ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ
«ФЕДЕРАЛЬНЫЙ НАУЧНЫЙ ЦЕНТР
НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ СИСТЕМНЫХ
ИССЛЕДОВАНИЙ РОССИЙСКОЙ АКАДЕМИИ НАУК»

На правах рукописи
УДК 519.684.6

Сударева Ольга Юрьевна

ВСТРЕЧНАЯ ОПТИМИЗАЦИЯ КЛАССА ЗАДАЧ ТРЁХМЕРНОГО МОДЕЛИРОВАНИЯ ДЛЯ АРХИТЕКТУР МНОГОЯДЕРНЫХ ПРОЦЕССОРОВ

Специальность 05.13.11 —
«Математическое и программное обеспечение вычислительных машин,
комплексов и компьютерных сетей»

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
кандидат физико-математических наук, доцент
Кушниренко Анатолий Георгиевич

Москва — 2018

Оглавление

Стр.

Введение

Данная диссертационная работа посвящена встречной оптимизации архитектуры вычислительных систем и ряда вычислительных процедур, применяемых для решения задач трёхмерного моделирования.

Актуальность темы исследования обусловлена следующими причинами. Согласно принятой Стратегии развития электронной промышленности России до 2025 года [32], к числу приоритетных направлений развития относится преодоление технологического отставания отечественной электронно-компонентной базы (ЭКБ) от мирового уровня, повышение её конкурентоспособности и увеличение доли на внутреннем и мировом рынках. Наиболее важными являются такие отрасли электронной промышленности, как производство электроники специального назначения (для современных средств вооружения, авиационных и космических систем, и других) и производство профессиональной электроники, в том числе электроники для систем безопасности, энергетической аппаратуры, медицинского оборудования, а также высокопроизводительных вычислительных систем (ВС) обработки информации.

Вместе с тем, государственная программа «Развитие науки и технологий» [7] предполагает формирование конкурентоспособного сектора разработок и исследований (фундаментальных и прикладных), обладающего технологической базой мирового уровня. Существенная доля современных российских научных исследований включает на определённом этапе численное решение тех или иных задач, связанное с проведением масштабных расчётов [46; 47]. Таким образом, высокопроизводительные ВС несомненно относятся к технологической базе исследований.

В настоящее время для расчётов — как в науке, так и в промышленности — повсеместно применяются параллельные ВС, собранные из розничных комплектующих мировых производителей. Наряду с ВС классической архитектуры, построенными из универсальных процессоров, в последние годы широкое распространение получили гибридные ВС, включающие графические ускорители (GPU). Это могут быть как настольные системы или кластеры из нескольких узлов, для одного предприятия или научного коллектива, так и крупные совместно используемые суперкомпьютеры. К последней категории относят-

ся суперкомпьютеры К100 [38] и Ломоносов-2 [12]. Несмотря на актуальность проблемы импортозамещения и обеспечения информационной безопасности, на данный момент отечественные технологии в этой области практически не представлены.

Что касается, однако, вычислений специального назначения, здесь имеется ряд конкурентоспособных и успешно применяемых отечественных разработок. Так, линейка КОМДИВ микропроцессоров цифровой обработки сигналов (ЦОС), имеющих гибридную архитектуру [25], и коммуникационные СБИС, разработанные в НИИСИ РАН, используются в вычислительных комплексах реального времени для обработки гидроакустических и радиолокационных данных. Альтернативой для ЦОС являются системы на основе процессоров архитектуры Эльбрус [13] разработки АО МЦСТ. Имеется и ряд универсальных процессоров с архитектурой Эльбрус; на их основе в настоящее время выпускаются персональные компьютеры и серверные системы, однако в перспективе сфера применения может быть расширена и охватить также высокопроизводительные ВС.

Одним из возможных путей увеличения доли отечественной ЭКБ на внутреннем, а в перспективе и мировом, рынке, и одновременно обеспечения технической базы для научных исследований, представляется развитие имеющихся аппаратных решений, с целью дальнейшего применения в вычислениях общего назначения. Такое развитие, безусловно, должно учитывать мировой опыт в разработке процессоров и высокопроизводительных ВС.

Данное исследование нацелено на анализ ключевых факторов, влияющих на производительность расчётов на гибридной ВС, выявление, с учётом этих факторов, достоинств и недостатков отечественных многоядерных микропроцессоров, разработанных в НИИСИ РАН, и подготовку предложений по модернизации программной модели архитектуры этих процессоров.

Степень разработанности проблемы. Высокопроизводительные вычисления (ВПВ) являются одной из актуальных в современном мире предметных областей. К ВПВ традиционно относят наиболее трудоёмкие расчётные задачи, требующие большого объёма памяти, мощности процессоров и, главным образом, времени на проведение расчётов. По мере развития технологий расширяется класс целевых задач, возрастают алгоритмическая сложность и

объём обрабатываемых данных. Соответственно, к ВС предъявляются всё более жёсткие требования.

Неослабевающий интерес к изучению различных аспектов ВПВ и высокопроизводительных ВС нашёл отражение в многочисленных исследованиях как зарубежных, так и российских авторов, которые можно условно разделить на ряд категорий.

1. Эффективная реализация определённых вычислительных процедур на ВС той или иной архитектуры, в частности, на гибридных системах: [53; 89; 110; 124] — быстрое преобразование Фурье; [28; 125; 131] — многосеточные методы и другие трафаретные вычисления; [70; 100; 104] — вычисления с плотными матрицами; [45; 54; 60; 68; 75; 84; 90; 102; 109; 113; 119; 129; 134; 158] — вычисления с разреженными матрицами.
2. Методы распараллеливания вычислений: [39; 77; 79; 95; 96; 143; 151; 156].
3. Описания различных программных библиотек, предназначенных для эффективного решения того или иного класса задач: [48; 63; 64; 72].
4. Особенности конкретных ВС и их использования в ВПВ: [59; 62; 81; 107; 116; 122; 141].
5. Виды параллелизма в различных архитектурах, их эффективное использование: [42; 85; 99; 149].
6. Особенности подсистемы памяти на различных ВС, оптимизация работы с ней: [40; 42; 50; 53; 88; 89; 105; 119; 134; 147; 153].
7. Высокопроизводительные коммуникационные среды: [78; 93; 114; 120; 142].
8. Энергоэффективность ВС: [85; 87; 108].
9. Инструментальные программные средства разработки: [42; 112; 126; 128; 132; 158].
10. Переносимость кодов, в том числе и так называемая «переносимость производительности», автоматизация разработки: [54; 73; 82; 88; 115; 118; 125; 130–132; 152; 157].
11. Обзоры современного состояния и перспектив ВПВ и связанные вопросы: [67; 106; 133; 136; 155].

Все упомянутые исследования, в которых рассматриваются конкретные аппаратные платформы, посвящены, однако, вычислениям на универсальных процессорах и ускорителях — графических и других — от мировых производителей. Это неудивительно, поскольку на настоящий момент аналогичные системы на отечественной ЭКБ в розничной продаже отсутствуют.

Тем не менее, имеется некоторое количество публикаций, в которых рассматриваются системы на платформе Эльбрус и их производительность на некоторых задачах: [6; 11; 15; 20]. Для процессоров Эльбрус разработана библиотека оптимизированных математических процедур EML [10]. В [29] сообщается об успешном завершении первого этапа переноса программного комплекса вычислительной аэро- и гидродинамики FlowVision на платформу Эльбрус и говорится о начале исследований производительности архитектуры Эльбрус на различных задачах.

Для процессоров КОМДИВ разработана библиотека цифровой обработки сигналов (БЦОС), включающая низкоуровневые оптимизированные математические процедуры ([1; 30]). В [27] рассматривается программирование задач ЦОС. Ряд публикаций [14; 21—23] посвящён особенностям коммуникационной среды в многопроцессорных комплексах на платформе КОМДИВ. Однако исследований влияния специфики архитектуры гибридных процессоров НИИСИ РАН на производительность определённых вычислительных процедур систематически не проводилось.

Как будет продемонстрировано в дальнейших главах, архитектура процессоров КОМДИВ и многопроцессорных комплексов на их основе имеет целый ряд общих черт с архитектурой современных гибридных систем. Возникает вопрос о возможности доработки имеющейся элементной базы и программного обеспечения, с учётом мировых тенденций, с целью дальнейшего использования для более широкого класса научных и инженерных расчётов и, в перспективе, построения отечественного суперкомпьютера, производительность которого будет измеряться в ПетаОП/с.

Таким образом, **целью диссертационной работы** является разработка методов моделирования и оценки влияния ключевых характеристик гибридной архитектуры вычислительной системы на производительность системы при решении задач трёхмерного моделирования.

Поскольку круг вопросов, связанных с производительностью гибридных ВС, очень обширен, он не может быть охвачен в одной работе. Исследование ограничено рамками нескольких типовых процедур, часто используемых в науке и промышленности, в том числе, для трёхмерного моделирования различных процессов, и гибридной вычислительной системы с сопроцессорами массивно-параллельной архитектуры.

Для достижения поставленной цели необходимо было решить следующие **задачи**.

1. Разработать метод, который позволит получать теоретические оценки ожидаемой производительности той или иной вычислительной процедуры до начала её реализации на выбранной вычислительной системе.
2. Выбрать тестовый набор процедур и применить к нему разработанный метод. Реализовать выбранные процедуры на доступных системах из рассматриваемого класса: как собранных из импортных комплектующих, так и на базе отечественных процессоров КОМДИВ. Подтвердить применимость метода, сопоставив результаты тестирования этих реализаций с теоретическими оценками, выведенными при помощи разработанного метода.
3. На основании полученных данных подготовить проект доработки программных эмуляторов, используемых в ходе разработки гибридных микропроцессоров ФГУ ФНЦ НИИСИ РАН.

Научная новизна.

1. Разработана новая модель гибридной вычислительной системы, которая позволяет для заданной вычислительной процедуры вывести теоретическую оценку производительности этой процедуры и оценить сбалансированность вычислительной системы для выполнения этой процедуры.
2. Разработан метод оценки ожидаемой производительности вычислительной процедуры на гибридной системе.
3. Выведен формальный критерий сбалансированности вычислительной системы на заданной вычислительной процедуре.
4. С помощью разработанного метода исследованы наблюдения о влиянии пропускной способности канала доступа к памяти на производительность вычислений на гибридных системах на базе GPU.

Теоретическая и практическая значимость. Разработанный метод позволяет для каждой новой вычислительной процедуры оценить ожидаемую производительность на целевой ВС. С учётом этой оценки может быть принято решение о целесообразности использования данной системы, до начала работ по реализации процедуры. С другой стороны, можно определить потенциальные преимущества каждой новой системы при решении имеющихся задач путём подстановки параметров в формулы. В дальнейшем, сопоставив реальную производительность разработанной оптимизированной процедуры с выведенной при помощи метода оценкой, можно выявить, какие особенности архитектуры не позволяют достичь теоретического максимума производительности и требуют доработки.

С помощью разработанного метода впервые проведено исследование производительности набора процедур, применяемых в трёхмерном моделировании, на гибридных процессорах КОМДИВ ВМ7 и ВМ9 оригинальной отечественной архитектуры.

Разработан и обоснован проект доработки программной модели гибридных многоядерных процессоров НИИСИ РАН с целью приближения реальной производительности к ожидаемой производительности, выведенной при помощи разработанного метода. Разработка следующих поколений процессоров с учётом предложенных усовершенствований позволит добиться высокой производительности и эффективности вычислений не только на задачах ЦОС, но и на широком классе других задач.

В ходе исследования получен опыт написания законченной иерархии кодов оптимизированных процедур для процессоров ВМ7/9: от вычислительных ядер на сопроцессоре СР2 до MPI-программ на управляющих процессорах. В частности, разработаны процедуры БПФ и свёртки, которые вошли в состав программного обеспечения (ПО) обработки сигналов для вычислительных комплексов реального времени и имеют производственные применения. Неулучшаемость этих процедур обоснована при помощи разработанного метода оценки ожидаемой производительности.

Реализация процедуры NPB MG для GPU, выполненная автором в ходе диссертационного исследования, оказалась более производительной, чем аналогичные процедуры, описанные в открытых публикациях.

Результаты исследования представляют интерес для специалистов в области параллельных и высокопроизводительных вычислений, в том числе на ВС гибридной архитектуры, а также для разработчиков программных эмуляторов таких систем.

Методология и методы исследования. Для оценки ожидаемой производительности вычислительной процедуры на гибридной ВС построена обобщённая модель гибридной ВС и разработан метод, позволяющий оценить производительность как величину, зависящую от параметров ВС и от параметров самой процедуры. На основе этого метода выведен критерий сбалансированности ВС на заданной вычислительной процедуре. Математическую основу исследования составляет теория алгоритмов.

Процедуры для гибридных систем на базе GPU были реализованы с использованием открытого стандарта OpenCL для вычислений на GPU, MPI для обменов данными между ускорителями на узле и между узлами, а также OpenMP для вспомогательных вычислений на управляющем процессоре. Эти процедуры были протестированы на гибридных узлах и кластере НИИСИ РАН, а также на суперкомпьютере К100 ИПМ им. М. В. Келдыша РАН. Процедуры для процессоров КОМДИВ ВМ7 и ВМ9 были реализованы при помощи специализированного интерфейса на языке С для управляющего процессора и языка ассемблера для сопроцессора СР2, а также стандарта MPI для обменов данными между процессорами. Эти процедуры были протестированы на эмуляторах СР2 и контроллера DMA и на существующих процессорных модулях ВМ7 и ВМ9.

Достоверность исследования обеспечивается тем, что рассмотренные вычислительные процедуры из набора тестов NAS Parallel Benchmarks были реализованы для гибридных систем на базе GPU, с использованием открытого стандарта OpenCL для вычислений на GPU, и были протестированы на гибридных узлах и кластере НИИСИ РАН, а также на суперкомпьютере К100 ИПМ им. М. В. Келдыша РАН. Реализации процедур для процессоров ВМ7 и ВМ9 использованы в производственных приложениях.

Основные положения, выносимые на защиту.

1. Разработана модель гибридной вычислительной системы, охватывающая широкий класс отечественных и зарубежных архитектур.

2. Разработан метод оценки ожидаемой производительности вычислительной процедуры на гибридных вычислительных системах; выведен формальный критерий сбалансированности вычислительной системы на заданной вычислительной процедуре.
3. Применимость метода подтверждена результатами измерения производительности разработанных автором реализаций нескольких широко применяемых в трёхмерном моделировании вычислительных процедур из набора тестов NAS Parallel Benchmarks на ряде отечественных и импортных вычислительных систем.
4. При помощи разработанного метода обоснована неулучшаемость разработанных оптимизированных библиотечных процедур БПФ и свёртки для отечественных гибридных процессоров ВМ7 и ВМ9.
5. Разработан проект оптимизации архитектуры гибридных процессоров НИИСИ РАН, позволяющий за счёт локальных усовершенствований программных моделей подсистем процессора достичь существенного роста производительности на классах вычислительных задач, рассмотренных в диссертации. Проект предполагает расширение функциональных возможностей контроллера DMA и сопроцессора CP2, а также улучшение нескольких количественных характеристик сопроцессора.

Апробация работы. Основные результаты работы докладывались на следующих конференциях.

1. Международная конференция «High Performance Computing 2013». Киев, 7-11 октября 2013.
2. XV международная конференция «Супервычисления и математическое моделирование». Саров, 13-17 октября 2014.
3. Международная конференция «The 5th GPU Workshop — The Future of Many-Core Computing in Science 2015». Будапешт, 20-21 мая 2015.
4. 14-й Международный Междисциплинарный Семинар «Математические Модели и Моделирование в Лазерно-Плазменных Процессах и Передовых Научных Технологиях». Москва, 4-9 июля 2016.
5. Научная конференция «Ломоносовские чтения – 2017». Москва, МГУ им. М. В. Ломоносова, апрель 2017.

Основные результаты по теме диссертации изложены в 10 печатных изданиях [2—5; 30; 33–36; 49]: 2 публикации в журнале, рекомендованном

ВАК [3; 4], 1 свидетельство на программу для ЭВМ [30], 1 монография [33], 4 публикации в научных журналах [2; 34—36], 2 публикации в тезисах докладов [5; 49]. В работах [2; 49] вклад автора состоит в построении модели гибридной вычислительной системы, оценке производительности и реализации алгоритмов из NAS Parallel Benchmarks. Вклад автора в работах [3—5] состоит в оценке производительности алгоритмов на КОМДИВ при помощи разработанного метода, реализации оптимизированных процедур БПФ и MG, сравнительном анализе результатов тестирования и предложениях по дальнейшей оптимизации архитектуры гибридных процессоров НИИСИ РАН.

Личный вклад. Все представленные в диссертации результаты получены лично автором.

Объем и структура диссертации. Диссертация состоит из введения, четырёх глав, заключения и двух приложений. Полный объём диссертации составляет 175 страниц с 19 рисунками и 24 таблицами. Список литературы содержит 158 наименований.

Глава 1. Производительность процедур трёхмерного моделирования на гибридных системах

1.1 Современные высокопроизводительные вычислительные системы общего назначения и специализированные вычислительные системы разработки НИИСИ РАН

Развитие архитектур высокопроизводительных систем находит отражение в списке Top500 [148]. Это список самых производительных суперкомпьютеров в мире, который выходит дважды в год, с 1993 года. Список составляется при участии представителей мирового научного сообщества, экспертов в области ВПВ и производителей оборудования. Компьютеры в списке упорядочены по убыванию производительности на известном тесте LINPACK [66].

Согласно данным Top500, симметричные многопроцессорные системы ушли в прошлое более 10 лет назад. На смену им пришли сначала системы массивно-параллельной архитектуры, а следом, к 2006 году, резко возросла доля кластерных систем. С тех пор соотношение количества систем этих архитектур в списке остаётся примерно постоянным: 85% кластерных машин, 15% массивно-параллельных.

В большинстве кластерных машин списка вычислительные узлы соединяются между собой через сеть Ethernet. Массивно-параллельные системы отличаются тем, что узлы в них более тесно связаны, через интерфейс InfiniBand. С течением времени сменяются поколения этих соединений — в настоящий момент широко распространены 10G Ethernet и InfiniBand FDR.

Если в прежние годы большинство суперкомпьютеров были предназначенными для решения задач из той или иной конкретной области, то сейчас практически все системы являются универсальными. Резкий рост доли универсальных систем начался в 2012 году, что отражено на графике 1.1 (приводятся данные выпусков Top500 за июнь).

Это явление связано с тем, что к настоящему моменту выработались общие принципы, по которым строятся современные суперкомпьютеры. Можно заметить, что на протяжении последних 10 лет в список включалось всё больше гибридных систем, объединяющих процессоры различной архитектуры —

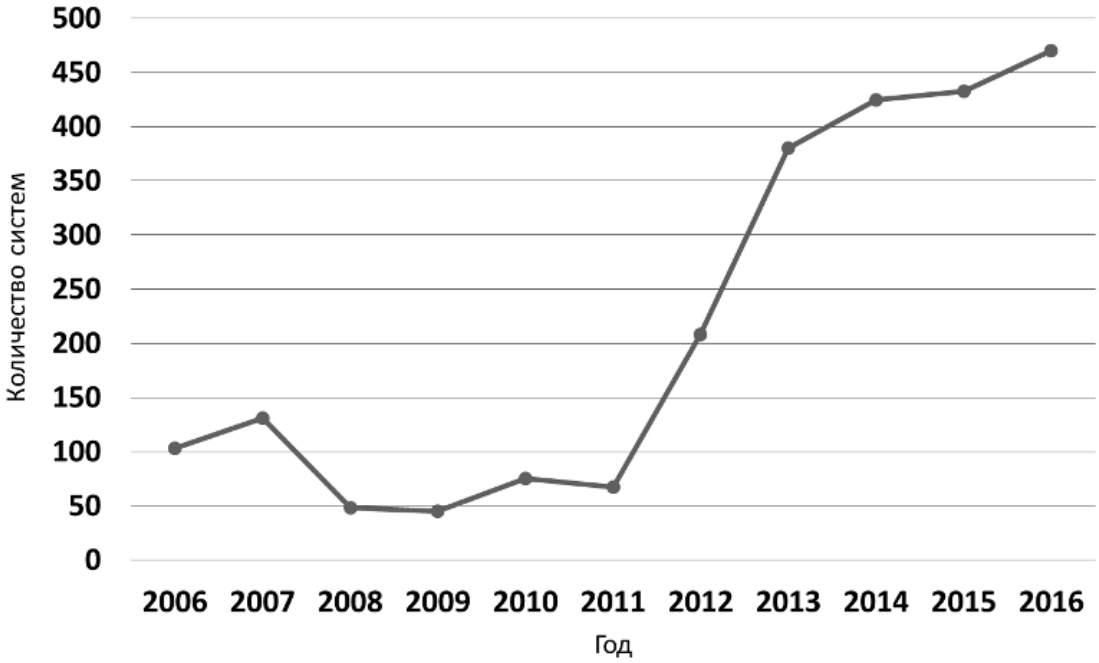

Рисунок 1.1 — Количество универсальных систем в списке Топ500

эта тенденция отражена на графике на рисунке 1.2. В таких гибридных системах каждый узел имеет универсальный процессор, который выполняет функции управляющего (CPU), и один или несколько сопроцессоров. В качестве сопроцессоров чаще всего используются графические ускорители (GPU) производства компаний NVIDIA или AMD, либо ускорители Intel Xeon Phi [92]. К июню 2016 года доля гибридных систем в Топ500 составила 19%.

В качестве примеров гибридных систем можно назвать суперкомпьютеры К100 [38], развёрнутый в ИПМ РАН им. М. В. Келдыша, и Ломоносов-2 [12] НИВЦ МГУ. Комплектующие для таких вычислительных систем представлены на рынке — соответственно, и коммуникационные системы выбираются из числа стандартных, поддерживаемых оборудованием. По тому же принципу могут быть построены и меньшие вычислительные системы, которые будут отличаться от современного большого суперкомпьютера скорее количественно, чем качественно. Построение таких «мини-суперкомпьютеров» для собственных нужд является альтернативой совместному использованию больших машин, с разделением по времени, и становится всё более актуальным, поскольку в большинстве областей применения — и в научных, и в коммерческих расчётах — сложность алгоритмов, объёмы данных, а зачит и требования к производительности вычислений, стремительно возрастают.

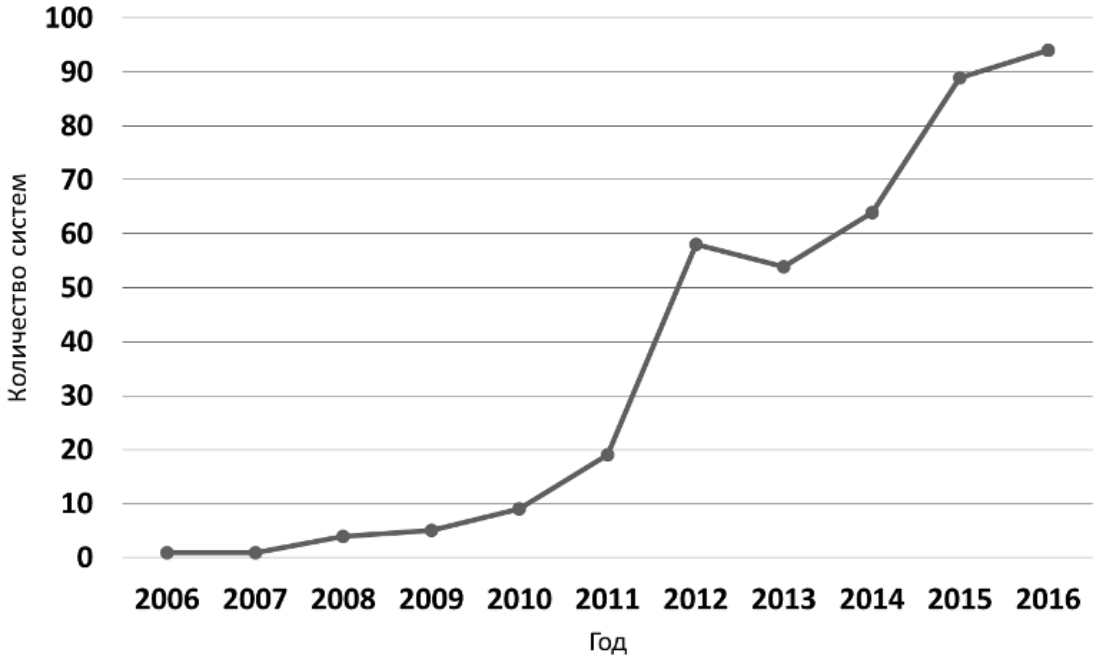

Рисунок 1.2 — Количество гибридных систем в списке Top500

Таким образом, грань между суперкомпьютерами и рабочими станциями в последние годы стирается. Из этого в частности следует, что при проектировании суперкомпьютера различные его характеристики и параметры, а также способы реализации тех или иных алгоритмов, можно исследовать и апробировать на системах меньших масштабов.

Отдельно необходимо отметить, что «чистая» производительность вычислений не может рассматриваться в отрыве от других характеристик системы, в частности, от её энергопотребления. В связи с этим, с 2007 года в дополнение к списку Top500 выпускается также список Green500 [137], частично доступный и на сайте Top500. В этом списке компьютеры упорядочены по убыванию энергоэффективности, то есть МОП/с на Вт мощности. В июне 2013 года среди верхних 10 в списке 4 системы были гибридными, остальные 6 — классическими системами архитектуры IBM BlueGene/Q. В более поздних редакциях все 10 первых позиций стабильно занимали гибридные системы — единственным исключением в июне 2016 года стал Sunway TaihuLight, построенный в Китае, который занимает третье место в Green500 и первое в самом Top500. Можно сделать вывод, что современная высокопроизводительная ВС с хорошей энергоэффективностью — это как правило система гибридной архитектуры.

Одним из первых представителей поколения гибридных параллельных систем является процессор STI Cell [94], включающий одно управляюще ядро

(PPE) и 8 сопроцессоров (SPE). Ядро SPE имеет RISC-архитектуру и поддерживает специализированные SIMD-команды. Процессор изначально разрабатывался для игровых консолей и, соответственно, должен был обеспечивать высокую производительность на мультимедийных приложениях, однако его архитектура позволила эффективно использовать процессор и для вычислений общего назначения.

В основу архитектуры были положены принципы, описанные позднее в [87]: бóльшая энергоэффективность за счёт отказа от архитектурных блоков, которые не могут быть явно использованы для повышения производительности. Это подчёркивается и в [143]: высокой энергоэффективности процессора удалось достичь за счёт избавления от сложной аппаратной верхней логики, такой как переупорядочение инструкций и предсказание переходов. Ставка в архитектуре делается на высокую производительность, за счёт многоядерности, сопроцессоров параллельной архитектуры и канала прямого доступа от сопроцессоров к системной памяти (DMA) с высокой пропускной способностью. В качестве системной памяти используется XDR, которая характеризуется низкой задержкой и высокой пропускной способностью. Помимо этого, каждый сопроцессор обладает собственной локальной памятью с высокой скоростью доступа — её можно рассматривать как управляемый программно аналог кэша, которого на SPE нет. В совокупности эти особенности обеспечивают потенциально высокую производительность вычислений, но для её достижения требуется выполнять множество архитектурно-зависимых оптимизаций вручную: управление локальной памятью SPE, пересылками данных по DMA с учётом требований к выравниванию, и др..

Примером использования Cell в ВПВ является суперкомпьютер IBM RoadRunner [37], который первым преодолел рубеж в 1 ПетаОП/с производительности на тесте LINPACK и занимал первое место в Топ500 в 2008-2009 годах. Отметим, что суперкомпьютер классической архитектуры Jaguar, который вышел на первое место в ноябре 2009 года, обладал производительностью в 1,7 раз выше (1,759 ПетаОП/с против 1,042 ПетаОП/с), но энергопотреблением в 3 раза выше (6950 кВт против 2345 кВт).

Графические ускорители по количественным характеристикам во много раз превосходят процессор Cell, однако качественно их архитектуры имеют много общего. GPU, как и следует из названия, разрабатывались для работы с гра-

фикой, с высокой производительностью и энергоэффективностью, поэтому изначально их архитектура также была упрощена относительно CPU: множество SIMD-ядер, рассчитаных на однотипную потоковую обработку больших массивов данных, с низкой повторной используемостью, и широкий канал доступа к памяти. Процессорные ядра группируются в т.н. вычислители: ядра на одном вычислителе одновременно исполняют один поток инструкций, но над разными данными. Память имеет иерархическую организацию и управляется программно: глобальная память, общая для всего ускорителя, локальная память на каждом вычислителе, общая для его ядер, и регистровый файл на вычислителе, в котором каждое ядро использует выделенную область. Таким образом, один GPU представляет собой массивно-параллельную систему (в роли узлов с локальной памятью выступают вычислители), которая обладает, однако, и общей памятью. В современных GPU в качестве глобальной используется память семейства GDDR с высокой пропускной способностью, за счёт частоты и ширины шины доступа к ней — например, на ускорителе NVIDIA GeForce GTX TITAN используется 384-битная шина, и пиковая пропускная способность составляет 336 ГБ/с.

Первые GPU поддерживали только целочисленную арифметику, которой было достаточно для обработки видео и изображений. Внутренняя кэш-память ускорителя была предназначена только для чтения данных из памяти текстур. Однако в дальнейшем поддержка арифметики с плавающей точкой и другие усовершенствования, а также появление специальных инструментов разработки, обеспечили возможность использования этих ускорителей и в расчётных приложениях ([73]). Первой процедурой, для которой были предприняты попытки перенесения с CPU на GPU, стало матричное умножение ([70; 100]). При программировании под GPU приходилось сталкиваться с такими трудностями, как представление данных в формате текстуры, для использования кэш-памяти, и эмуляция вычислений с двойной точностью через вычисления с одинарной (см. [58; 135]). Развитие архитектуры самих графических ускорителей, и одновременно совершенствование средств разработки, позволило добиться на GPU более высокой производительности вычислений, чем на CPU ([104]). С тех пор на GPU было перенесено большое количество приложений; появились целые библиотеки оптимизированных на GPU процедур. В числе факторов, обусловивших рост популярности вычислений на GPU — низкая стоимость и

энергопотребление, в пересчёте на 1 МОП/с пиковой производительности. В настоящее время выпускаются специализированные модели GPU для высокопроизводительных вычислений — например, серии NVIDIA Tesla и AMD FirePro. Усложнение архитектуры повлекло, однако, и рост энергопотребления ([108]): у наиболее мощных из современных GPU оно в несколько раз превышает энергопотребление CPU, хотя, поскольку для многих приложений разница в производительности ещё больше, энергоэффективность GPU на этих приложениях остаётся более высокой.

В отличие от Cell, GPU не включает в себя управляющий процессор. Типичный гибридный узел с GPU состоит из CPU (одного или нескольких) и от 1 до 8 GPU, подключённых через шину PCI Express. Отсюда возникает одна из основных проблем, связанных с вычислениями на GPU: необходимость пересылок данных между системной памятью и глобальной памятью GPU; накладные расходы на эти пересылки в некоторых случаях могут нивелировать эффект от высокой производительности вычислений на самом GPU. Эта проблема исследуется подробно далее в работе.

Перспективным направлением развития является разработка систем, которые будут включать CPU и GPU на одном кристалле, с общей памятью. Такое решение позволит, во-первых, избавиться от «бутылочного горлышка» PCI Express, а во-вторых, повысить энергоэффективность. В настоящий момент разработка такого модуля для ВПВ — APU [71] — ведётся компанией AMD. В APU планируется использовать память, построенную по новой технологии HBM [80], которая обеспечит на порядок более высокую скорость доступа к данным.

Вычислительный кластер представляет собой набор узлов – например, гибридных с GPU, — объединённых через ту или иную коммуникационную среду. Скорость обменов данными по ней — один из ключевых факторов для производительности всей вычислительной системы. Наибольшей популярностью традиционно пользуются Ethernet и InfiniBand [93], однако в некоторых из наиболее мощных суперкомпьютеров используются специализированные среды — например, TH Express [142] в суперкомпьютере Tianhe-2 [144] и Aries Interconnect от Cray [56], на базе которого построен целый ряд суперкомпьютеров из последней редакции списка Top500.

Примером систем с собственной коммуникационной средой, а также нестандартной топологией соединения узлов, являются и кластеры

SiCortex [114], которые производились с 2003 по 2009 годы. Эти системы интересны ещё и тем, что каждый узел в них представляет собой однокристальную систему, включающую несколько процессорных ядер RISC-архитектуры MIPS64. Благодаря высокой энергоэффективности и относительно малому размеру (что в частности позволяет выделить больше площади кристалла под память), RISC-процессоры имеют потенциал и для использования в ВПВ. Например, ранее на основе процессоров архитектуры MIPS была построена серия рабочих станций компании Silicon Graphics. Архитектура MIPS имеет целый ряд расширений для параллельных вычислений: SIMD, поддержка многопоточности, векторное расширение MDMX для работы с видео и аудио данными. Имеются и другие RISC-архитектуры с перспективой использования в ВПВ — например, ARMv8-A [43] и открытая архитектура набора команд RISC-V [121].

Ещё одна известная коммуникационная технология — RapidIO, которая также объединяет устройства в сеть ([74], см. также [21] и [23]) и основана на коммутации пакетов. RapidIO обладает высокой энергоэффективностью и большой гибкостью: на его основе могут быть построены как малые (несколько микросхем на печатной плате, несколько таких плат в блоке), так и очень большие вычислительные системы (комплекс из нескольких блоков), включающие процессоры разных типов от различных производителей. Если речь идёт о SoC-системе, RapidIO, как и PCI Express, тесно интегрирован с подсистемой памяти, что позволяет добиться более высокой пропускной способности и более низких задержек, чем на InfiniBand или Ethernet. Простой механизм коммутации пакетов позволяет реализовать любую топологию соединения. Поддерживаются различные операции, включая как пересылки данных из памяти в память без вмешательства процессора, так и обмен сообщениями между процессорами.

Технология RapidIO многие годы успешно применяется в высокопроизводительных встроенных системах, в особенности, благодаря низким задержкам, в системах реального времени, а так же в системах, включающих в себя FPGA. Например, на технологии RapidIO основаны коммерческие и оборонные вычислительные комплексы производства компании Mercury Systems. Кроме того, RapidIO применяется в беспроводных сетях. В последнее время возрастает интерес к RapidIO и в области ВПВ [120].

Отдельная ветвь развития вычислительных систем, не связанная напрямую с ВПВ — системы и процессоры цифровой обработки сигналов. К ЦОС-

процессорам предъявляются специфические требования, по сравнению с универсальными процессорами, в том числе ограничения по потребляемой мощности и задержке вычислений. Такие процессоры предназначены для потоковой обработки данных и зачастую имеют канал DMA для доступа к памяти, с возможностью одновременного выбора нескольких значений, а также поддержку таких моделей параллелизма, как SIMD и/или VLIW. Часто ЦОС-процессоры имеют специализированные аппаратные функции для обработки сигналов — главным образом, вычисления БПФ — которые могут быть вынесены в отдельный сопроцессор. Вместе с тем, в ЦОС-процессоре могут отсутствовать возможности, присущие универсальным процессорам — например, поддержка вычислений с двойной точностью.

Пример ЦОС-процессора — TigerSHARC от Analog Devices [145]. Процессор имеет 2 ядра, которые могут работать как в режиме SIMD, так и по отдельности, и DMA для доступа к внешней памяти. Поддерживает вещественные числа с фиксированной точкой, а также с плавающей точкой одинарной (32 бит) и расширенной (40 бит) точости. Система команд включает VLIW-инструкции для работы со 128-битными векторами. Другой пример — TMS320C6678 от Texas Instruments. Он имеет 8 ядер и поддерживает вещественные числа двойной точности, система команд включает VLIW-инструкции для работы с 256-битными векторами. Кроме того, процессор имеет внешний интерфейс RapidIO.

Что касается отечественных разработок, микропроцессор Эльбрус-2С+ [8] представляет собой гибридную систему на кристалле, в которой сочетаются два управляющих ядра архитектуры Эльбрус и четыре ЦОС-ядра разработки АО НПЦ «Элвис». Как отмечается в [11], использование сопроцессоров в ВС оказывается эффективным, когда аппаратура и решаемая задача приспособлены друг к другу, однако в общем случае сопряжено с рядом трудностей программирования, в том числе с необходимостью явного управления обменами данными — это можно наблюдать и на примере гибридных ВС с GPU. Последние разработки на архитектуре Эльбрус — сервер Эльбрус-4.4 и 8-ядерный процессор Эльбрус-8С [19; 31] — представляют собой системы классической архитектуры на основе многоядерных процессоров, поэтому обладают большей универсальностью.

Процессоры КОМДИВ — это специализированные ЦОС-процессоры, которые сочетают в себе ряд описанных выше перспективных технологий. В данной работе будут подробно рассмотрены два процессора этой линейки: ВМ7 и ВМ9.

Процессор ВМ7 (микросхема 1890ВМ7Я [17; 25]) разрабатывался приблизительно в те же годы, что и STI Cell, и близок к нему архитектурно. Он представляет собой гибридную систему, включающую управляющий процессор RISC-архитектуры КОМДИВ (развитие MIPS64) и сопроцессор СР2. Максимальная частота процессорного ядра — 200 МГц.

СР2 — специализированный 128-разрядный математический сопроцессор SIMD-архитектуры, включающий:

- память инструкций объёмом 8192 64-разрядных инструкции;
- 4 вычислительных секции, позволяющие выполнять вычисления над целыми числами, вещественными числами одинарной точности и комплексными одинарной точности;
- локальную память объёмом 64 Кбайт в каждой вычислительной секции;
- регистровый файл объемом 64 64-разрядных регистра в каждой вычислительной секции;
- набор из 16 64-разрядных регистров общего назначения, один для всех секций;
- блок генерации адресов в локальной памяти, общий для всех секций;
- память коэффициентов для БПФ объёмом 64 Кбайт.

Кроме того, СР2 является VLIW-сопроцессором и имеет 3 физических конвейера:

- для арифметических операций и элементарных функций, глубины 9;
- для обменов данными между локальной памятью и регистрами, глубины 4;
- для обменов между различными регистрами и операций управления, глубины 3.

Все секции работают параллельно, выполняя один и тот же набор инструкций (вычислительное ядро), но над различными данными.

Архитектура СР2 оптимизирована для вычислений в задачах потоковой обработки сигналов. В частности, имеется ПЗУ коффициентов для вычисления БПФ. Система команд СР2 включает специализированную команду «бабочка Фурье», использование которой обеспечивает выполнение до 40 операций

с 32-разрядными вещественными числами за один такт рабочей частоты, что соответствует пиковой производительности 8 ГОП/с (при работе на частоте 200 МГц).

ВМ7 обладает контроллером доступа к системной памяти типа DDR2, объёмом до 2 Гбайт. Максимальная частота шины памяти совпадает с частотой процессорного ядра и составляет 200 МГц.

Обмен данными между глобальной (системной) памятью и локальной памятью СР2 осуществляется через канал прямого доступа к памяти (DMA). Единица пересылки данных — 128-разрядное слово (столько же данных может быть за один такт считано из глобальной памяти, а также из локальной памяти в регистры одной секции СР2). Аппаратно поддерживаются двумерные массивы данных в системной памяти и два режима распределения данных по секциям СР2. Локальная память секций СР2 разделяется на два равных банка, что позволяет совмещать вычисления на СР2 над данными в одном банке памяти и обмены данными по DMA в другом банке.

Обмены данными с другими процессорами в многопроцессорном комплексе осуществляются через параллельный 8-разрядный интерфейс RapidIO.

Процессор изготавливается по проектным нормам КМОП 0,18 мкм.

Перспективный универсальный микропроцессор ВМ9 (микросхема 1890ВМ9Я, документация к управляющему ядру: [18]) программно совместим с ВМ7 и изготавливается по технологии TSMC 65 нм. Первые микросхемы и модули были выпущены в сентябре 2016 года. Проектная частота процессорного ядра составляет 1000 МГц.

Архитектура ВМ9 представляет собой развитие архитектур ВМ7 и ВМ6 (разработанного ранее универсального 64-разрядного микропроцессора линейки КОМДИВ). ВМ9 имеет два процессорных ядра на одном кристалле, соединённых через внутренний коммутатор с программной моделью RapidIO. Каждое ядро имеет собственный сопроцессор СР2, канал DMA и выделенную половину системной памяти.

Ряд усовершенствований внесён как в управляющее ядро, так и в СР2. В частности, память коэффициентов Фурье на СР2 теперь представляет собой не ПЗУ, а ОЗУ — это расширяет возможности использования памяти коэффициентов в различных алгоритмах. Для доступа от каждого управляющего ядра к системной памяти имеется L2-кэш объёмом 512 Кбайт. Новая версия

контроллера DMA позволяет пересылать данные не только между глобальной и локальной памятью, но и непосредственно из одной области глобальной памяти в другую, что потенциально позволяет ускорить процедуры копирования в два раза. Поддерживается системная память типа DDR2 и DDR3, объёмом до 12 Гбайт.

Каждый процессор ВМ9 имеет три последовательных внешних интерфейса RapidIO 1x/4x.

