ВЕСТНИК

ЮЖНО-УРАЛЬСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2015
Т. 4, № 3

ISSN 2305-9052

СЕРИЯ

«ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И ИНФОРМАТИКА»

Решением ВАК включен в Перечень научных изданий, в которых должны быть опубликованы результаты диссертаций на соискание ученых степеней кандидата и доктора наук

Учредитель — Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Южно-Уральский государственный университет» (национальный исследовательский университет)

Тематика журнала:

- Вычислительная математика и численные методы
- Математическое программирование
- Распознавание образов
- Вычислительные методы линейной алгебры
- Решение обратных и некорректно поставленных задач
- Доказательные вычисления
- Численное решение дифференциальных и интегральных уравнений
- Исследование операций
- Теория игр
- Теория аппроксимации
- Информатика
- Математическое и программное обеспечение высокопроизводительных вычислительных систем
- Системное программирование
- Перспективные многопроцессорные архитектуры
- Облачные вычисления
- Технология программирования
- Машинная графика
- Интернет-технологии
- Системы электронного обучения
- Технологии обработки баз данных и знаний
- Интеллектуальный анализ данных

Редакционная коллегия
С.М. Абдуллаев, д.г.н., проф.
А.В. Мовчан, *техн. секретарь*
А.В. Панюков, д.ф.-м.н., проф.
Л.Б. Соколинский, д.ф.-м.н., проф., *отв. редактор*
В.П. Танана, д.ф.-м.н., проф., *зам. отв. редактора*
М.Л. Цымблер, к.ф.-м.н., доц., *отв. секретарь*

Редакционный совет
А. Андреяк, PhD, профессор (Германия)
В.И. Бердышев, д.ф.-м.н., акад. РАН, *председатель*
В.В. Воеводин, д.ф.-м.н., чл.-кор. РАН
Дж. Донгарра, PhD, профессор (США)
С.В. Зыкин, д.т.н., профессор
Д. Маллманн, PhD, профессор (Германия)
А.Н. Томилин, д.ф.-м.н., профессор
В.Е. Третьяков, д.ф.-м.н., чл.-кор. РАН
В.И. Ухоботов, д.ф.-м.н., профессор
В.Н. Ушаков, д.ф.-м.н., чл.-кор. РАН
М.Ю. Хачай, д.ф.-м.н., профессор
П. Шумяцки, PhD, профессор (Бразилия)
Е. Ямазаки, PhD, профессор (Бразилия)

BULLETIN
OF THE SOUTH URAL STATE UNIVERSITY
SERIES "COMPUTATIONAL MATHEMATICS AND SOFTWARE ENGINEERING"

2015
Vol. 4, no. 3

ISSN 2305-9052

Vestnik Yuzhno-Ural'skogo Gosudarstvennogo Universiteta.
Seriya "Vychislitel'naya Matematika i Informatika"

South Ural State University

The scope of the journal:

- Numerical analysis and methods
- Mathematical optimization
- Pattern recognition
- Numerical methods of linear algebra
- Reverse and ill-posed problems solution
- Computer-assisted proofs
- Numerical solutions of differential and integral equations
- Operations research
- Game theory
- Approximation theory
- Computer science
- High performance computing
- System software
- Advanced multiprocessor architectures
- Cloud computing
- Software engineering
- Computer graphics
- Internet technologies
- E-learning
- Database processing
- Data mining

Editorial Board
S.M. Abdullaev, South Ural State University (Chelyabinsk, Russia)
A.V. Movchan, South Ural State University (Chelyabinsk, Russia)
A.V. Panyukov, South Ural State University (Chelyabinsk, Russia)
L.B. Sokolinsky, South Ural State University (Chelyabinsk, Russia)
V.P. Tanana, South Ural State University (Chelyabinsk, Russia)
M.L. Zymbler, South Ural State University (Chelyabinsk, Russia)

Editorial Counsil
A. Andrzejak, Heidelberg University (Germany)
V.I. Berdyshev, Institute of Mathematics and Mechanics, Ural Branch of the RAS (Yekaterinburg, Russia)
J. Dongarra, University of Tennessee (USA)
M.Yu. Khachay, Institute of Mathematics and Mechanics, Ural Branch of the RAS (Yekaterinburg, Russia)
D. Mallmann, Julich Supercomputing Centre (Germany)
P. Shumyatsky, University of Brasilia (Brazil)
A.N. Tomilin, Institute for System Programming of the RAS (Moscow, Russia)
V.E. Tretyakov, Ural Federal University (Yekaterinburg, Russia)
V.I. Ukhobotov, Chelyabinsk State University (Chelyabinsk, Russia)
V.N. Ushakov, Institute of Mathematics and Mechanics, Ural Branch of the RAS (Yekaterinburg, Russia)
V.V. Voevodin, Lomonosov Moscow State University (Moscow, Russia)
Y. Yamazaki, Federal University of Pelotas (Brazil)
S.V. Zykin, Sobolev Institute of Mathematics, Siberian Branch of the RAS (Omsk, Russia)

Содержание

© Издательский центр ЮУрГУ, 2015

Contents

УДК 004.052.3 DOI: 10.14529/cmse150301

МОДЕЛИРОВАНИЕ ОТКАЗОВ В ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ В РАМКАХ СТАНДАРТА MPI И ЕГО РАСШИРЕНИЯ ULFM[1]

А.А. Бондаренко, М.В. Якобовский

Рассматривается проблема выполнения длительных расчетов на высокопроизводительных вычислительных системах, компоненты которых подвержены отказам. Для программ, запускаемых на подобных системах, существенным является возможность обработки отказов путем автоматического продолжения расчета на оставшихся работоспособных узлах системы. Возможность обработки отказов предусматривается в разрабатываемом стандарте MPI 3.1. В работе кратко описывается библиотека моделирования отказов для тестирования отказоустойчивых алгоритмов, использующих функционал разрабатываемого стандарта MPI 3.1. Описана техника отказоустойчивости на примере тестовой задачи. Проведено сравнение записи контрольных точек в оперативную память и в распределенную файловую систему.

Ключевые слова: параллельные вычисления, отказоустойчивость, контрольные точки, MPI, ULFM, моделирование отказов.

Введение

Несмотря на прогресс, достигнутый в последние годы, «проблема устойчивости для Экзафлопсных систем не решена, а перед сообществом по-прежнему стоит сложная задача выдачи корректных результатов приложениями, работающими на неустойчивых системах» [1]. Одновременное повышение надежности и производительности высокопроизводительной системы, как единого целого, на аппаратном уровне является трудноразрешимой задачей. В связи с этим, для организации вычислений на современных суперкомпьютерах необходимо развитие новых отказоустойчивых технологий, позволяющих с помощью программных решений корректно продолжать вычисления даже при отказе части оборудования.

Программное обеспечение отказоустойчивости осуществляется на системном уровне или на уровне пользователя. Отказоустойчивость на уровне системы основана на записи всей памяти приложения в глобальную контрольную точку и последующем перезапуске программы из этой контрольной точки в случае отказа. Операции сохранения и перезапуска могут быть выполнены автоматически, поскольку не зависят от специфики прикладной программы. Преимуществом автоматических операций сохранения и перезапуска является простота использования, так как от пользователя не требуется внесения изменений в код программы, что достигается за счет очевидного недостатка — высоких накладных расходов. В контрольную точку записывается все данные, используемые приложением, в том числе данные, которые не являются существенными для запуска программы из контрольной точки.

[1] Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии – 2015».

В случае реализации отказоустойчивости на уровне пользователя, в контрольную точку входит только то, что явно укажет прикладной программист. В идеале, только те данные, которые необходимы для восстановления утерянной в результате сбоя информации. Так же на уровне пользователя контролируется частота и момент создания контрольных точек при выполнении программы. То есть накладные расходы могут быть значительно уменьшены, однако, потребуется дополнительная работа прикладного программиста для реализации отказоустойчивости в приложении.
Использование методов организации отказоустойчивости вычислений на системном уровне представляется трудно реализуемым для систем Экзафлопсного уровня, поэтому значительное внимание специалистов в этой области уделяется именно методам уровня пользователя [1, 2]. Чтобы применять методы организации отказоустойчивости вычислений на уровне пользователя, средства обмена сообщениями должны обеспечивать возможность коммуникации между вычислительными процессами даже при наличии отказов в вычислительной системе, а также обеспечивать способность восстановления приложения в согласованное состояние, из которого вычисления могут быть продолжены.

Для удовлетворения этих требований ключевым для стандарта MPI является условие [2], согласно которому любая операция должна завершиться за конечное время, даже в случае отказа части оборудования. В противном случае, если некоторая MPI-операция никогда не завершается, то MPI-процесс, вызвавший ее, не может принять участия в восстановлении приложения, тем самым организация отказоустойчивых вычислений становится невозможной. Все обеспечивающие связь между процессами MPI-операции должны возвращать коды ошибок, информирующие приложение о любом состоянии, возникшем в ходе выполнения операции. После того как некоторые процессы были уведомлены об отказе в системе работа по восстановлению может быть начата. Для предотвращения блокировок в MPI должны быть реализованы механизмы предупреждения всех процессов об отказах, когда это необходимо.

Перечисленные выше требования учитываются в расширении ULFM [2, 3] и дорабатываются для внесения в стандарт MPI 3.1. Программные реализации MPI с расширением ULFM позволят реализовывать различные техники отказоустойчивости на уровне пользователя. Следует отметить, что сроки появления пригодных к эксплуатации реализаций MPI с расширением ULFM в настоящее время не определены, а отладка и тестирование отказоустойчивых алгоритмов на реальных вычислительных системах сегодня неэффективны в силу, пока еще, относительно большой длительности времени безотказной работы. Поэтому актуальна задача проверки и тестирования приложений, основанных на техниках отказоустойчивости.

Цель данной работы состоит в разработке методов и средств моделирования отказов в высокопроизводительных системах согласно спецификации ULFM, обеспечивающих возможность изучения и тестирования различных стратегий построения отказоустойчивых параллельных приложений.

Статья организована следующим образом. В первом раздел статьи приведены принципы и основные функции спецификации ULFM, а также кратко описывается содержание библиотеки моделирования отказов. Во втором разделе описаны техники отказоустойчивости для решения тестовой задачи и их программные реализации. В заключении приводится описание результатов работы и направление дальнейших исследований.

