} >$ по величине аппаратных затрат критического ресурса $\psi(p_i)$.

1.4. **Если** <есть нередуцированный параметр в $< P_{CS} >$>, **то**

- выбрать для редукции параметр быстродействия $p_i$ с максимальным значением $\psi(p_i)$

**иначе**: перейти к п.7 для выбора кадровой структуры с наибольшей производительностью.

2. **Определение эффективного шага редукции** $R^*$.

2.1. Для выбранного параметра быстродействия $p_i$ определить эффективный шаг редукции $R^*$ из возможных вариантов:

1) $R^* = R$, {если достичь $R$ можно сокращением параметра $p_i$ при $p_i \geq R$ };

2) $R^* = f(R, p_i), R^* < R$ {если редукция по параметру $p_i$ эффективна до значения $f(R, p_i)$, после которого не сокращаются аппаратные затраты}

3) $R^* = \|p_i\|, R^* > R$, {если редукция эффективна только для предельного или минимального значения параметра $\|p_i\|$, которое в ряде случаев является единственно возможным, когда один из параметров намного больше другого $R \gg p_i$ или $p_i \gg R$}

3. **Редукция (сокращение параметров быстродействия) кадровой структуры**

3.1. Сократить параметр $p_i$ с эффективным шагом $R^*$: $p'_i = \Theta(p_i, R^*)$ и сохранить его в кортеж: $< P'_{CS} >=< P_{CS} > \leftarrow p'_i$

4. **Оценка текущей кадровой структуры и выбор альтернатив**

4.1. Оценить аппаратные затраты $A'_{CS} = \Psi(P'_{CS})$ и производительность $Perf(P'_{CS})$ кадровой структуры после редукции.

4.2. Оценить достижение цели портации по условиям $A'_{CS} \leq A_{HCS}$ и $Perf(P'_{CS}) \geq Perf_з$.

4.3. **Если** оба условия выполняются {портация успешна}, **то**

- перейти к п.6 для сохранения текущих параметров кадровой структуры

**иначе:** {Выбор альтернатив дальнейших действий}

**Если** $A'_{CS} > A_{HCS}$ {аппаратные затраты структуры превышают доступный ресурс}, **то**:

**Если** $R^* = R$ {коэффициента $R$ не хватило для сокращения аппаратных затрат}, **то**

- увеличить коэффициент редукции $R \coloneqq R + \Delta$, где $\Delta = \left\lceil \frac{A'_{CS}}{A_{HCS}} - 1 \right\rceil$;

- восстановить исходное значение параметра быстродействия $< P'_{CS} > = < P_{CS} > \leftarrow p_i$;

- перейти к п.1.4 для редукции с увеличенным коэффициентом R.

**Иначе** {редукции по $p_i$ было недостаточно для сокращения аппаратных затрат}

- перейти к п.1 для перерасчета и дальнейшего сокращения критического ресурса следующим параметром быстродействия $p_{i+1}$.

{если $A'_{CS} \leq A_{HCS}$ и $Perf(P'_{CS}) < Perf_з$, произойдет естественный переход к индукции}

**5. Индукция кадровой структуры (увеличение сокращенных параметров быстродействия) при** $A'_{CS} \leq A_{HCS}$ и $Perf(P'_{CS}) < Perf_з$

5.1. Сформировать кортеж ранее сокращенных параметров $< p_{i-1}, \dots p_1 >$.

5.2. Выбрать очередной элемент кортежа $p_k$, определить для него коэффициент масштабирования по аппаратному ресурсу $K_M = min\left(\frac{A'_{CS}}{A_{HCS}}\right)$ и коэффициент индукции $Ind = \lfloor K_M \rfloor$.

5.3. Для выбранного параметра быстродействия $p_k$ определить эффективный шаг индукции $Ind^*$ из возможных вариантов:

1) $Ind^* = Ind$ {индукция достижима полным или частичным увеличением $p_k$};

2) $Ind^* = \varphi(Ind, p_k) < Ind$, {индукция эффективна для определенного значения параметра $p_k$ }

3) $Ind^* = \|p_k\|$, {индукция эффективна только для предельного или максимального значения параметра $\|p_k\|$, которое в ряде случаев является единственно возможным}

5.4. Увеличить параметр $p_k$ с выбранным шагом $Ind^*$: $p'_k = \Theta^{-1}(p_k, Ind^*)$ и сохранить его в кортеж: $< P'_{CS} >=< P'_{CS} > \leftarrow p'_k$

5.5. Перейти к п.4 для оценки достижения цели портации.

**6. Сохранение параметров кадровой структуры**

6.1. Добавить параметры быстродействия кадровой структуры в список, упорядоченный по минимуму производительности.

6.2. **Если** $Perf(P'_{CS}) < Perf_з$ **и** <были другие критические ресурсы>, **то**

- выбрать следующий критический ресурс и перейти к п.1.1.{для поиска, возможно, более рациональной структуры по другому критическому ресурсу}

**7. Выбор наиболее рационального варианта портации**

7.1. Выдать первый элемент списка кадровых структур с наибольшей производительностью $Perf(P'_{CS})$.

В отличие от ранее известных методов масштабирования структурно-процедурных вычислений (технологии индуктивных программ и редукции производительности частных задач), в представленной методике процесс преобразования непрерывный, обеспечивающий непрерывное движение к заданным показателям аппаратного ресурса и производительности с помощью формальных преобразований как для увеличения, так и для сокращения параллелизма. Целью методики является последовательное достижение сначала локальной цели $A'_{CS} \leq A_{HCS}$, а потом – глобальной $Perf(P'_{CS}) \geq Perf_з$. Если глобальная цель недостижима, в результате будет выдано реализуемое на целевых архитектурах ГВС, наиболее близкое к заданному, решение. Причины, исключающие достижение глобальной цели, могут быть различными, например:

– если $Perf_з$ равна пиковой производительности вычислительной системы;

– специфика редукционных преобразований, обусловленная свойствами целочисленного деления: редукция по разрядности 32-разрядных данных с коэффициентом $R = 17$ будет выполнена с эффективным шагом $R^* = 32$, т.к. реализация других вариантов затруднительна и требует дополнительных накладных расходов аппаратного ресурса на синхронизацию;

– для ряда кадровых структур, как правило, с непосредственной информационной зависимостью, соответствующей последовательному выполнению операций даже в максимально параллельной форме, эффективный шаг редукции соответствует переходу к процедурной, т.е. последовательной, организации вычислений. В этом случае локальная цель $A'_{CS} \leq A_{HCS}$ будет достигнута, а для достижения глобальной цели (заданной производительности или сокращения времени), полученное решение должно быть распараллелено с помощью индукции.

Индукция кадровой структуры в п.5 может рассматриваться как возврат к кадровой структуре до выполненной редукции и выполнение редукции с меньшим шагом $R \coloneqq R - \Delta$, но из-за нелинейной зависимости аппаратных затрат от параметров быстродействия формализация такого преобразования будет сложнее рассмотренного варианта.

Возврат к п.1 и п.4 методики необходим для перерасчета критического ресурса, т.к. редукция одного параметра быстродействия может привести к сокращению других из-за нелинейной зависимости аппаратных затрат.

Разложение коэффициента редукции $R > 1$ на простые множители и параметров быстродействия кадровой структуры $P_{CS} = (p_1, p_2, \ldots, p_n)$ согласно теоремам 3.1 и 3.3 и следствиям из них выполняется для удобства поиска корней диофантова неравенства $R_{p_1} \cdot R_{p_2} \cdot \ldots \cdot R_{p_n} \geq R$ при определении эффективного шага редукции $R^*$.

Предложенная обобщенная методика преобразования кадровой структуры абстрагируется от параметров архитектуры вычислительной системы, поэтому может применяться для всех рассматриваемых целевых архитектур и их комбинаций. Общее количество анализируемых вариантов и «осцилляций» (этапов редукции и индукции) зависит от числа принимаемых во внимание параметров быстродействия кадровой структуры, которые влияют на аппаратные затраты в целевой архитектуре. Оценка числа

вариантов «сверху» не превышает $P!$, где $P = \|P_{CS}\|$ – количество параметров быстродействия кадровой структуры, влияющих на аппаратные затраты. Для целевых архитектур процессоров и графических ускорителей число параметров минимально, т.к. обычно они позволяют изменять только число каналов $Ch$ и количество вычислительных узлов $CPU/GPU$. Набольшее количество варьируемых параметров, влияющих ан аппаратные затраты, у целевой архитектуры ПЛИС $A_{FPGA} = (Ch^{FPGA}, LUT, BRAM, HBM, DSP, FF)$, где $Ch_{FPGA}$ – количество каналов, $LUT$ – число ячеек **L**ook-**U**p **T**able в кристалле ПЛИС, $BRAM$ – количество блоков внутренней памяти **B**lock **RAM**, $HBM$ – число блоков высокоскоростной памяти **H**igh **B**andwidth **M**emory, $DSP$ – количество блоков цифровой обработки сигналов **D**igital **S**ignal **P**rocessor, $FF$ – число регистров **F**lip-**F**lop. Все упомянутые параметры целевой архитектуры ПЛИС крайне редко бывают критическими для реальных задач, обычно их не более 3. Даже для всех 6 параметров общее число вариантов на превысит 720, что сравнительно немного, особенно с учетом возможной автоматизации их анализа.

Поэтому в диссертации разработаны ряд алгоритмов преобразования кадровой структуры для наиболее сложной среди рассматриваемых целевой архитектуры ПЛИС, обладающей наибольшим количеством параметров быстродействия $P_{CS} = \{L_i^{it}, It_i^{it}, Ch_i^{it}, Op_i^{it}, \rho_i^{it}, g_i\}$: $L_i^{it}$ – число информационно-независимых подграфов в слое $F_{ij}$, $It_i^{it}$ – число итераций в функции $\Phi_i$, $Ch_i^{it}$-число внешних каналов кадровой структуры, $Op_i^{it}$ – число устройств кадровой структуры $FG^{it}$, $\rho_i^{it}$ – разрядность обрабатываемых данных в аппаратной реализации. Общее число подграфов $N_i^{it} = L_i^{it} \cdot It_i^{it}$ представляется произведением числа итерационных $It_i^{it}$ и слоевых $L_i^{it}$ подграфов.

Рассмотрим применение обобщенной методики на примере преобразования функционально-регулярной максимально параллельной кадровой структуры $FG^{it} = \{L_i^{it}, It_i^{it}, Ch_i^{it}, Q_i^{it}, \rho_i^{it}, g_i\}$ к архитектуре гибридной вычислительной системы с коэффициентом редукции $R > 1$. Диапазон допустимых целочисленных значений всех редукционных коэффициентов (кроме $R_i^I$) ограничен параметрами кадровой структуры с дискретным целочисленным шагом:

- для коэффициента редукции по слоям $R_i^L = 1..L_i^{it}$;

- для коэффициента редукции по итерациям $R_i^{It} = 1..It_i^{it}$;

- для коэффициента редукции по числу устройств $R_i^Q = 1..Q_i^{it}$;

- для коэффициента редукции по разрядности $R_i^\rho = 1..\rho_i^{it}$.

Произведение числа итерационных $It_i^{it}$ и слоевых $L_i^{it}$ подграфов составляет общее число подграфов $N_i^{it} = L_i^{it} \cdot It_i^{it}$.

Блок-схема алгоритма преобразования кадровой структуры с коэффициентом $R = R_{ch}^{it} \cdot R_{cell}^{it} > 1$ согласно приведенной методике представлена на рис. 3.6. Входными данными алгоритма являются характеристики кадровой структуры $FG^{it} = \{It_i^{it}, L_i^{it}, Ch_i^{it}, Q_i^{it}, \rho_i^{it}, g_i\}$ и значение коэффициента редукции $R_{ch}^{it}$.

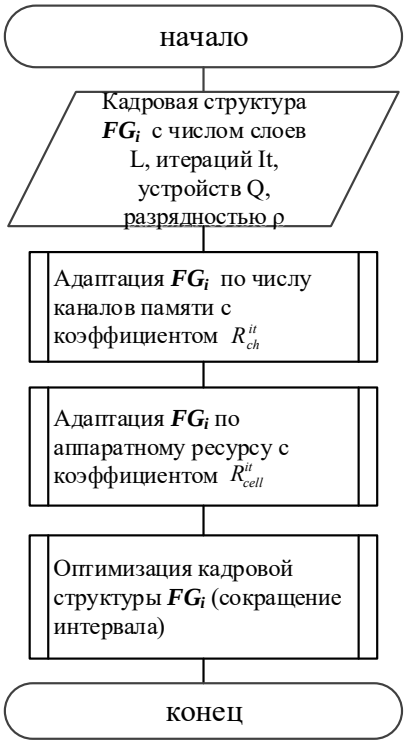



Рис. 3.6. Обобщенная блок-схема алгоритма преобразования кадровой структуры $FG^{it} = \{It_i^{it}, L_i^{it}, Ch_i^{it}, Q_i^{it}, \rho_i^{it}, g_i\}$

Алгоритм в общем случае состоит из 4 этапов, на каждом из которых для сокращения числа каналов и достижения области доступного аппаратного ресурса редуцируется один из параметров текущей кадровой структуры. Основным и наиболее результативным способом сокращения числа каналов является сокращение числа слоевых подграфов. Одновременно с сокращением требуемого для реализации кадровой структуры числа каналов памяти, в

результате редукции по числу подграфов происходит сокращение и аппаратных затрат. Поэтому для задач ряда предметных областей, например, для задач символьной обработки, возможна ситуация, когда после сокращения числа каналов может быть достигнута область доступного аппаратного ресурса и не требуется дальнейшее преобразование кадровой структуры. Блок-схема алгоритма преобразования кадровой структуры $FG^{it}$ по числу каналов памяти с коэффициентом $R_{ch}^{it}$, являющегося первым этапом преобразования задачи согласно методике и блок-схеме (рис. 3.6), приведена на рис. 3.7.

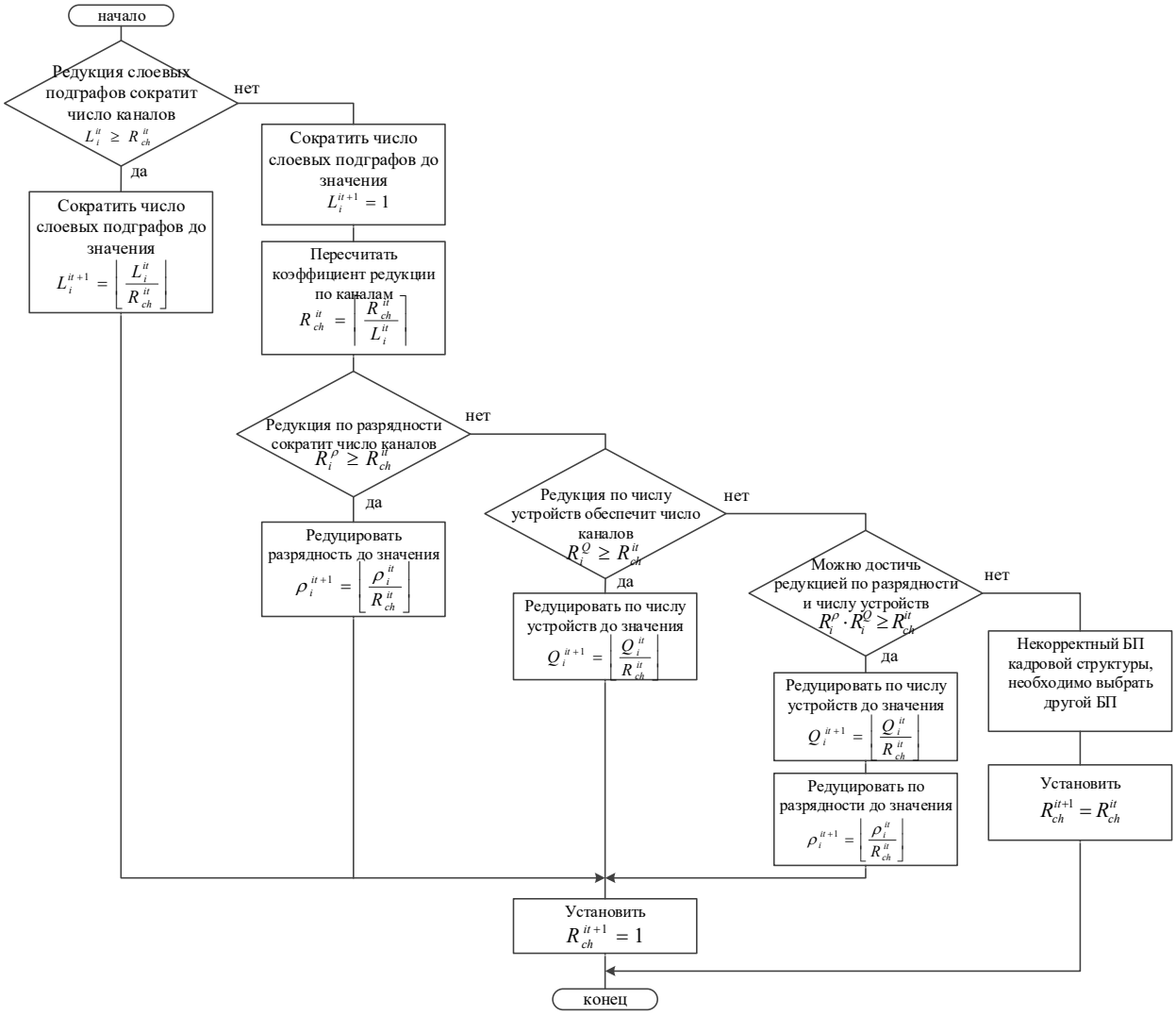



Рис. 3.7. Блок-схема алгоритма преобразования кадровой структуры по числу каналов памяти

Если сокращение числа слоевых подграфов не обеспечивает числа каналов, достаточного для реализации текущей кадровой структуры, из-за информационных зависимостей или особенностей алгоритма решаемой задачи, т.е. $L_i^{it} < R_{ch}^{it}$, то число слоевых подграфов сокращается до минимума,

а дальнейшее сокращение числа каналов выполняется с помощью редукций по разрядности и числу устройств. Для этого коэффициент редукции по каналам необходимо представить произведением числа слоевых подграфов $L_i^{it}$ и коэффициента дополнительной редукции по числу устройств или разрядности $R_{\text{доп}}^{it}$: $R_{ch}^{it} = L_i^{it} \cdot R_{доп}^{it}$.

Для дополнительной редукции по каналам с коэффициентом $R_{\text{доп}}^{it}$ можно использовать как редукцию по числу устройств, так и редукцию по разрядности. Если можно достичь коэффициента $R_{доп}^{it}$ только редукцией по разрядности, то она более предпочтительна, чем редукция по числу устройств. Редукция по разрядности линейно и пропорционально коэффициенту редукции сокращает число каналов памяти и так же линейно, с тем же коэффициентом увеличивает интервал обработки данных текущей кадровой структуры, что позволяет достаточно точно оценить параметры результирующей кадровой структуры.

Редукция по числу устройств также может линейно сократить число каналов и увеличить интервал обработки данных, но сокращение аппаратных затрат происходит нелинейно. Поэтому применение редукции по разрядности более предпочтительно, если только с ее помощью достижим коэффициент $R_{доп}^{it}$. В редких случаях, когда для сокращения каналов требуется совместное применение редукции по разрядности и числу устройств следует сначала выполнить редукцию по числу устройств, после чего редуцировать разрядность обрабатываемых кадровой структурой данных. Согласно теореме 3.3 производительность кадровой структуры не зависит от порядка применения редукционных преобразований, поэтому производительность вычислительной структуры не изменится при изменении последовательности выполнения редукций. Однако, порядок редукций важен для дальнейшей оптимизации, сокращающей интервал обработки данных.

После успешно выполненного масштабирования по каналам коэффициент редукции по каналам $R_{ch}^{it+1}$ должен быть равен 1. Если после редукций по числу слоевых подграфов, числу устройств и разрядности область доступного аппаратного ресурса по числу каналов памяти не достигнута, т.е. не выполняется последнее условие в блок-схеме алгоритма (рис. 3.6), это свидетельствует об ошибке при исходном формировании кадровой структуры

или в значении коэффициента редукции по каналам. Для целочисленных данных такая ситуация невозможна из-за наличия в пространстве реализаций минимума, т.е. последовательной поразрядной обработки. Для данных с плавающей запятой такая ситуация также маловероятна, т.к. предельная редукция по числу устройств позволяет сократить число каналов до одного.

Преобразование кадровой структуры к рациональному варианту организации параллельных вычислений по числу каналов памяти – это важнейшее преобразование методики, после выполнения которого в пространстве возможных реализаций достигается первая точка выбора альтернатив (см. параграф 2.6). В зависимости от комплекса примененных редукционных преобразований, параметров полученной кадровой структуры и текущего критического ресурса, в точке выбора альтернатив необходимо выбрать один из алгоритмов дальнейшего преобразования кадровой структуры для достижения области доступного аппаратного ресурса, т.е. алгоритм преобразования структурно-процедурной реализации, алгоритм преобразования на основе м-кадров или алгоритм преобразования процедурной реализации.

## 3.5. Алгоритмы преобразования кадровой структуры к рациональному варианту организации параллельных вычислений на доступном аппаратном ресурсе гибридной вычислительной системы

После успешной редукции по каналам с помощью информационно-независимых слоевых подграфов и достижения точки выбора альтернатив, для дальнейшего сокращения аппаратных затрат с коэффициентом $R_{cell}^{it}$ кадровой структуры $FG^{it} = \{It_i^{it}, L_i^{it}, Ch_i^{it}, Q_i^{it}, \rho_i^{it}, g_i\}$ при движении к области доступного ресурса нужно выбрать новое направление движения из рациональных решений диофантова неравенства $R_i^L \cdot R_i^{It} \cdot R_i^Q \cdot R_i^I \geq R_{cell}^{it}$ для $FG^{it}$.

### 3.5.1 Алгоритм преобразования кадровой структуры к структурной реализации вычислений

При преобразовании кадровой структуры по числу каналов памяти в предыдущем алгоритме, число слоевых подграфов было сокращено для

достижения области доступного аппаратного ресурса. Если при преобразовании прикладной задачи по числу каналов выполнялась редукция только по числу слоевых подграфов, без редукций по числу устройств и разрядности, то точка выбора альтернатив будет находиться в верхнем октанте, что соответствует структурной реализации $g_{стр}$ базового подграфа кадровой структуры $FG_i$. Для текущей кадровой структуры $FG_i$ необходимо вычислить критический ресурс, по которому определить направление дальнейшего движения к области доступного ресурса с помощью редукционных преобразований. Если число слоевых подграфов сократилось не до единичного значения, то для дальнейшего преобразования можно снова использовать редукцию по числу подграфов, максимально сокращающую аппаратный ресурс кадровой структуры. Блок-схема алгоритма преобразования кадровой структуры $FG_i$ к структурной реализации с помощью редукции по числу подграфов с коэффициентом $R_{cell}^{it}$ показана на рис. 3.8.

При наличии слоевых подграфов, для дальнейшего преобразования следует сокращать число итерационных, а не слоевых подграфов для получения кадровой структуры, удобной для построения «конвейера в конвейере» (Приложение 1).

Поэтому первый этап алгоритма (рис. 3.8) – это сокращение именно итерационных подграфов. Если итерационных подграфов недостаточно для достижения коэффициента редукции, то проверяется возможность одновременного сокращения слоевых и итерационных подграфов для достижения области доступного аппаратного ресурса.

Число итерационных подграфов сокращается до единичного значения, а число слоевых подграфов – в зависимости от абсолютных значений числа слоев и коэффициента редукции для построения «конвейера в конвейере». Поэтому второе условие (рис. 3.8) – это проверка нахождения точки выбора альтернатив в верхнем или нижнем октанте. Если условие выполняется, то текущая кадровая структура редуцируется до мультиконвейерной или конвейерной реализации $g_{стр}$, которую можно оптимизировать в «конвейер в конвейере». Если условие не выполняется и коэффициент редукции недостижим путем сокращения только слоев и итераций, то выполняется

переход в нижний октант пространства возможных реализаций с синтезом м-кадров.

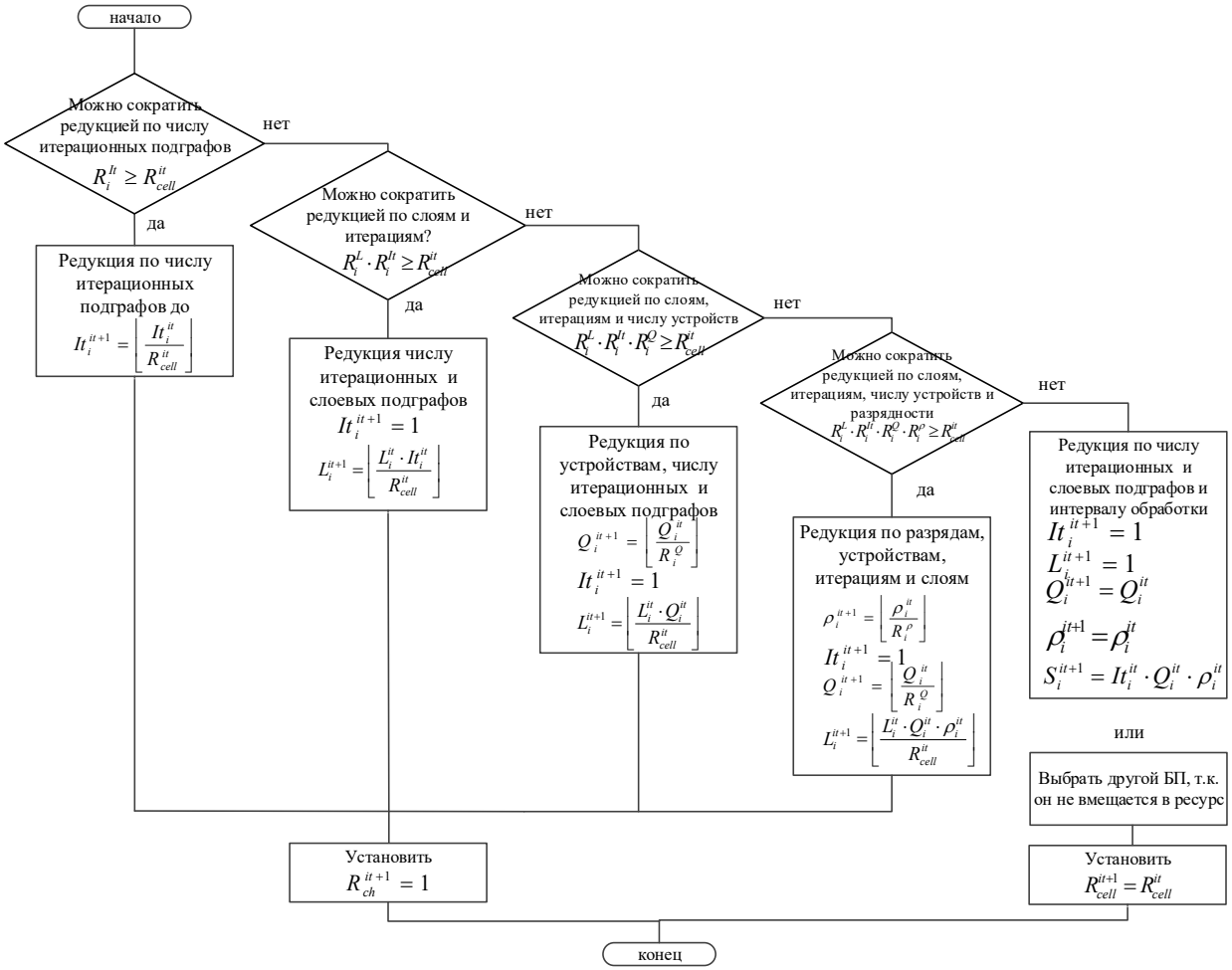



Рис. 3.8. Блок-схема алгоритма преобразования кадровой структуры к структурной реализации

Для этого в дополнение к редукции слоевых и итерационных подграфов используется редукция по числу устройств, а если недостаточно и ее, то и редукция по разрядности. Как и в предыдущем случае, при преобразовании по числу каналов, если после выполнения всех редукций область доступного аппаратного ресурса не достигнута, это свидетельствует об ошибке при формировании исходной кадровой структуры или некорректном значении коэффициента редукции.

После выполнения алгоритма преобразования кадровой структуры к структурной реализации (рис. 3.8), будет достигнута очередная точка выбора альтернатив, требующая перерасчета критического ресурса. В зависимости от полученного значения, выполняется либо дальнейшее сокращение

аппаратных затрат (если накладные коммутационные расходы превысили ограничения доступного ресурса), либо оптимизация полученной кадровой структуры для сокращения интервала обработки данных.

### 3.5.2 Алгоритм преобразования кадровой структуры к процедурной реализации вычислений

Процедурная реализация $g_{проц}$ при преобразовании кадровой структуры по числу каналов памяти достигается в результате перехода в нижний октант пространства возможных реализаций при использовании редукций по числу устройств и разрядности. Поскольку точка выбора альтернатив находится в нижнем октанте, то аппаратный ресурс текущей кадровой структуры находится ниже $g_{\text{стр}}$ и, как правило, область доступного аппаратного ресурса уже достигнута. Однако, в ряде случаев требуется дальнейшее сокращение аппаратного ресурса кадровой структуры, преобразованной к м-кадрам или представленной в виде полностью процедурной реализации, но характеризующейся наличием слоевых и итерационных подграфов $g_{проц}$. В этом случае, естественной целью является преобразование кадровой структуры таким образом, чтобы можно было на следующем этапе оптимизировать ее в макроконвейер (см. Приложение 1). Блок-схема алгоритма преобразования с коэффициентом $R_{cell}^{it}$ для масштабирования кадровой структуры $FG_i$ с процедурной реализацией $g_{\text{проц}}$ приведена на рис. 3.9. Отличием алгоритма (рис. 3.9) от предыдущих является отсутствие редукции по числу устройств среди применяемых преобразований, поскольку $g_{проц}$ изначально подразумевает процедурную обработку. Хотя, если реализация $g_{проц}$ не является предельным случаем, соответствующим последовательной процессорной реализации, а выполнена с помощью м-кадров, в алгоритм (рис. 3.9) можно добавить предельную редукцию по числу устройств. Как и в пункте 3.5.1, выполнение алгоритма преобразования кадровой структуры к процедурной реализации (рис. 3.9) означает достижение очередной точки выбора альтернатив и необходимость пересчета критического ресурса.

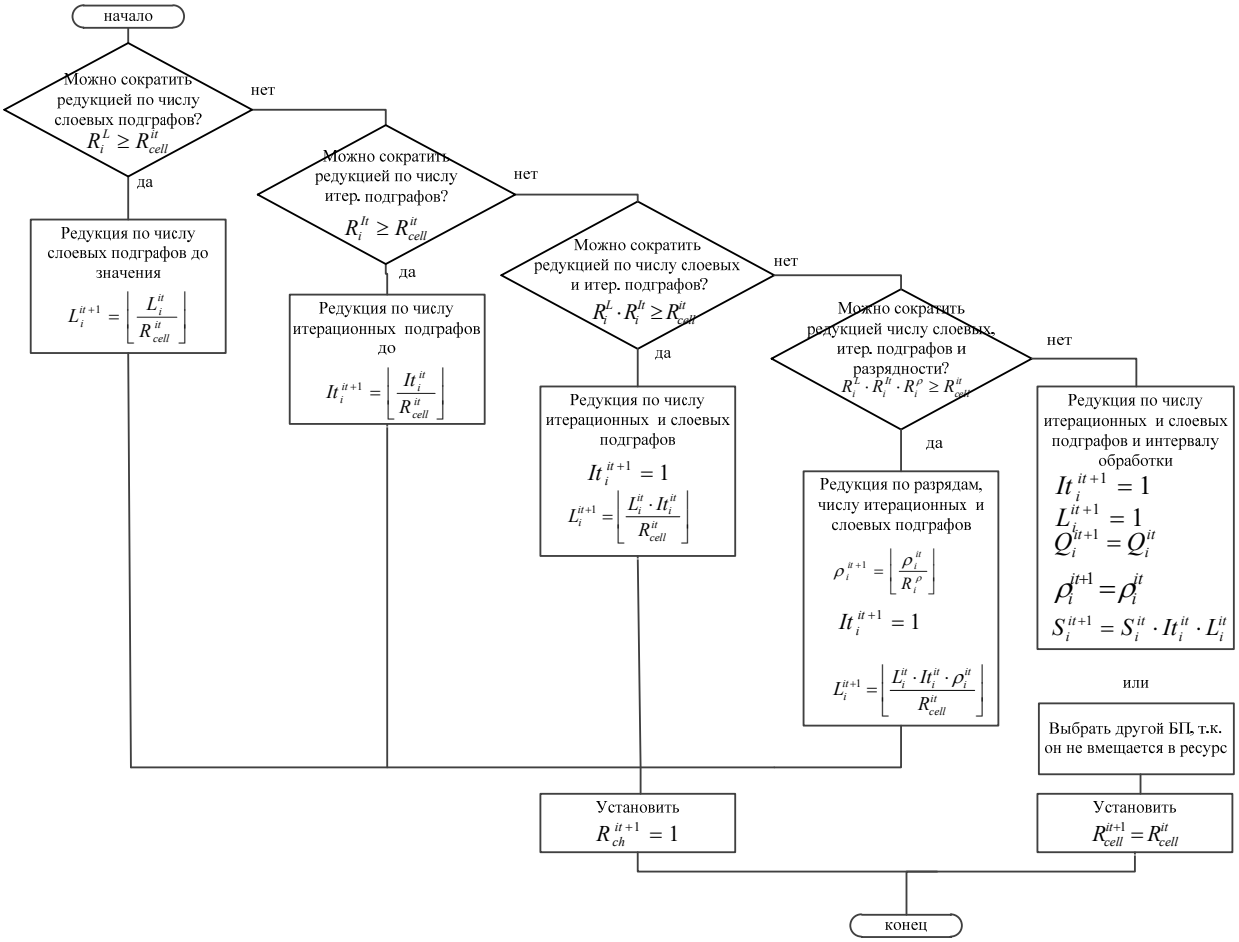



Рис. 3.9. Блок-схема алгоритма преобразования кадровой структуры к процедурной реализации $g_{проц}$

Как правило, после редукции кадровой структуры $g_{проц}$ не требуется дальнейшее сокращение аппаратных затрат. Поэтому после этого выполняется переход к решению второй подзадачи преобразования, т.е. к оптимизации полученной кадровой структуры для сокращения интервала обработки данных.

## 3.6. Выводы к главе 3

1) Механизмом преобразования кадровой структуры, обеспечивающим ее сбалансированность и информационную эквивалентность результатов вычислений при портации задачи, могут служить редукционные преобразования:

- редукция по числу подграфов $R^N$, сокращающая число реализуемых в кадровой структуре подграфов ИГЗ;

- редукция по числу устройств $R^{Op}$, сокращающая число одновременно работающих устройств кадровой структуры совмещением операций в одном устройстве;

- редукция по разрядности $R^{\rho}$, сокращающая число одновременно обрабатываемых разрядов за счет преобразования параллельной обработки разрядов каждого операнда в параллельно-последовательную;

- редукция по интервалу $R^{I}$, увеличивающая интервал (скважность) обработки/подачи данных без изменения аппаратных затрат;

- редукция по частоте $R^{Freq}$, сокращающая частоту работы кадровой структуры.

2) Для редукции по числу устройств $R^{Op}$ предложена новая форма организации вычислений – микро-кадр, выполняющий вычисления на ограниченном доступном аппаратном ресурсе целевых архитектур с меньшей степенью параллелизма. Основное отличие м-кадра от кадровой структуры – совмещение нескольких операций в одном вычислительном устройстве, приводящее к увеличению интервала обработки данных, но обеспечивающее единый и сбалансированный темп обработки данных без использования дополнительной памяти для хранения промежуточных результатов обработки входного потока данных.

3) Предложены методы формирования м-кадров и получены теоретические оценки их применимости при редукции по числу устройств $R^{Op}$ в зависимости от значения коэффициента редукции, приведены абсолютные значения коэффициента редукции для ПЛИС Xilinx семейства Virtex UltraScale.

4) Сформулированы и доказаны теоремы о комплексном применении редукционных преобразований при портации кадровой структуры в целевые архитектуры ГВС. Определены основные свойства комплексного применения редукционных преобразований:

- последовательная редукция одного типа с разными коэффициентами эквивалентна редукции с коэффициентом, равным их произведению;

- изменение порядка выполнения редукционных преобразований различных типов не влияет на результат, т.е. последовательная редукция коммутативна;

- общий коэффициент редукции при последовательной редукции неаддитивен, т.е. он не равен сумме коэффициентов примененных редукционных преобразований.

5) Доказанные теоремы и их следствия являются теоретической основой формализованных правил портации максимально параллельной кадровой структуры задачи к целевым архитектурам ГВС. Оценено число редукционных преобразований кадровой структуры, выполняемых для достижения локальной (области доступного аппаратного ресурса) и глобальной целей портации (рационального варианта организации параллельных вычислений на доступном вычислительном ресурсе). Показано, что число преобразований, выполняемых при портации кадровой структуры в целевые архитектуры, существенно меньше числа вариантов, анализируемого распараллеливающим компилятором для систем с распределенной памятью, что доказывает третье положение, выдвигаемое для защиты.

6) Предложена методика преобразования кадровых структур, основанная на следствиях из доказанных теорем, позволяющая выполнить портацию прикладной задачи с помощью автоматизированных программных средств. В отличие от ранее известных методов, методика обеспечивает непрерывное достижение заданных показателей аппаратного ресурса и производительности кадровой структуры с помощью формальных преобразований как увеличения, так и сокращения параллелизма.

7) Разработаны обобщенный алгоритм преобразования кадровой структуры и алгоритм преобразования к структурной (для ПЛИС и графических ускорителей) и процедурной (для процессоров) реализации при портации к целевым архитектурам ГВС. Разработан алгоритм редукции функционально-регулярной кадровой структуры для сокращения аппаратного ресурса и числа каналов памяти, а также алгоритмы последующей оптимизации полученных структур для достижения глобальной цели портации.

# 4. МЕТОДЫ И АЛГОРИТМЫ ПРЕОБРАЗОВАНИЯ ПРИКЛАДНЫХ ЗАДАЧ С ИНФОРМАЦИОННО-ЗАВИСИМЫМИ ПОДЗАДАЧАМИ

## 4.1. Принципы преобразования информационно-зависимых подзадач

В предыдущей главе при разработке методики преобразования кадровой структуры прикладной задачи к рациональному варианту организации параллельных вычислений рассматривались функционально-регулярные информационные графы задач с единственным базовым подграфом $g_{ijk}$. Задачи именно с такой структурой, т.е. содержащие одну подзадачу, успешно реализовывались методами структурно-процедурной организации вычислений и масштабировались с помощью технологии индуктивных программ. К ним относятся блочные алгоритмы символьной обработки DES, AES, ГОСТ, хеш-функции MD5, SHA и др., алгоритмы линейной алгебры – решения СЛАУ методом Якоби и методом Гаусса-Зейделя, алгоритм БПФ для цифровой обработки сигналов и др. Задачи с другой структурой, например, содержащие нерегулярные информационные обмены и/или несколько подзадач со своим «базовым подграфом» ${g^s}_{ijk}$, всегда реализовывались только прикладным программистом, создававшим сбалансированное решение из нескольких кадровых структур для заранее заданных параметров задачи и фиксированного аппаратного ресурса РВС с помощью искусного применения комплекса редукционных и оптимизационных преобразований. Автоматическое масштабирование таких задач не рассматривалось даже теоретически, т.к. требовало сложного согласования интервала обработки данных в кадровых структурах и выходило за рамки технологии индуктивных программ.

С развитием методов структурно-процедурной организации вычислений и расширением списка успешно решенных задач стало очевидно, что большинство прикладных задач из различных областей применения РВС обычно содержат не одну, а несколько информационно-зависимых подзадач. Характерные примеры

таких задач – это составные алгоритмы символьной обработки на основе композиции хеш-функций и блочных алгоритмов шифрования, задачи линейной алгебры (решение СЛАУ методом Гаусса, содержащее прямой и обратный ход [168, 169]), цифровой обработки сигналов (расчет свертки сигналов), молекулярного моделирования (докинг ингибиторов), томографического исследования Земли и многие другие. Подзадача обычно выделена в информационной структуре задачи на логическом уровне, в последовательной программе описывается процедурой, функцией или отдельной группой циклов, а информационные зависимости между ними задаются с помощью общих переменных.

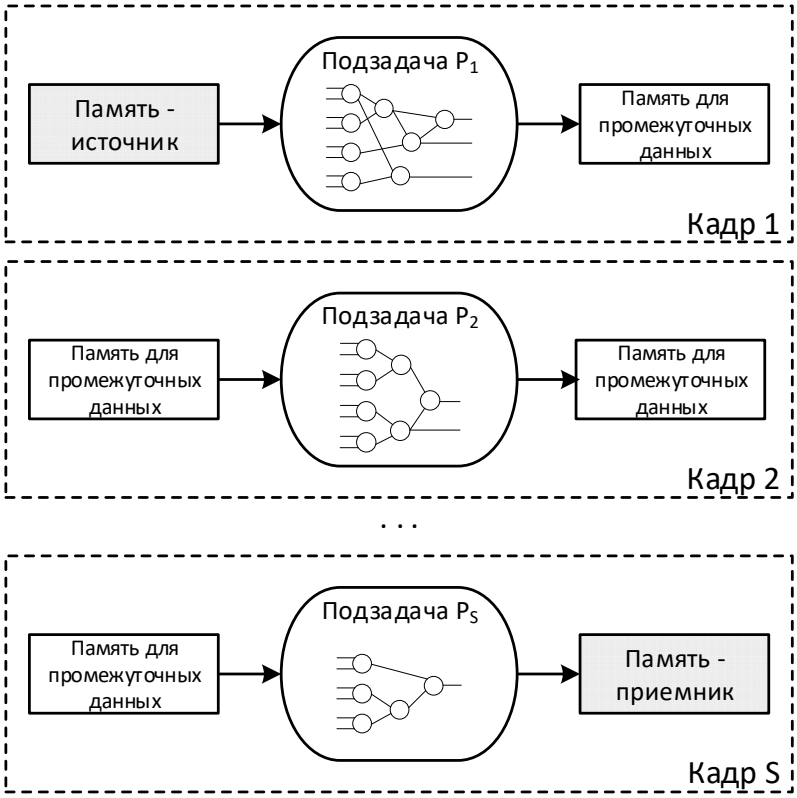
В информационном графе $s$-й подзадаче соответствует подграф (фрагмент), для которого можно выделить свой «базовый подграф» ${g^s}_{ijk}$ и функции отображения слоев ${F^s}_{ij}$ и итераций ${\Phi^s}_i$. Несмотря на разную степень параллелизма каждой подзадачи, разные подграфы ${g^s}_{ijk}$, функции отображения слоев ${F^s}_{ij}$ и итераций ${\Phi^s}_i$, вычисления в ИГЗ и соответствующей ему кадровой структуре будут согласованы за счет максимального параллелизма, при котором одновременно обрабатываются все входные данные задачи. Преобразование максимально параллельной кадровой структуры к рациональному для доступного вычислительного ресурса варианту организации параллельных вычислений сокращает степень параллелизма всего решения и каждой подзадачи в отдельности за счет сокращения числа одновременно реализованных подграфов и вычислительных устройств в каждой подзадаче. Информационная зависимость между подзадачами сохраняется, но в каждой из них при сокращении параллелизма формируется своя очередность обработки данных, которая определяет собственный поток данных. Поскольку каждая подзадача обладает своей степенью параллелизма, собственным числом слоев и итераций, то при сокращении параллелизма максимально параллельной кадровой структуры характеристики потоков данных (интервал обработки и длина) в разных подзадачах будут различаться, в ряде случаев – существенно. Поэтому

кадровая структура задачи становится совокупностью кадровых «подструктур» подзадач – каждая со своим «базовым подграфом» $g^s{}_{ijk}$, числом слоев $L^s{}_i$ и итераций $It^s{}_i$, числом устройств $Q^s{}_i$, разрядностью $\rho^s{}_i$ и интервалом обработки данных $I^s{}_i$ ($s = 1..\boldsymbol{S}$, где $\boldsymbol{S}$ – число подзадач). Несмотря на то, что эти подструктуры характеризуют подзадачи, а не задачу, то, не нарушая общности, далее будем называть их кадровыми структурами. Для рациональной реализации задачи с информационно-зависимыми подзадачами при преобразовании ее кадровых структур необходимо согласовать характеристики потоков данных между ними.

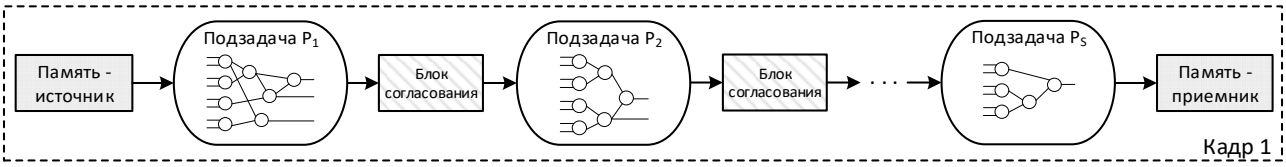
Согласование информационно-зависимых вычислительных фрагментов с разной степенью параллелизма при структурно-процедурной организации вычислений в общем случае выполнялось разбиением на кадровые структуры. Каждая кадровая структура считывала данные из памяти, обрабатывала поток данных и записывала промежуточные результаты обработки в память (рис. 4.1-а). Согласно процедурно заданной очередности смены структур, следующая кадровая структура считывала из памяти сохраненные предыдущей структурой промежуточные результаты, обрабатывала их по своему алгоритму и записывала данные в память (рис. 4.1-а).

Число кадровых структур определяется числом подзадач $\boldsymbol{S}$, а число операций чтения и записи промежуточных результатов составляет $\boldsymbol{S} - \boldsymbol{1}$. При такой организации вычислительного процесса согласование интервалов обработки данных в информационно-зависимых кадровых структурах не требовалось, т.к. результаты записывались в промежуточную память и читались из нее же. Интервал обработки данных в каждом кадре определялся только вычислительной структурой кадра и не зависел от предыдущих или последующих кадров. Информационные зависимости между подзадачами реализовывались структурно-процедурно: потоки данных через вычислительные структуры разных кадров потенциально могли различаться, но последовательность смены кадровых структур была задана процедурно.

Недостаток этого подхода в том, что общий поток данных разрывается при смене кадров, а время решения задачи увеличивается из-за записи-чтения промежуточных результатов обработки всего потока данных. Для потока данных постоянной длины N время решения задачи увеличится как минимум на $N \cdot (\boldsymbol{S} - \mathbf{1})$ тактов для записи промежуточных результатов в память.




а)




б)

Рис. 4.1. Организация вычислений для информационно-зависимых подзадач
а) структурно-процедурная реализация информационных зависимостей,
б) структурная реализация информационных зависимостей

В отличие от структурно-процедурной реализации информационных зависимостей для преобразования информационно-зависимых подзадач

предлагается синтезировать единую кадровую структуру (рис. 4.1-б), содержащую подзадачи, согласованные между собой по степени параллелизма и интервалу обработки данных. Это позволит создать рациональный вариант организации параллельных вычислений для всех информационно-зависимых подзадач, т.е. конвейерную вычислительную структуру всей задачи, работающую без разрывов потока данных памятью. Основная сложность решения этой задачи состоит в расчете параметров блоков согласования (рис. 4.1-б). В зависимости от согласуемых подзадач, блоки согласования БС могут существенно различаться как по своей структуре, так и по характеристикам, но должны согласованно работать в структуре задачи. Необходимость согласования степени параллелизма и интервала обработки данных информационно-зависимых подзадач обусловлена тем, что достижение общего для всех коэффициента редукции в каждой подзадаче может быть получено редукционными преобразованиями, различными как по составу, так и по абсолютным значениям коэффициентов. Различия в используемых редукционных преобразованиях могут приводить к разным значениям степени параллелизма и интервала обработки данных подзадачи, что требует их согласования между собой при организации вычислений в едином контуре (рис. 4.1-б). Эти различия могут потребовать дополнительных преобразований и аппаратных затрат для согласования и синхронизации потоков данных между кадровыми структурами.

Разработанная в предыдущей главе методика преобразования кадровых структур не учитывает информационную связь между подзадачами, поэтому для информационно-зависимых подзадач ее необходимо адаптировать:

1. Разработать принципы и правила применения методов редукции производительности для согласованного выбора параметров параллелизма и интервала обработки данных в информационно-зависимых кадровых структурах.

2. Разработать принципы и правила выделения каждой подзадаче части доступного аппаратного ресурса целевых архитектур гибридной вычислительной системы для преобразования информационно-зависимых кадровых структур.

По аналогии с базовым подграфом в структурно-процедурной организации вычислений, для описания подзадач введем понятие функционального подграфа (ФП) $g_i$, под которым будем понимать фрагмент (подграф) информационного графа (под)задачи с выделенными функциями отображения по слоям $F_{ij}$ и итерациям $\Phi_i$. ФП в максимально параллельной форме соответствует кадровая структура с числом слоев $L^s{}_i$, итераций $It^s{}_i$, числом устройств $Q^s{}_i$, разрядностью $\rho^s{}_i$ и интервалом обработки данных $I^s{}_i$ ($s = 1..$число подзадач). По сути, ФП – это аналог рассмотренного в предыдущих главах базового подграфа, с той разницей, что он характеризует кадровую структуру подзадачи, а не всей задачи. Для задач, содержащих одну подзадачу, функциональный подграф и базовый подграф совпадают. Функциональный подграф подзадачи соответствует группе циклов последовательной программы на алгоритмическом языке программирования, причем тело цикла определяет состав и связи устройств ФП $g_i$, а число итераций, информационные зависимости между переменными и индексация массивов данных задают функции отображения слоев $F_{ij}$ и итераций $\Phi_i$. Если подзадача содержит несколько ФП, то, по аналогии с принятыми для структурно-процедурной организации вычислений понятиями, будем считать, что композиция подграфов всех ФП подзадачи образует минимальный базовый подграф (МБП) подзадачи, позволяющий реализовать вычисления подзадачи структурно. Пример кадровой структуры задачи, содержащей несколько подзадач с разными ФП, представлен на рис. 4.2. Подзадачи $P_1$ и $P_S$ содержат несколько функциональных подграфов (ФП 1.1, ФП 1.2 и ФП S.1, ФП S.2, соответственно), с разным числом слоев, итераций и устройств (рис. 4.2). В остальном структура подзадачи с несколькими ФП аналогична структуре задачи с информационно-зависимыми подзадачами. Для структурной реализации

задачи нужно реализовать вычислительные структуры, содержащие МБП $P_1$, $P_2$, … $P_S$.

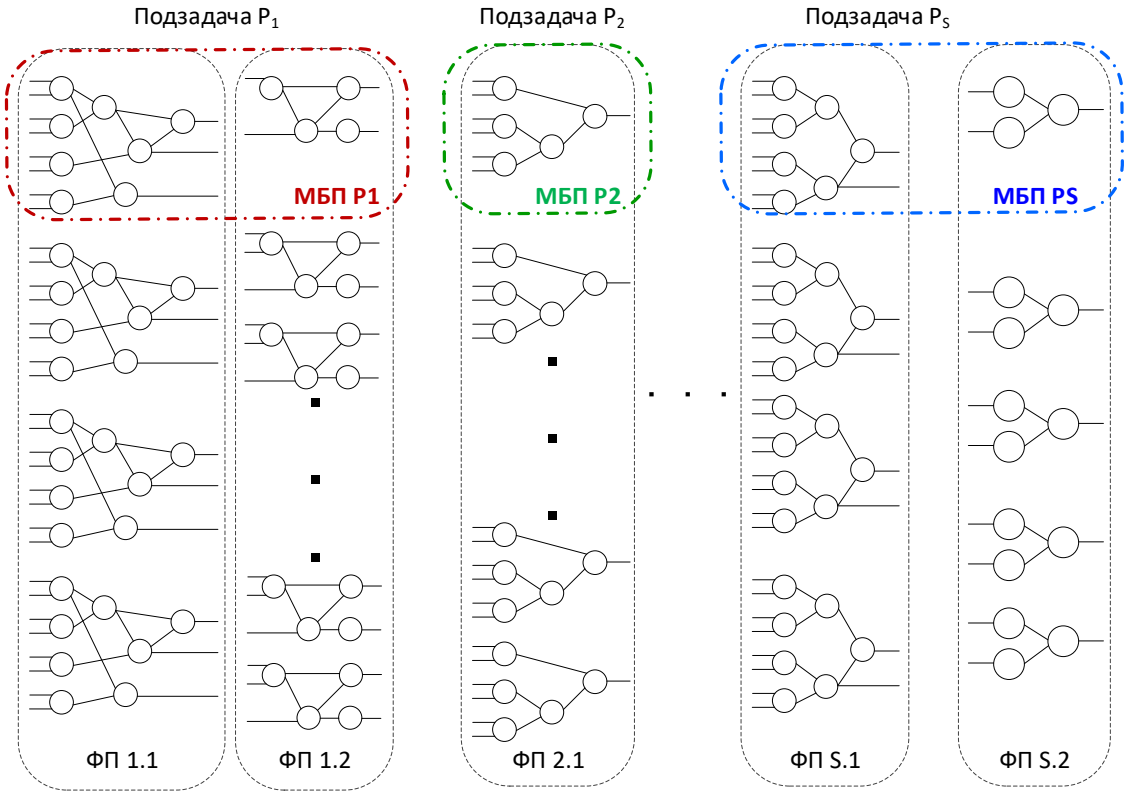



Рис. 4.2. Максимально-параллельная кадровая структура задачи с информационно зависимыми подзадачами

В результате редукции (рис. 4.2) максимально параллельной кадровой структуры задачи с некоторым коэффициентом R, в новой структуре (рис. 4.3) между информационно-зависимыми кадровыми структурами из-за разной степени параллелизма необходим блок согласования интервала (БСИ) обработки данных и других параметров параллелизма кадровых структур, обеспечивающий непрерывный и сбалансированный поток данных между подзадачами.

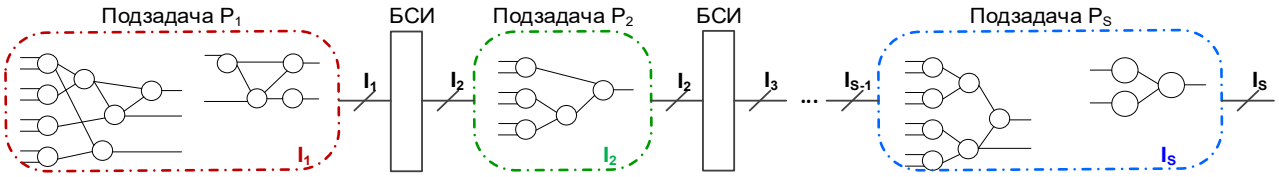



Рис. 4.3. Результат преобразования информационно-зависимых кадровых структур

Определим принципы согласованного преобразования информационно-зависимых кадровых структур. Будем рассматривать максимально параллельную кадровую структуру задачи, содержащую счетное множество информационно-зависимых кадровых структур, каждая из которых описывается функциональным подграфом ${g^s}_{ijk}$ с числом слоев ${L^s}_i$, итераций ${It^s}_i$, числом устройств ${Q^s}_i$, разрядностью ${\rho^s}_i$ и интервалом обработки данных ${I^s}_i$, где $s = 1..\boldsymbol{S}$. Как уже отмечалось в предыдущей главе, преобразование максимально параллельной кадровой структуры к целевым архитектурам гибридной вычислительной системы выполняется редукцией производительности с одинаковым для всех подзадач коэффициентом редукции $R$. Коэффициент редукции $R$ можно представить произведением сомножителей, каждый из которых характеризует редукцию одного из рассмотренных типов – число слоевых и итерационных подграфов, число устройств, разрядность, интервал обработки данных. В каждой подзадаче в зависимости от ее параметров параллелизма коэффициенты редукций разных типов могут варьироваться, что приводит к различным параметрам параллелизма в редуцированных кадровых структурах. Согласование параметров параллелизма и интервала обработки данных для всех кадровых структур соответствует поиску решения системы диофантовых неравенств вида

$$\begin{cases} R_1^L \cdot R_1^{It} \cdot R_1^Q \cdot R_1^\rho \cdot R_1^I \geq R \\ R_2^L \cdot R_2^{It} \cdot R_2^Q \cdot R_2^\rho \cdot R_2^I \geq R \\ \dots \\ R_s^L \cdot R_s^{It} \cdot R_s^Q \cdot R_s^\rho \cdot R_s^I \geq R \end{cases}, \tag{4.1}$$

где $R_s^L$ – коэффициент редукции по числу слоевых подграфов подзадачи $s$,

$R_s^{It}$ – коэффициент редукции по числу итерационных подграфов подзадачи $s$,

$R_s^Q$ - коэффициент редукции по числу устройств подзадачи $s$;

$R_s^\rho$ - коэффициент редукции по разрядности подзадачи $s$;

$R_s^I$ - коэффициент редукции по интервалу подзадачи $s$.

Каждое из неравенств системы (4.1) задает локальную цель преобразования одной подзадачи – пропорциональное сокращение ее производительности и аппаратных затрат. Простое объединение независимо преобразованных кадровых структур согласно информационной зависимости не приводит к сбалансированному решению всей задачи, когда все взаимосвязанные кадровые структуры работают согласованно, с единым темпом обработки данных. Для эффективной реализации информационно-зависимых кадровых структур нужно найти *рациональное* решение системы (4.1), в котором степени параллелизма во всех кадровых структурах будут согласованы для обеспечения минимально возможного интервала обработки данных. Поиск рационального решения (4.1) является трудноразрешимым уже при сравнительно небольшом числе подзадач, а с ростом числа неравенств точное решение системы диофантовых неравенств в целых числах не гарантированно, т.к. оно зависит от абсолютных значений параметров параллелизма кадровых структур подзадач и коэффициента редукции.

В общем случае, наиболее простой и эффективный способ упрощения поиска рационального решения (4.1) – это сокращение размерности решаемой системы. Для этого можно рассматривать не все неравенства системы, а только два – те, которые описывают информационно-взаимозависимые кадровые структуры, т.е. связанные непосредственной информационной зависимостью. Поиск рационального решения в целых числах для двух неравенств системы (4.1) намного проще, т.к. вариант решения, пусть даже и с некоторой потерей эффективности, всегда можно найти. Найденное для двух информационно-взаимозависимых подзадач решение считается единой «подзадачей» с общими рассчитанными параметрами $R_i^L$, $R_i^{It}$, $R_i^Q$, $R_i^\rho$, $R_i^I$, с которой согласовываются параметры следующей информационно-взаимозависимой подзадачи. Так, после согласования первых двух неравенств системы (4.1) сократилась на 1 не только ее размерность, но и для одного из уравнений получены параметры $R_s^L$, $R_s^{It}$, $R_s^Q$, $R_s^\rho$, $R_s^I$, к которым будут приводиться (адаптироваться) соответствующие

параметры присоединяемой к решению подзадачи. Согласование параметров кадровых структур с объединенной «подзадачей» будем называть операцией *присоединения* к промежуточному согласованному решению. Присоединение кадровых структур к согласованному решению последовательно повторяется для всех подзадач в (4.1). Такое построение схемы согласования параметров кадровых структур обеспечивает гарантированную сходимость преобразования информационно-зависимых подзадач даже при некоторой потере эффективности результирующего решения.

Таким образом, для существенного сокращения сложности решения системы диофантовых неравенств (4.1) и гарантированной сходимости преобразования для информационно-зависимых подзадач, на каждом шаге согласуются параметры двух информационно-взаимозависимых кадровых структур. Это сокращает на 1 число неравенств, а рассчитанные параметры объединенной «подзадачи» задают начальные значения для присоединяемых кадровых структур, уменьшая неопределенность и сокращая число анализируемых для рационального решения вариантов. Это утверждение можно сформулировать в виде первого принципа согласованного преобразования информационно-зависимых кадровых структур следующим образом: **на каждом этапе преобразования информационно-зависимых подзадач согласование параметров параллелизма и интервала обработки данных выполняется для двух информационно-взаимозависимых кадровых структур.**

Решение системы диофантовых неравенств (4.1) выполняется поэтапно, причем на каждом этапе рассматривается не более двух неравенств, соответствующих информационно-взаимозависимым кадровым структурам. Для этого неравенства системы (4.1) нужно переупорядочить согласно информационной зависимости подзадач. Методы решения СЛАУ всегда допускают такое преобразование. Далее будем полагать, что порядок неравенств в (4.1) соответствует информационной зависимости кадровых структур, а решение системы (4.1) всегда начинается с первых двух неравенств.

Выбор начальной пары кадровых структур для согласования важен, т.к. полученная объединенная «подзадача» влияет на параметры всех остальных присоединяемых кадровых структур. Поэтому выбирать кадровые структуры для начального согласования нужно не в порядке их следования в задаче, а по их вкладу в общую трудоемкость, критический ресурс и время решения задачи. Как правило, наибольший вклад во время решения задачи вносит подзадача с наибольшей вычислительной трудоемкостью и она же характеризуется наибольшим превышением аппаратных затрат по критическому ресурсу. Тогда самую трудоемкую подзадачу нужно преобразовать к целевым архитектурам гибридной вычислительной системы наиболее рациональным образом, обеспечивающим минимальный возможный интервал обработки данных, чтобы максимально сократить общее время решения. Это позволяет определить первое неравенство системы (4.1), соответствующее самой трудоемкой подзадаче. Тогда первой парой неравенств для согласования выбирается наиболее трудоемкая подзадача и информационно-взаимозависимая от нее кадровая структура. Если взаимозависимых кадровых структур несколько, то они так же ранжируются по критическому ресурсу и трудоемкости, и выбирается наиболее трудоемкая из них. Сформулируем это положение в виде второго принципа согласованного преобразования информационно-зависимых кадровых структур следующим образом: **для начального согласования параметров параллелизма и интервала обработки данных выбирается подзадача с наибольшей вычислительной трудоемкостью и максимальным превышением аппаратных затрат по критическому ресурсу.**

Для применения редукционных преобразований к информационно-зависимым подзадачам для каждой кадровой структуры нужно выделить область доступного аппаратного ресурса из общего ресурса целевых архитектур гибридной вычислительной системы. Разделение всего аппаратного ресурса на равные части по числу подзадач имеет серьезный недостаток, т.к. уравнивает подзадачи по занимаемому ресурсу, когда вклад каждой подзадачи в

вычислительную трудоемкость, аппаратный ресурс и общее время решения может существенно различаться. При равном распределении вычислительного ресурса наиболее трудоемкая по критическому ресурсу подзадача может потребовать редукции по числу устройств и разрядности до м-кадров, а наименее трудоемкие задачи могут быть не только реализованы структурно, но и распараллелены, что усилит несбалансированность и неэффективность решения. В этом случае, для согласования темпа обработки данных между подзадачами, наименее трудоемкие задачи потребуется редуцировать по интервалу обработки данных, что снизит удельную производительность решения. Производительность и эффективность такого решения будут невелики, а время решения значительно увеличится.