Программно-аппаратная платформа «Багет» разработана в НИИСИ РАН для построения многопроцессорных вычислительных комплексов. Такие комплексы имеют кластерную архитектуру. Основной структурной единицей является модуль, включающий процессоры КОМДИВ, память, периферийные контроллеры и вспомогательные микросхемы. При этом все межпроцессорные и межмодульные коммуникации осуществляются через сеть RapidIO.

1.2 Методы программирования

Одна из ключевых проблем высокопроизводительных ВС — выбор методологии программирования, которая будет учитывать гибридность и иерархическую организацию системы и обеспечит максимальное удобство и переносимость. Для коммуникаций между узлами в кластере или суперкомпьютере стандартом де-факто является MPI [111]. Различие между системами гибридной и классической архитектуры проявляется на уровне одного узла. Узел классической архитектуры представляет собой SMP или NUMA-систему, которая как правило программируется в многопоточной модели, например, при помощи OpenMP. Двухуровневое распараллеливание — через MPI и OpenMP — широко применяется в ВПВ [126; 158].

GPU, в силу архитектурных отличий от CPU, требует разработки специальных оптимизированных процедур. Единого стандарта программирования для GPU в настоящее время не существует. Открытый стандарт OpenCL («Open Computing Language», [140]) представляет собой альтернативу проприетарной платформе CUDA от NVIDIA [24] и реализует аналогичную модель программирования. OpenCL включает язык программирования на базе стандарта C99,

для написания вычислительных процедур для ускорителей, API для разработки управляющих программ, а также ряд библиотек и среду выполнения.

Вычислительная процедура для GPU в стандарте OpenCL подразделяется на две части:

– одно или несколько «вычислительных ядер» на языке OpenCL C – процедур, которые выполняются непосредственно на GPU, с входными и выходными данными в его глобальной памяти;
– управляющая процедура, использующая API OpenCL, которая выполняется на CPU. Эта процедура организует загрузку и выгрузку данных с ускорителей и запуск вычислительных ядер в необходимой последовательности, с установкой зависимостей между этими операциями.

GPU реализует модель параллельных вычислений SIMD, в сочетании с многопоточностью. По стандарту OpenCL, параллельная программа в ходе выполнения разделяется на некоторое количество потоков («work-items», WI). Каждый процессор ускорителя в определённый момент времени может выполнять один поток параллельной программы. Все WI исполняют набор инструкций одной и той же вычислительной программы, но над разными данными. Набор WI подразделяется на рабочие группы («work-groups», WG), все потоки одной группы работают параллельно на одном вычислителе и имеют доступ к его локальной памяти. Внутри одной WG возможна синхронизация между WI. Программист определяет количество WI в одной WG и способ распределения работы между ними, а также расположение данных в глобальной или локальной памяти. От типа используемой в ядре памяти и от того, какова схема доступа к ней от разных WI одной WG, зависит скорость, с которой осуществляется чтение/запись (см., например, [41]).

Процедуры на OpenCL являются переносимыми, однако для достижения максимальной производительности как правило необходимо производить «доводку» кода и подбор оптимальных параметров под каждое конкретное устройство [73; 115; 123]. Впрочем, данная проблема существует не только для OpenCL, но и для других средств программирования, как показано в [112].

Помимо управления сопроцессорами, CPU на гибридном узле может выполнять и часть общей вычислительной работы [107]. Таким образом получается гетерогенная процедура, задействующая все ресурсы узла. Вообще говоря, на CPU также могут исполняться процедуры на OpenCL, но как отмечается

в [128] и [152], имеют низкую эффективность, что объясняется недостаточной поддержкой OpenCL производителями процессоров. OpenMP остаётся более эффективным средством распараллеливания для CPU. Таким образом, на гибридной архитектуре к связке MPI и OpenMP добавляется OpenCL. Поскольку такое програмирование достаточно трудозатратно, предпринимаются попытки автоматизировать этот процесс — например, [152]. В [42] строится модель системы, на основе которой осуществляется автоматическое распределение заданий между сопроцессорами.

Процессоры Эльбрус-4С и Эльбрус-8С представляют собой универсальные процессоры, код для которых может разрабатываться на языках C/C++ и Fortran. Для обеспечения высокой производительности ставка делается на оптимизирующий компилятор, который задействует возможности VLIW-архитектуры процессора и векторизацию. Возможна, однако, и оптимизация вручную, с использованием языка ассемблера.

На процессорах КОМДИВ код для управляющего ядра также разрабатывается на высокоуровневых языках, для которых поддерживаются модификации стандартных компиляторов GCC. Все вычисления с использованием CP2, однако, оптимизируются вручную. Это включает разработку вычислительных ядер для сопроцессора CP2 и организацию обменов данными по DMA между памятью CP2 и системной памятью.

Ядра для CP2 разрабатываются на языке ассемблера [26]. На программиста ложится задача группировки команд в VLIW-инструкции, использования трёх конвейеров и отслеживания зависимостей между командами. Вычисления на CP2 выполняются асинхронно по отношению к работе управляющего процессора: во время выполнения задания на CP2 можно продолжать выполнять те или иные операции на CPU. Управление работой DMA осуществляется через специализированное API: формируется очередь дескрипторов, описывающих операции загрузки или выгрузки. Программисту доступна явная синхронизация с CPU как завершения всех операций в очереди DMA, так и завершения текущего задания на CP2 — это позволяет реализовать совмещение вычислений на CP2 с пересылками данных.

Помимо входных массивов, загружаемых через DMA, в процедуры CP2 могут передаваться дополнительные аргументы, через набор регистров CP2,

доступных для записи со стороны CPU. Для записи и чтения таких регистров предусмотрены специальные процедуры.

Описанные механизмы реализованы в рамках библиотеки цифровой обработки сигналов (БЦОС, [1; 30]) для ВМ7, которая совместима и с ВМ9. Кроме того, реализованы программные эмуляторы сопроцессора СР2 и контроллера DMA. Эмуляторы предоставляют возможность предустанавливать начальное состояние конкретного аппаратного блока (например, значения регистров СР2), а также выводить потактовую информацию о выполненных операциях.

На верхнем уровне, для программирования обменов данными через каналы RapidIO, разработана специализированная библиотека (рассматривается в [27]). Эта библиотека реализует модель программирования, наиболее удобную для потоковых вычислений: вся вычислительная работа подразделяется на стадии. В настоящее время ведутся работы по обеспечению поддержки стандарта MPI для программирования обменов данными через RapidIO ([14]). Здесь каждый процессор (ядро) исполняет один MPI-процесс.

Подводя итог, можно сказать, что программирование гибридных систем с GPU и многопроцессорных комплексов на базе КОМДИВ происходит в схожих программных моделях, с той разницей, что на КОМДИВ используются специализированные средства:

- вычислительные ядра для процессоров, работающие с данными в памяти сопроцессора и использующие параллелизм в его архитектуре: на OpenCL C для GPU и на ассемблере для СР2;
- управляющая процедура, организующая запуск ядер и обмены данными с сопроцессором: через PCI Express на узле с GPU, через DMA на ВМ7/9;
- передача MPI-сообщений между узлами кластера: например, над InfiniBand или Ethernet для гибридных систем с GPU, и над RapidIO для процессоров КОМДИВ.

1.3 Модель гибридной вычислительной системы

Построим модель гибридной вычислительной системы, которая отражает общность архитектурных решений гибридных систем на базе GPU и гибридных процессоров НИИСИ РАН.

Модель описывает системы следующей конфигурации:

– системная память;
– управляющий процессор (CPU);
– один или несколько сопроцессоров одинаковой архитектуры;
– канал для обменов данными между памятью сопроцессоров и системной памятью, разделяемый между всеми сопроцессорами.

Гибридная ВС такой конфигурации характеризуется следующими ключевыми параметрами:

– *HMEM* — объём системной памяти (Host Memory);
– *HPEAK* — пиковая производительность CPU;
– *K* — количество сопроцессоров;
– *DMEM* — объём внутренней памяти одного сопроцессора (Device Memory);
– *DPEAK* — пиковая производительность одного сопроцессора;
– bw_{CP} — пиковая скорость доступа к памяти на сопроцессоре;
– *BW* — пиковая пропускная способность канала обмена данными между системной памятью и сопроцессорами.

Схематичное изображение системы, соответствующей модели, представлено на рисунке 1.3.

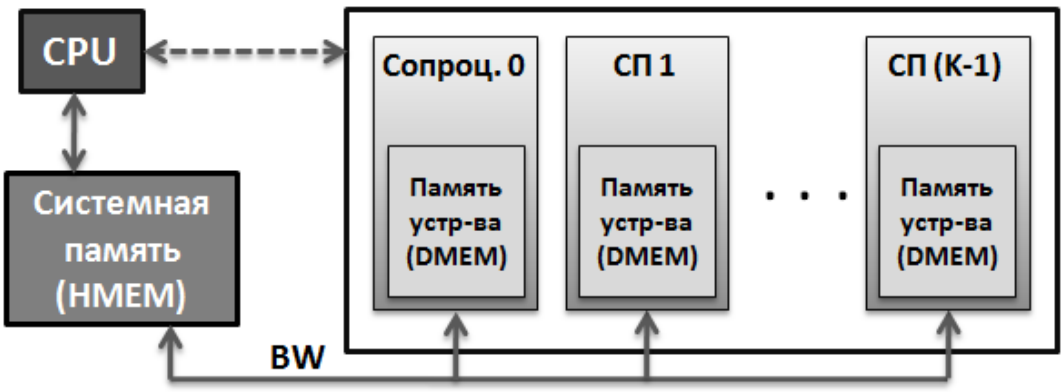

Рисунок 1.3 — Модель гибридной вычислительной системы

Система поддерживает выполнение операций четырёх типов:

– вычислительные «ядра» — вычислительные процедуры для сопроцессоров; все сопроцессоры параллельно исполняют одно и то же ядро, каждый — над данными в собственной памяти;
– загрузки данных из системной памяти в память сопроцессоров, по разделяемому каналу;
– выгрузки данных из памяти сопроцессоров в системную память, по разделяемому каналу;
– вычислительные процедуры на управляющем процессоре, над данными в системной памяти.

Как следствие, обмен данными между сопроцессорами возможен только через системную память.

При этом предполагается следующее:

– для каждой процедуры управляющий процессор устанавливает зависимости между операциями четырёх типов и запускает их на исполнение в необходимой последовательности;
– поддерживается параллельное исполнение вычислительного ядра на сопроцессорах и операции пересылки данных (загрузки или выгрузки), при условии, что ядро и операция пересылки обращаются к непересекающимся областям памяти каждого сопроцессора.

Каждой процедуре, которая выполняется на такой гибридной системе и задействует вычисления на сопроцессорах, соответствует собственный набор вычислительных ядер. Каждое ядро, в свою очередь, характеризуется параметром $Perf_{kern}$ — производительностью на одном сопроцессоре.

Декомпозиция вычислительной задачи в виде набора ядер не всегда однозначна и должна выполняться на этапе построения схемы вычислений. В качестве значения $Perf_{kern}$ может быть использована реальная производительность ядра — уже имеющегося или вновь разработанного. Можно оценить $Perf_{kern}$ и теоретически, определив базовую инструкцию в основном цикле программы и количество итераций — такая оценка будет специфичной для каждого сопроцессора, т.к. зависит от его архитектуры и набора инструкций. Оценки такого рода использовались, например, в [54]. В рамках данной работы такие оценки для GPU проводиться не будут. В грубом приближении можно считать, что $Perf_{kern}$,

в зависимости от процедуры, определяется либо пиковой производительностью сопроцессора ($DPEAK$), либо скоростью доступа к его памяти (bw_{CP}).

Построенная модель предназначена для теоретической оценки ожидаемой производительности заданной вычислительной процедуры на гибридной ВС. Входными параметрами при использовании модели являются:

– вычислительная процедура;

– набор значений параметров модели.

Данная модель охватывает гибридные узлы на базе GPU. Память сопроцессора в модели соответствует глобальной памяти ускорителя; параметр bw_{CP} характеризует скорость доступа к ней. В таблице A.1 приводятся значения параметров построенной модели для двух примеров вычислительных систем: один из стендовых узлов НИИСИ РАН и один из 64 узлов суперкомпьютера K100.

Процессор Cell не будет подробно рассматриваться в данной работе, однако построенная модель описывает и его архитектуру: управляющий процессор PPE и 8 сопроцессоров SPE.

В рамках модели может быть рассмотрен и один процессор BM7 с сопроцессором CP2, а также одно ядро BM9. Упрощённая структурная схема BM7 приводится на рисунке 1.4 (K64 — CPU архитектуры КОМДИВ64, LMEM — локальная память CP2).

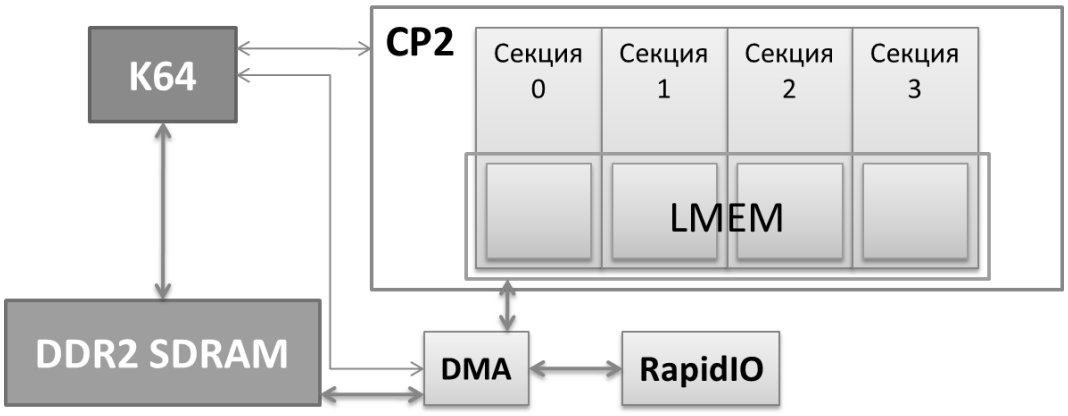

Рисунок 1.4 — Упрощённая схема BM7

Нетрудно увидеть аналогию между данной схемой и обобщённой схемой гибридной системы на рисунке 1.3. Сопроцессором, в терминах модели, является одна вычислительная секция CP2: каждая секция обладает собственной локальной памятью, а также собственным набором регистров FPU для выполнения арифметических операций над вещественными числами. Канал DMA яв-

ляется разделяемым для всех секций. Аналогично может рассматриваться и одно процессорное ядро ВМ9 с соответствующим сопроцессором СР2.

Процессоры ВМ7/9 поставляются в виде модулей, на которых могут различаться типы и объёмы памяти, частоты памяти и CPU (СР2 работает на частоте ядра). На существующих модулях ВМ7 доступно по 768 Мбайт памяти типа DDR2 на процессор, рабочая частота CPU и частота памяти совпадают — 200 МГц. Все замеры производительности на ВМ7, представленные в данной работе, производились на этих модулях. Что касается ВМ9, имеются модули с различными частотами CPU и памяти. В данном разделе для определённости рассматриваются модули с памятью DDR3 667 МГц и частотой CPU 1000 МГц. На каждое ядро доступно по 2 Гбайт памяти. Если не оговорено иное, далее в работе подразумеваются эти характеристики, однако в ряде случаев они будут уточняться, т.к. замеры проводились в разное время на различных аппаратных стендах.

Пиковая пропускная способность памяти DDR2 — 3,2 ГБ/с, однако по результатам замеров контроллер обеспечивает пропускную способность до 2,7 ГБ/с. Контроллер DMA может передавать 16 байт за один такт, что обеспечивает теоретическую скорость на частоте 200 МГц 3,2 ГБ/с, однако реальная скорость ограничена также пропускной способностью DDR2 — т.о., пиковую пропускную способность канала DMA можно считать равной 2,7 ГБ/с. На ВМ9 теоретический пик пропускной способности DDR3 на частоте 667 МГц — 10,7 ГБ/с, максимальная скорость на реальном модуле по результатам замеров на той же частоте — 8,4 ГБ/с.

Объём локальной памяти одной секции СР2 на обоих процессорах составляет 64 Кбайт. CPU поддерживает вещественные числа как одинарной, так и двойной точности, однако в текущей версии СР2 возможны только вычисления с одинарной точностью. Пиковая производительность СР2 зависит от арифметической инструкции. Наиболее производительной является инструкция «бабочка Фурье», которая включает 10 арифметических операций над вещественными числами с плавающей точкой. Поскольку вычисления на СР2 конвейеризованы, инструкция может быть запущена на исполнение на каждом такте работы процессора. Отсюда, пиковая производительность одной секции на «бабочке Фурье» на ВМ7 составляет 2 ГОП/с. Соответственно, на ВМ9, на частоте 1000 МГц, пиковая производительность одной секции составляет 10 ГОП/с.

Параметр bw_{CP} соответствует скорости чтения и записи данных из локальной памяти CP2 в регистры FPU и обратно. Имеются инструкции чтения и записи двойного слова (128 бит), которые могут быть запущены на каждом такте. Таким образом, bw_{CP} на BM7 составляет 3,2 ГБ/с. На BM9 эта величина составляет 16 ГБ/с.

Параметры модели для микропроцессора BM7 и одного ядра BM9 приводятся в таблице A.2.

Сравнивая эти значения с таблицей A.1, можно увидеть, что и на BM7, и на BM9 производительность и пропускная способность каналов передачи данных более сбалансированы, чем в гибридной системе на основе GPU. Так, объём данных, считанных из локальной памяти за время выполнения одной операции с плавающей точкой, на CP2 выше, чем на современном GPU. Кроме того, скорость обменов данными между системной памятью и памятью сопроцессора и скорость обменов между памятью сопроцессора и регистрами — на CP2 величины одного порядка (хотя на BM9 разница больше, чем на BM7).

Построенная модель может быть расширена, чтобы описывать не только один узел, но и кластеры из нескольких узлов. Для этого к введённым выше параметрам необходимо добавить количество узлов, пропускную способность коммуникационных каналов, а также учесть топологию соединения узлов — фактор, существенно влияющий на скорость обменов данными. Т.о., хотя в данной работе основное внимание уделяется одному узлу, аналогичный подход к исследованию алгоритмов может быть применён к системам бóльших масштабов.

В НИИСИ, помимо описанного выше гибридного узла, имеется гибридный кластер из четырёх узлов, на каждом узле — CPU и два GPU, описанных в таблице A.1. Узлы соединяются по InfiniBand через коммутатор, пропускная способность каждого канала — 6 ГБ/с.

Аналогично можно рассматривать и многопроцессорные комплексы на базе гибридных процессоров НИИСИ РАН. BM7 имеет параллельный 8-разрядный интерфейс RapidIO, с пиковой пропускной способностью 500 МБ/с в каждом из двух направлений. При этом аппаратно поддерживается совмещение пересылок в двух направлениях и вычислений на процессоре. BM9 имеет три последовательных интерфейса RapidIO 1x/4x с пиковой скоростью 3,125 Гбод и пропускной способностью 1,25 ГБ/с в каждом направлении. Пиковая скорость

коммуникаций между ядрами одного процессора, через внутренний коммутатор, вдвое выше, чем скорость внешних интерфейсов: по 2,5 ГБ/с в каждом направлении.

1.4 Целевые вычислительные процедуры

Для рассмотрения в данной работе были выбраны три процедуры (алгоритма), входящие в состав пакета NAS Parallel Benchmarks (Numerical Aerodynamics Simulation Parallel Benchmarks, или NPB), [139]. Это набор тестов, разработанный в центре NASA для оценки производительности параллельных суперкомпьютеров. Тесты представляют собой некоторые упрощённые базовые алгоритмы, на основе задач вычислительной гидродинамики, которые позволяют оценить характеристики ВС, существенные при решении реальных задач. Алгоритмы из NPB используются для оценки возможностей архитектуры и программных средств многими исследователями (см., например, [28; 112; 116; 123]).

Все алгоритмы из NPB имеют так называемую «бумажную» спецификацию: они полностью описаны в тексте [138], и программисту предлагается реализовать их для конкретной системы любым удобным способом. Накладывается ряд ограничений на используемые средства: например, не разрешается использовать язык ассемблера и подключать нестандартные библиотеки. В тексте также задаются входные данные и эталонные результаты, для проверки корректности реализации алгоритмов. Все вычисления вещественной арифметики требуется производить с двойной точностью.

Классический состав NPB — это 5 вычислительных «ядер» (не следует путать их с ядрами как процедурами на сопроцессоре):

- IS — целочисленная сортировка;
- EP — «чрезвычайная параллельность»,
- CG — метод сопряжённых градиентов,
- MG — многосеточный метод,
- FT — дискретное преобразование Фурье;

— и 3 «псевдоприложения» — численных решения системы пяти нелинейных уравнений в частных производных с использованием:

– BT — блочного три-диагонального решателя;
– SP — скалярного пяти-диагонального решателя,
– LU — LU-решателя Гаусса-Зейделя.

В диссертации рассматриваются алгоритмы FT, MG и CG, которые широко применяются при решении задач трёхмерного моделирования.

Первый алгоритм — FT (Fourier Transform) — быстрое преобразование Фурье (БПФ). В [67] предлагается список из 10 алгоритмов, оказавших, по мнению авторов, наибольшее влияние на развитие науки и промышленности в XX веке. К их числу относится и БПФ. Этот алгоритм является ключевым в ЦОС, однако используется и во многих других областях, в том числе и в моделировании различных процессов ([52; 101]).

Алгоритм NPB FT сводится к трёхмерному комплексному БПФ и ориентирован на тестирование эффективности вычисления БПФ небольших векторов на узле, а также межузловых обменов большими объёмами данных. Трёхмерное БПФ по своей сути и по схеме программной реализации аналогично двумерному БПФ, а также «длинному» одномерному БПФ, вычисляемому по алгоритму «БПФ за четыре шага» («4-step FFT», [150, p.139-141]). Их производительность определяется одними и теми же характеристиками системы. В главе 2 будет рассматриваться реализация алгоритма FT на GPU, а в главе 3 — реализация длинного БПФ на процессорах КОМДИВ, т.к. именно эта процедура потребовалась и вошла в состав БЦОС.

Второй алгоритм — MG (MultiGrid) — многосеточный метод. Многосеточные методы — класс алгоритмов, который подразделяется на геометрические и алгебраические многосеточные методы. В сложных практических задачах многосеточные методы могут использоваться как предобуславливатели для других методов [156]. Область применения многосеточных методов в настоящее время очень широка: трёхмерное моделирование в медицине, моделирование эффективных двигателей внутреннего сгорания, исследование магнитных полей, компьютерная графика, и так далее (см., например, [16; 77; 97]).

Алгоритм NPB MG — классический геометрический многосеточный метод, полностью описанный в [138]. Он ориентирован на тестирование регулярных обменов данными между процессорами. Формулировка задачи: найти при-

ближённое решение уравнения Пуассона

$$\bigtriangledown^2 u = v \tag{1.1}$$

на сетке $N \times N \times N$, с периодическими граничными условиями.

Все вычисления выполняются над вещественными числами. На каждой итерации алгоритма поправка к решению вычисляется многосеточным методом с использованием V-цикла, т.е. осуществляется коррекция приближённого решения на наборе сеток от самой мелкой ($N \times N \times N$) до самой грубой ($2 \times 2 \times 2$): производится последовательность проекций, на всё более и более грубые сетки; на самой грубой сетке вычисляется поправка к приближению, после чего эта поправка последовательно интерполируется на более мелкие сетки. Т.о., алгоритм сводится к последовательности линейных разностных операторов (РО) на трёхмерной сетке, каждый из которых применяется к результату предыдущего. Используются 4 различных оператора: невязка, проекция, интерполяция и сглаживание.

Данный алгоритм представляет собой частный случай трафаретных вычислений («stencil computations» в англоязычной литературе). Трафаретные вычисления можно кратко описать следующим образом: имеется пространство точек, и новое значение в каждой точке определяется как функция от старых значений в соседних точках. При этом задаётся «трафарет», по которому определяются соседние точки, одинаковый для всех точек. На каждом шаге вычисления для различных точек независимы. К трафаретным вычислениям сводятся все задачи, решаемые конечно-разностным способом, методом Якоби или многосеточным. Трафаретные вычисления используются в самых разных алгоритмах и входят во многие наборы тестов производительности — например, PARKBENCH [65] и HPFBench [83]. Они находят применение в моделировании землетрясений, различных молекулярных процессов, задачах вычислительной механики, обработке изображений и других областях.

Характерная черта программных реализаций таких вычислений — необходимость чтения на каждое выходное значение нескольких входных, которые могут быть соседними в многомерном массиве, но в памяти располагаться по далеко отстоящим друг от друга адресам и не выравненно. Повторная используемость считанных данных как правило невелика.

Большинство параллельных реализаций вычислений такого рода имеет схожие черты. Пространство ячеек разбивается на области, каждая из которых обрабатывается на одном вычислителе — в зависимости от архитектуры ВС, это может быть процессорное ядро, процессор, вычислительный узел и т.д.. Для вычисления новых значений в ячейках на границе области требуются также некоторые старые значения из соседних областей (дополнительные, или «теневые» значения, в англоязычной литературе называемые также «halo») — какие именно, зависит от трафарета. Перед началом следующей итерации вычислений эти дополнительные значения необходимо обновить — либо путём явных обменов между вычислителями, либо через разделяемую память. Эффективная реализация таких обменов на примере MG будет обсуждаться далее в работе.

Третий алгоритм — CG — метод сопряжённых градиентов. Это широко используемый метод решения СЛАУ, который представляет собой применение метода бисопряжённых градиентов к симметричной матрице. Часто на практике применяется метод сопряжённых градиентов с предобуславливанием многосеточным методом (например, [97]).

В NPB рассматривается метод CG для разреженных матриц. Формулировка задачи: дана симметричная положительно определённая разреженная матрица $\tilde{A}$ порядка $N \times N$; найти её наибольшее собственное значение. Алгоритм также описан в документации NPB, в нём наиболее затратной с точки зрения объёма вычислений операцией является умножение разреженной матрицы на вектор (далее в работе эта операция обозначается «SpMV»). Т.о., алгоритм тестирует нерегулярный доступ к памяти. Все вычисления выполняются с вещественными числами.

Процедура SpMV представляет самостоятельный интерес, не только в составе алгоритма CG. Разреженные матрицы встречаются в широком спектре научных и инженерных задач из многих практических областей: механика деформируемого твёрдого тела, электродинамика и газовая динамика, экономические и проектно-технологические задачи, базы данных, генетика, социология, и другие [61].

SpMV относится к категории процедур, производительность которых определяется скоростью доступа к памяти. Множество исследований посвящено эффективной реализации SpMV на различных архитектурах [113; 122; 158], в том числе гибридных [45; 68; 90; 109; 129; 132]. Производительность проце-

дуры SpMV зависит в первую очередь от двух факторов: от выбранного формата хранения разреженной матрицы и от структуры самой входной матрицы. Процедура генерации из NPB даёт на выходе матрицу в формате CSR (см., напр., [113]), однако допускается перед началом вычислений переупаковать её в любой другой формат.

В NPB вводится понятие классов задач — от S до F (последнего на настоящий момент). Различные классы соответствуют различным объёмам входных данных, количествам итераций внутренних циклов в алгоритмах и т.п. — соответственно, рассчитаны на ВС разных масштабов. В таблице A.3 приводятся параметры задач FT, MG и CG различных классов.

Имеются готовые референсные реализации алгоритмов, доступные на сайте NPB, последняя редакция на момент написания данного текста — NPB 3.3.1. Представлены различные версии кодов, в том числе версии кодов FT, MG и CG на языке Fortran: последовательная версия и параллельные с использованием моделей программирования MPI и OpenMP.

Кратко остановимся на других алгоритмах из состава NPB. Алгоритм EP представляет собой вычисление методом Монте-Карло с малым количеством коммуникаций. Без дополнительного исследования можно утверждать, что его реализация на массивно-параллельных сопроцессорах покажет очень высокую эффективность, как и следует из названия «чрезвычайная параллельность» (см., например, [28]). IS — алгоритм целочисленной сортировки, включающий большое количество условных переходов, которые не поддерживаются на CP2. BT, LU и SP — более сложные алгоритмы вычислительной гидродинамики, которые не сводятся к одной базовой операции и поэтому менее наглядны и удобны для теоретических оценок.

1.5 Метод оценки ожидаемой производительности

Под оптимизацией алгоритма под определённую ВС в данной работе подразумевается достижение как можно более высокой производительности вычислений, т.е. количества арифметических операций в единицу времени, в независимости от затрат труда на программирование. Производительность зависит от

архитектуры ВС, в частности, различных её характеристик: количества процессорных ядер, пропускной способности памяти и т.д., — а также от выбранного способа реализации конкретной процедуры, иными словами, схемы вычислений.

При проектировании новой ВС понимание зависимостей между характеристиками аппаратуры и производительностью вычислений становится особенно важным. Например, как многократно отмечалось многими исследователями, простое увеличение количества процессоров не означает пропорционального ускорения вычислений: во-первых, такое ускорение ограничено законом Амдала, но даже если алгоритм обладает достаточным параллелизмом, бутылочным горлышком могут стать доступ к памяти или межпроцессорные обмены данными [133].

Крайний случай оптимизации архитектуры под конкретную вычислительную задачу — построение ВС на базе FPGA или ASIC. Однако и в более универсальных системах остаётся свобода выбора: набор инструкций и функциональные возможности процессоров, их количество и взаимосвязи между ними, тип и объём памяти, её иерархическая организация, тип и топология коммуникационной среды, и т.д.. В системе, сбалансированной по отношению к конкретному классу задач, минимизированы простои вычислителей — в ожидании данных или из-за недостаточного параллелизма в самом приложении — но и сами коммуникационные каналы используются с высокой эффективностью.

В данной работе рассматриваются гибридные ВС, архитектура которых соответствует модели, построенной в разделе 1.3. Основное внимание уделяется оптимизации обменов данными между сопроцессорами и системной памятью, а также вычислений на одном сопроцессоре. С этой точки зрения, разработку эффективной вычислительной процедуры можно разделить на несколько этапов:

1. разбить процедуру на более простые подпроцедуры и выделить среди них одну или несколько базовых — требующих больше всего по порядку арифметических операций и занимающих в сумме бóльшую часть времени работы процедуры;
2. построить схему вычислений, т.е. представить процедуру как последовательность вычислительных ядер на сопроцессорах, загрузок и выгрузок данных и вспомогательных операций на CPU — так, чтобы количество пересылок было минимальным;

3. реализовать набор необходимых вычислительных ядер на сопроцессорах;
4. реализовать построенную схему вычислений, с совмещением вычислений и пересылок.

Первые два этапа выполняются до начала фактических работ по программированию. Используя построенную модель ВС и зная схему вычислений, можно вывести предварительную оценку сверху для ожидаемой производительности — в виде формулы, зависящей от параметров модели. В целом, оценка сводится к сравнению времени вычислений в ядре и времени приходящихся на них пересылок. Совмещение вычислений и пересылок — типичный приём оптимизации приложений; в предположении, что в реальной процедуре оно будет полным, общее время работы можно оценить как большее из двух: суммарное время работы ядра или время пересылки всех необходимых данных. Такие оценки на данном этапе достаточно получить для базовых операций, отнеся все прочие вычисления к накладным расходам.

Таким образом, предлагается следующий метод оценки производительности вычислительной процедуры на гибридной ВС, которая соответствует построенной модели:

1. выделить в процедуре наиболее нагруженный вычислениями этап — базовую операцию;
2. представить вычисление базовой операции как последовательность операций трёх типов:
   - загрузка данных из системной памяти на сопроцессоры по разделяемому каналу;
   - запуск вычислительного ядра, параллельно на всех сопроцессорах;
   - выгрузка данных с сопроцессоров в системную память по разделяемому каналу;
3. установить зависимости между этими операциями, определить, на каких этапах может быть использовано совмещение операций пересылки данных (загрузки или выгрузки) и запуска ядер;
4. с учётом совмещения, выделить в последовательности операций основной цикл;

5. оценить производительность ядра ($Perf_{kern}$) — с учётом набора инструкций и особенностей архитектуры сопроцессора (в случае, если имеются готовые реализации аналогичных ядер, вместо оценок можно воспользоваться данными об их производительности при исполнении на аппаратном стенде или потактовом программном эмуляторе);
6. оценить время работы ядра в основном цикле как количество арифметических операций на одном сопроцессоре, делённое на производительность ядра;
7. оценить время пересылок в основном цикле как общий объём пересылаемых данных для всех сопроцессоров, делённый на пропускную способность канала (BW);
8. оценить время на одну итерацию цикла: сравнить время вычислений в ядре и время пересылок, которые выполняются с совмещением, и к большему из двух прибавить время на оставшиеся операции, которые выполняются без совмещения;
9. оценить общее время работы процедуры как время на одну итерацию цикла, умноженное на количество итераций;
10. оценить производительность процедуры как общее количество арифметических операций, делённое на оценку времени работы.

Данный метод позволяет до начала реализации процедуры оценить целесообразность использования той или иной ВС для задач выбранного класса. Ключевым является этап 8, на котором в общие формулы подставляются параметры конкретной ВС и определяется, чем в действительности ограничена производительность процедуры на этой ВС: производительностью ядра на сопроцессоре или пропускной способностью канала между сопроцессорами и системной памятью.

В дальнейшем, в ходе работы над реализацией, оценки могут быть уточнены. В частности, в формулы может быть подставлена производительность реализованных ядер и скорость пересылок на конкретных схемах доступа к памяти, вместо теоретических максимумов. Сравнение таких уточнённых оценок с результатами замеров для разработанной процедуры позволяет оценить качество программного кода и выявить те существенные особенности архитектуры, которые ограничивают производительность данной процедуры и не позволяют достичь теоретического максимума производительности.

С другой стороны, формулы, отражающие зависимость между производительностью алгоритма и параметрами системы, позволяют оценить ожидаемый прирост производительности за счёт изменения того или иного параметра и таким образом определить приоритетные направления развития архитектуры.

Предложенный подход к анализу производительности процедур перекликается с рядом исследований, посвящённых эффективной реализации расчётных задач на различных аппаратных платформах, влиянию характеристик архитектуры на производительность и потенциалу использования платформы в ВПВ. Например, в [141] такое исследование проводилось для процессора STI Cell. Сопроцессоры на нём не имеют кэш-памяти и не поддерживают внеочередное исполнение инструкций, что позволяет достаточно точно оценивать производительность вычислительных ядер по количеству тактов. Это свойство присуще и процессорам ВМ7/9.

В [152] оценка целесообразности использования GPU для тех или иных вычислений выполняется автоматически. Генерируется код ядер на OpenMP для CPU, а на его основе — код на OpenCL C для GPU, который переупорядочивается в более эффективные структуры данных при помощи машинного обучения. Затем, на основе анализа синтаксического дерева программ и параметров времени исполнения, автоматически определяется, будут ли вычисления на GPU более производительными, чем на CPU. Другими словами, упрощается и автоматизируется этап разработки ядра и оценки $Perf_{kern}$. Целесообразность, однако, определяется по эффективности ядра как такового, без учёта обменов данными между GPU и системной памятью: копии массивов данных хранятся как в системной памяти, так и на GPU, и при необходимости осуществляются пересылки.

Работа [105], напротив, посвящена минимизации потерь на пересылку данных, но без переработки программного кода. Выбирается оптимальная политика выделения памяти, что позволяет приблизиться к пиковой скорости пересылок. Такие оптимизации могут быть эффективны, в особенности для алгоритмов, требующих пересылки данных из разреженных областей. Тем не менее, как будет показано в последующих главах, в алгоритмах, связанных с интенсивными обменами данными, даже пиковая пропускная способность каналов может оказаться недостаточной.

При наличии в системе вычислителей с разными характеристиками выбор схемы вычислений и оценка производительности связаны с дополнительным этапом — распределением работы между вычислителями. Например, в [99] рассматриваются распределённые гетерогенные ВС. Предлагается алгоритм, позволяющий на основе теоретического анализа выбрать эффективную схему распределения работы между вычислителями.

В [87] рассматриваются энергоэффективные гибридные системы в целом и технические приёмы, которые используются при разработке высокопроизводительных процедур под эти системы.

Данное исследование можно противопоставить работе [20], в которой представлены результаты запусков нескольких известных наборов тестов, в том числе NPB, на 1-16 ядрах Эльбрус. Архитектурно-зависимые оптимизации выполняются на этапе компиляции, однако отмечается, что для более полного использования потенциала процессора требуется доработка ряда процедур вручную.

1.6 Формальный критерий сбалансированности вычислительной системы на заданной вычислительной процедуре

Использование описанного в предыдущем разделе метода оценки производительности позволяет определить ВС, сбалансированную на заданной вычислительной процедуре, и вывести формальный критерий сбалансированности ВС на вычислительной процедуре.

Система является сбалансированной, если в основном цикле вычисления базовой операции время работы ядра совпадает со временем пересылок. На этапах 6 и 7 разработанного метода выводятся оценки времени работы ядра и времени пересылок как величины, зависящие от параметров построенной модели гибридной ВС и от параметров процедуры. Критерий сбалансированности состоит в равенстве этих величин.

Данный критерий будет конкретизирован в главе 4 для гибридных процессоров НИИСИ РАН и рассматриваемых вычислительных процедур.

Дальнейшие главы посвящены исследованию выбранных процедур, с использованием построенной модели гибридной системы и разработанного метода оценки производительности.

В главе 2 рассматриваются современные гибридные системы с графическими ускорителями. В соответствии с методом, строятся схемы вычислений для трёх процедур и выводятся теоретические оценки производительности. Описываются разработанные автором реализации процедур и приводятся результаты их тестирования, которые подтверждают применимость метода.

В главе 3 метод рассматривается на примере гибридных процессоров КОМДИВ: ВМ7 и ВМ9. Для выбранных процедур строятся схемы вычислений и выводятся теоретические оценки производительности. Описываются разработанные автором реализации, оптимизированные под сопроцессор СР2, и архитектурные ограничения, которые оказываются существенными для этих процедур. Приводятся результаты тестирования разработанных реализаций, которые также подтверждают применимость метода. Результаты сопоставляются с производительностью аналогичных процедур на процессорах мировых производителей.

В главе 4 делаются выводы о достоинствах и недостатках архитектурных решений гибридных процессоров НИИСИ РАН, с точки зрения рассмотренных классов задач. Для этих процессоров на основе оценок, полученных в главе 3, сформулированный выше критерий сбалансированности конкретизируется для каждой из рассмотренных процедур. Проводится анализ особенностей архитектуры гибридных процессоров НИИСИ РАН, которые не позволяют достичь теоретического максимума производительности. Предлагается проект доработки архитектуры процессоров.

Глава 2. Исследование реализаций процедур на GPU

Реализации алгоритмов из NPB для GPU, с использованием различных подходов и средств, разрабатывались ранее и другими исследователями.

Например, в [112] реализация проводилась при помощи директив OpenACC. Авторы отмечают, что часть важных оптимизаций, в том числе управление пересылками данных, не может быть полностью автоматизирована и возложена на компилятор, поэтому программирование под GPU неизбежно подразумевает доработку кода вручную.

В ИПМ РАН алгоритм NPB MG был реализован на языке Fortran-DVMH (разработанном в ИПМ расширении языка Fortran для параллельного программирования, включающем специализированные директивы): [28]. Основное внимание в статье уделяется оптимизации циклов, исполняемых в ядрах на сопроцессоре, а не обменов данными. Отмечается, однако, что из-за относительно низкой скорости обменов наблюдается замедление работы процедуры. Для алгоритмов FT и CG реализация не проводилась.

Реализации, описанные далее в этой главе, были разработаны в рамках стандарта OpenCL. Одна из первых попыток портировать тесты NPB на OpenCL описана в [123]. В дальнейшем были разработаны более эффективные процедуры. В частности, такие процедуры входят билиотеку SnuCL [128], причём OpenCL используется здесь не только для вычислений на гибридном узле, но и для межузловых коммуникаций, в качестве альтернативы MPI. В этой реализации распределение и пересылка буферов данных между устройствами просходит прозрачно для программиста, что упрощает процесс разработки, но оставляет меньше возможностей для оптимизации. Авторы отмечают, что на производительности алгоритмов FT, MG и CG отрицательно сказываются пересылки данных и операции с памятью. В данной главе этот вывод получит формальное обоснование.

В [152] представлена автоматическая система генерации кодов и оценки их эффективности, которая тестируется в частности и на NPB. Здесь оптимизации также выполняются на уровне ядра, но, в отличие от [28], этот процесс автоматизирован.

2.1 Процедура NPB FT

Исследования путей реализации БПФ на GPU начались ещё до того, как GPU стали применяться для ВПВ повсеместно — например, в [110] предлагается реализация, в которой входные данные представляются в виде изображения и записываются в память текстур GPU. В дальнейшем появились специализированные библиотеки для вычисления БПФ на GPU, среди которых одна из самых популярных — CuFFT на CUDA для GPU от NVIDIA.

2.1.1 Схема вычислений

В алгоритме FT рассматривается трёхмерное комплексное БПФ, причём все три размерности входного массива — степени 2. Обозначим размер массива как $N_1 \times N_2 \times N_3$. Общепринятый алгоритм вычисления такого преобразования состоит из трёх шагов.