1. Моделирование отказов с помощью расширения ULFM

В стандарте MPI 3.0 отсутствует описание каких-либо функций связанных с обеспечением отказоустойчивости в вычислительных системах. Желая предоставить пользователю гибкий интерфейс, авторы ULFM ушли от выбора конкретной техники отказоустойчивости или обязательств восстанавливать приложение целиком после отказа. «Этот интерфейс должен информировать приложение о специальных условиях мешающих передаче сообщений, и предоставлять конструкции и определения, которые позволят приложению восстановить MPI–объекты и способность передавать сообщения» [2, стр. 246]. Выбор наиболее подходящей для данного приложения техники восстановления и реализация остается за пользователем.

Обработка отказов в ULFM основывается на следующих двух положениях [2, 3]. В случае отказа хотя бы одного из MPI-процессов:

- MPI-операция, включающая в себя отказавший MPI-процесс, не должна блокироваться бесконечно, а должна быть либо выполнена успешно, либо вызвать MPI-исключение.
- MPI-операция, не включающая в себя отказавший MPI-процесс, должна завершиться нормально, если только не будет прервана пользователем с помощью специальных функций ULFM.

MPI-исключение возникает только у участников операции, содержащей отказавший процесс, однако может так быть, что не все MPI-процессы, участвующие в этой операции, вернут MPI-исключение. Может возникнуть ситуация, когда MPI-процесс завершит свое участие в этой операции, а отказ произойдет в следующий момент времени. Еще раз подчеркнем, что MPI-исключение отражает только локальное воздействие отказа на операцию, и нет никаких гарантий, что другие процессы, не участвующие в данной операции, также будут уведомлены о возникновении этого отказа. Асинхронное распространение информации о возникновении отказа не гарантируется, и пользователи должны проявлять осторожность при выявлении множества процессов, для которых будет зафиксирован отказ и возникнет MPI-исключение.

Обработка исключений и реализация различных техник восстановления осуществляется, в основном за счет функций [3]:

- MPI_COMM_REVOKE — прекращает все текущие нелокальные операции на коммуникаторе и отмечает коммуникатор в качестве аннулированного. Все последующие вызовы нелокальных функций, связанные с этим коммуникатором, должны завершаться значением MPI_ERR_REVOKED, за исключением функций MPI_COMM_SHRINK и MPI_COMM_AGREE;
- MPI_COMM_SHRINK — создает на основе существующего коммуникатора новый, не содержащий отказавшие процессы;
- MPI_COMM_FAILURE_GET_ACKED — возвращает группу, состоящую из процессов, которые были определены как отказавшие к моменту последнего вызова функции MPI_COMM_FAILURE_ACK;
- MPI_COMM_AGREE — согласовывает значение булевой переменной, если нет отказавших MPI-процессов в коммуникаторе или возвращает исключение о наличии отказа всем не отказавшим процессам в коммуникаторе.

Для тестирования различных техник отказоустойчивости была создана библиотека моделирования отказов, основанная на расширении ULFM для стандарта MPI. Исполь-

зование спецификации ULFM вызвано естественным стремлением сохранить будущую совместимость с разрабатываемым стандартом MPI 3.1. В данной библиотеке каждому MPI-процессу поставлен в соответствие атрибут, значение которого отражает состояние этого процесса: отказал он или нет. Также в ней реализованы модификации MPI функций обмена. Функции обмена данной библиотеки возвращают исключения согласно спецификации [3] расширения ULFM, не предусмотренные стандартом MPI 3.0, позволяющие моделировать отказ в нормально функционирующей распределенной вычислительной системе. Для реализации техник отказоустойчивости с помощью функций стандарта MPI 3.0 в этой библиотеке реализованы функции MPI_COMM_REVOKE, MPI_COMM_SHRINK, MPI_COMM_AGREE, MPI_COMM_FAILURE_ACK, MPI_COMM_FAILURE_GET_ACKED.

2. Два примера реализации техники отказоустойчивости

В данной работе были разработаны три программы. В первой программе реализован параллельный алгоритм решения задачи о распространении тепла в однородном стержне. Остальные программы используют библиотеку моделирования отказов и содержат процедуры автоматического восстановления вычислений, при этом во второй программе сохранение контрольных точек осуществляется в оперативную память вычислительных узлов, а в третье программе запись контрольных точек осуществляется в распределенную файловую систему.

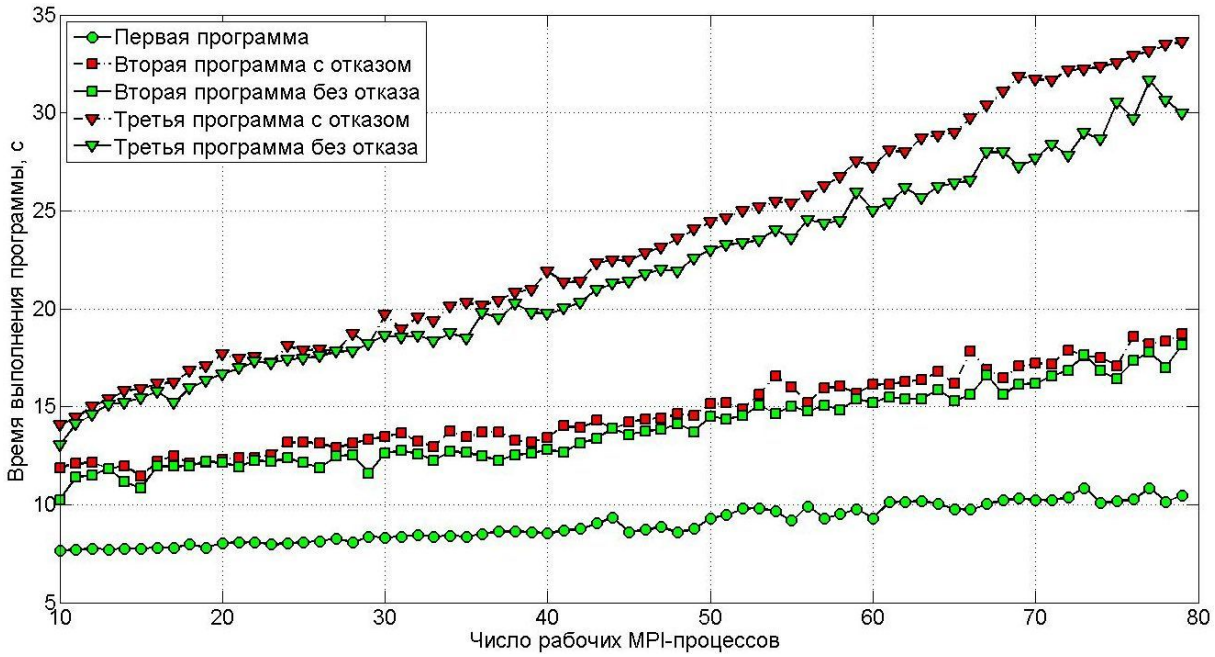
Опишем используемую технику обеспечения отказоустойчивости. Для реализации автоматического восстановления вычислений все множество MPI-процессов разделено на рабочие процессы, формирующие рабочее поле, в котором выполняется основной алгоритм прикладной программы, и на резервные процессы, которые простаивают в ожидании вызова обработчика отказа в системе. В случае возникновения отказа, MPI-процессы, отмеченные как отказавшие, выводятся из расчетного поля, а их место занимают новые из списка резервных процессов. Здесь следует отметить, что разделение на рабочие и резервные процессы происходит внутри самой программы, с помощью стандартных средств MPI. Количество резервных MPI-процессов является параметром для программы, и должно определяться пользователем исходя из предположений о возможном количестве отказов во время выполнения программы. Такое разделение на рабочие и резервные MPI-процессы, связано с тем, что для большинства алгоритмов, описывающих решение задач математической физики, затратными являются процедуры перераспределения работы на меньшее число MPI-процессов, по сравнению с начальными условиями запуска программы.

Создание контрольных точек происходит по координированному протоколу, то есть в определенный момент работы программы, когда отсутствуют обмены между MPI-процессами, осуществляется запись контрольных точек для всех рабочих MPI-процессов. В случае сохранения контрольных точек в локальные устройства хранения, в том числе оперативную память, для MPI-процессов, введенных в рабочее поле, может быть недоступна информация о контрольных точках отказавших MPI-процессов. Для того чтобы устранить этот недостаток необходимо осуществлять дублирование при сохранении контрольных точек в локальные устройства хранения. Во второй программе сохранение контрольных точек в оперативную память с дублированием осуществляется по схеме сохранения описанной в работе [4]. При сохранении контрольных точек в рас-

пределенную файловую систему все контрольные точки непосредственно доступны MPI-процессам, введенным в расчетное поле в случае отказа. Таким образом, для третьей программы дублирование не требуется.

Из вышесказанного следует, что процедуры восстановления для второй и третьей программы отличаются. Для второй программы необходимо: осуществить восстановление MPI-среды; осуществить восстановление рабочего поля; определить глобальную контрольную точку, с которой необходимо продолжить вычисления; осуществить копирование необходимых локальных контрольных точек новым MPI-процессам, введенным в расчетное поле; затем осуществить восстановление работы прикладной программы с локальных контрольных точек. Для третьей программы необходимо: осуществить восстановление MPI-среды; осуществить восстановление рабочего поля; затем перейти к процедуре восстановление процесса вычислений с последней глобальной контрольной точки, записанной в распределенной файловой системе.

Вторая и третья программы запускались в двух режимах: без отказов во время вычислений и с одним отказом двух MPI-процессов в одном и том же месте основного алгоритма. В обоих режимах работы осуществлялось сохранение контрольных точек через одинаковое число итераций основного цикла. Подобный эксперимент позволяет оценить объем накладных расходов для организации отказоустойчивости, а именно временные затраты на сохранение контрольных точек и на восстановление системы после отказа.



Сравнение программ, реализующие разные техники отказоустойчивости

Параметры задачи были выбраны таким образом, чтобы размер контрольной точки был порядка 2 Мб. Для второй и третьей программ разбиение на рабочие и запасные MPI-процессы было проведено так, чтобы было два запасных MPI-процесса. Вычисления проводились на научном кластере ИПМ [5]. Уже на небольшом числе MPI-процессов (до 80), накладные расходы на организацию отказоустойчивости, при сохранение контрольных точек в распределенную файловую систему, превышают соответствующие накладные расходы, при сохранении контрольных точек в оперативную па-

мять с дублированием. Теоретические оценки показывают, что если запустить вторую или третью программу на всем суперкомпьютере Sequoia и взять локальные контрольные точки объемом равным 100 MB каждая, тогда для координированного протокола запись глобальной контрольной точки в распределенную файловую систему будет составлять около 3 минут, а сохранение в оперативную память с дублированием (параметры схемы сохранения SD = 2, DF = 3 [4]), — порядка 2,5 секунд. Отметим, что при больших объемах локальных контрольных точек сохранение в оперативную память вычислительных узлов по описанной в [4] схеме может и не иметь смысла в силу ограничения объема оперативной памяти. А при наличии более быстрых коммуникационных соединений для распределенной файловой системы может существовать объем контрольных точек, при котором будут равны накладные расходы для обеспечения отказоустойчивости второй и третьей программы.