Наиболее рациональный способ распределения доступного аппаратного ресурса между подзадачами – это выделение наибольшей его доли самой трудоемкой подзадаче, а менее вычислительно-трудоемкие, вклад которых в общее время решения незначителен, можно реализовывать на оставшемся ресурсе гибридной вычислительной системы, в том числе активно используя процессорные вычислительные узлы. Выделение аппаратных ресурсов должно быть гибким, т.е. если какие-то из ресурсов целевых архитектур не требуются для реализации текущей кадровой структуры, то они остаются свободными для реализации других подзадач. Поэтому третий принцип согласованного выбора параметров параллелизма и интервала обработки данных в информационно-зависимых кадровых структурах можно сформулировать следующим образом: **аппаратный ресурс гибридной вычислительной системы распределяется между информационно-зависимыми кадровыми структурами пропорционально их вкладу в общую вычислительную трудоемкость задачи так, чтобы задача с наибольшей трудоемкостью получала наибольшую часть доступного ресурса**.

Для дальнейшего описания и анализа методов преобразования информационно-зависимых кадровых структур следует различать подзадачи по

величине или вкладу в общую трудоемкость задачи. Вклад каждой подзадачи в общую вычислительную трудоемкость оценивается по числу операций в информационном графе задачи или числу устройств в максимально параллельной кадровой структуре. В любой задаче ее информационно-зависимые подзадачи можно сравнить по числу вычислительных устройств в максимально параллельной кадровой структуре, которые характеризуют вычислительную трудоемкость и занимаемый аппаратный ресурс каждой подзадачи. Уже на уровне модели – информационного графа задачи, среди всех подзадач можно выделить ту, которая характеризуется наибольшим подграфом (по числу операционных вершин) и которой соответствует наибольшая кадровая подструктура (по числу устройств) в максимально параллельной кадровой структуре. Для описания различий подграфов по числу вершин можно использовать антонимы, на качественном уровне характеризующие объекты разной величины – например, «флагманский корабль», характеризующий подграф с наибольшим число вершин, и «катер» для подграфа с существенно меньшим числом вершин. В дальнейшем кадровую структуру с наибольшим числом устройств и/или занимаемым в максимально параллельной форме аппаратным ресурсом будем называть «флагманской» или «флагманом». Кадровые структуры или фрагменты ИГЗ с существенно меньшим числом устройств или ресурсом (как минимум на 1 десятичный порядок,) будем называть «вспомогательными» или «катером», подчеркивая существенное различие между ними.

На основе сформулированных принципов согласованного выбора параметров параллелизма и интервала обработки данных и методики преобразования параметризуемых ресурсом кадровых структур (см. главу 3) нужно разработать метод преобразования информационно-зависимых подзадач к рациональному варианту организации параллельных вычислений для целевых архитектур гибридной вычислительной системы.

## 4.2. Метод преобразования информационно-зависимых подзадач к рациональному варианту организации параллельных вычислений

Согласно первому и второму принципам согласованное преобразование информационно-зависимых подзадач будем рассматривать как сбалансированную редукцию двух информационно-взаимозависимых кадровых структур – «флагмана» и «катера» (рис. 4.4), различающихся вычислительной трудоемкостью и временем выполнения вычислений. «Флагман» и «катер» характеризуют подзадачи относительными соотношениями между требуемыми аппаратными затратами, поэтому будем считать, что кадровая структура «флагмана» занимает не менее 50% от общих аппаратных затрат максимально параллельной формы задачи, а «катер» – не более 10%. Занимаемый подзадачами ресурс в максимально параллельной кадровой структуре можно оценить с помощью произведения числа слоев, итераций и устройств, а в последовательной программе – с помощью произведения числа итераций всех циклов и числа операций в теле цикла.

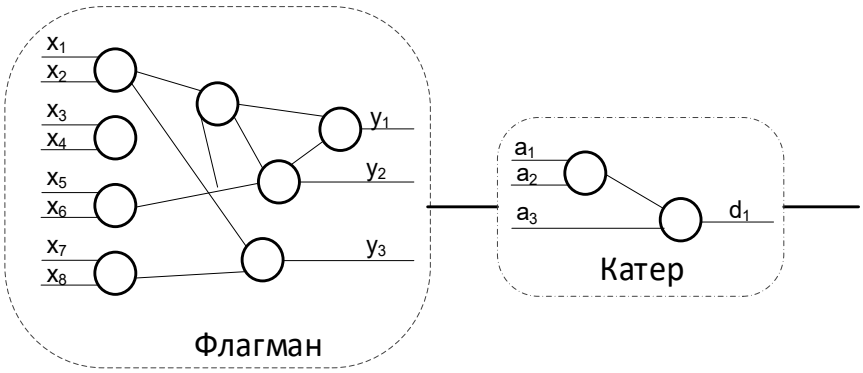



Рис. 4.4. Информационно-зависимые подзадачи «флагман» и «катер»

Подзадачи «флагман» и/или «катер» можно выделить в большинстве прикладных задач:

- в задачах линейной алгебры: например, в решении СЛАУ методом Гаусса можно выделить 2 информационно-зависимые подзадачи с разными БП – прямой и обратный ход алгоритма, вычислительная трудоемкость которых

определяется по числу итераций циклов в последовательной программе: прямой ход алгоритма является «флагманом», а обратный ход – «катером»;

- в задачах символьной обработки с композицией блочных и поточных алгоритмов (например, MD5 и RC4), реализация блочного алгоритма по занимаемому ресурсу и вычислительной трудоемкости, как правило, является «флагманом», а реализация поточного алгоритма – «катером»;
- в задачах цифровой обработки сигналов (например, при реализации свертки), «флагманом» обычно является прямое БПФ, а вспомогательные операции его обработки, например, комплексное умножение являются «катером».

Рассмотрим согласование информационно-зависимых кадровых структур «флагман» и «катер» на основе методики (см. главу 3) и сформулированных принципов. Согласно первому принципу, для согласования параметров параллелизма и интервала обработки данных будут выбраны две информационно-взаимозависимые подзадачи «флагман» и «катер» (по вычислительной трудоемкости). Согласно второму принципу, выбор «опорной» подзадачи для согласования определяется максимумом вычислительной трудоемкости и критического ресурса, поэтому в качестве нее выбирается «флагман». Согласно третьему принципу, «флагману» будет выделена наибольшая часть доступного аппаратного ресурса ГВС пропорционально его трудоемкости. Далее, к кадровой структуре «флагмана» и части доступного аппаратного ресурса, выделенной для его реализации, можно применить методику преобразования (см. главу 3). «Катер» редуцируется на оставшемся после преобразования «флагмана» аппаратном ресурсе, а затем согласуются параметры редуцированных кадровых структур.

Основная цель редукции «флагмана» в представленной методике – это получение конвейерного решения, поэтому важно, чтобы результат редукции «флагмана» оставался в верхнем октанте пространства возможных реализаций кадровых структур (см. главу 2) и сохранил минимальный интервал обработки

данных $I = 1$. При невыполнении этого условия увеличится интервал обработки данных «флагмана», что практически линейно увеличит и общее время решения задачи. В наилучшем случае, конвейерное решение нужно найти и для «катера», но это не всегда возможно. В соответствии с третьим принципом согласованного преобразования информационно-зависимых кадровых структур, для «катера» выделяется существенно меньшая часть оставшегося доступного аппаратного ресурса. Поэтому найти структурное решение для «катера» и остаться в верхнем октанте пространства возможных реализаций кадровых структур не всегда возможно и не исключен переход в нижний октант к структурно-процедурной или процедурной реализации.

Из-за значительной разницы вычислительных трудоемкостей «флагмана» и «катера», не всегда возможно обеспечить конвейерное решение для «катера» при редукции с тем же коэффициентом. Однако, меньшая вычислительная трудоемкость означает и меньший вклад в общее время решения задачи, поэтому даже процедурная реализация «катера» вполне допустима, т.к. не оказывает существенного влияния на общее время. Тогда результат редукции для «флагмана» должен приближаться к оптимальному по интервалу обработки данных $I = 1$, а для «катера» допустимо любое, в том числе процедурное решение, но обязательно согласованное с «флагманом». В этом случае, полученное решение всей задачи будет не оптимальным, а рациональным.

В связи с этим согласованную редукцию нужно реализовать так, чтобы наилучшим образом преобразовать кадровую структуру «флагмана» как наиболее трудоемкого фрагмента к наиболее эффективному варианту организации параллельных вычислений с минимальным интервалом обработки данных. После этого, сбалансированно сократить занимающий существенно меньший ресурс «катер» с тем же коэффициентом редукции до рациональной реализации. Под рациональной реализацией «катера» понимается решение, которое наиболее просто воплощается в целевых архитектурах ГВС без перехода к последовательной поразрядной обработке данных. Последовательная

поразрядная обработка, формально правильная с точки зрения редукционных преобразований, крайне неэффективна при практической реализации в современных вычислительных системах. Она существенно увеличивает интервал обработки данных не только в этой подзадаче, но и во всей кадровой структуре, сокращая тем самым эффект от рациональной реализации «флагмана». В этом случае более целесообразна процедурная реализация «катера» на любом свободном ресурсе целевых архитектур ГВС (ПЛИС, процессорах или графических ускорителях) с увеличением интервала обработки данных до значения, необходимого для согласования с «флагманом».

Рассмотрим математическое представление согласованной редукции производительности подзадач «флагман» и «катер», представленных в максимально параллельной кадровой структуре прикладной задачи. Согласно (4.1), общее число вычислительных операций «флагмана» и «катера» в максимально параллельной кадровой структуре выразим как

$$NC_{\Phi} = N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi} \cdot Op_{\Phi} \cdot \rho_{\Phi}$$

$$NC_{K} = N_{K}^{F} \cdot N_{K}^{\Phi} \cdot Op_{K} \cdot \rho_{K} ., \qquad (4.2)$$

где нижние индексы «ф» и «к» параметров параллелизма относятся к «флагману» и «катеру», соответственно. Далее, для компактного описания параметров параллелизма подзадач будем обозначать общее число вычислительных операций (4.2) в форме кортежа $\langle N^{F}, N^{\Phi}, Op, \rho \rangle$ из абсолютных значений их параметров.

Согласно введенному определению, число операций «флагмана» существенно больше числа операций «катера» $NC_{\Phi} \gg NC_{K}$. Это значит, что число обрабатываемых «флагманом» данных больше числа данных «катера», т.е. число слоевых и итерационных подграфов «флагмана» значительно превышает аналогичный показатель «катера»: $N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi} \gg N_{K}^{F} \cdot N_{K}^{\Phi}$. Теоретически возможный, но на практике маловероятный случай, когда число подграфов «флагмана» незначительно превышает число подграфов «катера»

$N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi} \cong N_{K}^{F} \cdot N_{K}^{\Phi}$, а число устройств меньше ($Op_{\Phi} < Op_{K}$), не будем принимать во внимание на данном этапе.

Условием структурной реализации подзадач и завершения редукции с коэффициентом $R$ в верхнем октанте пространства возможных реализаций кадровых структур (см. главу 2) является сокращение только слоевых и/или итерационных подграфов, без применения редукций по числу устройств и разрядности к функциональным подграфам подзадач. Для «флагмана» и «катера» это условие описывается выражениями

$$\left(N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi}\right)/R \geq 1 \text{ и } \left(N_{K}^{F} \cdot N_{K}^{\Phi}\right)/R \geq 1, \tag{4.3}$$

где операция «/» является целочисленным делением.

При выполнении условия (4.3), как уже отмечалось, завершение редукционных преобразований кадровой структуры в верхнем октанте пространства возможных реализаций (см. главу 2) позволяет обеспечить минимальный интервал обработки данных кадровой структуры $I = 1$. При переходе в нижний октант (см. главу 2) интервал неизбежно возрастет, а также пропорционально увеличится и время решения. При редукции подзадач «флагман» и «катер» в зависимости от соотношений между абсолютными значениями параметров задач и доступным аппаратным ресурсом гибридной вычислительной системы возможны четыре качественно различающихся случая:

1. Редукция обеих подзадач завершилась в верхнем октанте и они реализуемы структурно $\left(N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi}\right)/R \geq 1$ и $\left(N_{K}^{F} \cdot N_{K}^{\Phi}\right)/R \geq 1$.

2. Редукция «флагмана» завершилась в верхнем октанте, а «катер» перешел в нижний октант $\left(N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi}\right)/R \geq 1$ и $\left(N_{K}^{F} \cdot N_{K}^{\Phi}\right)/R < 1$ (наиболее распространенный случай).

3. Редукция обеих подзадач завершилась в нижнем октанте, поэтому они не реализуемы структурно $\left(N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi}\right)/R < 1$ и $\left(N_{K}^{F} \cdot N_{K}^{\Phi}\right)/R < 1$.

4. Редукция «флагмана» завершилась в нижнем октанте, а «катер» остался в верхнем: $\left(N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi}\right)/R < 1$ и $\left(N_{K}^{F} \cdot N_{K}^{\Phi}\right)/R \geq 1$.

В первом случае возможна структурная реализация с одинаковым минимальным интервалом обработки данных в обеих подзадачах, что обеспечит рациональное, близкое к минимальному, время решения. Поэтому, если совпадают интервалы обработки данных в структурных реализациях редуцированных кадровых структур «флагмана» и «катера», то согласования параметров не требуется. После редукции и согласования параметров интервала обработки данных «флагман» и «катер» объединяются в одну подзадачу, которая является «опорной» для присоединения других информационно-взаимозависимых подзадач. Параметры параллелизма и интервала обработки данных объединения «флагмана» и «катера» не меняются для всех остальных присоединяемых подзадач, характеристики которых адаптируются (приводятся) к рассчитанным параметрам «флагмана» и «катера».

Во втором случае, наиболее распространенном на практике, при $\left(N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi}\right)/R \geq 1$ и $\left(N_{K}^{F} \cdot N_{K}^{\Phi}\right)/R < 1$, требуется явное согласование параметров параллелизма и интервала обработки данных редуцированных кадровых структур, т.к. у «флагмана», оставшегося в верхнем октанте пространства возможных реализаций, интервал обработки данных остается минимальный, а у «катера», перешедшего в нижний октант, - неизбежно увеличится. В зависимости от соотношения между исходными параметрами «катера», коэффициентом редукции и интервалами обработки данных в редуцированных структурах «флагмана» и «катера», для последнего возможны следующие основные способы согласованной с «флагманом» реализации:

- реализация с помощью м-кадров с согласованием интервала обработки данных;

- процедурная реализация при наличии процессорных вычислительных узлов в целевых архитектурах ГВС;

- структурная реализация с редукцией по интервалу обработки данных.

Реализация «катера» с помощью м-кадров с согласованием интервала обработки данных наиболее целесообразна при построении м-кадра $\mu_2$, когда

число устройств, каналов или разрядность кратны множителям коэффициента редукции. Также реализация «катера» с помощью м-кадров возможна при отсутствии процедурных узлов среди целевых архитектур гибридной вычислительной системы. В этом случае нужно использовать критерий целесообразности редукции по числу устройств и переходу к м-кадрам (см. главу 3), согласно которому накладные расходы на коммутацию потоков данных не должны превышать аппаратных затрат на реализацию вычислений. По оценкам (см. главу 3), переход к м-кадрам для 32-разрядной арифметики целесообразен для коэффициентов редукции, не превышающих 68 и 35 для операций с плавающей и фиксированной точкой для ПЛИС семейства Xilinx Virtex-7. Если коэффициент редукции превышает указанные значения, то в редуцируемой подзадаче существенно возрастет не только интервал обработки данных $I$, но и аппаратные затраты на вспомогательное коммутационное оборудование для выравнивания и синхронизации потоков данных. Построение конвейерной схемы для таких вычислений не имеет смысла, т.к. не приведет к заметному сокращению времени решения. Поэтому лучше реализовать эти вычисления процедурно, если среди целевых архитектур ГВС имеются процессорные узлы. Кроме того, в качестве процессорных узлов можно использовать аппаратные реализации PowerPC, входящие в состав ряда ПЛИС (микросхемы семейства Virtex), процессоры ARM (для перспективных кристаллов с архитектурой Versal) или софт-процессоры (например, MicroBlaze). Использование процессорных узлов гибридной вычислительной системы освободит аппаратный ресурс других целевых архитектур (в первую очередь - ПЛИС) и в ряде случаев позволит эффективнее реализовать «флагман», дополнительно сократив общее время решения задачи.

Структурная реализация «катера» с редукцией по интервалу обработки данных оправданна, когда процессорные вычислительные узлы отсутствуют, но имеется свободный аппаратный ресурс ПЛИС, неприменимый для улучшения реализации «флагмана». Вычисления «катера» вносят значительно (на 1-2

десятичных порядка) меньший вклад в общее время решения задачи, а его аппаратная реализация занимает существенно меньший по сравнению с «флагманом» аппаратный ресурс, поэтому основное сокращение аппаратных затрат при преобразовании кадровой структуры к рациональному варианту организации параллельных вычислений обеспечивается при редукции «флагмана», а не «катера». При наличии свободного аппаратного ресурса ПЛИС преобразование кадровой структуры «катера» следует остановить в верхнем октанте пространства реализаций на плоскости слоев и итераций (см. главу 2), т.к. редукция с переходом в нижний октант приведет к увеличению интервала обработки данных, потребует специализированных библиотечных элементов, а также дополнительных шагов и аппаратного ресурса для согласования потоков данных между остальными подзадачами и входами/выходами «катера». Для согласования характеристик потоков данных между разными фрагментами можно использовать буферы (при незначительной разнице) или элементы памяти (в т.ч. внутренней памяти ПЛИС), выполняющие ту же функцию в случае существенного разрыва в длине потоков между разными ФП.

В третьем случае, когда редукция и «флагмана», и «катера» завершилась в нижнем октанте $\left(N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi}\right)/R < 1$ и $\left(N_{K}^{F} \cdot N_{K}^{\Phi}\right)/R < 1$, и ни одна из подзадач не может быть реализована структурно из-за нехватки ресурса, возможны два качественно различных способа согласования:

- если интервалы обработки данных «флагмана» и «катера» совпадают (или различаются незначительно), то можно построить конвейерную вычислительную структуру с общим интервалом обработки данных $I > 1$. Время решения задачи увеличится кратно общему интервалу $I$;

- если интервалы различаются существенно, то реализовать «катер» на процессорных вычислительных узлах, а освободившийся ресурс выделить для структурной реализации «флагмана». Если это невозможно из-за отсутствия процессорных узлов среди целевых архитектур ГВС, то единственным вариантом остается реализация «катера» м-кадрами или процедурно.

Для «флагмана», вносящего основной вклад в длительность решения всей задачи, крайне важна максимально эффективная реализация, которая обеспечит минимально возможное время обработки данных для заданного аппаратного ресурса. Поэтому редукционные преобразования «флагмана» должны завершиться в верхнем октанте пространства реализаций кадровых структур, т.е. нужно преобразовывать конвейерную (как наиболее эффективную) вычислительную структуру по слоям и итерациям для минимизации времени обработки данных. Такую стратегию преобразования кадровой структуры к рациональному варианту организации параллельных вычислений назовем «флагман получает все». Полученный для «флагмана» интервал обработки данных задает его и для всех остальных подзадач, т.е. для согласования потоков данных необходимо обеспечить такой же интервал во всех подзадачах. Если для «флагмана» получен минимальный интервал обработки данных, то и в остальных подзадачах он без оптимизации будет таким же, а общее число шагов преобразования сократится. Таким образом, «флагман» - это основной фрагмент, к которому приводятся все остальные подзадачи, т.е. от реализации «флагмана» зависит остаток аппаратного ресурса для прочих подзадач и интервал обработки потоков данных. Присоединение подзадач к «флагману» без изменения его параметров позволяет сократить число преобразований и взаимных согласований кадровых структур.

Четвертый случай, когда редукция «флагмана» завершилась в нижнем октанте, т.е. $\left(N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi}\right)/R < 1$, а «катер» остался в верхнем, т.е.$\left(N_{K}^{F} \cdot N_{K}^{\Phi}\right)/R \geq 1$, практически невозможен, что легко формально доказывается. По определению «флагмана» и «катера» число операций «флагмана» существенно больше числа операций «катера» $N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi} \gg N_{K}^{F} \cdot N_{K}^{\Phi}$, поэтому при делении обеих частей на коэффициент редукции (неотрицательное число) знак неравенства не меняется: $(N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi})/R \gg (N_{K}^{F} \cdot N_{K}^{\Phi})/R$. Поскольку редукция «флагмана» завершилась в нижнем октанте, т.е. $\left(N_{\Phi}^{F} \cdot N_{\Phi}^{\Phi}\right)/R < 1$, то и для «катера» должно выполняться условие $\left(N_{K}^{F} \cdot N_{K}^{\Phi}\right)/R < 1$, что противоречит условию четвертого случая

$\left(N_K^F \cdot N_K^Ф\right)/R \geq 1$, поэтому они не могут находиться в разных октантах. Если выделенного «флагману» аппаратного ресурса недостаточно для реализации его минимального базового подграфа, то, согласно стратегии «флагман получает все», «катер» реализуется на процессорных вычислительных узлах, а освободившийся ресурс выделяется для структурной реализации «флагмана».

В качестве детализации рассмотренных случаев информационно-зависимых подзадач для общей схемы (рис. 4.3), предложена полученная в результате преобразования максимально параллельной кадровой структуры задачи «флагман» и «катер» кадровая структура (рис. 4.5). В отличие от структуры (рис. 4.3), блок согласования интервала представляется не одним, а несколькими непосредственно соединенными между собой вспомогательными коммутационными блоками, на аппаратном уровне выполняющими согласование параметров параллелизма в информационно-зависимых кадровых структурах.

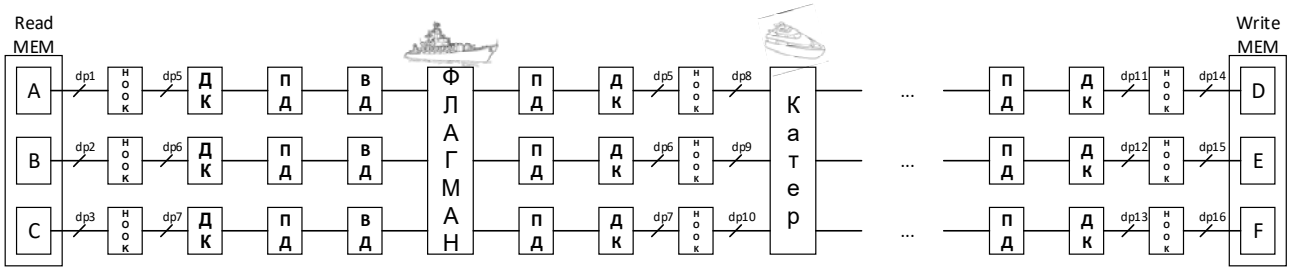



Рис. 4.5. Согласование информационно-зависимых подзадач «флагман» и «катер»

Блок согласования интервала обработки данных (рис. 4.3) представлен на рис. 4.5 последовательностью блоков согласования каналов данных (hook), динамической коммутации (ДК), подготовки данных из разных каналов (ПД) и выбора данных (ВД), каждый из которых выполняет определенную функцию согласования.

Блок согласования каналов данных (рис. 4.6) преобразует число входных каналов данных к требуемому параметрами кадровой структуры: на рис. 4.6-а

представлено сокращение, а на рис. 4.6-б – увеличение числа входных каналов данных.

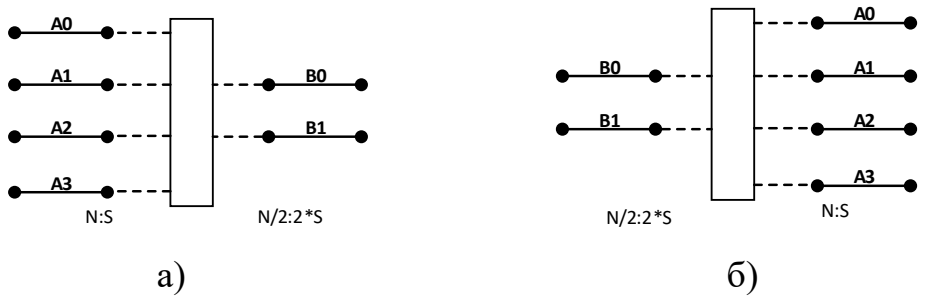



Рис. 4.6. Структурная схема блока согласования каналов данных hook:

а) при сокращении числа входных каналов данных,

б) при увеличении числа входных каналов данных

Согласование каналов данных, выполняющееся путем увеличения интервала подачи данных, необходимо для согласования потоков данных в структурах с разной степенью параллелизма. Блок динамической коммутации ДК (рис. 4.7) обеспечивает корректную подачу данных при изменении порядка коммутации каналов на разных итерациях алгоритма.

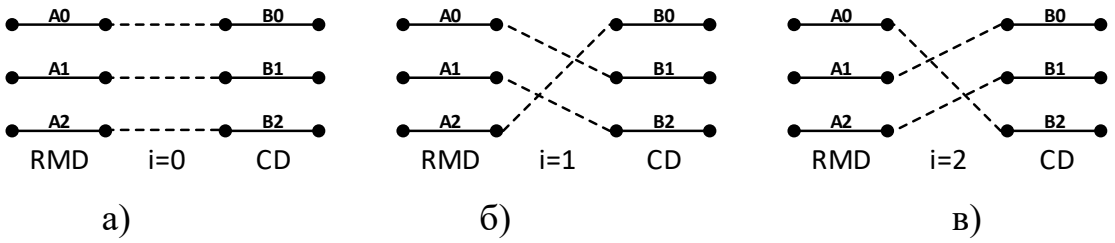
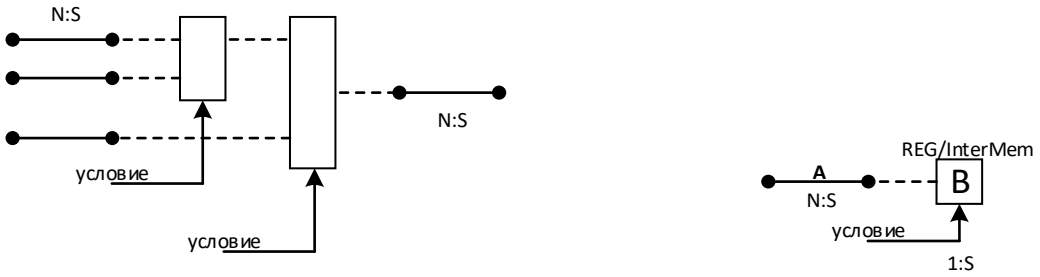



Рис. 4.7 Структурная схема блока динамической коммутации

а) исходный порядок поступления данных, б) порядок поступления данных на первой итерации, в) порядок поступления данных на второй итерации

Блок подготовки данных (рис. 4.8-а) из разных каналов ПД формирует единый непрерывный поток данных, поступающих в разные моменты из разных каналов. Блок выбора данных ВД сохраняет одно (REG) или несколько (InterMem) значений из потока данных, которые многократно используются в дальнейших вычислениях.




а) б)

Рис. 4.8. Структурные схемы блоков подготовки и выборки данных

а) блок подготовки данных, б) блок выборки данных

Исходя из рассмотренных принципов преобразования «флагмана» и «катера», основные этапы метода преобразования информационно-зависимых кадровых структур к рациональному варианту организации параллельных вычислений можно сформулировать следующим образом:

1. Выделить все подзадачи в структуре задачи и определить информационную зависимость между ними.

2. Для каждой подзадачи определить список функциональных подграфов, число слоев и итераций в каждом ФП, выделить минимальный подграф подзадачи, определить минимальный БП всей задачи, вес каждой подзадачи, коэффициент сокращения аппаратных затрат и коэффициент редукции производительности.

3. Проверить возможность реализации минимального БП всей задачи на минимальном и доступном аппаратных ресурсах гибридной вычислительной системы.

4. Выделить подзадачи «флагман» и «катер», составить систему диофантовых уравнений (4.1), упорядочить ее согласно сформулированным принципам и разрешить ее для «флагмана» и «катера».

5. Выполнить согласованную редукцию информационно-зависимых подзадач «флагман» и «катер», объединить в их единую структуру.

6. Выполнить оптимизацию полученной кадровой структуры по интервалу обработки данных.

7. Оценить аппаратные затраты на реализацию оптимизированной вычислительной структуры наиболее трудоемкой задачи и определить доступный для остальных подзадач аппаратный ресурс.

8. Выполнить согласование для остальных подзадач, используя интервал обработки данных и степень параллелизма «флагмана» в качестве целевого значения, при необходимости используя редукцию по интервалу обработки данных.

Одним из основных преимуществ представленного метода являются сравнительно небольшое число анализируемых вариантов реализации подзадач и возможность его автоматической реализации. Если структурно с минимальным интервалом реализовать все остальные подзадачи, кроме «флагмана», мы получим незначительное (как правило, не более 3%) сокращение времени решения, но израсходуем большое количество аппаратного ресурса, которого затем будет недостаточно для рациональной параллельно-конвейерной реализации «флагмана». В то же время, улучшение реализации «флагмана» как конвейерной вычислительной структуры кратно сокращает время решения задачи, поэтому именно «флагман» следует реализовать наиболее рационально. Найденные параметры реализации «флагмана» определяют и ряд параметров присоединяемых подзадач, в первую очередь – «катера». В результате согласованной редукции по предложенному методу, «флагман», содержащийся во всех рассматриваемых типах задач, является образующим центром для присоединения всех остальных подзадач.

На основе представленного метода, предлагается обобщенный алгоритм преобразования кадровой структуры задачи с информационно-зависимыми подзадачами к рациональному варианту организации параллельных вычислений (рис. 4.9). Алгоритм основан на принципах движения в пространстве реализаций кадровых структур (см. параграф 2.4), согласно которым для задачи, состоящей из нескольких подзадач, необходимо определить положение исходного ИГЗ и минимального БП задачи (составленного из минимальных базовых подграфов

всех подзадач $g_{стр}$) относительно области доступного аппаратного ресурса. Поэтому в алгоритме (рис. 4.9) одним из первых действий сравнивается аппаратный ресурс и число каналов гибридной вычислительной системы с аппаратными затратами на реализацию итерационных и слоевых подграфов.

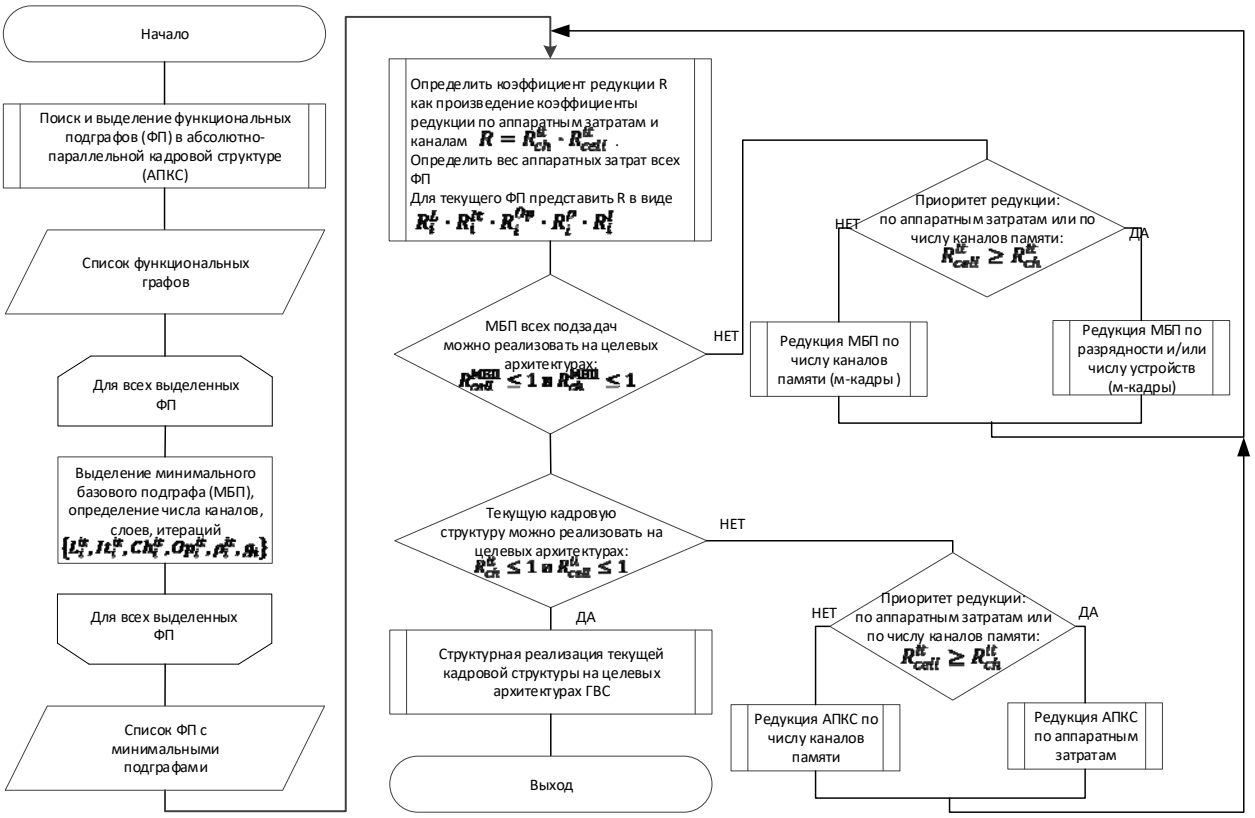



Рис. 4.9. Обобщенный алгоритм согласованного преобразования кадровых структур

Сначала в задаче выделяются подзадачи, в каждой из которых проводится поиск функциональных подграфов, определение числа слоев и итераций. На основе ФП всех подзадач формируется минимальный БП всей задачи, для которого оценивается занимаемый аппаратный ресурс и рассчитывается коэффициент редукции. Если даже минимальный БП задачи превышает доступный аппаратный ресурс, его нужно редуцировать до приемлемого аппаратного ресурса с помощью разбиения на м-кадры (по каналам, числу устройств, разрядности), а затем преобразовать кадровую структуру к

рациональному варианту организации параллельных вычислений. Если минимальный БП реализуем на доступном ресурсе ГВС (т.е. находится в верхнем октанте), проверяется возможность его полной структурной реализации: если ИГЗ аппаратно реализуем, то преобразование кадровой структуры не нужно. В большинстве случаев аппаратного ресурса недостаточно для структурной реализации и преобразования нужны, а для их успешного выполнения необходимо определить методы редукции, которые обеспечат получение эффективной кадровой структуры.

Представленный обобщенный алгоритм согласованного преобразования информационно-зависимых кадровых структур является базовым для большинства задач с информационно-зависимыми подзадачами, в том числе и для задачи «флагман»-«катер». Вместе с тем, у базовой информационной зависимости «флагман»-«катер» есть важные частные случаи с особенностями, учет которых при преобразовании кадровой структуры позволяет получить более рациональный вариант организации параллельных вычислений. Одна из таких наиболее распространенных зависимостей – это информационная зависимость нескольких «катеров» от «флагмана» и друг от друга.

## 4.3. Метод преобразования «флагмана» и множества информационно-зависимых «катеров» к рациональному варианту организации параллельных вычислений

В предыдущем параграфе рассматривалось преобразование двух информационно-зависимых подзадач «флагман» и «катер», качественно различающихся по вычислительной трудоемкости. Многие прикладные задачи (например, решение СЛАУ методом Гаусса, расчет спектральных характеристик сигналов и др.) содержат в своей структуре больше одного «катера», что позволяет определить такую информационную зависимость как «флагман»-множество «катеров». Для сокращения записи обозначим задачи с такой

структурой как «ф-{к}». Здесь символы «{}» обозначают непустое множество «катеров» аналогично символу повторения в описании формальных грамматик с помощью расширенных форм Бэкуса-Наура. В структуре реальных прикладных задач множество подзадач-«катеров» могут как предшествовать «флагману», так и находиться после него (рис. 4.10). Более того, у одного «флагмана» может быть несколько информационно-взаимозависимых «катеров» (рис. 4.10-б), каждый из которых также может иметь информационно-взаимозависимые подзадачи. Далее, в качестве основной будем рассматривать структуру задачи на рис. 4.10-а, при необходимости оговаривая особенности для задач со структурой на рис. 4.10-б.

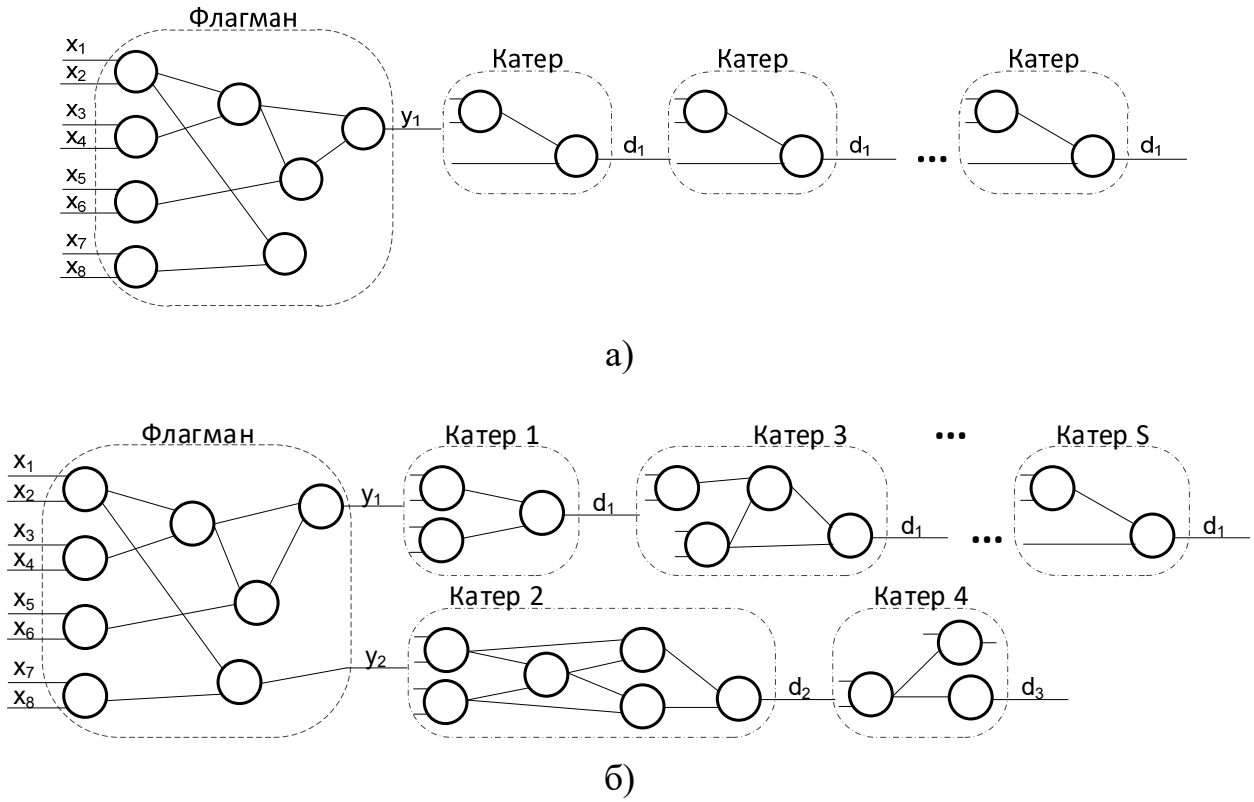



Рис. 4.10. Структура задачи «флагман»-множество «катеров»

Максимально параллельная кадровая структура исходного представления задачи с выделенными подзадачами «ф-{к}» показана на рис. 4.11-а. Согласно методу (см. параграф 4.2) преобразование такой задачи начинается с информационно-взаимозависимой пары «флагман»-«катер» (рис. 4.11-б).

Поэтому выделенный пунктиром «флагман» (рис. 4.11-а) с помощью блока согласования БС будет объединен с информационно-взаимозависимым «Катером 1», образуя в результате промежуточный «Флагман 1’» (рис. 4.11-б).

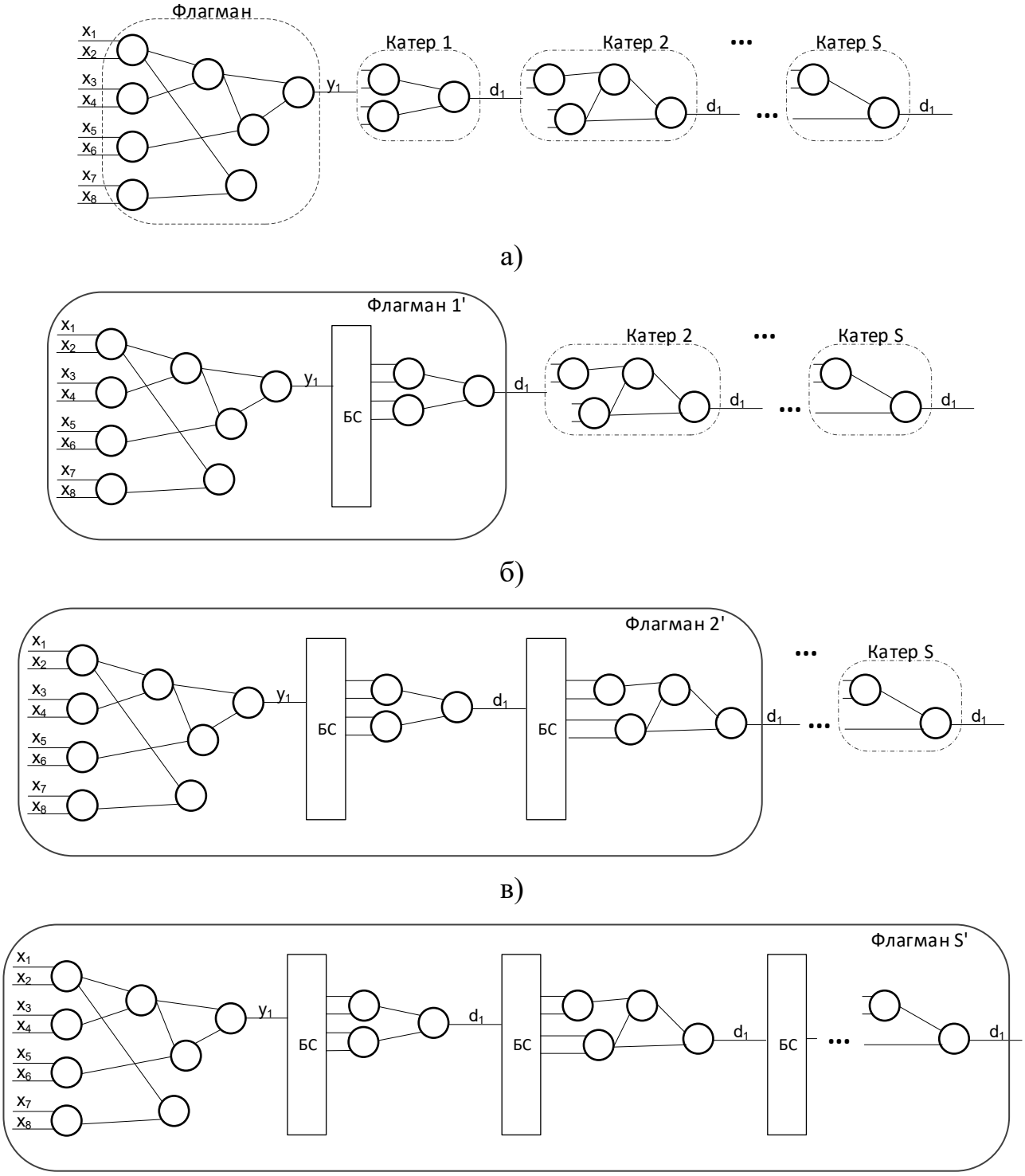



Рис. 4.11. Порядок присоединения «катеров» в задаче «ф-{к}»

Как и для задачи «флагман»-«катер», тип блока согласования БС зависит от абсолютных значений параметров параллелизма $\langle N^F, N^\Phi, Op, \rho\rangle$ соединяемых подзадач «флагман» и «Катер 1» (рис. 4.5-4.8). На втором этапе преобразований (рис. 4.11-в) к сформированному промежуточному «Флагману 1'» присоединяется информационно-взаимозависимый «Катер 2», образуя промежуточный «Флагман 2'». Согласование параметров параллелизма «Флагмана 1'» и «Катера 2» также выполняется блоком согласования БС. Остальные «катера» присоединяются аналогично. Преобразование задачи завершается с присоединением последнего «Катера S» и образованием итоговой кадровой структуры задачи «Флагман S'» (рис. 4.11-г).

Таким образом, преобразование задачи «ф-{к}» к рациональному варианту организации параллельных вычислений состоит в последовательном и поэтапном применении метода (см. параграф 4.2), т.е. присоединении к «флагману» непосредственно связанного с ним «катера», которое можно описать выражением

$$\left(\left((\text{ф}-\text{к}_1)-\text{к}_2\right)-\cdots-\text{к}_s\right)=\cup_{i=1}^{S}\Phi\leftarrow k_i\ , \qquad (4.4)$$

где скобки означают операцию присоединения.

Для нескольких информационно-взаимозависимых «катеров» (рис. 4.10-б) порядок присоединения не меняется. В соответствии со вторым принципом согласованного преобразования информационно-зависимых кадровых структур (см. параграф 4.1), для первого присоединения выбирается наиболее трудоемкий из информационно-взаимозависимых «катеров», а далее присоединение продолжается с учетом критерия информационной взаимозависимости. Для примера на рис. 4.10-б, на первом этапе выбираются «флагман» и «Катер 2», а на втором этапе можно выбрать как «Катер 1», так и «Катер 4», т.к. они оба информационно-взаимозависимы от сформированного промежуточного «Флагмана 1'». Порядок присоединения остальных «катеров» к сформированным промежуточным «флагманам» для задачи со структурой

(рис. 4.10-б), не оказывает принципиального влияния на результат преобразования, потому что:

- вклад «катера» в общую вычислительную трудоемкость и время решения задачи незначителен;

- параметры параллелизма реализаций всех следующих «катеров» будут приводиться к параметрам промежуточных «флагманов»;

- целью метода является поиск рационального, а не оптимального, решения прикладной задачи в пространстве возможных реализаций кадровых структур.

Для примера (рис. 4.10-б), присоединяемый «катер» можно выбрать из нескольких информационно-взаимозависимых на втором и следующих этапах по дополнительному критерию, т.е. максимальной вычислительной трудоемкости присоединяемой подзадачи. В начале преобразования задачи «ф-{к}» все неравенства системы (4.1) упорядочиваются с учетом информационной зависимости, поэтому изменение порядка присоединения означает изменение порядка следования неравенств в системе.

Рассмотрим варианты согласования результатов редукции множества «катеров» с «флагманом». Учитывая обозначения (см. параграф 4.2), будем считать, что «катер», заданный в максимально параллельной форме задачи кортежем своих параметров параллелизма $\text{К} = \langle N_{\text{к}}^{F}, N_{\text{к}}^{\Phi},\ Op_{\text{к}}, \rho_{\text{к}} \rangle$ при редукции с коэффициентом $R$ преобразуется в кортеж $\langle N_{\text{к}^R}^{F}, N_{\text{к}^R}^{\Phi},\ Op_{\text{к}^R}, \rho_{\text{к}^R} \rangle$:

$$\langle N_{\text{к}}^{F}, N_{\text{к}}^{\Phi},\ Op_{\text{к}}, \rho_{\text{к}} \rangle^{R} \rightarrow \frac{\langle N_{\text{к}}^{F}, N_{\text{к}}^{\Phi}, Op_{\text{к}}, \rho_{\text{к}} \rangle}{R} \rightarrow \langle N_{\text{к}^R}^{F}, N_{\text{к}^R}^{\Phi},\ Op_{\text{к}^R}, \rho_{\text{к}^R} \rangle\ . \qquad (4.5)$$

Нижний индекс $k$ в кортеже $\langle N_{k}^{F}, N_{k}^{\Phi},\ Op_{k}, \rho_{k} \rangle$ означает его принадлежность «катеру», а индекс $\text{к}^R$ у результата $\langle N_{\text{к}^R}^{F}, N_{\text{к}^R}^{\Phi},\ Op_{\text{к}^R}, \rho_{\text{к}^R} \rangle$ соответствует значению этого же параметра после редукции.

В результате редукции (4.5), в зависимости от соотношения между абсолютными значениями параметров параллелизма «катера» и коэффициентом $R$, кадровую структуру можно преобразовать к одному из следующих вариантов: структурная, структурно-процедурная, процедурная или последовательная

одноразрядная реализация «катера» с дополнительной задержкой. Рассмотрим каждый из них подробнее.

В первом случае, при структурной реализации «катера» в верхнем октанте пространства возможных реализаций кадровых структур (см. главу 2), изменяются только число слоев и/или итераций, а число устройств, разрядность и интервал обработки данных остаются без изменений. Поэтому диапазоны значений параметров параллелизма «катера» описываются следующими выражениями:

$$\begin{cases} N_{\text{к}^R}^{F} \geq 1, \\ N_{\text{к}^R}^{\Phi} \geq 1, \\ Op_{\text{к}^R} = Op_{\text{к}}, \\ \rho_{\text{к}^R} = \rho_{\text{к}}. \end{cases} \tag{4.6}$$

В кортежной форме редукционное преобразование кадровой структуры «катера» $\langle N_{\text{к}^R}^{F}, N_{\text{к}^R}^{\Phi},\ Op_{\text{к}^R}, \rho_{\text{к}^R}\rangle$ к структурной реализации описывается кортежем $\langle N_{\text{к}^R}^{F}, N_{\text{к}^R}^{\Phi},\ Op_{\text{к}}, \rho_{\text{к}}\rangle$, где исходные параметры $Op_{\text{к}}$ и $\rho_{\text{к}}$ показывают, что при редукции сократилось только число слоев и итераций, а число устройств и разрядность не изменились. Предельным случаем структурной реализации «катера» в верхнем октанте пространства возможных реализаций будет структурная реализация $\langle 1, 1,\ Op_{\text{к}}, \rho_{\text{к}}\rangle$ одного базового подграфа «катера».

Во втором случае, при структурно-процедурной реализации «катера» с помощью м-кадров (см. главу 3), от структурной реализации $\langle 1, 1,\ Op_{\text{к}}, \rho_{\text{к}}\rangle$ происходит переход в нижний октант пространства возможных реализаций кадровых структур (см. главу 2), где число устройств и разрядность сокращаются с учетом коэффициента редукции. Диапазоны значений параметров параллелизма «катера» изменятся следующим образом:

$$\begin{cases} N_{\text{к}^R}^{F} = 1, \\ N_{\text{к}^R}^{\Phi} = 1, \\ 1 < Op_{\text{к}^R} \leq Op_{\text{к}}, \\ 1 < \rho_{\text{к}^R} \leq \rho_{\text{к}}. \end{cases} \tag{4.7}$$

Структурно-процедурная реализация «катера» описывается кортежем $\langle 1, 1,\ Op_{\text{к}^R}, \rho_{\text{к}^R}\rangle$, отражающим сокращение числа слоев и итераций до минимума, а также сокращение числа устройств $Op_{\text{к}}$ и/или разрядности $\rho_{\text{к}}$, но без перехода к процедурным ($Op_{\text{к}^R} > 1$) и/или одноразрядным ($\rho_{\text{к}^R} > 1$) вычислениям.

Третий случай – это процедурная реализация «катера», соответствующая точке с координатой С=1 на оси команд в нижнем октанте пространства возможных реализаций кадровых структур (см. главу 2). Диапазоны значений параметров параллелизма «катера» в этом случае сокращаются еще сильнее:

$$\begin{cases} N_{\text{к}^R}^{F} = 1, \\ N_{\text{к}^R}^{\Phi} = 1, \\ Op_{\text{к}^R} = 1, \\ \rho_{\text{к}^R} = \rho_{\text{к}}. \end{cases} \tag{4.8}$$

Процедурная реализация описывается кортежем $\langle 1, 1,\ 1, \rho_{\text{к}}\rangle$, отражающим выполнение вычислений на одном устройстве без сокращения разрядности ($\rho_{\text{к}^R} = \rho_{\text{к}}$). Отличием задачи «ф-{к}» с несколькими «катерами» от ранее рассмотренного случая «ф»-«к» является большее число «катеров», которые можно реализовать процедурно на процессорных узлах и/или в других целевых архитектурах гибридной вычислительной системы. Поэтому именно задачи типа «ф-{к}» для успешной реализации требуют наличия различных целевых архитектур в гибридной вычислительной системе.

Последний, четвертый случай – это последовательная одноразрядная реализация «катера» с дополнительной задержкой, когда коэффициент редукции превышает произведение всех параметров параллелизма «катера». В этом случае даже при последовательном поразрядном выполнении вычислений нельзя достичь заданного коэффициентом редукции снижения производительности, и требуется применение редукции по интервалу для дополнительной задержки (паузы). Такая реализация описывается кортежем $\langle 1, 1,\ 1, \frac{\rho_{\text{к}}}{r_1}\rangle$, где $r_1 = \left\lceil \frac{R}{N_{\text{к}}^{F} \cdot N_{\text{к}}^{\Phi} \cdot\ Op_{\text{к}}} \right\rceil > \rho_{\text{к}}$, т.е. требуемая редукция процедурной реализации «катера» превышает разрядность выполняемых операций. Более того, как

правило, $r_1 \gg \rho_к$, поэтому даже при последовательной одноразрядной реализации «катера» кортежем $\langle 1, 1, 1, 1 \rangle$ потребуется организовать паузу не менее, чем $\left\lceil \frac{r_1}{\rho_к} \right\rceil$, а интервал обработки данных «катером» составит не менее $I_{кR} = Op_к + r_1$. Этот случай достаточно часто встречается на практике, поэтому он требует дополнительного рассмотрения.

Реализация последовательных одноразрядных вычислений с дополнительным интервалом $I_{K^{доп}} = \left\lceil \frac{r_1}{\rho_к} \right\rceil$ технически сложна, т.к. требует не только специализированных одноразрядных вычислительных элементов и сложных схем учета переноса и коммутации, но и синхронизации одноразрядных вычислений «катера» с многоразрядными вычислениями в остальных фрагментах задачи. Такой вариант организации вычислений неудобен для любой целевой архитектуры гибридной вычислительной системы. Универсальные процессоры и графические ускорители оперируют кратными байтам данными, команды для обработки битовых последовательностей произвольной длины отсутствуют, поэтому одноразрядные вычисления будут реализовываться серией из нескольких команд битовой обработки байтовых данных. Для ПЛИС обработка битовых последовательностей произвольной длины возможна, но необходимы специализированные библиотеки одноразрядных устройств со схемами организации переноса, а коммутационная система усложнится. Наиболее рациональный способ реализации таких вычислений для целевых архитектур гибридной вычислительной системы – это замена комбинации редукций производительности по разрядности и интервалу обработки данных на редукцию по интервалу с кратным увеличением ее коэффициента. Для этого редукционные преобразования, сокращающие аппаратные затраты, останавливаются на процедурной реализации, а дальнейшее сокращение производительности в $r_1$ раз выполняется путем редукции по интервалу обработки данных с коэффициентом $r_1$, без редукции по разрядности. Благодаря сокращению производительности для процедурной реализации «катера»

увеличением интервала обработки данных до значения $r_1$, существенно упростится его практическая реализация как на универсальных процессорах или графических ускорителях, так и на софт-процессорах или в схемотехнических реализациях на ПЛИС.

Процедурная реализация «катера» с дополнительной редукцией по интервалу обработки данных наиболее интересна именно для задач «ф-{к}», когда несколько информационно-связанных «катеров» преобразуются к процедурной реализации с собственной дополнительной паузой $r_i$. Особенность этого случая в том, что несколько «катеров» можно реализовать на одном процессорном узле, если пауза преобразуемого «катера» $r_i$ не меньше, чем число тактов работы следующего «катера» $Op_{k_{i+1}}$. Благодаря объединению или совмещению реализаций «{к}» в одном процессорном узле, ресурсы целевой архитектуры универсальных процессоров гибридной вычислительной системы будут использоваться равномернее, аппаратные затраты сократятся, удельная производительность решения задачи увеличится. Здесь сокращение числа используемых процессорных узлов не самоцель, а способ создания более рационального решения, т.к. благодаря сокращению используемого оборудования можно освободить ресурс для более эффективной реализации других фрагментов или решения других вычислительных задач.

Рассмотрим условия для объединения нескольких процедурных реализаций {к} в одном процессорном узле. Последовательность формирования результатов, вычисляемых информационно-зависимыми «катерами», будем представлять кортежем данных вида $\{p_1, p_2, \ldots, p_{s-1}\}$, где $p_i$ – результат обработки данных катером $k_i$.Каждый из процедурно реализованных «катеров» из {к} обрабатывает элемент входного потока данных за $Op_{k_i}$ тактов, после чего следует пауза длительностью $r_i$, поэтому интервал обработки данных этого «катера» оценивается как $I_{k_i} = Op_{k_i} + r_i$. Если длительность процедурной обработки элемента данных $(\mathrm{i}+1) - \mathrm{м}$ информационно-зависимым «катером» меньше паузы $\mathrm{i}-\mathrm{го}$, т.е. $Op_{k_{i+1}} < r_i$, то его вычисления (при готовых операндах)

реализуемы на этом же процессорном узле за $Op_{k_{i+1}}$ тактов. Для множества «катеров» из {к} это утверждение можно сформулировать в виде теоремы.

Теорема 4.1. **Информационно-зависимые «катера» можно бесконфликтно процедурно реализовать на одном процессорном узле гибридной вычислительной системы, если выполняется условие:**

$$r_i \geq \sum_{j=i+1}^{w} Op_{k_j} , \qquad (4.9)$$

где $w \leq \boldsymbol{S}$.

Доказательство.

Процессорный узел гибридной вычислительной системы, процедурно реализующий «катер» $k_i$ с дополнительной редукцией по интервалу обработки данных, выполняет вычисления за $Op_{k_i}$ тактов и за $r_i$ тактов выполняет команду ***пор***, увеличивающую интервал обработки данных. Команда ***пор*** не влияет на информационное состояние задачи и результаты вычислений. Поэтому замена команды ***пор*** на любую другую команду, не изменяющую вычисленные результаты «катера» $k_i$, не влияет на информационную эквивалентность результатов. Возможный вариант замены команды ***пор*** – это не одиночная информационно-незначимая команда, а ансамбль команд, выполняющих определенные вычисления, не изменяющие результаты «катера» $k_i$. Частный случай такого ансамбля – это команды процедурной реализации информационно-зависимых «катеров», выполняющих вычисления за $Op_{k_{i+1}}, Op_{k_{i+2}}, \ldots, Op_{k_s}$ тактов. При готовности входных данных «катера» $k_{i+1}$, процессорный узел может бесконфликтно выполнить его команды во время паузы $r_i$ при замене $Op_{k_{i+1}}$ команд ***пор*** на вычисления «катера» $k_{i+1}$ при информационной эквивалентности результатов задачи. Оставшуюся паузу $(r_i - Op_{k_{i+1}})$ можно использовать для реализации следующего информационно-зависимого «катера», выполняющего вычисления за $Op_{k_{i+2}}$ тактов. Эту процедуру можно повторять до тех пор, пока пауза в вычислениях неотрицательна и можно бесконфликтно реализовать вычисления $w$ информационно-зависимых «катеров», т.е.

$$(((r_i - Op_{k_{i+1}}) - Op_{k_{i+2}}) - \dots \ -Op_{k_w}) \geq 0 \qquad (4.10)$$

Раскрывая скобки и группируя члены $Op_{k_{i+1}}, Op_{k_{i+2}}, \dots, Op_{k_w}$ в (4.10), получим

$$r_i \ - \sum_{j=i+1}^{w} Op_{k_j} \ \geq \ 0.$$

Перегруппировав неравенство и перенеся $\sum_{j=i+1}^{w} Op_{k_j}$ в его правую часть, получим условие (4.9), что доказывает теорему 4.1

$$r_i \geq \ \sum_{j=i+1}^{w} Op_{k_j}.$$

*Теорема доказана.*

Следствие 4.1.1. Для реализации нескольких информационно-зависимых «катеров», удовлетворяющих условию теоремы 4.1, целесообразно использовать аппаратный ресурс универсальных процессоров гибридной вычислительной системы.

Использование универсальных процессоров гибридной вычислительной системы позволит не только сбалансированно загрузить вычислительный ресурс ГВС, но и освободить аппаратный ресурс других целевых архитектур для возможной более эффективной реализации вычислительно-трудоемких фрагментов «флагман» и «софлагман».

## 4.4. Методы преобразования подзадач для особых случаев информационной зависимости

Информационные зависимости «флагман»-«катер» и «ф-{к}» (см. параграфы 4.2 и 4.3) довольно распространены, но существуют и другие типы задач. Среди них следует особо выделить часто встречающиеся в различных предметных областях задачи с информационной зависимостью «флагман»-«катер»-«софлагман» и «взаимодействующие равные», которые будут рассмотрены в этом параграфе.

### 4.4.1. Информационная зависимость «флагман»-«катер»-«софлагман»

Задачи с информационной зависимостью «флагман»-«катер»-«софлагман» (рис. 4.12) похожи на объединение двух ранее рассмотренных задач «ф»-«к». В отличие от задач «ф»-«к», рассчитанное «катером» значение (как правило, скалярное) используется в вычислениях «софлагманом», который и формирует выходной поток данных. Поэтому в задачах «флагман»-«катер»-«софлагман» можно выделить поток данных «флагмана» и поток данных «софлагмана», которые не обязательно равны, но, в большинстве случаев, сопоставимы.

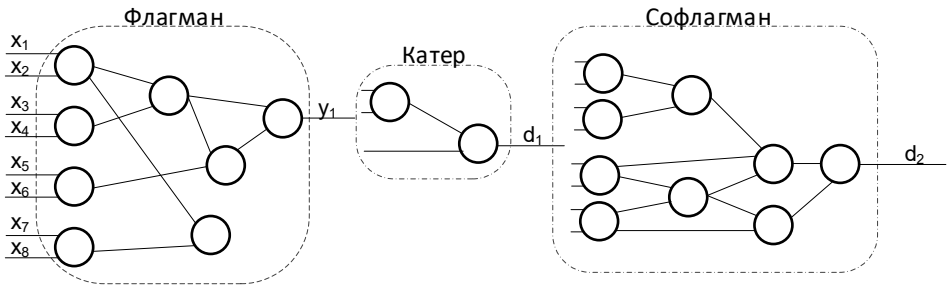



Рис. 4.12. Структура задач «флагман»-«катер»-«софлагман»

Задачи с информационной зависимостью «флагман»-«катер»-«софлагман» встречаются в большинстве прикладных областей. Примерами таких задач в области цифровой обработки сигналов является расчет свертки сигналов [195], в области линейной алгебры – предобуславливание исходной матрицы [196] для улучшения сходимости итерационных методов решения СЛАУ, в области математической статистики – расчет дисперсии потока случайной величины [197].

Задачу «флагман»-«катер»-«софлагман» будем рассматривать на одном из наиболее характерных примеров – расчете свертки сигналов, содержащем связанные информационной зависимостью прямое преобразование Фурье, комплексное умножение его результатов на заданное значение и обратное преобразование Фурье (рис. 4.13) [198].