1. Набор одномерных БПФ по первому измерению массива: $N_2 \times N_3$ БПФ длины N_1. Для этого используется известный алгоритм Кули-Тьюки [55].
2. Набор одномерных БПФ длины N_2 по второму измерению.
3. Набор одномерных БПФ длины N_3 по третьему измерению.

Для удобства рассмотрим сначала схему реализации на гибридной ВС двумерного БПФ порядка $N_1 \times N_2$: алгоритм аналогичен трёхмерному, но состоит только из двух шагов. Входные и выходные данные располагаются в системной памяти. От значений параметров *HMEM* и *DMEM* — объёмов памяти — зависит максимальный размер задачи, которая может быть решена на данной ВС. Пусть размер задачи таков, что памяти одного сопроцессора достаточно для обработки хотя бы одного вектора длины N_1 и длины N_2.

На первом шаге весь набор столбцов делится на группы — страницы — по p_1 векторов. Значение p_1 определяется исходя из общего количества сопроцессоров — K — и объёма памяти одного сопроцессора — *DMEM*. Оптимальное

значение p_1 должно уточняться экспериментально для каждой конкретной системы и размера задачи.

Каждая страница столбцов загружается в память одного сопроцессора, далее на всех сопроцессорах параллельно вызывается ядро БПФ длины N_1 для всех векторов страницы, после чего страницы результатов со всех сопроцессоров выгружаются во временный массив в системной памяти. Всего на каждом сопроцессоре может обрабатываться одна страница или несколько страниц последовательно, в зависимости от их общего количества. Второй шаг аналогичен первому: набор строк полученного временного массива разделяется на страницы по p_2 векторов длины N_2, каждая страница обрабатывается на одном из сопроцессоров, и окончательный результат собирается в системной памяти.

Отметим, что в каждой загрузке и выгрузке страницы столбцов неявно присутствует ещё одна операция — транспонирование — т.к. элементы одного столбца располагаются в памяти с некоторым шагом. Далее в этом и следующем разделах, при получении теоретических оценок производительности, время на транспонирования рассматривается как часть накладных расходов на загрузку данных и не учитывается.

Совмещение вычислений и пересылок на первом и на втором шаге алгоритма естественно организовать следующим образом: на каждом сопроцессоре обработка очередной страницы совмещается с выгрузкой предыдущей страницы результатов и загрузкой следующей страницы входных данных. На «стыке» двух шагов совмещения вычислений с пересылками не происходит: необходимо дождаться получения всех результатов БПФ по столбцам, чтобы применить к полученной матрице БПФ по строкам.

В ходе работы весь объём данных (массив порядка $N_1 \times N_2$) два раза пересылается по разделяемому каналу на сопроцессоры и обратно. При использовании выбранного алгоритма сократить количество пересылок невозможно: на первом шаге на одном сопроцессоре можно обработать лишь часть столбцов входной матрицы и получить для них результаты, однако для второго шага необходимы результаты для всех столбцов, полученные на других сопроцессорах.

В трёхмерном БПФ к схеме вычислений добавляется ещё один шаг алгоритма. Для массивов очень большой размерности, для которых в памяти одного сопроцессора единовременно может быть обработано лишь несколько векторов

длины N_1, N_2 или N_3, три шага выполняются по отдельности: на каждом шаге вычисляется набор одномерных БПФ, и весь объём данных пересылается на сопроцессоры и обратно. Для массивов меньшей размерности в памяти одного сопроцессора может быть вычислено двумерное БПФ порядка $N_1 \times N_2$ (либо $N_2 \times N_3$, тогда дальнейшие рассуждения аналогичны). В этом случае два первых шага алгоритма могут быть объединены: для каждого «слоя» входного массива — двумерного массива порядка $N_1 \times N_2$, соответствующего фиксированной третьей координате — набор БПФ длины N_1, а затем N_2, вычисляется независимо от других слоёв и представляет собой двумерное БПФ. Таким образом, трёхмерное БПФ подразделяется на два этапа: набор двумерных БПФ порядка $N_1 \times N_2$ и набор одномерных порядка N_3. На каждом этапе происходит в совокупности одна пересылка всего объёма данных в память сороцессоров и обратно. На первом этапе вычисление набора двумерных БПФ выполняется постранично, аналогично набору одномерных. В следующем разделе оценки будут получены для этой схемы вычислений.

2.1.2 Оценки производительности

Для получения теоретических оценок производительности целесообразно рассматривать операции, которые выполняются в основном цикле программы. В реальности всегда имеются различные накладные расходы — в частности, на «разгон» и «торможение» циклов, когда пересылки совмещаются с вычислениями не полностью. Однако при достаточно большом объёме входных данных и, соответственно, количестве итераций цикла эти расходы не оказывают существенного влияния на общее время работы, поэтому для простоты ими можно пренебречь.

Будем предполагать, что ядро одномерного БПФ на сопроцессоре реализовано и имеет производительность $Perf_{kern}$. Описанная выше схема вычисления трёхмерного БПФ с совмещениями состоит из двух этапов и, соответственно, двух циклов. На одной итерации каждого цикла обработка одной страницы выполняется параллельно с пересылкой одной страницы туда и одной — обратно. Таким образом, общее время выполнения каждого шага определяется

максимальным из двух: суммарным временем выгрузки и загрузки или временем вычислений над одной страницей данных. Очевидно, что соотношение между этими величинами не зависит от количества векторов в одной странице, поэтому в формулах ниже считается, что одна страница — это один вектор.

Второй этап представляет собой вычисление набора одномерных БПФ. Время на выгрузку и загрузку K векторов длины N_3, по одному на каждый сопроцессор, выражается через параметры построенной модели по формуле:

$$T^2_{L+S} = \frac{2KN_3}{BW} \tag{2.1}$$

(«$L+S$» условно обозначает загрузку и выгрузку — «load + store»).

Количество арифметических операций на вычисление БПФ одного вектора составляет $OP_{\text{FFT}}(N_3) = 5N_3 \log N_3$. Отсюда, время на вычисления в ядре с одним вектором, параллельно на K сопроцессорах, оценивается по формуле:

$$T^2_C = \frac{5N_3 \log N_3}{Perf_{kern}} \tag{2.2}$$

(«C» условно обозначает вычисления — «computations»).

При сравнении величин 2.2 и 2.1 необходимо принять во внимание единицы измерения, в которых берутся входящие в формулы параметры. Если BW представлено в МБ/с, $Perf_{kern}$ — в МОП/с, а каждый элемент массива — комплексное число двойной точности (16 байт), то сравнение $T^2_C \vee T^2_{L+S}$ может быть переписано в виде:

$$5\,BW \log N_3 \quad \vee \quad 32K \cdot Perf_{kern}\,. \tag{2.3}$$

Для первого этапа оценки времени на обработку одной страницы, т.е. вычисление двумерных БПФ, полностью аналогичны:

$$T^1_{L+S} = \frac{2KN_1N_2}{BW}, \quad T^1_C = \frac{5N_1N_2 \log(N_1N_2)}{Perf_{kern}}. \tag{2.4}$$

Сравнение имеет вид:

$$5\,BW \log(N_1N_2) \quad \vee \quad 32K \cdot Perf_{kern}\,. \tag{2.5}$$

В случае если в 2.3 и 2.5 значение выражения в правой части больше, общее время вычисления БПФ (T_{FFT}) определяется временем пересылок данных и

может быть оценено как время пересылки всего массива 2 раза на сопроцессоры и обратно:

$$T_{\text{FFT}} \approx \frac{4N_1N_2N_3}{BW}. \tag{2.6}$$

Тогда производительность вычислений оценивается сверху по формуле

$$PERF_{\text{FT}} = \frac{5N_1N_2N_3 \log(N_1N_2N_3)}{T_{\text{FFT}}} \approx 1{,}25\, BW \log(N_1N_2N_3). \tag{2.7}$$

2.1.3 Производительность на GPU

Рассмотрим конкретный пример вычислительного узла с четырьмя GPU NVIDIA GeForce GTX TITAN (далее GeForce TITAN), параметры которого приведены в таблице A.1.

Для оценки производительности OpenCL-ядра одномерного БПФ на одном ускорителе можно воспользоваться данными о производительности аналогичного ядра на CUDA, входящего в состав библиотеки cuFFT от NVIDIA. Поскольку платформа OpenCL в целом предоставляет те же возможности, что и CUDA, можно ожидать, что производительности ядер окажутся приблизительно равны. В [57] приводятся результаты тестирования процедуры БПФ двойной точности на ускорителе NVIDIA Tesla K40m: порядка 170 ГОП/с, что составляет 12% от пика. Результаты тестирований, приводимые в независимых источниках для различных GPU, сопоставимы: одномерное БПФ вычисляется на графических ускорителях с эффективностью порядка 10-15% от пика. Например, согласно [127], производительность ядра БПФ на GeForce TITAN составляет до 222 ГОП/с. Ту же эффективность показывает и библиотека clFFT от AMD, реализованная на OpenCL.

Таким образом, можно приближённо считать, что производительность ядра одномерного БПФ на одном ускорителе составляет $Perf_{kern} \approx 200$ ГОП/с.

Рассмотрим задачу класса C, в которой $N_1 = N_2 = N_3 = 512$. Подставляя в 2.5 значения параметров узла, получаем сравнение:

$$5 \cdot 32000 \cdot 18 \quad \vee \quad 32 \cdot 4 \cdot 200000. \tag{2.8}$$

Значение выражение справа на два порядка больше, чем выражения слева. В 2.3 значение выражения слева ещё в два раза меньше. Таким образом, наблюдение, что производительность БПФ больших массивов определяется не производительностью ядра на GPU, а скоростью пересылки данных по PCI Express, которое было ранее сделано другими авторами (например, [53]), получило строго обоснование в терминах модели.

Производительность алгоритма FT может быть оценена по формуле 2.7:

$$PERF_{\text{FT}} \approx 1{,}25 \cdot 32 \text{ ГБ/с} \,/ 8 \text{ байт} \cdot 3 \cdot 9 \text{ ОП} = 135 \text{ ГОП/с}\,. \tag{2.9}$$

Сравним полученную оценку с той производительностью, которой мы могли бы ожидать при вычислениях на CPU.

В качестве эталона для сравнения была выбрана C-библиотека FFTW (Fastest Fourier Transform in the West, [72]) — самая быстрая из известных библиотек ДПФ, реализованная на большинстве современных универсальных процессоров. Были проведены замеры производительности процедур FFTW на двух процессорах Intel Xeon E5-2670, результаты приведены в таблице 2.1. Рассматриваются 1-мерное БПФ длины 512, 2-мерное порядка 512×512 и 3-мерное порядка $512 \times 512 \times 512$. Для каждого проводились замеры на трёх уровнях планирования: `ESTIMATE`, `MEASURE` и `PATIENT`. В силу того, что максимальная производительность вычислений достигается на больших объёмах данных, для 1-мерного БПФ приводятся результаты замеров множественной процедуры (БПФ 10000 векторов). Все производительности приводятся в ГОП/с.

Таблица 2.1 — FFTW: производительность на 2x Xeon E5-2670

	`ESTIMATE`	`MEASURE`	`PATIENT`
1D	59,90	60,37	60,51
2D	44,19	56,48	82,01
3D	11,66	50,30	52,48

В частности, производительность 3-мерного БПФ, к которому сводится алгоритм FT класса C, для уровня планирования `PATIENT` составляет

52,48 ГОП/с. Сравнивая эту величину с оценкой 2.9, можно увидеть, что на рассматриваемом гибридном узле при расчётах по алгоритму FT использование GPU даст выигрыш в производительности почти в 2,5 раза. Этот выигрыш, однако, не зависит от количества установленных ускорителей, в то время как реализации на CPU хорошо масштабируются с ростом количества процессоров.

Сделанный на основе теоретических оценок вывод о низкой эффективности вычисления FT на GPU подтверждается и данными о производительности существующих реализаций алгоритма.

Так, процедура FT вошла в упомянутую выше библиотеку SnuCL. В [128] приводятся результаты её тестирования на кластере, включающем 9 гибридных узлов по 4 GPU NVIDIA GTX 480, и сравнение с производительностью вычисления FT на одном ядре CPU Xeon X5680. При вычислении FT класса B использование 4 GPU даёт ускорение в 3 раза, 8 GPU — в 5,5 раз, 16 GPU — в 9 раз, 32 GPU — в 14 раз. Суммарное количество процессорных ядер в 32 рассматриваемых ускорителях — 15360, однако производительность оказывается всего в 14 раз выше, чем на одном ядре CPU, в то время как другие процедуры библиотеки показывают ускорение в тысячи раз. Это подтверждает вывод о нецелесообразности использования GPU для расчётов по данному алгоритму.

В [152] процедура FT для GPU, полученная при помощи автоматических средств, имеет более высокую производительность, чем процедура из SnuCL. При этом рассматриваются только такие классы задач, для которых все необходимые данные помещаются в память одного GPU — это означает, что данные пересылаются на GPU и обратно один раз, а не два. Тем не менее, расчёт на CPU с использованием OpenMP оказывается более эффективным.

Отметим, что поскольку производительность БПФ определяется пересылками данных, возможность обменов напрямую между GPU, в обход системной памяти, при той же пропускной способности канала означала бы прирост производительности вдвое: две пересылки — с GPU в системную память и из системной памяти на GPU — заменялись бы на одну, с GPU на GPU. Такая возможность реализована в технологии NVlink [78] для GPU от NVIDIA.

2.2 Процедура NPB MG

Как отмечалось в разделе 1.4, вычисление MG представляет собой последовательность PO — линейных разностных операторов на трёхмерной сетке. В [28] при оптимизации ядер основной акцент делается на процедуры проекции и интерполяции, в силу более сложной организации внутренних циклов. Однако, замеры, проведённые в НИИСИ для референсных кодов NPB MG, показали, что приблизительно 90% общего времени работы алгоритма на современных универсальных процессорах занимает вычисление невязки и сглаживания на самой мелкой сетке, то есть для массива порядка $N \times N \times N$. Поэтому в данной работе в качестве базовой операции алгоритма будет рассматриваться упрощённый PO на самой мелкой сетке, с четырьмя ненулевыми коэффициентами, оставляющий сетку неизменной: выходное значение в каждой точке — сумма 27 соседних старых значений с весовыми коэффициентами (см. рисунок 2.1).

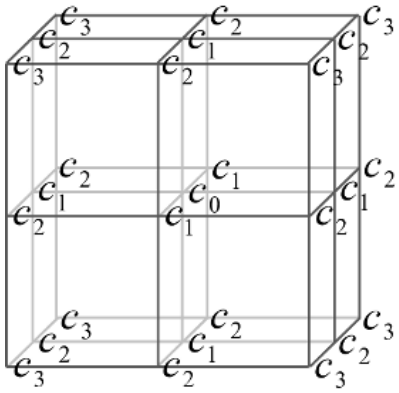

Рисунок 2.1 — PO: схема вычисления нового значения в точке

Упрощённая задача, для которой будут проводиться оценки: последовательность таких PO, в которой выходной массив на каждом шаге является входным для следующего шага.

2.2.1 Схема вычислений

Методы распараллеливания трафаретных вычислений исследовались и раньше, для систем классической архитектуры. Так, в работе [39] рассматри-

ваются классические многоядерные системы, и отмечается, что оптимизация должна производиться на трёх уровнях: коммуникации между узлами (например, через MPI), многопоточность на одном узле (например, через OpenMP) и использование SIMD-инструкций (например, расширения Intel SSE3).

В [131] рассматриваются параллельные трафаретные вычисления на широком классе архитектур: как классических, с кэш-памятью, так и гибридных, с программно контролируемой локальной памятью — в том числе GPU и процессорах Cell. Основное внимание уделяется работе с различными иерархическими системами памяти. Предлагается единая схема параллелизации, на основе многоуровневого разбиения массива на блоки, и система автоматического подбора параметров такого разбиения под конкретную архитектуру. Полученная процедура на GPU представляет собой пример эффективного ядра. С учётом, однако, пересылок данных по PCI Express итоговая производительность оказывается крайне низкой — это подтверждает необходимость отдельно исследовать и оптимизировать коммуникации между GPU и системной памятью.

Параллельную реализацию последовательности РО можно организовать следующим образом. Массив делится на блоки, и очередной РО вычисляется по отдельности для точек каждого блока. При этом для вычисления значений в точках, принадлежащих граням блока, требуется 6 дополнительных граней (таким образом блок увеличивается на 2 элемента по каждому измерению). Эти дополнительные грани являются копиями граней соседних блоков, либо противоположных граней того же блока, если по какому-то направлению разбиение массива не производилось. Для единообразия и удобства реализации входной массив также хранится с дополнительными гранями, которые являются копиями противоположных граней целого массива.

В случае гибридной системы естественным образом каждый блок обрабатывается на одном сопроцессоре. Ядро на каждом сопроцессоре получает на вход блок с дополнительными гранями и вычисляет РО, давая на выходе новый блок (без дополнительных граней).

Классическая схема разбиения состоит в том, что массив разбивается на блоки, по возможности равные по всем измерениям: делится пополам поочерёдно по всем направлениям. Эта схема подходит для случая, когда число сопроцессоров является степенью 2. Преимущества этого способа при реализации алгоритма MG:

– общий объём пересылаемых данных минимально возможный;
– на одном сопроцессоре выполняется максимальное возможное количество уровней алгоритма MG (поскольку при переходе от уровня к уровню сетка уменьшается вдвое по каждому измерению);
– схема реализована в MPI-версии референсного кода NPB.

Количество «разрезов», которые нужно сделать, и, соответственно, размеры блоков подбираются таким образом, чтобы памяти одного сопроцессора было достаточно для вычисления РО для одного блока. При этом количество блоков может оказаться бóльшим, чем K, либо меньше или равным ему. Таким образом, возникает два случая и, соответственно, две возможных схемы работы.

В данном разделе рассмотрим случай, когда массив разбивается на K блоков, и памяти одного сопроцессора достаточно для обработки одного блока: такая ситуация имеет место, например, при решении задачи NPB MG классов B и C на гибридных узлах с современными GPU.

Блоки входного массива, вместе с дополнительными гранями, загружаются в память сопроцессоров один раз, перед началом работы: по одному блоку на каждый сопроцессор. В ходе вычислений все сопроцессоры параллельно вызывают ядро РО, каждый — для своего блока. Перед вычислением следующего РО последовательности необходимо произвести обмен гранями с соседними сопроцессорами: с каждого сопроцессора скопировать вычисленные на нём грани в дополнительные грани на соседних сопроцессорах — тогда на каждом сопроцессоре окажется блок нового массива с актуальными дополнительными гранями.

Чтобы совместить вычисления в ядре с пересылками, для каждого РО работа разделяется на два этапа: вычисление граничных точек блока и вычисление внутренних точек, одновременно с обменом гранями с другими сопроцессорами.

В целом, та же схема распараллеливания применяется для трафаретных вычислений в [118] и реализована средствами CUDA и MPI. Распараллеливание происходит на уровне компилятора; вычисления с внутренними точками совмещаются с пересылками граней. Похожая схема вычислений автоматизирована и в [125], также для GPU от NVIDIA. В реализации библиотеки вычисления на узле распараллеливаются между GPU при помощи OpenMP, при этом для GPU с технологией NVlink ([78]) обмены между ними происходят

напрямую; коммуникации между узлами осуществляются через MPI. Авторы отмечают потенциальную эффективность совмещения вычислений и обменов данными, однако к моменту публикации статьи оно не было реализовано. В библиотеке [117], поддерживающей и GPU, также реализована отдельная обработка граней и совмещения.

Альтернативный подход предложен в [82]: с каждым блоком загружать не по одной дополнительной грани с каждой стороны, а по несколько. Это позволяет вычислять несколько РО подряд, и только после этого выполнять обмены. Такая схема оказывается выгодной для двумерных шаблонов, однако в трёхмерном случае прироста производительности не даёт — из-за избыточности, связанной с увеличением размеров блоков.

2.2.2 Оценки производительности

Получим оценки времени вычисления одного РО, по описанной выше схеме. Введём следующие обозначения:

- T_{CB} — время на вычисление для точек на гранях блока («compute boundary»);
- T_{CI} — время на вычисление для внутренних точек блока («compute inner»);
- $T_C = T_{CB} + T_{CI}$ — время на вычисление новых значений во всех точках блока;
- T_{SB} — время на обмен гранями между процессорами («swap boundaries»).

Вычисления для внутренних точек совмещаются с обменами гранями, поэтому общее время вычисления одного РО (T_{PO}) зависит от соотношения между T_{CI} и T_{SB}: $T_{\text{PO}} = T_{CB} + \max\{T_{CI}, T_{SB}\}$. Однако для получения теоретической производительности алгоритма MG можно использовать и более грубую оценку. Для этого примем во внимание, что количество внутренних точек блока совпадает по порядку с общим количеством точек, и время на вычисление граничных точек несущественно по сравнению со временем вычисления внутрен-

них точек. Если упрощённо положить, что $T_{CB} = 0$, $T_{CI} = T_C$, то получим: $T_{\text{PO}} = \max\{T_C, T_{SB}\}$.

Пусть размер одного блока — $N_1 \times N_2 \times N_3$, тогда количество граничных точек, для каждого блока, составляет по порядку $2(N_1N_2 + N_2N_3 + N_1N_3)$, причём с каждого сопроцессора необходимо выгрузить все вычисленные грани, а затем загрузить новые. Для базовой операции PO (см. рис. 2.1) на вычисление нового значения в каждой точке сетки требуется 30 арифметических операций. Пусть имеется реализованное ядро PO с производительностью $Perf_{kern}$. Тогда формулы для оценки времени на вычисления и на пересылки получаются следующие:

$$T_C = \frac{30N_1N_2N_3}{Perf_{kern}}, \tag{2.10}$$

$$T_{SB} = \frac{4K(N_1N_2 + N_2N_3 + N_1N_3)}{BW}. \tag{2.11}$$

Параметр K появляется в 2.11, но отсутствует в 2.10, поскольку обмены гранями осуществляются через разделяемый канал, а вычисления выполняются на всех сопроцессорах одновременно. При сравнении этих величин необходимо учесть единицы измерения всех входящих в формулы параметров.

В случае, если $T_{SB} \ll T_C$, производительность вычисления одного PO, а значит и алгоритма MG в целом, определяется производительностью ядра и оценивается сверху по формуле:

$$PERF_{\text{MG}} \approx K \cdot Perf_{kern}\,. \tag{2.12}$$

Получим оценку для рассмотренного ранее узла с четырьмя GPU GeForce TITAN. Автором было прежде всего разработано ядро на OpenCL C для вычисления PO (а именно, невязки) на одном ускорителе, которое будет подробнее обсуждаться в следующем разделе. Его производительность на массивах порядка $256\times256\times256$ составила приблизительно 128,42 ГОП/с — примем эту величину за $Perf_{kern}$.

Рассмотрим, для определённости, задачу класса B. Самая мелкая сетка соответствует массиву порядка $256 \times 256 \times 256$, который разбивается на четыре блока, по числу ускорителей. Каждый блок имеет размер $N_1 \times N_2 \times N_3 = 256 \times 128 \times 128$. Подставляя значения всех величин в формулы 2.10 и 2.11,

получаем:

$$T_C = \frac{30 \cdot 128 \cdot 128 \cdot 256 \text{ ОП}}{128420 \text{ МОП/с}} \approx 979{,}82 \cdot 10^{-6} \text{ с}\,, \tag{2.13}$$

$$T_{SB} = \frac{4 \cdot 4(128^2 + 2 \cdot 128 \cdot 256) \cdot 8 \text{ байт}}{32000 \text{ МБ/с}} \approx 327{,}68 \cdot 10^{-6} \text{ с}\,. \tag{2.14}$$

Время на вычисления — T_C — оказывается больше, поэтому производительность алгоритма MG определяется производительностью ядра PO и оценивается как

$$PERF_{\text{MG}} = 4\, Perf_{kern} \approx 513 \text{ ГОП/с}\,. \tag{2.15}$$

Аналогичный результат получается и для задач других классов.

Чтобы получить эталонную производительность алгоритма MG на CPU, в НИИСИ была протестирована референсная OpenMP-реализация. Замеры производились на ряде универсальных процессоров, с перебором различного количества потоков OpenMP. На рисунке Б.1 приводятся диаграммы с результатами замеров на трёх процессорах: Xeon X5670 и Xeon E5-2660 от Intel и Opteron 6178 от AMD (во всех случаях замеры производились на двух CPU). Падение производительности на классах C и D связано с большим объёмом данных, а значит, увеличением количества кэш-промахов при чтении из памяти.

Сравнивая эти результаты с полученными теоретическими оценками, можно сделать вывод, что использование гибридных архитектур с GPU для вычислений по алгоритму MG оправданно и потенциально даёт многократное увеличение производительности. Нужно, однако, учитывать, что в рассмотренной схеме суммарный объём памяти ускорителей накладывает ограничение на размер решаемой задачи. Чем больше класс задачи, тем в общем случае больше ускорителей потребуется для её решения.

2.2.3 Практическая реализация

Автором была разработана гетерогенная реализация алгоритма MG на OpenCL. При этом использовался следующий простой приём: части референсного кода NPB на языке Fortran были «подменены» вызовами разрабо-

танных процедур на C, которые используют API OpenCL и запускают вычислительные ядра на GPU. Генерация данных, замеры времени работы и верификация результата остались без изменений — это позволило сосредоточиться непосредственно на реализации вычислительной процедуры.

Был разработан полный набор вычислительных ядер на OpenCL, которые требуются в алгоритме MG, а именно:

- PO: невязка, проекция, интерполяция (с накоплением и без), сглаживание (с накоплением и без);
- вычисление нормы;
- вспомогательные ядра для обработки, копирования и обменов граней.

«Наивная» реализация ядра невязки, в которой один WI вычисляет значение в одной точке, неэффективна: множества входных точек, которые требуются для получения соседних выходных значений, пересекаются, поэтому чтение данных из глобальной памяти повлекло бы конфликты банков и сериализацию доступа. Чтобы избежать конфликтов, а также увеличить повторную используемость данных, в разработанном ядре была задействована локальная память: при чтении одного и того же элемента соседними WI из локальной памяти конфликтов не возникает. В ходе вычислений в каждый момент времени вычисляется один «слой» выходного блока — массив порядка $N_1 \times N_2$, соответственно, в локальной памяти находятся три соседних входных слоя. При переходе к следующему слою элементы очередного входного слоя считываются из глобальной памяти, по одному элементу каждым WI, на место самого старого слоя.

Поскольку объёма локальной памяти одного вычислителя может быть не достаточно для хранения трёх входных слоёв, каждый слой дополнительно разбивается на прямоугольные блоки размера $M_1 \times M_2$. Одна WG обрабатывает все слои, соответствующие этому блоку; один WI вычисляет в каждом слое один элемент или несколько (r) с шагом M_1 (т.е. находящихся в одном столбце). Распределение работы между WG и WI проиллюстрировано на рисунке 2.2. Здесь $G0 = N_1/M_1$, $G1 = N_2/M_2$, $G0 \times G1$ — общее количество WG; $L0 = M_1$, $L_1 = M_2/r$, $L0 \times L1$ — количество WI в одной WG.

Вычисление невязки в алгоритме MG, помимо описанного вычисления PO, включает также поэлементное вычитание. В ядре сглаживания вычисление PO устроено полностью аналогично. Ядра операторов проекции и интерполяции имеют рад отличий, связанных с изменением размера сетки. В частности,

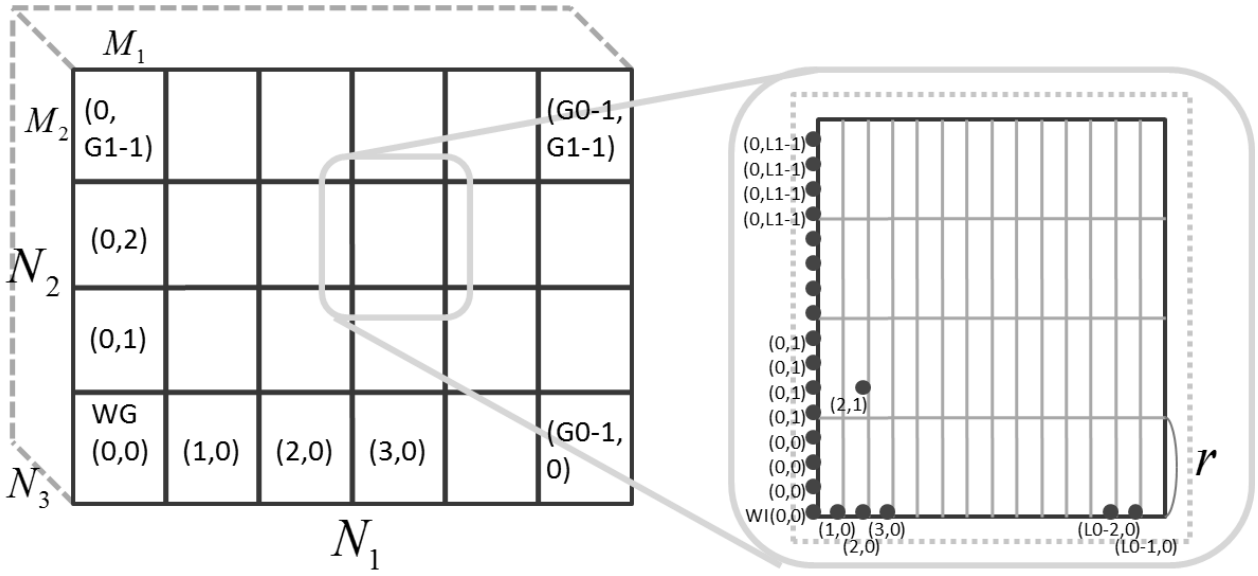

Рисунок 2.2 — Пространство индексов для ядра PO

в ядре интерполяции каждый WI вычисляет 8 выходных элементов, по 8 различным формулам.

Каждое из описанных вычислительных ядер имеет три параметра: M_1, M_2 и r. Оптимальные значения этих параметров должны подбираться экспериментально для каждого размера массива и конкретного GPU. Полный перебор, проведённый автором для нескольких GPU, показал, что в целом оптимальными являются блоки, «вытянутые» по горизонтали ($M_1 \gg M_2$), и малые значения r.

Вычисление нормы, которое выполняется один раз в конце алгоритма, сводится к вычислению суммы квадратов элементов трёхмерного массива — оно выполняется путём попарного суммирования и редукции, через локальную память. Помимо этого, реализация MG использует ряд вспомогательных ядер, описанных далее.

На первом этапе была разработана процедура, выполняющая все вычисления на одном сопроцессоре, без разбиения — это возможно для небольших классов задачи. Входные данные (начальное приближение решения, правая часть, коэффициенты операторов) загружаются в память сопроцессора один раз перед началом вычислений, далее вызываются различные ядра в последовательности, соответствующей алгоритму, и в конце результат выгружается в системную память. После каждого PO вызываются вспомогательные ядра, копирующие в глобальной памяти GPU каждую из 6 вычисленных граней в противополож-

ную дополнительную грань — таким образом происходит учёт периодических граничных условий.

В дальнейшем была разработана процедура MG для гибридного кластера с GPU, на основе референсного MPI-кода. В этой реализации одним GPU управляет один MPI-процесс — т.о. с точки зрения кода обмены данными между GPU на одном и на разных узлах не различаются.

Синхронная реализация, без совмещений, устроена следующим образом:

- одним сопроцессором управляет один MPI-процесс;
- после каждого РО производится выгрузка граней блоков с каждого GPU в системную память;
- происходит обмен вычисленными гранями между процессами, посредством MPI: каждый процесс обменивается с шестью соседними;
- полученные каждым процессом 6 граней загружаются в память GPU, на место дополнительных граней блока;
- в конце вычислений на каждом сопроцессоре вычисляется норма для соответствующего блока, общий результат собирается на CPU.

В результате обменов перед началом вычисления очередного РО в памяти каждого GPU располагается обновлённый блок массива с актуальными дополнительными гранями.

В этой последовательности действий неявно присутствуют процедуры копирования. Это связано с тем, что, во-первых, MPI-обмены выполняются с буферами данных, занимающими непрерывные области в памяти. Во-вторых, элементы граней блоков (за исключением граней, перпендикулярных оси Oz) естественным образом располагаются разреженно в памяти GPU — пересылка таких областей в системную память крайне неэффективна. С учётом этих соображений, для процедуры MG было разработано два набора вспомогательных ядер: копирование вычисленных граней блока в непрерывные области на GPU, перед пересылкой в системную память, и копирование полученных от соседних процессов буферов данных, загруженных в память GPU, в дополнительные грани блока. При этом MPI-обмены выполняются непосредственно с выгруженными непрерывными буферы, без дополнительного копирования на CPU.

На последнем этапе разработки была реализована мульти-буферизация, которая позволяет совместить вычисления с пересылками, как было описано в

разделе 2.2.1. Более точно, 6 граней разделяются на три пары параллельных. Каждая пара обрабатывается специализированными ядрами, которые транспонируют массивы «на лету», при загрузке в локальную память. Выгрузка каждой пары и MPI-обмены выполняются параллельно с обработкой следующей пары. Обмены и загрузки новых граней завершаются на фоне обработки внутренних точек блока. Зависимости между всеми описанными операциями устанавливаются в управляющей процедуре. Совмещение вычислений с пересылками, а пересылок — с MPI-обменами, происходит автоматически, средствами управляющей библиотеки OpenCL и аппаратуры.

В случае, когда общее количество сопроцессоров, K, больше 8, возникает ещё один нюанс, связанный со «спуском» по уровням алгоритма MG. На нижнем уровне алгоритма массив имеет порядок $2 \times 2 \times 2$ и не может быть разделён на более чем 8 «блоков», состоящих из одной точки. Более того, обработку маленького массива целесообразно производить на одном GPU. Поэтому на некотором уровне необходимо перераспределить работу: выгрузить данные со всех сопроцессоров в системную память и загрузить полученный массив на один из сопроцессоров. После выполнения вычислений на нижних уровнях, при последующем вычислении интерполяций и «подъёме» на более мелкие сетки, данные перераспределяются обратно между всеми сопроцессорами. Оптимальный уровень, на котором работа будет перераспределяться, требуется подбирать экспериментально. Следует отметить, что основное время занимают именно вычисления на верхнем уровне, вместе с сопутствующими пересылками данных, поэтому выбор той или иной схемы работы на нижних уровнях является скорее техническим вопросом и может дать относительно небольшой выигрыш в производительности.

Для выполнения вычислений на одном GPU в окончательную версию кода была интегрирована первая версия, в которой обмены через системную память отсутствуют.

2.2.4 Результаты тестирования

Диаграмма на рисунке 2.3 показывает результаты замеров производительности разработанной автором реализации MG на одном ускорителе и, для сравнения, лучшие результаты, полученные на CPU (см. рис. Б.1).

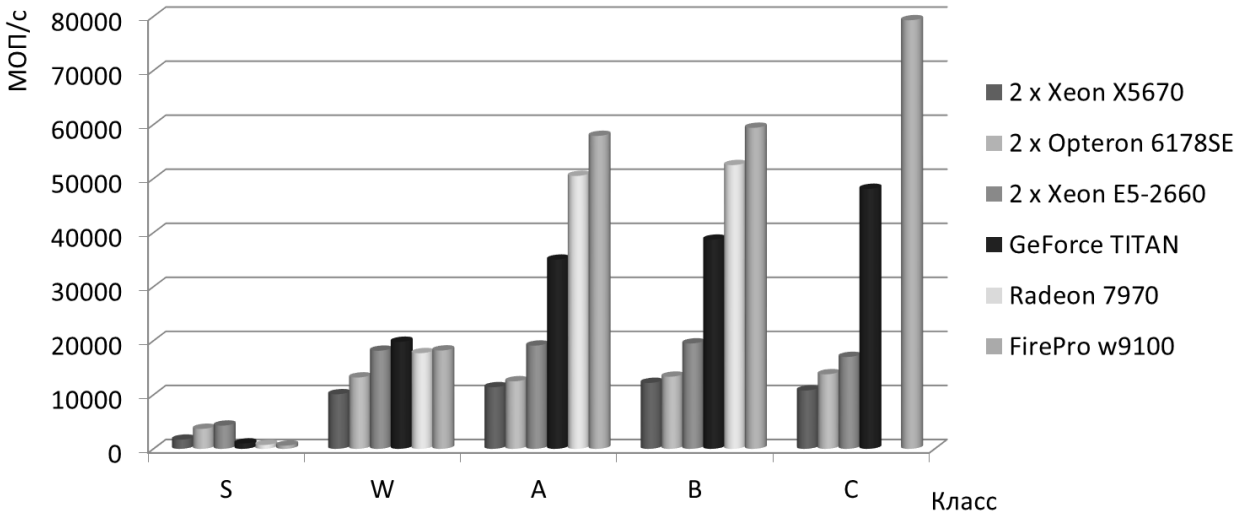

Рисунок 2.3 — MG: лучшие результаты на CPU и процедура на одном GPU

Глобальной памяти GPU AMD Radeon 7970 достаточно для решения задач классов до B включительно, AMD FirePro w9100 (далее FirePro w9100) и GeForce TITAN — классов до C. Уже на классе W производительность вычислений на GPU сопоставима с производительностью на CPU, а для бóльших классов использование GPU даёт преимущество в несколько раз.

В данной реализации всё время расчётов — это суммарное время работы ядер, а пересылки данных отсутствуют. Производительность алгоритма MG в целом грубо оценивается как производительность базовой операции — ядра невязки, однако в реальности оказывается меньше, поскольку алгоритм включает РО разных видов на всех сетках, а также дополнительные ядра копирования и вычисления нормы; имеют место и другие накладные расходы.

На рисунке 2.4 приводятся два графика с результатами замеров производительности полученной гетерогенной реализации алгоритма MG на одном узле, для двух видов GPU и различного их количества. Тестирование на двух FirePro проводилось на ускорителях разных версий: w9100 и w8100. Они обладают одинаковой архитектурой, но отличаются некоторыми количественными

характеристиками: w8100 имеет меньший объём глобальной памяти и пиковую производительность. Вычислительная работа, однако, при тестировании распределялась между ними поровну.

Для класса D приводится всего одно значение, для 8 ускорителей, т.к. памяти меньшего количества ускорителей не достаточно для обработки массива. Реализация показывает хорошую масштабируемость: линейную для класса C, с коэффициентом примерно 0,5. Также очевиден и абсолютный выигрыш от использования GPU, по сравнению с референсной реализацией — например, на классе C:

– 17000 МОП/с на двух современных CPU с OpenMP;
– 161700 МОП/с на восьми современных GPU.

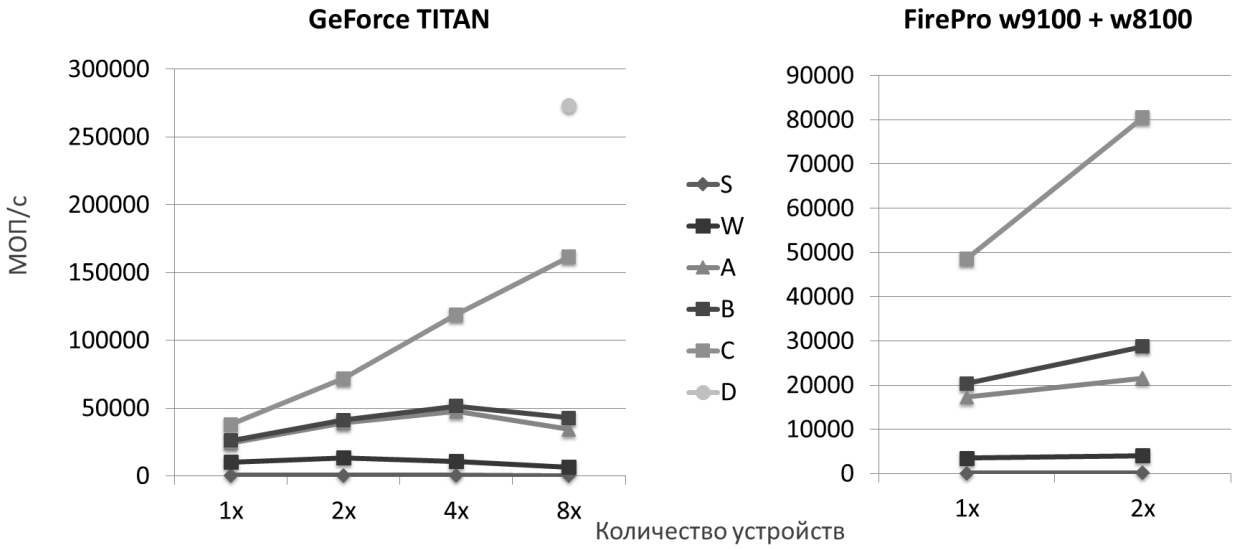

Рисунок 2.4 — Гетерогенная процедура MG: масштабируемость на узле

Сопоставим результаты, полученные на одном и на четырёх GeForce TITAN, с теоретическими оценками, выведенными в 2.2.2. В таблице 2.2 сравниваются значения для задачи класса C. Теоретическая оценка определяется производительностью ядра базовой операции (128,42 ГОП/с). Однако для корректности сравнения необходимо принять во внимание следующее соображение. При подсчёте производительности ядра количество операций определялось как фактическое количество умножений и сложений, выполняемых при вычислении PO: например, по 31 на каждую точку, т.е. $31N^3$ в совокупности, для оператора невязки. Для подсчёта точного количества операций, выполняемых во всём алгоритме, необходимо просуммировать операции по всем уровням алгоритма MG, по всем вызываемым операторам, умножить

на количество итераций и добавить количество операций на вычисление нормы. В референсном коде NPB вместо этого для простоты используется условная величина: общее количество операций для массива порядка $N \times N \times N$ оценивается как $58N^3 \cdot I$, где I — количество итераций в алгоритме. Подсчёт показал, что для задачи класса C эта величина составляет 52% от точного значения. Поэтому в таблице теоретические оценки также домножены на 0,52. Эффективность — соотношение реальной производительности и теоретического максимума, с учётом накладных расходов.