Заключение

Для тестирования и определения эффективности разрабатываемых техник отказоустойчивости необходимо создание соответствующих программных средств. В данной работе представлена библиотека функций моделирования отказов в нормально функционирующих системах, основанная на существующей реализации стандарта MPI 3.0. Она позволяет тестировать различные алгоритмы восстановления вычислений, в том числе, с использованием локальных устройств хранения. В частности, для тестовой задачи на вычислительной системе [5] показано, что при организации автоматического восстановления вычислений сохранение в оперативную память предпочтительнее, чем в распределенную файловую систему. При этом временные затраты на сохранение контрольных точек в оперативную память и на восстановление системы после отказа относительно малы.

Основным направлением дальнейших исследований является разработка и изучение эффективных техник обеспечения отказоустойчивости для решения мультимасштабных задач механики сплошной среды на вычислительных системах сверхвысокой производительности.

Работа выполнена при поддержке Российского фонда фундаментальных исследований по гранту 13-01-12073 офи_м.

Литература

1. Cappello, F. Toward Exascale Resilience: 2014 update / F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, M. Snir // Supercomputing frontiers and innovations. — 2014. — Vol. 1, No. 1. — P. 1–28. DOI: 10.14529/jsfi140101.
2. Bland, W. Post-failure recovery of MPI communication capability: Design and rationale / W. Bland, A. Bouteiller, T. Hérault, G. Bosilca, J. Dongarra // International Journal of High Performance Computing Applications. — 2013. — Vol. 27, No. 3. — P. 244–254. DOI: 10.1177/1094342013488238.
3. ICL Fault Tolerance URL: http://fault-tolerance.org/ulfm/ulfm-specification (дата обращения: 01.03.2015).
4. Бондаренко, А.А. Обеспечение отказоустойчивости высокопроизводительных вычислений с помощью локальных контрольных точек / А.А. Бондаренко,

М.В. Якобовский // Вестник Южно-Уральского государственного университета. Серия «Вычислительная математика и информатика». — 2014. — Том. 3, No. 3. — С. 20–36. DOI: 10.14529/cmse140302.

5. Scientific Cluster of Keldysh Institute of Applied Mathematics RAS. URL: http://imm6.keldysh.ru/~informer/(дата обращения: 01.03.2015).

Бондаренко Алексей Алексеевич, к.ф.-м.н., научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация), bondaleksey@gmail.com.

Якобовский Михаил Владимирович, д.ф.-м.н., заведующий сектором «Программное обеспечение вычислительных систем и сетей», Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация), lira@imamod.ru

Поступила в редакцию 13 апреля 2015 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2015, vol. 4, no. 3, pp. 5–12

DOI: 10.14529/cmse150301

SIMULATION OF FAILURES IN HIGH-PERFORMANCE COMPUTING SYSTEMS UNDER MPI-ULFM

A.A. Bondarenko, Keldysh Institute of Applied Mathematics (Moscow, Russian Federation) bondaleksey@gmail.com,

M.V. Iakobovski, Keldysh Institute of Applied Mathematics (Moscow, Russian Federation) lira@imamod.ru

In this paper, we consider one of the main problems that occur in the area of high-performance computing is to continue computations despite of failures. For the programs running on such systems it is very important to handle failures and continue computations on working nodes. One of the MPI 3.1 standardization efforts aim is adding new techniques, approaches, or concepts to support for fault tolerance in MPI applications. The paper briefly describes a library for simulation of failures and testing fault-tolerant algorithms using functional of developing MPI 3.1 standard. In the test problem we describe one of the techniques of fault tolerance and we compare checkpoint in operational memory versus checkpoint in the distributed file system.

Keywords: parallel computing, fault tolerance, checkpoint, simulation of failures, MPI, ULFM.

References

1. Cappello F., Geist A., Gropp W., Kale S., Kramer B., Snir M. Toward Exascale Resilience: 2014 update. // Supercomputing frontiers and innovations. 2014. Vol. 1, No. 1. P. 1–28. DOI: 10.14529/jsfi140101.
2. Bland W., Bouteiller A., Hérault T., Bosilca G., Dongarra J. Post-failure recovery of MPI communication capability: Design and rationale // International Journal of High

Performance Computing Applications. 2013. Vol. 27, No. 3. P. 244–254. DOI: 10.1177/1094342013488238.

3. ICL Fault Tolerance URL: http://fault-tolerance.org/ulfm/ulfm-specification (accessed: 01.03.2015).
4. Bondarenko A.A., Yakobovskiy M.V. Obespechenie otkazoustoychivosti vysokoproizvoditel'nykh vychisleniy s pomoshch'yu lokal'nykh kontrol'nykh tochek [Fault Tolerance for HPC by Using Local Checkpoints]. Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta. Seriya «Vychislitel'naya matematika i informatika» [Bulletin of South Ural State University. Series: Computational Mathematics and Software Engineering]. 2014. Vol. 3, No. 3. P. 20–36. DOI: 10.14529/cmse140302.
5. Scientific Cluster of Keldysh Institute of Applied Mathematics RAS. URL: http://imm6.keldysh.ru/~informer/ (accessed: 01.03.2015).

Received April 13, 2015.

УДК 004.4'23, 004.4'24 DOI: 10.14529/cmse150302

АВТОМАТИЗИРОВАННОЕ ПРЕОБРАЗОВАНИЕ ФОРТРАН-ПРОГРАММ, НЕОБХОДИМОЕ ДЛЯ ИХ ЭФФЕКТИВНОГО РАСПАРАЛЛЕЛИВАНИЯ С ПОМОЩЬЮ СИСТЕМЫ САПФОР[1]

Н.А. Катаев, А.А. Буланов

Автоматическое отображение последовательных программ на вычислительные системы с распределенной памятью может потребовать предварительного преобразования программ, ориентированного на данный класс систем. Использование системы САПФОР для распараллеливания прикладных программ позволило выделить преобразования, выполнение которых может быть автоматизировано. В статье представлены преобразования, повышающие возможность эффективного распараллеливания программ за счет устранения причин, препятствующих распараллеливанию циклов. Выполнение данных преобразований позволило автоматизировать получение последовательной реализации, эффективно отображаемой на современные кластеры автоматически распараллеливающим компилятором системы, для задачи гидродинамики.

Ключевые слова: распараллеливание, автоматизация, преобразование последовательных программ, Фортран.

Введение

Основной целью распараллеливания прикладной программы является ее эффективное выполнение на современных вычислительных системах. Отличительной чертой программирования для таких систем является совместное использование различных технологий параллельного программирования. Сложность применения данных технологий подчеркивает актуальность исследований в области автоматизации распараллеливания программ.

К автоматически распараллеливающим компиляторам можно отнести PLUTO [1], Par4all [2], Parallware [3], задать опции для автоматического распараллеливания можно при использовании компиляторов Intel. Основным языком программирования является C. Язык Fortran поддерживается в компиляторах Par4all (только Fortran 77) и Intel. Следует отметить, что компиляторы Intel и Parallware являются коммерческими. Распараллеливание ориентировано на системы с общей памятью, компилятор Par4all также обеспечивают распараллеливание для графических ускорителей. Считается, что автоматическое распараллеливание для систем с распределенной памятью пока не достижимо [4].

К автоматизированным системам распараллеливания относятся САПФОР [5], Intel Parallel Studio [6], Parawise [7], ДВОР [8]. Особенностью системы САПФОР является использование автоматически распараллеливающего компилятора – взаимодействие с пользователем осуществляется на этапе подготовки последовательной программы к автоматическому распараллеливанию. Пользователь, во-первых, предоставляет системе информацию о свойствах программы, которую не удалось получить автоматическими

[1] Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии – 2015».

средствами анализа (статическими и динамическими). Во-вторых, он участвует в преобразовании последовательной программы в последовательную программу с целью устранения проблем, выявленных системой и мешающих автоматическому распараллеливанию.

Входным языком системы САПФОР является Fortran, результат распараллеливания представляет собой программу на языке Fortran OpenMP, Fortran DVM/OpenMP или Fortran DVMH. Модель DVMH [9] является расширением модели DVM [10] и обеспечивает распараллеливание программы на кластер с ускорителями. Основными компонентами САПФОР являются анализаторы последовательных программ (статические и динамические), блоки преобразования последовательных программ в параллельные программы (эксперты), диалоговая оболочка для взаимодействия с пользователем, генератор кода, создающий на основе принятых экспертом решений параллельную версию программы.

Целью данной работы является автоматизация преобразований последовательной программы, выполняемых на этапе подготовки ее к автоматическому распараллеливанию экспертом системы САПФОР.

Статья организована следующим образом. В первом разделе описаны три подхода к преобразованию программ, используемых в системе САПФОР. Во втором разделе рассмотрены примеры преобразований последовательных программ. Указанные преобразования могут быть выполнены системой САПФОР автоматически с целью устранения проблем, препятствующих эффективному распараллеливанию программы. В третьем разделе описан итерационный процесс распараллеливания двумерной задачи «Каверна» о течении несжимаемой жидкости или слабосжимаемого газа около прямоугольной выемки. Каждая итерация включает в себя попытку автоматического построения вариантов распараллеливания программы и автоматическое применение преобразований, устраняющих в последовательной программе проблемы, мешающие ее эффективному распараллеливанию. В четвертом разделе кратко рассмотрена программная реализация нового компонента системы САПФОР – преобразователя. В заключении суммируются результаты проведенного исследования и рассматриваются направления дальнейших работ.

1. Подходы к преобразованию программ

Большинство компиляторов использует заранее определенные последовательности фаз для оптимизации и распараллеливания программ. Данные последовательности могут сильно различаться в зависимости от цели проводимых оптимизаций: по памяти, по времени выполнения на одном процессоре или на нескольких. При распараллеливании необходимо учитывать архитектуры вычислительных систем, на которых предполагается выполнение программы. При этом оптимальность таких последовательностей сильно зависит от прикладной программы, а поиск наилучшей последовательности для конкретной программы сильно затруднен из-за огромных размеров пространства возможных оптимизационных последовательностей [11].