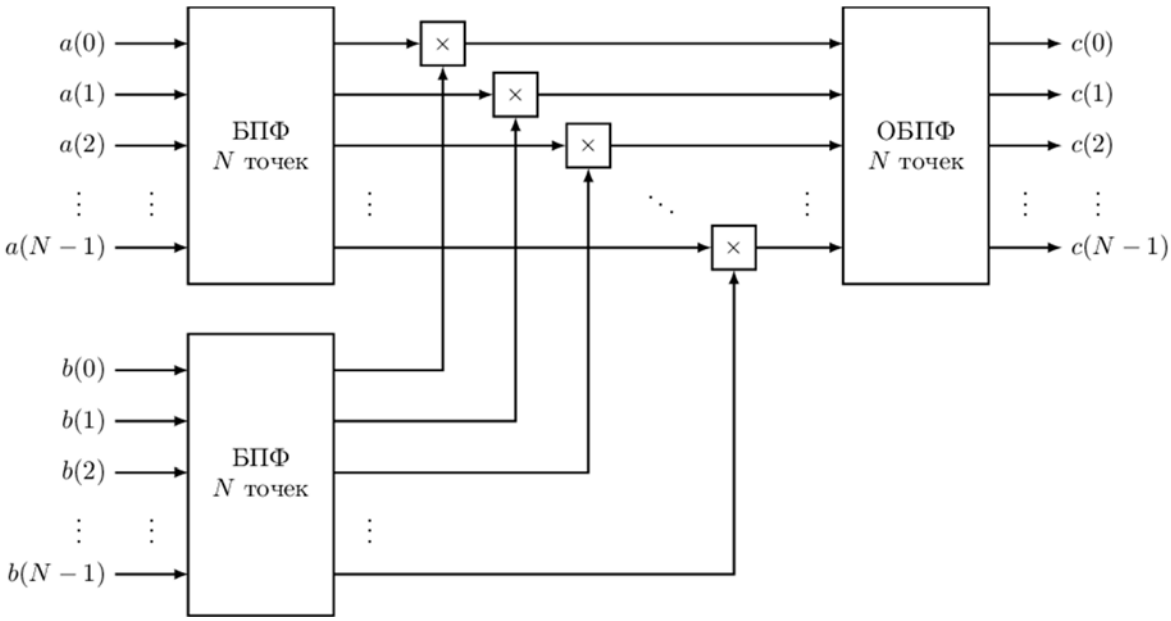



Рис. 4.13. Общая структура вычислений свертки сигналов

Вычислительная трудоемкость прямого и обратного преобразования Фурье примерно одинаковы и существенно превышают трудоемкость комплексного умножения. Поэтому ФП прямого и обратного преобразования Фурье можно отнести к «флагманам», а комплексное умножение является «катером» и структура задачи (рис. 4.12) соответствует информационной зависимости (рис. 4.11).

В отличие от рассмотренных ранее типов информационной зависимости, у «флагмана»-«катера»-«софлагмана» существуют особенности:

- необходимость согласования результатов преобразования кадровых структур «флагмана» и «софлагмана» по темпу обработки данных, чтобы обеспечить близкое к линейному время обработки входных данных, свойственное конвейерным вычислительным структурам. Интервалы обработки данных «флагмана» $I_{\text{Ф}}$ и «софлагмана» $I_{\text{С}}$ должны совпадать, а для рационального решения – быть минимальными: $I_{\text{Ф}} = I_{\text{С}} = 1$;

- для рациональной реализации задачи и «флагман», и «софлагман» должны быть реализованы структурно, т.е. редукционные преобразования этих подзадач должны завершиться в верхнем октанте пространства реализаций кадровых структур (см. главу 2);

- необходимость рационального распределения аппаратного ресурса целевых архитектур ГВС между «флагманом» и «софлагманом» для согласованной структурной реализации с одинаковым интервалом обработки данных.

Для представления задачи в виде системы диофантовых неравенств (4.1) это означает, что первыми тремя неравенствами системы должны быть неравенства, соответствующие фрагментам «флагман», «софлагман» и «катер». При согласовании параметров «флагмана» и «катера», найденные для них значения параметров параллелизма нужно проверять и для «софлагмана».

Если интервалы обработки данных «флагмана» и «софлагмана» не совпадают, то процедура согласования параметров повторяется с согласования параметров «софлагмана» и «катера», а полученные значения проверяются для «флагмана».

При повторном несовпадении параметров можно увеличить коэффициент редукции и/или перейти к структурно-процедурной реализации «флагмана» и «софлагмана» с помощью м-кадров, чтобы обеспечить совпадение интервалов обработки данных для этих фрагментов. Эти утверждения можно сформулировать в виде леммы.

Лемма 4.1. **Несогласованность интервалов обработки данных в информационно-зависимых кадровых структурах «флагмана» и «софлагмана» увеличивает время решения задачи.**

Доказательство.

Время решения задачи «флагман»-«катер»-«софлагман» при структурной конвейерной реализации кадровых структур «флагмана» и «софлагмана» можно представить как

$$T_{\text{з}} = N_{\text{ф}} \cdot I_{\text{ф}} \cdot \tau + T_{\text{к}} + N_{\text{с}} \cdot I_{\text{с}} \cdot \tau \,, \qquad (4.11)$$

где $N_{\text{ф}}$ и $N_{\text{с}}$ – длины потока данных «флагмана» и «софлагмана», соответственно, $\tau$ – длительность такта, $N_{\text{ф}} \cdot I_{\text{ф}} \cdot \tau$ – время обработки потока данных «флагманом», $T_{\text{к}}$ – время вычислений «катера», $N_{\text{с}} \cdot I_{\text{с}} \cdot \tau$ – время обработки

своего потока данных «софлагманом». Латентностью «флагмана» и «софлагмана» можно пренебречь, полагая, что $N_ф$ и $N_с$ существенно ее превышают.

Если интервалы обработки данных «флагмана» $I_Ф$ и «софлагмана» $I_С$ равны $I_Ф = I_С$, то время решения задачи (4.11) можно выразить как

$$T_1 = N_ф \cdot I_ф \cdot \tau + T_к + N_с \cdot I_ф \cdot \tau = (N_ф + N_с) \cdot I_ф \cdot \tau + T_к . \quad (4.12)$$

Если интервалы обработки данных «флагмана» $I_Ф$ и «софлагмана» $I_С$ несогласованы, то больший из них можно представить линейной комбинацией меньшего, например $I_С = I_Ф + \Delta I$, где $\Delta I \geq 1$. Тогда время решения задачи (4.11) примет вид

$$T_2 = N_ф \cdot I_ф \cdot \tau + T_к + N_с \cdot (I_ф + \Delta I) \cdot \tau = (N_ф + N_с) \cdot I_ф \cdot \tau + T_к + N_с \cdot \Delta I \cdot \tau \quad (4.13)$$

Разница во времени решения задачи при несогласованных и согласованных значениях интервалов обработки данных «флагмана» и «софлагмана» составит:

$$T_2 - T_1 = (N_ф + N_с) \cdot I_ф \cdot \tau + T_к + N_с \cdot \Delta I \cdot \tau - (N_ф + N_с) \cdot I_ф \cdot \tau - T_к = N_с \cdot \Delta I \cdot \tau$$

Таким образом, время решения задачи для несогласованных интервалов «флагмана» и «софлагмана» увеличилось на величину $N_с \cdot \Delta I \cdot \tau$, что и требовалось доказать. *Лемма доказана.*

Увеличение времени решения задачи будет прямо пропорционально разнице интервалов обработки $\Delta I$. Полагая $N_ф \cong N_с$ и $I_Ф = 1$, при увеличении интервала обработки данных $\Delta I = 1$, время решения возрастает примерно в 1,5 раза, при $I_Ф > 1$ увеличение времени решения из-за несогласованности интервалов обработки будет меньше 1,5. Помимо увеличения времени решения, несогласованность интервалов обработки может привести к росту накладных аппаратных затрат из-за использования дополнительной памяти для хранения промежуточных результатов обработки и усложнения схемы организации и синхронизации вычислений.

С учетом доказанной леммы можно сформулировать теорему о портации задач с информационной зависимостью «флагман»-«катер»-«софлагман».

Теорема 4.2. **Для рационального решения задач с информационной зависимостью «флагман»-«катер»-«софлагман» при портации на целевые архитектуры гибридной вычислительной системы необходимо обеспечить согласованный интервал обработки данных в кадровых структурах «флагмана» и «софлагмана».**

Доказательство.

Для рационального решения необходимо обеспечить минимизацию времени решения задачи. В соответствии с доказанной леммой 4.1 несогласованность интервалов обработки данных в информационно-зависимых кадровых структурах «флагмана» и «софлагмана» увеличивает время решения задачи. Поэтому время решения задачи с несогласованными интервалами обработки данных не будет минимальным, а само решение – не будет рациональным.

*Теорема доказана.*

Для использования теоремы 4.2 на практике можно получить оценку, когда целесообразно увеличить согласованный интервал обработки данных в обеих информационно-зависимых кадровых структурах «флагмана» и «софлагмана», чтобы сократить время решения задачи.

В соответствии с (4.12) время решения задачи для равных интервалов обработки данных «флагмана» и «софлагмана» $I_{\text{Ф}} = I_{\text{С}}$ можно выразить как

$$T_1 = (N_{\text{ф}} + N_{\text{с}}) \cdot I_{\text{ф}} \cdot \tau + T_{\text{к}} \,. \tag{4.14}$$

Время решения задачи для несогласованных интервалов обработки данных «флагмана» и «софлагмана» $I_{\text{С}} = {I_{\text{Ф}}}^1 + \Delta I$, где $\Delta I \geq 1$, можно выразить как:

$$T_2 = (N_{\text{ф}} + N_{\text{с}}) \cdot {I_{\text{Ф}}}^1 \cdot \tau + T_{\text{к}} + N_{\text{с}} \cdot \Delta I \cdot \tau. \tag{4.13}$$

Пусть $I_{\text{Ф}} = {I_{\text{Ф}}}^1 + \Delta I^1$, т.е. $I_{\text{Ф}} > {I_{\text{Ф}}}^1$. Подставляя это выражение в (4.12), получим

$$T_1 = (N_{\text{ф}} + N_{\text{с}}) \cdot ({I_{\text{Ф}}}^1 + \Delta I^1) \cdot \tau + T_{\text{к}}.$$

Разница времени составит

$$T_2 - T_1 = N_{\text{с}} \cdot (\Delta I - \Delta I^1) \cdot \tau - N_{\text{ф}} \cdot \Delta I^1 \cdot \tau.$$

Чтобы время решения задачи для несогласованных интервалов обработки данных «флагмана» и «софлагмана» $T_2$ превышало время решения задачи для равных интервалов обработки данных $T_1$, должно выполняться условие:

$$N_{\text{с}} \cdot (\Delta I - \Delta I^1) \cdot \tau \ > \ N_{\text{ф}} \cdot \Delta I^1 \cdot \tau.$$

Отсюда

$$\frac{\Delta I}{\Delta I^1} > \frac{N_{\text{ф}}}{N_{\text{с}}} + 1.$$

Полагая $N_{\text{ф}} \ \cong N_{\text{с}}$, получаем $\frac{\Delta I}{\Delta I^1} \geq 2$, т.е. увеличение интервала обработки данных «флагмана» и «софлагмана» до одинакового согласованного значения сократит время решения задачи при условии, что разница между несогласованными интервалами не менее 2.

В отличие от предыдущих типов задач, где основное внимание уделялось только «флагману», здесь преобразование кадровых структур должно обязательно учитывать наличие второй вычислительно-трудоемкой подзадачи. Более того, после редукции «флагмана» следует выполнить преобразование «софлагмана», а не «катера», т.к. преобразование «катера» в данном случае носит вспомогательный характер и незначительно (по сравнению с основными потоками данных) увеличивает латентность решения. Найденные параметры «флагмана» во многом определяют параметры «софлагмана». В качестве «опорного» фрагмента можно выбрать как «флагман», так и «софлагман», а затем реализовать их структурно на доступном аппаратном ресурсе. Если структурные реализации «флагмана» и/или «софлагмана» на доступном ресурсе невозможны, то оба фрагмента нужно реализовать с помощью м-кадров с согласованным интервалом обработки данных. В этом случае время решения задачи кратно возрастет, но все равно будет существенно меньше времени процедурной реализации на процессорных узлах гибридной вычислительной системы.

Теоремы 4.1 и 4.2 позволяют сформулировать условия для рациональной реализации задач «флагман»-«катер»-«софлагман», которые нужно учитывать во время преобразования кадровых структур при движении в пространстве

возможных реализаций. Так, для задачи «флагман»-«катер»-«софлагман», систему диофантовых неравенств (4.1) нужно упорядочить следующим образом: «флагман», «софлагман», «катер», а затем разрешить ее для «флагмана» и «софлагмана». «Катер» будет согласован с помощью операции присоединения. Очевидно, что при наличии нескольких «катеров» между «флагманом» и «софлагманом» схема преобразования существенно не изменится, т.к. после согласования «флагмана» и «софлагмана» «катера» будут присоединяться к согласованному решению с учетом информационной зависимости. Каждый из «катеров» можно реализовать на выделенном универсальном процессоре, а при выполнении условий теоремы 4.1 реализации некоторых «катеров» можно совместить в одном процессорном узле.

### 4.4.2. Информационная зависимость «взаимодействующие равные»

Еще одним распространенным типом информационной зависимости между подзадачами является «взаимодействующие равные» (рис. 4.14). В задачах с информационной зависимостью «взаимодействующие равные» несколько фрагментов примерно одинаковой трудоемкости обрабатывают входной поток данных. Структура задачи «взаимодействующие равные» (рис. 4.13) очень похожа на обычную итерационную структуру прикладных задач, но отличается тем, что базовые подграфы подзадач «взаимодействующие равные» существенно неизоморфны и их нельзя свести друг к другу, а для итерационной структуры все ступени изоморфны. Примерами «взаимодействующих равных» служат задачи символьной обработки, содержащие линейные комбинации известных криптографических преобразований для повышения стойкости. Наиболее известным примером таких линейных комбинаций является алгоритм TripleDES, содержащий три последовательно соединенных преобразования DES с разными ключами, но его можно считать примером «взаимодействующих равных» с определенными

допущениями, т.к. все преобразования DES структурно изоморфны, несмотря на разные ключи. Более характерным примером является целый класс задач с последовательным соединением других алгоритмов (DES, AES, ГОСТ 28147, TEA, Serpent и др.).

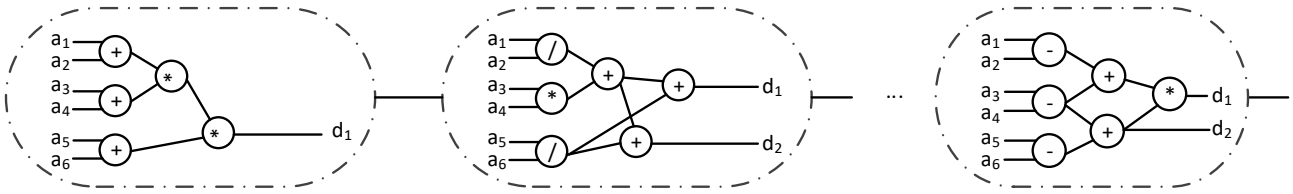



Рис. 4.14. Структура задачи «взаимодействующие равные»

В отличие от рассмотренных ранее задач, у «взаимодействующих равных» есть следующие особенности:

- в структуре задачи нельзя явно выделить наиболее вычислительно трудоемкий фрагмент, который требует структурной реализации;

- результаты преобразования всех кадровых структур должны быть согласованы, чтобы темп обработки данных по возможности совпадал;

- аппаратный ресурс целевых архитектур гибридной вычислительной системы должен быть распределен так, чтобы обеспечить решение задачи с единым согласованным темпом обработки данных;

- при поэтапном присоединении «взаимодействующих равных» наступит момент, когда их объединение сформирует «флагман», а оставшиеся информационно-зависимые подзадачи будут согласовываться с его параметрами параллелизма.

В отличие от предыдущих типов задач, где основное внимание уделялось только «флагману», здесь преобразование кадровых структур выполняется с учетом нескольких подзадач примерно одинаковой вычислительной трудоемкости. Выбор одного из них в качестве «опорного» фрагмента важен, т.к. может существенно повлиять на согласование остальных информационно-зависимых подзадач. В общем случае, когда нет явных критериев выбора

«опорного» фрагмента, можно анализировать несколько вариантов его выбора в рамках итерационного поиска рационального варианта реализации вычислений для целевых архитектур гибридной вычислительной системы. Критерием перехода к следующему варианту анализа может быть изменение интервала обработки данных в кадровой структуре после очередной операции присоединения: если интервал обработки данных увеличился для «флагмана», составленного из нескольких «взаимодействующих равных», то следует проанализировать очередной вариант выбора «опорного» фрагмента. Если интервал обработки данных после присоединения очередной подзадачи не увеличился, то преобразование кадровой структуры можно продолжать.

Каждый из вариантов выбора «опорного» фрагмента соответствует перестановке неравенств в (4.1), когда первым в системе становится очередной «опорный фрагмент», а вторым – информационно-взаимосвязанный с ним фрагмент. Можно оценить общее число возможных вариантов согласования фрагментов задачи «взаимодействующие равные» числом различных пар из $\boldsymbol{S}$ подзадач:

$$C^{s}{}_{2} = \frac{S!}{(s-2)!\cdot 2!} = \frac{S\cdot(S-1)}{2}. \tag{4.12}$$

Как правило, в реальных прикладных задачах число подзадач «взаимодействующие равные» не более 5 и сложно представить задачу, где число подзадач превысит 10, поэтому эти значения можно использовать для оценки граничных случаев числа возможных вариантов. Подставляя эти значения в (4.12), получим 10 различных вариантов согласования для 5 подзадач и 45 вариантов для 10 подзадач. Полученные значения невелики для автоматического или автоматизированного анализа вариантов с помощью программных средств, что позволяет даже в наихудшем случае проанализировать их при выборе разных «опорных» фрагментов и выбрать из них наиболее рациональный по минимальным интервалу обработки данных и времени решения задачи.

## 4.5. Выводы к главе 4

1) Рассмотрено преобразование задач, содержащих несколько связанных информационной зависимостью подзадач. Предложено представление таких задач в виде информационно-зависимых кадровых подструктур – функциональных подграфов, каждый из которых описывается своим «базовым подграфом» ${g^s}_{ijk}$, числом слоев ${L^s}_i$ и итераций ${It^s}_i$, числом устройств ${Q^s}_i$, разрядностью ${\rho^s}_i$ и интервалом обработки данных ${I^s}_i$ ($s = 1..\boldsymbol{S}$, где $\boldsymbol{S}$ – число подзадач). Установлено, что для поиска рационального варианта организации параллельных вычислений на доступном вычислительном ресурсе как глобальной цели портации таких задач требуется согласование степени параллелизма и интервалов обработки данных между подзадачами (подструктурами) при преобразовании максимально параллельной кадровой структуры к целевым архитектурам ГВС.

2) Для задач с несколькими информационно-зависимыми подзадачами предложено формальное представление портации в виде решения системы диофантовых неравенств, связывающих параметры параллелизма кадровых подструктур с коэффициентом редукции. Общее число неравенств в системе соответствует количеству согласуемых подзадач. Каждое неравенство задает локальную цель портации подзадачи, а согласованное решение системы определяет сбалансированное сокращение производительности и аппаратных затрат всех подзадач. Показано, что рациональное решение системы неравенств в целых числах зависит от абсолютных значений параметров параллелизма каждой подзадачи и коэффициента редукции, а сложность поиска решения системы неравенств растет с увеличением их числа.

3) Для сокращения сложности поиска предложен метод решения системы диофантовых неравенств на основе последовательного анализа и присоединения подзадач к промежуточному, уже согласованному, решению. На первом этапе метода рассматриваются два неравенства системы, описывающие информационно-зависимые функциональные подграфы, для каждого из которых

определяются и согласуются между собой параметры кадровых подструктур. На каждом следующем этапе метода к уже разрешенным неравенствам присоединяется одно неравенство с неопределенными параметрами, которые разрешаются относительно уже известных значений. Применение операции присоединения при поиске рационального решения системы диофантовых неравенств позволяет обеспечить согласованное преобразование информационно-зависимых кадровых подструктур с различной степенью параллелизма и существенно сократить сложность и время портации, что доказывает четвертое положение, выдвигаемое для защиты.

4) Выбор неравенств на первом этапе метода предложено выполнять по максимуму числа устройств и/или занимаемому аппаратному ресурсу кадровых подструктур. Для этого определены базовые понятия «флагман» и «катер», на качественном уровне характеризующие значительно различающиеся по числу устройств и/или занимаемому аппаратному ресурсу функциональные подграфы. «Флагманом» является кадровая подструктура с наибольшим числом устройств и/или занимаемым в максимально параллельной форме аппаратным ресурсом, а «катером» - структура с существенно меньшим числом устройств и/или ресурсом.

5) Рассмотрен метод портации на целевые архитектуры ГВС задач, содержащих информационно-зависимые кадровые подструктуры «флагман» и «катер». Показано, что параметры кадровой подструктуры «флагмана» являются определяющими при поиске рационального решения системы диофантовых неравенств. Рассмотрены возможные варианты и разработан алгоритм согласованного преобразования кадровых подструктур «флагмана» и «катера», предложены структурные схемы блоков согласования.

6) Рассмотрена информационная зависимость «флагмана» и множества информационно-зависимых «катеров», являющаяся важным частным случаем зависимости «флагман»-«катер». Определены критерии выбора «катеров» для операции присоединения при последовательном решении системы диофантовых

неравенств и проанализированы возможные варианты решения. Сформулирована и доказана теорема об условиях бесконфликтного совмещения процедурной реализации информационно-зависимых «катеров» на одном процессорном узле ГВС.

7) Рассмотрены особые случаи информационной зависимости подзадач – «флагман»-«катер»-«софлагман» и «взаимодействующие равные», для которых предложены варианты упорядочения системы диофантовых неравенств, позволяющие найти согласованное и сбалансированное решение за меньшее число шагов. Сформулированы и доказаны лемма и теорема о влиянии согласованности интервалов обработки данных в информационно-зависимых кадровых структурах на общее время решения задачи.

# 5. СРЕДСТВА ТЕХНОЛОГИИ РЕСУРСОНЕЗАВИСИМОГО ПРОГРАММИРОВАНИЯ ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

Для апробации совокупности принципов, моделей, положений, теорем, методов и алгоритмов, предложенных в диссертационной работе, были созданы программные средства технологии ресурсонезависимого программирования для целевых архитектур процессоров и ПЛИС, с помощью которых разработанные теоретические основы экспериментально проверены преобразованием кадровых структур и портацией ряда прикладных задач на целевые архитектуры действующих образцов гибридных вычислительных систем.

## 5.1. Структура программных средств технологии ресурсонезависимого программирования

Разработанные методы и алгоритмы преобразования кадровой структуры задач различных типов к доступному аппаратному ресурсу целевых архитектур гибридной вычислительной системы (см. главы 2-4 диссертации) реализованы в программном комплексе. Основным программным компонентом, реализующим преобразования ресурсонезависимой кадровой структуры задач с информационно-зависимыми подзадачами, является программа портации кадровой структуры на гибридные вычислительные системы различных архитектур и конфигураций [199], которую далее будем называть программой «Прокруст». «Прокруст» оценивает доступный аппаратный ресурс целевых архитектур гибридной вычислительной системы, преобразует число слоев и итераций кадровой структуры, согласует интервал и интенсивность обработки данных в разных функциональных подграфа и, если нужна редукция функционального подграфа задачи, рассчитывает требуемый коэффициент редукции и вызывает вспомогательный компонент - программу преобразования фрагментов кадровой структуры на языке COLAMO в микро-кадры [200], которую далее будем называть программой «Щелкунчик».

У программы «Щелкунчик» в разработанных средствах поддержки технологии ресурсонезависимого программирования вспомогательная роль. На основе методов и алгоритмов редукции производительности (см. главу 3 диссертации), он выполняет важное функционально завершенное преобразование – редукцию производительности переданного от «Прокруста» фрагмента кадровой структуры в заданное коэффициентом редукции число раз путем преобразования в микро-кадры.

Разделение функциональных преобразований (см. главы 2-4 диссертации) между двумя программными компонентами обусловлено удобством их применения для преобразования кадровой структуры задач различных типов и логикой построения программного комплекса. Схема функционирования программных компонентов в составе средств технологии ресурсонезависимого программирования представлена на рис. 5.1. Область взаимодействия программ «Прокруст» и «Щелкунчик» выделена пунктирным прямоугольником.

Транслятор языка COLAMO [201, 202, 203] и синтезатор Fire!Constructor [204, 205] разработаны ранее, а система автоматизированного проектирования (САПР) Xilinx Vivado является стандартным средством синтеза конфигурационных файлов компании Xilinx. Программы «Прокруст» и «Щелкунчик» реализованы на языке программирования C# в среде Microsoft Visual Studio и представляют собой исполняемые файлы, запускаемые с параметрами командной строки без графического интерфейса. Разработанная ранее интегрированная среда разработки прикладных программ [206, 207, 208] служит для взаимодействия с пользователем: изменения настроек, определения входных данных и отображения результатов работы.

«Прокруст» и «Щелкунчик» преобразуют входную ресурсонезависимую параллельную программу без участия пользователя в автоматическом режиме, поэтапно адаптируя прикладную задачу к целевым архитектурам и текущей конфигурации гибридной вычислительной системы.

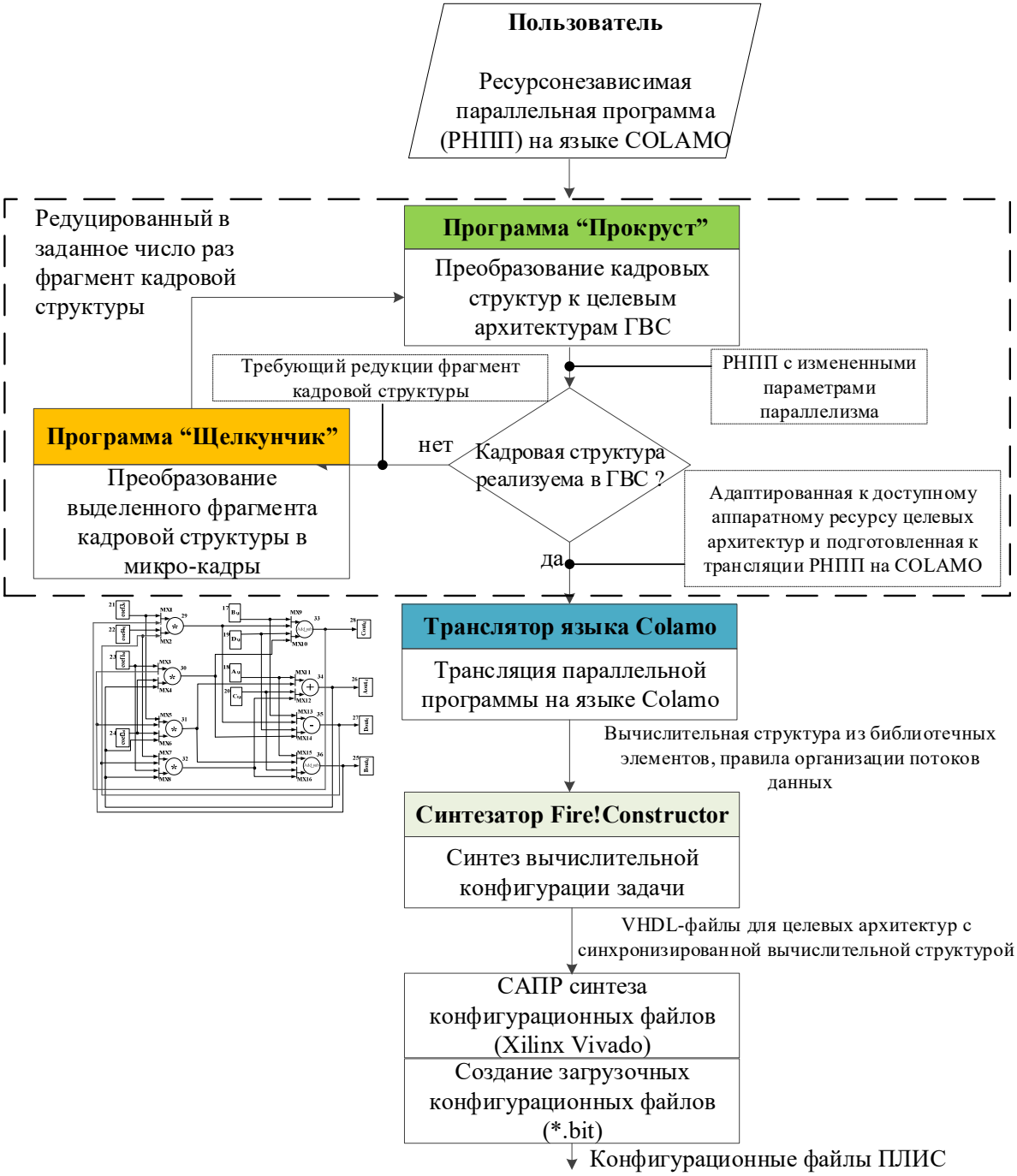



Рис. 5.1. Схема функционирования и взаимодействия программных средств технологии ресурсонезависимого программирования

Программа «Прокруст» преобразует входную ресурсонезависимую параллельную программу (РНПП) в параметризуемую доступным вычислительным ресурсом кадровую форму (см. главу 2 диссертации) с выделенными информационными зависимостями между подзадачами и характеристиками каждой из них: числом слоев и итераций, числом устройств и

занимаемым аппаратным ресурсом, разрядностью и интервалом обработки данных. Параметризация позволяет сохранить синтаксическую и семантическую корректность программы и ее информационную эквивалентность при подстановке фактического значения параметра. Для преобразования параметризуемой ресурсом РНПП к целевым архитектурам ГВС «Прокруст» выполняет следующие функции:

а) определяет аппаратный ресурс для реализации максимально параллельной кадровой структуры ресурсонезависимой параллельной программы;

б) на основе доступного ресурса целевых архитектур ГВС рассчитывает коэффициент редукции производительности и аппаратных затрат;

в) преобразует каждую подзадачу с помощью алгоритмов (см. главу 4 диссертации) и согласовывает все подзадачи с учетом информационных зависимостей. При необходимости вызывает программу «Щелкунчик» для редукции некоторых подзадач;

г) при необходимости согласует интервал обработки данных и оптимизирует полученную кадровую структуру для минимизации интервала обработки данных с помощью преобразований «конвейер в конвейере», «макроконвейер» и «автоподстановка» (см. Приложение 1 диссертации).

«Прокруст» анализирует и изменяет основные параметры параллелизма РНПП – число слоев и итераций кадровой структуры – в верхнем октанте пространства возможных реализаций (рис. 2.6). При нехватке доступного ресурса для реализации даже минимальной кадровой структуры и необходимости сокращения производительности и занимаемого аппаратного ресурса путем перехода в нижний октант пространства возможных реализаций «Прокруст» вызывает «Щелкунчик» для редукции выбранного фрагмента в число раз, заданное коэффициентом редукции. Программа «Щелкунчик», в свою очередь, преобразует полученный фрагмент кадровой структуры к микро-кадрам и возвращает их «Прокрусту» для дальнейших преобразований и согласования с остальными подзадачами (см. главу 4 диссертации).

Итогом преобразований кадровых структур, выполненных «Прокрустом» и «Щелкунчиком», является программа на языке COLAMO, адаптированная к доступному аппаратному ресурсу целевых архитектур гибридной вычислительной системы. Транслятор языка программирования COLAMO [201-203] и синтезатор многокристальных решений Fire!Constructor [204, 205] транслируют полученную программу в конфигурационные файлы целевой архитектуры ПЛИС и управляющие программы целевой процессорной архитектуры.

## 5.2. Описание программы «Прокруст»

«Прокруст» – это основной компонент программных средств ресурсонезависимого программирования, преобразующий кадровую структуру параллельной программы к целевым архитектурам гибридной вычислительной системы. «Прокруст» рассчитывает критический ресурс кадровой структуры, выбирает для его минимизации направление движения в верхнем октанте пространства возможных реализаций и выполняет необходимые преобразования кадровой структуры, а если нужна редукция по разрядности и/или устройствам, то вызывает программу «Щелкунчик». Ресурсонезависимая параллельно-конвейерная программа, поступающая на вход «Прокруста», описывает информационный граф на языке программирования COLAMO и параметризуется доступным аппаратным ресурсом.

РНПП отличается от обычных программ на языке программирования высокого уровня COLAMO. В РНПП любой массив (как мемориальный, так и коммутационный) описывается с помощью параметрической зависимости (рис. 5.2). Здесь константами DP1 и DP2 задается размер векторной составляющей, т.е. число информационно-независимых каналов данных, а длина потоковой определяется по формулам (rows+DP1-1)/DP1 и (columns+DP2-1)/DP2, где rows и columns – исходная длина массива, т.е. число строк и столбцов для матрицы на рис. 5.2. Здесь операция «/» - это целочисленное деление с

округлением вниз, т.к. длина потоковой составляющей должна быть только целым числом. Векторные измерения массивов связаны с потоковой составляющей так, чтобы произведение векторной и потоковой размерностей равнялось длине исходного массива в максимально параллельной кадровой структуре или информационном графе задачи.

```
const rows = (Integer) 100
const columns = (Integer) 90
const DP1 = rows;
const DP2 = columns;
Var matrix : Array Integer [DP1:Vector, (rows+ DP1-1)/DP1:Stream,
                            DP2:Vector, (columns+ DP2-1)/DP2: Stream] Mem
```

Рис. 5.2. Параметрическое описание массивов данных в РНПП на примере задачи решения СЛАУ методом Гаусса

Пример параметрической зависимости для двумерного массива данных (рис. 5.2), соответствующего входной матрице задачи решения системы линейных алгебраических уравнений методом Гаусса (см. параграф 2.1 диссертации), иллюстрирует характер зависимости. Параметрическую зависимость (рис. 5.2) можно использовать для объявления матриц в ресурсонезависимой параллельной форме и в других задачах линейной алгебры на языке программирования COLAMO.

Для сохранения информационной эквивалентности и тождественности РНПП максимально параллельной кадровой структуре длина векторной составляющей DP исходно равна длине размерности массива (DP1 = rows). С уменьшением векторной составляющей DP степень параллелизма сокращается и растет длина потоковой составляющей массива. Предельным значением параметра параллелизма DP является 1, что соответствует конвейерной подаче данных по одному каналу памяти. Для многомерных массивов каждое векторное измерение массива (см. параграфы 2.3-2.4 диссертации) параметризуется уникальным информационно-независимым параметром параллелизма DP.

Изменение значения параметра параллелизма DP для каждого массива в ресурсонезависимой параллельно-конвейерной форме приводит к автоматическому пересчету числа информационно-независимых каналов памяти и длины обрабатываемых потоков данных в каждом канале для кадровых структур и функциональных подграфов всех подзадач. Описанный механизм параметризации обеспечивает движение по слоям и итерациям в пространстве возможных реализаций кадровых структур (см. параграфы 2.4-2.5 диссертации).

Для соблюдения формальной и фактической корректности вычислений параметризация ресурсом применяется не только к описанию массивов данных, но и к циклам, которые делятся на циклы обработки векторной и потоковой составляющих. Все циклы обработки данных (рис. 5.3) должны быть двойными и содержать векторную (обычно с префиксом v для переменной цикла) и потоковую (обычно с префиксом s для переменной цикла) части.

| Цикл исходной программы | Параметрическое описание |
|---|---|
| For i:= 0 To N-1-1 Step 1 Do | **For** vi:= 0 **to** dp1-1 **Step** 1 **Do** |
| For j:= i+1 To N-1 Step 1 Do | **For** si:= 0 **to (N+dp1-1)/dp1** - 1 **Step** 1 Do |
| d:= matrix[j,i] / matrix[i,i]; | **For** vj:= 0 **to** dp2-1 **Step** 1 **Do** |
| | **For** sj:= 0 **to (N+dp2-1)/dp2**-1 **Step** 1 **Do** |
| | d:= matrix[vj,sj,vi,si] / matrix[vi,si,vi,si] |

Рис. 5.3. Параметрическое описание циклов в РНПП задачи решения СЛАУ методом Гаусса

Аналогично параметризуются границы каждого цикла обработки. Векторный цикл задается диапазоном от 0 до максимального значения параметра параллелизма (DP-1). Для определения границ связанного с ним потокового цикла обе границы исходного цикла делятся на параметр параллелизма, соответствующий обрабатываемому измерению массива данных, с округлением вниз (рис. 5.3). Если счетчик вложенного цикла зависит от счетчика цикла верхнего уровня, то разбиение выполняется только для границы, не зависящей от счетчика цикла верхнего уровня, т.е. только для потоковой части.

Слои, итерации и информационные зависимости подзадач и подграфов задаются переменными и их использованием в циклах программы. Путем изменения степени параллелизма ресурсонезависимую кадровую структуру можно преобразовать к информационно-эквивалентной кадровой структуре, требующей меньшего аппаратного ресурса. Для этого в ресурсонезависимой параллельно-конвейерной программе на языке COLAMO можно использовать параметры DP, зависящие от доступного аппаратного ресурса. Для целых значений степени параллелизма DP длина потоковой составляющей пересчитывается автоматически, соблюдается синтаксическая корректность ресурсонезависимой параллельной программы без нарушения правил однократного присваивания и единственной подстановки.

Входными данными «Прокруста» являются файл с расширением ***.clm** с текстом ресуронезависимой параллельной программы на языке высокого уровня COLAMO и файлы описания доступного аппаратного ресурса гибридной вычислительной системы в формате ***.xml**. У программы «Прокруст» нет графического интерфейса, он взаимодействует с пользователем путем установки параметров во вкладках конфигурации (рис. 5.4-а и 5.4-б) интегрированной среды разработки[206-208].

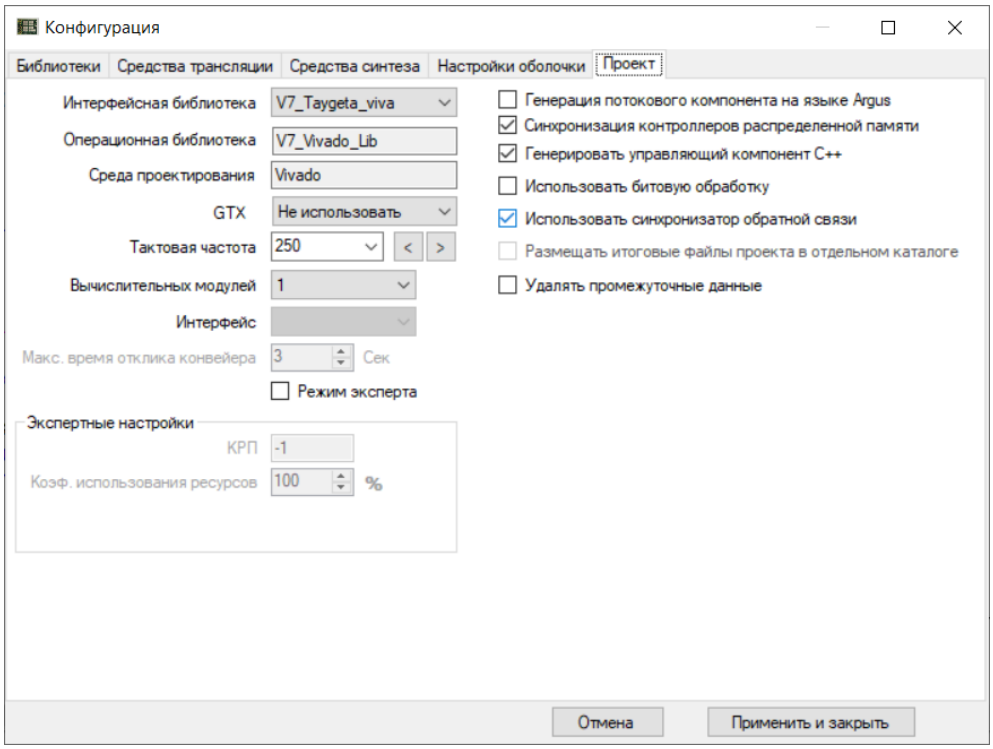
Пользователь может изменять следующие параметры программы «Прокруст»:

- архитектуру и конфигурацию гибридной вычислительной системы, для которой преобразуется кадровая структура задачи (пункт «Интерфейсная библиотека» на рис. 5.4-а);

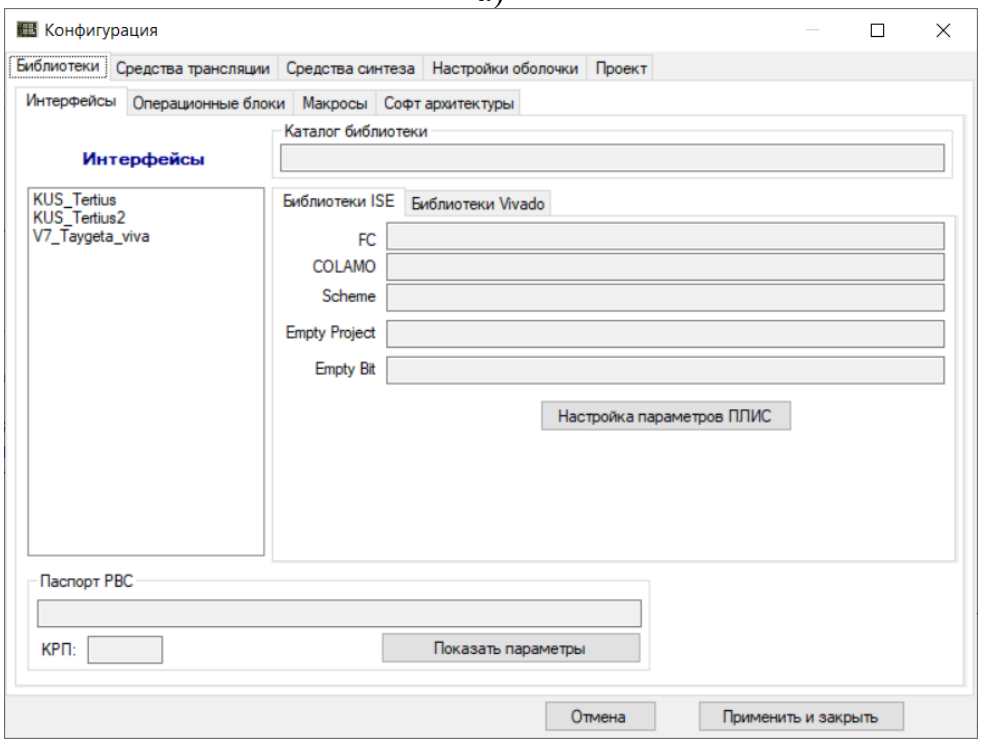
- библиотеку схемотехнической реализаций элементов и IP-ядер, для которой преобразуется кадровая структура задачи (пункт «Операционная библиотека» на рис. 5.4-а);

- доступный аппаратный ресурс гибридной вычислительной системы (пункт «Вычислительных модулей» на рис. 5.4-а);

- интерфейсы обращения к каналам памяти гибридной вычислительной системы, для которой преобразуется кадровая структура задачи (пункт «Интерфейсы» на рис. 5.4-б).




а)




б)

Рис. 5.4. Параметры программы «Прокруст»: а) основные параметры, б) вспомогательные параметры

Остальные параметры на рис. 5.4 не являются параметрами «Прокруста», но влияют на технические характеристики создаваемого решения. Результат работы «Прокруста» – модернизированный текст ресурсонезависимой программы с подобранными значениями параметров параллелизма, отображаемый в окне интегрированной среды и сохраняемый в файле с расширением ***.clm**.

Общий вид основной экранной формы интегрированной среды с загруженной ресурсонезависимой параллельной программой перед вызовом программы «Прокруст» представлен на рис. 5.5. Помимо строки меню, расположенной в верхней части окна, и поля с текстом программы, в нижней части окна расположена панель уведомлений о состоянии процесса преобразования программы к целевым архитектурам вычислительных систем, предупреждениях и найденных ошибках.

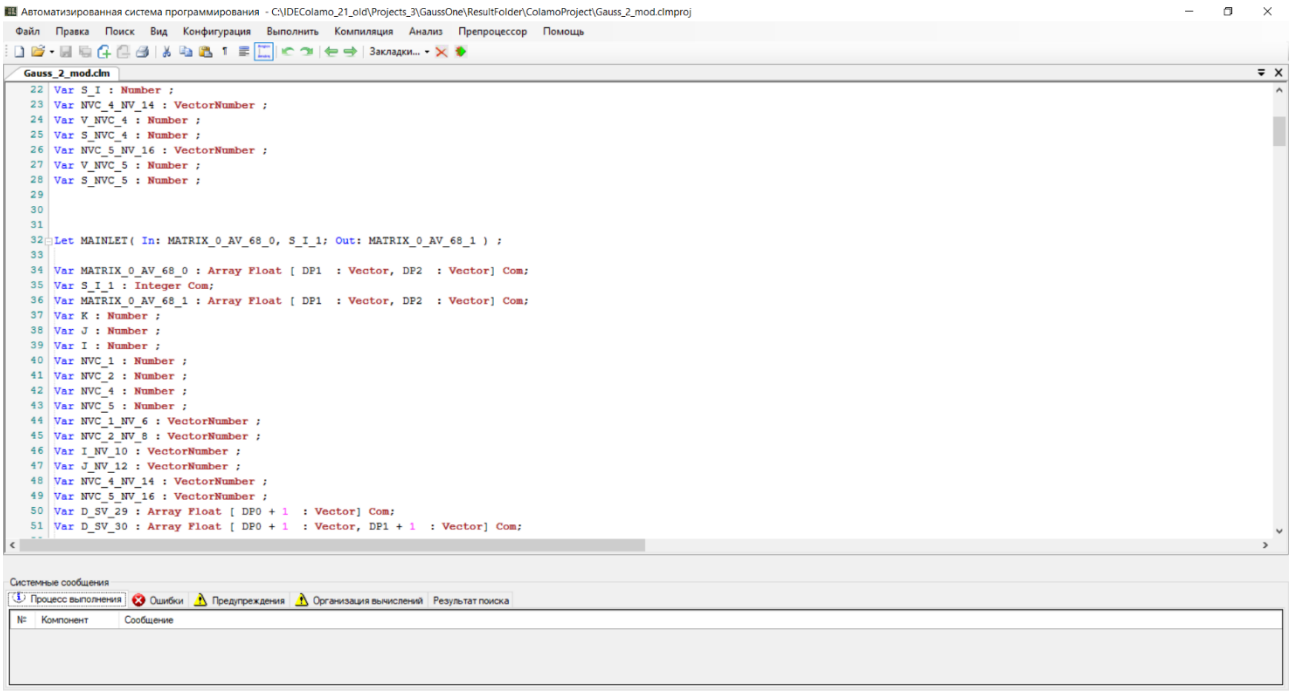



Рис. 5.5. Основная экранная форма интегрированной среды с загруженной ресурсонезависимой параллельной программой перед вызовом «Прокруста»

«Прокруст» рассчитывает параметры реализации всех ФП задачи. Далее с помощью нескольких итераций выбирает наиболее рациональную комбинацию параметров для получения сбалансированного и рационального (по времени) решения, а затем редуцирует кадровые структуры и согласует интервал обработки данных между подзадачами. Программа «Прокруст» вычисляет коэффициенты редукции по аппаратному ресурсу и числу каналов памяти на основе методов и алгоритмов, изложенных в параграфах 3.6-3.7, 4.2-4.4 диссертации, а интервал обработки данных согласует с помощью методов из параграфа 3.7 диссертации.

Во входной РНПП программа «Прокруст» ищет циклы обработки данных, анализирует информационные зависимости и выделяет функциональные подграфы. Например, для задачи решения СЛАУ методом Гаусса для обусловленной матрицы, представленной фрагментом последовательной программы (рис. 5.6), выделяются три подзадачи: прямой ход (представлен тройным циклом), расчет значения последнего неизвестного и обратный ход (цикл по числу строк матрицы).

```
//прямой ход метода Гаусса
#FuncGraph_0
for (i = 0; i < N; i++) {
  for (j = i+1 ; j < N ; j++)
  {
#Graph_0
    d = (matrix[j][i]/matrix[i][i]);
#EndGraph
for (k = i; k < N+1 ; k++)
#Graph_1
      matrix[j][k] = matrix[j][k] - (matrix[i][k] * d);
#EndGraph
   }
}
#EndFuncGraph
….
```

Рис. 5.6. Разметка ФП в последовательной программе решения СЛАУ

Функциональные подграфы в каждой подзадаче выделены разметкой #FuncGraph. Здесь к прямому ходу (#FuncGraph_0) относятся подграф расчета нормировочного коэффициента строки (#Graph_0) и подграф вычитания произведения нормировочного множителя и элемента ведущей строки из элемента столбца матрицы (#Graph_1). Максимально параллельная форма этой задачи, созданная с помощью автоматизированных средств[209, 210] в виде синтаксически корректной программы на языке COLAMO с разметкой функциональных подграфов, представлена на рис. 5.7. Общая структура программы, характерная для циклов (рис. 5.7), сохранилась, а синтаксис и описание информационных зависимостей изменились, согласно требованиям COLAMO. Так, переменная **d** преобразована в двумерный коммутационный массив d_SV_52 для устранения нарушений правил однократного присваивания и единственной подстановки.

```
Cadr main;
….
#FUNCGRAPH 0;
…
  For i:= 0 To N - 1 - 1 Step 1  Do
…
    For j:= i + 1 To N - 1 Step 1  Do
…
     #GRAPH 0  ;
   d_SV_52[i_NV_9,j_NV_11] := matrix_0_AV_88[j,i,i] /matrix_0_AV_88[i,i,i];
     #ENDGRAPH ;
…
    For k:= i To N + 1 - 1 Step 1  Do
     Begin
     #GRAPH 1  ;
      matrix_0_AV_89[j,k,i]:= matrix_0_AV_88[j, k, i] -
                  (matrix_0_AV_88[i, k, i]*d_SV_52[i_NV_9, j_NV_11]);
     #ENDGRAPH ;
#ENDFUNCGRAPH ;
```

Рис. 5.7. Фрагмент РНПП решения СЛАУ методом Гаусса на языке COLAMO

Исходный текст ресурсонезависимой параллельно-конвейерной программы на языке COLAMO (рис. 5.8) и результат его преобразования к целевым архитектурам ГВС с историей трансляции отображаются в окне интегрированной среды (рис. 5.9).

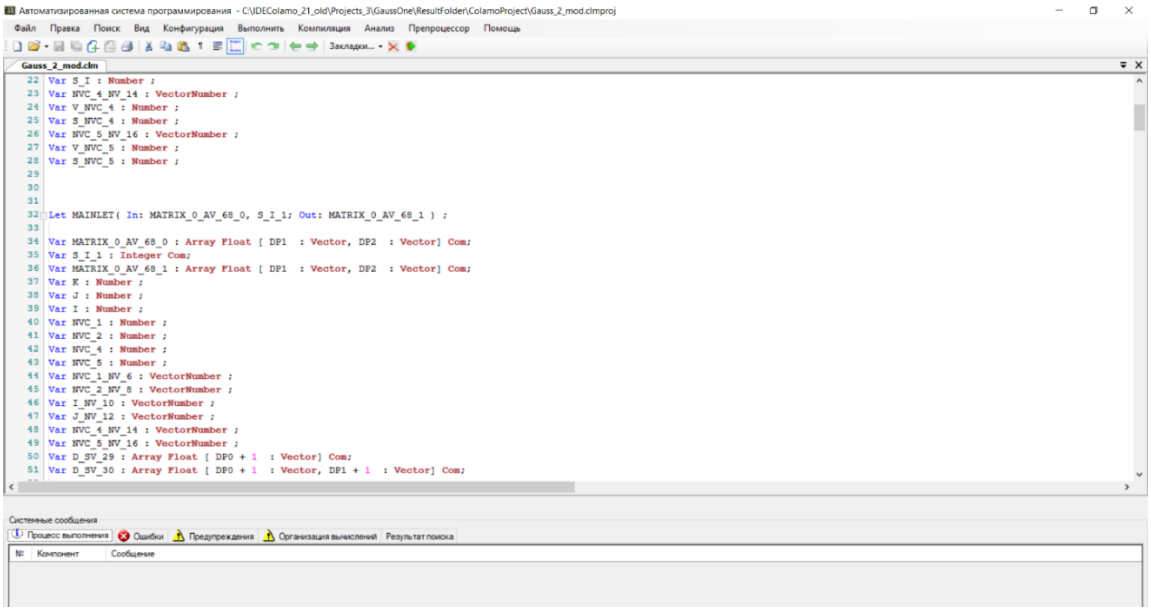



Рис. 5.8. Исходный текст РНПП решения СЛАУ методом Гаусса на языке COLAMO

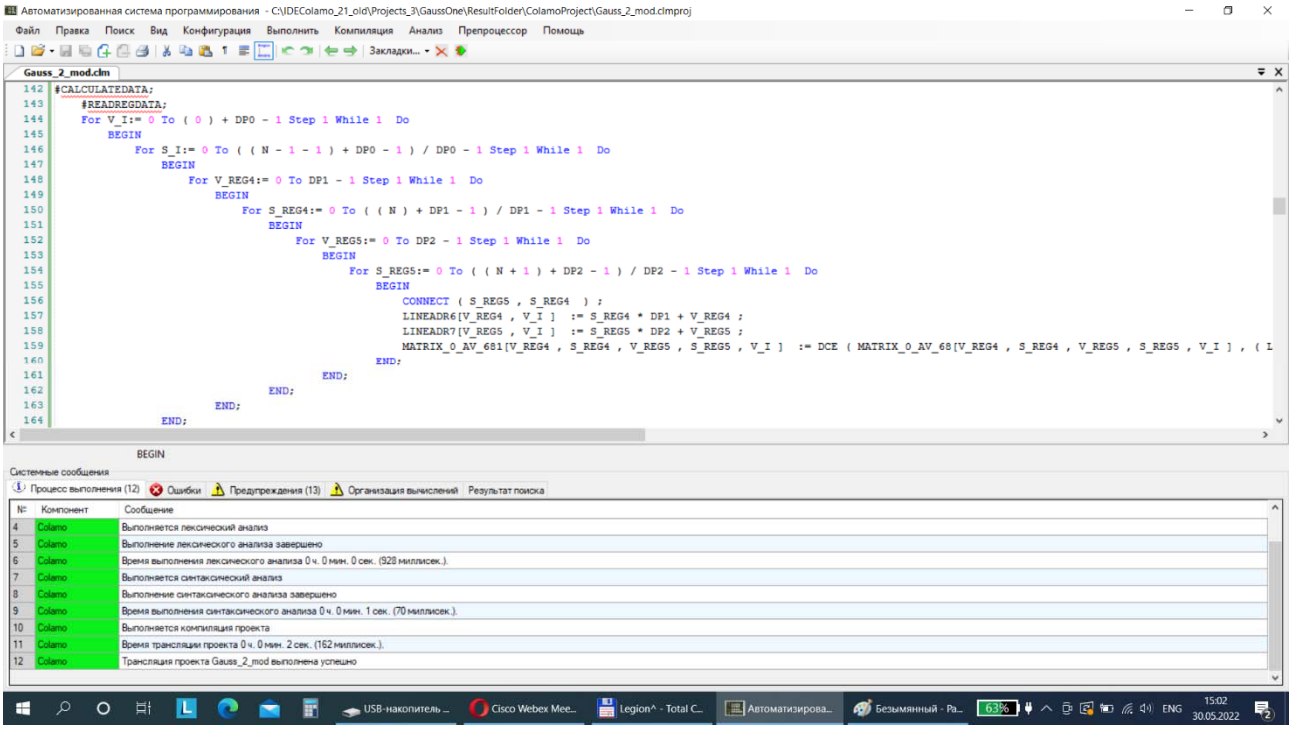



Рис. 5.9. Результат успешного преобразования РНПП решения СЛАУ методом Гаусса к целевым архитектурам ГВС

При необходимости редукции, программа «Прокруст» анализирует параллельную программу и выделяет в ней фрагменты, несбалансированные по темпу обработки данных или по числу входных/выходных каналов фрагменты, которые нужно передать программе «Щелкунчик». «Прокруст» решает, какие из этих фрагментов нужно редуцировать, оценивает коэффициент и тип редукции (по числу устройств или внешних каналов связи) и размечает текст директивами «#REDUCTION OF» для выделения нужного «Щелкунчику» фрагмента. Для полученного от «Щелкунчика» результата редукции подбираются параметры его сбалансированной реализации в общей структуре задачи.

Результат преобразований, выполненных «Прокрустом» – файл с расширением ***.clm** с текстом параллельно-конвейерной программы на языке COLAMO, которая бесконфликтно транслируется на заданный вычислительный ресурс транслятором языка программирования высокого уровня COLAMO [203].

## 5.3. Описание программы «Щелкунчик»

«Щелкунчик» преобразует фрагмент кадровой структуры к микро-кадрам. Он вызывается «Прокрустом» и получает от него фрагмент программы на языке COLAMO с описанием одного из функциональных подграфов задачи, а также рассчитанный коэффициент редукции, требуемой для преобразования фрагмента. Программа «Щелкунчик» выполняет функционально-завершенное преобразование переданного фрагмента кадровой структуры, т.е. путем редукции производительности формирует последовательность микро-кадров, реализуя движение в нижнем октанте пространства возможных реализаций. Редукция производительности функциональных подграфов задачи выполняются на основе методов и алгоритмов из параграфов 3.6-3.7 диссертации.

Входными данными «Щелкунчика» являются созданный «Прокрустом» текст параллельно-конвейерной программы на языке COLAMO с размеченными директивами редукции фрагментами кадровой структуры (файл с расширением ***.clm**) и файл в формате ***.xml** с параметрами требуемого редукционного преобразования. Выходными данными программы «Щелкунчик» является файл

с расширением *.**clm** с текстом программы на COLAMO, содержащим микро-кадры, описанные специализированными программными конструкциями **MLet**.

Входной фрагмент кадровой структуры, полученный от «Прокруста», имеет следующие особенности:

1. Текст программы содержит фрагменты (выделены директивами редукции #Reduction и #EndReduction) с указанием типа и коэффициента выполняемой редукции;
2. Все циклы в переданных фрагментах содержат не более одной итерации, т.к. движение в верхнем октанте завершилось.

В параллельной программе «Щелкунчик» ищет размеченные директивами фрагменты и редуцирует каждый найденный фрагмент в отдельности, а затем генерирует код параллельной программы с последовательностью микро-кадров. Этапы редукции фрагмента параллельной программы:

- анализ редуцируемого фрагмента кадровой структуры;

- преобразование редуцируемого фрагмента к графовой форме, разбиение на непересекающиеся подграфы и назначение вершин подграфов устройствам микро-кадров;

- объединение внешних каналов данных;

- генерация структурной составляющей микро-кадра «MLet» и вызывающего ее подкадра.

Схема взаимодействия и передаваемых данных между программами «Прокруст» и «Щелкунчик» представлена на рис. 5.10.

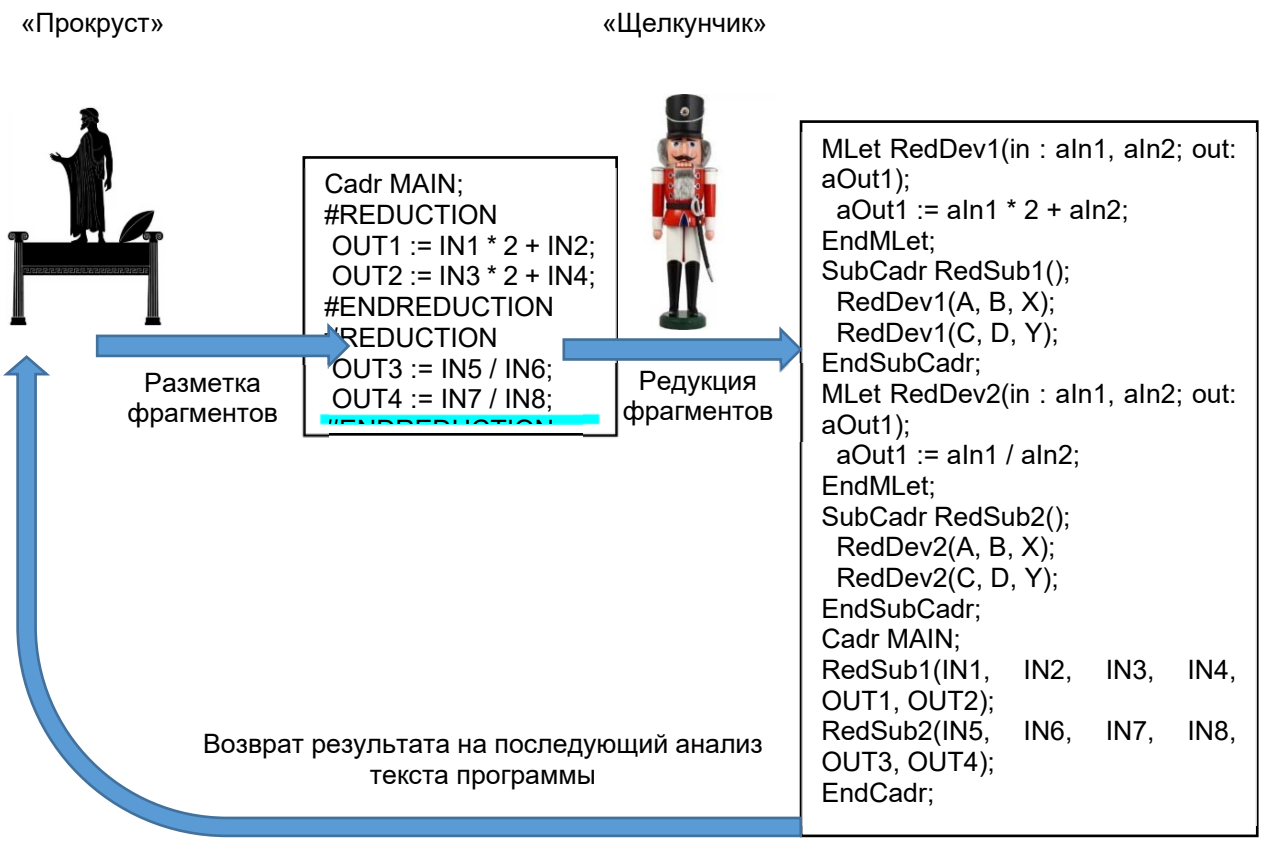



Рис. 5.10. Структурная схема взаимодействия программ «Прокруст» и «Щелкунчик» при синтезе м-кадров

Файл с параметрами редукционного преобразования представляет собой XML-файл, который содержит не только директивы редукции, но и пары информационно-взаимосвязанных мемориальных и коммутационных переменных, структура и формат которых приведены на рис. 5.11. В каждом блоке Link, описывающем информационную зависимость, указываются коммутационная и соответствующая ей мемориальная переменные.

Программа «Щелкунчик», как и «Прокруст», не имеет графического интерфейса, его взаимодействие с пользователем не предполагается, поэтому настраиваемых параметров у него нет. Он взаимодействует с «Прокрустом» с помощью входных и выходных данных. Результат работы «Щелкунчик» – это модернизированный текст ресурсонезависимой программы с синтаксическими конструкциями «MLet» и подкадром с их вызовом, который отображается в окне интегрированной среды и сохраняется в файл с расширением ***.clm**.