Таблица 2.2 — MG, класс C: сравнение предварительных оценок и результатов

Количество GPU	**Теория (ГОП/с)**	**Практика (ГОП/с)**	**Эффективность (%)**
1x	66,78	41,7	62
4x	267,13	119,0	45

Реальная производительность алгоритма MG, с учётом всех ядер и накладных расходов, составляет порядка 50% от производительности ядра базовой операции в чистом виде. На четырёх GPU дополнительные потери возникают в связи с пересылками данных, которые совмещаются с вычислениями не полностью, в силу особенностей драйверов GPU. Кроме того, при разделении работы на каждом GPU обрабатывается массив меньшего размера, с меньшей эффективностью. Тем не менее, полученная в итоге реальная производительность совпадает с предварительной оценкой по порядку, и таким образом, вывод о целесообразности использования гибридных установок с GPU для расчётов по алгоритму MG подтверждается.

Диаграмма на рисунке 2.5 показывает результаты запусков гетерогенной реализации MG на вычислительном кластере НИИСИ РАН из нескольких узлов. Каждый узел кластера включает один или несколько GPU GeForce TITAN и два Xeon E5-2670 в качестве CPU (кроме случая 8 ускорителей на одном узле — на этом узле установлены Xeon E5-2690v2). Здесь «np» — количество процессов в терминологии MPI, совпадает с общим количеством ускорителей на всех узлах; во всех случаях ускорители (и, соответственно, процессы) распределены между узлами поровну.

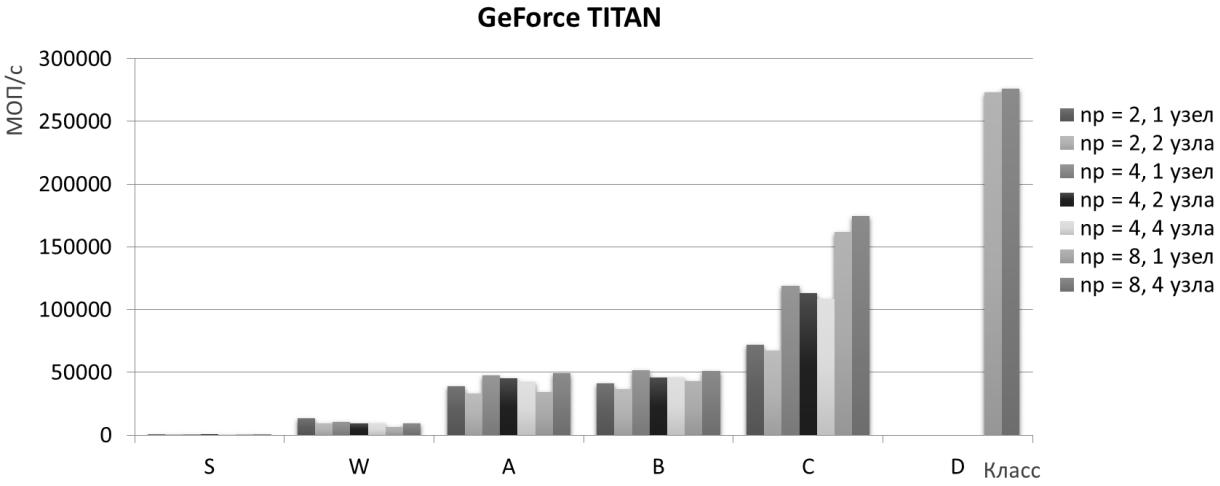

Рисунок 2.5 — Гетерогенная процедура MG: результаты на вычислительном кластере НИИСИ РАН

С увеличением количества узлов, при постоянном общем количестве сопроцессоров, возрастает время на передачи данных, однако сокращается время на дополнительные операции на CPU. В зависимости от общего количества сопроцессоров, тот или иной фактор является определяющим, однако в целом производительность на распределённой установке незначительно отличается от производительности на одном узле, т.к. в большей степени зависит от производительности ядра. На классе D производительность составляет:

– порядка 18 ГОП/с на двух современных CPU с OpenMP;
– до 276 ГОП/с на восьми современных GPU.

Чтобы показать масштабируемость задачи на большее количество сопроцессоров, приведём на рисунке 2.6 результаты запусков на суперкомпьютере K100, с GPU NVIDIA Tesla C2050. Здесь точное значение зависит от взаимного расположения узлов, на которых исполняется задача, поэтому приводятся средние значения по 10 запускам. Параметр «ppn» — в терминологии MPI, количество сопроцессоров на одном узле.

Задача класса D масштабируется до 64 ускорителей:

– 342 ГОП/с на 16 GPU;
– 1,035 ТОП/с на 64 GPU.

Сравним полученные результаты с процедурой MG из библиотеки SnuCL, а также с процедурой на Fortran, разработанной в ИПМ (процедура MG из [152], полученная при помощи автоматических средств, показывает более низкую эффективность, чем процедура на CPU и SnuCL MG). Используются данные о

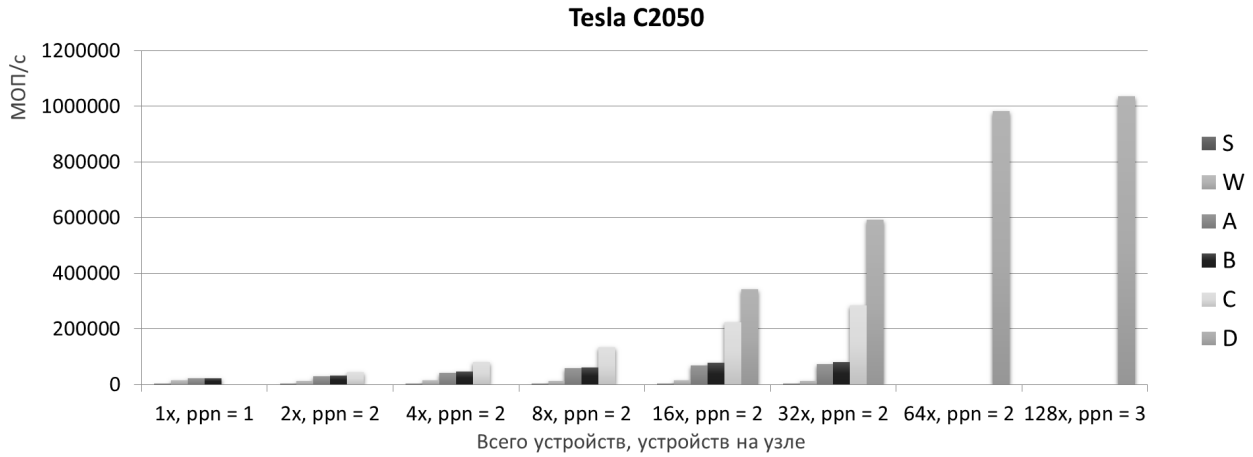

Рисунок 2.6 — Гетерогенная процедура MG: производительность на K100

производительности этих процедур из [28]. В таблице 2.3 приводится ускорение процедур на одном GPU относительно референсной реализации MG на одном ядре CPU Xeon X5670, для задач классов A, B и C. Рассматриваются две модели GPU от NVIDIA: GeForce GTX TITAN и Tesla C2050. Данных о производительности тех же процедур на нескольких GPU в открытых источниках найти не удалось. Ускорение для разработанной реализации вычислено на основе данных, представленных на диаграммах Б.1 (для одного ядра X5670), 2.3 и 2.6.

Таблица 2.3 — MG: ускорение процедур на GPU относительно CPU

	SnuCL		**ИПМ**		**НИИСИ**	
	TITAN	**Tesla C2050**	**TITAN**	**Tesla C2050**	**TITAN**	**Tesla C2050**
A	x20,0	x7,1	x12,8	x7,8	x16,1	x10,3
B	x15,7	x7,0	x13,2	x8,1	x16,5	x9,7
C	x14,8	x7,0	x17,9	x9,8	x21,3	-

Полученные результаты сопоставимы с результатами других авторов. При этом более высокой эффективности на задачах больших классов удалось добиться за счёт оптимизации обменов данными с GPU, в том числе совмещения обменов гранями с вычислениями, а также за счёт эффективного доступа к памяти при обработке самих граней.

2.3 Процедура NPB CG

В данном разделе основное внимание уделяется базовой операции алгоритма CG — вычислению SpMV. Прочие вычисления в алгоритме представляют собой векторные или скалярные операции, доля которых в общем времени работы незначительна.

2.3.1 Формат упаковки разреженной матрицы

При реализации SpMV наиболее важный этап — выбор формата упаковки разреженной матрицы. В отличие от вычислений с плотными матрицами, которые показывают на GPU высокую эффективность за счёт регулярного и выравненного доступа к памяти, в процедуре SpMV доступ нерегулярный, причём эта нерегулярность может быть самых разных форм, в зависимости от конкретной входной матрицы. Поэтому разработчику приходится искать некоторый компромисс между, с одной стороны, компактностью и удобством представления матрицы, и с другой, производительностью доступа к памяти и вычислений.

Популярный формат CSR не удобен для массивно-параллельных сопроцессоров, в том числе GPU, т.к. во-первых, доступ к данным получается невыравненным, а во-вторых, не сбалансировано распределение работы между потоками. Оптимизации процедуры SpMV под GPU посвящено множество работ последних лет. Предлагаются различные варианты упаковки, которые оказываются более или менее выгодными в зависимости от структуры входной матрицы, но все они преследуют основную цель: «обойти» случайный доступ к памяти при чтении элементов входного вектора.

Формат ELL, или ELLpack/ITpack — был использован в двух пакетах: ELLpack [69], предназначенном для решения эллиптических уравнений, и ITpack [76], для решения больших разреженных линейных систем. В данном формате все строки матрицы рассматриваются как содержащие равное количество ненулевых элементов; те строки, в которых реальное число ненулевых элементов меньше, чем максимальное среди всех строк, дополняются нулями,

после чего элементы матрицы и соответствующие номера столбцов записываются в два массива по столбцам. Данный формат обеспечивает регулярный и выравненный доступ к памяти при чтении элементов матрицы. Однако дополнение рядов нулями означает избыточность как при хранении матрицы, так и при выполнении арифметических операций.

В работе [109] предложен формат Sliced ELLpack, который является своего рода компромиссом между ELL и CSR. Вводится параметр H; матрица нарезается на «слои» по H рядов, и каждый слой упаковывается в формате ELL. Таким образом, любая строка, содержащая существенно больше ненулевых элементов, чем в среднем по матрице, влияет на количество «лишних» нулей только в пределах одного слоя. При выборе $H = N$ этот формат эквивалентен ELL, при $H = 1$ — формату CSR. Второй параметр — количество потоков, обрабатывающих один слой. В случае, если на одну строку приходится более одного потока, в конце вычислений выполняется редукция полученных частичных сумм. Можно повысить производительность вычислений в этом формате, если перед вычислением SpMV произвести переупорядочивание строк, так, чтобы в один слой попадали ряды с равным количеством ненулевых элементов. Для матриц с большим разбросом в количестве ненулевых элементов дополнительное ускорение даёт усложнённый формат упаковки матрицы, со слоями переменного размера.

В [90] также предлагается формат на основе ELL — ELLR-T. Здесь используется другая идея: матрица упаковывается в формате ELL и формируется дополнительный массив, в котором сохраняется фактическое количество ненулевых элементов в каждой строке — по нему определяется количетво итераций цикла в ядре. Количество нитей, вычисляющих одно выходное значение, также является параметром. Благодаря этому удаётся избежать избыточных вычислений с нулевыми элементами, причём сохраняется выравненность доступа к памяти и сбалансированность работы между потоками, как в формате ELL. Однако данный формат требует большего объёма памяти, чем формат ELL, поскольку матрица сохраняется с тем же количеством дополнительных нулей, но при этом используется дополнительный массив.

Оба формата — Sliced ELLpack и ELLR-T — показывают сопоставимую производительность, но требуют подбора оптимальных значений параметров для каждого устройства. При оптимальных параметрах прирост в производи-

тельности за счёт использования GPU, по сравнению с современными им CPU, зависит от входной матрицы и составляет от нескольких до 30 раз. В среднем, производительность параллельных реализаций на GPU возрастает с ростом размеров матрицы и количества ненулевых элементов. Следует отметить, что во многих реальных задачах, в том числе и в алгоритме CG из NPB, требуется вычислять SpMV многократно с матрицей одной и той же структуры. Это позволяет произвести перебор параметров один раз, перед первым вызовом процедуры, без существенного влияния на общее время работы таких алгоритмов.

В последние годы, в связи с ростом популярности вычислений на GPU и актуальностью процедуры SpMV для многих задач, было разработано большое количество реализаций SpMV, для матриц в различных форматах. Существуют в том числе реализации с динамическим выбором оптимального формата упаковки, в зависимости от аппаратуры и от структуры конкретной входной матрицы — примером является clSpMV [132].

В данной работе исследование процедуры SpMV и алгоритма CG проводилось на примере формата Sliced ELLpack, с постоянной шириной слоя и без переупорядочивания рядов. При этом без ограничения общности считалось, что порядок матрицы кратен H — ширине слоя (в ином случае матрица дополняется нулевыми строками). В упакованном виде матрица представляется в виде трёх массивов:

1. val — вещественный массив значений — ненулевых элементов и дополнительных нулей;
2. col — массив соответствующих номеров столбцов;
3. ptr — массив индексов начал слоёв в массивах val и col.

2.3.2 Схема вычислений

В алгоритме CG все операции SpMV выполняются с одной и той же матрицей. Они перемежаются векторными и скалярными операциями, которые, в зависимости от реализации, могут выполняться как на CPU, так и на сопроцессорах. В целом, входной вектор очередного SpMV вычисляется по выходному вектору предыдущего. Соответственно, оценки можно проводить для упрощён-

ной процедуры: последовательность SpMV с одной и той же матрицей, в которой выходной вектор каждого SpMV является входным для следующего.

Рассмотрим случай, когда памяти каждого сопроцессора ($DMEM$) достаточно для хранения входного вектора целиком, и кроме того, суммарного объёма памяти всех сопроцессоров достаточно для хранения всей матрицы, упакованной в формате Sliced ELLpack.

Умножение каждого слоя на входной вектор не зависит от других слоёв: на выходе для каждого слоя получается фрагмент вектора из H последовательных элементов. Отсюда можно вывести следующую простую схему распараллеливания: каждый сопроцессор обрабатывает один или несколько слоёв входной матрицы. Соответствующий набор слоёв загружается в память сопроцессора один раз перед началом вычислений. Вычисление каждого SpMV состоит из трёх шагов: загрузка входного вектора в память каждого сопроцессора, умножение на него каждого слоя матрицы и выгрузка полученных фрагментов выходного вектора.

Для умножения любого слоя требуется весь входной вектор, поскольку индексы, по которым будут происходить обращения к нему, заранее не известны и считываются из массиа col. Совместить эту загрузку с вычислениями невозможно. Однако можно организовать совмещение при выгрузке: слои обрабатываются по одному, и вычисления над очередным слоем совмещаются с выгрузкой предыдущего фрагмента результата.

В реальности, как правило, скорость пересылок возрастает с увеличением объёма передаваемых данных. Поэтому может оказаться, особенно при малой ширине слоя и при низкой скорости обменов, что более выгодно обрабатывать в цикле не по одному, а по несколько слоёв, и соответственно выгружать более длинные фрагменты выходного вектора (как следствие, при этом может возрасти время на разгон и торможение цикла). Оптимальное количество обрабатываемых одновременно слоёв может быть подобрано экспериментально. Однако все теоретические оценки будут получены в предположении, что скорость пересылок равна пиковой (BW), поэтому далее будем считать, что на каждом сопроцессоре слои обрабатываются по одному — т.о., K слоёв составляют одну страницу.

2.3.3 Оценки производительности

Пусть матрица имеет порядок $N \times N$, ширина слоя равна H, тогда общее количество слоёв составляет $S = N/H$. Будем без ограничения общности предполагать, что S кратно K — количеству сопроцессоров в системе. Число ненулевых элементов матрицы равно NZ. В общем случае длина массива значений превышает NZ и зависит от структуры входной матрицы; кроме того, количество элементов массива, соответствующих разным слоям, может быть различным. При получении теоретических оценок мы будем, однако, предполагать, что длина массива значений равна NZ, и на каждый слой приходится по $NZ\,/S$ значений. В случае, если матрица в упакованном виде содержит большое количество дополнительных нулей, либо количество элементов сильно различается от слоя к слою, реальная производительность окажется ниже, чем полученная таким способом оценка. Однако для большинства матриц можно подобрать значение H таким образом, чтобы это влияние дополнительных нулей на общую производительность было незначительным и могло быть отнесено к накладным расходам.

Будем считать, что реализовано ядро SpMV для матрицы в формате Sliced ELLpack на сопроцессоре, и оно имеет производительность $Perf_{kern}$.

В описанной схеме вычислений в основном цикле время обработки одной страницы на K сопроцессорах — большее из двух: время вычисления SpMV на одном сопроцессоре для одного слоя (T_C) или время выгрузки K фрагментов выходного вектора со всех сопроцессоров (T_S). Отсюда, общее время вычисления SpMV — это время обработки одной страницы, умноженное на количество страниц, плюс время на загрузку K экземпляров входного вектора на сопроцессоры (T_L).

Время на загрузку оценивается по формуле

$$T_L = \frac{KN}{BW}. \tag{2.16}$$

Количество ненулевых элементов в одном слое составляет $NZ\,/S$, поэтому число арифметических операций на вычисление SpMV для слоя оценивается

как

$$2\frac{NZ}{S} = \frac{2\,NZ \cdot H}{N}. \tag{2.17}$$

Отсюда,

$$T_C = \frac{2\,NZ \cdot H}{N \cdot Perf_{kern}}. \tag{2.18}$$

Один фрагмент выходного вектора содержит H элементов, поэтому время выгрузки со всех сопроцессоров оценивается как

$$T_S = \frac{KH}{BW}. \tag{2.19}$$

С учётом единиц измерения входящих в формулы величин (все элементы — вещественные двойной точности, $Perf_{kern}$ — в МОП/с, BW — в МБ/с), сравнение между 2.18 и 2.19 приводится к виду:

$$NZ \cdot BW \quad \vee \quad 4KN \cdot Perf_{kern}\,. \tag{2.20}$$

Всего каждый сопроцессор обрабатывает S/K слоёв. Тогда, если больше значение выражение слева, то время обработки слоёв определяется временем вычислений, и общее время на одно SpMV оценивается по формуле:

$$T_{\text{SpMV}} \approx \frac{KN \cdot 8}{BW} + \frac{S}{K} \cdot \frac{2\,NZ}{S \cdot Perf_{kern}} = \frac{8KN}{BW} + \frac{2\,NZ}{K \cdot Perf_{kern}}. \tag{2.21}$$

Как правило, на современных системах имеет место именно эта ситуация: время на пересылку вектора пренебрежимо мало по сравнению со временем вычисления SpMV. Более того, первое слагаемое в 2.21 вносит незначительный вклад в общую сумму, поэтому приближённо можно считать, что производительность вычисления SpMV определяется производительностью ядра на сопроцессоре и линейно масштабируется с ростом количества сопроцессоров:

$$PERF_{\text{SpMV}} \approx K \cdot Perf_{kern}\,. \tag{2.22}$$

Оценка производительности SpMV является также оценкой и для всего алгоритма CG.

Рассмотрим пример: гибридный узел с четырьмя GPU GeForce TITAN (табл. A.1) и задачу класса C. Разработанное автором ядро SpMV для формата

упаковки Sliced ELLpack, которое будет подробно рассмотрено в разделе 2.3.4, имеет на соответствующей матрице производительность $Perf_{kern} = 16{,}5$ ГОП/с — величина на порядок меньше, чем производительность ядер БПФ и РО, поскольку операция SpMV требует случайного доступа к памяти при чтении элементов входного вектора.

Порядок матрицы — N — равен 150000, количество ненулевых элементов — $NZ = 36121058$. Тогда из 2.20 получаем сравнение:

$$36121058 \cdot 32000 \quad \vee \quad 4 \cdot 4 \cdot 150000 \cdot 16500. \tag{2.23}$$

Значение выражения слева на два порядка больше, чем выражения справа — это означает, что время пересылок пренебрежимо мало по сравнению со временем вычислений, и производительность оценивается как

$$PERF_{\mathrm{CG}} = 4\, Perf_{kern} \approx 66{,}0 \ \text{ГОП/с}\,. \tag{2.24}$$

На рисунке Б.2 для сравнения приводятся результаты замеров производительности референсной OpenMP-реализации CG на трёх процессорах: Xeon X5670 и Xeon E5-2660 от Intel и Opteron 6178 от AMD (на двух CPU). При вычислениях на CPU возникает та же трудность, что и на GPU: производительность ограничена скоростью чтения данных из памяти в случайном порядке. Падение скорости отчасти компенсируется работой системы кэш-памятей, однако с увеличением длины входного вектора растёт и количество кэш-промахов, и уже на задаче класса B производительность снижается. Можно ожидать, что вычисления на GPU покажут более высокую производительность и линейную масштабируемость с увеличением количества GPU.

2.3.4 Практическая реализация

До начала работы над алгоритмами из NAS PB автором было разработано OpenCL-ядро SpMV для матрицы в формате Sliced ELLpack и соответствующая процедура верхнего уровня.

Пусть (val, col, ptr) – входная матрица в формате Sliced ELLpack с параметром H, расположенная в глобальной памяти GPU. Вычисления в ядре огранизованы следующим образом. Одна WG обрабатывает один слой упакованной матрицы. Для этого требуется считать из глобальной памяти всего два элемента массива ptr. Более существенным является доступ к памяти при чтении из массивов val, col, а также x (входного вектора).

Каждая строка в слое обрабатывается одним или несколькими WI — их количество является параметром процедуры (r), в дополнение к H. В коде используется 2-мерное пространство индексов, WI с локальным номером (i,j) обрабатывает в i-м ряду ненулевые элементы с индексами j, $r + j$, $2r + j$, ... В формате Sliced ELLpack данные в массивах val и col располагаются таким образом, что доступ к ним получается регулярным: соседние WI считывают из памяти соседние элементы. Целесообразно выбирать значение параметра H равным степени 2 — тогда доступ к памяти получается также выравненным. Оптимальные значения H и r подбираются экспериментально, для каждой матрицы и каждого устройства.

Каждый WI накапливает частичную сумму произведений элементов матрицы, принадлежащих одному ряду, на соответствующие элементы входного вектора x, индексы которых считываются из массива col. В случае, если $r > 1$, для получения выходного элемента в конце выполняется редукция частичных сумм, с использованием локальной памяти. Результаты записываются в глобальную память, во фрагмент выходного вектора y, соответствующий слою; доступ к y также регулярный и выравненный.

Узким местом данного ядра является чтение из памяти элементов x. Доступ к ним происходит в случайном порядке от разных WI, при этом возникает большое количество конфликтов банков. Кроме того, один и тот же элемент вектора может использоваться несколько раз. В данной ситуации было бы желательно скопировать вектор в локальную память и производить чтение оттуда, однако в большинстве задач это невозможно, т.к. размер вектора существенно превышает объём локальной памяти одного вычислителя на GPU.

Чтобы задействовать локальную память, в ходе дальнейшей работы была предпринята попытка усовершенствовать формат упаковки матрицы. Новый формат был назван «Framed ELLpack». В этом формате матрица разбивается не на слои, а на блоки размера $H_y \times H_x$, такого, что вектор длины H_x помеща-

ется в локальную память одного вычислителя. Упаковка матрицы в форматах Sliced и Framed ELLpack наглядно изображена на рисунке 2.7: пример портрета исходной матрицы и её промежуточные представления в обоих форматах — слои/блоки, которые получаются после разбиения, сохранения номеров столбцов для всех ненулевых элементов и дополнения нулями. Белым обозначены дополнительные нули; заштрихованные области соответствуют нулевым элементам, которые не входят в упакованную матрицу; в формате Framed ELLpack тёмно-серым обозначены полностью нулевые блоки, которые также не включаются в упакованную матрицу.

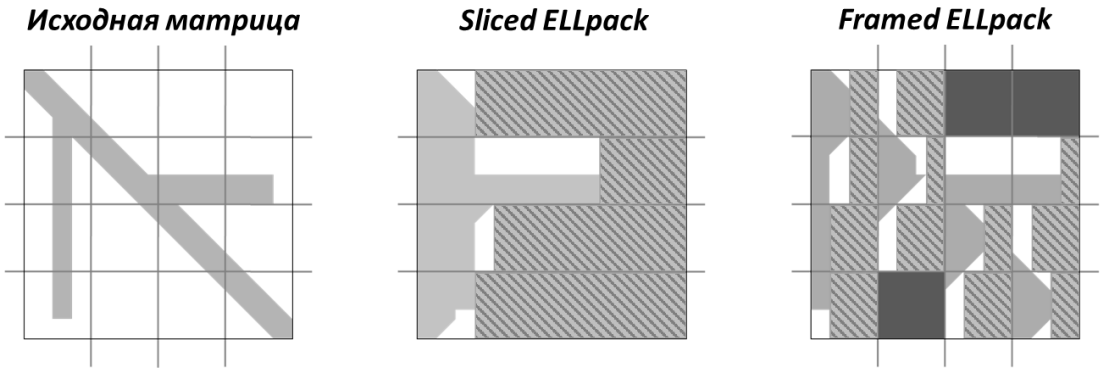

Рисунок 2.7 — Форматы упаковки Sliced и Framed ELLpack

В ходе вычислений в ядре входной вектор разбивается на части; доступ к каждой части остаётся случайным, но не в глобальной памяти, а в локальной, поэтому происходит быстрее. Пустые блоки в вычислениях не участвуют. Минусом данного формата является большее количество дополнительных нулей, чем в формате Sliced ELLpack.

Помимо процедуры SpMV, автором была разработана гетерогенная процедура на OpenCL, реализующая алгоритм CG в целом — на основе референсного кода NPB.

Для перенесения алгоритма целиком на один GPU потребовался, помимо ядра SpMV, набор простых вспомогательных ядер, реализующих различные векторные и поэлементные операции, которые встречаются в алгоритме: скалярное произведение, поэлементное умножение на коэффициент и др. (всего 5 ядер). Вычисление невязки и её нормы на последнем шаге метода сопряжённых градиентов объединено в одно ядро — SpMV, дополненное рядом операций. Ещё 5 ядер предназначены для установки исходных значений всех необходимых констант и векторов перед началом вычислений, вместо дополнительных загру-

зок в память сопроцессора. Выполнение всех вычислений непосредственно на GPU позволяет свести все пересылки к начальной загрузке входных данных и выгрузке результатов.

Второй этап разработки — реализация алгоритма CG для нескольких сопроцессоров. Здесь используется только ядро SpMV, все прочие вычисления выполняются на CPU (в референсном коде) — это позволяет избежать большого количества мелких пересылок данных между GPU и системной памятью.

В референсной MPI-реализации матрица разбивается на блоки, каждый из которых обрабатывается своим MPI-процессом. В разработанной процедуре один процесс управляет одним гибридным узлом со всеми его GPU, при этом каждый блок матрицы по отдельности упаковывается в формате Sliced ELLpack, и для него применяется схема распараллеливания, описанная выше. Замеры подтвердили, что время выгрузки выходных векторов с сопроцессоров пренебрежимо мало по сравнению со временем выполнения прочих операций: вычислений в ядре, вычислений на CPU и MPI-обменов, — поэтому перекрытие вычислений с пересылками не реализовывалось.

Чтобы добиться максимально возможной производительности, для различных GPU были выбраны оптимальные параметры для обоих форматов упаковки матрицы, путём полного перебора по всем допустимым значениям. Кроме того, ко всем вспомогательным вычислениям, которые выполняются на CPU, было применено распараллеливание OpenMP — по аналогии с референсной OpenMP-версией кодов. Таким образом, разработанная гетерогенная процедура задействует три средства параллелизации:

– массивный параллелизм при вычислении SpMV на GPU;
– OpenMP для векторных операций на CPU;
– MPI для обменов данными между узлами.

В итоговую версию кода, как и для алгоритма MG, была интегрирована процедура для одного сопроцессора.

2.3.5 Результаты тестирования

Как отмечалось выше, производительность SpMV зависит от структуры входной матрицы. Для тестирования разработанной процедуры использовался стандартный набор из 14 матриц, полученных из различных практических задач (см., например, [44]). Портреты матриц схематически изображены на рисунке 2.8.

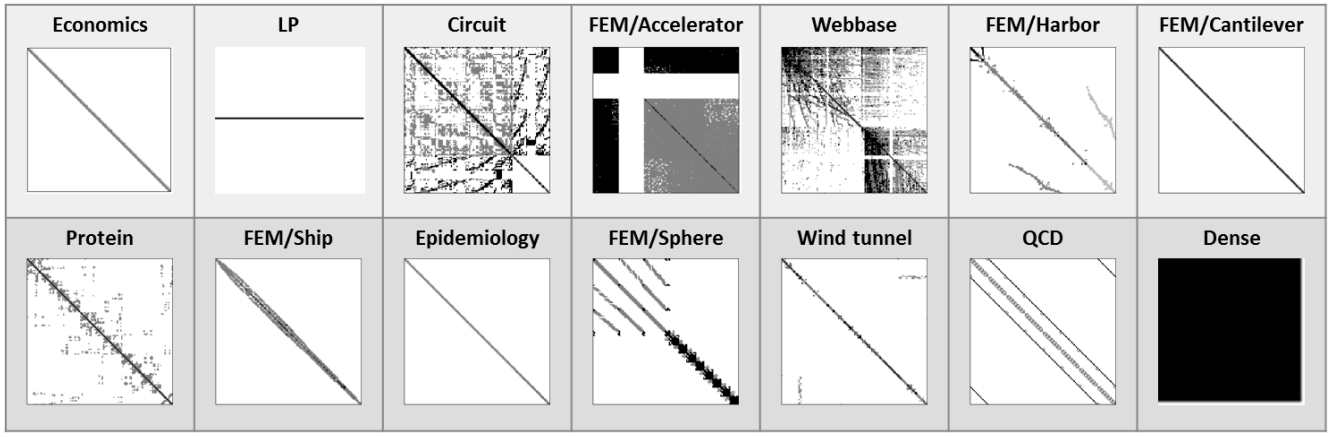

Рисунок 2.8 — Набор матриц для тестирования SpMV

На рисунке 2.9 приводится диаграмма с результатами замеров производительности OpenCL-процедуры на различных GPU и, для сравнения, производительности библиотечной процедуры SpMV для CPU от Intel и процедуры на CUDA для GPU от NVIDIA. Для каждого ускорителя указано значение bw_{CP} — пиковая скорость доступа к глобальной памяти (соответственно, для CPU — скорость доступа к системной памяти). Формат упаковки — Sliced ELLpack.

В [45] показано, что использование GPU для вычисления SpMV в среднем повышает производительность в несколько раз, по сравнению с оптимизированными реализациями для CPU. Соответствующая процедура была реализована на CUDA (см. [44]). Сопоставимые результаты (с учётом разницы в bw_{CP}) показала и разработанная переносимая процедура на OpenCL; производительность вычислений при этом напрямую зависит от производительности подсистемы памяти GPU.

Ядро SpMV было также протестировано на матрицах из алгоритма CG, на GPU GeForce TITAN от NVIDIA и FirePro w9100 от AMD. Полученные данные продемонстрировали существенное влияние параметров H и r на производи-

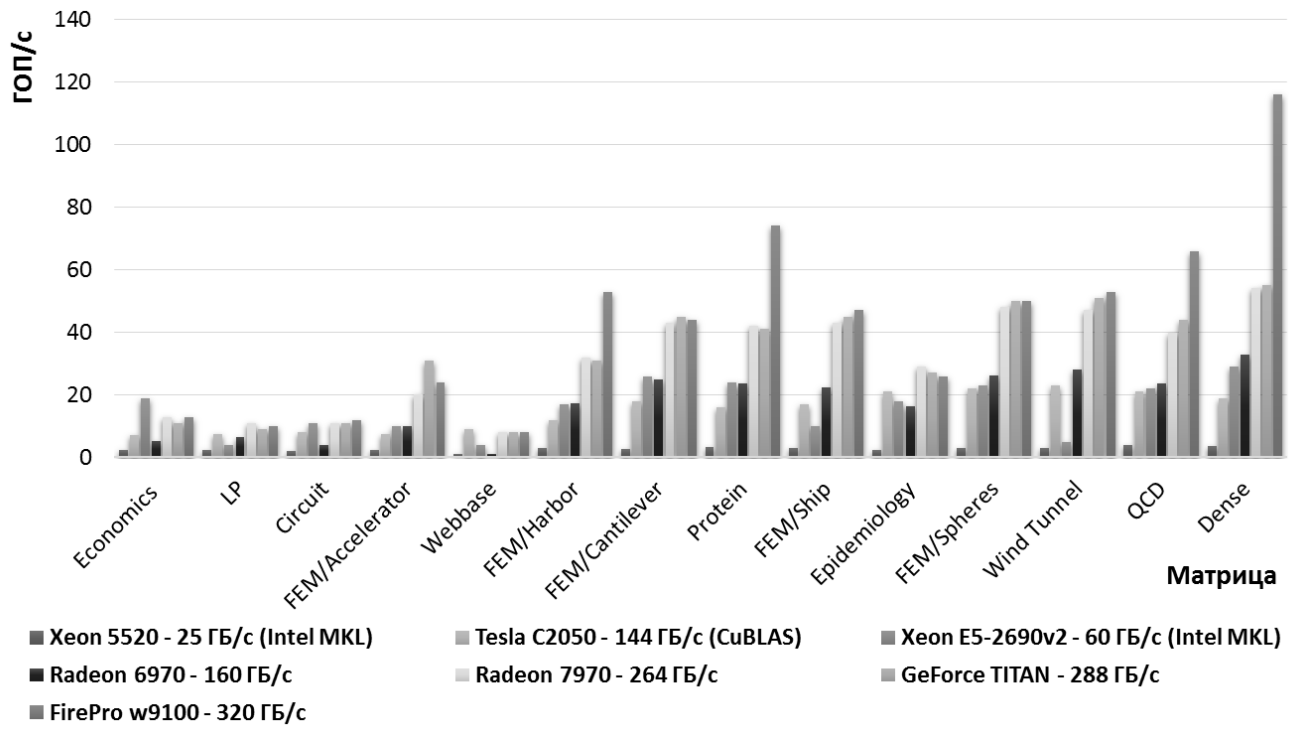

Рисунок 2.9 — Ядро процедуры SpMV на OpenCL:
сравнение с Intel MKL и CUDA

тельность ядра. Производительности, в МОП/с, и соответствующие оптимальные значения параметров приводятся в таблице Б.1.

На диаграмме 2.10 сравнивается производительность разработанной автором реализации алгоритма CG на одном ускорителе и лучшие результаты на CPU (см. рис. Б.2).

Глобальной памяти всех трёх рассматриваемых GPU достаточно для решения задач классов до C включительно. Начиная с класса B производительность вычислений на GPU в среднем выше, чем на CPU. На классе C производительность ниже, чем на B, по той же причине, что и на CPU: задача перестаёт помещаться в кэш.

На диаграмме 2.11 приводятся результаты замеров производительности двух версий гетерогенной процедуры: с упаковкой матрицы в формате Sliced и Framed ELLpack. Замеры производились на одном ускорителе, на узлах трёх различных конфигураций:

1. два CPU Xeon E5-2670 и GPU GeForce TITAN;
2. два CPU Xeon X5670 и GPU Tesla C2050;
3. два CPU Xeon E5-2690v2 и GPU FirePro w9100.

Здесь и далее значения параметров ядра SpMV — оптимальные.

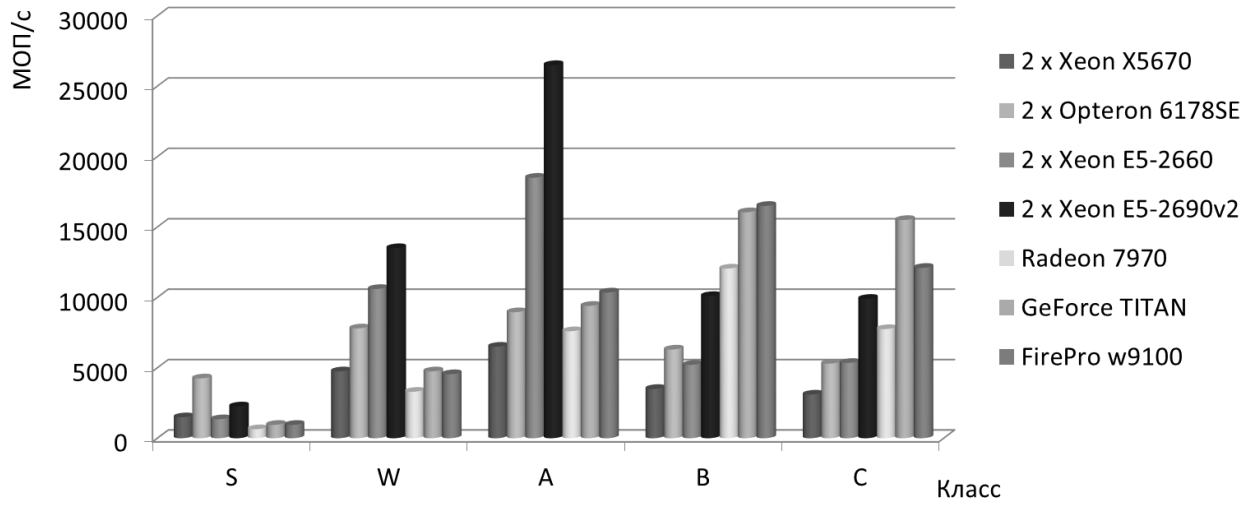

Рисунок 2.10 — CG: лучшие результаты на CPU и процедура на одном GPU

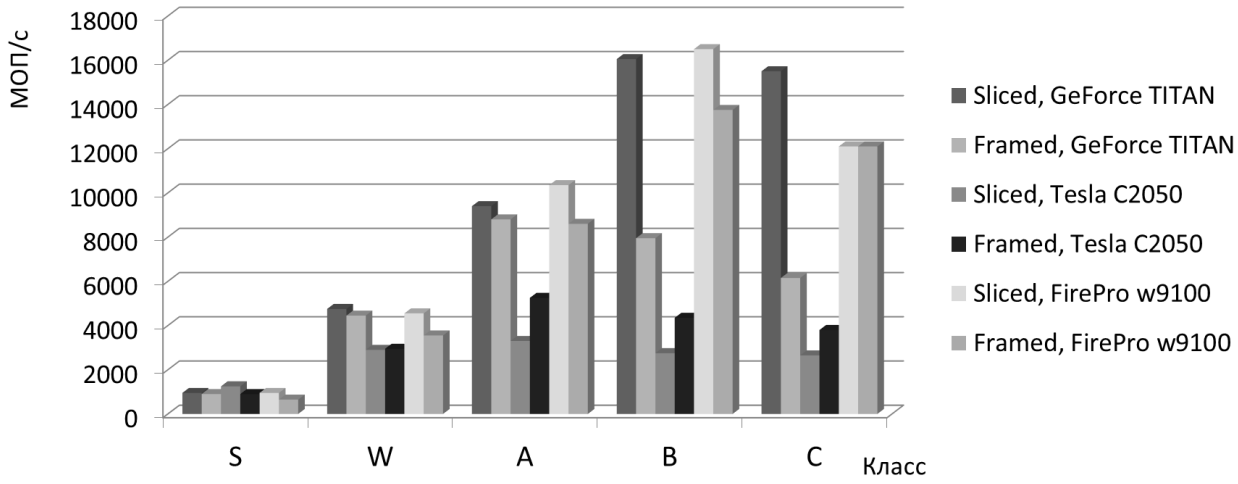

Рисунок 2.11 — CG: сравнение форматов упаковки матрицы

Можно сделать вывод, что использование формата Framed ELLpack целесообразно на GPU предыдущих поколений — например, Tesla C2050. В последние годы производителями ведутся работы по аппаратному ускорению случайного доступа к глобальной памяти, за счёт усовершенствования системы кэш-памятей. В результате на таких GPU, как GeForce TITAN и FirePro, выигрыш от использования локальной памяти становится менее значительным и покрывается потерями на избыточные вычисления с дополнительными нулями.

Результаты замеров производительности гетерогенной процедуры на различном количестве ускорителей приводятся на рисунке 2.12. Формат упаковки матрицы — Sliced ELLpack. При тестировании на двух различных FirePro вычислительная работа распределялась между ними поровну.