Распараллеливание в системе САПФОР представляет собой итерационный процесс, это обеспечивает возможность выбора трансформаций для решения проблем, мешающих эффективному распараллеливанию последовательной программы экспертом системы

САПФОР, по мере их возникновения. Эксперт системы выявляет особенности программы, влияющие на качество ее распараллеливания (см. пример на рис. 1).

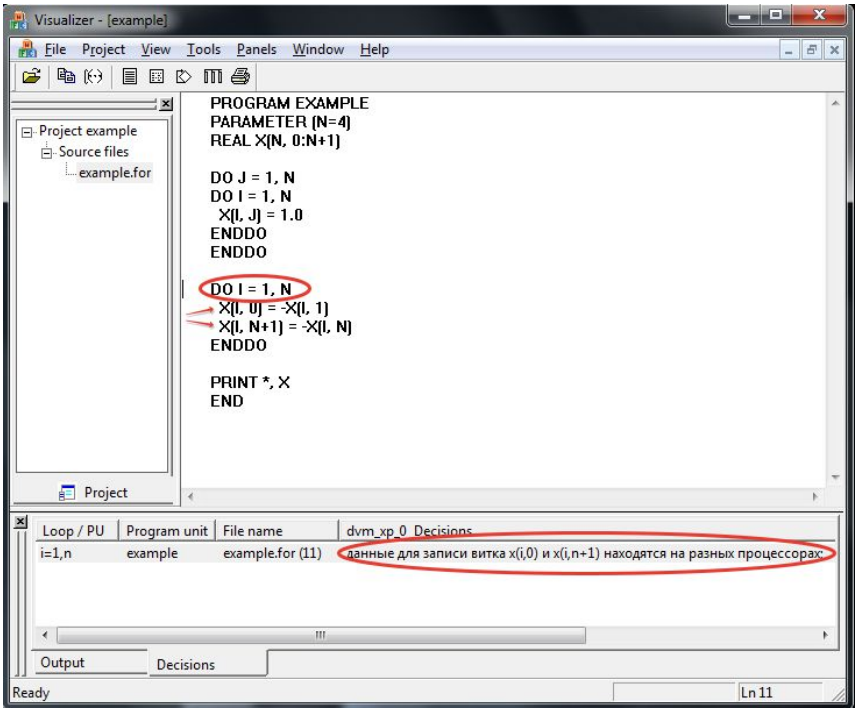


Рис. 1. Пример проблемы, мешающей эффективному отображению программы на параллельные компьютеры с распределенной памятью

Предлагается три подхода к преобразованию последовательной программы.

Первый подход состоит в автоматическом выборе способа устранения проблем. Решение о возможности устранения совокупности проблем принимается на основе удовлетворения каждой проблемы некоторому шаблону, описывающему ее решение. Процесс устранения проблем сводится к применению выбранных шаблонов. Каждая проблема рассматривается независимо от других, и в том случае если решения двух проблем конфликтуют между собой, необходимо участие пользователя для разрешения конфликта.

Второй подход состоит в активном вовлечении пользователя в процесс выбора решений выявленных проблем. Система предоставляет пользователю набор элементарных преобразований, для которых возможно автоматическое выполнение (например, разворачивание цикла, разделение цикла на несколько циклов, подстановку переменных и др.). Пользователь выбирает преобразование и указывает требующиеся характеристики, описывающие данное преобразование. Перед осуществлением преобразования система проверяет его допустимость. Допускается принудительное выполнения преобразования, если пользователь уверен в его допустимости, а требование консервативности средств анализа ограничивает его выполнение.

Если ни один из подходов не может быть применен, пользователь может вручную преобразовать исходный код программы, руководствуясь описаниями проблем, подготовленными экспертом системы САПФОР.

2. Автоматический выбор и применение шаблона

Использование системы САПФОР для распараллеливания прикладных программ [12–14]. позволило выделить проблемы, препятствующие эффективному распараллеливанию, для которых возможен автоматический поиск решения.

Одним из типов проблем является наличие в последовательной программе гнезд циклов с зависимостями по данным между различными итерациями циклов. В качестве шаблона решения проблем данного типа применяется разбиение гнезда циклов. Используемый алгоритм основан на алгоритме, предложенном в [15]. Чтобы цикл удовлетворял требованиям применимости алгоритма, системой САПФОР может выполняться автоматическое переупорядочивание операторов в цикле. В операторах, распределяемых по разным циклам, могут использоваться общие данные, вычисляемые внутри разделяемого цикла. В этом случае требующиеся вычисления либо будут продублированы в каждом из циклов, либо в программу будут добавлены вспомогательные массивы, в которых перед выполнением распределенных по циклам операторов будут размещены требующиеся данные.

Разбиение гнезда циклов также применяется для устранения проблемы, вызванной записью значений в разные элементы одного массива на одной итерации цикла (см. пример на рис. 1). Данная проблема характерна для отображения программ на системы с распределенной памятью. Перед выполнением вычислений данные должны быть распределены по процессорам таким образом, чтобы максимально снизить количество обменов. При параллельном выполнении итераций цикла процессорами системы необходимо, чтобы каждый процессор обладал необходимой ему частью распределенных данных. Для программы из примера на рис. 1 невозможно эффективно распределить данные и распараллелить выделенный цикл, несмотря на отсутствие в нем зависимостей по данным. На это указывает описание проблемы, полученное от эксперта. Результат успешного распараллеливания преобразованной программы приведен на рис. 2.

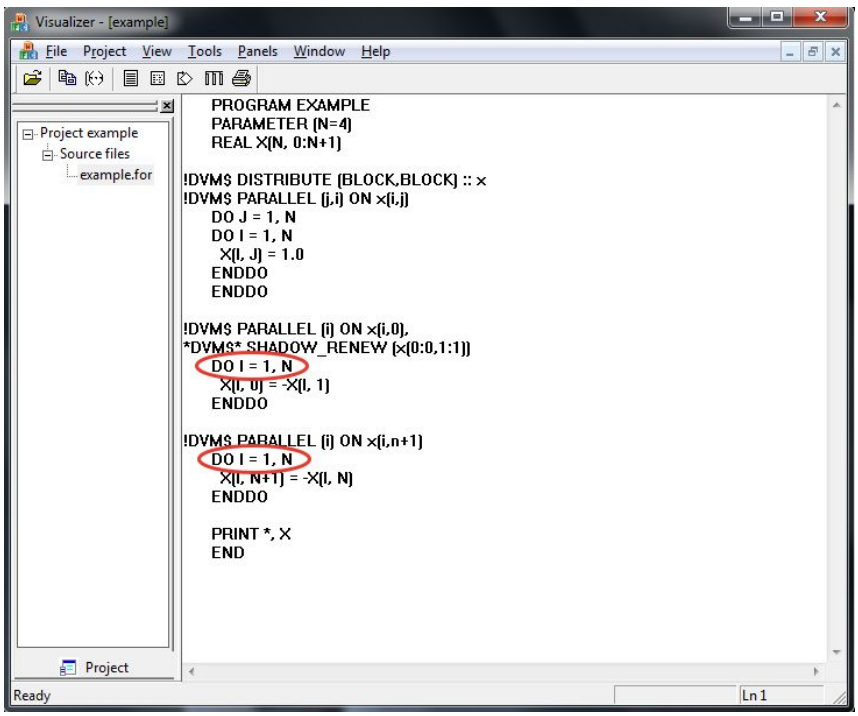


Рис. 2. Распараллеливание программы из примера на рис. 1 после разбиения цикла

Разные массивы, к элементам которых выполняется доступ на одной итерации цикла, должны быть соответствующим образом выравнены. Это накладывает ограничения на возможные варианты распределения массивов по процессорам вычислительной системы. В случае если эксперту системы не удается выполнить автоматическое распределение данных, чтобы увеличить количество возможных вариантов распределения, выполняется максимально возможное разбиение циклов программы. В один цикл попадают операторы, в которых выполняется запись в одни и те же элементы одного и того же массива. После распараллеливания программы, выполняется объединение циклов на основе информации о выравнивании массивов и распределении вычислений, полученной от эксперта.

3. Распараллеливание прикладной задачи

Предложенный подход к автоматизации получения последовательной программы, эффективно отображаемой на кластер, был протестирован на двумерной задаче Каверна о течении несжимаемой жидкости или слабосжимаемого газа около прямоугольной выемки [16]. Ручное преобразование данной задачи описано в [12].

Автоматическое преобразование потребовало проведения 6 итераций.

Запуск эксперта системы САПФОР на исходной не преобразованной программе выявил следующую проблему: «надо изменить оператор ввода-вывода на строке 520». В результате преобразования оператор вывода распределенного массива был заменен на оператор вывода скалярных переменных, содержащих предварительно скопированные элементы массива.

Следующий запуск эксперта выявил:

- четыре проблемы вида *«данные для записи витка ro(i,0) и ro(i, ny1) находятся на разных процессорах»*. В данном случае в указанных циклах не было зависимостей по данным, но на одной итерации выполнялась запись в разные элементы одного массива, указанные экспертом. Данная проблема аналогична проблеме, приведенной на рис. 1. При проведении преобразований было обнаружено, что в каждом из циклов аналогичным образом организована запись в пять разных массивов. Так как на момент преобразований не была доступна информация о выравнивании данных массивов, каждый цикл был разбит максимальным образом на десять циклов.
- пять проблем вида *«данные для записи витка ro1(:, j) находятся на разных процессорах»*. Данная проблема указывает, что на итерации цикла записывается целиком измерение массива и свидетельствует о том, что не был распараллелен один из циклов гнезда. Причиной этого является не тесная вложенность циклов. В результате преобразования во внутренний цикл был внесен инвариант, расположенный между заголовками циклов.

Следующий запуск эксперта выявил четыре проблемы вида «данные для записи витка ro1(i, j) и ro(i,j+1) находятся на разных процессорах». В указанных циклах были зависимости по данным, которые удалось устранить введением четырех временных массивов (по одному для каждого цикла) и разбиением каждого из циклов на два.