**Формат описания**

```
<Information>
  <LinkMemCom>
    <Link>
      <Mem>Имя мем. переменной </Mem>
      <Com> Имя ком. переменной </Com>
    </Link>
    ……————
    <Link>
      <Mem>Имя мем. переменной </Mem>
      <Com> Имя ком. переменной </Com>
    </Link>
  </LinkMemCom>
</Information>
```

**Пример**

```
<LinkMemCom>
  <Link>
    <Mem>INMEM1_MEMIN</Mem>
<Com>INMEM1OUT547DC550</Com>
  </Link>
  <Link>
    <Mem>INMEM1_MEMIN</Mem>
<Com>INMEM1OUT580DC583</Com>
  </Link>
  <Link>
    <Mem>INMEM1_MEMIN</Mem>
<Com>INMEM1OUT613DC616</Com>
  </Link>
  <Link>
    <Mem>INMEM1_MEMIN</Mem>
```

Рис. 5.11. Структура и формат файла с параметрами редукционного преобразования для программы «Щелкунчик»

«Щелкунчик» получает созданную «Прокрустом» параллельную программу на языке COLAMO, текст которой содержит фрагмент кадровой структуры с директивами редукции. Для редукции по устройствам используется директива «#Reduction Of Device N», где N – степень редукции. Для редукции по числу внешних каналов связи используется директива «#Reduction Of Channel N». Возможно совместное применение директив редукции по устройствам и канала в виде «#Reduction Of Device Channel N».

На первом этапе транслятор COLAMO выполняет синтаксический анализ входной параллельной программы для исключения ошибок преобразования, связанных с синтаксически неправильным текстом. На основе результатов успешного синтаксического анализа программы «Щелкунчик» находит директивы, определяет границы, тип и коэффициент редукции преобразуемых фрагментов кадровой структуры. Далее, для каждого редуцируемого фрагмента анализируются входные, выходные и локальные переменные: все переменные, являющиеся аргументами выражений, помещаются во временный список входных переменных, а все переменные, являющиеся результатами выражений, помещаются во временный список выходных переменных. Правило однократного присваивания языка COLAMO гарантирует, что одна и та же

переменная может быть результатом выражения или результирующим параметром функции только один раз. Поэтому с помощью операций пересечения и разности созданных списков определяются входные, локальные и выходные переменные.

Кроме того, сложные составные выражения разбиваются на двухместные операции, которые описываются одной вершиной в графовой форме и среди которых могут быть арифметическая, логическая, библиотечная функции или блок условного перехода «If».

На втором этапе работы программы «Щелкунчик» подготовленный редуцируемый фрагмент преобразуется в графовую форму для дальнейшего разбиения на непересекающиеся подграфы и назначения операций устройствам микро-кадров. Каждая двухместная операция, полученная на первом этапе, заменяется вершиной графа заданного типа (сумматор, умножитель, логическая операция «ИЛИ» и др.), а транзитные вершины, описывающие информационно-незначимые пересылки данных, удаляются из графа с переподключением входных и выходных дуг в смежных вершинах. Согласно алгоритмам (см. пункт 3.1.3 и параграф 3.5 диссертации), определяется состав устройств микро-кадра путем деления исходного числа операций на коэффициент редукции, а затем из всех возможных вариантов разбиения выбирается вариант с наименьшими коммутационными затратами на синхронизацию (см. параграф 3.5 диссертации). Время выполнения разбиения и назначения зависит от числа вершин в исходном графе, которое для графов больших размерностей можно сократить за счёт объединения нескольких вершин в одну без потери качества решения.

Следующим этапом является редукция по внешним каналам, которую можно задать директивой или выполнять из-за сокращения числа устройств для минимизации аппаратных ресурсов на синхронизацию в структурной реализации редуцируемого фрагмента. При объединении операций над разными входными данными в одном устройстве входные каналы объединяются с помощью мультиплексора (рис. 5.12-а), и тогда интервал обработки данных в каждом из каналов увеличится (в рассматриваемом примере до значения $I = 2$).

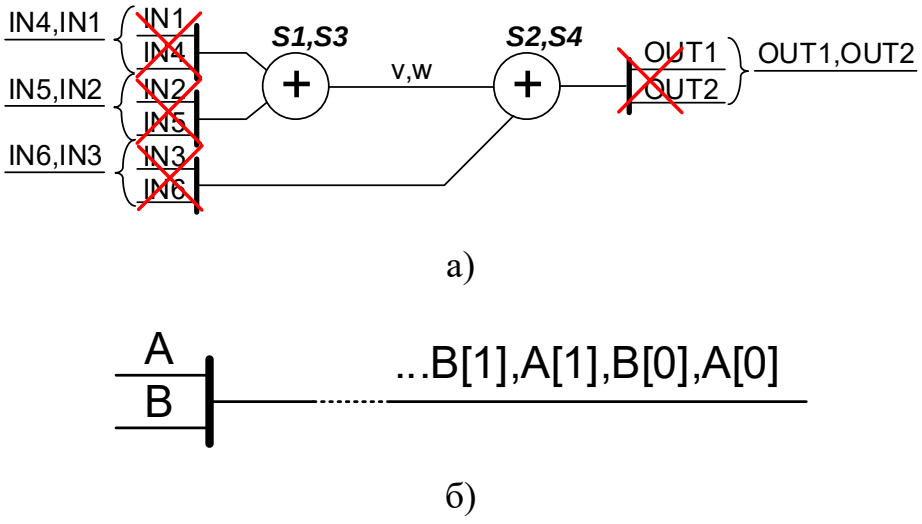



Рис. 5.12. Пример объединения двух каналов программой «Щелкунчик»:

а) исключение входных и выходных каскадов мультиплексоров

б) замена мультиплексора порядком размещения данных в канале

Для сокращения аппаратных ресурсов на синхронизацию объединяемых каналов их можно объединить в один канал с попеременным размещением данных (рис. 5.12-б), работающий с интервалом $I = 1$. Предложенная схема объединения каналов за счет организации порядка следования данных позволяет не только сократить аппаратные затраты на синхронизацию мультиплексорами (рис. 5.12-а), но и сократить число используемых каналов, которые обычно являются критическим ресурсом. Это может улучшить реализацию других функциональных подграфов задачи, что важно для информационно-зависимых подзадач.

На четвертом этапе работы выполняется переход от графовой формы к текстовой, т.е. создаются синтаксические конструкции описания микро-кадров «MLet» и структурного подкадра, вызывающего все микро-кадры «MLet» с корректными параметрами. В результате формируется модернизированная параллельная программа с описанием и порядком вызова микро-кадров и объединёнными каналами, упрощенный пример которой представлен на рис. 5.13.

| **Размеченный директивами редукции входной фрагмент кадровой структуры** | **Результат работы «Щелкунчика»** |
|---|---|
| *#REDUCTION OF DEVICE 2*;<br>*#REDUCTION OF CHANNEL 2*;<br>X := A + B;<br>Y := C + D;<br>*#ENDREDUCTION*;<br>*#ENDREDUCTION*; | **MLet** ReductionDevice<br>(in : aIn1, aIn2; out: aOut1);<br>aOut1 := aIn1 + aIn2;<br>**EndMLet**;<br><br>**SubCadr** ReductionSub<br>(In: A, B, C, D; Out: X, Y);<br>**InterfaceChannel**;<br>AB is <A, B>;<br>CD is <C, D>;<br>**EndInterfaceChannel**;<br><br>ReductionDevice(A, B, X);<br>ReductionDevice(C, D, Y);<br>EndSubCadr; |

Рис. 5.13. Пример преобразования фрагмента кадровой структуры к микро-кадрам программой «Щелкунчик»

Конструкция «MLet» описывает вычислительную структуру нового фрагмента, который получен путем редукции по устройствам и является общим для всех м-кадров. Конструкция «InterfaceChannel» описывает конкатенацию внешних каналов (т.е. внешних переменных по отношению к исходному фрагменту программы), поэтому порядок перечисления входных каналов задает последовательность данных в объединенном канале. Редуцированная программа всегда содержит конструкцию MLet, которая описывает вычислительную структуру редуцированного фрагмента, организацию информационных потоков данных (т.е. информационную зависимость между коммутационными переменными в операторах вызова конструкции MLet) и правила объединения каналов памяти в блоке InterfaceChannel. Пример размеченного директивами редукции фрагмента реальной программы быстрого преобразования Фурье представлен на рис. 5.14-а, а результат редукции этого фрагмента программы, выполненной «Щелкунчиком», – на рис. 5.14-б.

```
for i2 := 0 to N-1  Step 1 do
  Begin
   #REDUCTION OF DEVICE 4;
      ReA1[i2] := ReA[i2] + ReB[i2];
      ImA1[i2] := ImA[i2] + ImB[i2];
      ReCC[i2] := ReA[i2] - ReB[i2];
      ImCC[i2] := ImA[i2] - ImB[i2];

      ReWT[i2] := ReWB[i2] * ReWH[i2] - ImWB[i2] * ImWH[i2];
      ImWT[i2] := ReWB[i2] * ImWH[i2] + ImWB[i2] * ReWH[i2];

      ReB1[i2] := ReCC[i2] * ReWT[i2] - ImCC[i2] * ImWT[i2];
      ImB1[i2] := ImCC[i2] * ReWT[i2] + ReCC[i2] * ImWT[i2];
   #ENDREDUCTION;
end;
```

а)

```
MLet ReductionDevice(in : aIn1, aIn2, aIn3, aIn4, aIn5, aIn6, aIn7, aIn8; out: aOut1, aOut2);
Var aIn1, aIn2, aIn3, aIn4, aIn5, aIn6, aIn7, aIn8, aOut1, aOut2: Integer Com;
Var SPV_0, SPV_1 : Integer Com;
    aOut1 := ADD_SUB(aIn1, aIn2, aIn3);
    SPV_0 := aIn4 * aIn5;
    SPV_1 := aIn6 * aIn7;
   aOut2 := ADD_SUB(SPV_0, SPV_1, aIn8);
EndMLet;

SubCadr ReductionSub(In: ReA, ImA, ReB, ImB, ReWB, ImWB, ReWH, ImWH; Out: ReA1,
ImA1, ReB1, ImB1);
Var ReA, ImA, ReB, ImB, ReWB, ImWB, ReWH, ImWH, ReA1, ImA1, ReB1, ImB1 : Integer
Com;
Var SPV_0, SPV_1 : Integer Com;

  InterfaceChannel;
   IC1 is <ReA, ImA, ReB, ImB>;
   IC2 is <ReWB, ImWB, ReWH, ImWH>;
   IC3 is <ReA1, ImA1, ReB1, ImB1>;
  EndInterfaceChannel;

  ReductionDevice(IC1[0], IC1[2], 0, IC2[0], IC2[2], IC2[1], IC2[3], 0, SPV_0, SPV_1);
  ReductionDevice(IC1[1], IC1[3], 0, IC2[0], IC2[3], IC2[1], IC2[2], 1, SPV_2, SPV_3);
  ReductionDevice(IC1[0], IC1[2], 1, SPV_0, SPV_1, SPV_2, SPV_3, 0, IC3[0], IC3[2]);
  ReductionDevice(IC1[1], IC1[3], 1, SPV_2, SPV_1, SPV_0, SPV_3, 1, IC3[1], IC3[3]);
EndSubCadr;
```

б)

Рис. 5.14. Редукция фрагмента РНПП БПФ по устройствам со степенью 4:

а) размеченный директивами редукции исходный фрагмент РНПП;

б) результат редукции по устройствам со степенью 4 фрагмента РНПП БПФ «Щелкунчиком»

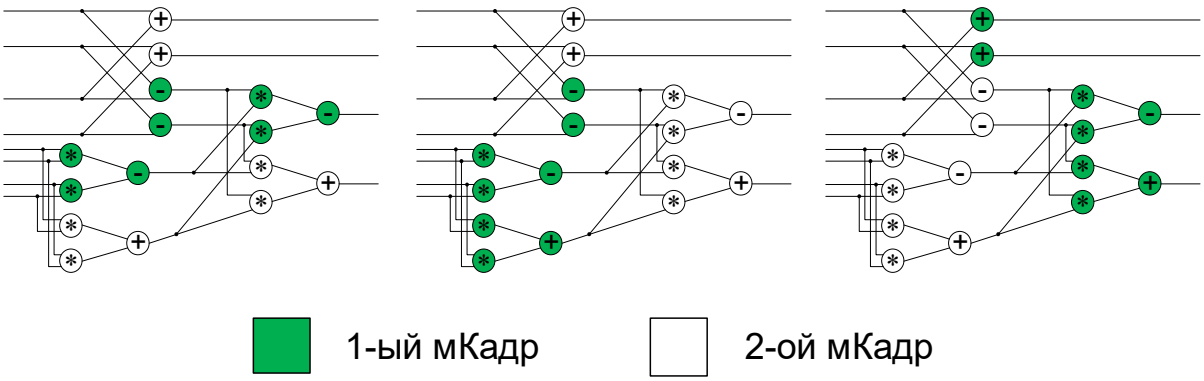
Для алгоритма БПФ программой «Щелкунчик» были получены и отмоделированы в среде разработки Xilinx Vivado следующие варианты редукционных преобразований:

- 3 варианта редукции базовой операции БПФ по устройствам с коэффициентом 2 (рис. 5.15);

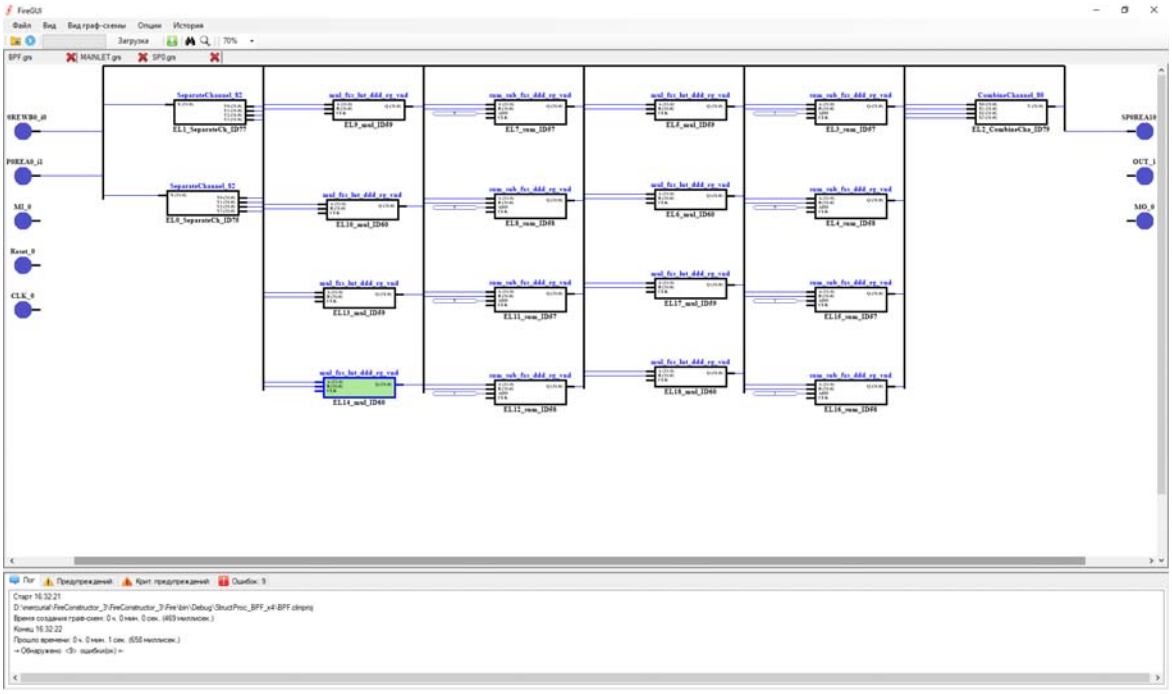
- 1 вариант редукции по устройствам с коэффициентом 4;

- 2 варианта одновременной редукции по каналам и по устройствам с коэффициентом 2;

- варианты редукции по каналам и устройствам с коэффициентами 4 и 8;




а)



б)

Рис. 5.15. Варианты м-кадров при редукции базовой операции БПФ

В результате работы «Щелкунчика» для каждого из перечисленных вариантов были получены тексты программ с м-кадрами на языке программирования COLAMO. Для варианта одновременной редукции по каналам и по устройствам с коэффициентом 4 синтезатором Fire!Constructor создан VHDL-файл, описывающий полученное решение. Этот вариант в виде процедурного блока добавлен в библиотеку схемотехнических элементов для РВС «Тайгета». После успешного синтеза загрузочного bit-файла проверена информационная эквивалентность полученных результатов на РВС «Тайгета». Объем тестируемых входных данных составил 40 000 (по 10 000 данных для каждого массива), а число запусков - не меньше 100. В результате проверки не выявлены несовпадения в полученных и тестовых данных, что свидетельствует о работоспособности предложенных алгоритмов.

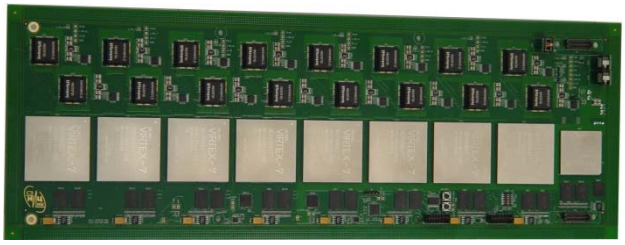
## 5.4. Преобразование прикладных задач к целевым архитектурам ГВС

Апробация алгоритмов преобразования кадровых структур (см. главы 3-4 диссертации) и проверка работоспособности инструментальных средств технологии ресурсонезависимого программирования (программ «Прокруст» и «Щелкунчик») выполнялись при решении ряда прикладных задач путем преобразования ресурсонезависимых программ на языке программирования высокого уровня COLAMO в конфигурационные файлы для целевых архитектур действующих образцов ГВС. Разработанные методы и теоретические положения применимы ко всем рассмотренным целевым архитектурам различных гибридных вычислительных систем, т.к. архитектура вычислительной системы - это один из параметров преобразования кадровых структур. Экспериментальная проверка портации ресурсонезависимых параллельных программ с помощью средств технологии ресурсонезависимого программирования проводилась на доступных автору вычислительных системах, содержащих ПЛИС различных семейств, универсальные микропроцессоры и софт-процессоры. Проверка работоспособности решений прикладных задач, полученных автоматическим

преобразованием компонентами комплекса, проводилась на РВС «Тайгета», «Терциус», «Терциус-3» и ГВС «Атлант-1» и «Атлант-Д».

РВС «Тайгета» [211-215], на которой выполнялись экспериментальные исследования, представляет собой вычислительные модули (ВМ) в конструктивном исполнении высотой 2U, установленные в стандартную 19" вычислительную стойку и соединенные высокоскоростными каналами передачи данных. ВМ «Тайгета» (рис. 5.16-а), содержит 4 ПВМ 8V7-200 (рис. 5.16-б), соединенных быстрыми LVDS-каналами; встроенную управляющую ЭВМ; систему питания; систему управления; систему охлаждения и другие подсистемы.

а)

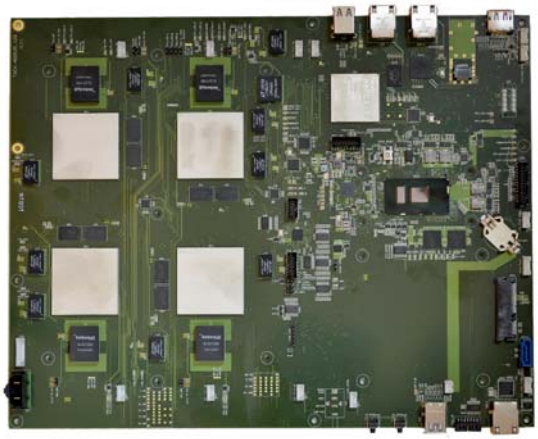


б)

Рис. 5.16. РВС Тайгета (а –со снятой верхней крышкой, б – плата 8V7-200)

ПВМ 8V7-200, лежащая в основе ВМ «Тайгета», представляет собой 20-слойную печатную плату с двухсторонним монтажом элементов, на которой расположены 8 ПЛИС XC7VX485T-1FFG1761, содержащих 48,5 миллионов эквивалентных вентилей, 16 микросхем распределенной памяти SDRAM DDR2 общим объемом 2 Гбайт, интерфейсы LVDS и Ethernet и другие компоненты. Производительность ВМ «Тайгета» составляет 2,66 Тфлопс при обработке 32-разрядных данных с плавающей запятой. Производительность РВС на основе ВМ «Тайгета» при комплектации 18 ВМ «Тайгета» составляет 48 Тфлопс при обработке 32-разрядных данных с плавающей запятой и 23 Тфлопс при обработке 64-разрядных данных с плавающей запятой.

Второй аппаратной платформой, использовавшейся для сравнения полученных результатов, служила РВС «Терциус», представляющая собой настольный реконфигурируемый компьютер (НРК) [212]. В данном НРК поле ПЛИС выполнено не на отдельной плате, как в ПВМ 8V7-200, а интегрировано с материнской платой (рис. 5.17). Печатная плата НРК «Терциус» (рис. 5.17-б), содержит 4 взаимосвязанные ПЛИС фирмы Xilinx семейства Kintex UltraScale XCKU095 емкостью 100 млн. эквивалентных логических вентилей каждая, которые также используются в системах с жидкостным охлаждением[216]. Все ПЛИС соединены между собой в кольцо с помощью LVDS- и GTY/GTH-каналов и к каждой ПЛИС подключено по два модуля динамической памяти емкостью 1 Гбайт каждый, общий объем памяти 8 Гб. Производительность вычислительного поля составляет 2,5 Тфлопс на рабочей частоте 550 МГц. Для управления и конфигурирования ПЛИС, подготовки и отладки параллельной программы используется процессор Intel Core I5 6300U, установленный на плате НРК.



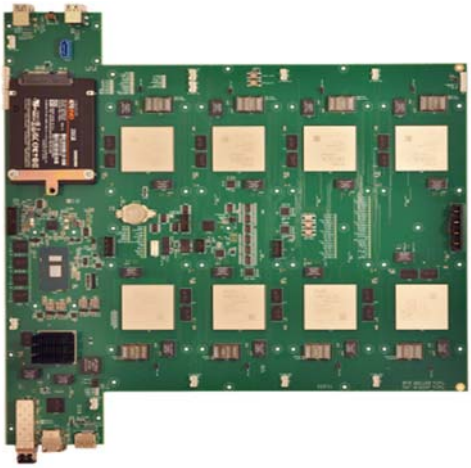
а) Печатная плата «Терциус»

б) РВС «Терциус» со снятой верхней крышкой

Рис. 5.17. Настольный реконфигурируемый компьютер «Терциус»

Третьей аппаратной платформой для проверки работоспособности разработанных программных средств был НРК «Терциус-3» (рис. 5.18), содержащий вдвое больший вычислительный ресурс и ПЛИС другого семейства - 8 ПЛИС Xilinx семейства Virtex UltraScale XCVU095-1FFVB1760C емкостью

100 млн. эквивалентных логических вентилей каждая. Производительность НРК «Терциус-3» составляет 5,6 Тфлопс.



а) Печатная плата НРК «Терциус-3»

б) НРК «Терциус-3» со снятой верхней крышкой

Рис. 5.18. Настольный реконфигурируемый компьютер «Терциус-3»

Для проверки совместной работы целевых архитектур ПЛИС и процессоров при решении прикладных задач использовались действующие образцы гибридных вычислительных систем «Атлант-1» и «Атлант-Д».

ГВС «Атлант-10» (рис. 5.19) предназначена для обработки сетевых пакетов IP с пропускной способностью 1-10 Гбит/сек в режиме реального времени, при которой используются как вычислительные ресурсы ПЛИС, так и возможности универсального процессора Intel и встроенных процессоров с архитектурой ARM.

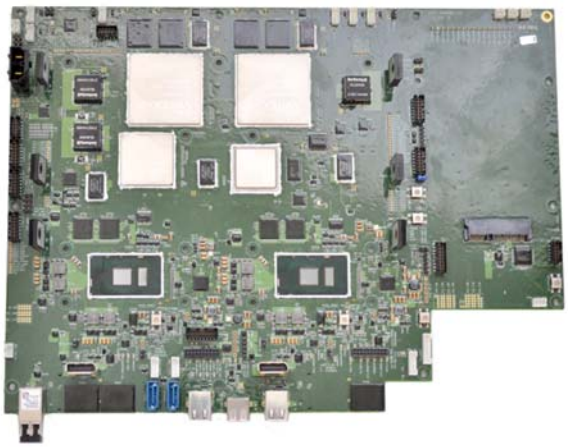
а) Печатная плата ГВС «Атлант-10»

б) ГВС «Атлант-10» с установленной верхней крышкой

Рис. 5.19. Гибридная вычислительная система «Атлант-10»

ГВС «Атлант-10» содержит 4 ПЛИС Xilinx семейства Kintex-7 XC7K325T емкостью 326 тыс. логических ячеек, одну управляющую «систему на кристалле» Xilinx семейства Zynq-7000 XC7Z030 логической ёмкостью 125 тыс. логических ячеек с двухъядерным процессором ARM Cortex-A9 MPCore и SIMD-сопроцессором NEON, а также 2 процессора Intel Core i5-6300U, установленных на печатной плате и объединенных в единое вычислительное поле. Производительность ГВС «Атлант-10» составляет 7 Тфлопс при решении специализированных задач при потребляемой мощности 290 Вт.

Второй аппаратной платформой ГВС для сравнения полученных результатов служила ГВС «Атлант-Д» (рис. 5.20), предназначенная для высокоскоростной специализированной обработки пакетов данных с пропускной способностью 100 Мбит/сек и содержащая 2 ПЛИС Xilinx семейства Kintex-7 XC7K160T емкостью 162 тыс. логических ячеек и 3 специализированных «системы на кристалле» Xilinx семейства Zynq-7000 XC7Z030, установленных на печатной плате и объединенных в единое вычислительное поле.



а) Печатная плата ГВС «Атлант-Д»

б) ГВС «Атлант-Д» с установленной верхней крышкой

Рис. 5.20. Гибридная вычислительная система «Атлант-Д»

Экспериментальная проверка разработанных теоретических основ выполнялась преобразованием кадровых структур ресурсонезависимых параллельных программ и их портацией на целевые архитектуры рассмотренных

реконфигурируемых и гибридных вычислительных систем программными средствами технологии ресурсонезависимого программирования (ТРНП) без участия разработчика. Работоспособность результатов портации, полученных средствами ТРНП, для каждой задачи проверялась запуском на целевой архитектуре выбранной аппаратной платформы.

При портации прикладной задачи для каждой аппаратной платформы ГВС измерялось время преобразования кадровой структуры и рассчитывалось время портации задачи как сумма времени преобразования и времени синтеза загрузочного модуля целевой архитектуры. Время преобразования кадровой структуры разработанными средствами ТРНП сравнивалось с оценкой усредненного времени преобразования задачи прикладным программистом, которое для всех аппаратных платформ было принято равным двум 8-часовым рабочим дням, выраженным в секундах: $2 \cdot 8 \cdot 60 \cdot 60 = 57\,600$ с. Время синтеза загрузочного конфигурационного файла ПЛИС (проекта) одинаково для средств ТРНП и решений прикладных программистов, поскольку определяется рабочей частотой, логической емкостью и уровнем заполнения кристалла ПЛИС. Для всех портируемых прикладных задач рабочая частота составляла 250 МГц, а уровень заполнения кристалла составлял не менее 90%, поэтом время синтеза проекта определялось логической емкостью ПЛИС аппаратных платформ и составляло 3 часа для РВС «Тайгета», 7 часов для РВС «Терциус» и «Терциус-3», 1 час для ГВС «Атлант-10» и 3 часа для ГВС «Атлант-Д». Использовавшиеся для оценки эффективности средств ТРНП показатели времени преобразования и портации РНПП прикладными программистами представлены в таблице 5.1.

Таблица 5.1. Усредненные оценки времени преобразования и портации задач на различные целевые архитектуры прикладными программистами

| **Время прикладных программистов при** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| преобразовании РНПП, с | 57600 | 57600 | 57600 | 57600 | 57600 |
| портации РНПП, с | 68400 | 82800 | 82800 | 61200 | 68400 |

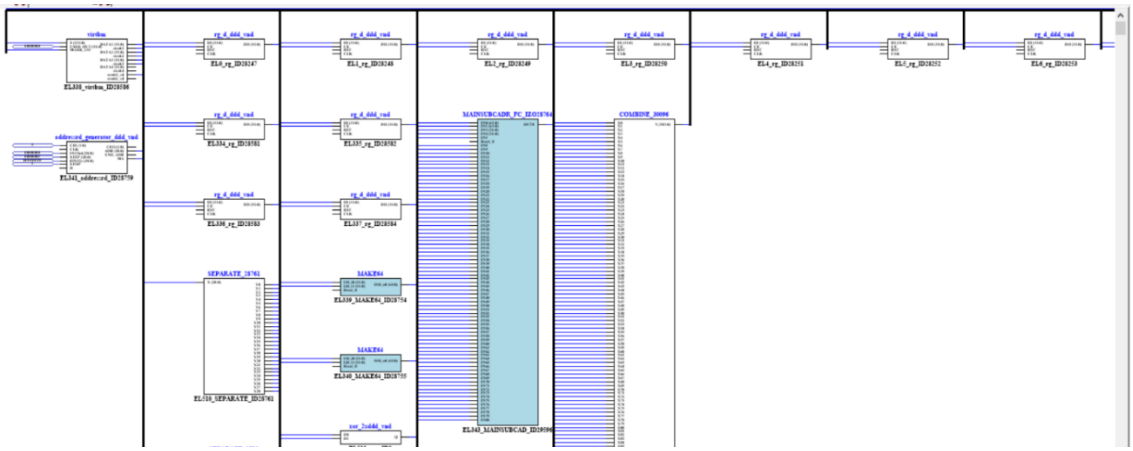
Преобразование и портация задач выполнялись на персональном компьютере с процессором Intel (R) Core (TM) i7-8750H @2,2 ГГц и 16 Гб оперативной памяти, работающим под управлением операционной системы Windows 10 Pro. Эффективность применения средств ТРНП определялась по двум основным критериям: выигрыш во времени (при преобразовании кадровой структуры и портации задачи) и достигнутый уровень реальной производительности по сравнению с эталонным решением, полученным прикладными программистами. Выигрыш во времени преобразования и портации прикладной задачи рассчитывался как отношение времени, затраченного прикладными программистами, к времени работы средств ТРНП. Достигнутый при портации задачи уровень реальной производительности оценивался как отношение количества подграфов в решениях, полученных средствами ТРНП, к числу подграфов в решении прикладных программистов. Количество реализованных подграфов в параллельно-конвейерном решении для целевой архитектуры ПЛИС позволяет достаточно точно оценить реальную производительность задачи.

Результаты портации прикладных задач с рассмотренными (см. главу 4 диссертации) типами информационной зависимости: задачи с единственным базовым подграфом, задачи «флагман»-«катер», задачи «флагман»-{катер}, задачи «флагман»-«катер»-«софлагман» и задачи «взаимодействующие равные» представлены в следующих пунктах настоящего параграфа и в Приложении 3.

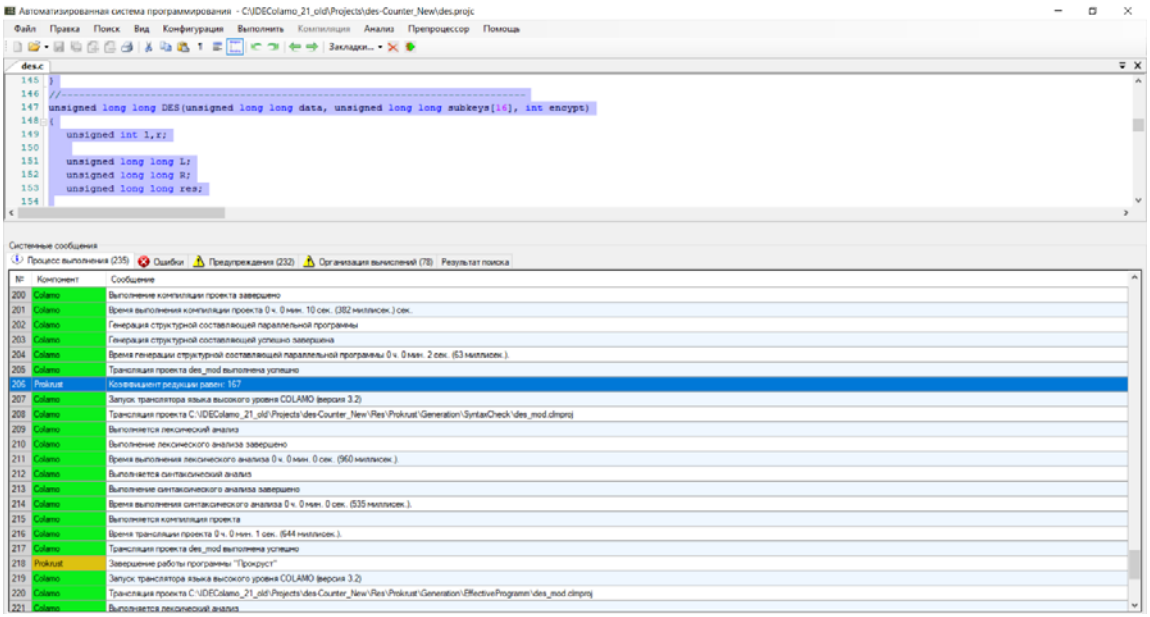
### 5.4.1 Задачи с единственным информационно-независимым базовым подграфом

Прикладные задачи, содержащие одну подзадачу с единственным базовым подграфом, долго преобладали среди задач, решаемых на РВС. К таким задачам относятся алгоритмы символьной обработки и задачи линейной алгебры. В задачах символьной обработки базовые подграфы, как правило, информационно независимы, а в задачах линейной алгебры преобладает прямая информационная (или итерационная) зависимость. Портация задач символьной обработки

средствами ТРНП выполнялась для 5 симметричных блочных алгоритмов с информационно-независимыми базовыми подграфами: DES [217, 218] (рис. 5.21), ГОСТ 28147-89 [217], AES [218] и хеш-функций с информационно-независимыми базовыми подграфами MD5 [217, 218] и SHA-1 [217, 218]. Представление задачи символьной обработки DES в окнах программных компонентов разработанных средств ТРНП показано на рис. 5.21.



а)




б)

Рис. 5.21. Представление задачи DES в синтезаторе Fire!Constructor

а) общий вид задачи; б) ход портации РНПП в интегрированной среде

Результаты экспериментальной проверки времени и эффективности портации задачи DES средствами ТРНП и прикладными программистами на

целевые архитектуры рассмотренных аппаратных платформ приведены в табл. 5.2.

Таблица 5.2. Результаты экспериментальной проверки портации задачи DES

| **Портация прикладной задачи DES** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 36,62 | 38,58 | 39,02 | 37,04 | 39,36 |
| Время портации РНПП средствами ТРНП, с | 10836,62 | 25238,58 | 25239,02 | 10837,04 | 10839,36 |
| Выигрыш преобразование (портация) РНПП | 1573,08 (6,31) | 1492,96 (3,28) | 1476,05 (3,28) | 1554,95 (5,65) | 1463,30 (6,31) |
| Уровень реальной производительности (число БП РНПП/ пр. прогр.) | **0,63** (151/240) | **0,65** (267/410) | **0,65** (267/410) | **0,63** (50/80) | **0,65** (101/155) |

Как видно из данных таблицы 5.2, преобразование кадровой структуры задачи DES средствами ТРНП выполняется существенно (на 2-3 десятичных порядка) быстрее, чем преобразование прикладным программистом (см. таблицу 5.1), что является ожидаемым эффектом автоматизации. Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3,28 до 6,31 раз. Снижение выигрыша во времени портации по сравнению с временем преобразования РНПП объясняется заметным вкладом времени синтеза проекта, зависящего от логической емкости ПЛИС и одинакового для средств ТРНП и прикладных программистов. Незначительное изменение времени преобразования кадровой структуры одной и той же задачи для разных аппаратных платформ обусловлено сравнительно небольшим количеством вариантов, рассматриваемых при выборе рационального решения, что экспериментально подтверждает третье и четвертое положения, выдвигаемые для защиты. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,63, что соответствует уровню создаваемых прикладными программистами решений задач на РВС и превышает уровень реальной

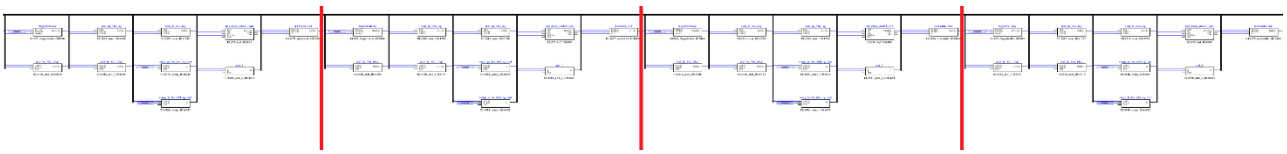
производительности, обеспечиваемый рассмотренными в первой главе HLS-компиляторами.

Результаты портации других задач символьной обработки с информационно-независимыми подграфами (ГОСТ 28147-89, AES, хеш-функций MD5 и SHA-1), приведенные в Приложении 3, демонстрируют схожие с задачей DES зависимости. Так, время преобразования РНПП одной и той же задачи при портации на различные аппаратные платформы меняется незначительно и не зависит от величины кристалла ПЛИС, что подтверждает сделанные в главах 2-4 теоретические оценки об ограниченном и сравнительно небольшом числе преобразований кадровой структуры при поиске рационального варианта ее реализации в целевых архитектурах. Существенный выигрыш во времени преобразования РНПП (на 2-3 десятичных порядка) обусловлен автоматическим ее преобразованием с помощью разработанных алгоритмов, основой которых стали предложенные в главах 2-3 теоретические основы ресурсонезависимого программирования. При синтезе загрузочного модуля выигрыш сокращается до 3-6 раз, но остается ощутимым, особенно для получения корректно работающего загрузочного модуля, требующего неоднократных корректировок РНПП и стадий синтеза конфигурационных файлов ПЛИС. Представленные в таблице 5.2 и Приложении 3 результаты портации задач символьной обработки с информационно-независимыми подграфами подтверждены актами о внедрении результатов в ФГУП «НИИ Квант», НИИ МВС ЮФУ, НИЦ СЭ и НК.

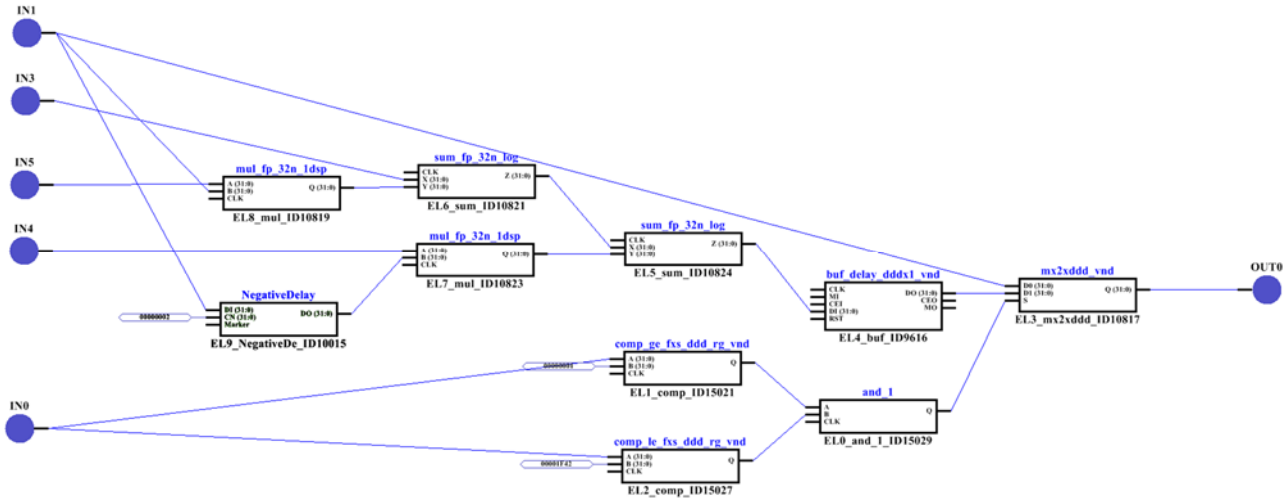
Полученные результаты портации подтверждают теоретические положения (см. главы 2-3 диссертации) ресурсонезависимого программирования, методы и алгоритмы преобразования кадровых структур в пространстве возможных реализаций. Сокращение времени портации задач с информационно-независимыми подграфами с обеспечением заданного уровня реальной производительности экспериментально подтверждает достижение цели, поставленной в диссертационной работе.

### 5.4.2 Задачи с единственным информационно-зависимым базовым подграфом

Для алгоритмов с изоморфными информационно-зависимыми базовыми подграфами портация средствами ТРНП выполнялась для трех задач линейной алгебры: решения СЛАУ методом Якоби [219, 220], преобразования матрицы к верхнетреугольному виду (LU-разложения) [219, 220] и решения СЛАУ методом Гаусса-Зейделя [220, 221]. Результаты портации решения СЛАУ методом Якоби как наиболее характерного представителя задач с информационно-зависимыми базовыми подграфами представлены ниже, а данные для других алгоритмов линейной алгебры приведены в Приложении 3. Представление задачи решения СЛАУ методом Якоби с информационно-зависимыми базовыми подграфами в окнах программных компонентов средств ТРНП показано на рис. 5.22.



а)




б)

Рис. 5.22. Представление задачи решения СЛАУ методом Якоби в синтезаторе Fire!Constructor: а) фрагмент общего вида задачи из 4 итераций; б) представление базового подграфа одной итерации;

В отличие от задач символьной обработки с информационно-независимыми базовыми подграфами, для итерационных алгоритмов линейной

алгебры и других предметных областей, характерно информационно-зависимое построение кадровой структуры.

Результаты экспериментальной проверки времени и эффективности портации задачи решения СЛАУ методом Якоби для матриц размером 8000×8000 средствами ТРНП и прикладными программистами на целевые архитектуры рассмотренных аппаратных платформ приведены в табл. 5.3.

Таблица 5.3. Результаты экспериментальной проверки портации задачи решения СЛАУ методом Якоби

| **Портация прикладной задачи решения СЛАУ методом Якоби** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 2629,90 | 712,87 | 714,02 | 327,53 | 332,12 |
| Время портации РНПП средствами ТРНП, с | 13429,90 | 25912,87 | 25914,02 | 11127,53 | 11132,12 |
| Выигрыш преобразование (портация) РНПП | 21,90 (5,1) | 80,8 (3,2) | 80,67 (3,2) | 175,86 (5,5) | 173,43 (6,14) |
| Уровень реальной производительности (число БП РНПП/ пр. прогр.) | **0,86** (1242/1450) | **0,85** (1975/2320) | **0,85** (1975/2320) | **0,86** (415/483) | **0,84** (810/960) |

Как видно из данных таблицы 5.3, преобразование кадровой структуры задачи решения СЛАУ методом Якоби автоматизированными средствами ТРНП также выполняется существенно (на 2-3 десятичных порядка) быстрее, чем преобразование прикладным программистом (см. таблицу 5.1). Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3,2 до 6, 1 раз, что, как и для задач с информационно-независимыми подграфами, объясняется заметным вкладом времени синтеза проекта. Незначительное изменение времени преобразования кадровой структуры одной и той же задачи для разных аппаратных платформ обусловлено сравнительно небольшим количеством вариантов, рассматриваемых при выборе рационального решения, что экспериментально подтверждает третье и четвертое

положения, выдвигаемые для защиты. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,8, что соответствует уровню создаваемых прикладными программистами решений задач на РВС и превышает уровень реальной производительности, обеспечиваемый рассмотренными в первой главе HLS-компиляторами.

Результаты портации других задач линейной алгебры с информационно-зависимыми подграфами (LU-разложения и решения СЛАУ методом Гаусса-Зейделя), приведенные в Приложении 3, демонстрируют схожие с задачей решения СЛАУ методом Якоби зависимости. Так, время преобразования РНПП одной и той же задачи при портации на различные аппаратные платформы меняется незначительно и не зависит от величины кристалла ПЛИС, что подтверждает сделанные в главах 2-4 теоретические оценки об ограниченном и сравнительно небольшом числе преобразований кадровой структуры при поиске рационального варианта ее реализации в целевых архитектурах. Существенный выигрыш во времени преобразования РНПП (на 2-3 десятичных порядка) обусловлен автоматическим ее преобразованием с помощью разработанных алгоритмов, основой которых стали предложенные в главах 2-3 теоретические основы ресурсонезависимого программирования. При синтезе загрузочного модуля выигрыш сокращается до 3-6 раз, но остается ощутимым. Представленные в таблице 5.3 и Приложении 3 результаты портации задач линейной алгебры с информационно-зависимыми подграфами подтверждены актами о внедрении результатов в СПбГУ, НИИ МВС ЮФУ, НИЦ СЭ и НК.

Полученные результаты портации подтверждают теоретические положения (см. главы 2-3 диссертации) ресурсонезависимого программирования, методы и алгоритмы преобразования кадровых структур в пространстве возможных реализаций. Сокращение времени портации задач с информационно-зависимыми подграфами с обеспечением заданного уровня реальной производительности экспериментально подтверждает достижение цели, поставленной в диссертационной работе.

### 5.4.3 Пример задачи с зависимостью «флагман»-«катер»

Прикладные задачи с информационно-взаимозависимыми подзадачами разной вычислительной трудоемкости, обозначенные в 4 главе термином «флагман»-«катер», при решении на РВС требовали сложных способов согласования структурно реализованных фрагментов задачи с фиксированными параметрами на заданном вычислительном ресурсе. Только высококвалифицированный прикладной программист мог эффективно преобразовать кадровую структуру задачи, поэтому обе подзадачи редко реализовывались в виде согласованной вычислительной структуры на РВС. Примером задачи с такой структурой является решение СЛАУ методом Гаусса [219, 220], состоящее из прямого («флагман») и обратного («катер») хода и демонстрирующее информационную зависимость между подзадачами. В задаче решения СЛАУ методом Гаусса выделяют два этапа: прямой ход, в котором последовательно исключаются переменные с пересчетом коэффициентов в оставшихся уравнениях системы, и обратный ход, в котором последовательно вычисляются все неизвестные, начиная с последнего уравнения СЛАУ. Прямой ход задачи решения СЛАУ методом Гаусса заключается в итерационном расчете коэффициентов с использованием части значений, полученных на предыдущей итерации. На каждой итерации прямого хода объем вычислений сокращается за счет исключения строк и столбцов, лежащих выше ведущей строки. Обратный ход решения СЛАУ методом Гаусса заключается в итерационном вычислении всех неизвестных, начиная с последнего, причем для каждого следующего неизвестного число выполняемых операций увеличивается. Фрагмент текста последовательной программы решения СЛАУ методом Гаусса на языке С показан на рис. 5.23.

Из текста программы (рис. 5.25) видно, что прямой ход описан тройным циклом по переменным i, j, k с общим числом итераций, пропорциональным $N^3$, где $N$ – число строк (столбцов) исходной матрицы. Число итераций обратного хода не превышает $N^2$, поэтому для матриц реальных задач, содержащих от

тысячи строк и больше, число вычислительных операций обратного хода в десятки тысяч раз меньше, чем прямого.

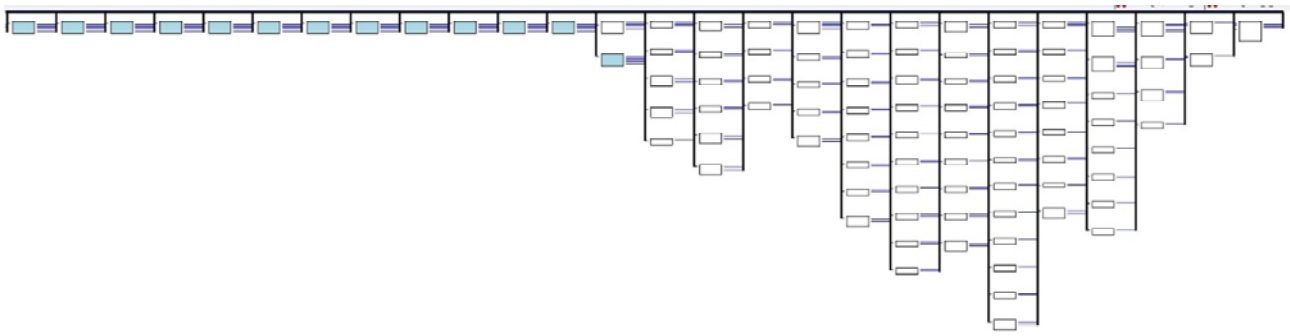
```
1.    //прямой ход метода Гаусса
2.    #FuncGraph_0
3.    for (i = 0; i < N; i++) {
4.      for (j = i+1 ; j < N ; j++)
5.      {
6.    #Graph_0
7.        d = (matrix[j][i]/matrix[i][i]);
8.    #EndGraph
9.
10.    for (k = i; k < N+1 ; k++)
11.   #Graph_1
12.       matrix[j][k] = matrix[j][k] - (matrix[i][k] * d);
13.   #EndGraph
14.     }
15.   }
16.   #EndFuncGraph
17.   //обратный ход метода Гаусса
18.   #FuncGraph_1
19.   #Graph_2
20.   otvet[N-1] = matrix[N-1][N-1]/matrix[N-1][N];
21.   #EndGraph
22.   #EndFuncGraph
23.   #FuncGraph_2
24.   for (t=N-2;t >= 0;t --)
25.     {
26.   #Graph_3
27.       d=0;
28.   #EndGraph
29.       for (r=0;r<N-t-1;r ++)
30.   #Graph_4
31.         d = d - (otvet[N-r-1]*matrix[t][N-r-1]);
32.   #EndGraph
33.   #Graph_5
34.       otvet[t]=(matrix[t][N]+d)/matrix[t][t];
35.   #EndGraph
36.     }
37.   #EndFuncGraph
```

Рис. 5.23. Фрагмент текста программы решения СЛАУ методом Гаусса

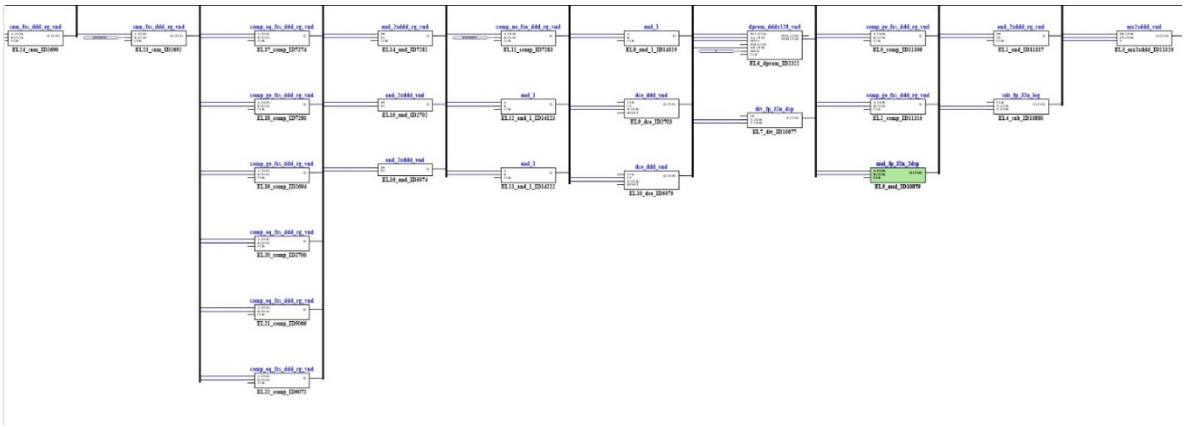
Явная информационная зависимость между подзадачами разной вычислительной трудоемкости не позволяет начать вычисления обратного хода до завершения всех операций прямого хода. В тексте программы (рис. 5.23) решения СЛАУ методом Гаусса специальной разметкой (#) выделены фрагменты, которые «Прокруст» относит к отдельным функциональным

подграфам. На основе анализа числа слоев и итераций каждого выражения по методам (см. параграфы 2.3, 3.5, 4.2 диссертации), «Прокруст» выделяет три функциональных подграфа: прямой ход, вычисление значения последнего неизвестного и обратный ход.

Прямой ход выделен директивой #FuncGraph_0 и состоит из выражений с разными характеристиками слоев и итераций, размеченными директивами #Graph_0 и #Graph_1. Оператор в директиве #FuncGraph_1 рассчитывает значение последнего неизвестного. Обратный ход представлен двойным циклом по индексным переменным t, r, которые выделены директивой #FuncGraph_2 и состоят из выражений с разными характеристиками слоев и итераций, размеченными директивами #Graph_3, #Graph_4 и #Graph_5. Представление задачи решения СЛАУ методом Гаусса, одной итерации «флагмана» - прямого хода и «катера», в окнах программных компонентов разработанных средств ТРНП показано на рис. 5.24.



а)



б)

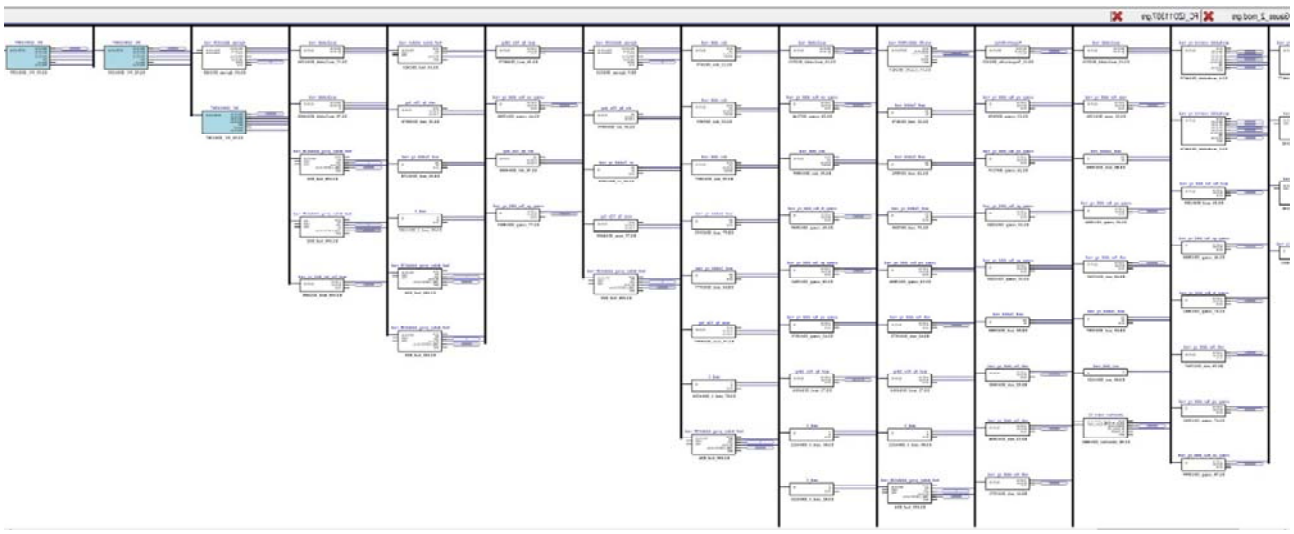


Рис. 5.24. Представление задачи решения СЛАУ методом Гаусса в синтезаторе Fire!Constructor: а) фрагмент общего вида задачи; б) одна итерация прямого хода («флагман»); в) обратный ход («катер»)

Результаты экспериментальной проверки времени и эффективности портации задачи решения СЛАУ методом Гаусса для матриц размером 8000×8000 средствами ТРНП и прикладными программистами на целевые архитектуры рассмотренных аппаратных платформ приведены в табл. 5.4.

Таблица 5.4. Результаты экспериментальной проверки портации задачи решения СЛАУ методом Гаусса

| **Портация прикладной задачи решения СЛАУ методом** Гаусса | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 1516,38 | 1789,25 | 1791,12 | 985,28 | 1001,68 |
| Время портации РНПП средствами ТРНП, с | 12316,38 | 26989,25 | 26991,12 | 11785,28 | 11801,68 |
| Выигрыш преобразование (портация) РНПП | 37,99 (5,55) | 32,19 (3,07) | 32,16 (3,07) | 58,46 (5,19) | 57,50 (5,8) |
| Уровень реальной производительности (число БП РНПП/ пр. прогр.) | **0,86** (498/580) | **0,8** (695/867) | **0,8** (695/867) | **0,68** (123/180) | **0,65** (251/386) |

Как видно из данных таблицы 5.4, преобразование кадровой структуры задачи решения СЛАУ методом Гаусса автоматизированными средствами ТРНП также выполняется существенно (на 2-3 десятичных порядка) быстрее, чем преобразование прикладным программистом (см. таблицу 5.1). Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3,2 до 6, 1 раз, что, как и для предыдущих рассмотренных задач, объясняется заметным вкладом времени синтеза проекта. Незначительное изменение времени преобразования кадровой структуры этой задачи для разных аппаратных платформ обусловлено сравнительно небольшим количеством вариантов, рассматриваемых при выборе рационального решения, что экспериментально подтверждает третье и четвертое положения, выдвигаемые для защиты. Существенный выигрыш во времени преобразования РНПП (на 2-3 десятичных порядка) обусловлен автоматическим ее преобразованием с помощью разработанных алгоритмов, основой которых стали предложенные в главах 2-3 теоретические основы ресурсонезависимого программирования. При синтезе загрузочного модуля выигрыш сокращается до 3-6 раз, но остается ощутимым. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,8, что соответствует уровню создаваемых прикладными программистами решений задач на РВС и превышает уровень реальной производительности, обеспечиваемый рассмотренными в первой главе HLS-компиляторами. Представленные в таблице 5.4 результаты портации задачи решения СЛАУ методом Гаусса подтверждают теоретические положения ресурсонезависимого программирования и подтверждены актами о внедрении результатов в ЮНЦ РАН, НИИ МВС ЮФУ, НИЦ СЭ и НК.

### 5.4.4. Пример задачи с зависимостью «флагман»-{катер}

Прикладные задачи, содержащие множество информационно-взаимозависимых подзадач, одна из которых значительно превышает остальные

по вычислительной трудоемкости, получили название «флагман»-{катер}. Такие задачи редко реализовывались структурно на РВС из-за недостатка ресурса и сложности синхронизации. Обычно применялся структурно-процедурный подход, когда «флагман» реализовывался структурно, а вычисления «катеров» выполнялись в режиме пост-обработки на управляющем компьютере или хост-машине с использованием данных, выгруженных из РВС. С появлением встроенных в ПЛИС процессоров, например PowerPC, и развитием софт-процессоров, например MicroBlaze, появилась возможность реализации задач «флагман»-{катер} на нескольких целевых архитектурах в едином контуре, что, в свою очередь, требовало участия двух разработчиков для реализации и синхронизации вычислений для каждой целевой архитектуры с учетом используемых шин и интерфейсов взаимодействия.

Характерным представителем таких задач является расчет энергетических характеристик взаимодействия разных молекул как единой конфигурации [184-187]: по исходной химической формуле требуется создать очередную пространственную конфигурацию одной из молекул, считающейся подвижной, позиционировать ее относительно центра другой и вычислить энергетические характеристики полученной конфигурации. Создание очередной пространственной конфигурации подвижной молекулы – самая вычислительно трудоемкая часть задачи, в которой учитываются вращение отдельных фрагментов молекулы и ее позиционирование в многомерном пространстве как целого, что требует многократного пересчета координат каждого атома по сложным формулам [184]. На основе типа каждого атома и рассчитанных координат пространственной конфигурации подвижной молекулы вычисляются информационно-независимые энергетические характеристики, сумма которых составляет общую энергию взаимодействия (связывания) разных молекул как целого. Структура подзадач задачи расчета энергетических характеристик взаимодействия разных молекул в виде информационного графа представлена на рис. 5.25.

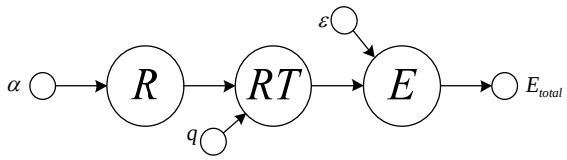
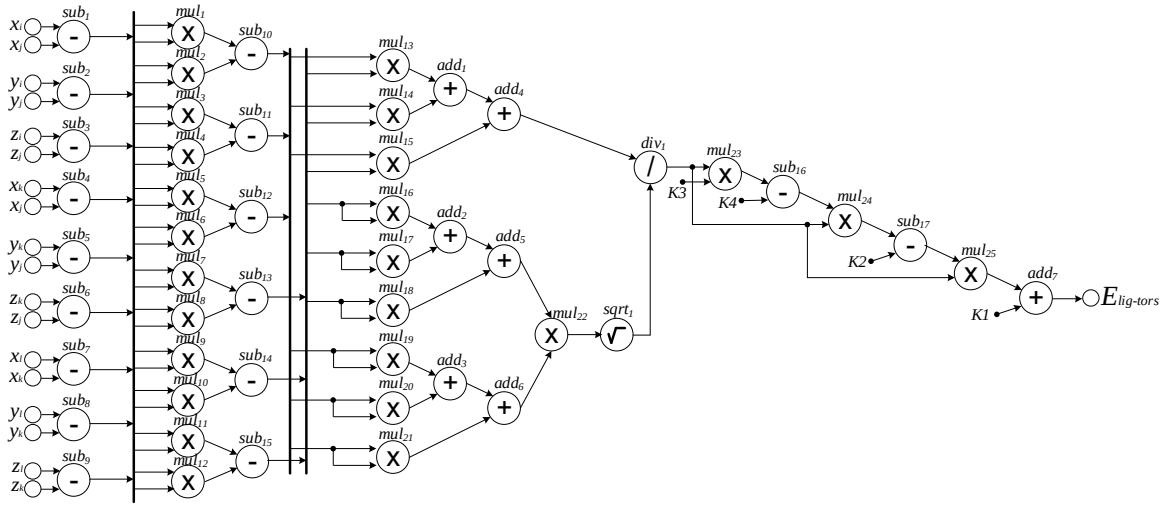
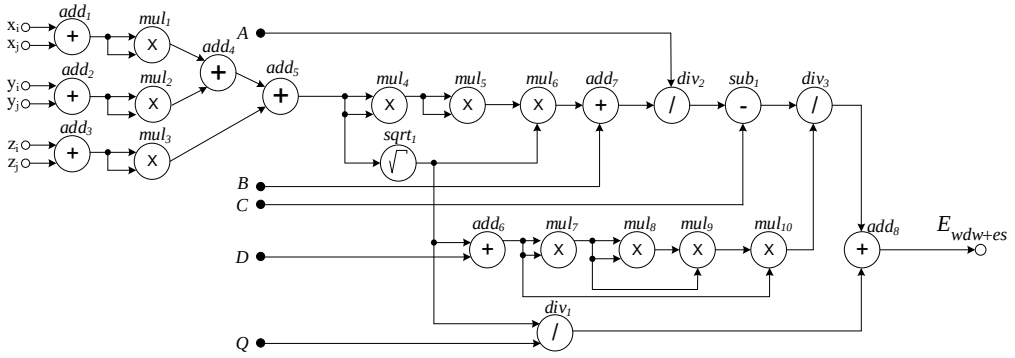



Рис. 5.25. Структура задачи расчета энергетических характеристик взаимодействия разных молекул

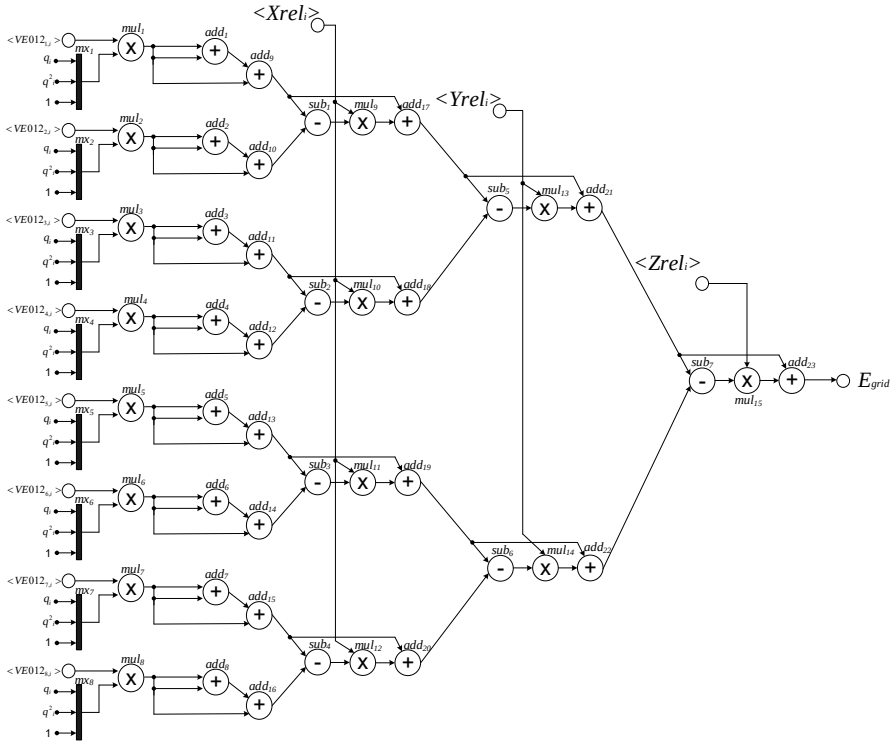
Здесь $R$ – подграф подзадачи вращения отдельных фрагментов молекулы, $RT$ – подграф подзадачи позиционирования подвижной молекулы в многомерном пространстве как целого, $E$ – подграф подзадачи расчета энергии $E_{total}$, состоящий из расчета энергетических характеристик E_tors, E_VdW, E_es и E_grid, $\alpha, \varepsilon, q$ – параметры молекул. По сравнению с расчетом пространственной конфигурации R и ориентации RT, подзадачи вычисления каждого из слагаемых E_tors, E_VdW, E_es и E_grid значительно менее трудоемкие, поэтому они являются «катером», а создание очередной пространственной конфигурации подвижной молекулы –«флагманом». Число учитываемых энергетических характеристик для расчета общей энергии взаимодействия прямо влияет на точность моделирования. В [175, 178-180] использовались 3 слагаемых, но число слагаемых и «катеров» в структуре задачи может увеличиться в зависимости от используемой модели взаимодействия и требуемой точности. Структурная реализация подзадачи вычисления каждого фрагмента энергетических характеристик торсионного взаимодействия E_tors, расчета энергии Ван-дер-Ваальсовых и электростатических взаимодействий между атомами E_VdW, расчета энергии подвижной молекулы в поле неподвижной E_es и E_grid представлена на рис. 5.26. Как уже отмечалось, в рассматриваемой задаче фрагменты расчета энергетических характеристик значительно менее трудоемки по сравнению с расчетом координат всех атомов молекулы при повороте и позиционировании. Поэтому объединенные фрагменты расчета координат R и RT являются «флагманом», а фрагменты расчета энергий являются «катерами», информационно-взаимозависимыми от «флагмана».