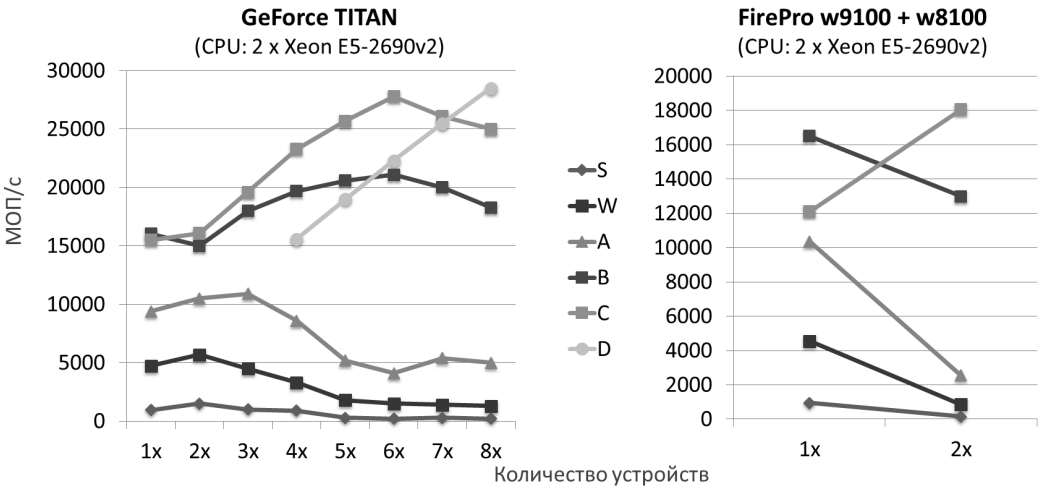

Рисунок 2.12 — Гетерогенная процедура CG: масштабируемость на узле

Падение производительности на двух FirePro, до класса B, обусловлено, во-первых, потерями за счёт обменов данными и вычислений на CPU, а во-вторых, меньшей вычислительной мощностью w8100. В целом использование GPU оправданно при обработке больших объёмов данных; маленьких объёмов не достаточно, чтобы задействовать все ресурсы и возможности параллелизма графического ускорителя.

На задачах больших классов реализация НИИСИ показывает хорошую масштабируемость: например, на классе D, на GeForce TITAN, масштабируемость близка к линейной, с коэффициентом 0,8 (результаты для количества устройств менее 4 отсутствуют, т.к. их суммарной памяти не хватает для хранения необходимого объёма данных). Помимо масштабируемости, реализация показывает заметно бóльшую производительность вычислений, чем референсный код на CPU:

– 5400 МОП/с на двух современных CPU с OpenMP;
– 28500 МОП/с на восьми современных GPU.

В таблице 2.4 результаты, полученные на одном и на четырёх GeForce TITAN, сравниваются с теоретическими оценками, выведенными в разделе 2.3.3. Рассматривается задача класса C. Теоретическая оценка определяется производительностью ядра SpMV.

Эффективность реализации на одном GPU, по сравнению с теоретической оценкой, очень высока — это означает, что основное время приходится на ядро SpMV, а доля векторных операций незначительна. При распределении работы между несколькими GPU, однако, возникают дополнительные накладные рас-

Таблица 2.4 — CG, класс C: сравнение предварительных оценок и результатов

Количество GPU	**Теория (ГОП/с)**	**Практика (ГОП/с)**	**Эффективность (%)**
1x	16,5	15,5	94
4x	66,0	23,3	35

ходы на разбиение матрицы и пересылки; кроме того, выполнение векторных операций на CPU требует больше времени, чем на GPU. В результате эффективность вычислений снижается.

На рисунке 2.13 приводятся результаты запусков гетерогенной процедуры CG на суперкомпьютере K100: до 24 ускорителей Tesla C2050, по три на узле; CPU на каждом узле — два Xeon X5670. Одному узлу соответствует один MPI-процесс, np — общее количество процессов. Для сравнения приводится также производительность референсного кода на CPU.

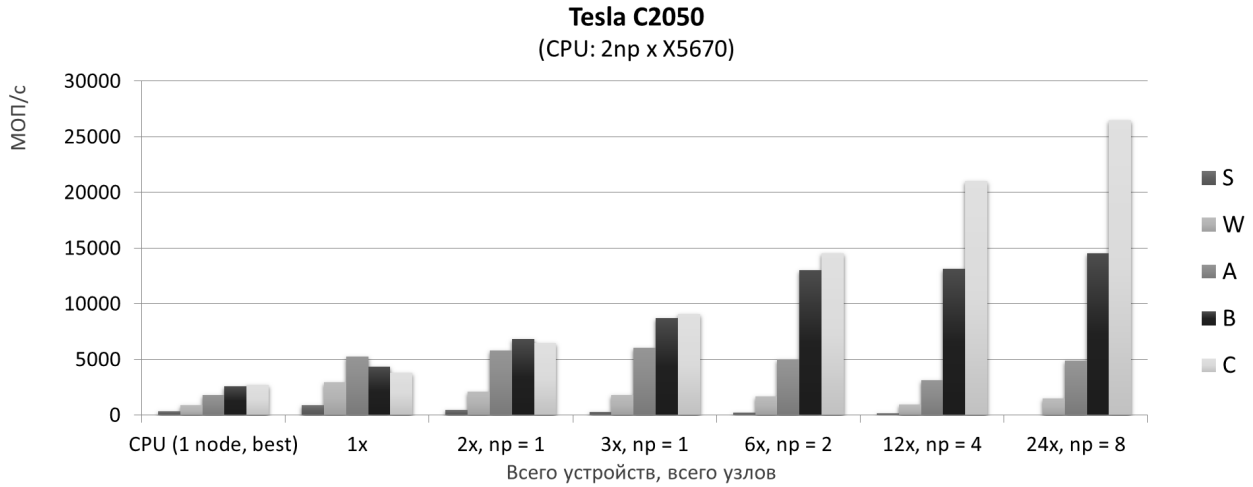

Рисунок 2.13 — Гетерогенная процедура CG: производительность на K100

Задача класса C масштабируется до 24 ускорителей, производительность составляет:

– 9,1 ГОП/с на 3-х Tesla C2050,

– 26,5 ГОП/с на 24-х.

Для сравнения, процедура CG, вошедшая в библиотеку SnuCL, показывает достаточно скромный результат ([128]): на классе C процедура CG на 1-2 GPU оказывается медленнее, чем на одном ядре CPU. Использование 16 GPU

даёт ускорение в 3,5 раза относительно одного ядра, 32 GPU — в 5 раз. Это можно объяснить тем, что код ядер в SnuCL был сгенерирован автоматически из кода процедур для универсального процессора и не использует специального формата упаковки матрицы. В [152] процедура CG на GPU эффективнее, чем на CPU, только для больших классов задачи (B, C), и менее эффективна, чем SnuCL CG.

2.4 Выводы

Наблюдения других исследователей о том, что производительность большинства расчётных задач на гибридных системах на базе GPU определяется скоростью доступа к памяти на том или ином уровне, в данной главе были обобщены и формально обоснованы при помощи построенного метода оценки производительности. Применимость метода подтверждается результатами замеров производительности разработанных процедур.

Так, производительность 2- и 3-мерного БПФ больших массивов ограничена пропускной способностью канала между сопроцессорами и системной памятью.

Сеточные вычисления связаны с пересылками меньших объёмов данных, и параметры современных гибридных систем позволяют реализовать их эффективно, при условии тщательной оптимизации обменов данными и перекрытия их с вычислениями. Кроме того, требуется организация эффективного доступа к памяти в ядре — например, с использованием локальной памяти на GPU. Процедура MG для гибридных кластеров с GPU, разработанная с учётом этих соображений, является переносимой и показывает на больших классах задачи более высокую производительность, чем аналогичные процедуры, описанные в открытых публикациях.

Производительность вычислений с разреженными матрицами, в случае если не требуется пересылок самой матрицы между системной памятью и сопроцессорами, определяется ядром. Производительность самого ядра, в свою очередь, определяется скоростью доступа к памяти в случайном порядке, которая как правило существенно ниже пиковой. Различные форматы упаковки

матрицы позволяют в той или иной степени снизить, однако не полностью исключить, влияние этого фактора — как правило, за счёт увеличения других накладных расходов, например, количества дополнительных нулей. Разработанные переносимые процедуры SpMV для GPU и CG для гибридных кластеров показывают производительность, сопоставимую и в ряде случаев превосходящую результаты, представленные в открытых публикациях.

Глава 3. Исследование реализаций процедур на гибридных процессорах КОМДИВ

В данной главе рассматриваются оптимизированные реализации для процессоров ВМ7 и ВМ9, использующие сопроцессор СР2. В терминах модели, построенной в разделе 1.3, сопроцессором является одна из четырёх вычислительных секций СР2. Ядро — вычислительная процедура, которая выполняется в одной секции СР2, причём входные и выходные данные располагаются в локальной памяти. Все секции параллельно исполняют одно и то же ядро.

3.1 Процедура БПФ

Сотрудниками НИИСИ и ИСП РАН были ранее разработаны процедуры для процессора ВМ7, вычисляющие одномерное БПФ набора коротких векторов, т.е. векторов такой длины, что один вектор может быть целиком обработан в локальной памяти одной секции СР2. Рассматриваются только длины, являющиеся степенями 2, и БПФ вычисляется по алгоритму Кули-Тьюки. В данном разделе оценки будут проводиться для «длинного» БПФ — одномерного БПФ векторов, которые не помещаются в локальную память одной секции СР2.

3.1.1 Схема вычислений

Пусть $N = 2^n = N_1 \cdot N_2$, $\bar{x}$ — входной вектор БПФ длины N, $\bar{y}$ — выходной. Вектор $\bar{x}$ можно представить в виде матрицы X порядка $N_1 \times N_2$, в которой элементы вектора располагаются по строкам, а вектор $\bar{y}$ — в виде матрицы Y порядка $N_2 \times N_1$, также по строкам. Пусть БПФ длины N_1 и N_2 может быть вычислено по короткому алгоритму. Тогда алгоритм «БПФ за 4 шага» состоит в следующем:

1. набор коротких БПФ матрицы X по столбцам: N_2 БПФ длины N_1;

2. поэлементное домножение матрицы на экспоненты;
3. короткое БПФ по строкам: N_1 БПФ длины N_2;
4. транспонирование.

На выходе получается матрица Y. Т.о., вычисление длинного одномерного БПФ сводится к вычислению двумерного БПФ с некоторыми дополнительными операциями.

Входные и выходные данные располагаются в системной памяти. Первые два шага алгоритма можно объединить, если после вычисления БПФ столбцов на первом шаге, не выгружая результаты, передать в локальную память соответствующий набор коэффициентов, на которые нужно домножить эти результаты на втором шаге. Коэффициенты зависят от N, но не от конкретных входных данных, поэтому могут быть вычислены заранее и храниться в постоянном массиве в системной памяти. Контроллер DMA процессоров ВМ7/9 позволяет произвести транспонирование матрицы в процессе загрузки и выгрузки данных на третьем шаге, за счёт использования двух режимов чтения/записи данных в память секций CP2: последовательного и с чередованием. Таким образом, все вычисления для каждой матрицы разделяются на два этапа, на каждом из которых происходит в сумме одна загрузка матицы в локальную память и одна выгрузка.

Для простоты при получении теоретической оценки умножение можно отнести к накладным расходам и оценить производительность двумерного БПФ. Схема его вычисления — постраничная, аналогична описанной в разделе 2.1.1. В [33, гл. 5] были выведены более точные оценки, с учётом умножения на коэффициенты, разгона/торможения циклов и множественности, т.е. вычисления БПФ набора векторов.

Пересылку данных по DMA выгодно осуществлять по возможности большими порциями: чем больше объём данных, тем ближе скорость пересылки к пиковой. При этом для обработки одной страницы используется только половина памяти секций CP2, благодаря чему становится возможным совмещение вычислений с пересылками.

3.1.2 Вычислительные ядра для CP2

Оценим производительность ядра короткого БПФ на одной секции CP2 по минимальному необходимому количеству инструкций и, соответственно, тактов рабочей частоты. При этом не будут учитываться различные накладные расходы на разгон/торможение циклов и другие вспомогательные операции в ядре. Конвейеризация позволяет запускать на каждом такте как арифметические команды, так и команды загрузки данных из локальной памяти в регистры и выгрузки обратно, поэтому можно упрощённо считать, что время исполнения каждой команды — один такт. В одной VLIW-инструкции можно совместить команду обменов данными с регистрами и арифметическую команду — соответственно, время работы ядра оценивается по количеству команд того или другого типа, которых требуется больше.

Вычисление БПФ длины N требует в сумме $N/2 \log N$ арифметических команд «бабочка Фурье» (`cbutterfly`). Из локальной памяти необходимо загрузить, как минимум, весь входной вектор, т.е. N 64-битных слов, и выгрузить весь выходной вектор. Бит-реверсирование элементов входного вектора можно осуществить «на лету», в ходе загрузки: CP2 поддерживает бит-реверсивный режим адресации в локальной памяти. Элементы при этом загружаются по одному. Выгрузка элементов может осуществляться парами: система команд CP2 включает команды загрузки и выгрузки двойного слова (128 бит). Необходимые коэффициенты Фурье считываются из ПЗУ — это не требует дополнительных команд загрузки. Таким образом, в сумме необходимо N команд загрузки и $N/2$ команд выгрузки. Отсюда, при $N > 8$, количество тактов работы ядра БПФ определяется количеством бабочек Фурье: $N/2 \log N$. Количество арифметических операций в алгоритме БПФ — в 10 раз больше, $5N \log N$, поэтому производительность ядра оценивается сверху по формуле:

$$Perf_{kern}{}^{peak} \approx 10F, \tag{3.1}$$

где F — тактовая частота процессора. Оценка не зависит от N.

Для вычисления двумерного БПФ требуются ядра короткого БПФ длин N_1 и N_2. Вообще говоря, для заданного N значения N_1 и N_2 можно выбирать произвольными, но из соображений сбалансированности работы на первом и

втором этапе вычислений представляется разумным выбирать N_1 и N_2 по возможности равными. В рамках библиотеки БЦОС было реализовано длинное БПФ четырёх длин: $N = 8192, 16384, 32768$ и 65536 — для этого использовались ядра короткого БПФ длин 64, 128 и 256.

Более точно, был разработан набор из 6 специализированных ядер короткого БПФ: по 3 ядра для первого и для второго этапа вычислений. Это связано с использованием двух режимов пересылки данных по DMA. На первом этапе в локальной памяти обрабатываются вектора-столбцы, при этом элементы каждого из них располагаются в памяти секции с шагом — соответственно, в ядре элементы каждого вектора зачитываются в регистры с тем же шагом, и так же записываются элементы выходных векторов. В ядрах для второго этапа чтение входных векторов происходит аналогично, но после вычисления БПФ каждый вектор записывается в непрерывную область локальной памяти. Это позволяет выгрузить страницу результатов как набор строк выходного массива, и таким образом осуществить транспонирование (подробнее см. в [33, разд. 5.3]). Все описанные отличия касаются индексации элементов входного и выходного массивов и не влияют на производительность: на CP2 скорость доступа к локальной памяти не зависит от порядка адресов.

Производительность готовых ядер можно оценить по коду на языке ассемблера, либо по логу работы ядра на эмуляторе. Она составляет ту же долю от пиковой, вычисленной выше, что и количество тактов, на которых запускается команда `cbutterfly`, от общего количества тактов. В таблице Б.2 приводятся данные о количестве команд `cbutterfly` в коде ядер БПФ длины 64, 128 и 256 и соответствующие оценки производительности ядер на BM7 и BM9, в МОП/с. Графа «%» означает эффективность ядра в процентах от пика: на BM7 $F = 200$ МГц и $Perf_{kern}{}^{peak} = 2000$ МОП/с, на BM9 $F = 1000$ МГц и $Perf_{kern}{}^{peak} = 10000$ МОП/с.

Высокая эффективность — порядка 80% — достигается за счёт «плотного» кода ядра, в котором в VLIW-инструкциях совмещаются арифметические операции и операции чтения/записи данных в локальную память.

В дальнейшем, путём перебора и замеров производительности, были подобраны следующие оптимальные сочетания N_1 и N_2:

$$\begin{aligned} N = 8192: &\quad N_1 = 64,\ N_2 = 128, \\ N = 16384: &\quad N_1 = 64,\ N_2 = 256, \\ N = 32768: &\quad N_1 = 128,\ N_2 = 256, \\ N = 65536: &\quad N_1 = 256,\ N_2 = 256. \end{aligned}$$

3.1.3 Оценки производительности

Оценим производительность двумерного БПФ, а значит и длинного одномерного, по аналогии с разделом 2.1.2. В формулы 2.1 и 2.2 — время пересылок и время работы ядра при постраничном вычислении набора одномерных БПФ — подставляются N_1 и N_2. На CP2 вычисления выполняются с одинарной точностью, поэтому одно комплексное число занимает 8 байт. С учётом этого, сравнение 2.3 принимает вид:

$$5\,BW \log N_j \quad \vee \quad 16K \cdot Perf_{kern}, \tag{3.2}$$

где $j \in \{1,2\}$ — номер этапа.

Производительность длинного БПФ оценивается как общее количество операций, делённое на суммарное время на первый и второй этап вычислений. Оценки получаются путём подстановки в формулы значений $Perf_{kern}$ из таблицы Б.2, оптимальных пар N_1 и N_2, приведённых выше, и параметров модели из таблицы А.2 (для рабочей частоты BM9 1000 МГц и частоты памяти 667 МГц).

Проведённые выкладки показали следующее:

1. на BM7 вычисления преобладают над пересылками для коротких БПФ длины 256, хотя в целом для трёх рассмотренных длин время пересылок и время вычислений — величины одного порядка;
2. на BM9 для всех длин пересылки требуют существенно больше времени, чем вычисления.

Последнее объясняется тем, что на ВМ9 BW увеличено по сравнению с ВМ7 в 3,1 раза, в то время как тактовая частота CPU, а значит и производительность ядер, возросла в 5 раз. На ВМ7 производительность CP2 и DMA была сбалансирована, т.к. совпадали рабочие частоты памяти и CPU, а на ВМ9 CP2 имеет перевес.

Полученные теоретические оценки приводятся в следующем разделе, в таблицах 3.1 и 3.2. Ожидаемый прирост производительности БПФ на ВМ9 по сравнению с ВМ7 — в среднем в 3,1 раза, что совпадает с разницей в значениях BW.

3.1.4 Результаты тестирования

Разработанная процедура множественного длинного БПФ была протестирована на процессоре ВМ7 и на одном ядре процессора ВМ9.

В таблице 3.1 приводятся данные о производительности процедуры на одном ВМ7. Рассматривалось БПФ $M = 256$ векторов — типичное значение в реальных задачах. В той же таблице для каждого N приводится теоретический максимум производительности, а также эффективность реальной процедуры в процентах от этого максимума.

Таблица 3.1 — Производительность длинного БПФ на ВМ7

N	**Теор. максимум, МОП/с**	**Производ-ть, МОП/с**	**Эфф-ть, %**
8192	5484	3108	57
16384	5895	3207	54
32768	6316	3494	55
65536	6725	3768	56

Итак, эффективность процедуры длинного БПФ составляет 56% от теоретического максимума. Часть потерь связана с домножением на коэффициенты на первом этапе вычислений, которое включает дополнительную загрузку в локальную память массивов коэффициентов, сопоставимых по объёму с входными данными; время на само умножение на CP2 несущественно по сравнению со временем вычисления БПФ — таким образом, усиливается зависимость производительности алгоритма от скорости пересылок данных по DMA.

Вообще говоря, на производительность процедуры влияет множественность, то есть, значение M: чем оно больше, тем выше производительность, т.к. доля накладных расходов в общем времени работы уменьшается. Для длинного БПФ, однако, это влияние невелико, поскольку БПФ каждого вектора само по себе подразумевает обработку в цикле большого количества страниц на CP2. Например, уже при $M = 8$ БПФ длины 65536 имеет примерно ту же производительность, что и при $M = 256$.

Данные о производительности той же процедуры на одном ядре BM9 и одном сопроцессоре CP2 приводятся в таблице 3.2. В последней колонке указан также прирост производительности относительно BM7.

Таблица 3.2 — Производительность длинного БПФ на BM9

N	**Теор. макс., МОП/с**	**Произв-ть, МОП/с**	**Эфф-ть, %**	**Прирост**
8192	17062	8920	52	x2,9
16384	18375	10333	56	x3,2
32768	19687	10418	53	x3,0
65536	21000	9809	47	x2,6

Разница в производительности по порядку соответствует разнице в частотах памяти, а не ядра — это подтверждает тезис о том, что производительность вычисления длинного одномерного БПФ определяется именно производительностью памяти и канала DMA.

Результаты показали некоторое снижение эффективности вычислений, по сравнению с ВМ7, особенно заметное на $N = 65536$. Это объясняется различиями типов памяти — DDR2 на ВМ7 и DDR3 на ВМ9 — и соответствующих контроллеров (в частности, на ВМ9 контроллер поддерживает 2 параллельных транзакции к памяти, в то время как на ВМ7 — 8 транзакций). В алгоритме длинного БПФ используется схема доступа, при которой из памяти вычитываются непрерывные строки малой длины, при этом шаг между началами строк в несколько раз больше, чем длина строки. В среднем чем больше N, тем меньше длина строки и больше шаг. При такой схеме доступа скорость оказывается ниже пиковой, которая достигается при чтении одной большой непрерывной области, и снижение скорости более существенно на DDR3, чем на DDR2. Скорости некоторых типов пересылок по DMA, которые встречаются при вычислении длинного БПФ, сравниваются в таблице Б.3. Длина строки и шаг между началами строк указаны в комплексных элементах (8 байт), DMA_GET — загрузки в локальную память CP2, DMA_PUT — выгрузки обратно. И на ВМ7, и на ВМ9 наблюдается несимметричность скоростей пересылок по DMA в двух направлениях. Она обусловлена различиями в программной модели дескрипторов типа DMA_GET и DMA_PUT. Эти различия отражаются и на итоговой производительности вычисления БПФ.

Дальнейшее существенное повышение производительности БПФ возможно только при условии увеличения скорости пересылок данных, либо при увеличении объёма локальной памяти CP2, что позволит для большего количество длин производить вычисления по более эффективной схеме, без использования алгоритма «БПФ за четыре шага».

3.1.5 Процедура свёртки

Помимо БПФ, автором была также разработана процедура свёртки для ВМ7/9 (см. [33]), вошедшая в состав БЦОС: вычисляется свёртка вектора $\bar{h}$ длины L с каждым из M векторов $\bar{x}$ длины N из набора X. Результаты образуют набор Y из M векторов $\bar{y}$ длины $N + L - 1$.

Процедура работает с векторами $\bar{h}$ и $\bar{x}$ таких длин N и L, что $\tilde{N} \leqslant 65536$, где $\tilde{N}$ — ближайшая сверху к $N + L - 1$ степень двойки. Свёртка h с каждым вектором вычисляется за $O(\tilde{N} \log \tilde{N})$ операций, через БПФ, по следующему известному алгоритму (см., например, [51, ch. 13]):

1. дополнение входных векторов нулями до длины $\tilde{N}$;
2. прямое БПФ длины $\tilde{N}$ обоих векторов;
3. поэлементное перемножение полученных Фурье-образов;
4. обратное БПФ длины $\tilde{N}$;
5. поэлементное домножение результата на $1/\tilde{N}$.

Во множественной процедуре дополнить вектор $\bar{h}$ нулями и вычислить его БПФ достаточно один раз. Всего в реализации используется 4 различных схемы вычисления свёртки, в зависимости от длины выходного вектора.

К описанной процедуре свёртки была применена та же методология оценки производительности, что и к БПФ. Оценки показали:

– и на BM7, и на BM9 для коротких векторов производительность определяется производительностью ядер БПФ на CP2;
– для длинных векторов на обоих процессорах производительность ограничена производительностью пересылки данных; пересылки преобладают по времени над вычислениями на большинстве этапов работы;
– для промежуточных длин ($\tilde{N} = 2048$ и 4096) на BM7 на всех этапах работы преобладают вычисления; на BM9 на ряде этапов преобладают пересылки, но в целом время вычислений на CP2 и время пересылок приблизительно равны.

В таблице Б.4 приводятся результаты замеров производительности свёртки на BM7 и BM9 и эффективность в процентах от теоретического максимума. Для наглядности рассматривается только случай $N = L$. Тестирование показало, что для малых длин существенное влияние на производительность оказывает множественность, поэтому для $N \leqslant 2048$ замеры проводились при одновременной обработке 4096 векторов, а для $N \geqslant 4096$ — при обработке 256 векторов. При малых длинах N становится также заметной доля накладных расходов в общем времени вычислений, поэтому и эффективность ниже, чем при свёртке векторов больших длин.

В целом процедура свёртки сопоставима с «чистым» БПФ по производительности и эффективности.

3.1.6 Сравнение с производительностью на других процессорах

В таблице Б.5 приводятся производительности БПФ различных длин, в т.ч. «коротких», и сравнительные характеристики различных процессоров:

- для ВМ7/9 приводятся производительности разработанной автором процедуры множественного длинного БПФ и БПФ длины 4096, а также процедуры множественного короткого БПФ из БЦОС. На ВМ9 замеры проводились на модуле с частотой процессорного ядра 1200 МГц и частотой памяти 800 МГц;
- для процессора обработки сигналов TMS320C6678 приводятся производительности на основе данных из открытых источников [98; 103; 146], приведённые к 8 ядрам и к одной частоте процессора и памяти;
- для процессора TigerSHARC (ADSP-TS201S) производительности вычислены по приведённым в [154] данным о количестве тактов на БПФ, с учётом рабочей частоты 600 МГц;
- для одного ядра универсального процессора Intel Xeon E5-2670v1 и для 16 ядер двух процессоров были проведены замеры производительности процедуры из библиотеки FFTW, использующей OpenMP;
- для STI Cell приводятся данные из [124] о производительности на двух процессорах в составе IBM BladeCenter QS20 [86].

Прочерк означает отсутствие данных.

Сравнение наглядно демонстрирует, что производительности БПФ малых длин соотносятся как частоты CPU и количество параллельных потоков исполнения, т.е. производительность алгоритма определяется пиковой производительностью процессора. Для БПФ больших длин, напротив, производительности соотносятся как частоты памяти и количество каналов доступа к ней, т.е. производительность определяется пропускной способностью памяти. В частности, производительность одного ядра ВМ9 (с одним сопроцессором СР2) сопоставима с производительностью 8 ядер TMS320C6678 в пересчёте на один канал памяти.

Полных данных о производительности БПФ на Эльбрус в открытых источниках найти не удалось. Оптимизации, которые применяются при вычислении БПФ, описаны в [15]. Используется асинхронная подкачка данных, VLIW-

команды и векторизация: обработка элементов парами — приёмы, аналогичные использованным для КОМДИВ. В [6] показано, что на типовой задаче ЦОС одно ядро Эльбрус-2С+ (без сопроцессоров) и одно ядро TMS320C6678 демонстрируют сопоставимую производительность, несмотря на различие в тактовых частотах. Авторы отмечают необходимость более детального анализа влияния особенностей функционирования процессоров на производительность процедур. В [20] приводится пиковая производительность на одном ядре Эльбрус-4С: для одинарной точности она составляет 8 ГОП/с, что соответствует по порядку производительности одного ядра Xeon.

В [124] отмечается, что производительность БПФ на Cell ограничена скоростью пересылки данных. Соответственно, эффективное вычисление подразумевает двойную буферизацию: совмещение вычислений с пересылками данных на SPE и обратно, а также обменов данными с регистрами и арифметических инструкций на самом SPE — аналогично вычислениям на СР2 на КОМДИВ. Тем не менее, даже при условии этих оптимизаций, пересылки преобладают по времени над вычислениями. С этой точки зрения процессор ВМ7 сбалансирован лучше: пересылки могут быть скрыты за вычислениями уже на наборе одномерных БПФ длины 256; на ВМ9 это справедливо только для длины 4096.

3.2 Процедура NPB MG

3.2.1 Схема вычислений

Параллельная реализация алгоритма MG для СР2 в целом устроена по тем же принципам, что и на других архитектурах. На одном процессоре параллелизация происходит на двух нижних уровнях, описанных в [39]: параллельная обработка на четырёх секциях СР2 и SIMD-вычисления в ядре (обработка пар элементов в одном регистре, подробнее см. далее в разд. 3.2.2). Верхний уровень — декомпозиция на блоки с обменам сообщениями — используется при реализации на многопроцессорном комплексе (см. разд. 3.2.6).

Ключевое отличие ВМ7/9 от гибридных узлов, рассмотренных выше в разделе 2.2.1, состоит в том, что локальной памяти СР2 не достаточно для хранения в процессе вычислений всех необходимых данных, даже в задаче NPB MG класса S. Соответственно, вместо описанной схемы с обменами гранями используется разбиение массива на небольшие блоки, которые обрабатываются постранично.

Реализация алгоритма MG представляет собой последовательный вызов процедур РО и дополнительных копирований граней. Входной массив для каждого РО располагается в системной памяти, вместе с дополнительными гранями. Набор блоков подразделяется на несколько страниц по K блоков. При обработке одной станицы блоки с дополнительными гранями загружаются на сопроцессоры (секции СР2), по одному на каждый. Далее на всех сопроцессорах параллельно вызывается ядро РО. Полученные блоки без дополнительных граней выгружаются в выходной массив в системной памяти. После обработки всех страниц в системной памяти оказывается целый выходной массив, в котором необходимо произвести копирование граней: скопировать каждую вычисленную грань в противоположную дополнительную грань.

Совмещение вычислений с пересылками можно реализовать через двойную буферизацию, аналогично алгоритму FT: обработка очередной страницы блоков выполняется параллельно с выгрузкой из памяти сопроцессоров предыдущей страницы блоков-результатов и загрузкой следующей страницы входных блоков.

Подобная схема трафаретных вычислений подходит для сопроцессоров с относительно малым объёмом памяти ($DMEM$). В частности, она может применяться и на GPU, для достаточно больших массивов — см., например, [95]. Поскольку, как показывает пример алгоритма FT, на GPU, в отличие от КОМДИВ, большое количество пересылок может критически сказаться на производительности вычислений, авторы применяют ряд методов, чтобы сэкономить время на пересылки — например, вычисляют несколько РО подряд, за счёт загрузки большего количества дополнительных граней.

Схема реализации на ВМ7/9 имеет некоторую специфику, связанную с особенностями архитектуры.

В-первых, трёхмерные массивы представляются в специальном формате. Каждая процедура РО получает на вход и даёт на выходе трёхмерный мас-

сив 32-разрядных вещественных чисел, включающий дополнительные грани. Контроллер DMA пересылает данные 128-разрядными словами, т.е. по 4 вещественных числа. Согласно требованиям управляющей библиотеки, при пересылке адрес соответствующей области в глобальной памяти, а также шаг и длина каждого непрерывного фрагмента, должны быть кратны 16 байтам, т.е. тем же 4 вещественным числам. При вычислении РО требуется загружать в память секций блок с дополнительными гранями, а выгружать — без дополнительных граней. Отсюда, адреса загружаемого и выгружаемого массива как в глобальной, так и в локальной памяти различаются на 1 элемент по каждому направлению, однако в обоих случаях должно соблюдаться выравнивание.

Чтобы удовлетворить этому требованию, для вычислений на CP2 входной массив дополняется ещё шестью нулевыми гранями, перпендикулярными Ox: по три с каждой стороны. При обработке каждый блок загружается в локальную память также с шестью «лишними» гранями, в которых могут содержаться нули или элементы соседнего блока, но в ходе вычислений эти грани не используются. Поскольку элементы массива 32-разрядные, эта избыточность позволяет добиться необходимого выравнивания как при загрузке, так и при выгрузке. Преобразование входных массивов в описанный формат происходит один раз перед началом вычислений, в процедуре инициализации, и не учитывается в замерах производительности.

Во-вторых, копирование граней в системной памяти, после вычисления каждого РО, выполняется посредством DMA: каждая грань загружается в локальную память, а затем выгружается на новое место в массиве в глобальной памяти. Что касается граней, перпендикулярных Ox, они состоят из отдельных элементов с шагом. Как говорилось выше, минимальная непрерывная область памяти, которую можно переслать по DMA — 128 бит, т.е. четыре соседних элемента. Поэтому процедура устроена таким образом, что фактически обмениваются блоки из четырёх плоскостей, параллельных боковым граням. Никаких операций на самом CP2 при этом не выполняется. Выбор такого способа реализации продиктован тем, что доступ к системной памяти со стороны DMA осуществляется значительно быстрее, чем со стороны CPU. При использовании DMA время на копирования не существенно по сравнению со временем вычисления РО.

3.2.2 Вычислительные ядра для СР2

Для реализации алгоритма MG потребовался набор из 6 оптимизированных под СР2 ядер РО: `resid` — невязка, `proj` — проекция, `interp` и `interp0` — интерполяция с накоплением и без, `smooth` и `smooth0` — сглаживание с накоплением и без; также было разработано ядро нормы.

Оценим предварительно производительность ядра РО — для определённости, невязки, которая вычисляется по формуле $r = v - Au$, где A — разностный оператор. Вычисление каждого выходного элемента требует 31 арифметическую операцию (см. рис. 2.1), включая вычитание. При этом операции умножения и сложения могут быть объединены в одну, с использованием команды умножения с накоплением — получаем 16 команд. Поскольку вычисления выполняются с одинарной точностью, а на СР2 имеются команды для работы с парами вещественных чисел (в 64-разрядных регистрах), теоретически возможно за 16 команд вычислить сразу 2 выходных элемента. Можно условно записать, что на вычисление одного элемента необходимо 8 команд. Это вычисление требует загрузки из локальной памяти 27 элементов массива u, одного элемента массива v и выгрузки одного элемента r — всего 29 операций. С учётом того, что на СР2 зарузки и выгрузки могут осуществляться 128-битными словами, на один элемент требуется 29/4 команд загрузки/выгрузки — меньше, чем арифметических. Т.о., при совмещении двух типов команд в VLIW-инструкциях, время работы ядра определяется количеством арифметических команд. Поскольку за одну команду выполняется две операции с парой элементов, производительность ядра можно грубо оценить как

$$Perf_{kern}{}^{peak} \approx 4F, \tag{3.3}$$

где F — тактовая частота процессора. Эта оценка не зависит от размеров входного массива и не учитывает накладные расходы на перегруппировку входных элементов в пары, о которой пойдёт речь ниже.

В ядрах РО используются арифметические команды, работающие с парами вещественных элементов: например, в ядре невязки по 27 парам соседних входных элементов путём редукции вычисляются 4 частичных суммы, которые затем складываются с весовыми коэффициентами и дают на выходе, после

вычитания, пару соседних выходных элементов. Чтобы задействовать VLIW-инструкции, во внутреннем цикле обработка пар конвейеризована: загрузка и начало обработки одной пары совмещается с завершением обработки предыдущей. Количество необходимых для такой реализации регистров FPU в ядре невязки, составило 60 из 64 в каждой секции. Т.о., общего количества регистров на CP2 не достаточно для одновременной обработки четырёх соседних элементов, поэтому используются команды загрузки и выгрузки 64-битных, а не 128-битных слов.

Ещё одна сложность связана с тем, что элементы каждой пары являются соседними в массиве, однако на CP2 в каждый регистр могут быть считаны только такие пары, в которых первый индекс чётный. Соответственно, чтобы получить необходимые пары с нечётным первым индексом, требуется перегруппировка, с использованием дополнительных команд. На CP2 наиболее эффективной для этой цели является команда `psprmsgn`, которая осуществляет перестановку полуслов в паре 64-битных слов и работает на конвейере обменов с регистрами, как и команды загрузки/выгрузки.

Описанные факторы обусловили то, что в разработанных ядрах команды загрузки/выгрузки и `psprmsgn` преобладают над арифметическими командами. Чтобы повысить эффективность, с каждым ядром вручную была проведена «компиляция»: команды в арифметическом конвейере и в конвейере обменов с регистрами были переупорядочены с учётом всех зависимостей, чтобы в основном цикле исключить VLIW-инструкции с пустой командой обменов с регистрами. Такое переупорядочение дало значительный прирост производительности — в 2 раза для ядра невязки.

Производительность разработанных ядер оценивается как общее количество арифметических операций, делённое на время работы. Время работы, в свою очередь, оценивается как общее количество тактов (по логу работы на эмуляторе), делённое на тактовую частоту. Количество арифметических операций находится из определения самих РО. Вообще говоря, производительность ядра зависит от размера блока. В целом, с точки зрения скорости пересылок выгодно использовать блоки наибольшего возможного размера. Кроме того, при равном общем количестве элементов производительность будет выше для блока с большей первой размерностью, т.к. при этом увеличивается количество итераций внутреннего цикла в ядре, а значит и плотность кода. Эти эвристи-

ческие соображения подтвердились и результатами замеров: для каждого ядра экспериментально был определён оптимальный размер блока. Ниже все оценки и результаты замеров приводятся для блоков оптимального размера: именно они используются на верхних уровнях алгоритма MG. В таблице Б.6 для каждого из шести операторов приводится оптимальный размер блока, формула для подсчёта арифметических операций, количество тактов в ядре и оценки производительности на BM7 и BM9, а также эффективность в процентах от пика, найденного по формуле 3.3.

Реальная производительность ядер — порядка 60% от теоретического максимума. Более низкая эффективность ядра проекции связана меньшей повторной используемостью данных: количество выходных элементов в 8 раз меньше, чем количество входных.

3.2.3 Оценки производительности

Каждая процедура PO обрабатывает блоки входного массива постранично, как описано в 3.2.1, и использует одно из ядер. Выведем формулы для оценки производительности этих процедур, на примере невязки.

В основном цикле обработка страницы блоков в ядре, за время T_C, совмещается с выгрузкой предыдущей страницы выходных блоков и загрузкой следующей страницы входных, за время T_{L+S}. Каждый входной блок загружается вместе с дополнительными гранями. Поскольку на CP2 блоки имеют небольшой размер (см. табл. Б.6), и кроме того, дополнительных граней, перпендикулярных Ox, загружается по 4 с каждой стороны, накладные расходы на их пересылку оказываются существенными. Например, блок размера $32 \times 8 \times 8$ включает 2048 элементов, а вместе с дополнительными гранями — 4000 элементов. Для простоты можно приближённо считать, что блок с дополнительными гранями вдвое больше, чем блок без них. В случае, если оператор включает накопление или вычитание — например, невязка, — загружается второй входной блок, без дополнительных граней.

Пусть размер входного массива — $N \times N \times N$, размер одного блока — $N_1 \times N_2 \times N_3$. Для оператора невязки время вычислений и пересылок для одной

страницы блоков оценивается по формулам:

$$T_C = \frac{31 N_1 N_2 N_3}{Perf_{kern}}, \tag{3.4}$$

$$T_{L+S} = \frac{4K N_1 N_2 N_3}{BW}. \tag{3.5}$$

В сравнении $T_C \vee T_{L+S}$ необходимо учесть единицы измерения и то, что вычисления выполняются с одинарной точностью. Если $Perf_{kern}$ берётся в МОП/с, BW — в МБ/с, то сравнение имеет вид:

$$31\, BW \quad \vee \quad 16K \cdot Perf_{kern}\,. \tag{3.6}$$

В случае, если больше значение выражения слева, $PERF_{resid}$ оценивается как $K \cdot Perf_{kern}$, в противном случае

$$PERF_{resid} \approx \frac{31N^3}{4N^3 \cdot 4/BW} = \frac{31}{16} BW. \tag{3.7}$$

Аналогичные формулы выводятся для других операторов, с учётом разницы в количестве операций, наличия или отсутствия накопления, а также изменения размера блока для операторов проекции и интерполяции. Полученные для BM7/9 теоретические оценки, со значениями $Perf_{kern}$ из таблицы Б.6, приводятся в следующем разделе в таблицах 3.3 и 3.4.

На BM7 вычисления преобладают над пересылками во всех процедурах, кроме `interp` и `interp0`, в которых выходной блок в 8 раз больше, чем входной, причём в `interp` ещё один такой же блок загружается на CP2 для накопления. На BM9 пересылки преобладают также в `proj`. Для остальных операторов производительности на BM9 и на BM7 различаются пропорционально соотношению рабочих частот процессора. Во всех случаях, однако, время на пересылки и вычисления — величины одного порядка, а значит, если в реальной процедуре скорость пересылок ниже пиковой, пересылки могут получить перевес.

Поскольку в алгоритме MG вычисление невязки на верхнем уровне — базовая операция, его производительность также приближённо оценивается как $PERF_{resid}$.

3.2.4 Результаты тестирования

Разработанные процедуры вычисления РО были протестированы на ВМ7 и одном ядре ВМ9 (частота CPU — 1000 МГц, памяти — 600 МГц). Производительности и эффективности на ВМ7 приводятся в таблице 3.3.