Следующий запуск эксперта выявил четыре проблемы вида «часть гнезда, данные для записи витка tmp1(i+1,j) и e1(i,j) находятся на разных процессорах». Данные проблемы указывали на циклы из предыдущего запуска. Причиной их возникновение стало

использование введенных временных массивов. Для решения данной проблемы необходимо разбить указанные циклы. Вынесение временного массива в отдельный цикл не решает данную проблему, так как в этом случае должен быть заведен новый временный массив с такой же структурой и операциями записи данных. Вынесению второго массива препятствовали зависимости по данным, которые удалось устранить введением еще четырех временных массивов (по одному для каждого цикла). Затем каждый из четырех циклов был разбит на два.

Следующий запуск эксперта выявил две проблемы: «часть гнезда, измерение 1 массива ro1 не распределено». Причиной возникновения данных проблем является вызов внутри цикла оператора вывода данных. Устранить данную проблему невозможно, но она не влияет на распараллеливание остальной части программы. В результате данного запуска программа была успешно распараллелена.

В результате разбиения циклов было добавлено 36+4+4=44 новых цикла. На следующем шаге количество циклов было уменьшено за счет объединения циклов, выполненного на основе информации о выравнивании массивов и распределении вычислений, полученной от эксперта. Количество добавленных циклов было сокращено до 8.

4. Программная реализация

Выбор подходящих шаблонов и выполнение определяемых ими преобразований реализован в виде отдельного компонента – преобразователя системы САПФОР. Преобразователь взаимодействует с другими компонентами и пользователем системы двумя способами.

Таблица

Пример специальных комментариев применяемых в системы САПФОР

Специальный комментарий	**Использование**
!PRG INDEPENDENT (<variable-name >)	Позволяет указать на отсутствие межитерационных зависимостей по данным в цикле
!PRG PRIVATE (<variable-name>) !PRG FIRST_PRIVATE (<variable-name>) !PRG LAST_PRIVATE (<variable-name>)	Позволяют указать приватные (приватные по входу или по выходу) переменные в цикле
!PRG FLOW|ANTI|OUTPUT <from> <to> [<dist>]	Позволяет указать наличие зависимости по данным между двумя операторами и расстояние указанной зависимости
!PRG REDUCTION (<variable-name> (<op>) [, <array-name>, <index-number>] <op> ::= SUM|PRODUCT|MAX|MIN|AND|OR|EQV|NEQV|MAXLOC|MINLOC	Позволяет указать редукционные переменные для цикла. Опциональный параметр <array-name> используется в случае редукции MAXLOC или MINLOC

Во-первых, через базу данных (внутреннее представление) системы САПФОР.

Во-вторых, через специальные комментарии с префиксом !PRG, добавляемые в исходный код программы. Данные комментарии позволяют задавать дополнительные свойства программы, которые не удалось выявить анализаторам системы САПФОР, или уточнить описание проблемы, возникшей при распараллеливании и выделенной экспер-

том системы САПФОР, для более эффективного подбора шаблона ее решения. Пример комментариев, задаваемых перед циклами программы, приведен в таблице.

Преобразователь написан на языке программирования Си++ с использованием библиотеки Sage++ [17] для выполнения преобразований исходного кода программы.

Заключение

Эффективное автоматическое распараллеливание последовательных программ, во многих случаях, невозможно без выполнения их преобразования. Выполнение в процессе распараллеливания заранее фиксированной последовательности фаз анализа и преобразований, не учитывает особенности каждой конкретной программы. Проведенное таким образом автоматическое распараллеливание часто оказывается неэффективным. Итерационный процесс распараллеливания программ, поддерживаемый системой автоматизированного распараллеливания САПФОР, позволяет применять только те преобразования, которые необходимы для устранения проблем, препятствующих распараллеливанию.

Выполняемые перед распараллеливанием преобразования не выводят программу из класса последовательных программ, оставляя ее понятной для пользователя, детально незнакомого с особенностями параллельного программирования. Выполнение преобразований может выполняться в автоматическом или полуавтоматическом режимах.

Реализация данных возможностей в системе САПФОР позволила автоматизировать получение последовательных реализаций программ, эффективно отображаемых на современные кластеры автоматически распараллеливающим компилятором системы САПФОР, и была проверена на прикладной задаче моделирующей течение несжимаемой жидкости или слабосжимаемого газа около прямоугольной выемки.

Дальнейшие исследования будут направлены на расширение множества автоматически выполняемых преобразований и совместное использование в системе САПФОР трех описанных подходов к преобразованию программ: автоматическое устранение проблем, мешающих распараллеливанию программы, автоматизированное выполнение преобразований под руководством пользователя и полностью ручное преобразование программы.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 14-01-00109 а.

Литература

1. PLUTO – An automatic parallelizer and locality optimizer for multicores. URL: http://pluto-compiler.sourceforge.net/ (дата обращения 28.11.2014).
2. Par4All – Par4All 1.4.5 documentation. URL: http://www.par4all.org/ (дата обращения 28.11.2014).
3. Source-to-Source Parallelizing Compiler – Appentra. URL: http://www.appentra.com/ (дата обращения 28.11.2014).
4. Бахтин, В.А. Автоматическое распрраллеливание последовательных программ для многоядерных кластеров. / В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: Труды Международной научной конференции (Новороссийск, 20–25 сентября 2010 г.). — М.: Изд-во МГУ, 2010. — С. 12–15.

5. Бахтин, В.А. Диалог с программистом в системе автоматизации распараллеливания САПФОР. / В.А. Бахтин, И.Г. Бородич, Н.А. Катаев, М.С. Клинов, Н.В. Ковалева, В.А. Крюков, Н.В. Поддерюгина // Вестник Нижегородского университета им. Н.И. Лобачевского. — 2012 — № 5 (2). — С. 242–245.
6. Intel Parallel Studio. URL: http://software.intel.com/en-us/intel-parallel-studio-home (дата обращения 28.11.2014) .
7. Automatic Parallelization for Multi-processor/Multi-cores systems URL: http://www.parallelsp.com/ (дата обращения 28.11.2014).
8. Юрушкин, М.В. Диалоговый высокоуровневый оптимизирующий распараллеливатель (ДВОР). / М.В. Юрушкин, В.В. Петренко, Б.Я Штейнберг, Е.В. Алымова, А.А, Абрамов, А.П. Баглий, С.А. Гуда, Д.В. Дубров, Е.Н. Кравченко, Р.И. Морылев, З.Я. Нис, С.В. Полуян, И.С. Скиба, В.Н. Шаповалов., О.Б. Штейнберг, Р.Б. Штейнберг // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: Труды Международной научной конференции (Новороссийск, 20 – 25 сентября 2010 г.). — М.: Изд-во МГУ, 2010. — С. 71–75.
9. Бахтин, В.А. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами. / В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов // Вестник Южно-Уральского государственного университета. Серия «Математическое моделирование и программирование». — 2012 — № 18 (277), вып. 12. — С 82–92.
10. Коновалов, Н.А Fortran DVM — язык разработки мобильных параллельных программ. / Н.А. Коновалов, В.А. Крюков, С.Н. Михайлов, А.А. Погребцов // Программирование. — 1995. — № 1. — С. 49–54.
11. Almagor, L. Finding effective compilation sequences. / L. Almagor, K.D. Cooper, A. Grosul, T.J. Harvey, S.W. Reeves, D. Subramanian, L. Torczon, T. Waterman. // Proceedings of the ACM SIGPLAN/SIGBED 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04) (Washington, DC, 11–13 June 2004). — New York: NY, USA. — Vol. 39, Issue 7. — P. 231–239. DOI: 10.1145/997163.997196.
12. Бахтин, В.А. Распараллеливание с помощью DVM-системы некоторых приложений гидродинамики для кластеров с графическими процессорами. / В.А. Бахтин, И.Г. Бородич, Н.А. Катаев, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов // Научный сервис в сети Интернет: поиск новых решений: Труды Международной суперкомпьютерной конференции (Новороссийск, 17–22 сентября 2012 г.). — М.: Изд-во МГУ, 2012. — С. 444–450.
13. Катаев, Н.А. Преобразования последовательных программ при их распараллеливании с помощью системы САПФОР. / Н.А. Катаев, М.С. Клинов, Н.В, Поддерюгина. // Параллельные вычислительные технологии (ПаВТ'2013): Труды международной научной конференции (Челябинск, 1–5 апреля 2013 г.). — Челябинск: Издательский центр ЮУрГУ, 2013. — С. 387–393.
14. Бахтин, В.А Автоматическое отображение программ на языке Фортран на кластеры с графическими процессорами. / В.А. Бахтин, М.С. Клинов, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула // Вестник Южно-Уральского государственного университета. Серия «Вычислительная математика и информатика». — 2014. — Т. 3, № 3. — С. 86–96. DOI: 10.14529/cmse140305.

15. Штейнберг, Б.Я. Разбиение циклов для исполнения на суперкомпьютере с архитектурой перестраиваемого конвейера. / Б.Я. Штейнберг // Искусственный интеллект. — 2002. — № 3. — С. 331–338.
16. Давыдов, А.А. Моделирование течений несжимаемой жидкости и слабосжимаемого газа на многоядерных гибридных вычислительных системах. / А.А Давыдов, Б.Н. Четверушкин, Е.В. Шильников // Вычислительная математика и математическая физика. — 2010. — Т. 50, № 12. — С. 2275–2284.
17. pC++/Sage++ Home Page. URL: http://www.extreme.indiana.edu/sage/ (дата обращения: 01.12.2012).

Катаев Никита Андреевич, научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация), kataev_nik@mail.ru.

Буланов Артем Андреевич, студент, факультет Вычислительной Математики и Кибернетики, Московский государственный университет им. М.В. Ломоносова (Москва, Российская Федерация), bulanov.artyom@gmail.com.

Поступила в редакцию 10 апреля 2015 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2015, vol. 4, no. 3, pp. 13–23

DOI: 10.14529/cmse150302

AUTOMATED TRANSFORMATION OF FORTRAN PROGRAMS ESSENTIAL FOR THEIR EFFICIENT PARALLELIZATION THROUGH SAPFOR SYSTEM

N.A. Kataev, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation) kataev_nik@mail.ru,

A.A. Bulanov, Lomonosov Moscow State University (Moscow, Russian Federation) bulanov.artyom@gmail.com

Automatic mapping of sequential programs on height performance computers with distributed memory may require preliminary transformation of programs oriented to this class of systems. Using the SAPFOR system for parallelization of applications allowed to explore transformations which can be executed in an automated way. The paper presents transformations that increase the possibility of efficient parallelization of programs by eliminating the reasons that prohibit the parallel execution of loops. Implementation of these transformations allowed to automate the obtainment of serial hydrodynamic program, which may be efficiently mapped on modern clusters by the automatically parallelizing compiler included in the SAPFOR.