а)




б)




в)

Рис. 5.26. Вычисление энергетических характеристик взаимодействия молекул а) фрагмент расчета энергии E_tors, б) фрагмент расчета энергии E_VdW, в) фрагмент расчета энергии E_grid

Результаты экспериментальной проверки времени и эффективности портации задачи расчета энергетических характеристик взаимодействия разных молекул средствами ТРНП и прикладными программистами на целевые архитектуры рассмотренных аппаратных платформ приведены в табл. 5.5.

Таблица 5.5. Результаты экспериментальной проверки портации задачи расчета энергетических характеристик взаимодействия разных молекул

| **Портация прикладной задачи расчета энергетических характеристик** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 956,45 | 1293,64 | 1295,36 | 732,12 | 741,76 |
| Время портации РНПП средствами ТРНП, с | 11756,45 | 26493,64 | 26495,36 | 11532,12 | 11541,76 |
| Выигрыш преобразование (портация) РНПП | 60,22 (5,82) | 44,53 (3,13) | 44,47 (3,13) | 78,68 (5,31) | 77,65 (5,93) |
| Уровень реальной производительности (число БП РНПП/ пр. прогр.) | **0,6** (3/5) | **0,63** (5/8) | **0,63** (5/8) | **0,66** (2/3) | **0,66** (2/3) |

Как видно из данных таблицы 5.5, преобразование кадровой структуры задачи решения СЛАУ методом Гаусса автоматизированными средствами ТРНП также выполняется быстрее, чем преобразование прикладным программистом (см. таблицу 5.1), но выигрыш не превышает 2 десятичных порядков. Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3 до 5,8 раз, что, как и для предыдущих рассмотренных задач, объясняется заметным вкладом времени синтеза проекта. Постоянство времени преобразования кадровой структуры этой задачи для разных аппаратных платформ обусловлено сравнительно небольшим количеством вариантов, рассматриваемых при выборе рационального решения, что экспериментально подтверждает третье и четвертое положения, выдвигаемые для защиты. Выигрыш во времени преобразования РНПП в 30-58 раз обусловлен автоматическим ее преобразованием с помощью разработанных алгоритмов,

основой которых стали предложенные в главах 2-3 теоретические основы ресурсонезависимого программирования. При синтезе загрузочного модуля выигрыш сокращается до 3-5,8 раз, но остается ощутимым. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,65, что соответствует уровню создаваемых прикладными программистами решений задач на РВС и превышает уровень реальной производительности, обеспечиваемый рассмотренными в первой главе HLS-компиляторами. Представленные в таблице 5.5 результаты портации задачи расчета энергетических характеристик взаимодействия разных молекул подтверждают теоретические положения ресурсонезависимого программирования и подтверждены актами о внедрении результатов в ЮНЦ РАН, НИИ МВС ЮФУ, НИЦ СЭ и НК.

### 5.4.5. Пример задачи с зависимостью «флагман»-«катер»-«софлагман»

Характерным представителем задач с информационной зависимостью «флагман»-«катер»-«софлагман» является расчет свертки сигналов, содержащий информационно-зависимые подзадачи прямого преобразования Фурье, комплексного умножения его результатов на заданное значение и обратного преобразования Фурье (рис.5.27) [221]. Вычислительная трудоемкость подзадач прямого и обратного преобразования Фурье примерно одинакова, а для комплексного умножения значительно ниже. Поэтому ФП прямого и обратного преобразования Фурье можно отнести к «флагманам», а комплексное умножение является «катером».

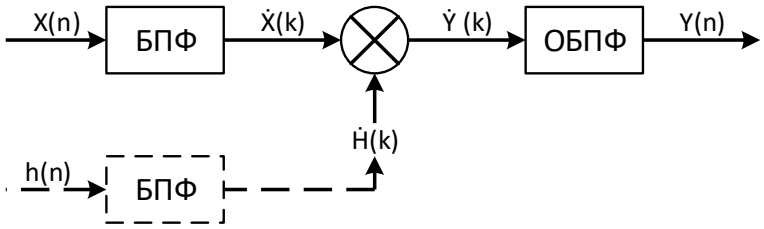
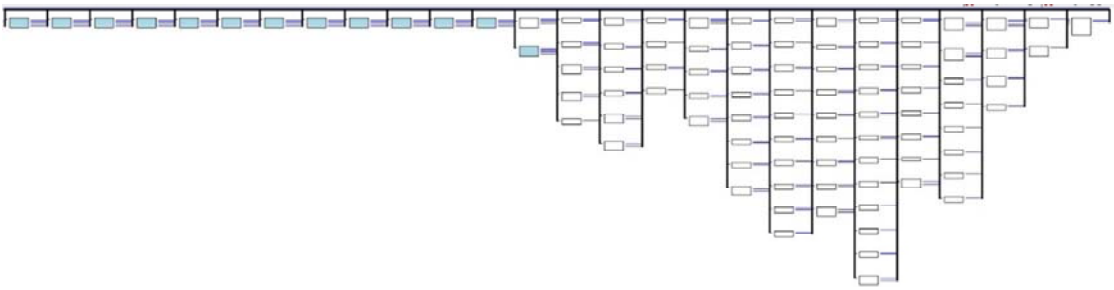
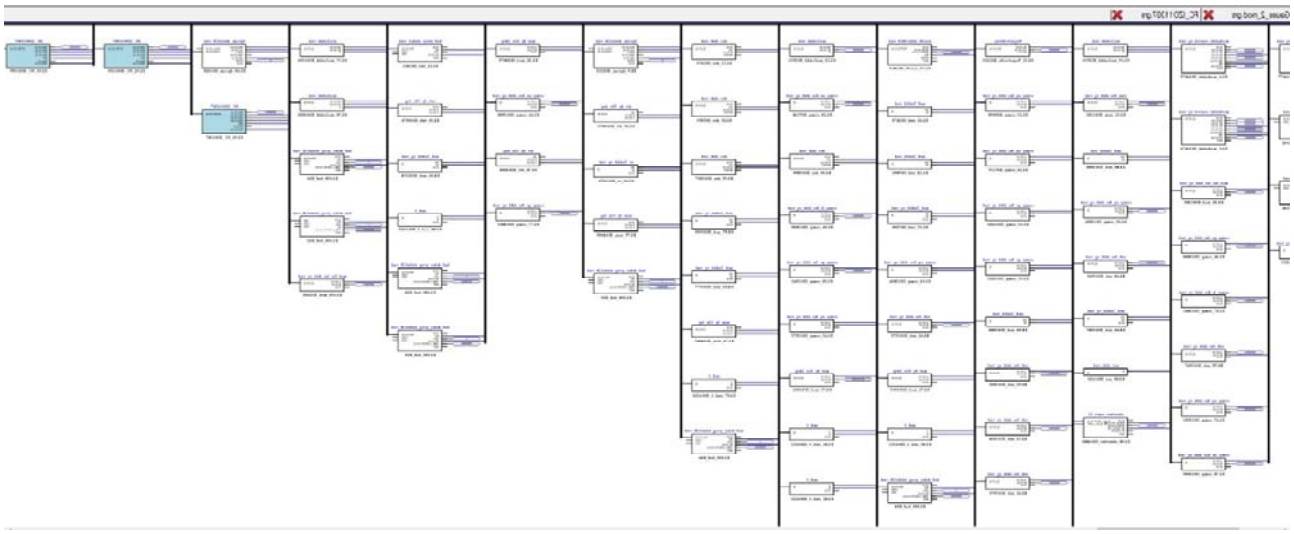



Рис. 5.27. Структура задачи свертки сигналов с расчетом коэффициентов H(K)

В зависимости от реализации свертки сигналов, коэффициенты преобразования $H_k$ можно вычислять одновременно с прямым преобразованием Фурье или использовать предварительно рассчитанные значения, если параметры задачи постоянны и не меняются для данных входного потока. Поэтому блок расчета коэффициентов преобразования $H_k$ (рис. 5.27) изображен пунктирной линией, показывающей, что коэффициенты $H_k$ можно получить заранее, т.е. не вычислять во время решения задачи, а считывать из памяти. Представление задачи свертки сигналов, структура одной итерации «флагмана» и «софлагмана» в окнах программных компонентов разработанных средств ТРНП показаны на рис. 5.28.



а)



б)

Рис. 5.28. Представление задачи свертки сигналов в синтезаторе Fire!Constructor: а) общий вид задачи; б) представление «флагмана»;

Преобразование алгоритма БПФ с редукцией по каналам и устройствам подробно рассмотрено в пункте 3.1.2 диссертации, синтез микро-кадров программой «Щелкунчик» для алгоритма БПФ описан в параграфе 5.3 диссертации, поэтому основное внимание уделим рассмотрению портации информационной зависимости «флагман»-«катер»-«софлагман».

Результаты экспериментальной проверки времени и эффективности портации задачи свертки сигналов для массивов размером 1024×1024 средствами ТРНП и прикладными программистами на целевые архитектуры рассмотренных аппаратных платформ приведены в табл. 5.6.

Таблица 5.6. Результаты экспериментальной проверки портации задачи свертки сигналов

| **Портация прикладной задачи свертки сигналов** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 956,45 | 1456,80 | 1458,44 | 921,12 | 987,54 |
| Время портации РНПП средствами ТРНП, с | 11756,45 | 26656,80 | 26658,44 | 11721,12 | 11787,54 |
| Выигрыш преобразование (портация) РНПП | 60,22 (5,82) | 39,54 (3,11) | 39,49 (3,11) | 62,53 (5,22) | 58,33 (5,80) |
| Уровень реальной производительности (число БП РНПП/ пр. прогр.) | **0,75** (6/8) | **0,77** (10/13) | **0,77** (10/13) | **0,67** (2/3) | **0,67** (4/6) |

Как видно из данных таблицы 5.6, преобразование кадровой структуры свертки сигналов автоматизированными средствами ТРНП также выполняется быстрее, чем прикладным программистом. Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3 до 5,8 раз, что, как и для предыдущих рассмотренных задач, объясняется заметным вкладом времени синтеза проекта. Постоянство времени преобразования кадровой структуры этой задачи для разных аппаратных платформ обусловлено сравнительно небольшим количеством вариантов, рассматриваемых при выборе рационального решения, что экспериментально подтверждает третье и четвертое

положения, выдвигаемые для защиты. Выигрыш во времени преобразования РНПП в 39-62 раза обусловлен автоматическим ее преобразованием с помощью разработанных алгоритмов, основой которых стали предложенные в главах 2-3 теоретические основы ресурсонезависимого программирования. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,67, что соответствует уровню создаваемых прикладными программистами решений задач на РВС и превышает уровень реальной производительности, обеспечиваемый рассмотренными в первой главе HLS-компиляторами. Представленные в таблице 5.6 результаты портации задачи с информационной зависимостью «флагман»-«катер»-«софлагман» подтверждают теоретические положения ресурсонезависимого программирования и подтверждены актами о внедрении результатов в ЮНЦ РАН, НИИ МВС ЮФУ, НИЦ СЭ и НК.

### 5.4.6. Пример задачи с зависимостью «взаимодействующие равные»

Прикладные задачи «взаимодействующие равные» содержат несколько (не менее трех) подзадач примерно одинаковой вычислительной трудоемкости с разными информационно-взаимозависимыми функциональными подграфами. Основным отличием таких задач от ранее рассмотренных типов является необходимость обеспечения сбалансированного темпа обработки данных во всех подзадачах с разными уровнями параллелизма. Темп обработки данных зависит от выбора начального фрагмента для преобразования кадровой структуры задачи. К числу таких задач относятся композиции блочных алгоритмов символьной разработки, применяемые для увеличения эффективной длины ключа или комбинация симметричных блочных шифров и хеш-функций в схемах устойчивой генерации ключей на основе паролей - Password-Based Key Derivation Function[222], таких как PBKDF или PBKDF-2 [223].

Одним из примеров таких задач является информационно-зависимое объединение алгоритмов DES, ГОСТ и AES (рис. 5.29), при котором шифртекст

одного алгоритма поступает в качестве открытого текста на вход другого. Если такая комбинация не образует симметрическую группу, она соответствует положению об увеличении стойкости при использовании комбинации преобразований, доказанному К. Шенноном в работе «Теория связи в секретных системах» [224].

Рис. 5.29. Структура композиции блочных алгоритмов DES-ГОСТ-AES с зависимостью «взаимодействующие равные»

Здесь $K_D, K_G, K_A$ – это ключи алгоритмов DES, GOST и AES соответственно. Портация алгоритма DES средствами ТРНП рассмотрена в пункте 5.4.1, а портация алгоритмов GOST и AES – в Приложении 3.

Результаты экспериментальной проверки времени и эффективности портации задачи DES-ГОСТ-AES средствами ТРНП и прикладными программистами на целевые архитектуры рассмотренных аппаратных платформ приведены в табл. 5.7.

Таблица 5.7. Результаты экспериментальной проверки портации задачи DES-ГОСТ-AES

| **Портация прикладной задачи DES-ГОСТ-AES** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 481,30 | 596,25 | 597,12 | 298,31 | 302,69 |
| Время портации РНПП средствами ТРНП, с | 11281,30 | 25796,25 | 25797,12 | 11098,31 | 11102,69 |
| Выигрыш преобразование (портация) РНПП | 119,68 (6,06) | 96,60 (3,21) | 96,46 (3,21) | 193,09 (5,51) | 190,29 (6,16) |
| Уровень реальной производительности (число БП РНПП/ пр. прогр.) | **0,67** (4/6) | **0,75** (6/8) | **0,75** (6/8) | **0,63** (2/3) | **0,60** (3/5) |

Как видно из данных таблицы 5.7, преобразование кадровой структуры задачи с информационной зависимостью «взаимодействующие равные» автоматизированными средствами ТРНП также выполняется быстрее, чем прикладным программистом. Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3,2 до 6,1 раз, что, как и для предыдущих рассмотренных задач, объясняется заметным вкладом времени синтеза проекта. Постоянство времени преобразования кадровой структуры этой задачи для разных аппаратных платформ обусловлено сравнительно небольшим количеством вариантов, рассматриваемых при выборе рационального решения, что экспериментально подтверждает третье и четвертое положения, выдвигаемые для защиты. Выигрыш во времени преобразования РНПП в 96-190 раз обусловлен автоматическим ее преобразованием с помощью разработанных алгоритмов, основой которых стали предложенные в главах 2-3 теоретические основы ресурсонезависимого программирования. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,6, что соответствует уровню создаваемых прикладными программистами решений задач на РВС и превышает уровень реальной производительности, обеспечиваемый рассмотренными в первой главе HLS-компиляторами. Представленные в таблице 5.7 результаты портации задачи с информационной зависимостью «взаимодействующие равные» подтверждают теоретические положения ресурсонезависимого программирования и подтверждены актами о внедрении результатов в ЮНЦ РАН, НИИ МВС ЮФУ, НИЦ СЭ и НК.

Представленные результаты проверки портации (табл. 5.2-5.7, рис. 5.21-5.29), позволяют сделать вывод о том, что применение разработанных средств ТРНП сокращает время портации прикладных задач с различными типами информационной зависимости, что экспериментально подтверждает достижение поставленной цели – минимизации времени портации прикладных задач для ГВС.

## 5.5. Возможные направления дальнейших исследований

Теоретические основы технологии ресурсонезависимого программирования, созданные для преобразования кадровых структур с различными типами информационной зависимости, позволяют использовать сформулированные общие принципы и методологические положения для новых потенциально возможных типов задач.

Ресурсонезависимая параллельная программа необязательно должна быть написана пользователем, она может быть создана автоматически из текста программы на языке высокого уровня COLAMO с помощью специализированной программы [226]. Программа «Русалка» [227-229] преобразует максимально параллельную кадровую структуру, описанную на языке COLAMO, к ресурсонезависимой параллельно-конвейерной форме с помощью параметризации аппаратным ресурсом. Для этого он выполняет лексический, синтаксический и семантический анализ входной параллельной программы. По результатам анализа для всех мемориальных и коммутационных массивов к измерениям с параллельным (векторным, Vec) типом доступа добавляется измерение с потоковым (последовательным, Str) типом доступа, связанное с векторным параметрической зависимостью с помощью уникальных параметров паралеллизма DP (см. параграф 5.2 диссертации). Для сохранения информационной эквивалентности и тождественности РНПП исходной максимально параллельной программе длина векторной составляющей остается без изменения (DP устанавливается равным длине исходной размерности), а длина добавленного потокового измерения, рассчитываемого с помощью параметра DP, будет равна 1. С уменьшением параметра DP степень параллелизма и векторная составляющая будут сокращаться, а длина потоковой составляющей массивов будет увеличиваться. Предельным значением параметра параллелизма DP является единица, соответствующая конвейерной организации вычислений с подачей данных из канала памяти каждый такт. Изменение значения параметра параллелизма DP для каждого массива в

ресурсонезависимой параллельно-конвейерной форме за счет параметрической зависимости приводит к автоматическому пересчету числа информационно-независимых каналов памяти и длины обрабатываемых в каждом из них потоков данных для кадровых структур и функциональных подграфов всех подзадач. С помощью изменения значения параметра параллелизма DP выполняется движение по слоям и итерациям в пространстве возможных реализаций кадровых структур (см. параграфы 2.4-2.5 диссертации).

Циклы обработки массивов данных параметризуются аналогичным образом: для каждого векторного цикла, адресующего измерение массива с параллельным типом доступа, добавляется потоковый цикл обработки добавленного измерения с потоковым типом доступа. Границы векторного и потокового циклов параметризуются следующим образом: векторный цикл задается диапазоном от 0 до (DP-1), потоковый цикл задается от 0 до длины потоковой размерности (длины исходной размерности массива, деленной на DP с округлением вниз), таким образом обеспечивается информационная эквивалентность для максимально параллельной и минимальной кадровых структур с возможностью синтеза промежуточных кадровых структур при движении на плоскости слоев и итераций.

Результатом работы «Русалки» является синтаксически корректная, без нарушения правил однократного присваивания и единственной подстановки, программа на языке COLAMO с кадровой структурой в ресурсонезависимой параллельно-конвейерной форме, которую изменением параметров паралеллизма DP программа «Прокруст» может преобразовать к информационно-эквивалентной кадровой структуре, требующей меньшего аппаратного ресурса.

«Русалка» взаимодействует с другими средствами ТРНП на уровне входных и выходных данных – программ на языке программирования COLAMO с расширением ***.clm**. Программа «Русалка», как «Прокруст» и «Щелкунчик», не имеет графического интерфейса и не взаимодействует с пользователем, поэтому у него нет настраиваемых параметров.

Средства создания ресурсонезависимых параллельных программ на языке COLAMO использовались в научно-исследовательских и опытно-конструкторских работах (НИР «Русалка», ОКР «Большая Медведица», НИОКР «Триада») при решении сложных прикладных задач из различных предметных областей, таких как докинг лигандов при молекулярном моделировании новых лекарственных средств [181, 184-187], моделирование движения воздушных масс и процессов их влияния на природные объекты в районе функционирования Ростовской АЭС, построение моделей на основе томографического исследования приповерхностных слоев Земли акустическими и электромагнитными волнами, диагностика дорожных покрытий и аэродромных взлетно-посадочных полос, прогнозирование распространения лесных пожаров, цифровая обработка сигналов для корректировки оптических изображений азимутального телескопа[214] и др.

Дальнейшее развитие разработанных инструментальных средств технологии ресурсонезависимого программирования возможно в направлении автоматизации создания информационного графа и максимально параллельной кадровой структуры, описанной в синтаксисе COLAMO, из последовательной программы на одном из широко распространенных языков программирования. Автоматическое построение информационного графа из последовательной программы, автоматическое его преобразование к ресурсонезависимой параллельно-конвейерной форме и портация последней в целевые архитектуры ГВС открывают возможности создания автоматического распараллеливающего компилятора для систем с распределенной памятью, что является новым практическим результатом, существенным для теории и практики параллельного программирования.

Построение информационного графа из последовательной программы на процедурном языке программирования автоматизированными средствами позволяет значительно упростить и ускорить преобразование большого числа разработанных алгоритмов к максимально параллельной кадровой структуре в программе на языке COLAMO. Методика преобразования последовательной

программы на языке программирования C в программу на COLAMO описана в [225, 228, 229]. Ее инструментальная реализация для последовательных программ на языке C в стандарте ISO/IEC 9899:1999 с GNU-расширением компилятора gcc в виде транслятора «Ангел» [228] выполняет преобразования в два этапа. На первом этапе транслятор «Ангел»:

- приводит программу на языке программирования C к каноническому виду (раскрывает директивы условной компиляции, тернарные операции, директивы #pragma, преобразует указатели к адресации массивов и другие вспомогательные преобразования) для упрощения построения информационного графа как максимально параллельной формы программы;

- строит максимально параллельную форму – информационный граф программы - на языке C, который описывает синтаксическими конструкциями языка COLAMO. Все инициализируемые в программе на C массивы становятся мемориальными массивами COLAMO с параллельным типом доступа (векторными по всем измерениям);

- объединяет вычислительные выражения в условных операторах для сокращения числа мультиплексоров, выносит общие операторы из основной и альтернативной ветвей условных операторов и выполняет ряд других преобразований.

На втором этапе транслятор «Ангел» преобразует произвольное обращение к памяти в последовательной программе к потокам данных параллельной программы с сохранением информационных зависимостей и семантической корректности. Для этого с помощью анализа информационных зависимостей, расщепления скалярных переменных и растягивания массивов по итерациям текст программы на COLAMO приводится в соответствие правилам единственной подстановки и однократного присваивания, характерных для языков потоков данных. Для соблюдения этих правил мемориальные массивы могут расщепляться – когда объявляется коммутационный массив-копия с тем же количеством векторных измерений и размерностью, не требующий аппаратного ресурса, который используется в программе для обеспечения

информационной эквивалентности результатов вычислений. Анализ информационных зависимостей, методика которого подробно рассмотрена в [227, 229], позволяет также определить предпочтительный способ распараллеливания вычислений (по слоям или по итерациям).

Интеграция транслятора «Ангел», программы «Русалка» и разработанных программных средств технологии ресурсонезависимого программирования в единый программный комплекс позволит полностью автоматизировать преобразование последовательной программы к различным конфигурациям целевых архитектур гибридных вычислительных систем и заложить фундамент для автоматического распараллеливающего компилятора для систем с распределенной памятью, что станет новым важным результатом в теории и практике высокопроизводительных вычислений.

Дальнейшим развитием идей ресурсонезависимого программирования для описания алгоритмов, портируемых на архитектуры как существующих, так и перспективных вычислительных систем, является архитектурно-независимое программирование [230-235]. Концепция архитектурно-независимого программирования объединяет идеи аспектно-ориентированного программирования, теоретико-множественного представления данных и неявного описания параллелизма в языках потоков данных в единую парадигму для решения проблемы портации на новом уровне – с помощью нового языка программирования SET@L. Язык архитектурно-независимого программирования Set@l объединяет основные принципы языков программирования COLAMO и SETL в парадигме аспектно-ориентированного программирования (АОП) [236], согласно которой алгоритм решения прикладной задачи и особенности его реализации описываются в виде отдельных модулей программы. Программа на языке Set@l представляет информационный граф решаемой вычислительной задачи не массивами, а множествами, разбиение и типизация которых задают разные варианты распараллеливания и другие аспекты реализации алгоритма. В отличие от других языков программирования на основе теории множеств, например языков SETL, SETL2 и SETLX, язык Set@l

использует типизацию множеств по различным критериям и позволяет оперировать нечеткими совокупностями согласно альтернативной теории множеств П. Вопенка [237]. Благодаря использованию парадигмы аспектно-ориентированного программирования, в языке Set@l разделяется описание алгоритма и особенностей его реализации в архитектуре вычислительной системы. Согласно этой парадигме, сквозная функциональность программы, вызывающая негативные эффекты спутывания и разбрасывания кода, представляется в виде отдельных программных модулей – аспектов. Исходный код содержит функционал и создаваемую пользователем разметку, которая определяет его взаимодействие с аспектами при трансляции или исполнении. Анализируя разметку и исходный код, транслятор-препроцессор формирует новую исполняемую программу с вплетенной сквозной функциональностью. Применение технологии АОП, как правило, упрощает разработку и дальнейшую поддержку программ при портации на различные целевые архитектуры.

Для неявного описания различных способов распараллеливания алгоритмов в языке архитектурно-независимого программирования ВС Set@l введена классификация совокупностей по типу параллелизма их элементов при обработке. В тех случаях, когда характер параллелизма элементов совокупности четко определен, используются типы «множество» (par – параллельно-независимая обработка), «кортеж» (seq – последовательная обработка), «кортеж-конвейер» (pipe – конвейерная обработка) и «множество обработки по итерациям» (conc – параллельно-зависимая обработка). Однако в некоторых аспектах программы тип параллелизма ряда совокупностей нельзя определить однозначно, т.к. неизвестна архитектура ВС, на которой будет реализован алгоритм. Тогда используется особый тип imp (неопределенный), а типизация совокупностей уточняется в других аспектах программы с помощью специальных синтаксических конструкций. Представленная особенность языка программирования Set@l проиллюстрирована примером архитектурно-независимой программы, реализующей прямой ход алгоритма решения СЛАУ методом Гаусса [238].

Возможности языка аспектно-ориентированного программирования Set@l позволяют описать информационный граф в виде множества с параметризуемым разбиением и преобразованием топологии при неизменном исходном коде программы. Если аспекты программы на языке Set@l не изменяют алгоритм в процессе его адаптации к архитектуре ВС, то решение задачи можно представить в рамках классической теории множеств Кантора–Больцано. Однако в ряде случаев следует модернизировать алгоритм, учитывая особенности его реализации на ВС с определенной архитектурой. В таких случаях некоторые совокупности выделены нечетко и не являются множествами, поэтому их невозможно задать как объекты классической теории множеств. Set@l позволяет описать разные способы реализации одного и того же алгоритма в единой аспектно-ориентированной программе. Для этого вводится классификация совокупностей по четкости выделения их элементов, предложенная в рамках альтернативной теории множеств П. Вопенка. Тип «множество» (set) соответствует классической четко выделенной совокупности элементов, тип «полумножество» (sm) – нечеткой совокупности, а тип «класс» (cls) – совокупности объектов, тип и разбиение которой нельзя однозначно определить на данном уровне абстракции. Примером вычислительной задачи, для описания которой на языке программирования Set@l необходимы нечеткие совокупности, является алгоритм решения СЛАУ методом Якоби [239]. В статьях [240-244] приведены примеры ресурсонезависимого описания различных топологий информационноых графов с ассоциативными операциями на языке программирования Set@l, в том числе для идеального случая единичной латентности операционной вершины.

В отличие от других подходов к параллельному программированию высокопроизводительных ВС, язык программирования Set@l задает не только граничные случаи реализации алгоритма, но и семейство промежуточных вариантов, которые неразличимы с точки зрения процедурного программирования, но обеспечивают непрерывность описания модели вычислений. Благодаря использованию языка Set@l, можно синтезировать

множество вариантов решения задачи и переходить между его элементами в зависимости от архитектуры и конфигурации вычислительной системы. Такая возможность проиллюстрирована [241] примером программы быстрого преобразования Фурье: в зависимости от доступного объема памяти вычислительной системы комплексные поворотные коэффициенты вычисляются заранее и загружаются из памяти или рассчитываются через базовые и вспомогательные составляющие в процессе вычислений. Алгоритм решения задачи в языке Set@l представляется в виде архитектурно-независимого исходного кода с помощью аспектно-ориентированного программирования, теоретико-множественного представления кода и реляционного исчисления. Особенности реализации алгоритма выделяются в отдельные аспекты, задающие разбиение на подмножества и типизацию основных совокупностей программы. Это позволяет заложить основы для нового направления разработки архитектурно- независимых программ, включающего в себя:

- создание автоматизированных средств портации и компиляции [245] для существующих и перспективных (например, фотонных и квантовых) целевых архитектур;

- создание архитектурно-независимых программ, которые можно автоматически портировать в наиболее эффективную реализацию для архитектуры заданной вычислительной системы.

Программы различных областей [246-248] могут оперативно портироваться между вычислительными системами с разными архитектурами и адаптироваться к любым изменениям аппаратных ресурсов с помощью аспектов метода обработки, архитектуры и конфигурации.

## 5.6. Выводы к главе 5

1) Совокупность научных понятий, методологических принципов, положений, выявленных закономерностей, методов и алгоритмов портации кадровых структур, составляющих разработанные теоретические основы ресурсонезависимого программирования, апробирована практической реализацией в инструментальных программных компонентах. Приведены

структура и схема функционирования комплекса средств технологии ресурсонезависимого программирования, реализующего формализованную методику портации прикладных задач.

2) Методы и алгоритмы преобразования кадровой структуры задач с информационно-зависимыми подзадачами реализованы в программе портации кадровой структуры на гибридные вычислительные системы различных архитектур и конфигураций. Представлено подробное описание входных и выходных данных, алгоритмов работы программы с иллюстрацией на фрагментах задач.

3) Методы редукции производительности и аппаратных затрат кадровой структуры при нехватке аппаратного ресурса для реализации базового подграфа реализованы в программе преобразования фрагментов кадровой структуры на языке COLAMO в микро-кадры. Представлено подробное описание входных и выходных данных, алгоритмов работы программы с иллюстрацией на фрагментах задач.

4) Описаны архитектура и конфигурация аппаратных средств и целевых архитектур, которые использовались для экспериментальной проверки разработанных программных средств и их взаимодействия при портации прикладных задач. Представлены результаты портации ряда прикладных задач к целевым архитектурам различных гибридных вычислительных систем. При портации ресурсонезависимой параллельной программы на каждую из целевых архитектур фиксировались время портации, синтеза проекта и получения загрузочного модуля, а также достигнутая реальная производительность, на основании которых рассчитывались коэффициенты сокращения затрат разработанными средствами технологии ресурсонезависимого программирования по сравнению с результатами, полученными прикладными программистами.

5) Рассмотрены преобразования задач с единственным информационно-независимым базовым подграфом, с единственным информационно-зависимым базовым подграфом и задач с различными

рассмотренными типами информационной зависимости: «флагман»-«катер», «флагман»-{катер}, «флагман»-«катер»-«софлагман», «взаимодействующие равные».

6) Сравнение времени преобразования кадровых структур различных по информационной зависимости задач: с единственным информационно-независимым базовым подграфом, с единственным информационно-зависимым базовым подграфом, «флагман»-«катер», «флагман»-{катер}, «флагман»-«катер»-«софлагман», «взаимодействующие равные» позволяет сделать вывод, что с помощью разработанных программных средств время преобразования существенно (на 2-3 десятичных порядка) сокращается. Применение разработанных программных средств технологии ресурсонезависимого программирования также сокращает время портации не менее, чем в 3 раза, для всех рассмотренных задач, обеспечивая реальную производительность не ниже 0,6, что экспериментально подтверждает достижение поставленной в диссертационной работе цели.

7) Сокращение времени портации задач с различными типами информационной зависимости на различные целевые архитектуры ГВС и достигнутые показатели эффективности подтверждены актами внедрения результатов работы в НИОКР, выполнявшихся в различных организациях.

8) Полученные результаты портации прикладных задач экспериментально подтверждают теоретические положения, методы и алгоритмы преобразования кадровых структур в пространстве возможных реализаций. Соответствие экспериментальных результатов теоретическим положениям позволяет сделать вывод о решении поставленной научной задачи. Практическое использование результатов и их внедрение в практику автоматического создания масштабируемых параллельно-конвейерных решений позволит сократить время портации прикладных задач различных областей для целевых архитектур гибридных вычислительных систем с 2 дней до 50 минут с обеспечением заданного уровня реальной производительности не ниже 60%.

# ЗАКЛЮЧЕНИЕ

Непрерывное развитие фундаментальных наук требует сохранения темпа роста производительности современных высокопроизводительных вычислительных систем. Обеспечить рост реальной производительности и преодолеть кризис высокопроизводительных вычислений возможно объединением специализированных вычислительных архитектур, таких как графические ускорители и программируемые логические интегральные схемы, в гибридную вычислительную систему, чтобы достичь эффективности не ниже 50% для широкого спектра вычислительных задач. Для повышения эффективности реализации вычислений необходимо совместное применение узлов разных архитектур ГВС для решения задачи, для чего необходимо совместное использование различных технологий программирования не только при создании единой программы, но и эффективных средств ее преобразования (портации) на часто меняющуюся конфигурацию гибридной вычислительной системы. Это обуславливает актуальность разработки новых моделей представления, методов организации и портации вычислений прикладных задач для различных архитектур и конфигураций гибридных вычислительных систем при заданном уровне их эффективности и реальной производительности.

В диссертации успешно решена **актуальная научная проблема** – разработка теоретических основ, методов и инструментальных программных средств ресурсонезависимого программирования, имеющая важное значение для повышения эффективности программирования и процессов обработки данных в гибридных вычислительных системах. Прикладной результат решения поставленной научной проблемы обеспечивает достижение поставленной цели – сокращение времени портации прикладных задач для различных архитектур и конфигураций гибридных вычислительных систем при заданном уровне реальной производительности. Совокупность сформулированных в диссертации научных понятий, выявленных

закономерностей и вытекающих из них принципов образуют теоретические основы ресурсонезависимого программирования гибридных вычислительных систем с совместным использованием узлов различных типов при решении прикладной задачи. Разработанные методы и инструментальные средства программирования, являющиеся развитием предложенных теоретических основ, объединены в технологию ресурсонезависимого программирования, обеспечивающую, с одной стороны, рациональную реализацию вычислений на различных узлах и доступном ресурсе ГВС, а с другой – предоставляющую программисту достаточный уровень удобства разработки, портации и отладки прикладных программ. Формализация методов и приемов технологии ресурсонезависимого программирования позволяет автоматизировать и сократить время портации прикладных программ для всего многообразия возможных архитектур и конфигураций ГВС.

Научная новизна диссертации заключается в том, что в ней:

- впервые предложена единая форма представления параллельных вычислений для различных целевых архитектур гибридной вычислительной системы – процессоров, графических ускорителей и ПЛИС, основанная на использовании информационного графа задачи, инвариантного к различным способам описания прикладной задачи;

- для отображения информационного графа задачи как единой формы параллельных вычислений на конфигурацию гибридной вычислительной системы впервые предложена максимально параллельная кадровая структура, характеризуемая числом информационно-независимых и информационно-зависимых подграфов, разрядностью обрабатываемых данных, числом команд и обладающая латентностью, интервалом обработки данных, длиной потоков данных и др.;

- для адаптации параллельных вычислений к архитектуре и конфигурации гибридной вычислительной системы предложено преобразование максимально параллельной кадровой структуры, параметризующее число информационно-независимых и информационно-

зависимых подграфов, разрядность обрабатываемых данных, число команд и интервал обработки значением доступного аппаратного ресурса;

- установлено, что преобразование максимально параллельной кадровой структуры для заданной конфигурации гибридной вычислительной системы может быть выполнено с помощью механизма методов редукции производительности и аппаратных затрат критического ресурса;

- установлено, что преобразование требуемого для реализации кадровой структуры критического аппаратного ресурса гибридной вычислительной системы происходит при движении в пространстве возможных реализаций кадровых структур к точке выбора альтернатив, где происходит смена критического ресурса и направления движения;

- получены оценки минимального и максимального числа точек выбора альтернатив для преобразования кадровой структуры к архитектуре и конфигурации гибридной вычислительной системы;

- показано, что максимальное число точек выбора альтернатив преобразования максимально параллельной кадровой структуры к архитектуре и конфигурации гибридной вычислительной системы значительно меньше числа вариантов, анализируемых распараллеливающим компилятором для систем с распределенной памятью;

- впервые разработаны методы преобразования кадровой структуры для случая, когда аппаратного ресурса вычислительной системы недостаточно для реализации минимальной кадровой структуры;

- впервые предложена новая форма организации вычислений – микро-кадры, являющиеся промежуточной между структурной и процедурной формами организации вычислений, позволяющие сократить занимаемый аппаратный ресурс по сравнению с минимальной кадровой структурой и время выполнения по сравнению с процедурной реализацией на процессоре;

- впервые разработан метод преобразования кадровой структуры задачи на основе редукции производительности и аппаратных затрат, в том числе для

случая, когда аппаратного ресурса вычислительной системы недостаточно для реализации минимальной кадровой структуры;

- установлено, что для эффективной адаптации кадровой структуры к архитектуре и конфигурации гибридной вычислительной системы необходимо согласованное преобразование связанных информационной зависимостью подзадач с различной степенью параллелизма;

- впервые разработан метод преобразования прикладных задач, содержащих связанные информационной зависимостью подзадачи с различной степенью параллелизма;

- впервые получены формализованные алгоритмы реализации разработанных методов преобразования кадровой структуры, позволяющие автоматизировать их применение;

- на основе сформулированных принципов единого представления параллельных вычислений, предложенных теоретических основ и разработанных методов адаптации и преобразования кадровых структур впервые разработаны программные средства технологии ресурсонезависимого программирования гибридных вычислительных систем;

- проведена практическая апробация разработанной технологии ресурсонезависимого программирования при решении прикладных задач различных предметных областей на ряде действующих образцов гибридных вычислительных систем, содержащих процессоры и ПЛИС, подтвердившая эффективность предложенных методов по сравнению с известными решениями этих задач.

Все разработанные методы преобразования кадровой структуры формализованы, адаптированы для автоматического применения и реализованы в виде инструментальных средств технологии ресурсонезависимого программирования. Построенная на базе разработанных теоретических основ технология ресурсонезависимого программирования гибридных вычислительных систем апробирована при решении ряда прикладных задач математической физики, символьной и цифровой

обработки сигналов, содержащих связанные информационной зависимостью подзадачи с разной степенью параллелизма. Корректность разработанных методов преобразования кадровой структуры и функциональность программных средств поддержки технологии ресурсонезависимого программирования подтверждены успешным решением типовых задач с наиболее распространенными видами информационной зависимости.

Таким образом, совокупность предложенных принципов, моделей, положений, доказанных теорем и сформулированных следствий, разработанных методов и алгоритмов, объединенных в теоретические основы, позволили создать технологию программирования гибридных вычислительных систем различных архитектур и конфигураций, обеспечивающую сокращение время портации прикладных задач при заданном уровне реальной производительности. Предложенные теоретические положения ресурсонезависимого программирования гибридных вычислительных систем можно квалифицировать как новое научное достижение, а разработанные методы и алгоритмы – как новые научно обоснованные технические решения, внедрение которых вносит значительный вклад в развитие страны.

Практическая значимость работы и полученные результаты эффективности портации ресурсонезависимых программ подтверждаются актами внедрения результатов в академических, научно-исследовательских и промышленных организациях. Внедрение полученных в диссертации результатов позволяет сократить сроки и стоимость портации прикладных задач для существующих и перспективных образцов гибридных вычислительных систем. Внедрение разработанной технологии ресурсонезависимого программирования гибридных вычислительных систем открывает возможность создания автоматического распараллеливающего компилятора для систем с распределенной памятью, позволяющего сократить время создания и портации программ на процедурных языках программирования в ресурсонезависимые параллельно-конвейерные решения

для гибридных вычислительных систем. Помимо прямого экономического эффекта, предложенные в диссертации новые решения, с учетом рекомендаций по использованию полученных научных положений, имеют важное значение для развития критической технологии «Технологии и программное обеспечение распределенных и высокопроизводительных вычислительных систем».

Основные научные результаты диссертации опубликованы в [85-102, 109, 167, 168, 175, 178-180, 186-199, 208-214].

Все научные результаты, полученные при решении научной проблемы создания теоретических основ технологии ресурсонезависимого программирования гибридных вычислительных систем с разнородными по архитектуре вычислительными узлами, **получены автором лично**.

СПИСОК БИБЛИОГРАФИЧЕСКИХ ИСТОЧНИКОВ

1. Воеводин, В.В. Параллельные вычисления / В.В. Воеводин, Вл.В. Воеводин. - СПб. : БХВ-Петербург, 2002. - 608 с.: ил. ISBN 5-94157-160-7. – Текст : непосредственный.

2. Аладышев, О.С. СуперЭВМ: области применения и требования к производительности / О.С. Аладышев, Н.И. Дикарев, А.П. Овсянников [и др.]. - Текст : непосредственный // Известия высших учебных заведений. Серия: Электроника. – 2004. – № 1. – С. 13-17.

3. Moore, Gordon. No Exponential is Forever: But “Forever” Can Be Delayed! [Текст] / Gordon Moore // International Solid-State Circuits Conference (ISSCC), 2003. Session 1. Plenary 1.1. of the IEEE, november 2003, - Vol. 91. - No. 11. Pp. 1934-1939.

4. Intel объявила новый план: переименовать техпроцессы и перегнать TSMC за четыре года // https://3dnews.ru/1045230/intel-obyavila-noviy-plan-pereimenovat-tehprotsessi-i-dognat-tsmc-cherez-chetire-goda (дата обращения 06.09.2022). – Текст : электронный.

5. Антонов А.С., Афанасьев И.В., Воеводин Вл.В. Высокопроизводительные вычислительные платформы: текущий статус и тенденции развития / А.С. Антонов, И.В. Афанасьев, Вл.В. Воеводин - Текст : непосредственный // Вычислительные методы и программирование – 2021 – Вып. **22**. – с. 135-177. doi 10.26089/NumMet.v22r210

6. Гузик, В.Ф. Реконфигурируемые вычислительные системы : учебное пособие / В.Ф. Гузик, И.А. Каляев, И.И. Левин. – Ростов-на-Дону: ЮФУ, 2016. - 472 с. - ISBN 978-5-9275-1918-7. - Библиогр.: с. 461-471. - Текст : непосредственный.

7. Heterogeneous Processing: a Strategy for Augmenting Moore's Law // https://www.linuxjournal.com/article/8368 (дата обращения 06.09.2022). – Текст : электронный.

8. Digital Convergence and Human Intelligence // https://2021.russianscdays.org/files/2021/pdf/keynote/MResch.pdf (дата обращения 06.09.2022). – Текст : электронный.

9. Finogeev, A.G., Parygin, D.S. & Finogeev, A.A. The convergence computing model for big sensor data mining and knowledge discovery. Hum. Cent. Comput. Inf. Sci. **7,** 11 (2017). https://doi.org/10.1186/s13673-017-0092-7

10. TOP500 https://www.top500.org/lists/top500/2022/06/ (дата обращения 06.09.2022). – Текст : электронный.

11. ORNL Launches Summit Supercomputer // https://www.ornl.gov/news/ornl-launches-summit-supercomputer (дата обращения 06.09.2022). – Текст : электронный.

12. Supercomputer Fugaku, named after Mt. Fuji, makes its debut // https://www.asahi.com/ajw/articles/13462617 (дата обращения 06.09.2022). – Текст : электронный.

13. The Sunway TaihuLight supercomputer: system and applications // https://www.sciengine.com/SCIS/doi/10.1007/s11432-016-5588-7 (дата обращения 06.09.2022). – Текст : электронный.

14. Federated architecture // https://en.wikipedia.org/wiki/Federated_architecture (дата обращения 06.09.2022). – Текст : электронный.

15. Конкретно о CMDB // https://www.osp.ru/os/2007/06/4339018? (дата обращения 06.09.2022). – Текст : электронный.

16. Федеративная архитектура СЭД: в чем преимущество? // https://www.cnews.ru/articles/federativnaya_arhitektura_sed_v_chem? (дата обращения 06.09.2022). – Текст : электронный.

17. Костенко В.А., Смелянский Р.Л. Проблемы построения бортовых комплексов с архитектурой интегрированной модульной авионики. Радиопромышленность. – 2016. – No 3. – С. 63–70.

18. Жоголев, Е.А. Технология программирования / Е.А. Жоголев. – М. : Научный мир, 2004. – 216 с. - Текст : непосредственный.

19. Иванова, Г.С. Технология программирования : учебник для вузов / Г.С Иванова. – М. : Изд-во МГТУ им. Н.Э. Баумана, 2002. – 320 с. - Текст : непосредственный.

20. Баула, В.Г. Архитектура ЭВМ и операционные среды [Текст]: учебник для вузов / В.Г. Баула, А.Н. Томилин, Д.Ю. Волканов. – М.: Академия, 2011. – 336 с.

21. Хорошевский, В.Г. Архитектура вычислительных систем [Текст]: учеб. пособие / В.Г. Хорошевский. – М.: МГТУ имени Н .Э. Баумана, 2008. – 520 с.

22. The Computer as von Neuman Planned It // IEEE Annals of the History of Computing. — 1993. — Vol. 15, no. 1

23. Гладких, Б.А. Информатика. Введение в специальность [Текст]: учебник для студентов вузов / Б.А. Гладких. - Томск: Научно-техн. литература, 2002. - 350 с.

24. Ульянов, М.В. Архитектуры процессоров [Текст]: учеб. пособие / М.В. Ульянов. - М.: МГАПИ, 2002. - 68 с.

25. Попов, А.Ю. Организация суперскалярных процессоров [Текст]: учеб. пособие по курсу «Организация ЭВМ» / А.Ю. Попов. - М.: МГТУ им. Н.Э. Баумана, 2011. – 57 с.

26. Озеров, С. Архитектура процессоров [Текст] / С. Озеров // Компьютерра, 2005. – 25.10.2005.

27. Dipert, B. ARM versus Intel: a successful stratagem for RISC or grist for CISC's tricks [Текст] / B. Dipert // EDN NETWORK, 2011. - №7.

28. Фрунзе, А. Процессоры для персональных компьютеров [Электронный ресурс] / А. Фрунзе // Радио, 1998. - № 4. URL: http://www.chipinfo.ru/literature/radio/199804/p33-35.html (дата обращения 24.04.2017).

29. Лобода, В.Г. Модели архитектуры MISC-процессора [Текст] / В.Г. Лобода, Д.Б. Ельчанинов, В.Ю. Цуканов // Радиоэлектроника и информатика, 1999. - № 1(6). – С. 85-89.

30. Петраков, И.Е. Модели и алгоритмы гибридизации стратегий кэширования [Текст] / И.Е. Петраков // Программные продукты, системы и алгоритмы, 2016. - №2. – С. 1-8.

31. Алан Джей Смит. "Проектирование кэш-памяти процессора". Proc. IEEE TENCON, 1987

32. Shaul Dar, Michael Franklin, Bjorn Jonsson, Divesh Srivastava, Michael Tan. Semantic Data Caching and Replacement // Proceedings of the 22nd VLDB Conference MumbaQBombay), India, 1996

33. Клинов, М.С. Особенности использования многоядерных процессоров в научных вычислениях [Текст] / М.С. Клинов, С.Ю. Лапшина, П.Н. Телегин, Б.М. Шабанов // Вестник Уфимского государственного авиационного технического университета. – Уфа: Изд-во ФГБОУ ВПО

«Уфимский государственный авиационный технический университет», 2012. - №6(51). – Т. 16. – С. 25-31.

34. Черняк, Л. Архитектура фон Неймана, реконфигурируемые компьютерные системы и антимашина [Электронный ресурс] // Открытые системы. СУБД, 2008. - №06. URL: http://www.osp.ru/os/2008/06/5340894 (дата обращения 06.09.2022).

35. John Backus. Can programming be liberated from the von Neumann style? a functional style and its algebra of programs. Commun. ACM 21, 8 (Aug. 1978), 613–641. https://doi.org/10.1145/359576.359579

36. Breaking the Von Neumann Bottleneck: A Key to Powering Next-Gen AI Apps // https://www.electronicdesign.com/technologies/embedded-revolution/article/21156009/gsi-technology-breaking-the-von-neumann-bottleneck-a-key-to-powering-nextgen-ai-apps (дата обращения 06.09.2022). – Текст : электронный.

37. Wulf, W.A. Hitting the Memory Wall: Implications of the Obvious [Текст] / W.A. Wulf, S.A. McKee // Computer Architecture News. Mar., 1995. - Vol. 23. - No. 1. - Pp. 20-24.

38. John L. Hennessy, David A. Patterson - Computer architecture, A Quantitative Approach (6th edition) Elsevier : Kaufmann, 2017, 936 с.

39. DDR4 SDRAM [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/DDR4_SDRAM (дата обращения 06.09.2022).

40. Алешинцев, А.В. Обзор технологических норм производства транзисторов [Текст] / А.В. Алешинцев, П.А. Холодков // T-Comm - Телекоммуникации и Транспорт, 2010. - № 10. – С. 88-91.

41. 10 лет до 10 нм: закон Мура все ещё работает [Электронный ресурс] // PCNews, 12.07.2008. URL: http://pcnews.ru/news/10-channalweb-intel-pat-gelsinger-100-tsmc-45-2009-1965-33-1971-1978-1989-1997-25-2005-65-pentium-233904.html (дата обращения 06.09.2022).

42. Bérut, Antoine. Information and Thermodynamics: Experimental Verification of Landauer’s Principle Linking Information and Thermodynamics [Электронный ресурс] / Antoine Bérut. URL: https://arxiv.org/pdf/1503.06537.pdf (дата обращения 06.09.2022)

43. Anderson, Neal G.. “Landauer’s limit and the physicality of information.” The European Physical Journal B 91 (2018): 1-9. (https://link.springer.com/article/10.1140/epjb/e2018-80716-6)

44. Концепция квитирования [Электронный ресурс]. URL: http://www.xserver.ru/computer/protokol/tcpip/6/3.shtml (дата обращения 06.09.2022).

45. Борисенко, Н. Синтезируемая модель арбитра для доступа к среде передачи данных [Текст] / Н. Борисенко // Компоненты и Технологии, 2011. - №8. – С. 168-176.

46. Колганов, А.С. Опыт применения механизма областей для поэтапного распараллеливания программных комплексов с помощью системы SAPFOR / А.С. Колганов. - Текст : непосредственный // Вычислительные методы и программирование. - 2020. – Вып. 21. – С. 388-404. doi 10.26089/NumMet.v21r432

47. Hartenstein, R.W. A High Performance Machine Paradigm Based on Auto-Sequencing Data Memory [Электронный ресурс] / R.W. Hartenstein, A.G. Hirschbiel, M. Riedmuller, K. Schmidt, M. Weber // HICSS-24, Hawaii Int. Conference on System Sciences, Koloa Hawaii, 1991. URL: http://hartenstein.de/xputer/Xputer-paper-036.pdf (дата обращения 06.09.2022).

48. Штромайер, Э. Рейтинг Top500 и прогресс высокопроизводительных вычислений [Текст] / Э. Штромайер, Д. Донгарра, Х. Саймон // Открытые системы. СУБД, 2016. - №. 1. – С. 10-13.

49. Yamazaki, I. Mixed-precision Cholesky QR Factorization and Its Case Studies on Multicore CPUs with Multiple GPUs [Текст] / I. Yamazaki, S. Tomov, J. Dongarra // *SIAM Journal of Scientific Computing*, 2015. - Pp. 307-330.

50. Тестов, А. Intel представит первые процессоры Xeon со встроенными FPGA в первом квартале 2016 [Электронный ресурс] / А. Тестов. URL: http://www.3dnews.ru/923925 (дата обращения 06.09.2022).

51. Edwards, G. Hybrid Threading: A New Approach for Performance and Productivity [Электронный ресурс] / G. Edwards. URL: https://www.micron.com/~/media/documents/products/white-paper/wp_convey_ht_overview.pdf (дата обращения 06.09.2022).

52. Орлов, С. Многоядерные процессоры в высокопроизводительных вычислениях [Текст] // Журнал сетевых решений, 2014. - № 1. – С. 42-46.

53. Черняк, Л. Закон Амдала и будущее многоядерных процессоров [Электронный ресурс] / Л. Черняк // Открытые системы. СУБД, 2009. - № 04. URL: https://www.osp.ru/os/2009/04/9288815/ (дата обращения 06.09.2022).

54. Андреев, С.С. Гибридный суперкомпьютер К-100: что дальше? [Текст] / С.С. Андреев [и др.] // Информационные технологии и вычислительные системы, 2012. - №2. – С. 29-35.

55. Богданов, А.В. Высокопроизводительные вычисления на гибридных системах: будут ли решены «задачи большого вызова»? [Текст] / А.В. Богданов, А.Б. Дегтярев, В.Н. Храмушин // Компьютерные исследования и моделирование, 2015. - Т. 7. - № 3. – С. 429-437.

56. Климов, А.В. Суперкомпьютеры, иерархия памяти и потоковая модель вычислений [Текст] / А.В. Климов, Н.Н. Левченко, А.С. Окунев, А.Л. Стемпковский // Программные системы: теория и приложения, 2014. - Т. 5. - № 1(19). – С. 15-36.

57. Горбунов, В.С. Комплексная методика тестирования производительности суперкомпьютеров [Текст] / В.С. Горбунов, Л.К. Эйсымонт // Вычислительные методы и программирование, 2013. – Т. 14. - № 4. – С. 115-121.

58. Hoare, T. Grand Challenges in Computing Science [Электронный ресурс] / T. Hoare, R. Milner // Microsoft Research Ltd., Cambridge, UK. URL: http://www2.fiit.stuba.sk/~bielik/courses/psi-slov/slajdy/grand-challenges-for-computing-research.pdf (дата обращения 06.09.2022).

59. Allan, R. Computing Grand Challenges [Электронный ресурс] / R. Allan. URL: http://www.grids.ac.uk/Complex/grand_challenges/ (дата обращения 06.09.2022).

60. Шах, А. Пять причин, побудивших Intel купить Altera [Электронный ресурс] // Открытые системы, 15.06.2015 URL: http://www.osp.ru/news/articles/2015/24/13046230/ (дата обращения 06.09.2022).

61. Sodani, Avinash et al. Knights Landing: Second-Generation Intel Xeon Phi Product (англ.) // IEEE Micro: journal. — 2016. — Vol. 36, no. 2. — P. 34—46. — doi:10.1109/MM.2016.25.

62. Mittal, S. A survey on evaluating and optimizing performance of Intel Xeon Phi. *Concurrency Computat Pract Exper*. 2020; 32:e5742. https://doi.org/10.1002/cpe.5742

63. Horowitz, Paul; Hill, Winfield (2015). Horowitz P., Hill W. - The Art of Electronics. New York. p. 764.

64. Holdsworth, B.; Woods, R. C. (2003). Digital Logic Design (4th ed.). Elsevier.

65. Соловьев, В.В. Архитектуры ПЛИС фирмы XILINX: CPLD и FPGA 7-й серии [Текст] / В.В. Соловьев. - М.: Горячая Линия-Телеком, 2016. - 392 с.

66. Тарасов, И. ПЛИС XILINX семейства Ultrascale+ и перспективы их применения [Текст] / И. Тарасов // Компоненты и технологии, 2015. - № 4 (165). – С. 49-52.

67. UltraScale Architecture and Product Data Sheet: Overview [Электронный ресурс] // Xilinx, 2017. URL: http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf (дата обращения 06.09.2022).

68. Vanderbauwhede, W. High-Performance Computing Using FPGAs [Текст] / W. Vanderbauwhede, K. Benkrid [Eds.]. – New York: Springer-Verlag, 2013. - Pp. 803.

69. Cray [Электронный ресурс] // Cray-Inc., 2017. URL: http://www.cray.com (дата обращения 06.09.2022).

70. Cray XT4 Supercomputer Debuts with Petascale Capability and Significant Backlog of Large Orders [Электронный ресурс] // Cray-Inc., 2017. URL: http://investors.cray.com/phoenix.zhtml?c=98390&p=irol-newsArticle&ID=930002 (дата обращения 06.09.2022).

71. Silicon Graphics. Новые продукты [Электронный ресурс] // САПР и графика, 2006. - №6. URL: http://sapr.ru/article/ (дата обращения 06.09.2022).

72. Jamro, E. Computation Acceleration on SGI RASC: FPGA Based Reconfigurable Computing Hardware [Текст] / E. Jamro // Computer Science, 2008. - Vol. 9. - Pp. 21-34.

73. Choose the Right FPGA Card for Your Application [Электронный ресурс] // Nallatech, 2017. URL: http://www.nallatech.com/solutions/fpga-cards/ (дата обращения 06.09.2022).

74. George, A. Novo-G: At the Forefront of Scalable Reconfigurable Supercomputing [Электронный ресурс] / A. George, H. Lam, G. Stitt // Computing in science and engineering, 2011. - № 11. URL:

http://www.ann.ece.ufl.edu/other_pubs/CiSE-Novo-G-article.pdf (дата обращения 06.09.2022).

75. Каляев А.В., Левин И.И. Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений. М.: Янус-К; 2003, 380 с.

76. Корнеев, В. Будущее высокопроизводительных вычислительных систем [Электронный ресурс] / В. Корнеев // Открытые системы. СУБД, 2003. - № 05. URL: https://www.osp.ru/os/2003/05/183009 (дата обращения 06.09.2022).

77. SciEngines Products and Services. Reconfigurable High-Performance Computing Solutions [Электронный ресурс] // SciEngines. URL: http://www.sciengines.com/ (дата обращения 06.09.2022)].

78. Big FPGA Boards - ASIC Prototyping - High Performance Computing [Электронный ресурс] // DiniGroup. URL: http://www.dinigroup.com/web/index.php (дата обращения 06.09.2022).

79. The Xputer Index Page [Электронный ресурс] // Xputer. URL: http://xputers.informatik.uni-kl.de/xputer/index_xputer.html (дата обращения 06.09.2022).

80. The Trusted Leader in High Performance Computing [Электронный ресурс] // SGI. URL: http://www.sgi.com/products (дата обращения 06.09.2022).

81. Reilly, M. Mathstar Raises $10M on New Chip [Текст] / M. Reilly // Minneapolis, St. Paul Business Journal, Friday, April, 2, 2004.

82. Cryptanalysis with COPACOBANA [Текст] // Transactions on Computers, Nov. 2008. - Volume 57. - Pp. 1498-1513.

83. RIVYERA S6-LX150 DDS Desktop Development System FPGA-computers in small to medium configurations [Электронный ресурс] // SciEngines, 2017. URL: http://www.sciengines.com/products/development-systems/rivyera-dds.html (дата обращения 06.09.2022).

84. Каляев И.А., Левин И.И., Семерников Е.А., Дордопуло А.И. Высокопроизводительные многопроцессорные системы с реконфигурируемой архитектурой для цифровой обработки сигналов // Вестник Концерна ПВО Алмаз-Антей. 2011. № 2 (6). С. 88-104.

85. Dordopulo A.I., Levin I.I., Kalyaev I.A., Slasten L.M. High-performance reconfigurable computer systems based on VIRTEX-6 AND VIRTEX-

7 FPGAs // Numerical algebra with applications (CRC-NAA13). The second China-Russia conference: Abstracts of Lecturers and Young Scientists. Editors: Zhong-Ci SHi, Lev A. Krukier. 2013. С. 95-99.

86. Дордопуло А.И., Левин И.И., Каляев И.А., Гудков В.А. Высокопроизводительные реконфигурируемые вычислительные системы на основе ПЛИС VIRTEX-7 // Программная инженерия. 2014. № 6. С. 3-7.

87. Каляев, И.А. Реконфигурируемые вычислительные системы на основе ПЛИС семейства Virtex-6 [Текст] / И.А. Каляев, И.И. Левин, А.И. Дордопуло, Е.А. Семерников // Вестник УГАТУ. – Уфа: УГАТУ, 2011. - Т. 15. - № 5(45). – С. 148-154.

88. Левин И.И., Дордопуло А.И., Доронченко Ю.И., Раскладкин М.К., Федоров А.М. Реконфигурируемая вычислительная система на основе ПЛИС с жидкостным охлаждением // Вычислительные методы и программирование. 2016. Т. 17. № 1. С. 111-120. DOI: 10.26089/NumMet.v17r111

89. Левин И.И., Дордопуло А.И., Доронченко Ю.И., Раскладкин М.К., Федоров А.М. Погружная система охлаждения реконфигурируемых вычислительных систем на основе ПЛИС // Программные системы: теория и приложения. 2016. Т. 7. № 4 (31). С. 65-81. DOI: 10.25209/2079-3316-2016-7-4-65-81.

90. Kalyaev I.A., Levin I.I., Dordopulo A.I., Slasten L.M. FPGA-based reconfigurable computer systems // В сборнике: Proceedings of 2013 Science and Information Conference, SAI 2013. 2013. С. 148-155. https://ieeexplore.ieee.org/document/6661730

91. Гудков В.А., Гуленок А.А., Дордопуло А.И., Сластен Л.М. Средства программирования реконфигурируемых многопроцессорных вычислительных систем // Известия ТРТУ. 2006. № 16 (71). С. 16-20.

92. Dordopulo A., Kalyaev I., Levin I., Slasten L. High-performance reconfigurable computer systems // Lecture Notes in Computer Science. 2011. Т. 6873 LNCS. С. 272-283. DOI: 10.1007/978-3-642-23178-0_24.