Таблица 3.3 — Производительность разностных операторов на ВМ7

РО	**Теор. максимум, МОП/с**	**Производ-ть, МОП/с**	**Эфф-ть, %**
`resid`	1788	1465	82
`proj`	1187	703	59
`interp`	1312	831	63
`interp0`	1822	1101	60
`smooth`	1788	1478	83
`smooth0`	1731	1526	88

Операторы `resid` и `smooth`, на долю которых приходится бóльшая часть времени вычислений в алгоритме MG, а также `smooth0`, показывают эффективность более 80% от теоретического максимума. При вычислении проекции и интерполяций эффективность ниже — порядка 60%. Это связано с тем, что входные и выходные блоки в этих операторах существенно различаются по объёму, а чем меньше объём, тем ниже реальная скорость пересылки данных по DMA. В целом, производительности всех РО — величины одного порядка.

В таблице 3.4 приводятся производительности и эффективности тех же процедур на одном ядре ВМ9 с одним СР2, а также прирост относительно ВМ7.

Более низкая эффективность, чем на ВМ7, в особенности на процедурах проекции и интерполяции, обусловлена различиями в контроллерах DDR3 на ВМ9 и DDR2 на ВМ7, которые обсуждались выше в разделе 3.1.4. Во всех

Таблица 3.4 — Производительность разностных операторов на BM9

РО	**Теор. максимум, МОП/с**	**Производ-ть, МОП/с**	**Эфф-ть, %**	**Прирост**
`resid`	8943	5186	58	x3,5
`proj`	3335	1272	38	x1,8
`interp`	3674	1643	45	x2,0
`interp0`	5103	2091	41	x1,9
`smooth`	8943	5184	58	x3,5
`smooth0`	8655	7065	82	x4,6

процедурах данные передаются небольшими трёхмерными блоками, их строки располагаются в памяти с шагом — скорость пересылки таких блоков по DMA существенно ниже пиковой. В процедурах невязки и сглаживания, которые являются базовыми для алгоритма MG, входные и выходные блоки имеют равный размер, поэтому потери меньше. Однако влияние пересылок на общую производительность остаётся заметным — это отражается и на коэффициенте прироста производительности.

Реализация всего алгоритма MG, разработанная на основе референсных кодов и процедур для CP2, также была протестирована на одном BM7 и одном ядре BM9. Теоретический максимум производительности алгоритма — производительность процедуры невязки на верхнем уровне. В задачах всех классов на верхнем уровне используются блоки одного и того же размера, поэтому можно использовать полученные выше результаты: 1465 МОП/с на BM7 и 5186 МОП/с на BM9. Для корректности сравнения, из-за различия в способах подсчёта количества арифметических операций в алгоритме (см. разд. 2.2.4), эти величины должны быть домножены на 0,52. Итого, теоретический максимум производительности MG составляет:

1. 762 МОП/с на BM7,

2. 2697 МОП/с на ВМ9.

В таблице 3.5 приводятся результаты замеров производительности процедуры MG, в МОП/с, и её эффективность в процентах от теоретического максимума, а также прирост производительности на ВМ9 относительно ВМ7.

Таблица 3.5 — Производительность MG на СР2

	S	W	A	B
ВМ7 СР2	267	648	680	692
эффективность (%)	35	85	89	91
ВМ9 СР2	632	1911	1980	2011
эффективность (%)	23	71	73	75
прирост	x2,4	x1,8	x2,9	x2,9

В задаче класса S обрабатывается массив относительно небольшого размера — $32 \times 32 \times 32$, — поэтому на общую производительность алгоритма заметно влияют накладные расходы: вычисление операторов на нижних уровнях алгоритма, а также копирование граней после каждого оператора. Дополнительно проведённые замеры показали, что на вычисления на нижних уровнях приходится порядка 39% времени работы процедуры, а на копирование граней — 15%. Задачи остальных классов показывают эффективность порядка 90% на ВМ7 и свыше 70% на ВМ9 — это подтверждает тезис, что при достаточном объёме данных производительность базовой операции является адекватной оценкой производительности всего алгоритма.

Прирост производительности на больших классах задачи соответствует различию частот памяти. Таким образом, производительность реальной процедуры ограничена скоростью пересылок. Для расчётов на ВМ9 с той же эффективностью, что на ВМ7, требуется доработка контроллера DDR3.

3.2.5 Сравнение с производительностью на других процессорах

В таблице Б.7 приводятся производительности процедур, реализующих алгортим NPB MG, и сравнительные характеристики различных процессоров:

- для BM7/9 приводятся производительности разработанной автором процедуры, оптимизированной под CP2, а также референсной реализации на CPU. На BM9 замеры проводились на модуле с частотой процессорного ядра 1200 МГц и частотой памяти 800 МГц;
- для одного ядра универсального процессора Intel Xeon E5-2670v1 и для 16 ядер двух процессоров были проведены замеры производительности референсной OpenMP-реализации;
- для Эльбрус-4С данные получены на основе результатов тестирования MPI-версии из [20]. Версии OpenMP и MPI на одном процессоре как правило имеют одинаковые показатели производительности, но в случае Эльбрус-4С поддержка распараллеливания с помощью OpenMP носит экспериментальный характер, поэтому для тестирования использовалась MPI-версия. В таблице приводятся оценки производительности вычислений с одинарной точностью: вдвое выше, чем производительность вычислений с двойной точностью.

Рассматриваются задачи классов до B включительно — ограничение сверху связано с объёмом системной памяти BM7 и BM9. Прочерк означает отсутствие данных.

Замеры показали, что на Xeon, при достаточно большом объёме данных, масштабируемость близка к линейной до 8 потоков, что соответствует количеству каналов к памяти. При дальнейшем увеличении количества потоков рост производительности замедляется. Это свидетельствует о том, что на производительности MG пропускная способность памяти сказывается в большей степени, чем пиковая производительность процессора.

Использование сопроцессора CP2 для расчётов позволяет добиться существенного прироста производительности относительно вычислений на CPU — как на BM7, так и на BM9, несмотря на доработки в управляющем процессоре и наличие кэш-памяти второго уровня.

Тем не менее, производительность MG на одном ядре процессора Эльбрус оказывается несколько выше, чем на одном ядре ВМ9. Выше также и эффективность: пиковая производительность одного ядра Эльбрус на операции «умножение с накоплением» одинарной точности составляет на частоте 800 МГц 12,8 ГОП/с, в то время как пиковая производительность одного ядра ВМ9 на той же операции, на частоте 1200 МГц, составляет 19,2 ГОП/с. Высокая производительность на Эльбрус достигается за счёт мощного оптимизирующего компилятора, который в числе прочего раскладывает команды из программного потока в VLIW-инструкции. На ВМ9 оптимизированные вручную ядра для СР2 потенциально могут обеспечить производительность порядка 4500 МОП/с, однако эффективность снижается за счёт накладных расходов на пересылки данных, в частности, работы контроллера DDR3.

Данных о производительности NPB MG на процессоре STI Cell в открытых источниках найти не удалось, однако в ряде работ рассматривается эффективная реализация других трафаретных вычислений. Так, в [141] авторы отмечают проблему с выравниванием данных при пересылках по DMA. Предлагается обеспечить необходимое выравнивание за счёт дополнения массивов нулями — аналогично тому, как было сделано в описанной выше реализации для ВМ7/9. В [131] для трафаретных вычислений на Cell также используются в целом те же техники оптимизации, что и на ВМ7/9: приведение кода вручную к SIMD-формату вычислений, совмещение вычислений с пересылками по DMA за счёт двойной буферизации, дополнение массивов нулями для соблюдения требований к выравниванию. Высокая производительность достигается за счёт эффективной работа с DMA и с локальной памятью на SPE. Отмечается также, что общая производительность определяется вычислениями только при использовании малого количества ядер SPE, а на 8 и более ядрах — пересылками.

3.2.6 Реализация для многопроцессорного комплекса

По аналогии с однопроцессорной, автором была разработана и многопроцессорная оптимизированная реализация алгоритма MG для процессоров

ВМ7/9. Основой послужила MPI-версия референсных кодов, приведённая к вычислениям с одинарной точностью. На каждом процессоре вычисления выполняются на СР2.

В целом, процедура реализует схему с обменами гранями, аналогичную той, что использовалась в реализации на GPU (см. разд. 2.2.1). Для каждого РО в алгоритме на одном процессоре обрабатывается один блок — этот блок, в свою очередь, делится на более мелкие блоки, которые обрабатываются постранично, как описано в разд. 3.2.1.

После вычисления РО вызываются три процедуры обменов парами граней через MPI. Эти процедуры копируют грани трёхмерного блока, без дополнительных нулей, в непрерывные буферы, осуществляют обмен и затем копируют данные из принятых буферов в дополнительные грани блока. Копирования до и после пересылки можно выполнить посредством управляющего процессора, однако, как показало тестирование, время работы такой процедуры превышает время самих обменов. Это неудивительно, поскольку доступ к системной памяти со стороны CPU происходит медленно: для высокоскоростного доступа предназначен канал DMA. Поэтому был разработан набор вспомогательных процедур, которые осуществляют копирование при помощи DMA и СР2 — по две на каждую пару граней.

Распараллеливание на более чем 8 процессоров и, соответственно, MPI-процессов, было реализовано аналогично тому, как это было сделано для GPU (см. разд. 2.2.3): на заданном уровне алгоритма MG данные со всех процессоров собираются на один, на котором выполняются вычисления на нижних уровнях алгоритма.

Разработанная процедура была верифицирована и протестирована на эмуляторе СР2, а также на двух ядрах одного ПЛИС ВМ9. В настоящий момент продолжаются работы по портированию MPI на обмены по RapidIO для ВМ9. Можно, однако, оценить ожидаемую производительность вычислений на реальном многопроцессорном комплексе теоретически.

Вычисление РО на процессорах выполняется при помощи тех же оптимизированных процедур, что и в однопроцессорной реализации. Поэтому в качестве ${Perf_{kern}}^{mult}$ можно использовать производительность вычисления невязки. Вообще говоря, обмены по RapidIO могут отрицательно сказаться на производительности вычислений на самом процессоре, т.к. контроллер доступа к си-

стемной памяти является разделяемым ресурсом для DMA и RapidIO (см. [33, разд. 6.2]). Однако в разработанной реализации MG обмены выполняются после завершения вычисления PO; в реальности они могут на аппаратном уровне частично совмещаться с процедурами копирования в непрерывные буферы и обратно, которые относятся к накладным расходам. Таким образом, на BM9, при работе на частоте CPU 1000 МГц и памяти 600 МГц, ${Perf_{kern}}^{mult}$ составляет 5186 МОП/с (см. табл. 3.4).

Пусть размер входного массива — $N \times N \times N$, и комплекс состоит из M процессоров. Тогда на каждом процессоре обрабатывается блок размера $N_1 \times N_2 \times N_3$, причём $N_1 N_2 N_3 = N^3/M$. Время на вычисление невязки параллельно на всех процессорах оценивается по формуле:

$$T_C^{mult} = \frac{31 N_1 N_2 N_3}{{Perf_{kern}}^{mult}} = \frac{31 N^3}{M\ {Perf_{kern}}^{mult}}. \tag{3.8}$$

Время на обмены будем оценивать в предположении, что между любыми двумя процессорами, выполняющими обмен, существует выделенный канал RapidIO. Для алгоритма MG это означает, что процессоры объединяются в трёхмерный тор: каждый соединяется с шестью соседними — по два в каждом направлении. По каждому каналу одновременно пересылаются две грани, в двух направлениях. Количество элементов в одной грани, в зависимости от оси, равно $N_2 N_3$, $N_1 N_3$ или $N_1 N_2$, причём обмены по трём осям осуществляются последовательно. Если BW_{RIO} — пропускная способность RapidIO в одном направлении, то время на все обмены оценивается как

$$T_{L+S}^{mult} = \frac{N_2 N_3 + N_1 N_3 + N_1 N_2}{BW_{RIO}}. \tag{3.9}$$

Общее время вычисления невязки на многопроцессорном комплексе можно оценить по формуле $T^{mult} = T_C^{mult} + T_{L+S}^{mult}$. Отсюда находится производительность невязки, которая является оценкой производительности MG (с учётом коэффициента 0,52, см. разд. 3.2.4).

На BM9 $BW_{RIO} = 1250$ МБ/с. Рассмотрим в качестве примера задачу класса C и 8 процессоров; тогда $N = 512$, $N_1 = N_2 = N_3 = 256$. Получаем

следующие оценки:

$$T_C^{mult} \approx 0{,}1003 \text{ с}, \tag{3.10}$$

$$T_{L+S}^{mult} \approx 0{,}0025 \text{ с}\,. \tag{3.11}$$

Таким образом, влияние обменов по RapidIO на общую производительность несущественно.

Полученные оценки можно рассматривать как ориентировочные, т.к. они не учитывают отрицательное влияние на производительность ряда факторов:

- чем больше общее количество процессоров, тем меньше размер блока, обрабатываемого на каждом процессоре — соответственно, ниже производительность ядра;
- реальная скорость обменов данными между двумя процессорами отличается от пиковой и зависит в том числе от топологии соединения процессоров в комплексе;
- на производительности могут также сказаться накладные расходы на вызов самих MPI-процедур.

Тем не менее, на основании оценок можно утверждать, что пропускной способности каналов RapidIO потенциально достаточно для эффективной реализации алгоритма MG на многопроцессорном комплексе на базе BM9.

3.3 Процедура SpMV

3.3.1 Формат упаковки матрицы и схема вычислений

В реализации процедуры SpMV для GPU, описанной в разд. 2.3.1, использовался формат упаковки Sliced ELLpack, который имел два основных преимущества перед CSR. Во-первых, все строки в одном слое дополнялись нулями до одинаковой длины, что обеспечивало выравненный доступ к элементам матрицы и равномерное распределение работы между потоками. Во-вторых, элементы каждого слоя записывались в массив по столбцам, благодаря чему потоки,

вычислявшие соседние выходные значения, считывали из памяти соседние элементы.

При вычислении на CP2 первый фактор также является существенным: для пересылки данных по DMA необходима выравненность, и кроме того, все вычислительные секции исполняют один поток инструкций, в частности, равное количество итераций циклов, поэтому должны обрабатывать строки равной длины. С другой стороны, в каждой секции инструкции выполняются в один поток, поэтому расположение элементов слоя по столбцам на производительность не повлияло бы.

Исходя из этих соображений, для реализации процедуры SpMV на CP2 был выбран формат упаковки, который можно условно назвать «Sliced CSR»: матрица разбивается на «слои» из некоторого фиксированного количества строк (H), и в каждом слое строки дополняются нулями так, чтобы длина всех строк в нём была одинаковой. Далее каждый слой упаковывается в формате CSR. Упакованная матрица, как и в формате Sliced ELLpack, представляется в виде трёх массивов: вещественный массив значений ненулевых элементов, целочисленный массив соответствующих номеров столбцов и массив индексов начал слоёв.

Для реализации SpMV на CP2 имеет место то же ограничение, что и для реализации алгоритма MG: объём локальной памяти CP2 достаточно мал, соответственно, вычисление по схеме, аналогичной той, что была реализована на GPU (см. разд. 2.3.2), возможно только для очень маленьких матриц. Для матриц большего размера необходимо использовать схему вычислений, при которой слои матрицы постранично пересылаются и обрабатываются на CP2.

Второе ограничение связано с длиной входного вектора: для обработки одного слоя в локальной памяти требуется входной вектор целиком. Соответственно, если матрица достаточно велика, необходимо предварительное разбиение матрицы на блоки и соответствующее разбиение входного вектора. Далее каждый блок упаковывается в описанном выше формате и обрабатывается на CP2, в конце выполняется редукция частичных сумм для каждой строки. В данном разделе этот случай подробно не рассматривается: предполагается, что локальной памяти секции CP2 достаточно для одновременного хранения всего входного вектора, одного слоя матрицы и соответствующего выходного векто-

ра. В частности, среди матриц из задачи NPB CG различных классов только матрица класса S удовлетворяет этому ограничению.

Схема вычислений состоит в следующем. Матрица, переупакованная в описанный формат, а также входной и выходной вектора, размещаются в системной памяти процессора. Строки одного слоя распределяются поровну между четырьмя секциями; отсюда, в частности, следует, что H должно быть кратно 4. В начале каждой операции SpMV в каждую секцию CP2 загружается копия входного вектора. Далее в цикле на CP2 загружается каждый слой матрицы, вызывается ядро SpMV, и затем полученные фрагменты результата выгружаются в системную память. Загрузку входного вектора невозможно совместить с вычислениями. Совмещение происходит в цикле по слоям: вычисления над очередной страницей совмещаются с выгрузкой предыдущих фрагментов результата и загрузкой следующей страницы слоёв.

3.3.2 Вычислительное ядро для CP2

Ядро SpMV на CP2 для матрицы в формате «Sliced CSR» умножает набор строк равной длины на входной вектор x. Элементы массива значений и массива номеров столбцов занимают непрерывные области в локальной памяти и обрабатываются последовательно. Для каждого элемента из памяти считываются значение и номер столбца, далее считывается элемент x с индексом, равным номеру столбца; значения элементов матрицы и вектора x перемножаются и прибавляются к накапливаемой сумме — выходному элементу с индексом, равным номеру обрабатываемой строки. Массив индексов начал слоёв не используется в ядре: он требуется только при определении адреса области в глобальной памяти, которая пересылается на CP2.

Предварительно оценить производительность ядра SpMV можно следующим образом. Пусть NZ_l — общее количество элементов фрагмента матрицы (ненулевых и дополнительных нулей), который обрабатывается ядром в одной секции, H_l — количество строк, N — порядок матрицы. Тогда количество элементов в одной строке — $NZ_l\,/H_l$, выходной вектор состоит из H_l элементов. Выходной вектор инициализируется нулями. На вычисление каждого выход-

ного элемента требуется $2\,NZ_l\,/H_l$ арифметических операций, которые могут быть выполнены за $NZ_l\,/H_l$ команд умножения с накоплением. Поскольку на CP2 арифметические команды работают с парами вещественных чисел, можно условно считать, что на один элемент требуется $NZ_l\,/2H_l$ арифметических команд.

Элементы могут загружаться и выгружаться двойными словами (128 бит) — соответственно, на один выходной элемент требуется по $NZ_l\,/4H_l$ команд для загрузки значений и номеров столбцов, а также $1/4$ команды выгрузки — этим слагаемым в оценках можно пренебречь. Что касается, однако, вектора x, соседние элементы матрицы могут умножаться на элементы x с различными, не известными заранее индексами. Архитектура CP2 не позволяет в одной команде загружать элементы с четырьмя произвольными индексами, поэтому каждый элемент загружается отдельно. Кроме того, требуются элементы с различными индексами во всех $K = 4$ секциях CP2, однако параллельно в разных секциях обращение к локальной памяти возможно только по одному и тому же адресу. Доступ по разным адресам должен осуществляться последовательно. Следовательно, на обработку в ядре четырёх элементов матрицы приходится $4K$ команд загрузки элементов входного вектора, и таких четвёрок в каждой строке по $NZ_l\,/4H_l$. Итого, на вычисление одного выходного элемента приходится, как минимум,

$$2\frac{NZ_l}{4H_l} + 4K\frac{NZ_l}{4H_l} \approx \frac{(2K+1)\,NZ_l}{2H_l} \tag{3.12}$$

команд загрузки/выгрузки — больше, чем арифметических команд.

Общее количество тактов на загрузки для всего фрагмента матрицы оценивается по формуле $(2K+1)\,NZ_l\,/2$, количество арифметических операций составляет $2\,NZ_l$ — отсюда получаем оценку производительности:

$$Perf_{kern}{}^{peak} \approx \frac{4F}{2K+1}, \tag{3.13}$$

где F — тактовая частота процессора. Эта оценка не зависит ни от порядка матрицы, ни от количества ненулевых элементов.

На практике при реализации ядра на организацию чтения элементов x последовательно в четырёх секциях и объединение считанных элементов в пары (64-битные слова) требуется большое количество вспомогательных операций. Это связано, в частности, с тем, что на CP2 ветвление поддерживается только

в формате условного выполнения арифметических команд, по отдельно устанавливаемому коду условия. Кроме того, индексы элементов x считываются из массивов номеров столбцов парами — каждую пару необходимо разделить на два отдельных индекса (например, при помощи команды `mtrans`), каждый индекс записать в адресный регистр, произвести чтение двойного слова, выбрать верхнее или нижнее полуслово, в котором находится искомый элемент x, и затем два считанных элемента x объединить в пару.

В разработанном ядре для упрощения логики перед началом вычисления SpMV, после загрузки входного вектора в локальную память, производится разреживание его элементов нулями, так, чтобы каждый элемент x_i располагался в отдельном 64-битном слове: $(0, x_i)$. Отметим, что при этом усиливается ограничение на размер задачи, т.к. для хранения x требуется вдвое больше памяти. Даже с этим упрощением, однако, на описанную зачитку элементов x в ядре приходится более 95% всех тактов.

Поскольку из всех матриц алгоритма NPB CG ограничениям на размер удовлетворяет только матрица класса S, будем для полноты рассматривать также матрицы различного порядка, со случайным расположением и значениями ненулевых элементов и с коэффициентом заполнения 0,06 (т.е. матрицы, в которых 6% всех элементов являются ненулевыми). Тестирование показало, что для матриц любого порядка наибольшая производительность достигается, когда H является целой степенью 2 и максимально возможное. При фиксированном H оценить $Perf_{kern}$ можно по логу работы ядра на эмуляторе CP2: она составляет ту же долю от пиковой на команде `psmadd` (800 МОП/с на частоте 200 МГц и 4000 МОП/с на частоте 1000 МГц), что и количество таких команд от общего количества выполненных. Полученные таким образом оценки, эффективность в процентах от теоретического максимума 3.13, а также оптимальные значения H, приводятся в таблице Б.8. Теоретический максимум составляет 89 МОП/с для ВМ7 и 444 МОП/с для ВМ9 на частоте 1000 МГц. Для случайных матриц приводятся средние значения по 20 запускам. Матрица класса S имеет порядок 1400×1400.

Как и следовало ожидать, производительность ядра немного возрастает с ростом объёма данных, т.к. снижаются накладные расходы на разгон и торможение циклов, но для всех матриц эффективность составляет лишь около 30%.

3.3.3 Оценки производительности

Оценим производительность процедуры SpMV в целом, осуществляющей постраничную обработку слоёв матрицы. Чтобы вывести формулы, не зависящие от структуры матрицы, будем предполагать, что количество дополнительных нулей незначительно, а ненулевые элементы распределются между слоями и строками поровну. Выведенные таким образом оценки можно рассматривать как ориентировочные оценки сверху.

Пусть NZ — общее количество ненулевых элементов, N — порядок матрицы, H — ширина слоя. Тогда N/H — количество слоёв, $(NZ \cdot H)/N$ — количество ненулевых элементов в одном слое, из которых в каждой секции обрабатывается $(NZ \cdot H)/(N \cdot K)$.

В основном цикле вычисление SpMV на CP2 с очередным слоем совмещается с выгрузкой предыдущего фрагмента выходного вектора, которой можно пренебречь, и загрузкой следующего слоя матрицы. Слой упакованной матрицы состоит из двух массивов одинакового размера: массив значений из 32-битных вещественных чисел одинарной точности и массив номеров столбцов из 32-битных целых чисел. Время на вычисления (T_C) и на загрузку (T_L) оценивается по формулам:

$$T_C = \frac{2\,NZ \cdot H}{N \cdot Perf_{kern}}, \tag{3.14}$$

$$T_L = \frac{2\,NZ \cdot H}{N \cdot BW}. \tag{3.15}$$

Если $Perf_{kern}$ берётся в МОП/с, BW — в МБ/с, то сравнение $T_C \vee T_L$ имеет вид:

$$BW \quad \vee \quad 4\,Perf_{kern}\,. \tag{3.16}$$

Как для BM7, так и для BM9 значение выражения слева оказывается намного больше — это означает, что $PERF_{SpMV}$ определяется производительностью ядра и оценивается как $K \cdot Perf_{kern}$.

3.3.4 Результаты тестирования

Результаты замеров производительности процедуры SpMV на BM7 и эффективности в процентах от теоретического максимума приводятся в таблице 3.6.

Таблица 3.6 — Производительность SpMV на BM7

Матрица	**Теор. максимум, МОП/с**	**Производ-ть, МОП/с**	**Эфф-ть, %**
S	115	51	44
32×32	98	4	4
64×64	108	11	10
128×128	112	24	21
256×256	114	38	33
512×512	115	54	47
1024×1024	115	67	58
2048×2048	115	76	66
4096×2048	115	75	65

В таблице 3.7 приводятся аналогичные данные для BM9. На модуле, на котором производились замеры, частота памяти DDR3 — 700 МГц.

По результатам замеров, производительность возрастает с увеличением объёма входных данных, как и предсказывали теоретические оценки. Однако, на небольших матрицах наблюдается также потеря эффективности. Это связано с тем, что производительность ядра в теоретических оценках подсчиты-

Таблица 3.7 — Производительность SpMV на BM9

Матрица	**Теор. максимум, МОП/с**	**Произв-ть, МОП/с**	**Эфф-ть, %**	**Прирост**
S	575	248	43	x4,5
32×32	490	18	4	x4,6
64×64	542	50	9	x4,6
128×128	560	110	20	x4,7
256×256	568	180	32	x4,9
512×512	573	267	47	x5,1
1024×1024	575	336	59	x5,0
2048×2048	576	384	67	x5,1
4096×2048	576	383	67	x4,9

валась по фактически выполняемому числу умножений с накоплением. Сюда входят и операции, выполняемые с дополнительными нулями; в то же время, реальная производительность вычисляется по количеству ненулевых элементов в исходной матрице. Поскольку оптимальное значение H для всех матриц — не менее 32, и кроме того, для пересылок по DMA и вычислений в ядре требуется выравненность длин строк, количество дополнительных нулей оказывается достаточно велико. С этими дополнительными нулями выполняются как «лишние» вычисления на CP2, так и пересылки по DMA. В таблице Б.9 для каждой матрицы приводится количество ненулевых элементов, фактическая длина массива значений с дополнительными нулями и их соотношение, которое соответствует максимальной достижимой эффективности вычислений.

Более низкая эффективность на матрице класса S, чем на случайных матрицах порядка 2048 × 2048 и 4096 × 2048, означает неравномерность распределения ненулевых элементов между строками матрицы. Прочие накладные расходы в процедуре SpMV включают разреживание входного вектора нулями и другие вспомогательные действия. Производительности на BM7 и BM9 соотносятся как рабочие частоты процессоров — это подтверждает, что влияние пересылок на производительность SpMV незначительно.

3.3.5 Сравнение с производительностью на других процессорах

Производительность процедуры SpMV можно рассматривать и как оценку производительности алгоритма NPB CG в целом.

Для сравнения с реализацией на CP2 была также протестирована простейшая процедура SpMV с матрицей в формате CSR на CPU BM7/BM9. На BM9 замеры проводились на одном ядре, на модуле с частотой CPU 1000 МГц и памяти 700 МГц. Аналогичная процедура, использующая OpenMP, была протестирована на 1 и 8 ядрах универсального процессора Intel Xeon E5-2670v1 и на 16 ядрах двух таких процессоров. Тестовый набор матриц — тот же, что использовался в предыдущем разделе. Результаты замеров приводятся в таблице Б.10.

Несмотря на существенные различия в архитектуре CP2 и процессоров общего назначения с кэш-памятью, производительность ядра SpMV на CP2 также ограничена доступом к памяти. На BM9, который обладает кэш-памятью второго уровня, на большинстве матриц производительность вычислений на CPU оказывается выше. При этом на маленьких матрицах производительность ядра BM9 приблизительно равна производительности ядра Xeon: здесь частоты памяти различаются мало, и в обоих случаях используется один канал. На матрицах большего размера производительность Xeon выше, поскольку он имеет кэш-память большего объёма. При этом увеличение количества параллельных потоков исполнения сказывается на производительности отрицательно, т.к. доступ к памяти становится более разреженным. На больших матрицах это компенсируется бóльшим количеством задействованных каналов памяти.

В дальнейшем в НИИСИ П.Б. Богдановым была разработана более эффективная процедура SpMV для СР2, в которой производится предобработка матрицы с разбиением на блоки, а также не используется разреживание входного вектора нулями в ядре. Данная процедура может работать с матрицами большего размера и показывает на ВМ9 в несколько раз более высокую производительность на СР2, чем на CPU. Однако в этой процедуре, как и в описанной выше, происходит чтение элементов входного вектора последовательно в четырёх секциях СР2 — наиболее существенный фактор, снижающий производительность ядра. Результаты замеров для этой процедуры приведены в работе [4].

Данные о производительности SpMV на Эльбрус-4С не представлены в силу недоступности аппаратного стенда на базе этого процессора. В [20] отмечается, что вычисления с разреженными матрицами на процессорах Эльбрус требуют ручной архитектурно-зависимой оптимизации.

В [102] рассматривается реализация на Cell всего алгоритма CG. Отмечается, что наибольшую выгоду даёт параллелизация именно процедуры SpMV. В предложенной реализации матрица обрабатывается постранично: каждый SPE обрабатывает некоторое количество строк, и вычисления на SPE совмещаются с пересылками данных по DMA. Для оптимизации доступа ко входному вектору был предложен подход, при котором на управляющем процессоре создаётся новый вектор длины NZ, содержащий элементы входного вектора в порядке индексов из массива номеров столбцов. Это решение, однако, оказалось неэффективным из-за накладных расходов на копирование — ту же ситуацию можно ожидать и на ВМ7/9.

В [141] рассматривается реализация SpMV на Cell для матрицы в формате BCSR (CSR с разбиением на блоки), которая также обрабатывается постранично. Поскольку разбиение матрицы на блоки означает и разбиение входного вектора, снимается ограничение на размер задачи, связанное с необходимостью хранить в локальной памяти весь входной вектор. Хотя SPE не обладают кэш-памятью, доступ к локальной памяти на них осуществляется более эффективно, чем на СР2. Вычисления в ядре приводятся к SIMD-формату на этапе компиляции. При условии оптимального выбора параметров разбиения полученные в результате ядра для SPE оказываются достаточно производительными, чтобы их время работы было меньше, чем время соответствующих пересылок бло-

ков матрицы. Таким образом, производительность SpMV определяется пиковой скоростью пересылки данных по DMA.

В другой работе, [143], SpMV рассматривается как часть более сложной процедуры. Для оптимизации доступа ко входному вектору применяется следующий подход: после загрузки фрагментов массива номеров столбцов каждый SPE формирует списки необходимых элементов входного вектора, и далее DMA предаёт на SPE элементы по этим спискам, с некоторой избыточностью, связанной с требованиями к выравниванию. Контроллер DMA процессоров ВМ7/9 не обладает такой возможностью.

3.4 Выводы

В данной главе, при помощи метода, предложенного в разделе 1.5, была исследована производительность выбранных процедур на гибридных процессорах НИИСИ РАН. Сопоставление теоретических оценок с реальной производительностью на разных этапах реализации процедур позволило выявить особенности архитектуры процессоров, которые ограничивают производительность этих процедур.

Архитектура процессоров ВМ7/9 разрабатывалась для эффективного вычисления БПФ. Благодаря этому, низкоуровневая оптимизация позволила добиться производительности БПФ, сопоставимой и превосходящей производительность на процессорах мировых производителей. В частности, одно ядро ВМ9 на БПФ всех рассмотренных длин показывает более высокую производительность, чем одно ядро Xeon, несмотря на менее совершенный технологический процесс изготовления и, соответственно, более низкую тактовую частоту. Это достигается за счёт трёх факторов: специализированной архитектуры СР2, высокопроизводительного канала DMA и возможности совмещения вычислений на СР2 с пересылками по DMA.

На архитектуре ВМ7/9 производительность разработанных процедур БПФ и свёртки не может быть существенно улучшена и приближена к пиковой производительности СР2, т.к. производительность этих процедур ограничена скоростью пересылок данных.

Скорость пересылок зависит не только от теоретической скорости передачи данных по DMA (128 бит за такт рабочей частоты), но и от пропускной способности системной памяти и возможностей её контроллера. Так, если данные расположены в системной памяти в виде небольших непрерывных фрагментов, фактическая скорость пересылки оказывается заметно ниже, чем для одной непрерывной области. Кроме того, имеются требования к выравниванию для пересылаемых данных. Выравнивание означает в общем случае необходимость дополнять обрабатываемые массивы нулями — отсюда возникает избыточность и при пересылках, и при вычислениях на CP2. Несмотря на эти факторы, при условии совмещения вычислений с пересылками пропускной способности памяти и DMA оказывается достаточно для эффективной реализации не только длинного БПФ, но и других алгоритмов, связанных с массивно-параллельной обработкой больших массивов данных. Примером является алгоритм MG, производительность которого на BM9 сопоставима с производительностью на процессорах других производителей.

Для алгоритмов с нерегулярным доступом к памяти — таких, как SpMV — имеет место другая ситуация. Здесь пропускная способности DMA также не является бутылочным горлышком. Однако контроллер DMA имеет ограниченную функциональность и поддерживает только пересылки регулярных выравненных областей — соответственно, задача нерегулярного доступа ложится на сопроцессор CP2. В ходе таких вычислений высокая пиковая производительность CP2, которая достигается за счёт SIMD-архитектуры, фактически нивелируется: операции приходится выполнять последовательно для всех секций. В результате для процедуры SpMV выигрыш в производительности за счёт оптимизации под CP2 оказывается меньшим, чем для алгоритмов MG и БПФ.

В целом можно отметить, что производительность всех рассмотренных алгоритмов на процессорах КОМДИВ определяется, как и на гибридных системах с GPU, подсистемой памяти — либо доступом к локальной памяти на самом CP2, либо доступом к системной памяти со стороны DMA. Поэтому эффективная реализация подразумевает совмещение вычислений и операций с памятью: плотный код ядра для CP2, задействующий VLIW-инструкции, и двойная буферизация при обменах данными между CP2 и системной памятью. На КОМДИВ, в отличие от GPU, эти оптимизации производятся вручную.

Глава 4. Рекомендации по дальнейшему развитию архитектуры гибридных многоядерных процессоров НИИСИ РАН

4.1 Критерий сбалансированности процессоров НИИСИ РАН на выбранном классе задач

В предыдущей главе для трёх процедур были рассмотрены схемы вычисления на процессорах КОМДИВ ВМ7/9 с сопроцессором СР2. Для этих схем в разделах 3.1.3, 3.2.3 и 3.3.3 были выведены формулы для оценки производительности вычислений по производительности ядра и пропускной способности канала доступа к системной памяти — по методу оценки производительности, сформулированному в разделе 1.5. Хотя в данном разделе будут рассматриваться только процессоры НИИСИ РАН, аналогичные рассуждения можно провести и для гибридных узлов на базе GPU и соответствующих схем вычислений, на основе формул, полученных в главе 2.

Подстановка в формулы параметров реальных процессоров и процедур позволила судить о том, чем ограничена производительность каждой процедуры: ядром или пересылками данных, — и вывести теоретический максимум производительности, которого можно достичь на процессоре. Те же формулы могут быть использованы и для оптимизации архитектуры под процедуру. Если принципиальная структура процессора остаётся без изменений, то и вычисления можно проводить по тем же схемам. Однако, варьируя количественные параметры, можно добиться сбалансированности системы на выбранной процедуре.

Формальный критерий сбалансированности ВС на заданной процедуре был сформулирован в разделе 1.6 и состоит в равенстве между временем вычислений в ядре и временем пересылок. Время вычислений в ядре на СР2 определяется производительностью этого ядра, которая, в свою очередь, может быть оценена по количеству секций и тактовой частоте процессора.

В реальности параметры процессора не могут быть выбраны произвольно, т.к. имеются различные технологические ограничения; кроме того, точное равенство может быть достигнуто только в частных случаях, на определённых размерах задач. Соответственно, в зависимости от разницы между левой

и правой частью можно говорить о большей или меньшей сбалансированности процессора и о преобладании вычислений или пересылок в ходе работы.

Сформулируем критерий сбалансированности гибридных процессоров НИИСИ РАН на каждой из рассмотренных процедур: БПФ, MG и SpMV, — они представляют классы задач ЦОС, вычислений на сетках и вычислений с разреженными матрицами. В соотношения входят три ключевых параметра:

- F — тактовая частота сопроцессора, которая совпадает с частотой процессорного ядра и выражается в МГц;
- K — количество секций сопроцессора;
- BW — пропускная способность канала доступа к системной памяти, в МБ/с.

Последняя величина — BW — вообще говоря, зависит от целого ряда параметров: типа и частоты микросхемы памяти, количества каналов доступа к ней, архитектуры контроллера системной памяти и контроллера DMA, а также от формата пересылаемой области в системной памяти. Как показали проведённые замеры (см. разд. 3.1.4, 3.2.4 и табл. Б.3), реальная пропускная способность может существенно отличаться от пиковой, которая вычисляется по разрядности DMA и тактовой частоте процессора. Точное выражение зависимости между BW и аппаратными характеристиками в виде формулы требует отдельного исследования. Ниже под BW подразумевается максимальная пропускная способность, достижимая с учётом всех характеристик на пересылках того типа, которые используются в процедуре.

Для процедуры БПФ соотношение получается путём подстановки в 3.2 оценки производительности ядра 3.1. Система является сбалансированной на множественном БПФ длины $N \leqslant 4096$ (причём $N = 2^n$), если выполнено соотношение:

$$BW \log N = 32KF. \tag{4.1}$$

Например, на процессоре ВМ7, если в качестве BW подставить теоретический максимум пропускной способности DMA (3200 МБ/с), равенство достигается при $N = 256$. Ограничение на N сверху связано с объёмом локальной памяти СР2: БПФ бóльших длин вычисляется по длинному алгоритму. Увеличение объёма локальной памяти позволит ослабить это ограничение. Для многомерного или длинного одномерного БПФ в качестве N рассматривается каждая из размерностей коротких БПФ, к которым сводится процедура.

Для процедуры MG соотношение получается на основе оценок для PO на сетке: в 3.6 подставляется 3.3. Система является сбалансированной, если

$$31\,BW = 64KF. \tag{4.2}$$

Нетрудно увидеть, что при фиксированной тактовой частоте и количестве секций, а также при имеющемся объёме локальной памяти сопроцессора, процедура MG предъявляет более слабые требования к BW, чем БПФ: достаточно меньшей пропускной способности, чтобы все пересылки могли быть скрыты за вычислениями.

Для SpMV в 3.16 подставляется оценка производительности 3.13. Система является сбалансированной, если

$$BW = \frac{16F}{(2K+1)}. \tag{4.3}$$

Из этого соотношения следует, в частности, что процессоры BM7/9 для вычислений с разреженными матрицами сбалансированы в меньшей степени, чем для БПФ и вычислений на сетках. Причина, как было показано в разд. 3.3.2, заключается в особенностях архитектуры CP2, которые не позволяют эффективно реализовать ядро.

Все формулы были выведены в предположении, что вычисления выполняются с одинарной точностью. Если рассматривать поддержку вычислений с двойной точностью в архитектуре сопроцессора, в формулы необходимо внести соответствующие поправки. Так, например, в процедурах БПФ и MG объём пересылаемых данных возрастает вдвое. В SpMV только один из пересылаемых массивов содержит вещественные числа, а второй является целочисленным.

Формулы для оценки производительности ядер могут меняться в зависимости от того, как именно предполагается реализовать вычисления с двойной точностью. В частности, если будет промасштабирована вся архитектура сопроцессора: увеличена вдвое разрядность регистров FPR, GPR, объём локальной памяти и памяти коэффициентов, и т.д., — то оценки производительности ядер останутся без изменений. В этом случае в формулах 4.1 и 4.2 правая часть домножается на коэффициент 2, а в 4.3 — на коэффициент 1,5.

4.2 Проект оптимизации архитектуры гибридных процессоров НИИСИ РАН

В предыдущих главах были выведены теоретические оценки производительности процедур, которые, в частности, зависят от сбалансированности системы. Вместе с тем, реальная производительность разработанных процедур не достигает теоретического максимума, хотя и соответствует ему по порядку — это объясняется не только наличием накладных расходов в программных реализациях процедур, но и качественными особенностями архитектуры процессоров. Существенного повышения производительности на рассмотренных классах задач можно добиться за счёт усовершенствования отдельных подсистем процессора.

Что касается БПФ и других задач ЦОС, вычисления на CP2, в силу его специализированной архитектуры, выполняются здесь с очень высокой эффективностью. Разработанные процедуры БПФ и свёртки в целом также демонстрируют сбалансированность вычислений и пересылок и показывают высокую производительность. На BM9, однако, потери за счёт пересылок несколько больше, чем на BM7, из-за различия в типах и контроллерах системной памяти. Соответственно, повышение эффективности на задачах ЦОС возможно путём доработки контроллера доступа к системной памяти, иначе говоря, увеличения значения BW.

Кроме того, по результатам замеров, с увеличением длины входного вектора наблюдается резкое падение производительности — почти в 3 раза — при переходе от «короткого» алгоритма Кули-Тьюки к алгоритму «БПФ за 4 шага», который связан с пересылками большего объёма данных через DMA. Увеличение объёма локальной памяти CP2 позволило бы обрабатывать большее количество длин векторов по более эффективному алгоритму.