Keywords: parallelization, automation, transformation of serial programs, Fortran.

References

1. PLUTO – An automatic parallelizer and locality optimizer for multicores. URL: http://pluto-compiler.sourceforge.net/ (accessed: 28.11.2014).
2. Par4All – Par4All 1.4.5 documentation. URL: http://www.par4all.org/ (accessed: 28.11.2014).
3. Source-to-Source Parallelizing Compiler – Appentra. URL: http://www.appentra.com/ (accessed: 28.11.2014).
4. Bakhtin V.A, Klinov M.S., Krukov V.A., Podderugina N.V. Avtomaticheskoe rasprrallelivanie posledovatel'nyh programm dlja mnogojadernyh klasterov [Automatic parallelization of sequential programs for multi-core clusters.]. Nauchnyj servis v seti Internet: superkomp'juternye centry i zadachi: Trudy Mezhdunarodnoj nauchnoj konferencii (Novorossijsk, 20 – 25 sentjabrja 2010 g.) [Scientific service on the Internet: supercomputer centers and tasks: Proceedings of the International Scientific Conference (Novorossijsk, Russia, September, 20 – 25, 2010)]. — Moscow, Moscow University Press, 2010. P. 12–15.
5. Bahtin V.A., Borodich I.G., Kataev N.A., Klinov M.S., Kovaleva N.V., Krukov V.A., Podderugina N.V. Dialog s programmistom v sisteme avtomatizacii rasparallelivanija SAPFOR [Interaction with the programmer in the system for automation parallelization SAPFOR]. Vestnik Nizhegorodskogo universiteta im. N.I. Lobachevskogo [Vestnik of Lobachevsky State University of Nizhni Novgorod]. 2012 No. 5 (2). P. 242–245.
6. Intel Parallel Studio. URL: http://software.intel.com/en-us/intel-parallel-studio-home (accessed: 28.11.2014).
7. Automatic Parallelization for Multi-processor/Multi-cores systems URL: http://www.parallelsp.com/ (accessed: 28.11.2014).
8. Jurushkin M.V., Petrenko V.V., Shtejnberg B.Ja., Alymova E.V.,Abramov A.A., Baglij A.P., Guda S.A., Dubrov D.V., Kravchenko E.N., Morylev R.I., Nis Z.Ya., Polujan S.V., Skiba I.S., Shapovalov V.N., Shtejnberg O.B., Shtejnberg R.B. Dialogovyj vysokourovnevyj optimizirujushhij rasparallelivatel' (DVOR) [Interactive High-level Optimizing Parallelizer (IHOP). Nauchnyj servis v seti Internet: superkomp'juternye centry i zadachi: Trudy Mezhdunarodnoj nauchnoj konferencii (Novorossijsk, 20–25 sentjabrja 2010 g.) [Scientific service on the Internet: supercomputer centers and tasks: Proceedings of the International Scientific Conference (Novorossijsk, Russia, September, 20–25, 2010)]. — Moscow, Moscow University Press, 2010. P. 71–75.
9. Bakhtin V.A, Klinov M.S., Krukov V.A., Podderugina N.V., Pritula M.N., Sazanov Yu.L. Rasshirenie DVM-modeli parallel'nogo programmirovaniya dlya klasterov s geterogennymi uzlami [Extension of the DVM-model of parallel programming for clusters with heterogeneous nodesExtension of the DVM-model of parallel programming for clusters with heterogeneous nodes]. Vestnik Yuzho-Uralskogo gosudarstvennogo universiteta. Seriya "Matematicheskoe modelirovanie i programmirovanie" [Bulletin of South Ural State University. Series: Mathematical Modeling, Programming & Computer Software]. 2012 No. 18 (277), Issue 12. P 82–92.
10. Konovalov N.A., Krukov V.A, Mikhajlov S.N., Pogrebtsov A.A. Fortan DVM: a Language for Portable Parallel Program Development // Programming and Computer Software. 1995. Vol. 21, No. 1. P. 35–38.
11. Almagor L., Cooper K.D., Grosul A., Harvey T.J., Reeves S.W., Subramanian D., Torczon L., Waterman T. Finding effective compilation sequences // Proceedings of the

ACM SIGPLAN/SIGBED 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04) (Washington, DC, 11–13 June 2004). New York: NY, USA. Vol. 39, Issue 7. P. 231–239. DOI: 10.1145/997163.997196.

12. Bakhtin V.A, Borodich I.G, Kataev N.A., Klinov M.S., Krukov V.A., Podderugina N.V., Pritula M.N., Sazanov Yu.L. Rasparallelivanie s pomoshh'yu DVM-sistemy nekotoryx prilozhenij gidrodinamiki dlya klasterov s graficheskimi processorami [Parallelization using the DVM-system some applications hydrodynamics for clusters with GPUs]. Nauchnyj servis v seti Internet: poisk novyx reshenij: Trudy Mezhdunarodnoj nauchnoj konferencii (Novorossijsk, 17–22 sentjabrja 2012) [Scientific service on the Internet: search for new solutions: Proceedings of the International Scientific Conference (Novorossijsk, Russia, September, 17–22, 2012)]. — Moscow, Moscow University Press, 2012. P. 444–450.
13. Kataev N.A., Klinov M.S., Podderugina N.V. Preobrazovaniya posledovatel'nyx programm pri ix rasparallelivanii s pomoshh'yu sistemy SAPFOR [Transformations of sequential programs during programs parallelization with the help of SAPFOR.]. Parallelnye vychislitelnye tekhnologii (PaVT'2010): Trudy mezhdunarodnoj nauchnoj konferentsii (Chelyabinsk, 1–5 aprelya 2013) [Parallel Computational Technologies (PCT'2013): Proceedings of the International Scientific Conference (Chelyabinsk, Russia, April, 1–5, 2013)]. Chelyabinsk, Publishing of the South Ural State University, 2013. P. 387–393.
14. Bahtin V.A., Klinov M.S., Kolganov A.S., Krukov V.A., Podderugina N.V., Pritula M.N. Avtomaticheskoe otobrazhenie programm na jazyke Fortran na klastery s graficheskimi processorami [Automatic mapping of Fortran programs on clusters with accelerators]. Vestnik Yuzho-Uralskogo gosudarstvennogo universiteta. Seriya: "Vychislitel'naja matematika i informatika" [Bulletin of South Ural State University. Series: Computational Mathematics and Software Engineering]. 2014. Vol. 3, No. 3. P. 86–96. DOI: 10.14529/cmse140305.
15. Shtejnberg B.Ja. Razbienie ciklov dlja ispolnenija na superkomp'jutere s arhitekturoj perestraivaemogo konvejera [Splitting of loops for execution on a supercomputer with the architecture of configurable pipeline]. Iskusstvennyj intellekt [Artificial intelligence]. 2002. No. 3. P. 331–338.
16. Davydov A.A., Chetverushkin B.N., Shilnikov E.V. Modelirovanie techenij neszhimaemoj zhidkosti i slaboszhimaemogo gaza na mnogojadernyh gibridnyh vychislitel'nyh sistemah [Simulating flows of incompressible and weakly compressible fluids on multicore hybrid computer systems]. // Vychislitel'naja matematika i matematicheskaja fizika [Computational Mathematics and Mathematical Physics]. 2010, Vol. 50, No. 12. P. 2157–2165.
17. pC++/Sage++ Home Page. URL: http://www.extreme.indiana.edu/sage/ (accessed: 01.12.2012).

Received April 10, 2015.

УДК 004.43 DOI: 10.14529/cmse150303

СОВРЕМЕННЫЕ И ПЕРСПЕКТИВНЫЕ ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ С РЕКОНФИГУРИРУЕМОЙ АРХИТЕКТУРОЙ[1]

И.И. Левин, А.И. Дордопуло, И.А. Каляев, Ю.И. Доронченко, М.К. Раскладкин

В статье рассматриваются архитектура и сравнительные технические характеристики реконфигурируемых вычислительных систем (РВС) на основе программируемых логических интегральных схем (ПЛИС) семейства Xilinx Virtex-7: вычислительных модулей 24V7-750 и «Тайгета», устанавливаемых в вычислительный шкаф, и реконфигурируемого вычислительного блока «Калеано» для настольного использования. Рассматриваются также архитектура и компоновка РВС нового поколения на основе жидкостного охлаждения, приводятся результаты расчетов и макетирования основных технических решений. Рассматриваются технологии решения прикладных задач с помощью комплекса средств разработки прикладного программного обеспечения. РВС нового поколения на основе жидкостного охлаждения позволяют достичь производительности 1 Пфлопс в стандартном вычислительном шкафу высотой 47U при потребляемой мощности 150 кВт, обеспечивая, тем самым, существенное преимущество по таким технико-экономическим параметрам, как реальная и удельная производительность, энергоэффективность, массогабаритные характеристики и другим по сравнению с аналогичными системами.

Ключевые слова: реконфигурируемые вычислительные системы, программируемые логические интегральные схемы, вычислительный модуль, прикладное программное обеспечение.

Введение

Одним из перспективных способов достижения высокой реальной производительности вычислительной системы является адаптация ее архитектуры под структуру решаемой задачи и создание вычислительного устройства, одинаково эффективно реализующего как структурные, так и процедурные фрагменты вычислений.

Поэтому отечественные [1] и зарубежные производители вычислительной техники все чаще включают в состав разрабатываемых вычислительных систем программируемые логические интегральные схемы (ПЛИС) для ускорения работы при реализации вычислительно трудоемких алгоритмов.

Создаются как отдельные ускорители с одним-двумя кристаллами ПЛИС, так и целые вычислительные комплексы. Такие фирмы как Nallatech (http://www.nallatech.com) и Pico Computing (http://picocomputing.com) выпускают ряд ускорителей и несущих плат с небольшим числом (до четырёх) кристаллов ПЛИС, которые используются в создании серверов и гибридных кластерных систем фирмами HP и IBM.

[1] Статья рекомендована к публикации Программным комитетом конференции «Параллельные вычислительные технологии – 2015».

Компании Convey (http://www.conveycomputer.com) и Maxeler Technologies (http://www.maxeler.com) создают гибридные суперкомпьютеры на основе собственных гетерогенных кластерных узлов, каждый из которых может содержать от одного до четырех кристаллов ПЛИС и несколько универсальных процессоров.