93. Дордопуло, А.И. Высокопроизводительные реконфигурируемые вычислительные системы нового поколения [Текст] / А.И. Дордопуло, И.А. Каляев, И.И. Левин, Е.А. Семерников // Вычислительные методы и программирование. – М.: Издательский центр НИВЦ МГУ имени М.В. Ломоносова, 2011. – Т. 12. – С.82-89.

94. Levin I.I., Doronchenko Y.I., Raskladkin M.K., Fedorov A.M., Kalyaev Z.V., Dordopulo A.I. Immersion liquid cooling FPGA-based reconfigurable computer system // IFAC-PapersOnLine. 2016. Т. 49. № 25. С. 366-371. DOI: 10.1016/j.ifacol.2016.12.070.

95. Danilov I.G., Dordopulo A.I., Kalyaev Z.V., Levin I.I., Gudkov V.A., Gulenok A.A., Bovkun A.V. Distributed monitoring system for reconfigurable computer systems // В сборнике: Procedia Computer Science. 2016. С. 341-350. DOI: 10.1016/j.procs.2016.11.040.

96. Dordopulo, Alexey I. High-Performance Reconfigurable Computer Systems Based on Virtex FPGAs [Текст] / Alexey I. Dordopulo, Ilya I. Levin, Yuri I. Doronchenko, Maxim K. Raskladkin // Parallel Computing Technologies. 13th International Conference, PaCT 2015, Petrozavodsk, Russia, August 31-September 4, 2015, Proceedings. - Springer International Publishing, 2015. - Vol. 9251. - Pp. 349-362.

97. Levin I.I., Dordopulo A.I., Fedorov A.M., Gulenok A.A. Reconfigurable computer based on VIRTEX ULTRASCALE+ FPGAs with immersion cooling system // Communications in Computer and Information Science. 2017. Т. 753. С. 27-41. DOI: 10.1007/978-3-319-67035-5_3.

98. Levin I., Dordopulo A., Fedorov A., Doronchenko Y. Design of advanced reconfigurable computer systems with liquid cooling // Communications in Computer and Information Science. 2017. Т. 793. С. 442-455. DOI: 10.1007/978-3-319-71255-0_36.

99. Levin I., Dordopulo A., Fedorov A., Doronchenko Y. High-performance reconfigurable computer systems with immersion cooling // Communications in Computer and Information Science. 2018. Т. 910. С. 62-76. DOI: 10.1007/978-3-319-99673-8_5.

100. Левин, И.И. Реконфигурируемая вычислительная система на основе ПЛИС Virtex UltraScale с жидкостным охлаждением [Текст] / И.И. Левин, А.И. Дордопуло, Ю.И. Доронченко, М.К. Раскладкин // Труды международной научной конференции «Параллельные вычислительные технологии (ПаВТ’2016). Parallel computational technologies (PCT’2016)», г. Архангельск, 28 марта-1 апреля 2016 г. – Челябинск: Издательский центр ЮУрГУ, 2016. – С.221-230. - ISBN 978-5-696-04663-1.

101. Каляев, И.А. Развитие отечественных многокристальных реконфигурируемых вычислительных систем: от воздушного к жидкостному охлаждению [Текст] / И.А. Каляев, А.И. Дордопуло, И.И. Левин, А.М. Федоров // Труды СПИИРАН. - Санкт-Петербург: Изд-во СПИИРАН ФГБУН Санкт-Петербургский институт информатики и автоматизации РАН, 2017. – №1(50). – С. 5-31. - DOI 10.15622/sp.50.1

102. Каляев, И.А. Реконфигурируемые мультиконвейерные вычислительные структуры [Текст] / И.А. Каляев, И.И. Левин, Е.А. Семерников, В.И. Шмойлов. - Ростов-на-Дону: ЮНЦ РАН, 2008. - 397 с.

103. Валиев К. А., А. А. Кокин. Квантовые компьютеры: надежды и реальность. — Ижевск: РХД, 2004. — 320 с.

104. Степаненко С.А. Фотонный компьютер: структура и алгоритмы, оценка параметров. Фотоника № 7 / 67 / 2017. – С. 72–83. doi: 10.22184/1993-7296.2017.67.7.72.83.

105. Степаненко, С.А. Мультипроцессорные среды суперЭВМ [Текст] / С.А. Степаненко // Масштабирование эффективности. - М.: Физматлит, 2016. - 312 с.

106. Ершов А.П. Введение в теоретическое программирование. Беседы о методе. — М.: Наука, 1977/

107. Котов В. Е. Сети Петри. — М.: Наука, 1984. — 160 с.

108. Дордопуло А.И., Омаров О. М. Представление операторов параллельного программирования в унифицированной параллельной форме // Известия ТРТУ. Тематический выпуск «Интеллектуальные и многопроцессорные системы». – Таганрог: Изд-во ТРТУ,2004. – Т. 9. – С. 70-74.

109. Yonezawa, Akinori. ABCL: An Object-Oriented Concurrent System (Computer Systems Series) [Текст] / Akinori Yonezawa // The MIT Press (February 20, 1990). - 343 pp.

110. McCormick, John W. Building Parallel, Embedded, and Real-Time Applications with Ada [Текст] / John W. McCormick et al. // Cambridge University Press; 1 edition (April 7, 2011). - 388 pp.

111. Reppy, John H. Concurrent Programming in ML [Текст] / John H. Reppy // Cambridge University Press; 1 edition (September 17, 2007). - 328 pp.

112. Geraint, Jones. Programming in Occam [Текст] / Jones Geraint // Prentice-Hall International Series in Computer Science. Publisher: Prentice Hall (April 1, 1987) - ISBN: 0137297734. - Paperback: 182 pages.

113. Kerridge, Jon. Occam Programming: A Practical Approach (Computer Science Texts) [Текст] / Jon Kerridge // Alfred Waller Ltd (November 1987). – 172 pp. - ISBN-10: 0632016590, ISBN-13: 978-0632016594.

114. Bal, Henri E. Orca: A language for parallel programming of distributed systems [Электронный ресурс] / Henri E. Bal , M. Frans Kaashoek, Andrew S. Tanenbaum // IEEE Transactions on Software Engineering (1992). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.126.8912 (дата обращения 06.09.2022).

115. Fantom Programming Language [Электронный ресурс]. URL: http://fantom.org/ (дата обращения 06.09.2022).

116. Bohm, A.P.W. The SISAL 2.0 Reference Manual [Текст] / A.P.W. Bohm, R.R. Oldenhoeft, D.C. Cann, J.T. Feo // Livermore, CA, 1991. Prepr. / Lawrence Livermore Nat. Lab.; UCRL-MA-109098, LLNL.

117. Andrews, G.R. The SR Programming Language: Concurrency in Practice [Текст] / G.R. Andrews, and R.A. Olsson. - Benjamin/Cummings, Redwood City, CA, 1993. - 360 pp.)

118. MPI: The Message Passing Interface [Электронный ресурс]. http://parallel.ru/tech/tech_dev/mpi.html (дата обращения 06.09.2022).

119. Корнеев, В.Д. Параллельное программирование в MPI [Текст] / В.Д. Корнеев. - 2-е изд., испр. - Новосибирск: Изд-во ИВМиМГ СО РАН, 2002. - 215 с.

120. The OpenMP API specification for parallel programming [Электронный ресурс] http://www.openmp.org/ (дата обращения 06.09.2022).

121. Антонов, А.С. Параллельное программирование с использованием технологии OpenMP: учеб. пособие / А.С. Антонов. – М.: Изд-во МГУ, 2009. – 77 с.

122. High Performance Fortran Language Specification [Электронный ресурс] http://micro.ustc.edu.cn/Fortran/High Performance Fortran Language Specification hpf-v20-final.pdf (дата обращения 06.09.2022).

123. Кеннеди, Кен. Восход и закат High Performance Fortran: наглядный урок истории [Электронный ресурс] / Кен Кеннеди, Чарльз Коулбел, Ганс

Зима: пер. С. Кузнецова /Ken Kennedy, Charles Koelbel, Hans Zima, The Rise and Fall of High Performance Fortran: An Historical Object Lesson, Proceedings of the third ACM SIGPLAN conference on History of programming languages 2007, http://citforum.ru/programming/digest/hpf/ (дата обращения 06.09.2022).

124. Крюков, В.А. Разработка параллельных программ для вычислительных кластеров и сетей [Электронный ресурс] / В.А. Крюков // Информационные технологии и вычислительные системы. https://parallel.ru/sites/default/files/tech/articles/krukov-cldvm2002f.pdf (дата обращения 06.09.2022).

125. Система PVM [Электронный ресурс] http://www.opennet.ru/docs/RUS/linux_parallel/node211.html (дата обращения 06.09.2022).

126. PVM (Parallel Virtual Machine) [Электронный ресурс] http://www.csm.ornl.gov/pvm/ (дата обращения 06.09.2022).

127. Коновалов, Николай. Параллельные программы для вычислительных кластеров и сетей [Текст] / Николай Коновалов, Виктор Крюков // Открытые системы. СУБД, 2002. - № 03. – С. 12-18.

128. Андрианов, А.Н. Непроцедурный язык для решения задач математической физики [Текст] / А.Н. Андрианов, К.Н. Ефимкин, И.Б. Задыхайло // Программирование, 1991. - № 2. - С. 80-94.

129. Задыхайло И.Б. Содержательные обозначения и языки нового поколения [Текст] / И.Б. Задыхайло, К.Н. Ефимкин // Информационные технологии и вычислительные системы, 1996. - № 2. - С. 46-58.

130. FJTask // https://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/FJTask.html (дата обращения 06.09.2022)

131. Функциональная модель параллельных вычислений и язык программирования "Пифагор" // [ http://www.softcraft.ru/parallel/fpp/] (дата обращения 06.09.2022).

132. Легалов А. И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии : журнал. — 2005. — Т. 10, № 1. — С. 71-89.

133. Легалов А.И., Редькин А.В., Матковский И.В. Функционально-потоковое параллельное программирование при асинхронно поступающих

данных // Параллельные вычислительные технологии (ПаВТ'2009) : Труды международной научной конференции (Нижний Новгород, 30 марта – 3 апреля 2009 г.). — Челябинск: Изд. ЮУрГУ, 2009. — С. 573-578.

134. Patterson, David; Hennessy, John (2011). Computer Architecture: A Quantitative Approach (5th ed.). Morgan Kaufmann.

135. Бахтин В.А., Жукова О.Ф., Катаев Н.А. и др. Автоматизация распараллеливания программных комплексов. Тр. XI Всерос. науч. конф. “Научный сервис в сети Интернет”, Новороссийск, 2016. М.: ИПМ им. М.В. Келдыша РАН; 2016: 76–85.

136. Система автоматизированной параллелизации ФОРтран-программ (САПФОР). Адрес доступа: http://www.keldysh.ru/dvm/SAPFOR/ (дата обращения 06.09.2022).

137. Штейнберг Б.Я. Математические методы распараллеливания рекуррентных циклов для суперкомпьютеров с параллельной памятью. Ростов-на-Дону: Изд-во Ростовского гос. ун-та; 2004: 172.

138. Колганов А.С., Королев Н.Н. Статический анализ приватных переменных в системе автоматизированного распараллеливания Фортран-программ // Тр. Междунар. научной конференции “Параллельные вычислительные технологии” (ПаВТ’2018). Челябинск: Издательский центр ЮУрГУ, 2018. 286–294.

139. Клинов М.С., Крюков В.А. Автоматическое распараллеливание Фортран-программ. Отображение на кластер. Тр. XI Всерос. науч. конф. “Научный сервис в сети Интернет”, Новороссийск, 2009. М.: ИПМ им. М.В. Келдыша РАН; 2009:227–237.

140. Колганов А. С. Опыт применения механизма областей для поэтапного распараллеливания программных комплексов с помощью системы SAPFOR // Вычислительные методы и программирование. 2020. **21**. 388-404. doi 10.26089/NumMet.v21r432

141. Официальное руководство по программированию на CUDA, вер. 1.1 [Электронный ресурс] // CUDA Programming Guide. Chapter 1. Introduction to CUDA → 1.2 CUDA: A New Architecture for Computing on the GPU. http://docs.nvidia.com/cuda/#axzz4hnuUuogk (дата обращения 06.09.2022).

142. Еремеев, И.С. Параллельное программирование с использованием технологии CUDA: учеб. пособие [Электронный ресурс] / И.С. Еремеев,

А.В. Григорьев, М.И. Алексеева, В.А. Птицын. - Якутск: Издательско-полиграфический комплекс СВФУ, 2011. http://edu.chpc.ru/cuda/main.html (дата обращения 06.09.2022).

143. Боресков, А.В. Параллельные вычисления на GPU. Архитектура и программная модель CUDA: учеб. пособие [Текст] / А.В. Боресков и др. [послесл. В.А. Садовничий]. Серия «Суперкомпьютерное образование». – М.: Изд-во МГУ, 2012. - 336 с. ISBN: 978-5-211-06340-2.

144. Сандерс, Джейсон. Технология CUDA в примерах. Введение в программирование графических процессоров [Текст] / Джейсон Сандерс, Эдвард Кэндрот. - ДМК Пресс, 2011. - 232 стр. ISBN 978-5-94074-504-4, 978-0-13-138768-3.

145. Sherrod, Allen. Beginning DirectX 11 Game Programming [Текст] / Allen Sherrod. - Cengage Learning, 2011. – 384 с.

146. Дональд Херн, М. Паулин Бейкер. Компьютерная графика и стандарт OpenGL = Computer Graphics with OpenGL. — 3-е изд. — М.: Вильямс, 2005. — 1168 с.

147. Романенко, А.А. Особенности адаптации программ под GPU с использованием технологии OpenACC *[Текст]*: учеб. пособие / А.А. Романенко [Новосиб. гос. ун-т]. – Новосибирск: РИЦ НГУ, 2016. – 33 с. ISBN 978-5-4437-0479-1.

148. OpenACC online course // https://slidetodoc.com/openacc-online-course-lecture-1-introduction-to-open/ (дата обращения: 06.09.2022).

149. OpenCL 1.0 Specification [Электронный ресурс] / URL: http://www.khronos.org/ registry/cl/specs/opencl-1.0.48.pdf (дата обращения: 06.09.2022).

150. Gaster, Benedict R. Heterogeneous Computing with OpenCL Revised OpenCL [Текст] / Benedict R. Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, Dana Schaa // Advanced Micro Devices, Inc. Published by Elsevier Inc., 2013. - 308 p. - ISBN: 9780124055209, 9780124058941.

151. Лацис, А.О. Модели программирования для современных суперкомпьютеров [Текст] / А.О. Лацис // Программные системы: теория и приложения, 2010. – № 3(3). - С. 73-84. ISSN 2079-3316.

152. Mitrionics // https://web.archive.org/web/20070509171341/http://www.mitrionics.com/ (дата обращения: 06.09.2022).

153. Celoxica // https://celoxica.com/products/ (дата обращения: 06.09.2022).

154. Impulse Accelerated Technologies // https://web.archive.org/web/20110904033728/http://www.impulseaccelerated.com/ (дата обращения: 06.09.2022).

155. Языки высокого уровня ImpulseC, Mitrion-C и Handel-C // https://parallel.ru/fpga/lang.html (дата обращения: 06.09.2022).

156. R. Nane et al., "A Survey and Evaluation of FPGA High-Level Synthesis Tools," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 10, pp. 1591-1604, Oct. 2016.

157. R. Nane, V.-M. Sima, B. Olivier, R. Meeuws, Y. Yankova, K. Bertels. DWARV 2.0: A CoSy-based C-to-VHDL Hardware Compiler. In FPL, pp. 619-622. 2012.

158. C. Pilato and F. Ferrandi. Bambu: A Modular Framework for the High Level Synthesis of Memory-intensive Applications. In FPL, pp. 1–4, 2013.

159. A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J.H. Anderson, S. Brown, T. Czajkowski. LegUp: High-Level Synthesis for FPGA-based Processor/Accelerator Systems. In ACM FPGA, pp. 33-36. 2011.

160. . M. W. Numan, B. J. Phillips, G. S. Puddy and K. Falkner, "Towards Automatic High-Level Code Deployment on Reconfigurable Platforms: A Survey of High-Level Synthesis Tools and Toolchains," in *IEEE Access*, vol. 8, pp. 174692-174722, 2020, doi: 10.1109/ACCESS.2020.3024098.

161. Тарасов И. Проектирование для ПЛИС Xilinx с применением языков высокого уровня в среде Vivado HLS, Компоненты и технологии, №12’2013, https://kit-e.ru/fpga/vivado-hls/

162. Vitis Unified Software Platform Documentation. Application Acceleration Development. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug1393-vitis-application-acceleration.pdf, last accessed 2021/03/10.

163. Левин И.И., Дордопуло А.И., Гудков В.А. Ресурсонезависимое программирование многопроцессорных систем // Известия ТРТУ. 2006. № 9-1 (64). С. 174-178.

164. Дордопуло А.И., Левин И.И., Каляев И.А., Гудков В.А., Гуленок А.А., Евстафьев Г.А. Параллельно-конвейерная форма программы для программирования вычислительных систем гибридного типа // Вестник Уфимского государственного авиационного технического университета. 2016. Т. 20. № 3 (73). С. 122-128.

165. Дордопуло А.И., Левин И.И., Каляев И.А., Гудков В.А., Гуленок А.А. Программирование вычислительных систем гибридного типа на языке программирования COLAMO // Известия ЮФУ. Технические науки. 2016. № 11 (184). С. 39-54. DOI: 10.18522/2311-3103-2016-11-3954

166. Каляев И.А., Левин И.И. Реконфигурируемые вычислительные системы на основе ПЛИС : монография. – Ростов-на-Дону: Издательство ЮНЦ РАН, 2022. – 475 с. – ISBN 978-5-4358-0232-0

167. Баканов В.М. Программный инструментарий анализа информационной структуры алгоритмов по их информационным графам // Параллельные вычислительные технологии (ПаВТ’2016): Тр. междунар. науч. конф. Архангельск, 28 марта — 01 апреля 2016 г. Челябинск: Издательский центр ЮУрГУ, 2016 С. 432—441.

168. Волков Е.А. Численные методы. — М.: Физматлит, 2003

169. Левин И.И., Дордопуло А.И., Пелипец А.В. Реализация итерационных методов решения систем линейных уравнений в задачах математической физики на реконфигурируемых вычислительных системах // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2016. Т. 5, No 4. С. 5–18. DOI: 10.14529/cmse160401

170. Дордопуло А.И., Гудков В.А., Гуленок А.А., Коваленко В.Б. Объекты в иерархии софт-архитектур реконфигурируемых вычислительных систем // Известия ЮФУ. Технические науки. 2014. № 12 (161). С. 120-128.

171. Левин И.И., Дордопуло А.И., Коваленко В.Б., Гудков В.А., Гуленок А.А. Средства программирования реконфигурируемых вычислительных систем на основе ПЛИС VIRTEX-7 с использованием софт-архитектур // Вестник Южно-Уральского государственного университета.

Серия: Вычислительная математика и информатика. 2015. Т. 4. № 2. С. 20-32. DOI: 10.14529/cmse150202

172. Dordopulo A.I., Gudkov V.A., Gulenok A.A., Slasten L.M., Pelipets A.V., Kovalenko V.B., Chkan A.V. Parallel application porting by means of soft-architectures // IFAC-PapersOnLine. 2016. Т. 49. № 25. С. 372-377. DOI: 10.1016/j.ifacol.2016.12.071.

173. Каляев И.А., Дордопуло А.И., Левин И.И., Гудков В.А., Гуленок А.А. Технология программирования вычислительных систем гибридного типа // Вычислительные технологии. 2016. Т. 21. № 3. С. 33-44.

174. Gulenok A.A., Dordopulo A.I., Levin I.I., Gudkov V.A. Hybrid computer system programming technology with adaptation and scaling of calculations // Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2017. Т. 6. № 1. С. 73-86. DOI: 10.14529/cmse170105.

175. Иванов, А.И. Методы и средства создания эффективного параллельно-конвейерного программного обеспечения вычислительных систем, построенных на основе плис-технологии : специальность 05.13.11 "Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей" : автореферат диссертации на соискание ученой степени кандидата технических наук / Иванов Андрей Игоревич. – Таганрог, 2005. – 19 с.

176. Гордон В.О., Семенцов-Огиевский М.А. Курс начертательной геометрии. — М.: Наука, 1971. — С. 22.

177. Смирнова О.О. Основы стратегического планирования Российской Федерации // М., ИД "Наука", 2013, - 302 с.

178. Раскладкин, М.К. Методы и средства автоматизированного сопряжения функциональных узлов и блоков в приложениях для реконфигурируемых вычислителей : специальность 05.13.11 "Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей" : автореферат диссертации на соискание ученой степени кандидата технических наук / Раскладкин Максим Константинович. – Таганрог, 2010. – 18 с.

179. Коваленко А.Г. Макроконвейерная реализация алгоритмов аутентификации на языке высокого уровня COLAMO // Известия ЮФУ. Технические науки. – 2012. – № 4(129). – С. 210-215.

180. Дудко С.А. Эквивалентные преобразования некоторых видов рекурсивных нелинейных вычислительных структур для эффективной реализации на реконфигурируемых вычислительных системах // Известия ЮФУ. Технические науки. – 2020. – № 7(217). – С. 107-121. – DOI 10.18522/2311-3103-2020-7-107-121.

181. Левин И.И., Сорокин Д.А., Мельников А.К., Дордопуло А.И. Решение задач с существенно-переменной интенсивностью потоков данных на реконфигурируемых вычислительных системах // Вестник компьютерных и информационных технологий. – 2012. – № 2(92). – С. 49-56.

182. Семерников Е.А., Семерникова Е.Е. Перестановки элементов последовательностей в быстрых алгоритмах цифровой обработки сигналов // Вестник Южного научного центра РАН. – 2006. – Т. 2. – № 4. – С. 25-30.

183. Сорокин Д.А., Матросов А.Ю., Семерникова Е.Е., Алексеев К.Н. Структурно-процедурная реализация алгоритма прогнозирования кратных волн на ПЛИС // Известия ЮФУ. Технические науки. – 2016. – № 12(185). – С. 16-28. – DOI 10.18522/2311-3103-2016-12-1628.

184. Дордопуло А.И., Левин И.И., Сорокин Д.А. Реализация докинга для молекулярного моделирования на реконфигурируемых вычислительных системах // Известия ЮФУ. Технические науки. – 2011. – № 7(120). – С. 217-224.

185. Дордопуло А.И., Левин И.И., Сорокин Д.А. Реализация докинга для молекулярного моделирования на реконфигурируемых вычислительных системах (часть 2) // Известия ЮФУ. Технические науки. 2012. № 7 (132). С. 234-239.

186. Дордопуло А.И., Сорокин Д.А. Методика сокращения аппаратных затрат в сложных системах при решении задач с существенно-переменной интенсивностью потоков данных // Известия ЮФУ. Технические науки. – 2012. – № 4(129). – С. 194-199.

187. Дордопуло А.И., Левин И.И., Сорокин Д.А. Оптимизация вычислительной структуры задач с переменной интенсивностью потоков

данных на реконфигурируемых вычислительных системах // Известия ЮФУ. Технические науки. – 2011. – № 12(125). – С. 232-238.

188. Дордопуло А.И. Применение методов редукции производительности для сокращения числа анализируемых вариантов параллельной программы // Вестник компьютерных и информационных технологий. 2019. № 9 (183). С. 43-49. DOI: 10.14489/vkit.2019.09.

189. Дордопуло А.И. Методика редукционных преобразований для сокращения числа анализируемых вариантов параллельной программы // Вестник компьютерных и информационных технологий. 2019. № 10 (184). С. 52-56. DOI: 10.14489/vkit.2019.

190. Левин И.И., Дордопуло А.И. К вопросу об автоматическом создании параллельных прикладных программ для реконфигурируемых вычислительных систем // Вычислительные технологии. 2020. Т. 25. № 1. С. 66-81. DOI: 10.25743/ICT.2020.25.1.005.

191. Dordopulo A.I., Levin I.I. Performance reduction for automatic development of parallel applications for reconfigurable computer systems // Supercomputing Frontiers and Innovations. 2020. Т. 7. № 2. С. 4-23. DOI: 10.14529/jsfi200201

192. Dordopulo A.I., Kovalenko V.B., Gudkov V.A., Slasten L.M. Porting of parallel applications to reconfigurable computer systems with various architectures and configurations // В сборнике: 2016 5th International Conference on Informatics,Electronics and Vision, ICIEV 2016. 5. 2016. С. 1122-1127. DOI: 10.1109/ICIEV.2016.7760174.

193. Дордопуло А.И., Левин И.И., Каляев И.А., Гудков В.А., Гуленок А.А. Программирование вычислительных систем гибридного типа на основе метода редукции производительности // В сборнике: Параллельные вычислительные технологии (ПаВТ2016). Труды международной научной конференции. 2016. С. 131-140.

194. Кнут Д., Грэхем Р., Паташник О. Конкретная математика. Основание информатики. 2-е изд. М.: Мир; Бином. Лаборатория знаний; 1998: 703.

195. Семерников Е.А., Семерникова Е.Е. Перестановки элементов последовательностей в быстрых алгоритмах цифровой обработки сигналов // Вестник Южного научного центра РАН. – 2006. – Т. 2. – № 4. – С. 25-30.

196. Карпенко А.П., Чернов С.К. Решение систем линейных алгебраических уравнений методом предобуславливания на графических процессорных устройствах // Машиностроение и компьютерные технологии. 2013. №01.

197. Боровков А.А. Теория вероятностей. — 5-е изд. — М.: Либроком, 2009. — 656 с.

198. Линейная и циклическая свертка https://ru.dsplib.org/content/conv/conv.html (дата обращения: 06.09.2022).

199. Свидетельство о государственной регистрации программы для ЭВМ № 2022660796 Российская Федерация. Программа портации кадровой структуры на гибридные вычислительные системы различных архитектур и конфигураций : № 2022619611 : заявл. 25.05.2022 : опубл. 09.06.2022 / И.И. Левин, В.А. Гудков, А.И. Дордопуло, А.А. Гуленок ; заявитель Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров».

200. Свидетельство о государственной регистрации программы для ЭВМ № 2022660797 Российская Федерация. Программа преобразования фрагментов кадровой структуры на языке COLAMO в микро-кадры : № 2022619494 : заявл. 25.05.2022 : опубл. 09.06.2022 / А. А. Гуленок, А.И. Дордопуло, И.И. Левин, В.А. Гудков ; заявитель Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров».

201. Свидетельство о государственной регистрации программы для ЭВМ № 2006613613 Российская Федерация. Транслятор с языка COLAMO: № 2006613613: / И.И. Левин, А.И. Дордопуло, В.А. Гудков ; заявитель НИИ МВС им. академика А.В. Каляева ЮФУ.

202. Свидетельство о государственной регистрации программы для ЭВМ № 2016661590 Российская Федерация. Программный модуль программных средств трансляции единого языка высокого уровня для различных типов вычислительных узлов многопроцессорной вычислительной системы гибридного типа : № 2016618869 : заявл. 17.08.2016 : опубл. 14.10.2016 / И.И. Левин, А.И. Дордопуло, В.А. Гудков, Г.А. Евстафьев ; заявитель федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет» (Южный федеральный университет).

203. Свидетельство о государственной регистрации программы для ЭВМ № 2017617475 Российская Федерация. Транслятор языка программирования COLAMO для реконфигурируемых вычислительных систем : № 2017613660 : заявл. 21.04.2017 : опубл. 05.07.2017 / В.А. Гудков, И.И. Левин, А.И. Дордопуло, Г.А. Евстафьев ; заявитель Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров» (ООО «НИЦ СЭ и НК»).

204. Свидетельство о государственной регистрации программы для ЭВМ № 2017617778 Российская Федерация. Синтезатор масштабируемых параллельно-конвейерных решений прикладных программ для реконфигурируемых вычислительных систем : № 2017613662 : заявл. 21.04.2017 : опубл. 12.07.2017 / А.А. Гуленок, И.И. Левин, А.И. Дордопуло, К. Н. Алексеев ; заявитель Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров» (ООО «НИЦ СЭ и НК»)

205. Свидетельство о государственной регистрации программы для ЭВМ № 2016663176 Российская Федерация. Программный модуль программных средств синтеза масштабируемых параллельно-конвейерных решений прикладных задач для многопроцессорной вычислительной системы гибридного типа : № 2016619086 : заявл. 23.08.2016 : опубл. 29.11.2016 / И.И. Левин, А.И. Дордопуло, А.А. Гуленок, А.В. Чкан ; заявитель федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет» (Южный федеральный университет).

206. Свидетельство о государственной регистрации программы для ЭВМ № 2007614794 Российская Федерация. Интегрированная среда разработки аппаратно-программных решений: № 2007614794: Зарегист. в Реестре программ для ЭВМ 21.11.2007 г. / И.И. Левин, А.И. Дордопуло, В.А. Гудков; заявитель НИИ МВС им. академика А.В. Каляева ЮФУ

207. Свидетельство о государственной регистрации программы для ЭВМ № 2011612843 Российская Федерация. Интегрированная среда разработки аппаратно-программных решений: № 2011612843: Зарегист. в Реестре программ для ЭВМ 08.04.2011 г./ И.И. Левин, А.И. Дордопуло, В.А. Гудков; заявитель ФАГУ ВПО «Южный федеральный университет»

208. Свидетельство о государственной регистрации программы для ЭВМ № 2017618283 Российская Федерация. Среда разработки прикладных программ для реконфигурируемых вычислительных систем : № 2017613663 : заявл. 21.04.2017 : опубл. 26.07.2017 / И.И. Левин, А.И. Дордопуло, А.В. Бовкун, А.В. Касаркин ; заявитель Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров» (ООО «НИЦ СЭ и НК»).

209. Levin I., Dordopulo A., Gudkov V., Gulenok A., Bovkun A., Yevstafiyev G., Alekseev K. Software Development Tools for FPGA-Based Reconfigurable Systems Programming // (2019) In: Voevodin V., Sobolev S. (eds) Supercomputing. RuSC-Days 2019. Communications in Computer and Information Sci-ence, vol 1129. Springer, Cham, 2019, DOI:https://doi.org/10.1007/978-3-030-36592-9_51.

210. Dordopulo, A.I., Levin, I.I., Gudkov, V.A., Gulenok, A.A. High-Level Synthesis of Scalable Solutions from C-Programs for Reconfigurable Computer Systems. In: Malyshkin, V. (eds) Parallel Computing Technologies. PaCT 2021. Lecture Notes in Computer Science(), vol 12942. Springer, Cham. https://doi.org/10.1007/978-3-030-86359-3_7

211. Левин И.И., Дордопуло А.И., Каляев И.А. [и др.] Современные и перспективные высокопроизводительные вычислительные системы с реконфигурируемой архитектурой // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. – 2015. – Т. 4. – № 3. – С. 24-39. – DOI 10.14529/cmse150303

212. Левин И.И., Дордопуло А.И., Каляев И.А., Гудков В.А. Высокопроизводительные реконфигурируемые вычислительные системы на основе ПЛИС VIRTEX-7 // В сборнике: Параллельные вычислительные технологии (ПАВТ2014). Труды международной научной конференции. Ответственные за выпуск: Л.Б. Соколинский, К.С. Пан. 2014. С. 131-139.

213. Levin I.I., Dordopulo A.I., Fedorov A.M., Kalyaev I.A. Reconfigurable computer systems: from the first FPGAs towards liquid cooling systems // Supercomputing Frontiers and Innovations. 2016. Т. 3. № 1. С. 22-40. DOI: 10.14529/jsfi160102.

214. Levin I.I., Kalyaev I.A., Gudkov V.A., Slasten L.M., Dordopulo A.I., Kovalenko V.B. FPGA-based reconfigurable computer systems for digital image

processing // International Journal of Circuits, Systems and Signal Processing. 2015. Т. 9. С. 27-32.

215. Kalyaev I.A., Levin I.I., Dordopulo A.I., Slasten L.M. Reconfigurable computer systems based on VIRTEX-6 and VIRTEX-7 FPGAs // В сборнике: 12th IFAC/IEEE International conference on programmable devices and embedded systems. 2013. С. 71-75. DOI: 10.3182/20130925-3-CZ-3023.00009.

216. Levin I., Dordopulo A., Fedorov A., Doronchenko Y. Design technology for reconfigurable computer systems with immersion cooling // Communications in Computer and Information Science. 2019. Т. 965. С. 554-564. DOI: 10.1007/978-3-030-05807-4_47/

217. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си = Applied Cryptography. Protocols, Algorithms and Source Code in C. — М.: Триумф, 2002. — 816 с.

218. Баричев С. Г., Гончаров В. В., Серов Р. Е. Основы современной криптографии — 3-е изд. — М.: Диалог-МИФИ, 2011. — С. 30—35. — 176 с.

219. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем. — М.: Мир, 1991. — 376 с.

220. Ильин В.А., Позняк Э.Г. Линейная алгебра: Учебник для вузов. — 6-е изд., стер. — М.: ФИЗМАТЛИТ, 2004. — 280 с.

221. Колмогоров А.Н., Фомин С.В. Элементы теории функций и функционального анализа, — М.: Наука, 2004 (7-е изд.).

222. PKCS #5: Password-Based Cryptography Specification https://www.ietf.org/rfc/rfc2898.txt

223. PBKDF2 https://ru.wikipedia.org/wiki/PBKDF2

224. Шеннон К. Работы по теории информации и кибернетике, М., ИЛ, 1963, с. 333-369

225. Левин И.И., Гудков В.А., Евстафьев Г.А., Дордопуло А.И. [и др.] Методика трансляции программ на языке С для реконфигурируемых и гибридных вычислительных систем на основе ПЛИС // Вестник компьютерных и информационных технологий. – 2019. – № 12(186). – С. 54-60. – DOI 10.14489/vkit.2019.12.pp.54-60.

226. Свидетельство о государственной регистрации программы для ЭВМ № 2022660467 Российская Федерация. Программа преобразования информационного графа задачи в ресурсонезависимую кадровую структуру на

языке Colamo: № 2022619531 : заявл. 25.05.2022 : опубл. 03.06.2022 / В.А. Гудков, И.И. Левин, А.И. Дордопуло, А.А. Гуленок ; заявитель Общество с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров».

227. Дордопуло А.И., Левин И.И., Гудков В.А., Гуленок А.А. Программные средства высокоуровневого синтеза для многокристальных реконфигурируемых вычислительных систем // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2022. Т. 11. № 3. С. 5-21. DOI: 10.14529/cmse220301

228. Дордопуло А.И., Левин И.И., Гудков В.А., Гуленок А.А., Бовкун А.В., Дудко С.А. Комплекс средств трансляции программ на языке С в программы на языке потока данных COLAMO // Известия ЮФУ. Технические науки. 2020. № 7 (217). С. 94-106. DOI 10.18522/2311-3103-2020-7-94-106.

229. Левин И.И., Гудков В.А., Евстафьев Г.А., Дордопуло А.И., Гуленок А.А., Бовкун А.В. Методика трансляции программ на языке С для реконфигурируемых и гибридных вычислительных систем на основе ПЛИС // Вестник компьютерных и информационных технологий. 2019. № 12 (186). С. 54-60. DOI: 10.14489/vkit.2019.12.

230. Levin I., Dordopulo A., Pisarenko I., Melnikov A. Set classification in SET@L language for architecture-independent programming of high-performance computer systems // В сборнике: Supercomputing. 2020. С. 461-472. DOI: 10.1007/978-3-030-64616-5_40.

231. Levin I.I., Dordopulo A.I., Pisarenko I.V., Melnikov A.K. Parallelization of algorithms in SET@L language of architecture-independent programming // Communications in Computer and Information Science. 2020. Т. 1263. С. 31-45. DOI: 10.1007/978-3-030-55326-5_3.

232. Левин И.И., Дордопуло А.И., Писаренко И.В., Мельников А.К. Подход к архитектурно-независимому программированию вычислительных систем на основе аспектно-ориентированного языка Set@l // Известия ЮФУ. Технические науки. 2018. № 3. С. 46–58

233. Levin I.I., Dordopulo A.I., Pisarenko I.V., Melnikov A.K. Aspect-oriented SET@L language for architecture-independent programming of high-performance computer systems // Communications in Computer and Information Science. 2019. Т. 1129. С. 517-528. DOI: 10.1007/978-3-030-36592-9_42.

234. Левин И.И., Дордопуло А.И., Писаренко И.В., Мельников А.К. Язык архитектурно-независимого программирования вычислительных систем SET@L // Вестник компьютерных и информационных технологий. 2019. № 3 (177). С. 48-56. DOI: 10.14489/vkit.2019.03.pp.048-056.

235. Levin I.I., Dordopulo A.I., Pisarenko I.V., Melnikov A.K. Objects of alternative set theory in SET@L programming language/ Lecture Notes in Computer Science. 2019. Т. 11657 LNCS. С. 18-31. DOI: 10.1007/978-3-030-25636-4_3.

236. Safonov V.O. Using Aspect-Oriented Programming for Trustworthy Software Development. New York: John Wiley & Sons, 2008. – 352 p.

237. Вопенка, П. Альтернативная теория множеств: Новый взгляд на бесконечность. Пер. со славац. – Новосибирск: Изд-во Института математики, 2004. – 612 с.

238. Levin I.I., Dordopulo A.I., Pisarenko I.V., Melnikov A.K. The SET@L programming language and its application for coding Gaussian elimination // Communications in Computer and Information Science. 2019. Т. 1063. С.45-57. DOI: 10.1007/978-3-030-28163-2_4.

239. Левин И.И., Дордопуло А.И., Писаренко И.В., Мельников А.К. Описание алгоритма решения систем линейных алгебраических уравнений методом Якоби на языке архитектурно-независимого программирования Set@l // Известия ЮФУ. Технические науки. 2018. № 5. С. 34–48.

240. Левин И.И., Дордопуло А.И., Писаренко И.В., Михайлов Д.В. Представление графов с ассоциативными операциями на языке программирования SET@L // Известия ЮФУ. Технические науки. – 2020. – № 3(213). – С. 98-111. – DOI 10.18522/2311-3103-2020-3-98-111.

241. Левин И.И., Дордопуло А.И., Писаренко И.В., Мельников А.К. Архитектурно-независимая программа быстрого преобразования Фурье на языке программирования SET@L // Вестник Рязанского государственного радиотехнического университета. – 2019. – № 68. – С. 28-36. – DOI 10.21667/1995-4565-2019-68-2-28-36.

242. Levin I.I., Mikhailov D., Dordopulo A.I., Pisarenko I., Melnikov A. Resource-independent description of information graphs with associative operations in SET@L programming language // Lecture Notes in Computer Science. 2021. Т. 12942 LNCS. С. 74-87. DOI: 10.1007/978-3-030-86359-3_6.

243. Levin I., Dordopulo A., Pisarenko I., Melnikov A. Transformation of graphs with associative operations in terms of the SET@L programming language // Communications in Computer and Information Science. 2021. T. 1437. C.45-58. DOI: 10.1007/978-3-030-81691-9_4.

244. Левин И.И., Писаренко И.В., Михайлов Д.В., Мельников А.К., Дордопуло А.И. Ресурсонезависимое описание информационных графов с дистрибутивными операциями на языке программирования SET@L // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2022. Т. 11. № 2. С. 5-17. DOI 10.14529/cmse220201.

245. Свидетельство о государственной регистрации программы для ЭВМ № 2022665483 Российская Федерация. Транслятор языка архитектурно-независимого программирования Set@l (прототип): № 2022664790 : заявл. 04.08.2022 : опубл. 17.08.2022 / И.И. Левин, В.А. Гудков, А.И. Дордопуло, Д.В. Михайлов; заявитель федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет».

246. Levin I.I., Kovalenko A.G.E., Doronchenko Y.I., Dordopulo A.I. Realization of problem of liquid filtering in porous medium on reconfigurable computer system // В сборнике: 2016 5th International Conference on Informatics, Electronics and Vision, ICIEV 2016. 5. 2016. С. 1117-1121. DOI: 10.1109/ICIEV/2016.7760173.

247. Мельников А.К., Левин И.И., Сластен Л.М. Оценка возможностей перспективных вычислительных технологий для расчета точных приближений распределений вероятностей значений статистик // Известия ЮФУ. Технические науки. – Ростов/Д: Изд-во ЮФУ, 2022. – №4 (228). - С. 6-19. DOI 10.18522/2311-3103-2022-4-50-62. DOI 10.18522/2311-3103-2022-4-50-62.

248. Мельников А.К., Левин И.И., Дордопуло А.И., Писаренко И.В. Анализ возможностей современных вычислительных технологий для расчета точных приближений распределений вероятностей значений статистик // Известия ЮФУ. Технические науки. 2021. № 7 (224). С. 6-19. DOI 10.18522/2311-3103-2021-7-6-19.

**Приложение 1**

## Редукционные преобразования типовых кадровых структур

Представленные в настоящем приложении редукционные и оптимизационные преобразования, если специально не оговорено иное, применяются к абсолютно параллельной кадровой структуре функционально-регулярного информационного графа задачи.

### П1.1 Редукционные преобразования кадровых структур по числу подграфов

Абсолютно-параллельная кадровая структура $G_a$, соответствующая функционально-регулярному ИГЗ, содержащая $N$ слоев и $It$ итераций, в которых расположены изоморфные конвейерные операционные устройства «#» с латентностью $lat$ и единичным интервалом обработки входных данных представлена на рис. П1.1. Абсолютно-параллельную кадровую структуру $G_a$ можно представить как информационно-независимое объединение $N$ слоевых подграфов $G_b$, каждый из которых содержит $It$ операционных устройств «#».

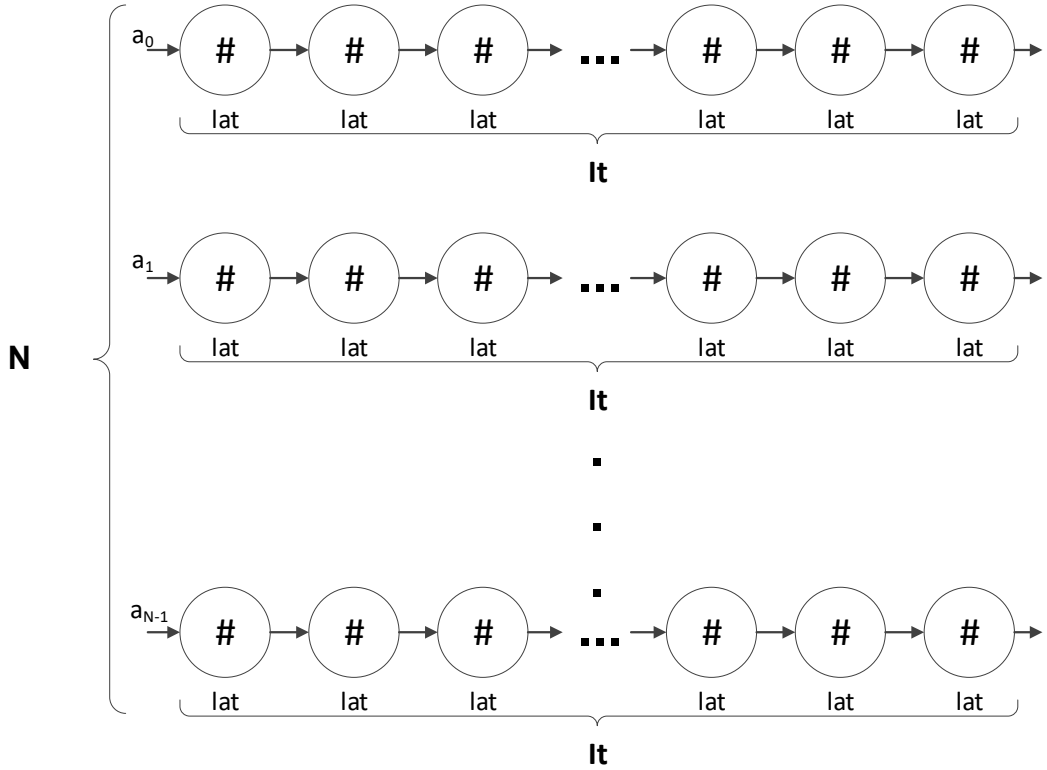



Рис. П1.1 Абсолютно-параллельная кадровая структура $G_a$ функционально-регулярного ИГЗ

Устройство (ступень) «#» может означать как отдельное устройство (сумматор, умножитель и т.д), так и группу устройств, связанных прямыми коммутационными связями (например, звено КИХ-фильтра «сумматор-умножитель», ступень «умножитель-сумматор» решения СЛАУ методом Гаусса-Зейделя и т.д.). Поскольку все операции в абсолютно-параллельной кадровой структуре выполняются одновременно для всех данных, а потоки отсутствуют, будем считать ее интервал обработки данных минимальным и равным 1: $I_a = 1$. Поскольку все $N$ итерационных структур информационно-независимы и работают одновременно, время обработки данных для $G_a$ составит

$$T_{\text{a}} = It \cdot lat \cdot \tau \qquad \text{(П1.1)},$$

где $\tau$ - длительность такта работы.

Применение редукции по числу слоевых подграфов $R^N$ с коэффициентом $R_b = N$ сократит число слоевых подграфов и кадровая структура примет вид $G_b$ (рис. П1.2).

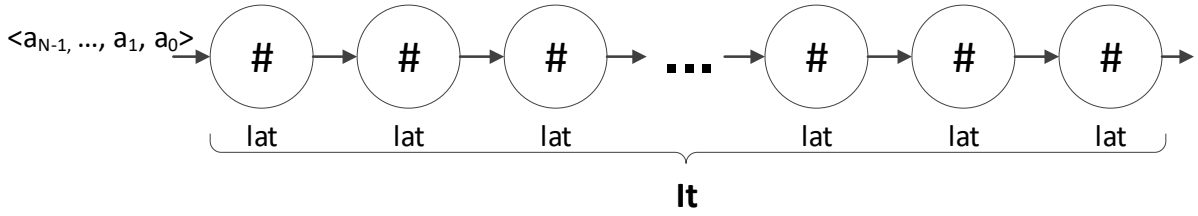



Рис. П1.2 Кадровая структура $G_b$ - результат редукции по числу слоевых подграфов $R^N$ с коэффициентом $R_b = N$

Как видно из рис. П1.2, число слоевых подграфов сократилось до одного, число итерационных вершин не изменилось, а входные данные абсолютно параллельной кадровой структуры (см. рис. П1.1), упорядочены и преобразованы в кортеж, элементы которого последовательно подаются на вход конвейерной кадровой структуры $G_b$. Для кадровой структуры $G_b$ латентность и интервал не изменятся ($Lat_b = It \cdot lat, I_b = 1$), но время обработки увеличится пропорционально числу редуцированных слоев $N$, данные которых образуют входной поток данных длины $N$:

$$T_b = It \cdot lat \cdot \tau + (N-1) \cdot \tau = (N - 1 + It \cdot lat) \cdot \tau \qquad \text{(П1.2)}$$

Если конвейерная структура работает с интервалом обработки данных больше 1 (I>1), то (П1.2) примет следующий вид

$$T_b = It \cdot lat \cdot \tau + (N-1) \cdot I \cdot \tau = \big((N-1) \cdot I + It \cdot lat\big) \cdot \tau \qquad (П1.3)$$

Дальнейшая редукция структуры $G_b$ по числу подграфов $R^N$ с коэффициентом $R_c = M < It$, приведет к сокращению числа информационно-зависимых операционных устройств «#» с $It$ до $It' = \left\lfloor \frac{It}{M} \right\rfloor$ в кадровой структуре $G_c$ (рис. П1.3).

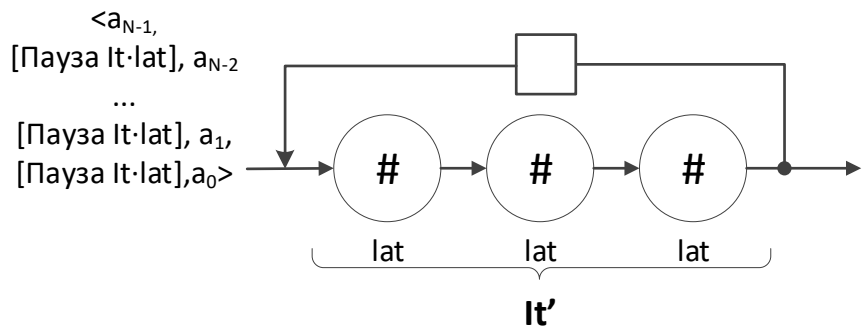



Рис. П1.3 Кадровая структура $G_c$ - результат редукции по числу итерационных подграфов $R^N$ с коэффициентом $R_c = M < It$

В кадровой структуре $G_c$ существенно, в $It \cdot lat$ раз, увеличится интервал обработки данных из-за необходимости выполнить $\left(\frac{It}{It'} - 1\right)$ проходов по цепи обратной связи для обработки одного данного. Поэтому входные данные кадровой структуры $G_c$ подаются с интервалом (паузой) длительностью $It \cdot lat$ тактов. Поэтому время обработки (П1.2) возрастет примерно в $It \cdot lat$ раз:

$$T_c = N \cdot It \cdot lat \cdot \tau \qquad (П1.4)$$

Для случая входного интервала обработки данных, больше 1 (I>1), (П1.4) примет вид

$$T_c = N \cdot I \cdot It \cdot lat \cdot \tau \qquad (П1.5)$$

При этом, как видно из формул (П1.4-П1.5) время обработки $T_c$ не зависит от числа реализованных ступеней $It'$, поскольку в соответствии с теоремой Иванова[168], при обработке одного (не потока!) данного на редуцированной итерационной структуре одна ступень с обратной связью

будет наиболее эффективной по удельной производительности кадровой структурой. Время решения (П1.4) одинаково для всех редуцированных итерационных структур, но аппаратные затраты будут минимальны при аппаратной реализации всего одной ступени, которая и будет наиболее рациональным способом организации и реализации вычислений.

## П1.2 Оптимизационные преобразования кадровых структур, редуцированных по числу подграфов

Для сокращения времени обработки данных кадровой структурой $G_c$(рис. П1.3) известен метод построения конвейера в конвейере (или конвейера конвейеров, вложенного конвейера), сокращающего интервал обработки данных с помощью конвейеризации [171, 172]. Поскольку каждое из устройств «#» является конвейерным, то на нем можно конвейерно обрабатывать не 1, а $lat$ данных, даже при наличии обратной связи. При объединении $It'$ реализованных устройств «#» в конвейерную структуру (см. рис. П1.3), размер обрабатываемого пакета данных растет кратно числу устройств. Для корректности получаемых результатов в цепь обратной связи добавляются $It' \cdot lat$ регистров с начальной инициализацией нейтральным для операции «#» значением, а на вход кадровой структуры $G_d$ (рис.П1.4) подается не 1 данное, а пакет из $It' \cdot lat$ данных с интервалом $I = 1$, после чего следует пауза длительностью $(It - It') \cdot lat$ тактов.

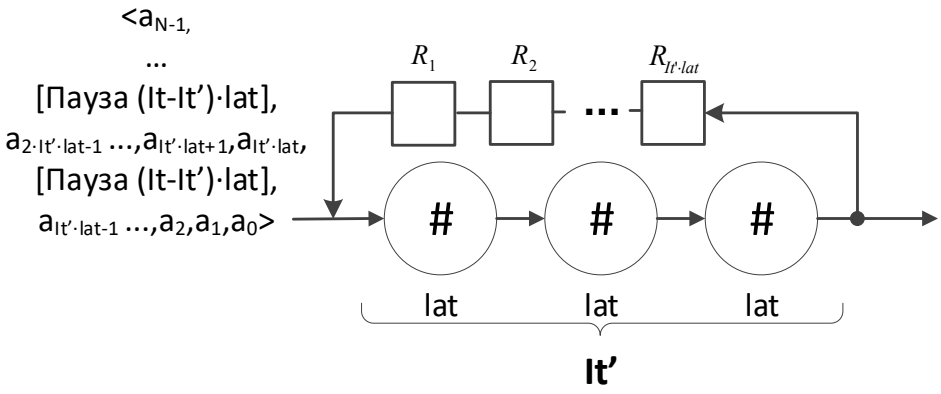


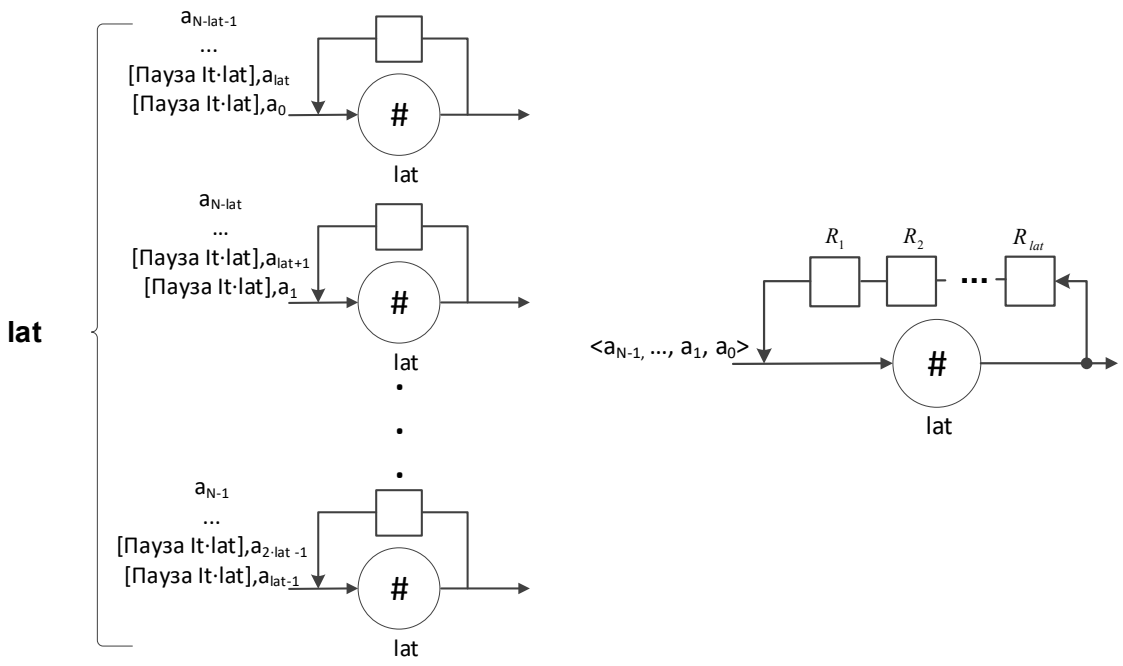

Рис. П1.4 Кадровая структура $G_d$ - оптимизация интервала обработки данных $G_c$ с помощью построения конвейера в конвейере

Входные данные пакета конвейерно обрабатываются всеми $It'$ ступенями, записываются в $It' \cdot lat$ регистров обратной связи и снова подаются на вход кадровой структуры $G_d$. Выигрыш по сравнению с кадровой структурой $G_c$ (рис. П1.3) состоит в том, что за время $It \cdot lat \cdot \tau$ обрабатывается пакет из $It' \cdot lat$ данных, а не 1. С помощью пропорции легко рассчитать общее время обработки, которое составит

$$T_d = \frac{N \cdot It \cdot \cancel{lat} \cdot \tau}{It' \cdot \cancel{lat}} = N \cdot \frac{It}{It'} \cdot \tau \qquad \text{(П1.6)},$$

При сравнении (П1.4) и (П1.6) становится очевидно, что организация конвейера в конвейере позволяет сократить время обработки данных структурой $G_c$ в $It' \cdot lat$ раз при незначительном росте аппаратных затрат (на $It' \cdot lat$ регистров в цепи обратной связи). Для конвейера в конвейере число реализованных устройств «#» $It'$ приобретает важное значение, поскольку, чем больше устройств реализовано, тем меньше время обработки (П1.6). Это преобразование часто используется инженерами-схемотехниками при решении прикладных задач, причем они выполняют оптимизацию не в общем виде, а для уже известных, заданных задачей, абсолютных значений $N, It, It', lat$ с учетом интенсивности данных в других фрагментах задачи.

Вычислительную структуру, аналогичную конвейеру в конвейере, можно построить также при оптимизации информационно-независимых (слоевых) мультиконвейерных кадровых структур с обратной связью, заменив $(lat - 1)$ устройств «#» регистрами в цепи обратной связи (рис. П1.5). Интервал обработки данных при этом сократится с $(It \cdot lat)$ в каждом устройстве «#» до $I = 1$ в объединенной структуре. Общее время обработки при этом не увеличится, а занимаемый кадровой структурой аппаратный ресурс и необходимое число каналов линейно сократятся. Освободившийся в результате преобразования аппаратный ресурс и число каналов можно использовать для рациональной реализации других фрагментов задачи.




а) б)

Рис. П1.5. Кадровая структура $G_e$ - оптимизация информационно-независимых кадровых структур (а) построением конвейера в конвейере (б)

Для структур с процедурной $g_{проц}$, а не конвейерной $g_{стр}$ обработкой данных, преобразование к конвейеру в конвейере неприменимо, но для сокращения времени обработки данных можно использовать макроконвейерную обработку данных, предложенную советским математиком и кибернетиком В.М. Глушковым[172]. Процедурное устройство «$\#_{pr}$» характеризуется числом тактов обработки $tproc$, численно равным интервалу обработки данных. Время обработки при процедурной реализации устройств «$\#_{pr}$» в структуре $G_b$ линейно возрастет пропорционально длине потока и составит

$$T_b = N \cdot It \cdot tproc \cdot \tau \qquad (П1.7)$$

Для сокращения времени обработки данных такой кадровой структурой целесообразно построить макроконвейер, который состоит в реализации не менее, чем $tproc$ процедурных устройств «$\#_{pr}$» таким образом, чтобы при подаче входных данных плотным потоком с интервалом $I = 1$

всегда было хотя бы одно свободное устройство, способное обработать поступившие данные (рис. П1.6).

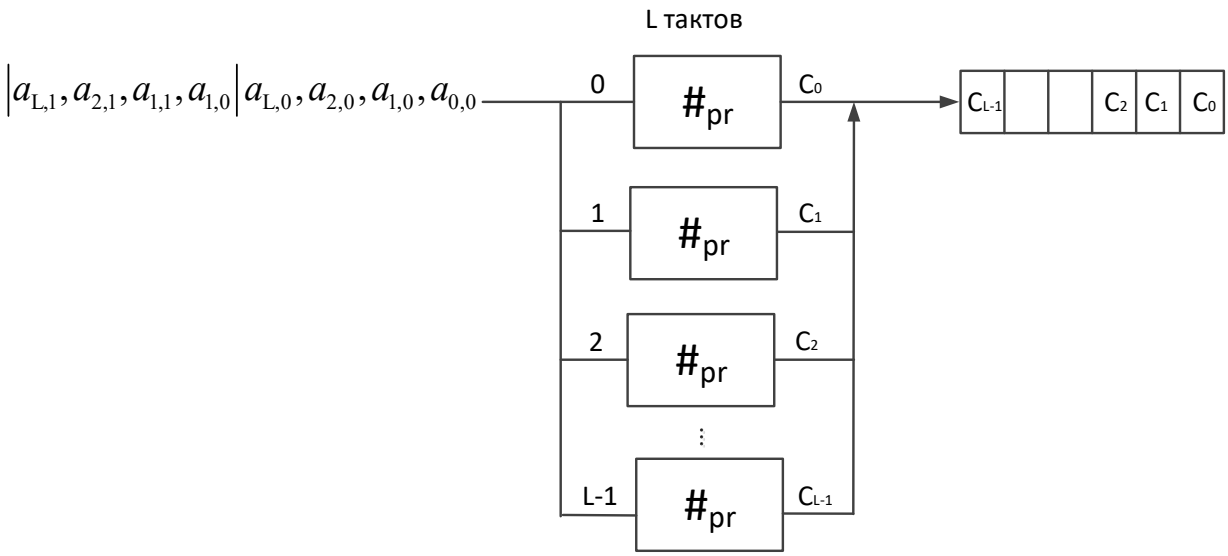


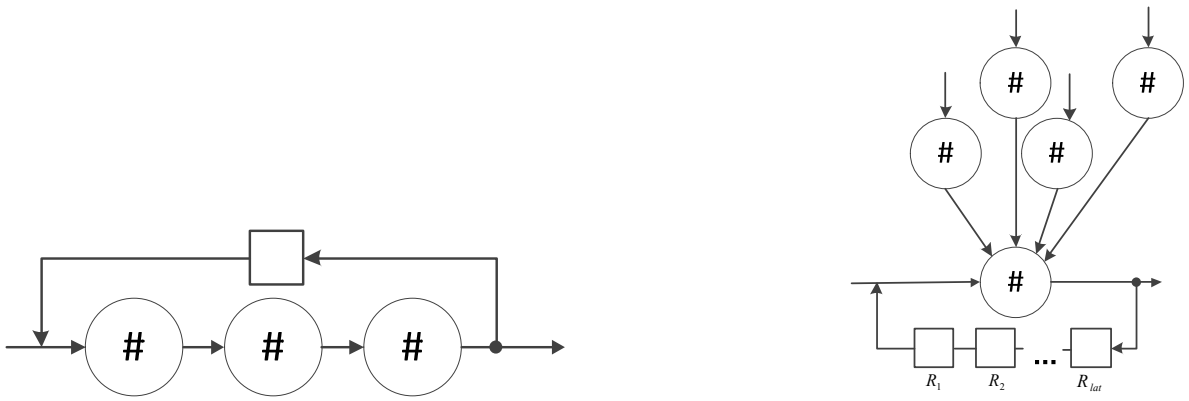

Рис. П1.6. Преобразование кадровой структуры к макроконвейеру

Очередное пришедшее данное распределяется на свободное процедурное устройство, которое обрабатывает его без взаимодействия с остальными. Поскольку число устройств больше или равно числу тактов его работы, после загрузки данных в последнее устройство освободится первое, что позволяет подавать данные плотным потоком с интервалом, равным 1. В результате из процедурных устройств формируется аналог конвейера, в котором каждое устройство обрабатывает собственную часть входных данных по своей программе, но все вместе они работают согласованно подобно ступеням конвейера.

Преобразование к макроконвейеру не задействует дополнительные каналы, но требует дополнительного аппаратного ресурса для аппаратной реализации $L - 1$ устройства, причем значение $L$ определяется параметрами задачи. Блоки «$\#_{pr}$» имеют существенно меньшую по сравнению с обычным процессором функциональность и, как правило, являются цифровыми автоматами, поэтому аппаратные затраты на их процедурную реализацию

невелики, но дополнительный аппаратный ресурс будет зависеть от их общего числа $L$.

Еще одним способом оптимизации интервала обработки данных является автоподстановка[173], применимая для конвейерных вычислительных структур с обратной связью и большим интервалом обработки данных. Автоподстановка[173] «раскрывает» обратную связь для сокращения времени обработки данных за счет использования дополнительного аппаратного ресурса. Суть преобразования состоит в увеличении числа информационно-зависимых ступеней итерационной структуры и сокращении числа регистров в цепи обратной связи, оптимизации получившейся вычислительной структуры и корректировке входных данных кадровой структуры. Автоподстановка требует достаточно большого аппаратного ресурса и целесообразна в случаях, когда требуется существенно сократить интервал обработки данных в фрагменте, охваченном обратной связью, при наличии большого доступного аппаратного ресурса. Этот метод (рис. П1.7) применим к различным операциям в структурах с обратной связью и раскрывает обратную связь с целью создания аналога конвейера в конвейере на входе структуры с обратной связью (рис. П1.7-а) с единичным интервалом обработки данных.