Ещё одна возможная оптимизация связана в большей степени с удобством программирования, хотя потенциально может дать и небольшой прирост производительности. В таких процедурах, как свёртка, требуется вычисление и прямого, и обратного БПФ. Если объём данных помещается в локальную память, эти операции должны выполняться подряд, без промежуточных выгрузок и загрузок. Однако требуется сменить направление БПФ, т.е., знак коэффициентов

Фурье, считываемых из памяти коэффициентов. В текущей версии сопроцессора знак задаётся битом в управляющем регистре — обновить его значение можно только со стороны CPU, поэтому требуется прерывание работы СР2. Добавление специального регистра, содержащего этот бит и доступного для записи со стороны СР2, позволило бы выполнять прямое и обратное БПФ подряд, без обращения к CPU.

Производительность разработанной процедуры MG в настоящий момент ограничена на ВМ7 производительностью ядер, а на ВМ9 — доступом к памяти. При условии доработки контроллера DDR3, в последующих версиях процессора вычисления в ядре могут снова получить перевес, поэтому увеличение производительности ядер за счёт усовершенствований в архитектуре СР2 будет означать и более высокую производительность вычислений на сетках в целом.

Реализация ядер разностных операторов (РО) на СР2 затруднена тем, что в них единицей обработки является одно вещественное число (а не комплексное число и не пара вещественных), однако данные адресуются в локальной памяти и обрабатываются в регистрах FPR 64-разрядными словами. Такой доступ осуществляется быстро, однако в ядре фактически приходится эмулировать 32-разрядный доступ, с использованием вспомогательных инструкций.

Представляется целесообразным реализовать возможность чтения и записи в память 32-разрядных слов, в старшую или младшую половину регистра. Такая возможность, разумеется, должна дополнять, но не заменять единовременное чтение/запись 64- и 128- разрядных слов, и использоваться только при необходимости. Возможный формат команды чтения:

```
lh fd, (rn)+nn, fmt
```

Здесь поле `fmt` — двубитное, каждый бит принимает значение 0 или 1. 0 означает нижнюю половину 64-разрядного слова, 1 — верхнюю, при этом младший бит указывает половину считываемого слова по адресу `rn` в локальной памяти, а старший — половину регистра `fd`.

В формуле 3.3 для оценки производительности ядра РО накладные расходы на перегруппировку не учитывались. Поддержка 32-разрядного доступа позволит повысить эффективность, относительно приведённой в табл. Б.6: в коде ядра будут преобладать команды на арифметическом конвейере, а не на конвейере обменов с памятью.

Повышение эффективности вычисления PO возможно также за счёт количественных изменений в архитектуре CP2. Так, увеличение количества регистров FPR в каждой секции позволило бы обрабатывать большее количество элементов во внутреннем цикле ядра (степень вложенности циклов в ядрах — 3) — соответственно, снизились бы накладные расходы на разгон и торможение циклов. В свою очередь, увеличение объёма локальной памяти позволило бы обрабатывать данные более крупными блоками, а значит, снизить накладные расходы на пересылку и хранение дополнительных граней, а также повысить скорость самих пересылок.

Производительность процедуры SpMV на BM7 и BM9 определяется вычислениями в ядре, соответственно, из трёх рассмотренных классов алгоритмов именно вычисления с разреженными матрицами имеют наибольший потенциал для ускорения за счёт развития архитектуры CP2.

Существенно упростить код ядра позволила бы описанная выше возможность чтения из памяти 32-разрядных слов. Здесь, однако, необходимость считать старшую или младшую половину слова из памяти определяется входными данными, поэтому поле `fmt` должно быть не константой, а значением в некотором регистре. Альтернативой является 32-разрядная адресация в локальной памяти — для неё возможный формат команды чтения:

```
lh fd.[hi/lo], (rn)+nn
```

Здесь `hi/lo` означает ту половину регистра `fd`, в которую производится загрузка.

С точки зрения производительности, однако, основная сложность чтения элементов входного вектора в процедуре SpMV состоит в том, что в разных секциях необходимо чтение элементов из локальной памяти по разным адресам. В настоящий момент в SpMV и других процедурах с этим свойством эмулируется последовательное чтение данных в четырёх секциях. Эту проблему можно решить, реализовав раздельные наборы адресных регистров для четырёх секций. При этом для записи значений в эти регистры потребуются два вида инструкций:

- инструкции, которые записывают одно значение в адресные регистры всех секций — например, копируют значение из регистра GPR;
- инструкции, которые записывают различные значения в разных секциях — например, копируют из регистров FPR.

Последовательный доступ к памяти секций был отражён в формуле 3.13 для оценки производительности ядра. Если будет использована раздельная адресация, формула приобретёт вид:

$$Perf_{kern}{}^{peak} \approx \frac{4}{3}F, \tag{4.4}$$

В частности, для $K = 4$ секций потенциальный прирост производительности — в 3 раза. Соответственно, изменится и соотношение на параметры системы 4.3:

$$BW = \frac{16}{3}F. \tag{4.5}$$

Альтернативный способ повысить эффективность SpMV, а также более сложных алгоритмов, задействующих условные переходы — расширение возможностей условного выполнения на CP2. В настоящий момент поддерживается только условное выполнение арифметических команд, а также выполнение одной и той же ветви программы во всех секциях (за счёт выполнения 0 или 1 итераций цикла). Для исполнения разных ветвей программы в разных секциях необходима, как минимум, поддержка условных инструкций чтения и записи в память. В процедуре SpMV таким образом возможно было бы, при сохранении последовательного доступа к памяти в четырёх секциях, снизить связанные с этим накладные расходы: в текущей реализации используется четыре условных копирования значений, считанных по четырём адресам, в регистры с одним и тем же номером во всех секциях.

При наличии, кроме того, раздельных счётчиков циклов станет возможным обрабатывать строки разной длины в разных секциях. Количество тактов на параллельную обработку K строк будет при этом определяться самой длинной из них, однако исчезнут накладные расходы на хранение и пересылку дополнительных нулей.

В алгоритмах, где вычисления, выполняемые в разных секциях CP2, не являются полностью независимыми, естественным образом требуется обмен данными между секциями. Простейший пример такой процедуры — скалярное произведение векторов, в конце которого требуется просуммировать значения, полученные в четырёх секциях. Аналогичная редукция требуется при вычислении нормы в алгоритме MG. В настоящий момент передача полученных значений из секции в секцию осуществляется через посредство регистров GPR, за

несколько команд копирования. Редукция также требуется и при вычислении SpMV, если входная матрица предварительно разбивается на блоки.

Более сложный случай — разностные операторы. При достаточно малых размерах массива (и/или при большем объёме локальной памяти) возможно было бы разделить его на 4 блока и обработать каждый в своей секции. Однако для вычисления следующего PO требуется произвести обмен гранями. Реализация обменов через регистры GPR была бы крайне неэффективной, особенно с учётом того, что этих регистров всего 16. Поэтому в текущей реализации даже для маленьких массивов обмены осуществляются через глобальную память: весь объём данных выгружается в неё, вызывается процедура копирования граней (в свою очередь, через DMA и CP2), затем обновлённые блоки снова загружаются в локальную память. Другими словами, все PO вычисляются по одному и перемежаются загрузками и выгрузками.

С учётом сказанного, представляется удобным иметь на CP2, помимо памяти четырёх секций, разделяемую локальную память меньшего объёма, доступ к которой имеют все секции — подобно локальной памяти современных GPU. С использованием этой памяти можно было бы эффективно осуществлять редукцию и различные обмены данными между секциями. В частности, для небольших массивов можно было бы вычислять последовательность PO автономно на CP2, без обращения к DMA и CPU — по схеме с обменами гранями, которая использовалась для реализации на GPU (разд. 2.2.1). Добавление такой разделяемой памяти означает добавление также набора адресных регистров. Кроме того, потребуются команды для пересылок данных между разделяемой памятью и собственной памятью секций, а также регистрами FPR.

Снижение накладных расходов на пересылки данных во многих алгоритмах возможно за счёт расширения функциональности контроллера DMA. В текущей версии контроллера и управляющей библиотеки обеспечена поддержка пересылок при условии выравненности адресов массивов в глобальной и в локальной памяти, а также длин, на 128 бит. В алгоритмах, где требуется доступ к массиву с более мелким шагом, это ограничение приходится обходить тем или иным способом — например, дополнять массивы нулями и пересылать области данных с избыточностью, как это было сделано в реализации алгоритма MG. В ряде процедур, входящих в БЦОС, используется также подход, когда для обработки на CP2 выделяется выравненная часть массивов, а оставшаяся невырав-

ненная часть обрабатывается на CPU. Это не только делает реализацию более трудоёмкой и менее прозрачной, но и снижает эффективность вычислений.

Описанных трудностей можно избежать, если снизить требование к выравненности данных для пересылки до 32 бит, т.е. 1 вещественного числа. Для PO, при вычислении которых используются дополнительные грани, в этом случае не требовалась бы пересылка и хранение в локальной памяти большого количества дополнительных нулей. В частности, это позволило бы обрабатывать на CP2 более крупные блоки.

Перспективное развитие этой идеи состоит в том, чтобы снять требование на организацию пересылаемых данных в глобальной памяти виде трёхмерного массива, в котором строки являются непрерывными в памяти: реализовать в контроллере DMA возможность формировать 128-разрядное слово для пересылки из четырёх произвольных 32-разрядных, по заданным пользователем адресам в глобальной памяти. Такого рода пересылки будут выполняться менее эффективно, чем пересылки выравненных трёхмерных массивов. Тем не менее, наличие этой возможности позволило бы более эффективно реализовать процедуры, в которых время вычислений на CP2 существенно превышает время пересылок.

В частности, для процедуры SpMV вместе с каждой строкой достаточно было бы передавать на CP2 только те элементы входного вектора, которые необходимы для её умножения, в порядке, заданном массивом номеров столбцов. Это означало бы:

- снятие ограничений на размер матрицы: можно обрабатывать в каждой секции одну строку или её фрагмент, который помещается в локальную память;
- отсутствие избыточности: пересылаются только те элементы, которые необходимы для вычислений;
- перенос логики случайного доступа к памяти с CP2 в контроллер системной памяти и в DMA, на этап формирования 128-разрядных слов для пересылки — при этом ядро на CP2 фактически свелось бы к скалярному произведению векторов.

Суммируя всё сказанное, предлагается внесение следующих доработок в архитектуру процессора:

- возможность обменов между локальной памятью и регистрами СР2 32-разрядными словами;
- раздельные наборы адресных регистров в секциях СР2;
- инструкции условной записи/чтения из локальной памяти СР2;
- раздельные счётчики циклов в секциях;
- общая память на СР2, разделяемая между секциями;
- доступ на запись к биту, определяющему направление БПФ, со стороны СР2;
- возможность пересылки по DMA произвольного набора 32-разрядных слов;
- увеличение количества регистров FPR;
- увеличение объёма локальной памяти СР2;
- доработка контроллера системной памяти.

Предварительные оценки показывают, что в совокупности эти усовершенствования способны обеспечить прирост производительности на процедурах БПФ и MG в 3 раза, а на процедуре SpMV — в 6 раз.

4.3 Достоинства и недостатки гибридных процессоров НИИСИ РАН в контексте высокопроизводительных вычислений

В настоящее время для высокопроизводительных вычислений, наряду с системами классической архитектуры на основе универсальных процессоров, широко применяются гибридные массивно-параллельные системы. Примерами являются системы на базе процессора STI Cell, а также гибридные узлы, включающие графические ускорители, и кластеры на их основе. Процессоры ВМ7 и ВМ9 линейки КОМДИВ, обладающие SIMD-сопроцессором СР2, несмотря на существенные количественные отличия от таких систем, сходны с ними по архитектурным решениям. Это отражено в построенной модели гибридной системы (разд. 1.3), которая охватывает и гибридные узлы с GPU, и процессоры ВМ7/9.

Проведённое исследование продемонстрировало, что в силу общности архитектур, при разработке высокопроизводительных вычислительных процедур

для гибридных процессоров НИИСИ РАН программист сталкивается с теми же задачами, что и при программировании для гибридных узлов с GPU:

1. распределение работы между управляющим процессором и сопроцессорами, выделение наиболее вычислительно-нагруженных частей алгоритма, которые потенциально могут быть ускорены на сопроцессорах;
2. выбор формата хранения данных, наиболее удобного для вычислений на сопроцессорах параллельной архитектуры;
3. разработка набора вычислительных ядер для сопроцессора, при помощи специальных программных средств;
4. эффективное использование системы памяти сопроцессора в ядрах;
5. сокращение избыточности при пересылках данных на сопроцессор и обратно, т.е. выполнение как можно большего количества вычислений над загруженными данными, до выгрузки результата;
6. организация пересылок данных по возможности в виде непрерывных и выравненных областей памяти, совмещение этих пересылок и вычислений на сопроцессорах.

Одно из ключевых различий между GPU и процессорами ВМ7/9, которое влияет на выбор схемы вычислений — относительно малый объём локальной памяти сопроцессора СР2. На GPU предпочтительными являются реализации, в которых основной объём данных, необходимых для вычислений, постоянно хранится в памяти ускорителей. Локальной памяти СР2 как правило не достаточно для этого, следовательно, приходится использовать схемы вычислений, связанные с постраничной пересылкой всего объёма данных в память секций и обратно. Как показывает пример многомерного БПФ больших массивов, на GPU при использовании таких схем вычислений пересылки данных через PCI Express становятся бутылочным горлышком. С другой стороны, на ВМ7/9 пропускная способность DMA более сбалансирована с пиковой производительностью сопроцессора — благодаря этому вычисления по таким схемам оказываются эффективными.

Другие существенные различия заключены в архитектуре самого сопроцессора СР2, который можно сравнить с одним вычислителем на современном GPU. И тот, и другой в сущности являются SIMD-процессорами и могут работать с векторными типами данных. Так, СР2 поддерживает инструкции для работы со 128-битными операндами, причём инструкции исполняются парал-

лельно в четырёх секциях, что в совокупности означает единовременную обработку 512-битных векторов (ту же разрядность имеет, например, векторное расширение AVX-512 для универсальных процессоров от Intel [91]). Однако на практике использование этих возможностей осложняется тем, что архитектура CP2 обладает меньшей универсальностью, чем GPU, в частности:

- единый блок генерации адреса не позволяет обращаться к локальной памяти по разным адресам в разных вычислительных секциях;
- в памяти 32-разрядные числа адресуются только парами или четвёрками;
- отсутствует память, разделяемая между секциями CP2 (аналог локальной памяти, присутствующей на каждом вычислителе GPU);
- отсутствует кэш-память.

Помимо перечисленного, на CP2 условные переходы в потоке инструкций поддерживаются в очень ограниченном объёме. Необходимо, однако, отметить, что и GPU, несмотря на наличие привычных конструкций языка программирования, не предназначен для эффективных расчётов по алгоритмам с большим количеством ветвлений: в случае, если потоки на одном вычислителе исполняют разные ветви программы, эти ветви исполняются последовательно. Такого рода вычисления целесообразно выполнять на универсальных процессорах.

Перечисленные особенности нельзя считать недостатками: CP2 изначально оптимизирован под алгоритмы потоковой обработки сигналов, при этом отказ от сложных архитектурных блоков позволил уменьшить площадь кристалла, снизить энергопотребление и стоимость. В частности, использование управляемой на уровне ПО локальной памяти, в качестве альтернативы кэш-памяти, является принципиальным архитектурным решением, которое обеспечивает более высокую эффективность регулярного доступа, за счёт усложнения программирования. Тот же принцип был положен в основу архитектуры Cell, а также GPU ранних моделей. Алгоритмы или их отдельные вычислительные стадии, в которых схема доступа к памяти нерегулярная, может быть выгоднее реализовать не на CP2 в его текущей версии, а на управляющем процессоре, обладающем кэш-памятью. С другой стороны, при расширении класса целевых задач для гибридных процессоров НИИСИ РАН внесение в архитектуру CP2 ряда изменений, описанных в предыдущем разделе, может оказаться оправданным.

Те же соображения касаются и функциональных возможностей DMA. Высокая пропускная способность достигается за счёт ограничений на расположение в памяти и выравненность передаваемых данных. Отсюда возникает избыточность: массивы необходимо дополнять нулями до нужного формата. Для сравнения, на GPU пересылки разреженных и невыравненных областей данных выполняются с низкой эффективностью, но тем не менее поддерживаются. Благодаря этому программист получает возможность в каждом конкретном случае найти компромисс между скоростью пересылок и избыточностью.

Общность между гибридными процессорами НИИСИ РАН и современными импортными гибридными системами проявляется и на более высоком уровне, при рассмотрении многопроцессорных комплексов на базе ВМ7/9 и гибридных кластеров. Здесь также применяются сходные методы разбиения задачи и схемы вычислений с обменами данными между процессорами/узлами. При этом пропускная способность RapidIO сбалансирована с производительностью процессоров КОМДИВ, поэтому потенциально возможна реализация распределённых процедур, работающих с большими объёмами данных, с незначительными потерями в производительности, относительно вычислений на одном процессоре.

К числу приоритетных направлений развития архитектуры процессоров НИИСИ РАН, которые не рассматривались в рамках диссертационного исследования, относится поддержка сопроцессором CP2 вычислений с двойной точностью, которая требуется в большинстве современных расчётных задач — в том числе и в наборе тестов NPB. Один из вариантов реализации — масштабирование всей архитектуры в 2 раза. При этом должна сохраниться возможность и для работы с вещественными числами одинарной точности. Также потребуется доработка блока вычисления элементарных функций. Ещё одна потенциальная возможность — расширение набора инструкций CP2 для эмуляции вычислений с двойной точностью на CP2 через вычисления с одинарной точностью; ряд методов для этого был разработан ранее для GPU. Такие вычисления с двойной точностью будут иметь заведомо более низкую производительность, чем реализованные в аппаратуре, однако могут рассматриваться как временное решение.

Подводя итоги, можно утверждать, что, при условии дальнейшей доработки архитектуры, гибридные многоядерные процессоры НИИСИ РАН имеют хороший потенциал для использования в ВПВ.

В дополнение необходимо сказать, что в проведённом исследовании основное внимание уделялось тому, чтобы добиться как можно более высокой производительности вычислений на той или иной архитектуре, без учёта затрат труда на программирование. Такая оптимизация возможна для критически важных задач. На практике использование ВС на базе процессоров ВМ7/9 в настоящий момент затруднено в связи с отсутствием поддержки стандартных средств программирования: ядра для CP2 программируются на языке ассемблера, для обменов данными по DMA, а также по RapidIO между процессорами, используется специализированный API. С этой точки зрения, поддержка MPI над RapidIO является первым шагом к применимости процессоров НИИСИ РАН в вычислениях не только специального, но и общего назначения.

Заключение

Основные результаты работы заключаются в следующем.

1. Разработана модель гибридной вычислительной системы, охватывающая широкий класс отечественных и зарубежных архитектур.
2. Разработан метод оценки ожидаемой производительности вычислительной процедуры на гибридных вычислительных системах; выведен формальный критерий сбалансированности вычислительной системы на заданной вычислительной процедуре.
3. Применимость метода подтверждена результатами измерения производительности разработанных автором реализаций нескольких широко применяемых в трёхмерном моделировании вычислительных процедур из набора тестов NAS Parallel Benchmarks на ряде отечественных и импортных вычислительных систем.
4. При помощи разработанного метода обоснована неулучшаемость разработанных оптимизированных библиотечных процедур БПФ и свёртки для отечественных гибридных процессоров ВМ7 и ВМ9.
5. Разработан проект оптимизации архитектуры гибридных процессоров НИИСИ РАН, позволяющий за счёт локальных усовершенствований программных моделей подсистем процессора достичь существенного роста производительности на классах вычислительных задач, рассмотренных в диссертации. Проект предполагает расширение функциональных возможностей контроллера DMA и сопроцессора CP2, а также улучшение нескольких количественных характеристик сопроцессора.

Разработанный метод оценки производительности позволяет априорно оценивать по порядку ожидаемую производительность вычислительной процедуры на гибридной вычислительной системе. Формальный критерий позволяет оценить также сбалансированность вычислительной системы на заданной процедуре.

Достоинством метода и критерия является то, что такие оценки могут быть получены априорно, до начала работ по реализации процедур на конкретной архитектуре, и не требуют значительных временных затрат. Кроме того, сопоставление реальной производительности каждой разработанной оптимизи-

рованной процедуры и выведенной при помощи метода оценки наглядно демонстрирует, какие особенности архитектуры не позволяют достичь теоретического максимума производительности и требуют доработки.

На гибридных процессорах оригинальной отечественной архитектуры с помощью предложенного метода впервые проведено исследование производительности набора процедур, применяемых в трёхмерном моделировании.

В планах работ по модернизации эмуляторов сопроцессора CP2 и контроллера DMA — реализация предложенного проекта оптимизации архитектуры.

Построенная модель гибридной системы в дальнейшем может быть расширена для рассмотрения других типов систем — например, многопроцессорных комплексов или систем с возможностью прямого обмена данными между сопроцессорами.

Разработанные процедуры БПФ и свёртки для BM7 в настоящий момент применяются в системах ЦОС реального времени. Полученный опыт разработки массивно-параллельных приложений, как под CP2, так и под GPU на OpenCL, может быть в дальнейшем использован при реализации масштабных вычислительных задач на различной аппаратуре.

Основное внимание в работе было уделено влиянию на производительность пропускной способности канала обменов данными между сопроцессорами и системной памятью, а также реализации вычислительных ядер — в том числе, эффективному доступу к памяти. Однако производительность вычислений и потенциал использования гибридной системы в вычислениях общего назначения определяется не только этими факторами. В частности, для гибридных процессоров НИИСИ РАН дальнейшего исследования требуют:

- архитектура контроллера доступа к системной памяти;
- организация эффективных обменов данными в многопроцессорном комплексе, в том числе выбор топологии сети RapidIO;
- обеспечение поддержки вычислений с вещественными числами двойной точности на CP2;
- совместное использование управляющего ядра и CP2 для вычислений, способы оптимального распределения работы между ними;
- оптимизация вычислений на управляющем ядре с использованием векторного расширения CPV [9].

Автор надеется, что внесённые предложения по дальнейшему развитию архитектуры будут учтены на практике и поспособствуют успеху отечественных технологий в области высокопроизводительных вычислений.

Выражаю благодарность Анатолию Георгиевичу Кушниренко, за неизменную поддержку и руководство на протяжении более чем десяти лет. Отдельная благодарность — Павлу Борисовичу Богданову, без увлечённости и деятельного участия которого проделанная работа не была бы возможной. Также хочу поблагодарить коллег из НИИСИ РАН: Райко Г. О., Павлова А. Н., Ткаченко Е. В., Кулешова А. С., Хропова М. С., — а также Ефремова А. А., за помощь в подготовке системного ПО и аппаратных стендов и за поддержку в преодолении многих трудностей, возникавших при проведении вычислительных экспериментов.

Список сокращений и условных обозначений

API Application Programming Interface — интерфейс программирования приложений

APU Accelerated Processing Unit — ускоренный процессор, гибридная однокристальная система, включающая CPU и GPU

ASIC Application-Specific Integrated Circuit — интегральная схема специального назначения

CG Conjugate Gradient — метод сопряжённых градиентов

CP2 сопроцессор вещественной арифметики в составе процессоров ВМ7 и ВМ9 линейки КОМДИВ

CPU Central Processing Unit — центральный (управляющий) процессор, например, универсальный процессор производства компании Intel

DDR Double Data Rate SDRAM — синхронная динамическая память с произвольным доступом и удвоенной скоростью передачи данных

DMA Direct Memory Access — канал прямого доступа к памяти

DMEM Device Memory — память сопроцессора в гибридной вычислительной системе

DRAM Dynamic Random Access Memory — динамическая память с произвольным доступом

FPGA Field-Programmable Gate Array — программируемая пользователем вентильная матрица (разновидность ПЛИС)

FPR Floating-Point Register — регистры CP2 для работы с вещественными числами с плавающей точкой

FT Fourier Transform — алгоритм на основе БПФ в составе NPB

GDDR Graphics DDR SDRAM — синхронная динамическая память с произвольным доступом и удвоенной скоростью передачи данных для использования в графических картах

GPR General-Purpose Register — регистры общего назначения сопроцессора CP2

GPU Graphics Processing Unit — графический ускоритель, или графическая карта

HBM High-Bandwidth Memory — память с высокой пропускной способностью, разновидность микросхем памяти

HMEM Host Memory — системная память в гибридной вычислительной системе

MG MultiGrid — алгоритм многосеточного метода

MPI Message-Passing Interface — интерфейс передачи сообщений

NPB NAS Parallel Benchmarks, Numerical Aerodynamics Simulation Parallel Benchmarks — набор тестов для параллельных суперкомпьютеров

NUMA Non-Uniform Memory Access — архитектура многопроцессорных систем с неравномерным доступом к памяти

OpenCL Open Computing Language — стандарт для программирования процессоров различной архитектуры, в том числе графических ускорителей

PPE PowerPC Processor Element — управляющее ядро в составе процессора STI Cell

RapidIO Rapid In-Out, РИО — интерфейс для обмена данными между процессорами

RISC Reduced Instruction Set Computer — архитектура компьютера с сокращённым набором команд

SDRAM Synchronous Dynamic Random Access Memory — синхронная динамическая память с произвольным доступом

SIMD Single Instruction, Multiple Data — модель параллельных вычислений с одиночным потоком команд и множественным потоком данных

SMP Symmetric Multiprocessing — симметричное мультипроцессирование в архитектуре многопроцессорных вычислительных систем

SnuCL — пакет программ для разработки OpenCL-приложений на гибридных кластерах

SoC System on a Chip — однокристальная система, в которой управляющий процессор и сопроцессоры располагаются на одном кристалле

SPE Synergistic Processor Element — сопроцессор в составе процессора STI Cell

SpMV Sparse Matrix-Vector Multiplication — умножение разреженной матрицы на вектор

VLIW Very Long Instruction Word — машинная инструкция, содержащая несколько команд, исполняющихся параллельно, и архитектура процессоров, поддерживающих такие инструкции

WG Work-Group — рабочая группа потоков в параллельном приложении, в терминологии OpenCL

WI Work-Item — поток в параллельном приложении, в терминологии OpenCL

XDR Extreme Data Rate DRAM — тип высокопроизводительных микросхем памяти

БПФ — алгоритм быстрого преобразования Фурье для вычисления ДПФ

БЦОС — библиотека цифровой обработки сигналов, разработанная в НИИСИ РАН

ВМ7 — микропроцессор КОМДИВ128-РИО, разработанный в НИИСИ РАН

ВМ7/9 — ВМ7 и ВМ9

ВМ9 — микропроцессор нового поколения КОМДИВ128-М, разработанный в НИИСИ РАН

ВПВ — высокопроизводительные вычисления

ВС — вычислительная система

ГБ — 10^9 байт

Гбайт — 2^{30} байт

ГОП/с — мера производительности вычислений: миллиардов (10^9) операций над вещественными числами с плавающей точкой в секунду

ДПФ — дискретное преобразование Фурье

Кбайт — 2^{10} байт

КОМДИВ — линейка микропроцессоров, разрабатываемая в НИИСИ РАН

МБ — 10^6 байт

Мбайт — 2^{20} байт

РО — линейный разностный оператор на сетке

МОП/с — миллионов (10^6) операций над вещественными числами с плавающей точкой в секунду

ПЛИС — программируемая логическая интегральная схема

ПО — программное обеспечение

ПетаОП/с — квадриллионов (10^{15}) операций над вещественными числами с плавающей точкой в секунду

СЛАУ — система линейных алгебраических уравнений

Топ500 — международный список самых производительных суперкомпьютеров, обновляемый ежегодно

ЦОС — цифровая обработка сигналов

ЭКБ — электронно-компонентная база

Список литературы

1. Библиотека цифровой обработки сигналов в режиме эмуляции для платформы x86. ЮКСУ.90973-01 / НИИСИ РАН. — Москва, 2015.
2. *Богданов П. Б.*, *Сударева О. Ю.* Гетерогенное программирование в рамках стандарта OpenCL // Супервычисления и математическое моделирование: труды XV Международной конференции, 13-17 октября 2014 г. / под ред. Р. М. Шагалиева. — Саров : ФГУП «РФЯЦ ВНИИЭФ», 2015. — С. 123—137.
3. *Богданов П. Б.*, *Сударева О. Ю.* Применение отечественных специализированных процессоров семейства КОМДИВ в научных расчётах // Информационные Технологии и Вычислительные Системы. — 2016. — Т. 3. — С. 45—65.
4. *Богданов П. Б.*, *Сударева О. Ю.* Производительность процессоров КОМДИВ на ряде типовых расчётных задач // Информационные Технологии и Вычислительные Системы. — 2017. — Т. 4. — С. 104—111.
5. *Богданов П.*, *Сударева О.* Применение отечественных специализированных процессоров семейства «КОМДИВ» в научных расчётах // Тезисы 14го Международного Междисциплинарного Семинара «Математические Модели и Моделирование в Лазерно-Плазменных Процессах и Передовых Научных Технологиях», г. Москва, 4-9 июля. — 2016. — URL: http://lppm3.ru/files/histofprog/LPpM3-2016-1-Programme.pdf (дата обращения: 23.08.2017).
6. *Витязев С. В.*, *Морозова С. А.*, *Савостьянов В. Ю.* Сравнение вычислительных возможностей процессоров «Эльбрус» и TMS320C66X в задачах цифровой обработки радиолокационных сигналов // REDS: Телекоммуникационные устройства и системы. — 2015. — Т. 5, № 3. — С. 272—275.
7. Государственная программа Российской Федерации «Развитие науки и технологий» на 2013-2020 годы: утв. постановлением Правительства Российской Федерации от 15 апреля 2014 г. № 301. — Москва. — URL: https://programs.gov.ru/Portal/programs/passport/15 (дата обращения: 21.03.2017).

8. Двухъядерная гетеронная система на кристалле «Эльбрус-2С+» / М. В. Исаев [и др.] // Вопросы радиоэлектроники. Сер. ЭВТ. — 2012. — № 3. — С. 42—52. — URL: http://www.mcst.ru/dvukhyadernaya-geterogennaya-sistema-na-kristalle-elbrus2s (дата обращения: 19.07.2017).

9. *Зубковский П. С.* Описание векторного сопроцессора процессора К64-М, версия 2.7 / НИИСИ РАН. — 2013.

10. *Ишин П. А.*, *Логинов В. Е.*, *Васильев П. П.* Ускорение вычислений с использованием высокопроизводительных математических и мультимедийных функций // Вестник воздушно-космической обороны. — 2015. — № 4(8). — С. 64—68. — URL: http://www.mcst.ru/files/52f220/590cd8/50136e/000004/ishin-loginov-vasilev-uskorenie_vychisleniy_s_ispolzovaniem_vysokoproizvoditelnyh_matematicheskih_i_multimediynyh_bibliotek_dlya_arhitektury_elbrus.pdf (дата обращения: 19.07.2017).

11. *Ким А. К.* Российские универсальные микропроцессоры и вычислительные комплексы высокой производительности: результаты и взгляд в будущее (к 20-летию ЗАО «МЦСТ») // Вопросы радиоэлектроники. Сер. ЭВТ. — 2012. — № 3. — С. 5—13. — URL: http://www.mcst.ru/rossiyskie_universalnye_mikroprotsessory_i_vk_vysokoy_proizvoditelnosti (дата обращения: 19.07.2017).

12. Конфигурация суперкомпьютеров / НИВЦ МГУ им. М. В. Ломоносова. — 2017. — URL: http://users.parallel.ru/wiki/pages/22-config (дата обращения: 19.07.2017).

13. Краткое описание архитектуры Эльбрус / АО «МЦСТ». — 2017. — URL: http://www.elbrus.ru/arhitektura_elbrus (дата обращения: 27.03.2017).

14. *Кулешов А. С.* Поддержка протокола MPI в ядре ОС Linux для многопроцессорных вычислительных комплексов на базе высокоскоростных каналов RapidIO // Программные продукты и системы. — 2015. — № 4. — С. 93—98.

15. *Логинов В. Е.*, *Ишин П. А.* Оптимизация для архитектуры «Эльбрус» быстрого преобразования Фурье применительно к 32-разрядным числам с плавающей точкой // Вопросы радиоэлектроники. Сер. ЭВТ. —

2012. — № 3. — С. 108—118. — URL: http://www.mcst.ru/optimizaciya-dlya-arkhitektury-elbrus-bystrogo-preobrazovaniya-fure-primenitelno-k-32razryadnym-chislam-s-plavayushhej-tochkoj (дата обращения: 19.07.2017).

16. *Максимов Д. Ю.*, *Филатов М. А.* Исследование нелинейных многосеточных методов решения задач однофазной фильтрации // Препринты ИПМ им. М. В. Келдыша. — 2011. — № 43. — 26 с. URL: http://library.keldysh.ru/preprint.asp?id=2011-43 (дата обращения: 12.10.2017).

17. Микросхема интегральная 1890ВМ7Я (КОМДИВ128-РИО). Указания по применению. ЮКСУ.431281.104Д4 / НИИСИ РАН. — Москва, 2009. — 371 с.

18. Микросхема интегральная 1890ВМ8Я. Указания по применению. ЮКСУ.431281.107Д4 / НИИСИ РАН. — Москва, 2016.

19. Новый 8-ядерный микропроцессор Эльбрус-8С / АО «МЦСТ». — 2017. — URL: http://www.mcst.ru/novyj-8yadernyj-mikroprocessor-elbrus-8c (дата обращения: 27.03.2017).

20. Оценка потенциала использования платформы Эльбрус для высокопроизводительных вычислений / С. С. Конюхов [и др.] // Суперкомпьютерные дни в России. Труды международной конференции. — 2016. — С. 373—385.

21. *Павлов А. Н.* Обзор коммуникационной среды RapidIO // Моделирование и визуализация. Многопроцессорные системы. Инструментальные средства разработки ПО / Сборник статей под редакцией академика РАН В. Б. Бетелина. — М. : НИИСИ РАН, 2009. — С. 105—122.

22. *Павлов А. Н.* Программная поддержка RapidIO // Моделирование и визуализация. Многопроцессорные системы. Инструментальные средства разработки ПО / Сборник статей под редакцией академика РАН В. Б. Бетелина. — М. : НИИСИ РАН, 2009. — С. 132—147.

23. *Павлов А. Н.* Формальная модель RapidIO // Моделирование и визуализация. Многопроцессорные системы. Инструментальные средства разработки ПО / Сборник статей под редакцией академика РАН В. Б. Бетелина. — М. : НИИСИ РАН, 2009. — С. 123—131.

24. Параллельные вычисления CUDA / NVIDIA Corporation. — 2017. — URL: http://www.nvidia.ru/object/cuda-parallel-computing-ru.html (дата обращения: 19.07.2017).

25. Пат. 2513759 Российская Федерация, МПК G 06 F 13/28 H 01 L 21/00. Гетерогенный процессор / П. Н. Осипенко, Е. А. Новожилов, А. Г. Кушниренко, Г. О. Райко. — Патентообладатель: Федеральное государственное бюджетное учреждение науки Российской академии наук Научно-исследовательский институт системных исследований РАН (НИИСИ РАН), № 2012146581/08; заявл. 01.11.2012, опубл. 20.04.2014, Бюл. № 11. — 9 с.

26. Программное изделие Ассемблер для специализированного сопроцессора СР2 в составе микропроцессора КОМДИВ128-РИО (АССК128). Руководство программиста. ЮКСУ.90986-01 33 01 / НИИСИ РАН. — Москва, 2013. — 67 с.

27. *Райко Г. О.*, *Павловский Ю. А.*, *Мелькановић В. С.* Технология программирования многопроцессорной обработки гидроакустических сигналов на вычислительных устройствах семейства «КОМДИВ» // Гидроакустика. — СПб., ОАО «Концерн "Океанприбор" », 2014. — № 20(2). — С. 85—92.

28. Распараллеливание на графические процессоры тестов NAS NPB3.3.1 на языке Fortran DVMH / В. Ф. Алексахин [и др.] // Вестник Уфимского государственного авиационного технического университета. — 2015. — Т. 19, № 1. — С. 240—250.

29. Российский программно-аппаратный комплекс для инженерных расчетов FlowVision на платформе Эльбрус / АО «МЦСТ». — 2017. — URL: http://www.elbrus.ru/rossijskij-programmnoapparatnyj-kompleks-dlya-inzhenernykh-raschetov-flowvision-na-platforme-elbrus (дата обращения: 29.03.2017).

30. Свидетельство о государственной регистрации программы для ЭВМ № 2017617058. Библиотека цифровой обработки сигналов для микропроцессора КОМДИВ128-РИО для ОС РВ Багет 3.5 (БЦОС 3.5) / Г. О. Райко, О. Ю. Сударева, М. С. Хропов, М. С. Аристов / Правообладатель: Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук» (ФГУ ФНЦ НИИСИ РАН). — 22 июня 2017.

31. Сервер Эльбрус-4.4 / ПАО «ИНЭУМ им. И.С. Брука». — 2017. — URL: http://www.ineum.ru/server_elbrus-4.4 (дата обращения: 27.03.2017).

32. Стратегия развития электронной промышленности России на период до 2025 года: утв. приказом Министерства промышленности и энергетики РФ от 7 августа 2007 г. № 311 // Еженедельник промышленного роста. — 24-30.09.2007. — № 31. — Документ предоставлен КонсультантПлюс: http://www.consultant.ru (дата сохранения: 17.03.2017).

33. *Сударева О. Ю.* Эффективная реализация алгоритмов быстрого преобразования Фурье и свёртки на микропроцессоре КОМДИВ128-РИО. — М. : НИИСИ РАН, 2014. — 266 с.

34. *Сударева О. Ю.* Реализация алгоритма MG из пакета NPB для многопроцессорного вычислительного комплекса на базе микропроцессора КОМДИВ128-РИО // Труды НИИСИ РАН. — 2015. — Т. 5, № 1. — С. 75—87.

35. *Сударева О. Ю.* Распределённые вычисления на процессорах КОМДИВ на примере алгоритма NPB MG // Сборник научных статей по итогам международной научно-практической конференции, г. Санкт-Петербург, 22-23 декабря 2017. — СПб : Изд-во «КультИнформПресс», 2017. — С. 60—63.

36. *Сударева О. Ю.* Развитие микропроцессоров линейки КОМДИВ для применений в научных расчётах: предложения по оптимизации архитектуры // Современные научные исследования и разработки. — 2018. — № 2(19). — С. 295—301. — URL: http://olimpiks.ru/d/1340546/d/zhurnal_219.pdf (дата обращения: 15.03.2018).

37. Суперкомпьютер RoadRunner / Лаборатория Параллельных информационных технологий НИВЦ МГУ. — 2008. — URL: http://parallel.ru/computers/reviews/RoadRunner.html (дата обращения: 25.08.2017).

38. СуперЭВМ К-100 / ИПМ им. М. В. Келдыша РАН. — 2017. — URL: http://www.kiam.ru/ (дата обращения: 19.07.2017).

39. A Multilevel Parallelization Framework for High-Order Stencil Computations / H. Dursun [et al.] // Lecture Notes in Computer Science — Euro-Par 2009 Parallel Processing. — 2009. — P. 642–653.

40. Adaptive Line Size Cache for Irregular References on Cell Multicore Processor / C. Cao [et al.] // Proceedings of the 2010 IFIP International Conference on Network and Parallel Computing. — 2010. — P. 314–328.

41. AMD APP SDK OpenCL Optimization Guide / Advanced Micro Devices, Inc. — August 2015. — URL: http://developer.amd.com/wordpress/media/2013/12/AMD_OpenCL_Programming_Optimization_Guide2.pdf (access date: 19.07.2017).

42. An OpenCL Framework for Heterogeneous Multicores with Local Memory / J. Lee [et al.] // Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques. — 2010. — P. 193–204.

43. ARM(R) Architecture Reference Manual. ARMv8, for ARMv8-A architecture profile / ARM Limited. — September 25, 2017. — URL: https://static.docs.arm.com/ddi0487/bb/DDI0487B_b_armv8_arm.pdf (access date: 12.10.2017).

44. *Bell N.*, *Garland M.* Efficient Sparse Matrix-Vector Multiplication on CUDA : tech. rep. / NVIDIA Corporation. — December 2008. — NVR-2008-004. — NVIDIA Technical Report.

45. *Bell N.*, *Garland M.* Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors // Proceedings of SC'09. — 2009. — P. 1–11.

46. *Betelin V. B., Kushnirenko A. G., Smirnov N. N., Nikitin V. F., Tyurenkova V. V., Stamov L. I.* Numerical investigations of hybrid rocket engines // Acta Astronautica. — 2018. — Vol. 144. — P. 363–370.

47. *Betelin V. B., Smirnov N. N., Nikitin V. F., Dushin V. R., Kushnirenko A. G., Nerchenko V. A.* Evaporation and ignition of droplets in combustion chambers modeling and simulation // Acta Astronautica. — 2012. — Vol. 70. — P. 23–35.

48. Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX) in hypre and PETSc / A. V. Knyazev [et al.] // SIAM Journal on Scientific Computing. — 2007. — Vol. 29, no. 5. — P. 2224–2239.