Похожее решение используется и компанией SRC (http://www.srccomp.com), которая выпускает узлы, названные MAP processor, для стойки (MAPstation) форм-фактором 1U, 2U и 4U. MAPstation 1U содержит один MAP processor. MAPstation 2U содержит до трех MAP processor. MAPstation 4U может содержать до 10 различных модулей — MAP processor, модуль с универсальным микропроцессором или модуль памяти.

В Научно-исследовательском центре суперЭВМ и нейрокомпьютеров («НИЦ СЭ и НК», г. Таганрог) в содружестве с Научно-исследовательским институтом многопроцессорных вычислительных систем Южного федерального университета (НИИ МВС ЮФУ, г. Таганрог) и Южным научным центром Российской академии наук (ЮНЦ РАН, г. Ростов-на-Дону) разрабатываются и производятся реконфигурируемые вычислительные системы класса суперЭВМ, где основным вычислительным ресурсом являются не микропроцессоры, а множество кристаллов ПЛИС, объединенных в вычислительные поля посредством высокоскоростных каналов передачи данных.

Спектр выпускаемых и проектируемых изделий достаточно широк: от полностью автономных малогабаритных реконфигурируемых ускорителей (вычислительных блоков), вычислительных модулей в настольном или стоечном конструктивном исполнении до вычислительных систем, состоящих из нескольких вычислительных шкафов, размещаемых в специально оборудованном машинном зале.

Реконфигурируемые вычислительные системы, содержащие большие вычислительные поля ПЛИС, находят применение для решения вычислительно трудоемких задач в различных областях науки и техники, поскольку обладают рядом существенных преимуществ по сравнению с многопроцессорными вычислительными системами кластерной архитектуры: высокими реальной и удельной производительностями, высокой энергоэффективностью и др.

В данной статье рассмотрены архитектура и технические характеристики реконфигурируемых вычислительных систем (РВС) на основе программируемых логических интегральных схем (ПЛИС) семейства Xilinx Virtex-7: вычислительных модулей 24V7-750 и «Тайгета», устанавливаемых в вычислительный шкаф, и реконфигурируемого вычислительного блока «Калеано» для настольного использования. Статья организована следующим образом. В разделе 1 описаны реконфигурируемые вычислительные системы на основе ПЛИС Xilinx Virtex-7. Раздел 2 посвящен программному обеспечению РВС. В заключении сделан вывод о том, что РВС на основе ПЛИС Xilinx Virtex UltraScale можно рассматривать как основу для создания высокопроизводительных вычислительных комплексов нового поколения, обеспечивающих высокую эффективность вычислений и близкий к линейному рост производительности при наращивании вычислительного ресурса.

1. РВС на основе ПЛИС Xilinx Virtex-7

1.1. Реконфигурируемый вычислительный блок «Калеано»

На основе ПЛИС Xilinx седьмой серии разработан реконфигурируемый вычислительный блок (РВБ) «Калеано» в конструктивном исполнении высотой 1U, предназначенный для обработки данных, поступающих по каналу Gigabit Ethernet без поддержки IP-протоколов. На рис. 1 представлены фотографии РВБ «Калеано».

РВБ «Калеано» выпускается в двух модификациях: «Калеано-K» на ПЛИС Kintex-7 XC7K160T и «Калеано-V» на ПЛИС Virtex-7 XC7VX485T. Технические характеристики РВБ «Калеано» для этих модификаций приведены в табл. 1.

а) общий вид РВБ «Калеано» б) РВБ «Калеано» со снятой верхней крышкой и печатная плата

Рис. 1. Фотографии РВБ «Калеано» и его составных частей

Таблица 1

Технические характеристики РВБ «Калеано» для модификаций «Калеано-K» и «Калеано-V»

Технический параметр	**Модификации РВБ «Калеано»**	
	Калеано-K	Калеано-V
Число и тип ПЛИС	6	6
Общее число эквивалентных вентилей в ПЛИС вычислительного поля, млн.	96	288
Объем оперативной памяти, Гб	3	3
Производительность вычислительного модуля Pi_{32}/Pi_{64}, Гфлопс	150/75	440/220
Частота работы, МГц	330	400
Скорость обмена данными по каналу Ethernet, Гбит/с	1	1
Частота передачи данных по LVDS между ПЛИС вычислительного поля, МГц	900	1200
Потребляемая мощность, Вт	200	320
Габаритные размеры, мм	480x270x70	480x270x70
Стоимость, млн. руб.	1,3	2,0

РВБ «Калеано» содержит вычислительное поле из шести ПЛИС, встроенную управляющую ЭВМ, систему питания, систему управления, систему охлаждения и другие подсистемы.

Все ПЛИС вычислительного поля соединены между собой по принципу близкодействия с помощью LVDS-каналов, ко всем ПЛИС РВБ подключены модули динамической памяти емкостью 256 Мбайт каждый.

Для управления и конфигурирования вычислительного поля РВБ используется ЭВМ семейства COM-Express фирмы Kontron, установленная непосредственно на плате вычислительного модуля.

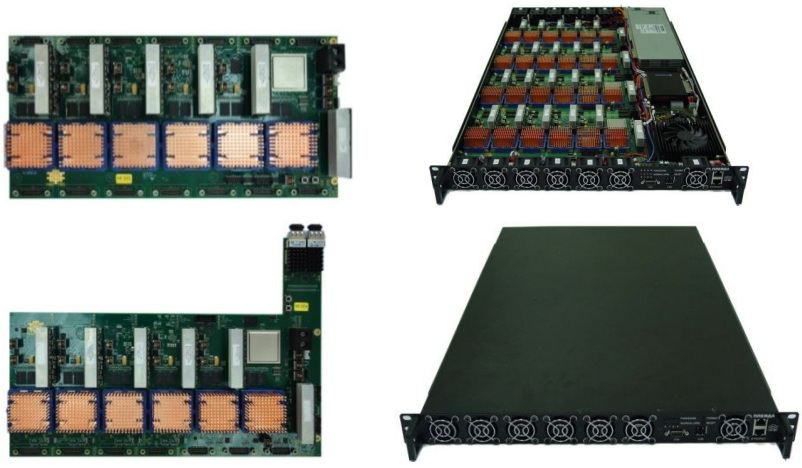
С ее помощью подключается периферийное оборудование, осуществляются подготовка и отладка параллельной программы для решения вычислительно трудоемких задач, формируются файлы исходных данных, которые вместе с исполняемым модулем задачи через шину PCI-Express загружаются в вычислительное поле по LVDS-каналу.

После выполнения задачи результаты решения пересылаются в процессорный модуль COM-Express.

1.2. Реконфигурируемый вычислительный модуль «Плеяда»

По государственному контракту № 14.527.12.0004 от 03.10.2011 г. в НИИ МВС ЮФУ была разработана реконфигурируемая вычислительная система РВС-7 на основе ПЛИС Virtex-7, которая содержит вычислительное поле из 576 микросхем ПЛИС Virtex-7 XC7V585T-FFG1761 объемом 58 миллионов эквивалентных вентилей каждая, конструктивно объединенных в один вычислительный шкаф высотой 47U с пиковой производительностью 10^{15} операций с фиксированной запятой в секунду.

Основным структурным компонентом РВС-7, предназначенным для установки в стандартную 19" вычислительную стойку, является вычислительный модуль (ВМ) 24V7-750 (ВМ «Плеяда»), представленный на рис. 2, в состав которого входят: четыре платы вычислительного модуля (ПВМ) 6V7-180; управляющий модуль УМ-7; подсистема питания; подсистема охлаждения и другие подсистемы.



а) платы вычислительного модуля «Плеяда»

б) вычислительный модуль «Плеяда» ВМ 24V7-750

Рис. 2. Фотография вычислительного модуля (ВМ) 24V7-750

В состав каждой платы ВМ 24V7-750 входят: вычислительное поле, состоящее из шести ПЛИС XC7V585T-1FFG1761 семейства Virtex-7, соединенных последовательно с возможностью передачи данных по 144 дифференциальным линиям LVDS-интерфейса на частоте 800 МГц, и распределенная память общим объемом 12 Гбайт (на каждой

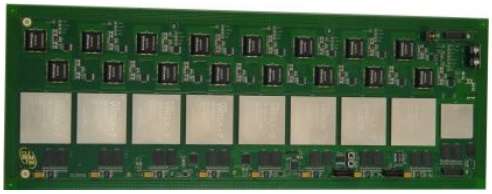
плате установлены 12 микросхем динамической памяти MT47H128M16HR-25E с организацией 128 M*16 и частотой записи/чтения до 400 МГц.

Производительность одной платы составляет 645,9 Гфлопс при обработке 32-разрядных данных с плавающей запятой, а производительность ВМ 24V7-750 составляет 2,58 Тфлопс при обработке 32-разрядных данных с плавающей запятой.

1.3. Реконфигурируемый вычислительный модуль «Тайгета»

В ООО «НИЦ СЭ и НК» на основе ПЛИС Virtex-7 разработан 19" вычислительный модуль «Тайгета» высотой 2U, предназначенный для создания высокопроизводительных многостоечных РВС. ВМ «Тайгета», представленный на рис. 3, содержит четыре платы 8V7-200, встроенную управляющую ЭВМ, систему питания, систему управления, систему охлаждения и другие подсистемы. Платы в составе ВМ «Тайгета» соединяются LVDS-каналами, работающими на частотах до 1000 МГц. Внешний вид платы 8V7-200 показан на рис. 3-б.

а) ВМ «Тайгета» со снятой верхней крышкой



б) плата 8V7-200

Рис. 3. Фотография ВМ «Тайгета»

Плата вычислительного модуля (ПВМ) 8V7-200 представляет собой 20-слойную печатную плату с двухсторонним монтажом элементов, на которой располагаются восемь ПЛИС типа XC7VX485T-1FFG1761, содержащих по 48,5 миллионов эквивалентных вентилей, 16 микросхем распределенной памяти SDRAM типа DDR2 общим объемом 2 Гбайт, интерфейсы LVDS и Ethernet и другие компоненты.

Производительность одной ПВМ 8V7-200 составляет 667 Гфлопс при обработке 32-разрядных данных с плавающей запятой, а производительность ВМ «Тайгета» составляет соответственно 2,66 Тфлопс.