а) б)

Рис. П1.7. Преобразование кадровой структуры с помощью автоподстановки

а) исходная вычислительная структура с обратной связью;

б) оптимизированная структура с интервалом $I = 1$

Способы реализации этого метода для различных сочетаний ассоциативных и дистрибутивных операций подробно описаны в [173]. Следует отметить, что автоподстановка для своей реализации требует наиболее значительных дополнительных аппаратных затрат среди рассмотренных методов оптимизации: для сокращения интервала обработки данных в $L$ раз потребуется от $3 \cdot L$ до $4 \cdot L$ дополнительных устройств (рис. П1.7-б) из числа входящих в цепь обратной связи и $L$ дополнительных регистров в цепи обратной связи как в конвейере конвейеров, где $L$ - это суммарная латентность устройств, охваченных обратной связью. Поэтому применение автоподстановки оправданно в тех случаях, когда сокращение интервала обработки данных является наиболее приоритетной задачей. Такие ситуации возникают тогда, когда в результате преобразования кадровой структуры интервал обработки данных одного из фрагментов задачи существенно больше, чем у остальных. Необходимо учитывать, что автоподстановка неприменима для ряда задач из-за плохой устойчивости результирующей вычислительной структуры. Механизм выполнения автоподстановки, особенности ее применения, оценки времени, требуемого аппаратного ресурса, применимости и устойчивости для разных вычислительных структур приведены в [173].

Основные характеристики методов оптимизации, которые можно использовать для сокращения интервала обработки данных, собраны в таблице П1.1.

Таблица П1.1. Основные характеристики методов оптимизации кадровых структур

| **Метод оптимизации** | **Критерии применения (входная структура)** | **Влияние на аппаратные затраты и число каналов** | **Подача входных данных и время решения** |
|---|---|---|---|
| Конвей-еризация (рис. П1.3, П1.4) | Конвейерная вычислительная структура с $It'$итерационными ступенями и обратной связью | Аппаратные затраты незначительно увеличиваются (на $It' \cdot lat$регистров). Число каналов не меняется. Интервал сокращается до $\frac{It}{It'} \cdot lat$ | Данные подаются и обрабатываются пакетами длины $It' \cdot lat$. Время решения сокращается в $It'$ раз. |
| Конвейер в конвейере | Мультиконвейерная вычислительная структура из 1 ступени с обратной связью. Число слоев должно быть не меньше латентности ступени $lat$ | Аппаратный ресурс и число каналов сокращается в $lat$ раз | Данные подаются плотным потоком с интервалом, равным 1. Время решения не меняется |
| Макроконвейер | Процедурный вычислительный блок, число слоев не меньше максимального числа тактов работы блока | Число каналов не меняется, аппаратные затраты возрастают по числу слоев | Данные подаются плотным потоком с интервалом, равным 1. Время решения сокращается по числу слоев |
| Автоподстановка | Конвейерная вычислительная структура с обратной связью | Число каналов возрастает, аппаратный ресурс существенно возрастает (пропорционально $4 \cdot lat$) | |

Как видно из таблицы, наибольшим приоритетом по соотношению аппаратных затрат и сокращения времени обработки обладает метод построения конвейера в конвейере, потом конвейеризации, но для их применения необходима определенная структура информационных зависимостей задачи. Реализация макроконвейера (для процедурных блоков) и автоподстановки (для конвейерных структур) требуют дополнительного свободного аппаратного ресурса.

## П1.3 Редукционные преобразования по числу устройств

Редукция по числу устройств $R^{Op}$ сокращает число устройств в минимальной кадровой структуре за счет совмещения и последовательного

выполнения операций с одинаковыми типами данных в одном устройстве. При этом происходит переход от параллельной обработки входных данных к параллельно-последовательной с увеличением интервала обработки данных в редуцированной вычислительной структуре. Результатом редукции кадровой структуры по числу устройств $R^{Op}$ является микро-кадр или м-кадр, принципиальным отличием которого от структурно-процедурной организации вычислений является совмещение нескольких операций в одном вычислительном устройстве, приводящее к увеличению интервала обработки данных, но обеспечивающее единый и сбалансированный темп обработки данных без использования дополнительной памяти для хранения промежуточных результатов обработки входного потока данных. Предельный случай редукции по числу устройств $R^{Op}$ – это последовательная (процедурная) обработка данных, когда все операции минимальной кадровой структуры совмещаются в одном вычислительном устройстве – процессоре, который в зависимости от управляющей команды выполняет очередную операцию кадровой структуры.

В общем случае, м-кадр содержит более одного устройства и позволяет при структурно-процедурной реализации выполнить все вычисления исходной кадровой структуры с соблюдением информационной эквивалентности результатов вычислений. В отличие от кадровой структуры, представляющей собой вычислительный конвейер, в котором каждый такт обрабатывается очередная порция данных, м-кадр – это своего рода «универсальная конвейерная ступень» структурно-процедурной реализации вычислений базового подграфа. По аналогии с упорядочиванием слоевых подграфов можно считать, что в м-кадре входные данные упорядочиваются и последовательно обрабатываются универсальным для данной кадровой структуры вычислителем, обеспечивая информационную эквивалентность результатов вычислений. Поэтому для обработки пакета исходных данных м-кадру требуется несколько тактов работы «универсальной конвейерной

ступени», что кратно увеличивает интервал обработки данных и время выполнения вычислений.

Рассмотрим формализованную постановку задачи преобразования кадровой структуры, соответствующей базовому подграфу задачи, в м-кадр. Будем рассматривать кадровую структуру $\varphi = \varphi(\mathrm{P,Q,L,C,S,}\rho\mathrm{,Dt})$, содержащую **P** устройств (структурно реализованных вычислительных операций) и **Q** внутренних связей между вершинами, расположенных в **L** ярусах ярусно-параллельной формы (ЯПФ). При этом кадровая структура $\varphi$ требует **C** каналов памяти для конвейерной реализации с интервалом **I** при обработке данных разрядностью $\rho$ с типом данных $Dt$. Будем полагать, что в каждом ярусе ЯПФ может располагаться более одной вершины с одновременной обработкой данных, поэтому в общем случае число устройств больше числа ярусов: $P \geq L$. Очевидно, что латентность кадровой структуры $\varphi$ равна числу ярусов **L**, т.е. числу внутренних ступеней конвейера кадровой структуры. Кадровая структура $\varphi$ обрабатывает поток данных длины **N** в конвейере с длительностью такта $\tau$ с интервалом **I**=1.

Для описания и оценки параметров исходной кадровой структуры, м-кадра и процедурной (процессорной) реализации вычислений БП, будем использовать индексы «БП», «М» и «ПР» соответственно для следующих основных характеристик.

1) Время **T** обработки потока данных длины **N**: $T_{\text{БП}}$, $T_{\text{М}}$ и $T_{ПР}$.

2) Аппаратный ресурс **A**, занимаемый вычислительной структурой: $A_{\text{БП}}$, $A_{\text{М}}$ и $A_{ПР}$.

3) Число каналов памяти **C**, требуемое для реализации кадровой структуры: $C_{\text{БП}}$,$C_{\text{М}}$ и $C_{\text{ПР}}$. Это число включает входные каналы $C^{In}$ и выходные каналы $C^{Out}$.

4) Коэффициент **R** редукции производительности и аппаратных затрат для соответствующей кадровой структуры: $R_{\text{БП}}$, $R_{\text{М}}$ и $R_{ПР}$.

Коэффициент редукции производительности $R_{\text{БП}}$ по определению равен 1. Коэффициент редукции производительности при процедурной обработке

данных на одном устройстве $R_{ПР}$ по определению равен числу операций базового подграфа **P**. Коэффициент редукции производительности для м-кадра должен находиться в интервалах $R_{БП} < R_M < R_{ПР}$, $1 < R_M < P$.

Редукция производительности и аппаратных затрат отображает кадровую структуру $\varphi$ с параметрами $P_{БП}$,$Q_{БП}$,$L_{БП}$,$C_{БП}$,$S_{БП}$, $\rho_{БП}$,$Dt_{БП}$ в м-кадр $\mu$ с параметрами $P_M$,$Q_M$,$L_M$,$C_M$,$S_M$, $\rho_M$,$Dt_M$

$$M_{R_M}(\varphi):\ \varphi(P_{БП},Q_{БП},L_{БП},C_{БП},S_{БП},\rho_{БП},Dt_{БП}) \rightarrow \mu(P_M,Q_M,L_M,C_M,S_M,\rho_M,Dt_M)$$

М-кадр $\mu$, полученный в результате редукции производительности и аппаратных затрат кадровой структуры $\varphi(P_{БП}, Q_{БП}, C_{БП}, L_{БП}, R_{БП}, \rho_{БП}, Dt_{БП})$, представим в виде функциональной зависимости от числа устройств и коэффициента редукции:

$$\mu = M(\varphi(P_{БП}, Q_{БП}, C_{БП}, L_{БП}, R_{БП}, \rho_{БП}, Dt_{БП}), R_M)$$
$$= \varphi^{R_M}(P_M, Q_M, C_M, L_M, R_M, \rho_M, Dt_M) =$$
$$M_{R_M}(\varphi(P_{БП}, Q_{БП}, C_{БП}, L_{БП}, R_{БП}, \rho_{БП}, Dt_{БП}))$$

где $\varphi^{R_M}(P_M, Q_M, C_M, L_M, R_M, \rho_M, Dt_M)$ - кадровая структура, соответствующая БП, редуцированная по производительности в $R_M$ раз и имеющая $C_M$ каналов и $A_M$ ячеек аппаратного ресурса. В соответствии с введенными определениями будем считать, что редукция производительности и аппаратных затрат с коэффициентами 0 и 1, соответственно, тождественно равна исходной кадровой структуре $M_0(\varphi) \equiv M_1(\varphi) \equiv \varphi$.

Функциональная зависимость $M_{R_M}(\varphi)$ является нелинейной, поэтому сокращение аппаратных затрат при редукции производительности непропорционально коэффициенту редукции $R_M$. При разных коэффициентах редукции можно получить вычислительные структуры с одинаковыми аппаратными затратами, но разным интервалом I и разным временем обработки данных $T_M$.

Время обработки потока данных $T_{БП}$ для конвейерной вычислительной структуры пропорционально длине потока данных **N**, длительности такта вычислительной системы $\tau$ и интервалу обработки данных $I_{БП}$:

$$T_{БП} = L \cdot \tau + (N-1) \cdot I_{БП} \cdot \tau = (N + L - 1) \cdot \tau \qquad (П1.8)$$

При процедурной организации вычислений время $T_{ПР}$ линейно возрастает пропорционально длине потока данных **N**, длительности такта вычислительной системы $\tau$ и интервалу обработки данных $I_{ПР}$ (числу вычислительных операций **P)**:

$$T_{ПР} = N \cdot I_{ПР} \cdot \tau = N \cdot P \cdot \tau. \qquad (П1.9)$$

Для м-кадра, являющегося результатом редукции производительности и аппаратных затрат, время обработки потока данных $T_{М}$ пропорционально длине потока данных **N**, длительности такта вычислительной системы $\tau$ и интервалу $I_M$

$$T_M = N \cdot I_M \cdot \tau \quad . \qquad (П1.10)$$

Здесь $I_{М}$ - это число тактов $\tau$, необходимых для обработки одного данного в конвейере.

Нетрудно заметить, что при процедурной организации вычислений, время возрастает пропорционально числу выполняемых операций, а при $I = P$ (П1.10) переходит в (П1.9). Разницу во времени решения задачи при различных способах организации вычислений можно оценить с помощью формул:

$$\frac{T_{ПР}}{T_{БП}} = \frac{N \cdot I_{ПР} \cdot \tau}{(N-1+L) \cdot I_{БП} \cdot \tau} = \frac{N \cdot P \cdot \tau}{(N-1+L) \cdot 1 \cdot \tau} = \frac{N}{(N+L-1)} \cdot P \approx P \ , \qquad (П1.11)$$

$$\frac{T_M}{T_{БП}} = \frac{N \cdot I_M \cdot \tau}{(N+L-1) \cdot I_{БП} \cdot \tau} = \frac{N}{(N+L-1) \cdot 1} \cdot I_M \approx I_M \ , \qquad (П1.12)$$

$$\frac{T_{ПР}}{T_M} = \frac{N \cdot I_{ПР} \cdot \tau}{N \cdot I_M \cdot \tau} = \frac{N \cdot P \cdot \tau}{N \cdot I_M \cdot \tau} = \frac{P}{I_M} \qquad (П1.13)$$

Как правило, для потоковых задач **N** на несколько десятичных порядков превышает латентность **L**. Тогда будем полагать, что $N + L - 1 \approx N$.

Поскольку именно интервал определяет время обработки данных м-кадром, то при редукции производительности целесообразно выбирать такие структуры, которые обладают наименьшим возможным интервалом при равных или сходных аппаратных затратах. Рациональными м-кадрами следует считать те, у которых сокращение аппаратных затрат примерно соответствует

увеличению интервала обработки данных, а наиболее рациональным среди них является м-кадр с наименьшим интервалом.

### П1.3.1 Методы синтеза м-кадров

Одним из первых прообразов м-кадров стало описанное в работах [174, 176] решение задач с существенно-переменной интенсивностью потоков данных. Для таких задач значение коэффициента редукции производительности в каждом фрагменте информационного графа задачи подбиралось экспериментально с учетом баланса темпа обработки данных внутри и между фрагментами и занимаемого аппаратного ресурса, что требовало высокой квалификации программиста. Редукция производительности и аппаратных затрат, примененная для этой задачи, требовала длительного и кропотливого подбора для каждого вычислительного фрагмента числа вычислительных устройств и рационального значения коэффициента редукции, обеспечивающего баланс между необходимым сокращением аппаратных затрат и интервалом обработки данных для приемлемого времени решения задачи.

Из-за отсутствия точного аналитического метода выбора рационального коэффициента редукции, в работах [177 – 179] использовался один из самых простых способов выбора значения коэффициента редукции – метод дихотомии или бисекции (итерационная последовательная двукратная редукция производительности фрагмента). В этом случае значения коэффициента редукции представляют собой ряд по степени 2: $K_{\mathrm{M}} = 1,2,4,8,\ldots log_2 P$. Такой же метод выбора коэффициента редукции использовался в последующих работах [188, 191] для редукции производительности базовой операции БПФ, содержащей $P = 2^4 = 16$ операций. Если число операций БП не является степенью числа 2, рассматриваемый метод имеет ряд ограничений:

- необходимо использовать дополнение числа операций или ступеней обработки до $P^1 = 2^{\lceil log_2 P \rceil}$, что увеличивает интервал обработки данных и время решения;

- очередной м-кадр не может быть получен сразу, очередная структура может быть получена только после редукции предыдущей;

- метод является итерационным, что может существенно увеличить число шагов редукционных преобразований при согласованной редукции множества информационно-связанных фрагментов задачи.

Помимо возможности масштабирования минимальной кадровой структуры, ранее считавшейся неделимой, основным отличием разрабатываемого метода является синтез принципиально новой структуры – м-кадра. В отличии от ранее известных работ (главным образом, И.И. Левина), м-кадр является новой кадровой структурой, которая синтезируется на основе БП, а не выбирается как другой фрагмент ИГЗ в качестве базового подграфа. В известных методах структурно-процедурной организации вычислений [6, 75, 102] при преобразовании в кадровую форму выделяется кортеж изоморфных подграфов, для которых структурно реализуется один подграф (базовый), а все остальные реализуются процедурно путем коммутации входов и выходов вычислительной структуры, соответствующей аппаратно реализованному БП. Важной особенностью кадровой формы является изоморфизм подграфов, означающий регулярность не только операционных вершин, но и внутренних связей между ними в задаче (дуги ИГЗ между БП). В м-кадре связи на входах его операционных вершин меняются каждый такт, поэтому м-кадр нельзя назвать в полном смысле изоморфным базовому подграфу, а можно рассматривать как обобщенную форму, позволяющую получить результаты вычислений, эквивалентные исходной кадровой структуре.

Поскольку кадровая структура $\varphi$ неделима с точки зрения слоев и итераций, м-кадр $\mu$ нельзя выделить из нее как фрагмент вычислительной структуры, т.е. представить м-кадр $\mu$ в виде масштабирования кадровой структуры $\varphi$ по слоям и итерациям

$$\varphi \neq \Phi(F(\mu)). \qquad (П1.14)$$

Если м-кадр $\mu$ может быть выделен из состава исходной кадровой структуры $\varphi$ как вычислительная структура, которую можно описать с помощью масштабирования по слоям и итерациям, это свидетельствует о неправильном выборе БП и требует повторить процедуру выбора для поиска корректного БП.

Как уже отмечалось, м-кадр μ – это кадровая структура, содержащая непустое подмножество вершин мощности не ниже 2 и инцидентных им дуг, позволяющая при последовательном обходе выполнить все вычисления базового подграфа с соблюдением информационной эквивалентности результатов вычислений. Поэтому для реализации вычислений исходной кадровой структуры на меньшей вычислительной структуре м-кадра необходимо привести вычисления во всех ярусах базового подграфа к единой обобщенной форме, с помощью которой можно последовательно реализовать вычисления любого яруса ЯПФ базового подграфа по аналогии с преобразованием информационного графа задачи в кадры [102].

Согласно требованиям информационной эквивалентности результатов вычислений, для такой организации вычислений нужно синтезировать коммутационную структуру, которая синхронизирует операнды и своевременно подает их на вход операционных устройств м-кадра.

Тогда метод преобразования минимальной кадровой структуры, соответствующей базовому подграфу, в м-кадр будет состоять из следующих действий.

1) В соответствии со значением коэффициента редукции производительности вычислительной структуры $K_M$ определить аппаратные затраты, достаточные для реализации м-кадра.
2) На основании отношения аппаратных затрат, требуемых для реализации БП и достаточных для реализации м-кадра, определить число фрагментов, на которые следует разделить БП (число вызовов или тактов работы «универсальной конвейерной ступени»).

3) Выделить «универсальную конвейерную ступень», т.е. структуру с неизменными операционными вершинами и коммутируемыми информационными входами, позволяющую реализовать вычисления любого яруса БП согласно принципу информационной эквивалентности.
4) Отобразить вершины каждого яруса БП на полученную «универсальную конвейерную ступень» и определить число тактов работы м-кадра для каждого яруса.
5) Синтезировать информационные связи и функцию коммутации между входами и выходами операционных вершин «универсальной конвейерной ступени в каждом такте работы, преобразовав вектор информационных входов и выходов в кортеж последовательно подаваемых данных.
6) Синтезировать м-кадр в виде совокупности «универсальной конвейерной ступени» и правил организации (кортежирования) информационных входов.
7) Оценить аппаратные затраты на реализацию «универсальной конвейерной ступени» м-кадра и функций коммутации.
8) Оценить параметры м-кадра (время обработки $T_{\mathrm{M}}$, занимаемый ресурс $A_{\mathrm{M}}$и число каналов$C_M$).

Для создания м-кадра наиболее важными преобразованиями являются выделение «универсальной конвейерной ступени», синтез информационных связей и функции коммутации между ее входами и выходами в каждом такте работы.

### П1.3.2 Выделение вычислительных операций для м-кадра

Для определения рационального состава м-кадра представим кадровую структуру $\varphi = \varphi(\mathrm{P,Q,L,C,S,\rho,Dt})$ в виде совокупности мультимножества Oper его вычислительных операций и мультимножества всех его информационных связей Link, объединяющего мультимножества входных каналов памяти $C_{\mathrm{In}}$, внутренних связей между вершинами **Q** и выходных каналов $C_{\mathrm{Out}}$

$$\varphi = (\mathrm{Oper}, \mathrm{Link}) \qquad (П1.15)$$

$$\mathrm{Link} = (C_{In} \cup Q \cup C_{Out}) = \sum_{t=1}^{L}({C_{In}}^{t}, Q^{t}, {C_{Out}}^{t}) \qquad (П1.16)$$

Каждое из мультимножеств вычислительных операций Oper и информационных связей Link можно представить в виде перечисления своих элементов с указанием их кратности

$$\mathrm{Link} = \{k_{\mathrm{Ln1}} \cdot q_1, k_{\mathrm{Ln2}} \cdot q_2, \ldots\}, k_{\mathrm{Ln}j} \geq 1, q_j \in \varphi,$$

$$\mathrm{Oper} = \{k_{\mathrm{Op1}} \cdot o_1, k_{\mathrm{Op2}} \cdot o_2, \ldots\}, k_{\mathrm{Op}l} \geq 1, o_l \in \varphi \qquad (П1.17)$$

В представлении (П1.17) $k_{\mathrm{Ln}j}$- это кратность информационной связи $q_j$, принадлежащей БП $\varphi$, $k_{\mathrm{Op}l}$- это кратность вычислительной операции $o_l$ БП $\varphi$.

Каждое из мультимножеств Link и Oper можно представить в виде суммы (объединения) мультимножеств $\mathrm{Link}^t$ и $\mathrm{Oper}^t$, t=1…L, соответствующих связям и операционным вершинам яруса ярусно-параллельной формы (ЯПФ) БП $\varphi$:

$$\mathrm{Link} = \sum_{t=1}^{L} \mathrm{Link}^t \ , \mathrm{Oper} = \sum_{t=1}^{L} \mathrm{Oper}^t \ . \qquad (П1.18)$$

Представим БП $\varphi$ в виде кортежа ярусов, связанных между собой информационной зависимостью. Каждый ярус можно описать совокупностью мультимножеств вычислительных операций и информационных связей

$$\varphi = \overset{t=1}{\underset{L}{\times}} < \mathrm{Oper}^t, \mathrm{Link}^t, >= \overset{t=1}{\underset{L}{\times}} < Oper^t, C_{In}^t, Q^t, C_{Out}^t, >= \overset{t=1}{\underset{L}{\times}} Tier^t \qquad (П1.19)$$

В формуле (П1.19) скобками «<>» обозначен кортеж информационных связей и операций, а символ конвейерного выполнения «×» (по классификации Хендлера) описывает информационную зависимость между ярусами ЯПФ БП.

В общем случае каждый ярус $Tier^t$представляет собой мультимножество одновременно работающих информационно–независимых устройств:

$$Tier^t = \begin{cases} Op_1^t, \alpha_{1,1}, \alpha_{2,1}, \alpha_{3,1} \\ Op_2^t, \alpha_{1,2}, \alpha_{2,2}, \alpha_{3,2} \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ Op_{P^t}^t, \alpha_{1,P^t}, \alpha_{2,P^t}, \alpha_{3,P^t} \end{cases}, \qquad (П1.20)$$

где кортеж $\langle Op_i^t, \alpha_{1,i}, \alpha_{2,i}, \alpha_{3,i} \rangle$ описывает выполняемую устройством операцию $Op$ с информационными входами $\alpha_{1,i}, \alpha_{2,i}$ и выходом $\alpha_{3,i}$.

Операции $\mathrm{Oper}^t$ яруса $Tier^t$ БП – это мультимножество по типам выполняемых операций $Op$, поскольку каждый ярус может содержать одинаковые операции. При этом ярус $Tier^t$ - множество, так как полностью совпадающих кортежей из (П1.20) в БП быть не может. Тогда каждый ярус можно представить в виде объединения информационно-независимых операций:

$$Tier^t = \cup_{l=1}^{P^t} \left\langle Op_l^t, \alpha_{1,l}^{Q_L^t}, \alpha_{2,l}^{Q_L^t}, \alpha_{3,l}^t \right\rangle, \qquad (П1.21)$$

где $P^t$ - число операционных вершин в ярусе **t**, $Q_L^t = C_{in} \cup (\cup_{j=1}^{t-1} Q^j)$ - множество всех информационных связей БП до яруса **t**, включающее информационные входы $C_{In}$ и внутренние связи кадровой структуры из множества выходов операционных вершин **Q** всех предыдущих ярусов. По аналогии введем множество всех информационных связей БП, начиная с яруса **t**, как $Q_R^t = C_{Out} \cup (\cup_{j=t+1}^{L} Q^j)$. Тогда $Link = Q_L^t \cup Q^t \cup Q_R^t$

Информационные входы $\alpha_{1,i}, \alpha_{2,i}$ любой операции $Op_l^t$ яруса **t** являются либо информационными входами из множества $C_{In}$, либо внутренними связями БП из множества выходов операционных вершин **Q**, поэтому справедливы соотношения:

$\forall Op_l^t \in Oper^t \subset Oper$ и $\forall \alpha_{1,l}^{Q_L^t}, \alpha_{2,l}^{Q_L^t} \in Q_L^t \subset Link$, причем $\alpha_{1,l}$ может быть равен $\alpha_{2,l}$, а $\alpha_{3,l}^t \in Q_R^t$

Мультимножества вычислительных операций Operи информационных связей Link БП характеризуются:

- опорными множествами Supp (Oper) и Supp ( Link), содержащими единичные экземпляры всех элементов мультимножества;

- мощностью мультимножества, равной общему количеству его элементов: $\mathrm{card}\,(\mathrm{Oper}) = P$ и $\mathrm{card}\,(\mathrm{Link}) = \mathrm{card}(C_{\mathrm{In}}) + \mathrm{Card}(Q) + \mathrm{card}(C_{\mathrm{Out}})$, причем $\mathrm{card}(C_{\mathrm{In}}) + \mathrm{card}(C_{\mathrm{Out}}) = \mathrm{C}_{\text{БП}}$;

- размерностью мультимножества dim (Oper) и dim (Link), равной общему числу различных элементов. Фактически, это мощность опорных множеств card (Supp (Oper)) и card(Supp ( Link)), соответственно.

Благодаря мультимножествам в представлении БП, можно просто и наглядно получить практически важные количественные характеристики для его масштабирования при увеличении или сокращении доступного вычислительного ресурса.

Опорное множество вычислительных операций Supp (Oper) определяет минимальное число операций, которое должен содержать м-кадр – по одной различающейся вычислительной операции БП. М-кадр, построенный на основе опорного множества Supp (Oper), будем называть минимальным по составу вычислительных операций, т.к. дальнейшее сокращение числа вычислительных устройств приведет к нарушению принципа информационной эквивалентности результатов вычислений.

Для минимального м-кадра можно аналитически определить наибольшее возможное значение коэффициента редукции производительности по устройствам как отношение общего числа устройств кадровой структуры $P$ к числу устройств минимального м-кадра

$$R_{M_{min}}^{\left\lceil \frac{P}{\dim Oper_{БП}} \right\rceil \left\lceil \frac{\mathrm{card} Oper_{БП}}{\dim Oper_{БП}} \right\rceil}. \qquad (\text{П}1.22)$$

Поэтому коэффициент редукции производительности и аппаратных затрат по устройствам для кадровой структуры ограничен интервалом $R_{M_{min}}^{M}$. В ряде случаев разные математические операции можно объединить в одном

устройстве, обрабатывающем одинаковые типы данных. Пример таких операций - это операции сложения и вычитания, операции умножения и деления, ряд операций сравнения (равно и не равно, больше либо равно и меньше, меньше либо равно и больше).

При соблюдении требований информационной эквивалентности результатов для выбора вычислительных устройств м-кадра можно использовать не только минимизацию аппаратных затрат, характерную для минимального м-кадра, но и другие критерии: минимизацию времени работы (интервала обработки данных), сбалансированность реализации (рациональное соотношение между сокращением аппаратных затрат и интервалом обработки и др.). Выбранный критерий является основным для интервала обработки данных в синтезируемом м-кадре.

Минимальный м-кадр (или опорное множество вычислительных операций $\mathrm{Supp}\ (\mathrm{Oper})$) – это всегда возможный, но не всегда эффективный вариант, т.к. характеризуется большим интервалом обработки данных, величина которого по построению не меньше числа ярусов ЯПФ исходного БП.

Один из наиболее рациональных, но не всегда возможный, вариант выбора состава м-кадра – это представление мультимножества операций кадровой структуры в виде репродукции мультимножеств с помощью общих делителей кратностей всех элементов, что эквивалентно математической операции вынесения общего множителя за скобки. Это возможно в случае, если число разных операций БП кратно общему делителю (ОД) и БП можно представить в виде:

$$\varphi = \{\text{ОД}\} \cdot \mu^*, \text{ где } ОД \in N,\ 1 < \text{ОД} < P. \qquad (\text{П}1.23)$$

Значение ОД определяет интервал обработки данных и коэффициент сокращения аппаратных затрат при выборе операций $\mu^*$ в качестве м-кадра. В ряде случаев возможны несколько значений ОД, что позволяет синтезировать не один, а серию м-кадров с различными коэффициентами сокращения аппаратных затрат и интервалом обработки данных.

При соблюдении принципа информационной эквивалентности возможны различные методы и стратегии синтеза состава вычислительных операций м-кадра, среди которых можно выделить следующие основные способы:

1) М-кадр $\mu_1$ – это минимальный м-кадр, образованный опорным множеством Supp (Oper) вычислительных устройств, т.е. $\mu_1(Supp(Oper_\varphi))$.
2) М-кадр $\mu_2$ – это м-кадр на основе репродукции мультимножеств с помощью ОД, т.е. $\mu_2(ОД(Oper_\varphi))$ (если это возможно).
3) М-кадр $\mu_3$ образуется из яруса ЯПФ с наибольшим **суммарным** числом операций с дополнением до опорного множества Supp (Oper) ( при необходимости), т.е. $\mu_3(max(\, card\ Oper_\varphi^t))$.
4) М-кадр $\mu_4$ образуется из яруса ЯПФ с наибольшим числом **различных** операций с дополнением до опорного множества Supp (Oper) (при необходимости), т.е. $\mu_4(max(\, dim\ \ Oper_\varphi^t))$.
5) М-кадр $\mu_5$ образуется из минимального м-кадра $\mu_1$ и дополнительной операции с наибольшей кратностью в БП, т.е. $\mu_5(\mu_1(Supp(Oper_\varphi)) + dim(\, Oper_\varphi))$. В ряде случаев это позволяет сократить интервал обработки м-кадра примерно вдвое при незначительных дополнительных аппаратных затратах.

Представленный перечень методов синтеза м-кадров получен на основе анализа БП задач цифровой обработки сигналов, символьной обработки, линейной алгебры и молекулярного докинга, и не является исчерпывающим. Для задач других предметных областей его можно дополнить новыми, специфичными для этих областей, стратегиями синтеза м-подграфов. Однако число различных способов выбора рационального состава устройств м-кадра невелико (не более 10), в том числе из-за конечного числа предметных областей для РВС.

Для каждого из рассмотренных м-кадров $\mu_1 \ldots \mu_5$ можно оценить прямое сокращение аппаратных затрат как отношение числа вычислительных устройств исходного БП к числу вычислительных устройств м-кадра:

$$\frac{A_{БП}}{A_M} = \frac{P}{\operatorname{card} \mu}$$

.

При этом дополнительное коммутационное оборудование не учитывается. Тогда

$$\frac{A_{\text{БП}}}{A_{\mu_1}} = \frac{P}{\dim Oper_{\varphi}}, \frac{A_{\text{БП}}}{A_{\mu_2}} = \frac{P}{\text{ОД}} \cdot \frac{1}{\dim Oper_{\mu^*}}, \frac{A_{БП}}{A_{\mu_3}} = \frac{P}{\max(\operatorname{card} Oper_{\varphi}^{t})},$$

$$\frac{A_{\text{БП}}}{A_{\mu 4}} = \frac{P}{\max(\dim Oper_{\varphi}^{t})}, \frac{A_{БП}}{A_{\mu_5}} = \frac{P}{\dim Oper_{\varphi}+1}.$$

Оценка сокращения аппаратных затрат будет справедлива для линейной структуры БП, каждый ярус которого реализуется м-кадром за 1 такт. В остальных случаях необходимо учитывать дополнительные аппаратные затраты на коммутационное оборудование.

### П1.3.3 Пример синтеза м-кадров для цифровой обработки сигналов

Рассмотрим методы синтеза м-кадров на примере операции быстрого преобразования Фурье (БПФ) для комплексных чисел. Информационный граф структурной реализации БО БПФ представлен на рис. П1.8 и содержит 16 операционных вершин, обрабатывающих данные, поступающие независимо по 8 входным каналам.

Информационный граф БО БПФ содержит 4 сумматора (вершины 1, 2, 10, 16), 4 вычитателя (вершины 3, 4, 9, 15) и 8 умножителей (вершины 5, 6, 7, 8, 11, 12, 13, 14). Поэтому мультимножество операций БП БО БПФ можно разделить на 3 группы (сложение «+», вычитание «-» и умножение «*») и представить операции БП БО БПФ в виде мультимножества: $g_{БПФ} = \{4 \cdot (+), 4 \cdot (-), 8 \cdot (*)\}$.

В соответствии с предложенными методами, построим м-кадры $\mu_1 \ldots \mu_5$:

$\mu_1 = \{1 \cdot (+), 1 \cdot (-), 1 \cdot (*)\}$,

$\mu_2^1 = 2 \cdot \{2 \cdot (+), 2 \cdot (-), 4 \cdot (*)\}$, $\mu_2^2 = 4 \cdot \{1 \cdot (+), 1 \cdot (-), 2 \cdot (*)\}$

$\mu_3 = \{2 \cdot (+), 2 \cdot (-), 4 \cdot (*)\}$,

$\mu_4 = \{2 \cdot (+), 2 \cdot (-), 4 \cdot (*)\}$,

$\mu_5 = \{1 \cdot (+), 1 \cdot (-), 2 \cdot (*)\}.$

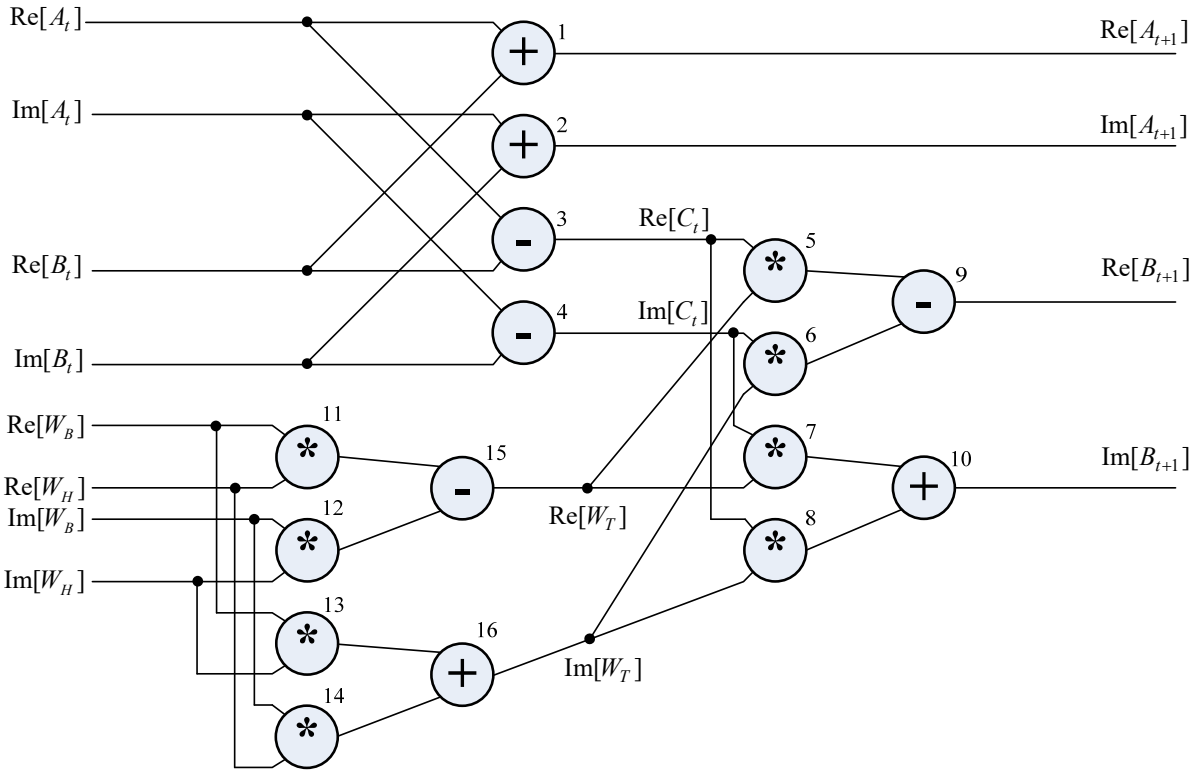



Рис. П1.8. Информационный граф структурной реализации БО БПФ

Очевидно, что в рассматриваемом примере некоторые м-кадры совпали:$\mu_2^2 = \mu_5$, $\mu_2^1 = \mu_3 = \mu_4$, что объясняется регулярной и достаточно удобной для редукции структурой БП, т.к. общее число операций и количество операций каждого типа является степенью 2. Более того, поскольку операции сложения и вычитания выполняются одним и тем же вычислительным устройством, их можно совместить и представить БП БО БПФ как $g_{БПФ} = \{8 \cdot (\pm), 8 \cdot (*)\}$, что сократит число различных групп операций и, соответственно, изменит состав м-кадров:

$\mu_1 = \{1 \cdot (\pm), 1 \cdot (*)\}$,

$\mu_2^1 = 2 \cdot \{4 \cdot (\pm), 4 \cdot (*)\}$, $\mu_2^2 = 4 \cdot \{2 \cdot (\pm), 2 \cdot (*)\}$, $\mu_2^3 = 8 \cdot \{1 \cdot (\pm), 1 \cdot (*)\}$

$\mu_3 = \{4 \cdot (\pm), 4 \cdot (*)\}$,

$\mu_4 = \{4 \cdot (\pm), 4 \cdot (*)\}$,

$\mu_5^1 = \{1 \cdot (\pm), 2 \cdot (*)\}$, $\mu_5^2 = \{2 \cdot (\pm), 1 \cdot (*)\}$

В этом случае тоже есть совпадение м-кадров: $\mu_2^3 = \mu_1$ и $\mu_2^1 = \mu_3 = \mu_4$, но из-за равной кратности операций появились два варианта м-кадра $\mu_5$. Вычислительные структуры, соответствующие построенным м-кадрам, приведены на рис. 3.2 в третьей главе. Согласно рис. 3.2, информационные связи между вершинами исходного БП БО БПФ в м-кадре реализуются с помощью мультиплексоров, которые являются дополнительным коммутационным оборудованием, и подают на вход устройства либо ранее вычисленный результат, либо входные данные. Примечательно, что коммутационные затраты для рассматриваемого примера остаются постоянными для всех м-кадров (см. рис.3.2), т.е. сокращение числа мультиплексоров кратно увеличению числа их входов, а число входов, в свою очередь, определяет интервал обработки входных данных для каждого м-кадра. Рассмотренный пример построения м-кадров для БП БО БПФ, не являясь формальным доказательством корректности предложенных методов, иллюстрирует суть предложенных преобразований.

### П1.3.4 Пример синтеза м-кадров для задачи молекулярного докинга

Рассмотрим применение методов синтеза м-кадров для подзадачи расчета декартовых координат торсионных вращений частей лиганда, информационный граф которой представлен на рис. П1.9.

Информационный граф подзадачи расчета декартовых координат торсионных вращений частей лиганда содержит 10 операций суммирования (вершины 10, 12, 29, 30, 34,35, 38, 41), 8-вычитания (вершины 1, 2, 3, 31, 32, 33, 36, 37), 23 – умножения (вершины 4-7, 17-28, 42-47), 3 – деления (вершины 14-16), 1 – извлечение квадратного корня (вершина 13), 1 – косинус (вершина 9) и 1 – синус (вершина 8), всего – 47 операций, размещенных в 10 ярусах ЯПФ.

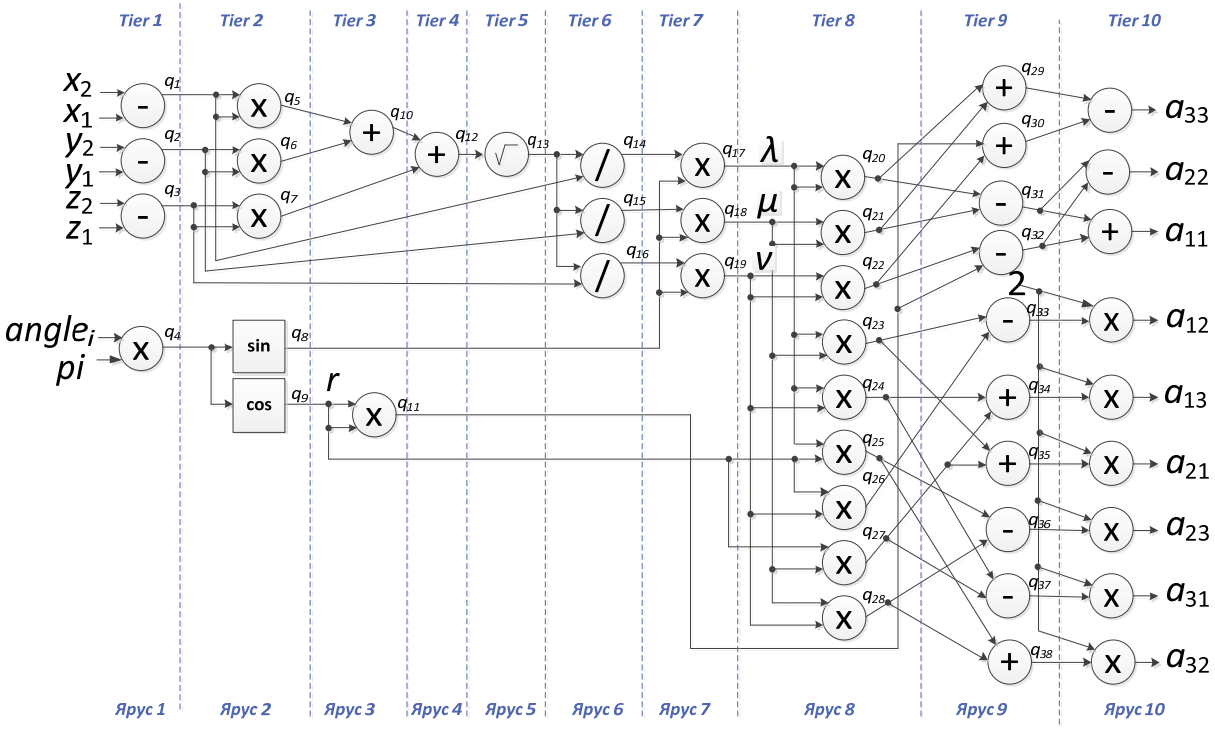



Рис. П1.9. Кадровая структура, соответствующая базовому подграфу фрагмента задачи молекулярного докинга

Мультимножество операций кадровой структуры (рис. П1.9) имеет вид:

$$\mathrm{Oper}_{Rot} = \{3\cdot(\pm),1\cdot(*),0\cdot(/),0\cdot(\sqrt{\ }),0\cdot(sin),0\cdot(cos)\}\times$$
$$\{0\cdot(\pm),3\cdot(*),0\cdot(/),0\cdot(\sqrt{\ }),1\cdot(sin),1\cdot(cos)\}\times$$
$$\{1\cdot(\pm),1\cdot(*),0\cdot(/),0\cdot(\sqrt{\ }),0\cdot(sin),0\cdot(cos)\}\times$$
$$\{1\cdot(\pm),0\cdot(*),0\cdot(/),0\cdot(\sqrt{\ }),0\cdot(sin),0\cdot(cos)\}\times$$
$$\{0\cdot(\pm),0\cdot(*),0\cdot(/),1\cdot(\sqrt{\ }),0\cdot(sin),0\cdot(cos)\}\times$$
$$\{0\cdot(\pm),0\cdot(*),3\cdot(/),0\cdot(\sqrt{\ }),0\cdot(sin),0\cdot(cos)\}\times$$
$$\{0\cdot(\pm),3\cdot(*),0\cdot(/),0\cdot(\sqrt{\ }),0\cdot(sin),0\cdot(cos)\}\times$$
$$\{0\cdot(\pm),9\cdot(*),0\cdot(/),0\cdot(\sqrt{\ }),0\cdot(sin),0\cdot(cos)\}\times$$
$$\{10\cdot(\pm),0\cdot(*),0\cdot(/),0\cdot(\sqrt{\ }),0\cdot(sin),0\cdot(cos)\}\times$$
$$\{3\cdot(\pm),6\cdot(*),0\cdot(/),0\cdot(\sqrt{\ }),0\cdot(sin),0\cdot(cos)\}$$

Мультимножество операций можно разделить на 7 групп: $g_{tors} = \{10 \cdot (+), 8 \cdot (-), 23 \cdot (*), 3 \cdot (/), 1 \cdot (\sqrt{\ }), 1 \cdot (cos), 1 \cdot (sin)\}$. С учетом совмещения сложения и вычитания в одном устройстве, можно сократить число групп и представить мультимножество операций как $g_{tors} = \{18 \cdot (\pm), 23 \cdot (*), 3 \cdot (/), 1 \cdot (\sqrt{\ }), 1 \cdot (cos), 1 \cdot (sin)\}$.

Нетрудно заметить, что в обоих случаях кратности операций являются взаимно простыми числами, поэтому построить подграф $\mu_2$ невозможно.

В соответствии с предложенными методами, построим м-кадры $\mu_1 \ldots \mu_5$:

$\mu_1 = \{1 \cdot (\pm), 1 \cdot (*), 1 \cdot (/), 1 \cdot (\sqrt{\ }), 1 \cdot (cos), 1 \cdot (sin)\}$,

$\mu_2 = \emptyset$,

$\mu_3 = \{10 \cdot (\pm), 1 \cdot (*), 1 \cdot (/), 1 \cdot (\sqrt{\ }), 1 \cdot (cos), 1 \cdot (sin)\}$,

$\mu_4 = \{1 \cdot (\pm), 3 \cdot (*), 1 \cdot (/), 1 \cdot (\sqrt{\ }), 1 \cdot (cos), 1 \cdot (sin)\}$,

$$\mu_5 = \left\{1 \cdot (\pm), 2 \cdot (*), 1 \cdot (/), 1 \cdot (\sqrt{\ }), 1 \cdot (cos), 1 \cdot (sin)\right\}$$

Оценка сокращения аппаратных затрат для построенных м-кадров имеет вид:

$\frac{A_{БП}}{A_{\mu_1}} = \frac{47}{7} = 6{,}7$, $\frac{A_{БП}}{A_{\mu_2}} = \emptyset$, $\frac{A_{БП}}{A_{\mu_3}} = \frac{47}{15} = 3{,}1$,

$\frac{A_{БП}}{A_{\mu 4}} = \frac{47}{8} = 5{,}9$, $\frac{A_{БП}}{A_{\mu_5}} = \frac{47}{7} = 6{,}7$.

Далее необходимо отобразить вершины каждого яруса базового подграфа на построенную «универсальную конвейерную ступень» каждого м-кадра и определить его интервал обработки данных, а затем выбрать наиболее рациональный вариант по критериям сокращения аппаратных затрат и интервала обработки данных.

При отображении вершин каждого яруса на устройства «универсальной конвейерной ступени» информационные входы и выходы исходного БП упорядочиваются и формируются кортежи входных и выходных данных. Упорядочивание входных данных реализуется либо с помощью мультиплексоров, как в предыдущем примере, либо в канале, что сокращает

число каналов для реализации м-кадра, но требует дополнительных аппаратных расходов на коммутацию и синхронизацию операндов, особенно при разрыве потока данных из-за особенности работы КРП.

Объединение операций в одном устройстве является элементом перехода от структурной организации вычислений к процедурной с сокращением занимаемого аппаратного ресурса. При этом если в одном слое встречаются обе объединяемые операции, то время реализации слоя на объединенном устройстве увеличивается пропорционально их количеству. Целесообразность объединения операций в одном устройстве зависит от кадровой структуры решаемой задачи, а ее эффективность определяется соотношением $K_{\cup} = \frac{\sum_{k=1}^{M_k} R(Op_k)}{R(Op_{1-M_k})}$. В качестве количественной оценки применимости этого приема можно использовать значение $K_{\cup} \geq 1{,}5$ для аппаратного ресурса при условии приемлемого роста интервала обработки данных м-подграфа.

## П1.4 Редукционные преобразования по разрядности

Редукция по разрядности $R^{\rho}$ - это известный способ сокращения аппаратных затрат, широко использовавшийся в вычислительной технике в 70-90 г.г. Редукция по разрядности $R^{\rho}$ сокращает число одновременно обрабатываемых разрядов за счет преобразования параллельной обработки разрядов каждого операнда в параллельно-последовательную (в предельном случае – в последовательную) обработку, что сокращает аппаратные затраты на реализацию вычислительного устройства. Редукция по разрядности $R^{\rho}$ для каждого информационного входа (канала) редуцируемой кадровой структуры формирует кортежи элементов разрядности $\frac{\rho}{R^{\rho}}$, конкатенация которых образует входные данные. Операнды каждого информационного входа преобразуются в кортеж из $r$ элементов разрядностью $\frac{\rho}{r}$каждый

$$< \alpha_1(1{:}\frac{\rho}{r}), \alpha_2(\frac{\rho}{r}+1{:}\frac{2\cdot\rho}{r}), \ldots, a_r(\frac{r-1}{r}\cdot\rho+1{:}\rho) >$$

$$< \beta_1(1{:}\frac{\rho}{r}), \beta_2(\frac{\rho}{r}+1{:}\frac{2\cdot\rho}{r}), \ldots, \beta_r(\frac{r-1}{r}\cdot\rho+1{:}\rho) >.$$

Каждый элемент кортежа обрабатывается на устройстве, реализующем $\frac{\rho}{r}$-разрядную операцию $Op_i(r)$, аппаратные затраты на реализацию которого меньше аппаратных затрат исходной $\rho$ -разрядной операции $Op_i$. Объединение (конкатенация) элементов входного кортежа образует входные данные $< \alpha_1(1{:}\frac{\rho}{r}), \beta_1(1{:}\frac{\rho}{r}) >, < \alpha_2(\frac{\rho}{r}+1{:}\frac{2\cdot\rho}{r}), \beta_2(\frac{\rho}{r}+1{:}\frac{2\cdot\rho}{r}) >, \ldots, < a_r(\frac{r-1}{r}\cdot\rho+1{:}\rho), \beta_r(\frac{r-1}{r}\cdot\rho+1{:}\rho) >$, а конкатенация элементов выходного кортежа образует выходные данные $\gamma_1(1{:}\frac{\rho}{r})||\gamma_2(\frac{\rho}{r}+1{:}\frac{2\cdot\rho}{r})||\ldots||\gamma_r(\frac{r-1}{r}\cdot\rho+1{:}\rho) = \gamma(1{:}\rho)$.

Интервал обработки данных в редуцированной по разрядности кадровой структуре увеличивается кратно значению коэффициента редукции $R^\rho$, а величина сокращения аппаратных затрат является нелинейной функцией, зависящей как от формата представления данных (логический, целочисленный, фиксированная точка, плавающая точка), так и от выполняемой операции. Для редукции по разрядности максимальное значение коэффициента редукции ограничено только длиной разрядной сетки входных данных (при обработке целочисленных операндов) вплоть до обработки однобитовых операндов при переходе от параллельного двоичного кода в последовательный. В отличие от редукции по числу вычислительных устройств, реализация редукции по разрядности $R^\rho$ не требует дополнительных аппаратных затрат на коммутацию (мультиплексоров) и синхронизацию (линий задержки) операндов и позволяет сократить число входных информационных каналов за счет их уплотнения при сокращении разрядности каждого из каналов. Увеличение числа внешних логических каналов – это важное свойство редукции по разрядности $R^\rho$, особенно с учетом рассмотренного в первой главе существенного недостатка числа внешних каналов ПЛИС.

Для логических и целочисленных данных с фиксированной точкой редукция по разрядности $R^\rho$ может применяться для широкого диапазона

коэффициентов редукции, ограниченного разрядной сеткой. Для данных с плавающей точкой эффективными значениями коэффициента редукции будут 2 для 32-разрядных данных и 2,4 для 64-разрядных из-за раздельной обработки разрядов мантиссы и порядка. Для стандартных арифметических операций целесообразно использовать коэффициент редукции, кратный $2^q$. При использовании коэффициентов, некратных степени 2, следует использовать вычислительные блоки с одноразрядной обработкой, что может увеличить аппаратные затраты на коммутацию. Комплексный анализ метода редукции производительности и аппаратных затрат кадровой структуры представлен в работах [174, 176], а применение при решении практических прикладных задач описано в [166].

Для операции суммирования с фиксированной точкой сокращение аппаратных затрат [174] близко к линейной зависимости от коэффициента редукции, для операции умножения с фиксированной точкой – к квадратичной, а для операций с плавающей точкой – существенно нелинейной и непропорциональной коэффициенту редукции. Эффективный шаг редукции по разрядности определяется с помощью оценок сокращения аппаратного ресурса:

- для 32-разрядного сумматора с фиксированной точкой общий аппаратный ресурс сокращается нелинейно от 1,5 до 1,8 раз [177-179] при редуцировании разрядности в 2 раза и зависит от используемого кристалла ПЛИС;
- для 32-разрядного умножителя с фиксированной точкой общий аппаратный ресурс сокращается нелинейно от 3 до 3,5 раз при редуцировании разрядности в 2 раза и зависит от используемого кристалла ПЛИС;
- для 64-разрядного умножителя с плавающей точкой общий аппаратный ресурс сокращается нелинейно от 2 до 2,5 раз при редуцировании разрядности в 2 раза и зависит от используемого кристалла ПЛИС.

Для операций с плавающей точкой применимость редукции по разрядности снижается из-за необходимости согласованно сокращать

разрядность представления целой и дробной частей, поэтому сокращение разрядности в 2 и 4 раза целесообразно для 64-разрядных операндов с плавающей точкой. Для 32-разрядных операндов оправданно сокращение разрядности не более, чем в 2 раза. Во всех остальных случаях сокращение разрядности приводит только к росту аппаратных затрат при одновременном увеличении интервала, что делает редукцию разрядности неэффективной для данных с плавающей точкой.

Поэтому для данных с фиксированной точкой и логических данных эффективный шаг определяется значением коэффициента редукции и наличием библиотечных элементов с сокращенной разрядностью. Для данных с плавающей точкой шаг для 32-разрядных операндов равен 2, для 64-разрядных – 2 или 4. Для данных с плавающей точкой увеличение значения шага выше рекомендованных значений оправданно только при существенном недостатке каналов или аппаратного ресурса, когда разработчик вынужден перейти к последовательной поразрядной обработке, что при постоянном росте числа вентилей в кристаллах ПЛИС встречается очень редко.

Критерии применимости редукции по разрядности $R^{\rho}$ :

- значение коэффициента редукции от 2 до 8 для кадровых структур, обрабатывающих логические, целочисленные или данные с фиксированной точкой;

- значение коэффициента редукции производительности по разрядности не превышает 8 для 64-разрядных данных, 4 – для 32-разрядных данных и 2 – для 16-разрядных данных;

- линейное сокращение аппаратных затрат и линейный рост интервала и общего времени обработки без дополнительных накладных расходов на коммутацию и синхронизацию операндов.

## П1.5 Доказательства теорем о применении редукционных преобразований

### П1.5.1 Теорема о последовательном применении редукционных преобразований

*Последовательно примененные редукции $R_m^T$ и $R_n^T$ одного типа $T$ с натуральными коэффициентами $m > 1$ и $n > 1$ эквивалентны редукции $R_{n \cdot m}^T$ того же типа с коэффициентом, равным их произведению $(m \cdot n) > 1$:*

$$R_n^T \circ R_m^T = R_{n \cdot m}^T. \qquad (П1.24)$$

**Доказательство**. Будем рассматривать фрагмент $F$ задачи, содержащий $N_F$ базовых подграфов, с числом устройств $Op_F$ каждый, обрабатывающих данные разрядности $\rho_F$. Общий объем вычислений во фрагменте $F$ равен:

$$NC_F = N_F \cdot Op_F \cdot \rho_F. \qquad (П1.25)$$

Поскольку редукционные преобразования независимы, условие (П1.24) теоремы 1 необходимо доказать для каждого вида редукции (по числу базовых подграфов, по числу вычислительных устройств, по разрядности).

Докажем условие (П1.24) для редукции по числу базовых подграфов $R_n^N$ с коэффициентом $n$. Число базовых подграфов фрагмента $F$ задачи сократится до величины $\left\lfloor \frac{N_F}{n} \right\rfloor_1$, а общий объем вычислений $NC_n^N$ будет равен

$$NC_n^N = \left\lfloor \frac{N_F}{n} \right\rfloor_1 \cdot Op_F \cdot \rho_F. \qquad (П1.26)$$

Последовательная редукция этого же фрагмента $R_m^N$ сократит число базовых подграфов в $m$ раз, и формула (П1.26) примет вид:

$$NC_{n \times m}^{N \times N} = \left\lfloor \frac{\left\lfloor \frac{N_F}{n} \right\rfloor_1}{m} \right\rfloor_1 \cdot Op_F \cdot \rho_F = \left\lfloor \frac{N_F}{n \cdot m} \right\rfloor_1 \cdot Op_F \cdot \rho_F. \qquad (П1.27)$$

Общее число вычислений при последовательных редукциях по числу базовых подграфов с коэффициентами $n$ и $m$ из выражения (П1.27) в точности соответствует результату редукции $R_{n \cdot m}^N$ фрагмента $F$ задачи по числу базовых подграфов при подстановке в формулу (П1.26) коэффициента $n \cdot m$ вместо $n$, что доказывает теорему П1.5.1 для редукции фрагмента задачи по числу

базовых подграфов. Аналогичным образом доказывается условие (П1.24) для редукции $R_n^{Op}$ по числу устройств $Op$ и редукции $R_n^{\rho}$ по разрядности обрабатываемых данных $\rho$, что доказывает теорему П1.5.1в целом.

**П1.5.2 Теорема о неаддитивности последовательных редукционных преобразований**

*Для редуцированного с коэффициентом* $\boldsymbol{n}$ *базового подграфа с помощью последовательного применения дополнительной редукции любого типа* $\boldsymbol{T}$ *с любым натуральным коэффициентом* $k > 1$ *в общем случае невозможно получить вычислительную структуру с коэффициентом редукции* $(n + x)$ *для любого наперед заданного* $x$, $\forall x \geq 1$

$$R_n^{T1} \circ R_k^{T2} \neq R_{n+x}^{T}. \qquad (П1.28)$$

**Доказательство.** В соответствии с теоремой П1.5.1 для редукций одного типа Т условие (П1.28) будет выполняться только в случае равенства коэффициентов редукции

$$n \cdot k = n + x. \qquad (П1.29)$$

Тогда

$$k = 1 + \frac{x}{n}. \qquad (П1.30)$$

Поскольку по условиям теоремы числа $n, k, x$ - натуральные, уравнение (П1.30) имеет решения для $\boldsymbol{k}$ в области натуральных чисел только для случая, когда $\boldsymbol{x}$ нацело делится на $\boldsymbol{n}$, а не для $\forall x \geq 1$, что доказывает теорему для редукции одного типа Т.

Для редукции разных типов у одинаковых вычислительных структур из левой и правой частей (П1.28) должно быть одинаковое общее число вычислительных операций

$$NC_{n\times k}^{T1\times T2} = NC_{n+x}^{T}, \qquad (П1.31)$$

поэтому

$$\left\lfloor \frac{NC}{n\cdot k} \right\rfloor_1 = \left\lfloor \frac{NC}{n+x} \right\rfloor_1. \qquad (П1.32)$$

Для выполнения условия (П1.32) необходимо, чтобы

$$n \cdot k \geq n + x \text{ и } n + x \geq n \cdot k. \qquad (П1.33)$$

Одновременное выполнение этих условий возможно только в случае

$$n \cdot k = n + x. \quad (П1.34)$$

Тогда

$$k = 1 + \frac{x}{n}. \quad (П1.35)$$

Поскольку по условиям теоремы числа $n, k, x$ - натуральные, уравнение (П1.35) имеет решения для $k$ в области натуральных чисел только для случая, когда $x$ нацело делится на $n$, а не для $\forall x \geq 1$, что приводит к противоречию и, тем самым, доказывает теорему.

**П1.5.3 Теорема о коммутативности редукционных преобразований**

*Суперпозиция редукций различных типов (T1 с коэффициентом* ***n*** *и T2 с коэффициентом* ***m****) коммутативна – изменение порядка выполнения редукций различных типов T1 и T2 в последовательных редукционных преобразованиях фрагмента задачи не изменяет результирующий информационный граф м-подграфа.*

$$R_n^{T1} \circ R_m^{T2} = R_m^{T2} \circ R_n^{T1},$$

где $T_1, T_2$ - типы редукционных преобразований,

$n, m$ - коэффициенты редукции.

**Доказательство.** Докажем коммутативность последовательной редукции по числу базовых подграфов и по числу устройств

$$R_n^N \circ R_m^{Op} = R_m^{Op} \circ R_n^N. \quad (П1.36)$$

После редукции $R_n^N$ фрагмента задачи по числу базовых подграфов с коэффициентом $n$ общее число вычислений над двоичными разрядами $NC_n^N$ составит

$$NC_n^N = \left\lfloor \frac{N_F}{n} \right\rfloor_1 \cdot Op_F \cdot \rho_F. \quad (П1.37)$$

Поскольку число базовых подграфов и число вычислительных устройств в базовом подграфе – независимые величины, последовательная редукция $R_m^{Op}$ фрагмента задачи по числу вычислительных устройств с коэффициентом $m$ сократит только число устройств, и общее число вычислений над двоичными разрядами составит

$$NC_{n\times m}^{N\times Op} = \left\lfloor \frac{N_F}{n} \right\rfloor_1 \cdot \left\lfloor \frac{Op_F}{m} \right\rfloor_1 \cdot \rho_F. \quad (П1.38)$$

Для правой части равенства (П1.36) последовательные редукции $R_m^{Op}$ и $R_n^N$ этого фрагмента задачи приведут к такому же общему числу вычислений над двоичными разрядами

$$NC_{m\times n}^{Op\times N} = \left\lfloor \frac{N_F}{n} \right\rfloor_1 \cdot \left\lfloor \frac{Op_F}{m} \right\rfloor_1 \cdot \rho_F. \quad (П1.39)$$

Аналогичным образом доказывается коммутативность для всех остальных возможных сочетаний последовательных редукций разных типов, что доказывает теорему П1.5.3.

**Приложение 2**

## Акты внедрения результатов работы

УТВЕРЖДАЮ
Директор ФГУП
«Научно-исследовательский институт
«Квант», д.т.н.

_____ Г.С. Елизаров

“20” декабря 2022 г.

**М.П.**

**АКТ**

о внедрении результатов докторской диссертации
Дордопуло Алексея Игоревича
«Теоретические основы технологии ресурсонезависимого программирования гибридных вычислительных систем»

Комиссия Федерального государственного унитарного предприятия «Научно-исследовательский институт «Квант» (ФГУП «НИИ «Квант») в составе: председателя, заместителя директора Титова А.Г., и членов комиссии: начальника отдела 111 Преснова А.В. и главного специалиста НИО-1 Патрикеева А.В. составила настоящий акт о том, что в ФГУП «НИИ «Квант» в рамках работ по государственным контрактам №390311/1, 390331/1, 390341/1, 330210/1 получили внедрение научные и практические результаты докторской диссертации Дордопуло А.И.