49. *Bogdanov P.*, *Efremov A.*, *Sudareva O.* Heterogeneous programming methodology based on OpenCL framework // Proceedings of High Performance Computing 2013, Kyiv, October 7-11. — 2013. — P. 392–392. — URL: http://hpc-ua.org/hpc-ua-13/files/proceedings/74.pdf (access date: 23.08.2017).

50. *Burgess D. A.*, *Giles M.* Renumbering unstructured grids to improve the performance of codes on hierarchical memory machines // Advances in Engineering Software. — 1996. — Vol. 28(3). — P. 189–201.

51. *Burrus C. S.* Fast Fourier Transforms. — 2008. — URL: http://cnx.org/content/col10550/latest (access date: 19.07.2017).

52. *Caratori Tontini F.*, *Cocchi L.*, *Carmisciano C.* Rapid 3-D forward model of potential fields with application to the Palinuro Seamount magnetic anomaly (southern Tyrrhenian Sea, Italy) // Journal of Geophysical Research: Solid Earth. — 2009. — Vol. 114. — B02103.

53. *Chen Y.*, *Cui X.*, *Mei H.* Large-Scale FFT on GPU clusters // Proceedings of the 24th ACM International Conference on Supercomputing. — 2010. — P. 315–324.

54. *Choi J. W.*, *Singh A.*, *Vuduc R. W.* Model-driven Autotuning of Sparse Matrix-vector Multiply on GPUs // Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. — 2010. — P. 115–126.

55. *Cooley J. W.*, *Tukey J. W.* An algorithm for the machine calculation of complex Fourier series // Math. comput. — 1965. — No. 19. — P. 297–301.

56. Cray XC Series Network / B. Alverson [et al.] ; Cray Inc. — 2012. — URL: http://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf (access date: 24.07.2017).

57. CUDA 8 Performance Overview / NVIDIA Corporation. — 2016. — URL: http://developer.download.nvidia.com/compute/cuda/compute-docs/cuda-performance-report.pdf (access date: 19.07.2017).

58. *Da Graça G.*, *Defour D.* Implementation of float-float operators on graphics hardware // Proceedings, In 7th conference on Real Numbers and Computers, RNC7. — 2006. — P. 23–32.

59. *Daga M.*, *Aji A. M.*, *Feng W.* On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing // 2011 Symposium on Application Accelerators in High-Performance Computing (SAAHPC). — 2011. — URL: http://saahpc.ncsa.illinois.edu/11/presentations/daga.pdf (access date: 19.07.2017).

60. *Dang H.-V.*, *Schmidt B.* The Sliced COO Format for Sparse Matrix-Vector Multiplication on CUDA-enabled GPUs // Procedia Computer Science. — 2012. — Vol. 9. — P. 57–66.

61. *Davis T. A.*, *Hu Y.* The university of Florida sparse matrix collection // ACM Transactions on Mathematical Software. — 2011. — Vol. 38(1). — Article No. 1.

62. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU / V. W. Lee [et al.] // Proceedings of the 37th annual international symposium on Computer architecture. — 2010. — P. 451–460.

63. *Dongarra J.* Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard // International Journal of High Performance Computing Applications. — 2002. — Vol. 16, no. 1. — P. 1–111.

64. *Dongarra J.* Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard // International Journal of High Performance Computing Applications. — 2002. — Vol. 16, no. 2. — P. 115–199.

65. *Dongarra J. J.*, *Hey T.*, *Strohmaier E.* Selected results from the ParkBench Benchmark // Proceedings of Euro-Par'96 Parallel Processing. — 1996. — P. 251–254.

66. *Dongarra J. J.*, *Luszczek P.*, *Petitet A.* The LINPACK benchmark: Past, present and future // Concurrency and Computation: Practice and Experience. — 2003. — Vol. 15. — P. 1–18.

67. *Dongarra J.*, *Sullivan F.* Guest Editors introduction to the top 10 algorithms // Computing in Science Engineering. — 2000. — Vol. 2(1). — P. 22–23.

68. *Dziekonski A.*, *Lamecki A.*, *Mrozowski M.* A Memory Efficient and Fast Sparse Matrix Vector Product on a GPU // Progress In Electromagnetics Research. — 2011. — Vol. 116. — P. 49–63.

69. ELLPACK — Software for Solving Elliptic Problems / Purdue University. — 2017. — URL: https://www.cs.purdue.edu/ellpack/ (access date: 19.07.2017).

70. *Fatahalian K.*, *Sugerman J.*, *Hanrahan P.* Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication // Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware. — New York, 2004. — P. 133–137.

71. *Feldman M.* Pondering AMD's Ambitions for High-Performance APUs. — 2016. — URL: https://www.top500.org/news/pondering-amds-ambitions-for-high-performance-apus/ (access date: 19.07.2017).

72. *Frigo M.*, *Johnson S. G.* The design and implementation of FFTW3 // Proceedings of the IEEE. — 2005. — Vol. 93(2). — P. 216–231. — URL: http://www.fftw.org/fftw-paper-ieee.pdf (access date: 19.07.2017).

73. From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming / P. Du [et al.] // Journal Parallel Computing. — 2012. — Vol. 38(8). — P. 391–407.

74. *Fuller S.* The Opportunity for Sub Microsecond Interconnects for Processor Connectivity. — 2017. — URL: http://www.rapidio.org/technology-comparisons/ (access date: 19.07.2017).

75. *Greathouse J. L.*, *Daga M.* Efficient sparse matrix-vector multiplication on GPUs using the CSR storage format // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. — 2014. — P. 769–780.

76. *Grimes R.*, *Kincaid D.*, *Young D.* ITPACK 2.0 User's Guide : tech. rep. / Center for Numerical Analysis, University of Texas. — 1979. — CNA–150.

77. *Haase G.*, *Langer U.* Multigrid methods: from geometrical to algebraic versions // Modern Methods in Scientific Computing and Applications / ed. by A. Bourlioux, M. J. Gander. — Dordrecht : Kluwer Academic Press, 2002. — P. 103–153. — Vol. 75 in the NATO Science Ser. II, Mathematics, Physics and Chemistry.

78. *Harris M.* How NVLink Will Enable Faster, Easier Multi-GPU Computing / NVIDIA Corporation. — 2014. — URL: https://devblogs.nvidia.com/parallelforall/how-nvlink-will-enable-faster-easier-multi-gpu-computing/ (access date: 12.10.2017).

79. *Hénon P.*, *Saad Y.* A Parallel Multistage ILU Factorization Based on a Hierarchical Graph Decomposition // SIAM Journal on Scientific Computing. — 2006. — Vol. 28(6). — P. 2266–2293.

80. High Bandwidth Memory, Reinventing Memory Technology / Advanced Micro Devices, Inc. — 2015. — URL: http://www.amd.com/en-us/innovations/software-technologies/hbm (access date: 19.07.2017).

81. High-Performance Computing Using FPGAs / ed. by W. Vanderbauwhede, K. Benkrid. — New York : Springer-Verlag, 2013. — XI, 803 p.

82. *Holewinski J.*, *Pouchet L.-N.*, *Sadayappan P.* High-performance Code Generation for Stencil Computations on GPU Architectures // Proceedings of the 26th ACM international conference on Supercomputing. — 2012. — P. 311–320.

83. HPFBench: a high performance Fortran benchmark suite / Y. C. Hu [et al.] // ACM Transactions on Mathematical Software. — 2000. — Vol. 26(1). — P. 99–149.

84. Hybrid-parallel sparse matrix-vector multiplication with explicit communication overlap on current multicore-based systems / G. Schubert [et al.] // Parallel Processing Letters. — 2011. — Vol. 21(3). — P. 339–358.

85. Hyper-Threading Technology Architecture and Microarchitecture / D. T. Marr [et al.] // Intel Technology Journal. — 2002. — Vol. 6(1). — P. 1–12. — URL: https://www.cs.sfu.ca/ fedorova/Teaching/CMPT886/Spring2007/papers/ hyper-threading.pdf (access date: 23.08.2017).

86. IBM BladeCenter QS20 blade with new Cell BE processor offers unique capabilities for graphic-intensive, numeric applications (Hardware Announcement) / IBM. — 2006. — URL: http://www-01.ibm.com/common/ssi/rep_ca/7/897/ENUS106-677/ENUS106-677.PDF (access date: 21.09.2017).

87. *Ibrahim K. Z.* Chapter 36: Code Development of High-Performance Applications for Power-Efficient Architectures // Handbook of energy-aware and green computing / ed. by I. Ahmad, S. Ranka. — Chapman & Hall / CRC, 2012. — P. 835–854.

88. *Im E.-J.*, *Yelick K.*, *Vuduc R.* Sparsity: Optimization Framework for Sparse Matrix Kernels // International Journal of High Performance Computing Applications. — 2004. — Vol. 18(1). — P. 135–158.

89. Implementation of 3D FFTs Across Multiple GPUs in Shared Memory Environments / N. Nandapalan [et al.] // Proceedings of the 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies. — 2012. — P. 167–172.

90. Improving the Performance of the Sparse Matrix Vector Product with GPUs / F. Vazquez [et al.] // Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology. — 2010. — P. 1146–1151.

91. Intel Architecture Instruction Set Extensions Programming Reference / Intel Corporation. — 2017. — URL: https://software.intel.com/sites/default/files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf (access date: 24.07.2017).

92. Intel Xeon Phi Core Micro-architecture / Intel Corporation. — 2013. — URL: http://software.intel.com/en-us/articles/intel-xeon-phi-core-micro-architecture (access date: 19.07.2017).

93. Interconnect Analysis: 10GigE and InfiniBand in High Performance Computing / HPC Advisory Council. — 2009. — URL: http://www.hpcadvisorycouncil.com/pdf/IB_and_10GigE_in_HPC.pdf (access date: 19.07.2017).

94. Introduction to the Cell multiprocessor / J. A. Kahle [et al.] // IBM Journal of Research and Development. — 2005. — Vol. 49(4/5). — P. 589–604.

95. *Jin G.*, *Endo T.*, *Matsuoka S.* A Parallel Optimization Method for Stencil Computation on the Domain that is Bigger than Memory Capacity of GPUs // Proceedings of the 2013 IEEE International Conference on Cluster Computing. — 2013. — P. 1–8.

96. *Karypis G.*, *Kumar V.* Parallel Multilevel Graph Partitioning // Proceedings of the 10th International Parallel Processing Symposium. — 1996. — P. 314–319.

97. *Krishnan D.*, *Szeliski R.* Multigrid and Multilevel Preconditioners for Computational Photography // ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2011). — 2011. — Vol. 30(5). — Article No. 177.

98. *Kumar M.* Comparing TI's TMS320C6671 DSP with ADI's ADSP-TS201S TigerSHARC(R) Processor (Texas Instruments white paper). — 2012. — URL: http://www.ti.com/lit/wp/sprabn8a/sprabn8a.pdf (access date: 21.09.2017).

99. *Kumar V.*, *Katti C. P.*, *Saxena P. C.* A Novel Task Scheduling Algorithm for Heterogeneous Computing // International Journal of Computer Applications. — 2014. — Vol. 85, no. 18. — P. 35–39.

100. *Larsen E. S.*, *McAllister D.* Fast matrix multiplies using graphics hardware // Proceedings of the 2001 ACM/IEEE conference on Supercomputing. — New York, 2001. — P. 55–60.

101. *Lebensohn R.* N-site modeling of a 3D viscoplastic polycrystal using Fast Fourier Transform // Acta mater. — 2001. — No. 49. — P. 2723–2737.

102. *Li D.*, *Huang S.*, *Cameron K.* CG-Cell: An NPB Benchmark Implementation on Cell Broadband Engine // Proceedings of the 9th international conference on Distributed computing and networking. — 2008. — P. 263–273.

103. *Li X.*, *Blinka E.* Very large FFT for TMS320C6678 processors: white paper / Texas Instruments Inc. — 2015. — URL: http://www.ti.com/lit/wp/spry277/spry277.pdf (access date: 21.09.2017).

104. LU-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphics Hardware / N. Galoppo [et al.] // Proceedings of the 2005 ACM/IEEE conference on Supercomputing. — 2004. — P. 3–14. — URL: http://gamma.cs.unc.edu/LU-GPU/ (access date: 23.08.2017).

105. *Margiolas C.*, *O'Boyle M. F. P.* Portable and Transparent Host-Device Communication Optimization for GPGPU Environments // Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization. — 2014. — URL: http://cgo.org/cgo2014/wp-content/uploads/2013/05/Host-Device_Communication_Optimization_GPU.pdf (access date: 23.08.2017).

106. *Mathew J.*, *Vijayakumar D. R.* The Performance of Parallel Algorithms by Amdahl's Law, Gustafson's Trend // International Journal of Computer Science and Information Technologies. — 2011. — Vol. 2(6). — P. 2796–2799.

107. *Mittal S.*, *Vetter J. S.* A survey of CPU-GPU heterogeneous computing techniques // ACM Computing Surveys. — 2015. — Vol. 47(4). — Article No. 1.

108. *Mittal S.*, *Vetter J. S.* A Survey of Methods For Analyzing and Improving GPU Energy Efficiency // ACM Computing Surveys. — 2015. — Vol. 47(2). — Article No. 19.

109. *Monakov A.*, *Lokhmotov A.*, *Avetisyan A.* Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures // Proceedings of HiPEAC 2010. — 2010. — P. 111–125.

110. *Moreland K.*, *Angel E.* The FFT on a GPU // Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware 2003. — 2003. — P. 112–119.

111. MPI: A Message-Passing Interface Standard / Message Passing Interface Forum. — 2015. — URL: http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf (access date: 24.07.2017).

112. NAS Parallel Benchmarks for GPGPUs Using a Directive-Based Programming Model / R. Xu [et al.] // Proceedings of the 27th International Workshop on Languages and Compilers for Parallel Computing. — 2015. — P. 67–81.

113. Optimization of Sparse Matrix-vector Multiplication on Emerging Multicore Platforms / S. Williams [et al.] // Proceedings of the 2007 ACM/IEEE Conference on Supercomputing. — New York, USA, 2007. — 38:1–38:12.

114. Performance Analysis of the SiCortex SC072 / B. J. Martin [et al.] // Sandia National Laboratories, Albuquerque, NM. — 2008. — URL: https://www.researchgate.net/publication/253434862_Performance_ Analysis_of_the_SiCortex_SC072 (access date: 24.07.2017).

115. Performance and Portability with OpenCL for Throughput-Oriented HPC Workloads Across Accelerators, Coprocessors, and Multicore Processors / C. Cao [et al.] // Proceedings of the 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems. — 2014. — P. 61–68.

116. Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi / A. Ramachandran [et al.] // Proceedings of the 2013 42nd International Conference on Parallel Processing. — 2013. — P. 736–743.

117. PETSc Users Manual : tech. rep. / S. Balay [et al.] ; Argonne National Laboratory. — 2016. — ANL–95/11, Revision 3.7. — URL: http://www.mcs.anl.gov/petsc (access date: 21.08.2017).

118. Physis: An Implicitly Parallel Programming Model for Stencil Computations on Large-Scale GPU-Accelerated Supercomputers / N. Maruyama [et al.] // Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. — 2011. — Article No. 11.

119. *Pinar A.*, *Heath M. T.* Improving performance of sparse matrix-vector multiplication // Proceedings of the 1999 ACM/IEEE conference on Supercomputing. — 1999. — Artice No. 30.

120. RapidIO based Low Latency Heterogeneous Supercomputing // CERN Openlab Day 2015 Presentation. — 2015. — URL: https://indico.cern.ch/event/389301/contributions/1822804/attachments/778387/1067372/CERN_OCP_HPC_IDT_Interconnect_with_RapidIO.pdf (access date: 21.08.2017).

121. RISC-V: The Free and Open RISC Instruction Set Architecture / RISC-V Foundation. — 2017. — URL: https://riscv.org/ (access date: 19.07.2017).

122. Scientific Computing Kernels on the Cell Processor / S. Williams [et al.] // International Journal of Parallel Programming. — 2007. — Vol. 35(3). — P. 263–298.

123. *Seo S.*, *Jo G.*, *Lee J.* Performance characterization of the NAS Parallel Benchmarks in OpenCL // Proceedings of the IEEE International Symposium on Workload Characterization. — 2011. — P. 137–148.

124. *Shaffer A.*, *Einfalt B.*, *Raghavan P.* PFFTC: An improved fast Fourier transform for the IBM Cell Broadband Engine // Procedia Computer Science. — 2010. — Vol. 1(1). — P. 1045–1054.

125. *Shimokawabe T.*, *Aoki T.*, *Onodera N.* A High-productivity Framework for Multi-GPU computation of Mesh-based applications // Proceedings of the 1st International Workshop on High-Performance Stencil Computations. — 2014. — P. 23–30.

126. *Smith L.*, *Bull M.* Development of mixed mode MPI/OpenMP applications // Scientific Programming. — 2001. — Vol. 9. — P. 83–98.

127. *Smith R.*, *Garg R.* NVIDIA's GeForce GTX Titan Review, Part 2: Titan's Performance Unveiled. — 2013. — URL: http://www.anandtech.com/show/6774/nvidias-geforce-gtx-titan-part-2-titans-performance-unveiled (access date: 19.07.2017).

128. SnuCL: an OpenCL Framework for Heterogeneous CPU/GPU Clusters / J. Kim [et al.] // Proceedings of the 26th International Conference on Supercomputing. — 2012. — P. 341–352.

129. Sparse matrix-vector multiplication on GPGPU clusters: A new storage format and a scalable implementation / M. Kreutzer [et al.] // IPDPS Workshops, IEEE Computer Society. — 2012. — P. 1696–1702.

130. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures / C. Augonnet [et al.] // Concurrency and Computation: Practice & Experience — Euro-Par 2009. — 2011. — Vol. 23(2). — P. 187–198.

131. Stencil Computation Optimization and Auto-tuning on State-of-the-Art Multicore Architectures / K. Datta [et al.] // Proceedings of the 2008 ACM/IEEE conference on Supercomputing. — 2008. — Article No. 4.

132. *Su B.-Y.*, *Keutzer K.* clSpMV: A Cross-Platform OpenCL SpMV Framework on GPUs // Proceedings of the 26th ACM international conference on Supercomputing. — 2012. — P. 353–364.

133. *Sun X.-H.*, *Chen Y.* Reevaluating Amdahl's law in the multicore era // J. Parallel Distrib. Comput. — 2010. — Vol. 70(2). — P. 183–188.

134. *Temam O.*, *Jalby W.* Characterizing the behavior of sparse algorithms on caches // Proceedings of the 1992 ACM/IEEE conference on Supercomputing. — 1992. — P. 578–587.

135. *Thall A.* Extended-precision Floating-point Numbers for GPU Computation // ACM SIGGRAPH 2006 Research Posters. — 2006. — Article No. 52.

136. The Future of Computing Performance: Game Over or Next Level? / ed. by S. H. Fuller, L. I. Millett. — Washington, D. C. : The National Academies Press, 2011. — 200 p.

137. The Green500 / CompuGreen LLC. — 2017. — URL: http://www.green500.org (access date: 19.07.2017).

138. The NAS Parallel Benchmarks : tech. rep. / D. H. Bailey [et al.] ; NASA Ames Research Center, Moffet Field, CA 94035, USA. — March 1991, revised 1994. — RNR-94-007.

139. The NAS Parallel Benchmarks web page / NASA Advanced Supercomputing Division. — 2015. — URL: http://www.nas.nasa.gov/publications/npb.html (access date: 19.07.2017).

140. The OpenCL Specification, version 2.1 / Khronos OpenCL Working Group ; ed. by L. Howes. — 2015. — URL: https://www.khronos.org/registry/OpenCL/specs/opencl-2.1.pdf (access date: 19.07.2017).

141. The Potential of the Cell Processor for Scientific Computing / S. Williams [et al.] // Proceedings of the 3rd conference on Computing frontiers. — 2006. — P. 9–20.

142. The TH Express High Performance Interconnect Networks / Z. Pang [et al.] // Frontiers of Computer Science. — 2014. — Vol. 8(3). — P. 357–366.

143. *Thöns C.* Parallelizing Multigrid Solvers for Contact Problems on IBM's Cell Processor : PhD thesis / Thöns C. — Berlin, Germany : Freie Universität Berlin, 07/2008. — URL: http://publications.imp.fu-berlin.de/151/1/_ma_tesis.pdf (access date: 21.08.2017).

144. Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P / TOP500.org. — 2017. — URL: https://www.top500.org/system/177999 (access date: 19.07.2017).

145. TigerSHARC Embedded Processor ADSP-TS201S / Analog Devices. — 2006. — URL: http://www.analog.com/media/en/technical-documentation/data-sheets/ADSP_TS201S.pdf (access date: 24.07.2017).

146. TMS320C66x multicore DSPs for high-performance computing / Texas Instruments, Inc. — 2011. — URL: http://www.farnell.com/datasheets/1737036.pdf (access date: 21.09.2017).

147. *Toledo S.* Improving the memory-system performance of sparse-matrix vector multiplication // IBM Journal of Research and Development. — 1997. — Vol. 41(6). — P. 711–726.

148. Top 500. The List / TOP500.org. — 2017. — URL: https://www.top500.org (access date: 19.07.2017).

149. *Tullsen D. M.*, *Eggers S. J.*, *Levy H. M.* Simultaneous Multithreading: Maximizing On-Chip Parallelism // Proceedings of the 22nd Annual International Symposium on Computer Architecture. — 1995. — P. 392–403.

150. *Van Loan C.* Computational Frameworks for the Fast Fourier Transform. — Philadelphia, PA, USA : Society for Industrial, Applied Mathematics, 1992. — 273 p.

151. *Vastenhouw B.*, *Bisseling R. H.* A Two-Dimensional Data Distribution Method For Parallel Sparse Matrix-Vector Multiplication // SIAM Review. — 2002. — Vol. 47(1). — P. 67–95.

152. *Wang Z.*, *Grewe D.*, *O'Boyle M. F. P.* Automatic and Portable Mapping of Data Parallel Programs to OpenCL for GPU-Based Heterogeneous Systems // ACM Transactions on Architecture and Code Optimization (TACO). — 2015. — Vol. 11(4). — Article No. 42.

153. When cache blocking of sparse matrix vector multiply works and why / R. Nishtala [et al.] // Applicable Algebra in Engineering, Communication and Computing. — 2007. — Vol. 18(3). — P. 297–311.

154. Writing Efficient Floating-Point FFTs for ADSP-TS201 TigerSHARC(R) Processors (EE-218), Rev. 2 / Analog Devices, Inc. — 2004. — URL: http://www.analog.com/media/en/technical-documentation/application-notes/EE-218.pdf (access date: 21.09.2017).

155. *Wulf W. A.*, *McKee S. A.* Hitting the memory wall: Implications of the obvious // Computer Architecture News. — 1995. — Vol. 23(1). — P. 20–24.

156. *Yang U. M.* Parallel algebraic multigrid methods — high performance preconditioners // Numerical Solution of Partial Differential Equations on Parallel Computers. — Berlin, Heidelberg : Springer, 2006. — P. 209–236.

157. *Yang X.*, *Parthasarathy S.*, *Sadayappan P.* Fast Sparse Matrix-vector Multiplication on GPUs: Implications for Graph Mining // Proceedings of the VLDB Endowment. — 2011. — Vol. 4(4). — P. 231–242.

158. *Ye F.*, *Calvin C.*, *Petiton S. G.* A Study of SpMV Implementation Using MPI and OpenMP on Intel Many-Core Architecture // High Performance Computing for Computational Science — VECPAR 2014. — 2014. — P. 43–56.

Список рисунков

Список таблиц

Приложение А

Параметры вычислительных систем и процедур

Таблица А.1 — Примеры гибридных установок: параметры модели

	Узел НИИСИ РАН	**Узел суперкомпьютера К100**
CPU	Intel Xeon E5-2670	Intel Xeon X5670
GPU	NVIDIA GeForce GTX TITAN	NVIDIA Tesla C2050
Конфигурация	2 × CPU + 4 × GPU + PCI Express	2 × CPU + 3 × GPU + PCI Express
K	4	3
HMEM	128 ГБ	96 ГБ
BW	32 ГБ/с	16 ГБ/с
DMEM	6 ГБ	2,5 ГБ
$\mathbf{bw_{CP}}$	288 ГБ/с	144 ГБ/с
DPEAK	1500 ГОП/с	515 ГОП/с
HPEAK	330 ГОП/с	140 ГОП/с

Таблица А.2 — Микропроцессоры ВМ7 и ВМ9: параметры модели

	K	**DMEM**	**DPEAK**	$\mathbf{bw_{CP}}$	**BW**	**HMEM**
ВМ7	4	64 Кбайт	2 ГОП/с	3,2 ГБ/с	2,7 ГБ/с	768 Мбайт
ВМ9	4	64 Кбайт	10 ГОП/с	16 ГБ/с	8,4 ГБ/с	2 Гбайт

Таблица А.3 — Классы задач FT, MG и CG

Тест	**Параметр**	**S**	**W**	**A**	**B**	**C**	**D**	**E**
FT	размер решётки	$64 \times 64 \times 64$	$128 \times 128 \times 32$	$256 \times 256 \times 128$	$512 \times 256 \times 256$	$512 \times 512 \times 512$	$2048 \times 1024 \times 1024$	$4096 \times 2048 \times 2048$
	объём данных	2 Мбайт	4 Мбайт	64 Мбайт	256 Мбайт	1 Гбайт	16 Гбайт	128 Гбайт
	число итераций	6	6	6	20	20	25	25
MG	размер решётки	$32 \times 32 \times 32$	$128 \times 128 \times 128$	$256 \times 256 \times 256$	$256 \times 256 \times 256$	$512 \times 512 \times 512$	$1024 \times 1024 \times 1024$	$2048 \times 2048 \times 2048$
	объём данных	0,25 Мбайт	16 Мбайт	128 Мбайт	128 Мбайт	1 Гбайт	8 Гбайт	64 Гбайт
	число итераций	4	4	4	20	20	50	50
CG	число строк	1400	7000	14000	75000	150000	1500000	9000000
	объём данных (матрица / вектор)	0,7 Мбайт / 10 Кбайт	4 Мбайт / 55 Кбайт	15 Мбайт / 110 Кбайт	112 Мбайт / 0,6 Мбайт	293 Мбайт / 1,2 Мбайт	5,4 Гбайт / 11 Мбайт	49 Гбайт / 69 Мбайт
	число итераций	15	15	15	75	75	100	100

Приложение Б

Результаты замеров производительности вычислительных процедур

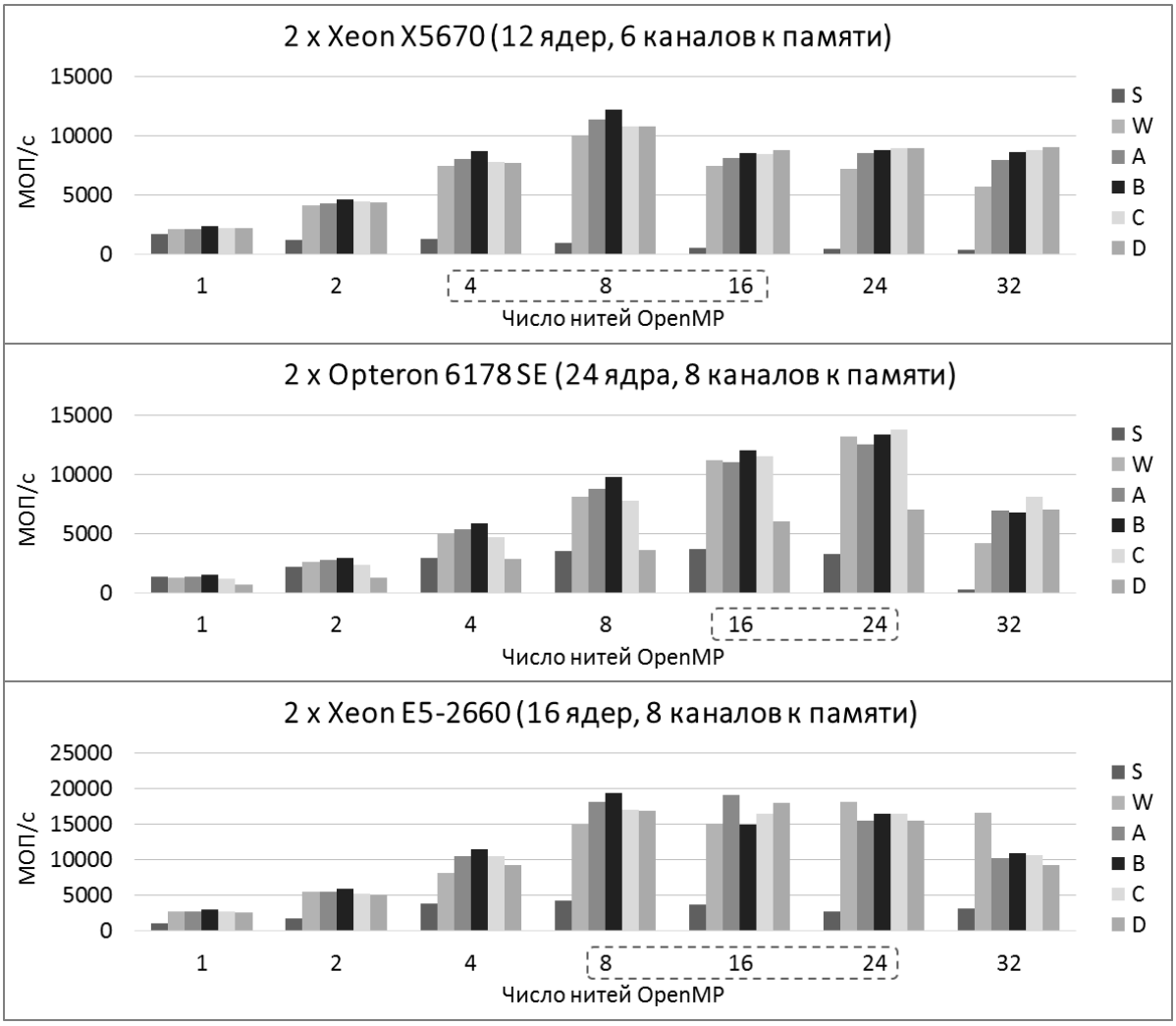

Рисунок Б.1 — MG: замеры для референсных кодов

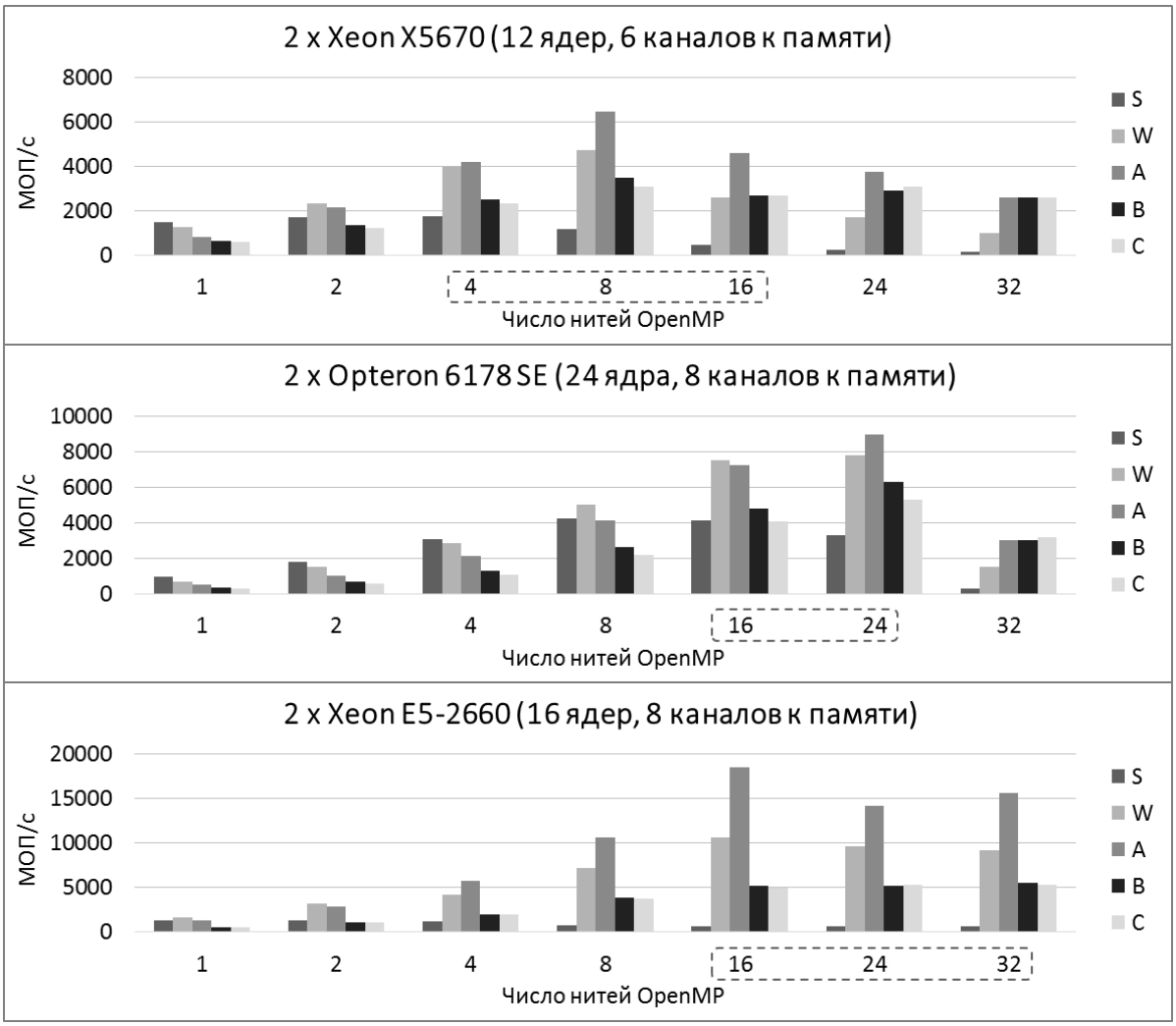

Рисунок Б.2 — CG: замеры для референсных кодов

Таблица Б.1 — Ядро процедуры SpMV на OpenCL: производительность на матрицах NPB CG

Класс	**GeForce TITAN**		**FirePro w9100**	
	Производит-ть	(H, r)	**Производит-ть**	(H, r)
S	6400	(32, 16)	8900	(4, 16)
W	12900	(16, 8)	19400	(4, 16)
A	15000	(8, 8)	22600	(4, 16)
B	18400	(8, 8)	19900	(16, 4)
C	16500	(128, 1)	12600	(64, 1)

Таблица Б.2 — Ядра короткого БПФ на BM7 и BM9

N	**всего тактов**	**cbutterfly**	**%**	**BM7** $Perf_{kern}$	**BM9** $Perf_{kern}$
64	247	192	77	1554	7773
128	579	448	77	1547	7737
256	1218	1024	84	1681	8407

Таблица Б.3 — Скорости пересылок по DMA (Мбайт/с)

Длина	**Шаг**	**DMA_GET**			**DMA_PUT**		
		BM7	**BM9**	**Прирост**	**BM7**	**BM9**	**Прирост**
128	128	2203	6867	x3,12	2361	6604	x2,80
128	256	2191	6655	x3,04	2352	6241	x2,65
64	256	2156	6661	x3,09	2348	5883	x2,51
32	256	2089	6129	x2,93	2343	4873	x2,08

Таблица Б.4 — Производительность свёртки на ВМ7 и ВМ9

	ВМ7		**ВМ9**		
L = N	**МОП/с**	**Эфф-ть, %**	**МОП/с**	**Эфф-ть, %**	**Прирост**
32	2412	39	8506	27	x3,5
64	3786	61	13193	43	x3,5
128	4855	72	17956	53	x3,7
256	5145	75	22201	64	x4,3
512	5447	77	25028	71	x4,6
1024	5348	80	26929	81	x5,0
2048	3620	86	16052	86	x4,4
4096	3016	54	12188	64	x4,0
8192	3193	68	12164	60	x3,8
16384	3547	70	12683	58	x3,6
32768	3831	71	12848	55	x3,4

Таблица Б.5 — Производительность БПФ на различных процессорах

	BM7 CP2	**BM9 CP2**	**TMS320 C6678**	**Tiger SHARC**	**Xeon E5-2670v1**		**2x STI Cell**
Частота CPU	200	1000	1250	600	2600	2600	3200
Ядер	1	1	8	2	1	16	16
Память	DDR2 200 МГц	DDR3 800 МГц	DDR3 800 МГц	DRAM 600 МГц	DDR3 800 МГц	DDR3 800 МГц	XDRAM 800 МГц
Каналов	1	1	2	4	1	8	4
N	**Производительность, МОП/с**						
64	3083	14347	-	-	9979	195199	-
128	3868	16147	-	-	7092	129584	-
256	4745	18639	-	3131	7829	64164	-
512	5457	21836	-	3227	8972	58695	-
1024	5946	24587	74881	3261	9358	75824	12200
2048	6041	27208	-	3265	8449	79782	18900
4096	5198	30141	60000	2109	10504	88756	25900
8192	3108	10724	-	2164	8766	86513	28900
16384	3207	12124	23457	2196	7323	79657	33600
32768	3494	12347	33108	2225	7281	68991	-
65536	3768	12910	44165	2251	7325	53758	-

Таблица Б.6 — Ядра разностных операторов на BM7 и BM9

Ядро	$N_1 \times N_2 \times N_3$	**Арифм. операций**	**Тактов**	**BM7** $Perf_{kern}$	**BM9** $Perf_{kern}$	**Эфф., %**
`resid`	$32 \times 8 \times 8$	$31N_1N_2N_3$	28394	447	2236	56
`proj`	$32 \times 8 \times 8$	$30N_1N_2N_3/8$	5176	297	1484	37
`interp`	$32 \times 4 \times 4$	$35N_1N_2N_3$	6287	570	2850	71
`interp0`	$32 \times 4 \times 4$	$27N_1N_2N_3$	5733	482	2411	60
`smooth`	$32 \times 8 \times 8$	$31N_1N_2N_3$	28394	447	2236	56
`smooth0`	$32 \times 8 \times 8$	$30N_1N_2N_3$	28392	433	2164	54

Таблица Б.7 — Производительность MG на различных процессорах

	BM7 CPU	**BM9 CPU**	**BM7 CP2**	**BM9 CP2**	**Xeon E5-2670v1**		**Эльбрус-4С**	
Частота CPU	200	1200	200	1200	2600	2600	800	800
Ядер	1	1	1	1	1	16	1	16
Память	DDR2 200 МГц	DDR3 800 МГц	DDR2 200 МГц	DDR3 800 МГц	DDR3 800 МГц	DDR3 800 МГц	DDR3 800 МГц	DDR3 800 МГц
Каналов	1	1	1	1	1	8	1	12
Класс	**Производительность, МОП/с**							
S	17	371	277	850	4897	3767	-	-
W	17	383	648	2760	5828	53282	-	-
A	17	355	680	3067	5739	41088	-	-
B	18	383	692	3091	6717	44587	3326	32080

Таблица Б.8 — Ядро SpMV на ВМ7 и ВМ9

Матрица	NZ	H	**всего тактов**	**psmadd**	**ВМ7** $Perf_{kern}$	**ВМ9** $Perf_{kern}$	**Эфф., %**
S	78148	64	11552244	415360	28,8	143,8	32
32×32	64	32	849	26	24,5	122,5	28
64×64	256	64	2421	82	27,1	135,5	30
128×128	1024	128	7688	269	28,0	140,0	31
256×256	4096	256	25755	915	28,4	142,1	32
512×512	16384	256	172311	6170	28,6	143,2	32
1024×1024	65536	64	4690377	168499	28,7	143,7	32
2048×2048	262144	32	67430322	2426163	28,8	143,9	32
4096×2048	500000	32	259405792	9340723	28,8	144,0	32

Таблица Б.9 — Избыточность формата упаковки матрицы для BM7/9

Матрица	**Число ненул. элементов (NZ)**	**Длина массива значений**	**Макс. эфф-ть, %**
S	78148	151040	52
32×32	64	256	25
64×64	256	768	33
128×128	1024	2048	50
256×256	4096	8192	50
512×512	16384	24576	67
1024×1024	65536	84736	77
2048×2048	262144	302976	86
4096×2048	500000	582144	86

Таблица Б.10 — Производительность SpMV на различных процессорах

	BM7 CPU	**BM7 CP2**	**BM9 CPU**	**BM9 CP2**	**Xeon E5-2670v1**		
Частота CPU	200	200	1000	1000	2600	2600	2600
Ядер	1	1	1	1	1	8	16
Память	DDR2 200 МГц	DDR2 200 МГц	DDR3 700 МГц	DDR3 700 МГц	DDR3 800 МГц	DDR3 800 МГц	DDR3 800 МГц
Каналов	1	1	1	1	1	8	12
Матрица	**Производительность, МОП/с**						
S (1400 × 1400)	11	51	209	248	1663	393	355
32 × 32	8	4	112	18	114	36	5
64 × 64	10	11	159	50	160	67	15
128 × 128	11	24	223	110	1255	235	65
256 × 256	9	38	243	180	1221	629	56
512 × 512	10	54	270	267	1428	586	214
1024 × 1024	10	67	288	336	447	1056	990
2048 × 2048	10	76	245	384	722	1994	2210
4096 × 2048	19	75	479	384	1461	3787	4544