При выполнении работ по составной части (СЧ) опытно-конструкторской работы (ОКР) «Б-7.СВК09/1» разработан комплекс средств разработки прикладного программного обеспечения (СРППО) на языке высокого уровня COLAMO. В составе комплекса получил внедрение метод преобразования кадровой структуры и сокращения ее аппаратных затрат с помощью редукции производительности. Под кадровой структурой понимается результат преобразования информационного графа задачи с учетом информационных зависимостей между его фрагментами и основных параметров вычислительных устройств. Метод используется при портации

задачи к доступному аппаратному ресурсу вычислительной системы. Для задач символьной обработки с информационно-независимыми подзадачами метод позволил сократить время портации для реконфигурируемых вычислительных систем РВС-1К и ВБ РСК-1М в 3 - 6 раз.

При выполнении работ по СЧ ОКР «Б-7.СВК10/3», «Б-7.СВК.10/3.2», «Б-8.СВК.1/1» проведена модернизация СРППО, разработан программный комплекс программирования ПЛИС реконфигурируемого суперкомпьютера. Разработаны две модельные задачи Заказчика на языке высокого уровня COLAMO для вычислительных модулей RSP529 на базе ПЛИС XC6VLX550T и реконфигурируемого вычислителя «Орион V6», разработаны необходимые для реализации модельных задач Заказчика библиотечные элементы, реализованы на ПЛИС актуальные сложные много итерационные модельные задачи, использующие типовые специальные алгоритмы: шесть модельных задач Заказчика для реконфигурируемых блоков (РВБ) РВБ-410К7 на базе ПЛИС XC7K410T.

В программных средствах модернизированного СРППО и программном комплексе программирования ПЛИС реконфигурируемого суперкомпьютера получил внедрение метод преобразования задач на основе согласованной редукции производительности различных по вычислительной трудоемкости кадровых структур с синтезом сбалансированного по интервалу обработки данных решения. Метод ориентирован на преобразование задач, содержащих связанные информационной зависимостью подзадачи с разной степенью параллелизма. Применение указанного метода при реализации модельных задач Заказчика сократило время портации задач символьной обработки с информационно-зависимыми подзадачами с разной степенью параллелизма:

– для вычислительного блока РБ-Х2 на основе модуля RSP529 в 3,2 - 3,8 раза;

– для реконфигурируемого вычислителя «Орион V6» в 6,1 - 6,3 раза;

– для вычислительного блока РВБ-410К7 в 4,6 - 5,1 раза.

Внедрение научных и практических результатов докторской диссертации Дордопуло А.И. обеспечивает сокращение времени портации задач символьной обработки с разными типами информационной зависимости на различные архитектуры вычислительных систем при уровне реальной производительности не ниже 60% от пиковой.

ПРЕДСЕДАТЕЛЬ КОМИССИИ

Заместитель директора _____ А.Г. Титов

ЧЛЕНЫ КОМИССИИ

Начальник отдела 111 _____ А.В. Преснов

Главный специалист НИО-1 _____ А.В. Патрикеев

МИНОБРНАУКИ РОССИИ
федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого» (ФГАОУ ВО «СПбПУ»)

ИНН 7804040077, ОГРН 1027802505279, ОКПО 02068574
Политехническая ул., 29, Санкт-Петербург, 195251
тел.: +7(812)297 2095, факс: +7(812)552 6080
office@spbstu.ru

_____ № _____
на № _____ от _____

УТВЕРЖДАЮ
Проректор по научно-организационной деятельности

_____ Ю.С. Клочков

"27" 12 2022 г.

# АКТ

о внедрении результатов докторской диссертации
Дордопуло Алексея Игоревича
«Теоретические основы технологии ресурсонезависимого программирования гибридных вычислительных систем»

Комиссия Санкт-Петербургского политехнического университета Петра Великого (СПбПУ) в составе председателя директора Высшей школы искусственного интеллекта ИКНТ (ВШИИ ИКНТ), к.т.н. Владимира Александровича Мулюхи, и членов комиссии: профессора ВШИИ ИКНТ, д.т.н. Владимира Сергеевича Заборовского и начальника управления СКЦ «Политехнический», к.т.н. Алексея Андреевича Лукашина составила настоящий акт о том, что в СПбПУ в рамках работ по договору №411/22-Д от 17.11.2022 г. при разработке, тестировании и эксплуатации аппаратно-программного комплекса, предназначенного для разработки и отладки прикладного программного обеспечения высокопроизводительного вычислительного кластера, внедрены и используются следующие научно-теоретические и практические результаты докторской диссертации Дордопуло А.И.:

014927

1. Методика синтеза рационального решения при портации ресурсонезависимого представления задачи к архитектуре и конфигурации вычислительной системы.

2. Алгоритмы функционирования инструментальных программных средств технологии ресурсонезависимого программирования: алгоритм преобразования параметризованной ресурсом кадровой структуры и алгоритм редукции производительности выделенного фрагмента прикладной задачи.

3. Программа портации кадровой структуры на гибридные вычислительные системы различных архитектур и конфигураций.

Внедрение указанных результатов сократило время портации в 3-5 раз при изменении конфигурации вычислительной системы для задачи решения системы линейных алгебраических уравнений методом Якоби, тестовых задач линейной алгебры и цифровой обработки сигналов для различных конфигураций реконфигурируемых и гибридных вычислительных систем на основе реконфигурируемого вычислительного блока ТАСН.466535.160.

ПРЕДСЕДАТЕЛЬ КОМИССИИ
Директор ВШИИ ИКНТ _____ В.А. Мулюха

ЧЛЕНЫ КОМИССИИ
профессор ВШИИ ИКНТ _____ В.С. Заборовский

начальник управления
СКЦ «Политехнический» _____ А.А. Лукашин

УТВЕРЖДАЮ
Заместитель директора ЮНЦ РАН по науке

_____В.В. Стахеев

«23» 12 2022 г.

М.П.

**АКТ ВНЕДРЕНИЯ**

результатов докторской диссертационной работы

Дордопуло Алексея Игоревича

на тему «Теоретические основы технологии ресурсонезависимого программирования гибридных вычислительных систем»

в федеральном государственном бюджетном учреждении науки «Федеральный исследовательский центр Южный научный центр Российской Академии наук» (ЮНЦ РАН)

Комиссия в составе:

Председатель – заместитель директора ЮНЦ РАН по научной работе, заведующий отделом физики, химии, информатики, д.т.н. Юрасов Ю.И.

Члены комиссии – Зав. лабораторией информационных технологий и процессов управления, д.т.н. Мельник Э.В., с.н.с. лаборатории информационных технологий и процессов управления, к.т.н. Орда-Жигулиной М.В.

Настоящим акт подтверждает, что в ЮНЦ РАН внедрены и используются научно-теоретические и практические результаты докторской диссертации Дордопуло А.И.:

При выполнении научно-исследовательских работ (НИР) по темам 00-07-09 «Принципы и методы программирования реконфигурируемых многопроцессорных вычислительных систем», 00-10-01 «Методы и средства

оптимизации параллельно-конвейерных программ для реконфигурируемых вычислительных систем», 00-14-08 «Методы и средства оптимизации информационных обменов в высокопроизводительных многопроцессорных вычислительных системах» внедрены следующие результаты:

- информационная модель параллельных вычислений в виде кадровой структуры на основе информационного графа задачи;

- параметризуемое вычислительным ресурсом представление кадровой структуры параллельно-конвейерной реализации задачи.

При выполнении НИР по темам 00-07-60 «Исследование возможности эффективной реализации вычислительно трудоемких фрагментов задач на реконфигурируемых вычислителях на основе ПЛИС», 00-09-13 «Разработка и исследование средств многоуровневого программирования реконфигурируемых вычислительных систем на основе ПЛИС для решения вычислительно трудоемких задач», 00-12-23 «Разработка и исследование методов и средств верификации и оптимизации параллельно-конвейерных программ для реконфигурируемых вычислительных систем на основе ПЛИС внедрены следующие результаты:

- принципы преобразования параметризованной аппаратным ресурсом кадровой структуры к архитектуре и конфигурации вычислительной системы с помощью редукции производительности;

При выполнении НИР по теме 00-14-37 «Исследование и разработка новых архитектур высокопроизводительных реконфигурируемых вычислительных систем на основе ПЛИС для многоканальной обработки цифровых сигналов в реальном масштабе времени» в рамках Программы фундаментальных исследований Президиума РАН № 1 «Фундаментальные основы технологий двойного назначения в интересах национальной безопасности» внедрены следующие результаты:

- метод преобразования кадровой структуры и сокращения аппаратных затрат с помощью редукции производительности при портации задачи к доступному аппаратному ресурсу вычислительной системы.

Внедрение указанных результатов позволило в 3-5,5 раз сократить время портации задач линейной алгебры и цифровой обработки сигналов с заданным уровнем реальной производительности при корректировке параллельно-конвейерных программ в связи с изменением конфигурации вычислительной системы.

ПРЕДСЕДАТЕЛЬ КОМИССИИ

Заместитель директора ЮНЦ РАН по научной работе,
заведующий отделом физики,
химии, информатики,
д.т.н. Ю.И. Юрасов

ЧЛЕНЫ КОМИССИИ

Заведующий лабораторией
информационных технологий и
процессов управления ЮНЦ РАН,
доктор технических наук Э.В. Мельник

Старший научный сотрудник лаборатории
информационных технологий и
процессов управления ЮНЦ РАН,
кандидат технических наук М.В. Орда-Жигулина

УТВЕРЖДАЮ
Технический директор
ООО «НИЦ СЭ и НК»,
к.т.н.

_____ Ю.И. Доронченко

"12" _____12_____ 2022 г.

**АКТ**

о внедрении результатов докторской диссертации
«Теоретические основы технологии ресурсонезависимого программирования гибридных вычислительных систем» начальника отдела математического и алгоритмического обеспечения общества с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров»
Дордопуло Алексея Игоревича

Комиссия Общества с ограниченной ответственностью «НИЦ супер-ЭВМ и нейрокомпьютеров» (ООО «НИЦ СЭ и НК») в составе председателя, заместителя директора по разработкам, к.т.н., с.н.с. О.В. Катаева, начальника отдела прикладного программного обеспечения, к.т.н. Д.А. Сорокина, начальника отдела системного программного обеспечения, к.т.н. И.Г. Данилова составила настоящий акт о том, что в разработках ООО «НИЦ СЭ и НК» внедрены следующие научные и практические результаты диссертации «Теоретические основы ресурсонезависимого программирования гибридных вычислительных систем» Дордопуло Алексея Игоревича:

1. Параметризуемое вычислительным ресурсом представление кадровой структуры параллельно-конвейерной реализации задачи.
2. Метод преобразования кадровой структуры и сокращения аппаратных затрат с помощью редукции производительности при портации задачи к доступному аппаратному ресурсу вычислительной системы.
3. Метод сокращения аппаратных затрат кадровой структуры преобразованием к микро-кадрам, выполняющим вычисления с меньшей степенью параллелизма на ограниченном подмножестве функциональных устройств и информационных каналов.

4. Метод преобразования задач, содержащих связанные информационной зависимостью подзадачи с разной степенью параллелизма, на основе согласованной редукции производительности различных по вычислительной трудоемкости кадровых структур с синтезом сбалансированного по интервалу обработки данных решения.

5. Алгоритмы функционирования инструментальных программных средств технологии ресурсонезависимого программирования: алгоритм преобразования параметризованной ресурсом кадровой структуры и алгоритм редукции производительности выделенного фрагмента прикладной задачи.

6. Программа портации кадровой структуры на гибридные вычислительные системы различных архитектур и конфигураций.

7. Программа преобразования фрагментов кадровой структуры на языке COLAMO в микро-кадры.

Указанные результаты диссертации Дордопуло А.И. были использованы:

- при создании экспериментального образца программного комплекса средств высокоуровневого синтеза ресурсонезависимых прикладных решений задач различных предметных областей для реконфигурируемых вычислительных систем в рамках научно-исследовательской и опытно-конструкторской работы «Разработка нового поколения программного обеспечения РВС для решения ресурсоемких научно-технических задач различных предметных областей», шифр «Поколение»;

- при разработке прикладных решений задач символьной обработки для высокопроизводительного вычислительного комплекса в рамках составной части ОКР «Разработка системы трудоемких вычислений реконфигурируемого высокопроизводительного вычислительного комплекса», шифр «Волопас-Т»;

- при разработке ресурсонезависимых программ символьной обработки, линейной алгебры и цифровой обработки сигналов в рамках опытно-конструкторской работы «Разработка высокопроизводительного реконфигурируемого компьютера на основе ПЛИС UltraScale», шифр «Терциус».

Внедрение указанных результатов позволило:

- для задач символьной обработки с информационно-независимыми базовыми подграфами: сократить время преобразования ресурсонезависимых программ в различные архитектуры и конфигурации вычислительных систем в 124-1573 раза, что сократило время портации прикладных задач в 3,2-6,3 раза;

- для задачи символьной обработки с взаимозависимыми неизоморфными подграфами: сократить время преобразования ресурсонезависимой программы в различные архитектуры и конфигурации вычислительных систем в 90-190 раз, что сократило время портации задачи в 3,2-6,1 раза;

- для задач линейной алгебры с информационно-зависимыми базовыми подграфами: сократить время преобразования ресурсонезависимой программы в различные архитектуры и конфигурации вычислительных систем в 20-170 раз, что сократило время портации в 3,2-5,5 раза;

- для задачи расчета энергетических характеристик взаимодействия разных молекул, содержащей информационно-взаимозависимые базовые подграфы: сократить время преобразования ресурсонезависимой программы в 44-77 раз, что сократило время синтеза загрузочных конфигурационных файлов в 3,1-5,8 раза.

ПРЕДСЕДАТЕЛЬ КОМИССИИ

Заместитель директора
по разработкам ООО «НИЦ СЭ и НК»,
к.т.н., с.н.с. О.В. Катаев

ЧЛЕНЫ КОМИССИИ

Начальник отдела прикладного
программного обеспечения ООО «НИЦ СЭ и НК»,
к.т.н. Д.А. Сорокин

Начальник отдела системного
программного обеспечения ООО «НИЦ СЭ и НК»,
к.т.н. И.Г. Данилов

МИНОБРНАУКИ РОССИИ

Федеральное государственное
автономное образовательное учреждение

высшего образования
«Южный федеральный университет»

**НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ИМЕНИ АКАДЕМИКА А.В. КАЛЯЕВА**

Чехова, ул., д. 2, г. Таганрог, 347922
Тел.: (8634) 615-459, факс: 360-376

e-mail: :mail@niimvs.ru http://www.mvs.sfedu.ru

ОКПО 2070157; ОРГН 102610316524
ИНН/КПП 6163027810/616301001

15.11.2022 № 705.07.02-26/203

На _____ № _____

УТВЕРЖДАЮ
Директор НИИ многопроцессорных
вычислительных систем ЮФУ

_____ Я.С. Коровин

"15" 11 2022 г.

# АКТ

о внедрении результатов докторской диссертации «Теоретические основы технологии ресурсонезависимого программирования гибридных вычислительных систем» Дордопуло Алексея Игоревича в НИОКР, выполнявшихся в Научно-исследовательском институте многопроцессорных вычислительных систем имени академика А.В. Каляева федерального государственного автономного образовательного учреждения высшего образования «Южный федеральный университет

Комиссия Научно-исследовательского института многопроцессорных вычислительных систем имени академика А.В. Каляева федерального государственного автономного образовательного учреждения высшего образования «Южный федеральный университет» (НИИ многопроцессорных вычислительных систем ЮФУ) в составе председателя, заместителя директора, д.т.н. Капустяна С.Г. и членов комиссии: заведующего лабораторией многопроцессорных информационно-управляющих систем мехатронных комплексов, к.т.н. Коробкина В.В. и старшего научного сотрудника лаборатории многопроцессорных информационно-управляющих систем интеллектуальных робототехнических комплексов (МИУС ИРК), к.т.н. Усачева Л.Ж. составила настоящий акт о том, что в НИИ многопроцессорных вычислительных систем ЮФУ при выполнении НИОКР внедрены результаты научных исследований Дордопуло А.И.

В ОКР «Разработка технологии создания высокопроизводительных модульно-наращиваемых многопроцессорных вычислительных систем с программируемой архитектурой на основе реконфигурируемой элементной базы» (г/к № 02.447.11.1007 от 06.07.2005 г., шифр "Медведь", № гос. рег.

0122.0510630, ИКРБС 02.2.00700661, 2005-2006) и ОКР «Создание семейства высокопроизводительных многопроцессорных вычислительных систем с динамически перестраиваемой архитектурой на основе реконфигурируемой элементной базы и их математического обеспечения для решения вычислительно трудоемких задач» (г/к № 02.524.12.4002 от 20.04.07 г., шифр “Большая Медведица”, № ГР 01.2007 05707, ИКРБС 0220.0900079, 2007-2009) внедрено параметризуемое вычислительным ресурсом представление кадровой структуры параллельно-конвейерной реализации задачи, что позволило в 3-6 раз сократить время преобразования прикладных программ для реконфигурируемых вычислительных систем, построенных на элементной базе Xilinx Virtex-4, Virtex-5 и Spartan-3.

В НИР «Разработка теоретических основ построения гетерогенных многопроцессорных вычислительных систем с программируемой архитектурой» (шифр “Карнавал”, № ГР 01.2007 02728, ИКРБС 0220.0800098, 2007-2010) и НИР «Разработка и исследование принципов построения и методов программирования реконфигурируемых гетерогенных вычислительных систем с открытой масштабируемой архитектурой» (шифр НШ-3289.2010.9, ИКРБС 01201155173, № ГР 01201155173, 2011) внедрены следующие результаты:

- принципы преобразования параметризуемой аппаратным ресурсом кадровой структуры к архитектуре и конфигурации вычислительной системы с помощью редукции производительности;
- методика синтеза рационального решения при портации ресурсонезависимого представления задачи к архитектуре и конфигурации гетерогенной вычислительной системы.

Внедрение указанных результатов позволило в 3 – 3,5 раза сократить время портации параллельно-конвейерных прикладных программ для различных архитектур гетерогенных вычислительных систем, содержащих реконфигурируемые и процессорные вычислительные узлы, с обеспечением реальной производительности на уровне не ниже 60% от пиковой.

В ОКР «Разработка реконфигурируемой вычислительной системы РВС-7 и организация на ее основе производства реконфигурируемых вычислительных систем с производительностью до $10^{15}$ операций в секунду в одностоечном конструктиве 47U» (шифр «Плеяда», ИКРБС 01201179033, № ГР И111102120521, 2011-2013) и НИР «Разработка и исследование технологии создания ресурсонезависимого прикладного программного обеспечения высокопроизводительных вычислительных систем гибридного типа» (шифр «Русалка», ИКРБС 215033040040, № ГР 114101540025, 2014-2016) внедрены следующие результаты:

- метод сокращения аппаратных затрат кадровой структуры преобразованием к микро-кадрам, выполняющим вычисления с меньшей степенью параллелизма на ограниченном на ограниченном подмножестве функциональных устройств и информационных каналов;
- метод преобразования задач, содержащих связанные информационной зависимостью подзадачи с разной степенью

параллелизма и вычислительной трудоемкостью кадровых структур с синтезом сбалансированного по интервалу обработки данных решения.

Внедрение указанных результатов позволило в 3,2-6,1 раза сократить время портации и обеспечить заданный уровень реальной производительности (не ниже 60% от пиковой) при решении задач символьной обработки, линейной алгебры и цифровой обработки сигналов с различными видами информационной зависимости на ограниченном доступном аппаратном ресурсе.

Практическое использование результатов диссертационной работы Дордопуло А.И. позволило сократить сроки разработки и портации параллельных программ для высокопроизводительных реконфигурируемых вычислительных систем по сравнению с известными решениями в 3-5 раз при решении задач различных проблемных областей.

ПРЕДСЕДАТЕЛЬ КОМИССИИ

Зам. директора, д.т.н. С. Г. Капустян

ЧЛЕНЫ КОМИССИИ:

Зав. лабораторией, к.т.н. В.В. Коробкин

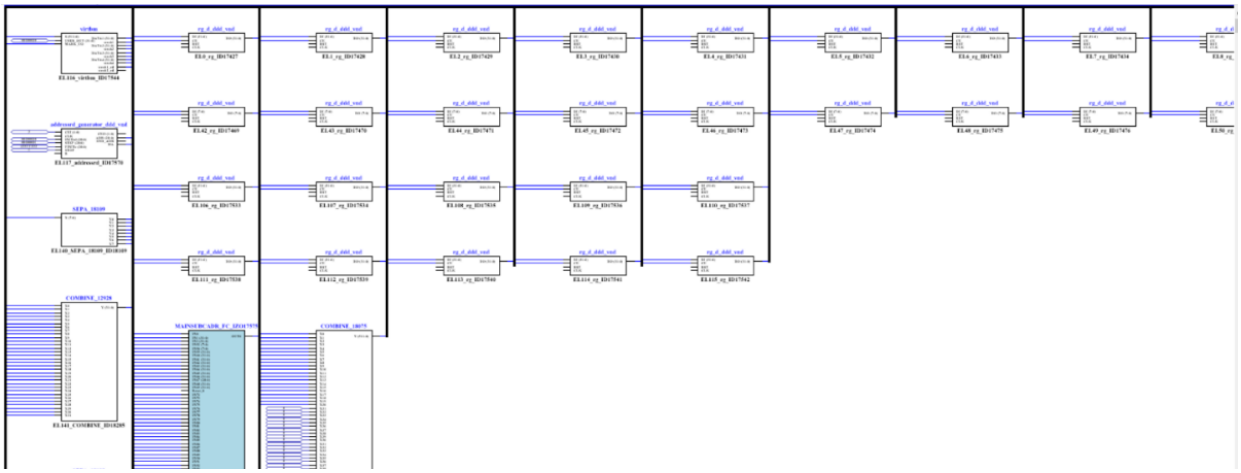
С.н.с. лабораториии МИУС ИРК, к.т.н. Л.Ж. Усачев

**Приложение 3**

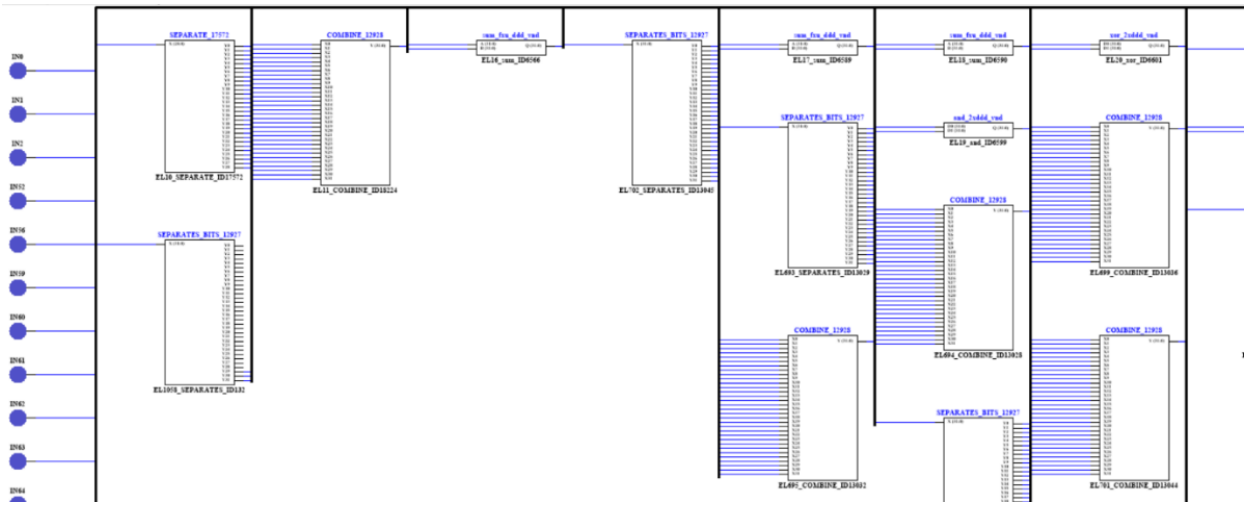
## Результаты портации прикладных задач программными средствами технологии ресурсонезависимого программирования

### П3.1 Задача символьной обработки SHA-128

Представление задачи символьной обработки SHA-128 в окнах программных компонентов разработанных средств ТРНП приведено на рис. П3.1.



а)



б)

Рис. П3.1 Представление задачи SHA-128 в синтезаторе Fire!Constructor а) общий вид задачи; б) представление функционального подграфа MainSubcadr;

Результаты экспериментальной проверки портации задачи SHA-128 разработанными средствами ТРНП на аппаратные платформы приведены в таблице П3.1.

Таблица П3.1. Результаты экспериментальной проверки портации задачи SHA-128

| **Портация прикладной задачи SHA-128** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 41,66 | 41,09 | 41,76 | 42,20 | 41,62 |
| Время портации РНПП средствами ТРНП, с | 10841,66 | 25241,09 | 25241,76 | 10842,20 | 10841,62 |
| Выигрыш преобразование (портация) РНПП | 1382,69 (6,31) | 1401,80 (3,28) | 1379,31 (3,28) | 1364,83 (5,64) | 1383,85 (6,31) |
| Уровень реальной производительности (число БП РНПП/ схемотехники) | **0,86** (12/14) | **1,0** (22/22) | **1,0** (22/22) | **0,80** (4/5) | **0,80** (8/10) |

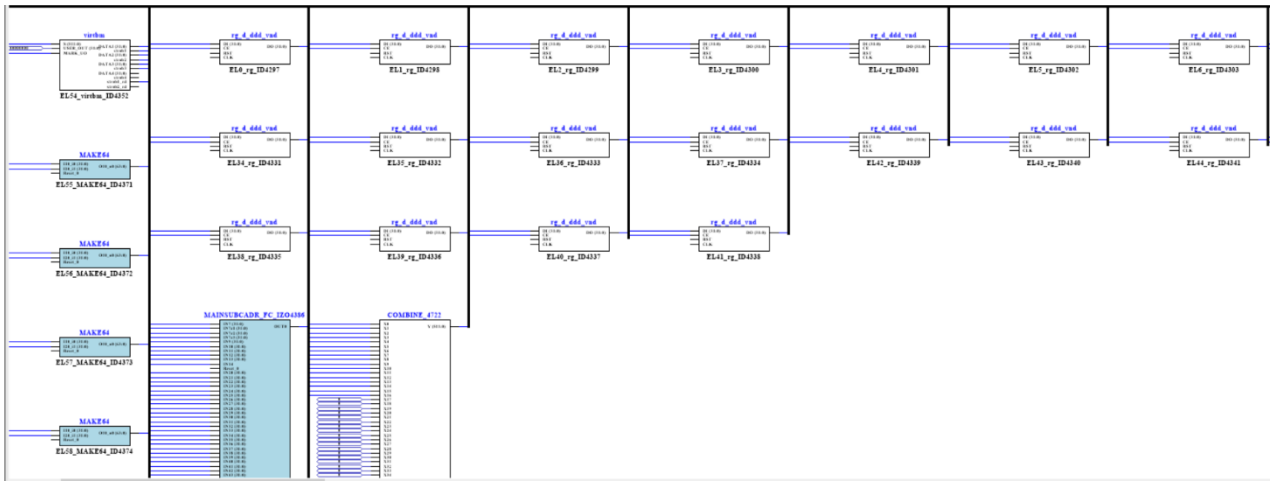
Как видно из данных таблицы П3.1, преобразование кадровой структуры прикладной задачи SHA-128 автоматизированными средствами ТРНП также выполняется существенно (на 2-3 десятичных порядка) быстрее, чем преобразование схемотехником (см. таблицу 5.1). Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3,2 до 6,3 раз, что, как и для других задач с информационно-независимыми подграфами, объясняется заметным вкладом времени синтеза проекта. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,8, что соответствует уровню создаваемых схемотехниками решений задач на РВС и превышает уровень реальной производительности, обеспечиваемый рассмотренными в первой главе HLS-компиляторами.

Полученные результаты портации подтверждают теоретические положения (см. главы 2-3 диссертации) ресурсонезависимого программирования, методы и алгоритмы преобразования кадровых структур в пространстве возможных реализаций. Сокращение времени портации задач с информационно-независимыми подграфами с обеспечением заданного уровня

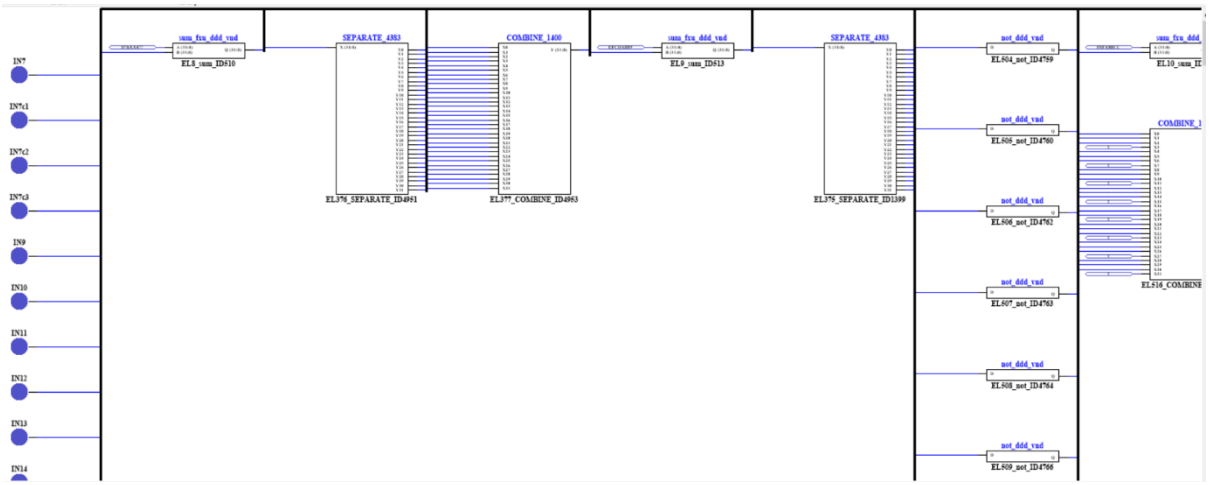
реальной производительности экспериментально подтверждает достижение цели, поставленной в диссертационной работе.

### П3.2 Задача символьной обработки MD5

Представление задачи символьной обработки MD5 в окнах программных компонентов разработанных средств ТРНП приведено на рис. П3.2.



а)



б)

Рис. П3.2 Представление задачи MD5 в синтезаторе Fire!Constructor

а) общий вид задачи; б) представление функционального подграфа MainSubcadr

Результаты экспериментальной проверки портации задачи MD5 разработанными средствами ТРНП на аппаратные платформы приведены в таблице П3.2.

Таблица П3.2. Результаты экспериментальной проверки портации задачи MD5

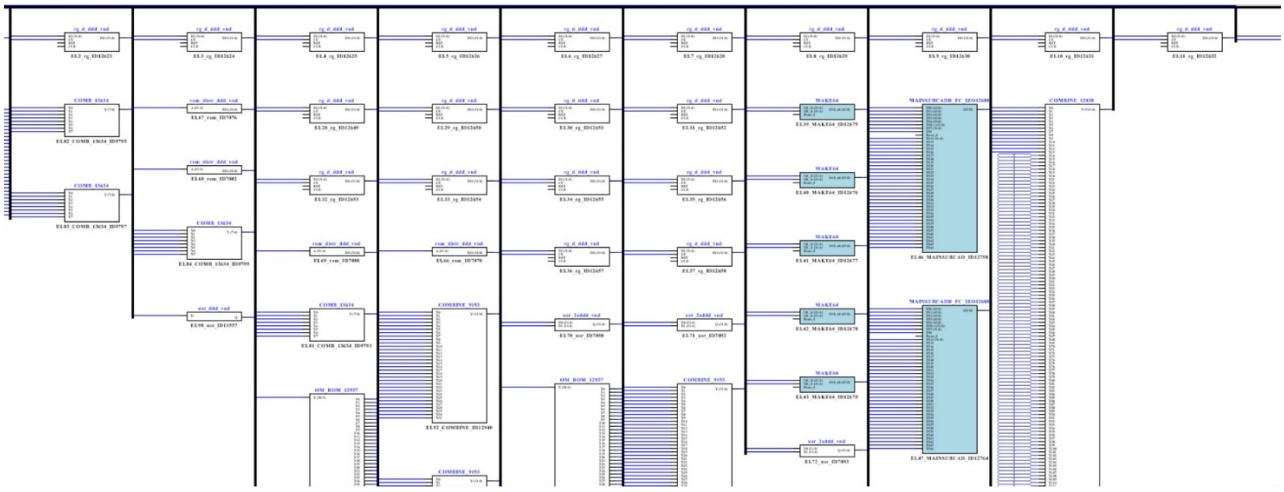
| **Портация прикладной задачи MD5** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 138,86 | 145,13 | 144,79 | 143,12 | 144,60 |
| Время портации РНПП средствами ТРНП, с | 10938,86 | 25345,13 | 25344,79 | 10943,12 | 10944,60 |
| Выигрыш преобразование (портация) РНПП | 414,80 (6,25) | 396,90 (3,27) | 397,83 (3,27) | 402,45 (5,59) | 398,33 (6,25) |
| Уровень реальной производительности (число БП РНПП/ схемотехники) | **0,63** (20/32) | **0,67** (37/55) | **0,67** (37/55) | **0,64** (7/11) | **0,67** (14/21) |

Как видно из данных таблицы П3.2, преобразование кадровой структуры прикладной задачи MD5 автоматизированными средствами ТРНП также выполняется существенно (на 2-3 десятичных порядка) быстрее, чем преобразование схемотехником (см. таблицу 5.1). Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3,2 до 6,2 раз, что, как и для других задач с информационно-независимыми подграфами, объясняется заметным вкладом времени синтеза проекта. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,6, что соответствует уровню создаваемых схемотехниками решений задач на РВС и превышает уровень реальной производительности, обеспечиваемый рассмотренными в первой главе HLS-компиляторами.

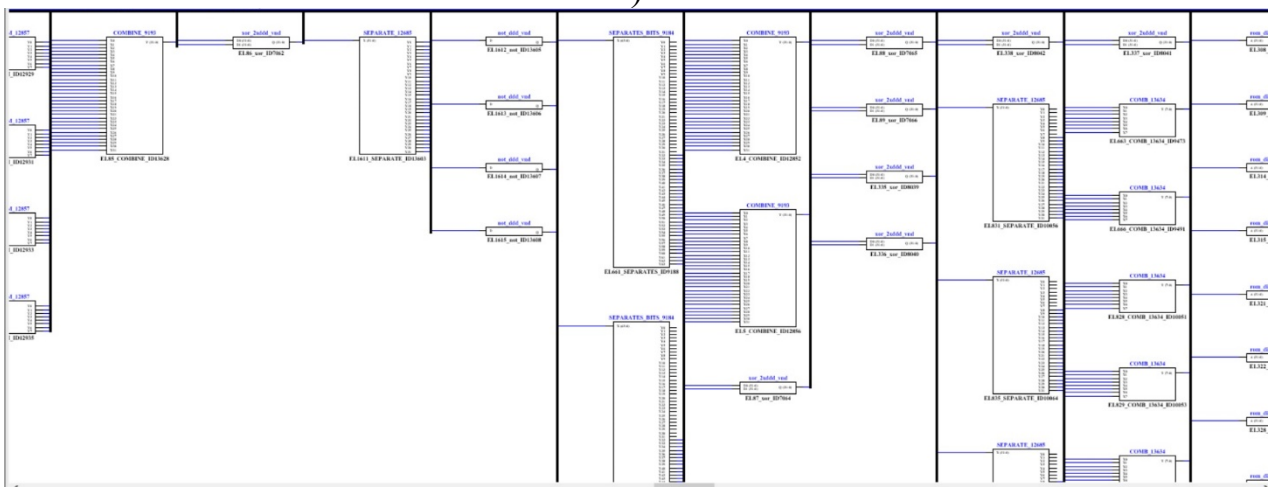
Полученные результаты портации подтверждают теоретические положения (см. главы 2-3 диссертации) ресурсонезависимого программирования, методы и алгоритмы преобразования кадровых структур в пространстве возможных реализаций. Сокращение времени портации задач с информационно-независимыми подграфами с обеспечением заданного уровня реальной производительности экспериментально подтверждает достижение цели, поставленной в диссертационной работе.

### П3.3 Задача символьной обработки AES

Представление задачи символьной обработки AES в окнах программных компонентов разработанных средств ТРНП приведено на рис. П3.3.



а)



б)

Рис. П3.3 Представление задачи AES в синтезаторе Fire!Constructor
а) общий вид задачи; б) представление функционального подграфа MainSubcadr

Результаты экспериментальной проверки портации задачи AES разработанными средствами ТРНП на аппаратные платформы приведены в таблице П3.3.

Таблица П3.3. Результаты экспериментальной проверки портации задачи AES

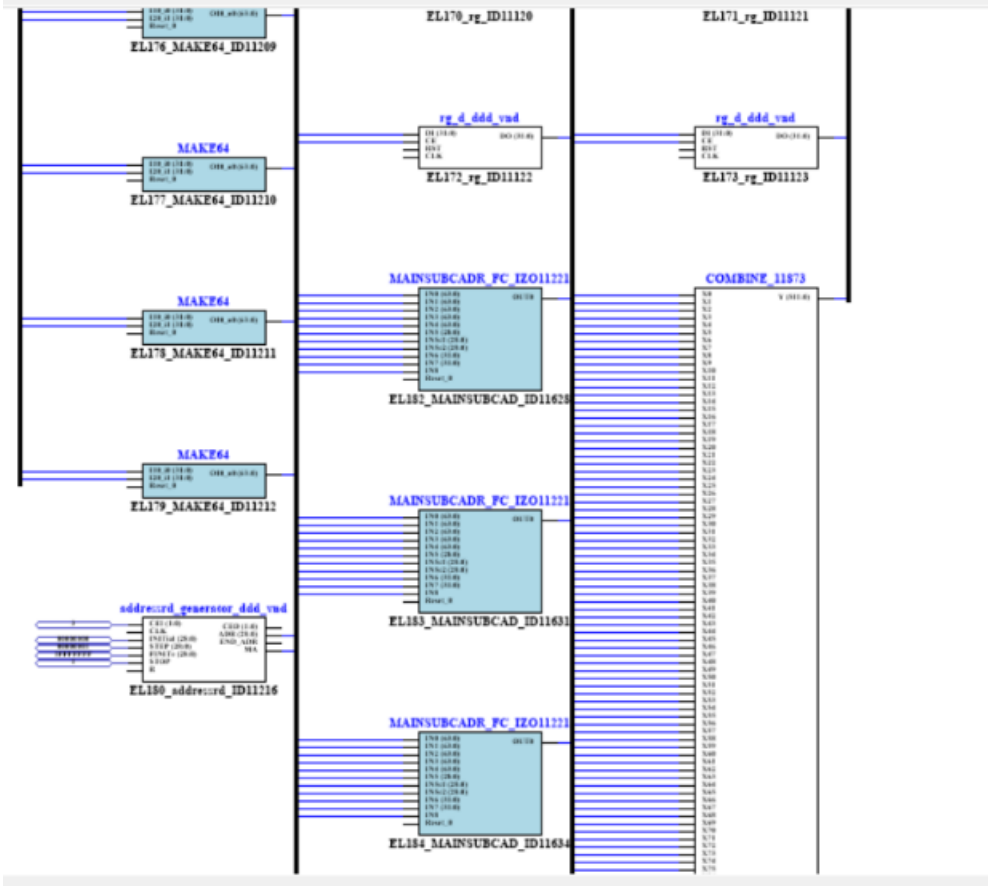
| **Портация прикладной задачи AES** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 447,07 | 461,83 | 462,27 | 460,11 | 459,07 |
| Время портации РНПП средствами ТРНП, с | 11247,07 | 25661,83 | 25662,27 | 11260,11 | 11259,07 |
| Выигрыш преобразование (портация) РНПП | 128,84 (6,08) | 124,72 (3,23) | 124,60 (3,23) | 125,19 (5,44) | 125,47 (6,08) |
| Уровень реальной производительности (число БП РНПП/ схемотехники) | **0,82** (14/17) | **0,89** (24/27) | **0,89** (24/27) | **0,67** (4/6) | **0,70** (7/10) |

Как видно из данных таблицы П3.3, преобразование кадровой структуры прикладной задачи AES автоматизированными средствами ТРНП также выполняется на 2 десятичных порядка быстрее, чем преобразование схемотехником (см. таблицу 5.1). Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3,2 до 6 раз, что, как и для других задач с информационно-независимыми подграфами, объясняется заметным вкладом времени синтеза проекта. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,6, что соответствует уровню создаваемых схемотехниками решений задач на РВС и превышает уровень реальной производительности, обеспечиваемый рассмотренными в первой главе HLS-компиляторами.

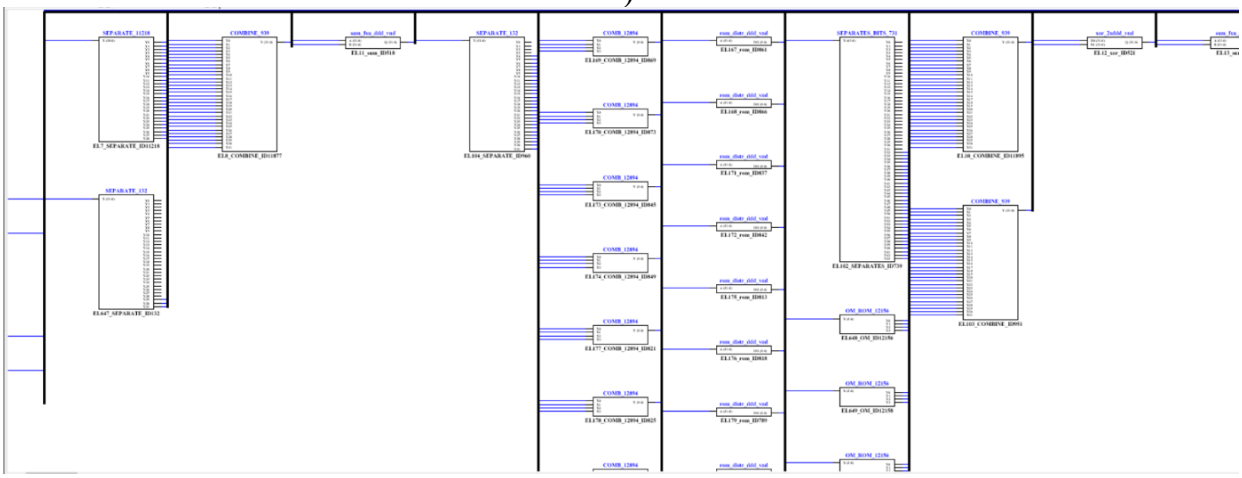
Полученные результаты портации подтверждают теоретические положения (см. главы 2-3 диссертации) ресурсонезависимого программирования, методы и алгоритмы преобразования кадровых структур в пространстве возможных реализаций. Сокращение времени портации задач с информационно-независимыми подграфами с обеспечением заданного уровня реальной производительности экспериментально подтверждает достижение цели, поставленной в диссертационной работе.

## П3.4 Задача символьной обработки ГОСТ

Представление задачи символьной обработки ГОСТ в окнах программных компонентов разработанных средств ТРНП приведено на рис. П3.4.




а)



б)

Рис. П3.4 Представление задачи ГОСТ в синтезаторе Fire!Constructor
а) общий вид задачи; б) представление функционального подграфа MainSubcadr

Результаты экспериментальной проверки портации задачи ГОСТ разработанными средствами ТРНП на аппаратные платформы приведены в таблице П3.4.

Таблица П3.4. Результаты экспериментальной проверки портации задачи ГОСТ

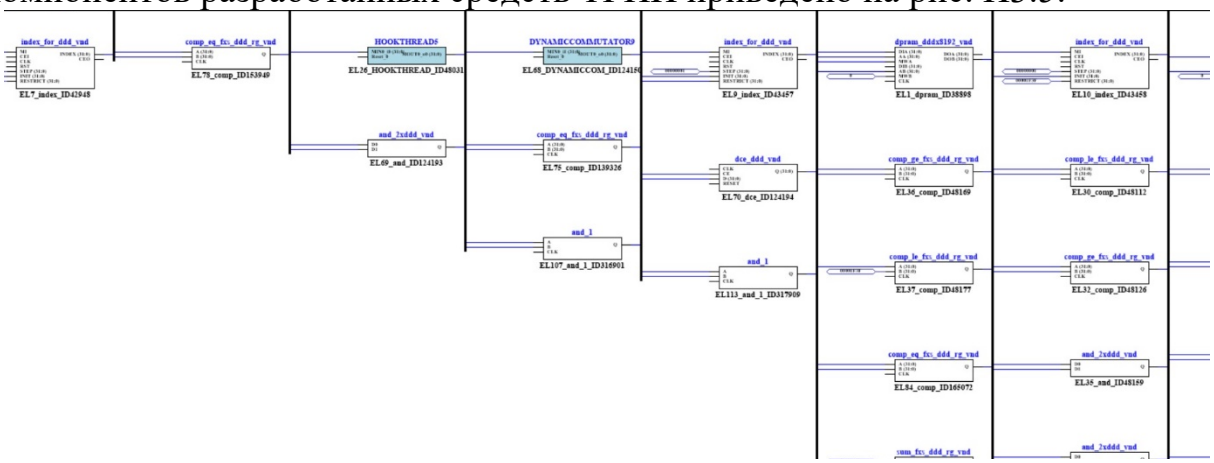
| **Портация прикладной задачи ГОСТ** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 19,77 | 20,90 | 21,01 | 14,65 | 15,23 |
| Время портации РНПП средствами ТРНП, с | 10819,77 | 25220,90 | 25221,01 | 10814,65 | 10815,23 |
| Выигрыш преобразование (портация) РНПП | 2914,09 (6,32) | 2755,59 (3,28) | 2741,55 (3,28) | 3931,74 (5,66) | 3782,01 (6,32) |
| Уровень реальной производительности (число БП РНПП/ схемотехники) | **0,60** (72/120) | **0,67** (124/185) | **0,67** (124/185) | - | - |

Как видно из данных таблицы П3.4, преобразование кадровой структуры прикладной задачи ГОСТ автоматизированными средствами ТРНП также выполняется на 3 десятичных порядка быстрее, чем преобразование схемотехником (см. таблицу 5.1). Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3,2 до 6,3 раз. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,6. Прочерки для ГВС «Атлант-10» и ГВС «Атлант-Д» вызваны отсутствием данных о схемотехнической реализации задачи ГОСТ на этих вычислительных системах.

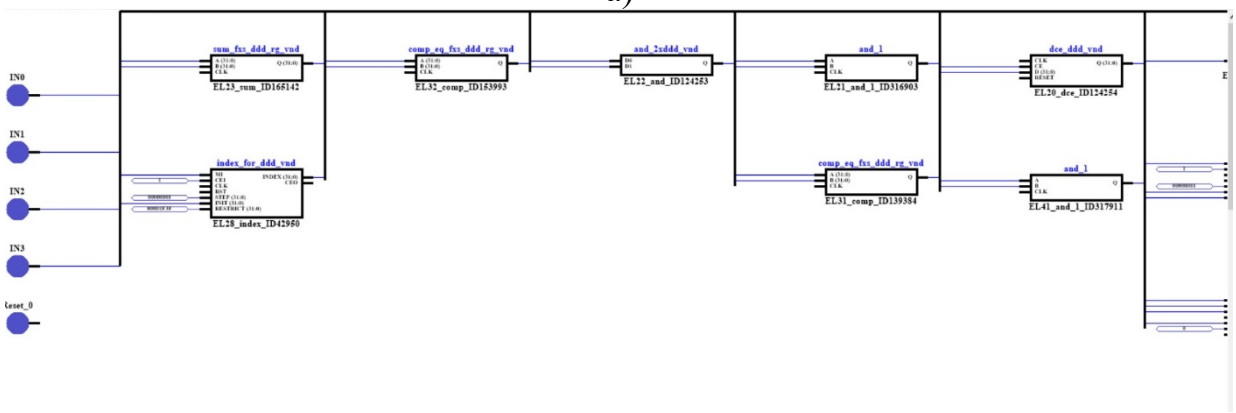
Полученные результаты портации подтверждают теоретические положения (см. главы 2-3 диссертации) ресурсонезависимого программирования, методы и алгоритмы преобразования кадровых структур в пространстве возможных реализаций. Сокращение времени портации задач с информационно-независимыми подграфами с обеспечением заданного уровня реальной производительности экспериментально подтверждает достижение цели, поставленной в диссертационной работе.

### П3.5 Задача LU-разложения

Представление задачи LU-разложения в окнах программных компонентов разработанных средств ТРНП приведено на рис. П3.5.



а)



б)

Рис. П3.5 Представление задачи LU-разложения в синтезаторе Fire!Constructor

а) общий вид задачи; б) представление функционального подграфа MainSubcadr

Результаты экспериментальной проверки портации задачи LU-разложения разработанными средствами ТРНП на аппаратные платформы приведены в таблице П3.5.

Таблица П3.5. Результаты экспериментальной проверки портации задачи LU-разложения

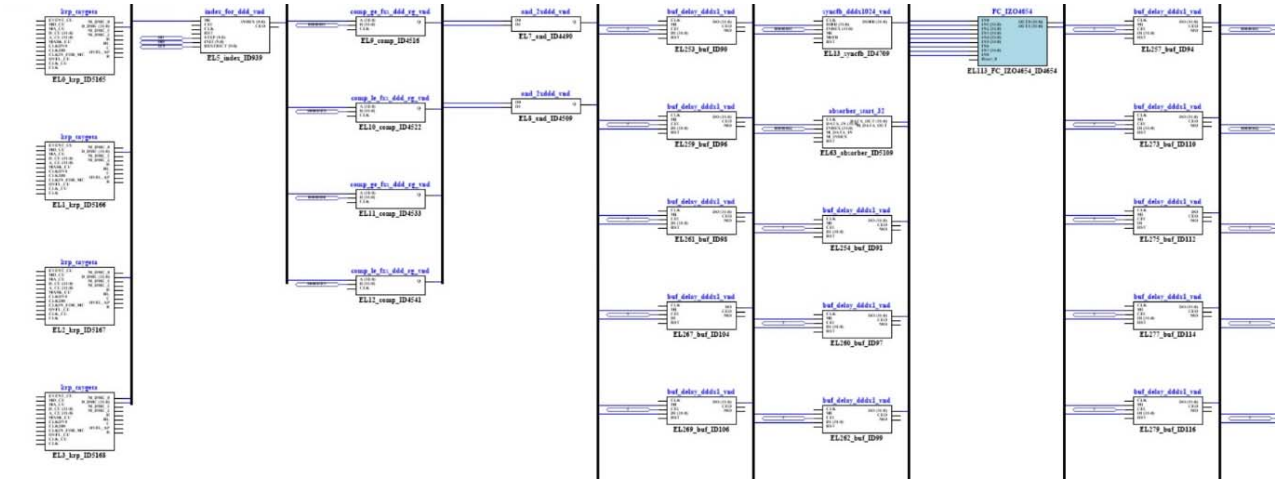
| **Портация задачи LU-разложения** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 726,93 | 945,34 | 945,03 | 513,28 | 517,47 |
| Время портации РНПП средствами ТРНП, с | 11526,93 | 26145,34 | 26145,03 | 11313,28 | 11317,47 |
| Выигрыш преобразование (портация) РНПП | 79,24 (5,93) | 60,93 (3,17) | 60,95 (3,17) | 112,22 (5,41) | 111,31 (6,04) |
| Уровень реальной производительности (число БП РНПП/ схемотехники) | **0,60** (211/350) | **0,60** (308/508) | **0,60** (307/508) | - | - |

Как видно из данных таблицы П3.5, преобразование кадровой структуры прикладной задачи LU-разложения автоматизированными средствами ТРНП также выполняется на 2 десятичных порядка быстрее, чем преобразование схемотехником (см. таблицу 5.1). Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3 до 6 раз. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,6. Прочерки для ГВС «Атлант-10» и ГВС «Атлант-Д» вызваны отсутствием данных о схемотехнической реализации задачи LU-разложения на этих вычислительных системах.

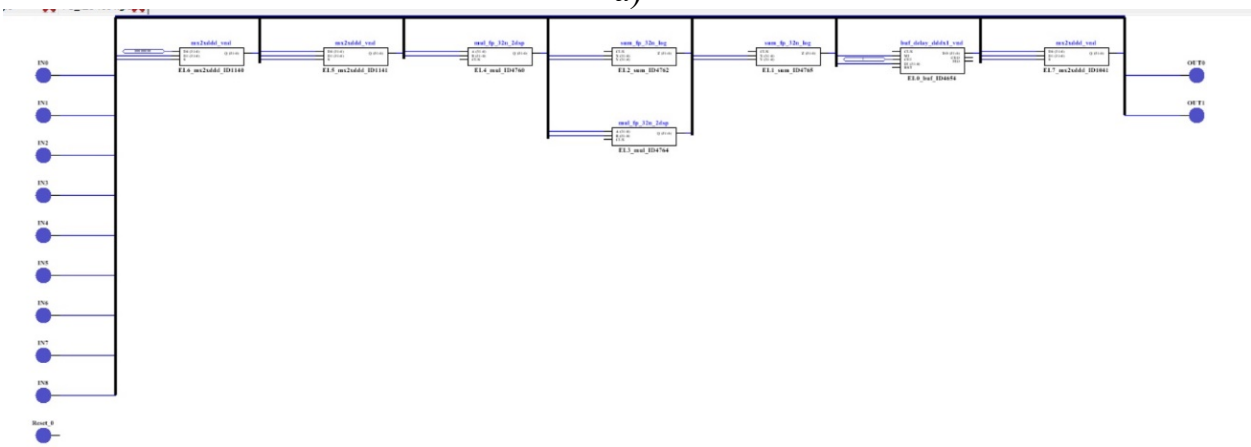
Полученные результаты портации подтверждают теоретические положения (см. главы 2-3 диссертации) ресурсонезависимого программирования, методы и алгоритмы преобразования кадровых структур в пространстве возможных реализаций. Сокращение времени портации задач с информационно-независимыми подграфами с обеспечением заданного уровня реальной производительности экспериментально подтверждает достижение цели, поставленной в диссертационной работе.

### П3.6 Задача решения СЛАУ методом Гаусса-Зейделя

Представление задачи решения СЛАУ методом Гаусса-Зейделя в окнах программных компонентов разработанных средств ТРНП приведено на рис. П3.6.



а)



б)

Рис. П3.6 Представление задачи решения СЛАУ методом Гаусса-Зейделя в синтезаторе Fire!Constructor

а) общая вид задачи; б) представление функционального подграфа MainSubcadr

Результаты экспериментальной проверки портации задачи решения СЛАУ методом Гаусса-Зейделя разработанными средствами ТРНП на аппаратные платформы приведены в таблице П3.6.

Таблица П3.6. Результаты экспериментальной проверки портации задачи решения СЛАУ методом Гаусса-Зейделя

| **Портация задачи решения СЛАУ методом Гаусса-Зейделя** | **Аппаратные платформы** | | | | |
|---|---|---|---|---|---|
| | РВС «Тайгета» | РВС «Терциус» | РВС «Терциус-3» | ГВС «Атлант-10» | ГВС «Атлант-Д» |
| Время преобразования РНПП средствами ТРНП, с | 125,48 | 175,90 | 176,10 | 84,38 | 90,23 |
| Время портации РНПП средствами ТРНП, с | 10925,48 | 25375,90 | 25376,10 | 10884,38 | 10890,23 |
| Выигрыш преобразование (портация) РНПП | 459,04 (6,26) | 327,46 (3,26) | 327,09 (3,26) | 682,63 (5,62) | 638,37 (6,28) |
| Уровень реальной производительности (число БП РНПП/ схемотехники) | **0,60** (493/824) | **0,63** (762/1204) | **0,63** (762/1204) | - | - |

Как видно из данных таблицы П3.6, преобразование кадровой структуры прикладной задачи решения СЛАУ методом Гаусса-Зейделя автоматизированными средствами ТРНП также выполняется на 2 десятичных порядка быстрее, чем преобразование схемотехником (см. таблицу 5.1). Выигрыш во времени портации с синтезом загрузочного конфигурационного файла находится в диапазоне от 3 до 6 раз. Достигнутый уровень реальной производительности для этой задачи варьируется в зависимости от целевой архитектуры и составляет не менее 0,6. Прочерки для ГВС «Атлант-10» и ГВС «Атлант-Д» вызваны отсутствием данных о схемотехнической реализации задачи LU-разложения на этих вычислительных системах.

### П3.7 Задача решения СЛАУ методом Гаусса-Зейделя

Представленные в таблицах 5.1-5.7 пятой главы и таблицах П3.1-П3.6 результаты времени портации прикладных программ и достигнутой ими реальной производительности могут быть представлены в виде диаграмм выигрыша по времени портации (рис. П3.8) и достигнутой реальной производительности (рис. П3.9).

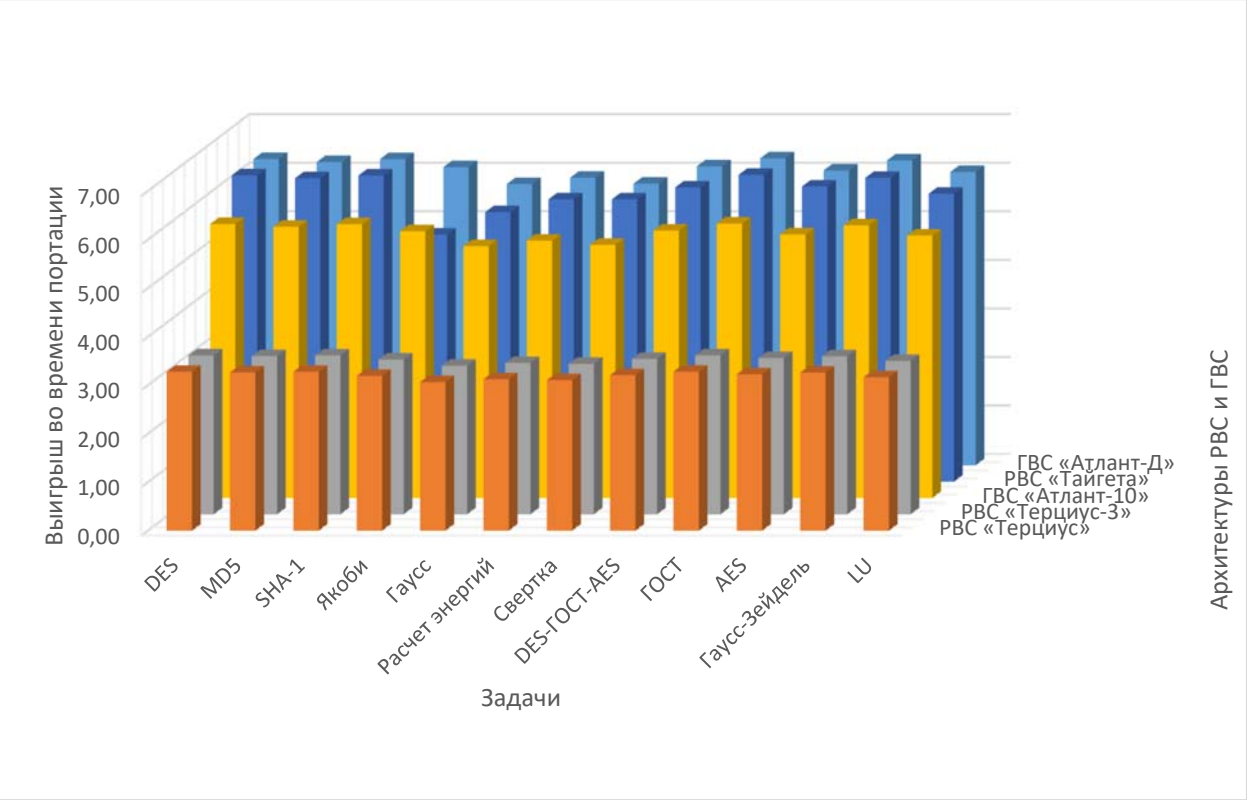



Рис. П3.12 Выигрыш по времени портации прикладных задач для РВС и ГВС

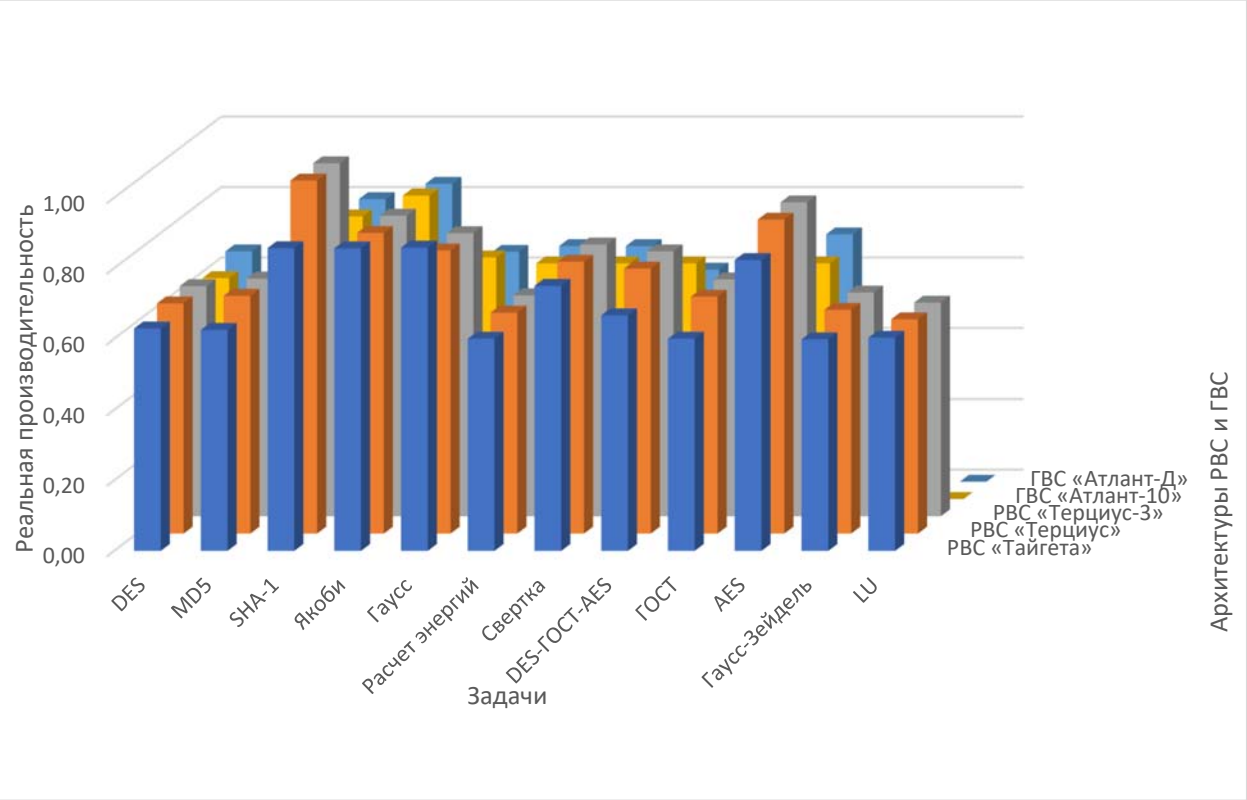



Рис. П3.13 Реальная производительность прикладных задач для РВС и ГВС

Полученные в результате апробации данные позволяют сделать вывод о том, что применение разработанных средств ТРНП сокращает время портации прикладных задач с различными типами информационной зависимости, что, в свою очередь, экспериментально подтверждает четвертое положение, выдвигаемое для защиты, и свидетельствует о достижении поставленной цели – минимизации времени портации прикладных задач для ГВС.