1.4. РВС на основе ВМ «Плеяда» и ВМ «Тайгета»

На основе рассмотренного выше ВМ «Плеяда» в 2013 году была создана реконфигурируемая вычислительная система РВС-7 (рис. 4-а), содержащая 24 вычислительных модуля и имеющая возможность наращивания системы до 36 вычислительных модулей. Производительность РВС-7 при комплектации от 24 до 36 ВМ 24V7-750 составляет от 62 до 93 Тфлопс при обработке 32-разрядных данных с плавающей запятой и 19,4÷29,4 Тфлопс при обработке 64-разрядных данных с плавающей запятой соответственно. Областями применения РВС-7 и вычислительных комплексов на ее основе являются решение задач цифровой обработки сигналов и многоканальная цифровая фильтрация.

На основе ВМ «Тайгета» была создана РВС, внешний вид которой представлен на рис. 4-б. Производительность одной стойки РВС при комплектации 18 ВМ «Тайгета»

составляет 48 Тфлопс при обработке данных с плавающей запятой одинарной точности и 23 Тфлопс при обработке 64-разрядных данных с плавающей запятой.

а) РВС-7 на основе ВМ «Плеяда» б) РВС на основе ВМ «Тайгета»

Рис. 4. Внешний вид РВС на основе ПЛИС Xilinx Virtex-7

Высокопроизводительные РВС на основе ВМ «Тайгета» ориентированы на решение вычислительно трудоемких задач науки и промышленности, задач синтеза лекарств и задач символьной обработки и обеспечивают при этом существенное конкурентное преимущество по большинству технико-экономических параметров: удельной производительности, энергоэффективности и другим по сравнению с суперЭВМ традиционной кластерной архитектуры.

1.5. Перспективные реконфигурируемые системы на основе ПЛИС Xilinx UltraScale

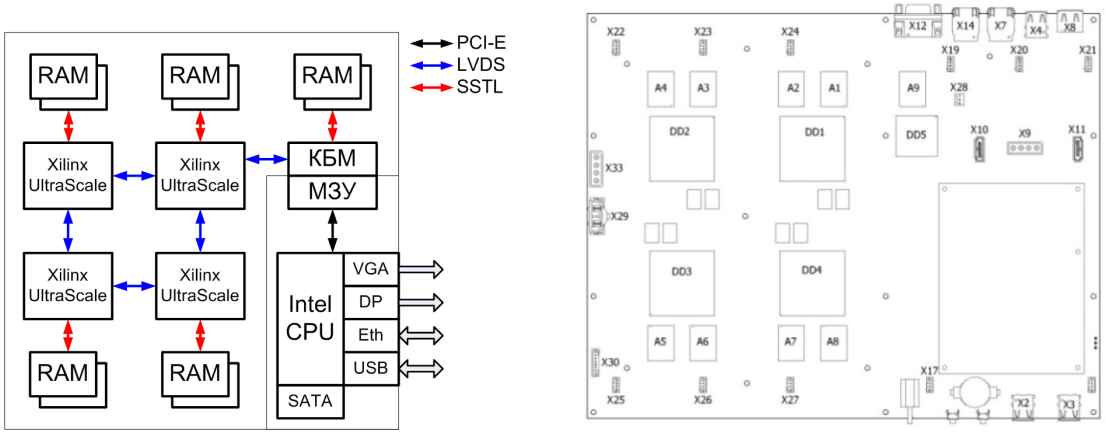
Дальнейшим развитием использованной при построении РВС на основе ПЛИС Xilinx Virtex-7 открытой масштабируемой архитектуры [2] является использование в новых разрабатываемых изделиях перспективной элементной базы — ПЛИС Xilinx нового поколения семейства UltraScale, выполненных по технологии 20 нм и обладающих сниженным энергопотреблением и повышенным быстродействием по сравнению с ПЛИС седьмого семейства.

1.6. Реконфигурируемый вычислительный блок на основе ПЛИС UltraScale

Разрабатываемый РВБ «Калеано-U» будет выполнен также в конструктиве высотой 1U, но в отличие от своих предшественников «Калеано-K» и «Калеано-V», будет содержать четыре ПЛИС Xilinx UltraScale XCVU095 объемом 95 млн. эквивалентных вентилей каждая, что позволит создать вычислительное поле общим объемом 380 млн. эквивалентных вентилей. Структурная схема РВБ «Калеано-U» и эскиз компоновки платы показаны на рис. 5,а-б.

На рис. 5 обозначены:

- DD1-DD4 — вычислительные ПЛИС Xilinx UltraScale XCVU095;
- DD5 — ПЛИС контроллера ПВМ Xilinx UltraScale XCKU040;
- A1-A9 — модули распределенной памяти;
- X2-X4, X7-X12 — разъемы различных типов интерфейсов.



а) структурная схема РВБ «Калеано-U»

б) эскиз компоновки платы РВБ

Рис. 5. РВБ «Калеано-U»

По сравнению с предыдущей версией РВБ «Калеано-V» производительность РВБ «Калеано-U» возрастет в 1,7–1,8 раза при увеличении потребляемой мощности не более чем в 1,3 раза.

1.7. РВС с жидкостным охлаждением на основе ПЛИС UltraScale

Использование воздушных систем охлаждения современных высокопроизводительных вычислительных систем и построенных на их основе суперкомпьютеров, в том числе реконфигурируемых, практически достигло своего предела. Большинство разработчиков вычислительной техники в качестве перспективы для решения проблем охлаждения проектируемых вычислительных комплексов ориентируются на применение жидкостного охлаждения.

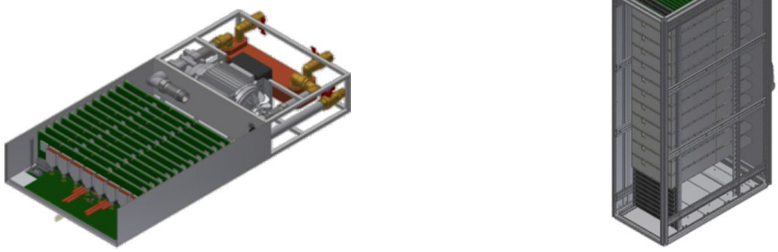
Для вычислительных модулей РВС, проектируемых на основе перспективных семейств ПЛИС, целесообразно использовать жидкостное охлаждение, в частности, непосредственное погружение плат вычислительных модулей в жидкостный хладагент на основе минерального масла.

В ООО «НИЦ СЭ и НК» активно развивается направление по созданию РВС нового поколения на основе жидкостного охлаждения. Разработаны новые конструкции печатных плат и вычислительных модулей, характеризующиеся высокой плотностью компоновки. В частности, в настоящее время ведется разработка перспективных вычислительных модулей «Скат-8» для многостоечных РВС сверхвысокой производительности.

Плата перспективного вычислительного модуля содержит восемь ПЛИС семейства Virtex UltraScale логической емкостью не менее 100 млн. эквивалентных вентилей каждая. Вычислительный модуль состоит из двух секций: в первой секции размещается 16 плат вычислительных модулей с потребляемой мощностью до 800 Вт каждая, причем все платы полностью погружены в электрически нейтральный жидкостный хладагент; во второй секции располагаются насосная группа и теплообменник, обеспечивающие проток и охлаждение хладагента. Эскиз ВМ с высотой конструктива 3U показан на рис. 6-а.

Согласно проведенным расчетам, использование жидкостного охлаждения и построение вычислительных систем на основе ВМ «Скат-8» обеспечивает сверхпетафлопсную производительность одного вычислительного шкафа РВС.

Вычислительная 19” стойка суперкомпьютера, эскиз которой представлен на рис. 6-б, может содержать до 12 ВМ «Скат-8» с жидкостным охлаждением. Производительность и потребляемая мощность перспективной РВС приведены в табл. 2.



а) эскиз ВМ «Скат-8»

б) эскиз вычислительной стойки на основе ВМ «Скат-8»

Рис. 6. Эскиз вычислительной системы на основе жидкостного охлаждения

Таблица 2

Производительность и потребляемая мощность перспективной РВС на основе ПЛИС Xilinx UltraScale

Характеристика	**Значение**
Производительность ВМ «Скат-8»	105 Тфлопс
Производительность вычислительной стойки на основе ВМ «Скат-8»	1 Пфлопс
Потребляемая мощность ВМ «Скат-8»	13 кВт
Потребляемая мощность вычислительной стойки на основе ВМ «Скат-8»	150 кВт

На основе производимых в «НИЦ СЭ и НК» реконфигурируемых систем можно проследить темпы роста производительности РВС при переходе от семейства к семейству ПЛИС (табл. 3).

Таблица 3

Производительность реконфигурируемых суперЭВМ

Изделие, год выпуска, семейство ПЛИС	Производительность платы Pi_{32}/Pi_{64}, Гфлопс	Производительность ВМ Pi_{32}/Pi_{64}, Гфлопс	Производительность шкафа 47U Pi_{64}, Тфлопс
«Орион-5», 2009 год, Virtex-5	250/85	1000/340	19,2—28,8
«Ригель», 2010/2012 год, Virtex-6	400/125	1600/500	34,5—51,8
«Тайгета», 2012/2013 год, Virtex-7	900/300	3600/1200	68—100
«Скат», 2015/2016 год, UltraScale	7250/2500	82500/30000	1000—1250

На основе представленного конструктивного исполнения в 2015–2016 годах будут созданы сверхпроизводительные вычислительные комплексы, в которых обеспечивается

эффективное охлаждение вычислительных ПЛИС как семейства UltraScale, так и следующего прогнозируемого семейства ПЛИС.

2. Программное обеспечение РВС

В настоящее время существует множество различных систем проектирования, позволяющих создавать структурные решения прикладных задач на ПЛИС. Наиболее популярными как в качестве отдельного средства разработки, так и в составе комплексов, являются программы-синтезаторы, предоставляемые фирмами-изготовителями ПЛИС: ISE и Vivado фирмы Xilinx [3], Quartus II фирмы Altera [4] и Actel Libero IDE фирмы Actel [5]. Данные средства, помимо непосредственно среды проектирования цифровых устройств, включают в себя ряд вспомогательных утилит: анализаторы временных характеристик, редакторы размещения, модули программирования ПЛИС, системы моделирования цифровых устройств и др. Благодаря широкому инструментарию эти системы проектирования обеспечивают полный цикл разработки цифровых устройств в пределах одного кристалла ПЛИС: создание исходного описания проекта, синтез, моделирование, размещение, трассировку, конфигурирование кристалла.

В связи с постоянным ростом ёмкости кристаллов ПЛИС проектирование решений прикладных задач в кристаллах ПЛИС с помощью языков описания аппаратуры (VHDL, AHDL, Verilog и др.), а также разработка цифровых устройств в графических редакторах становится все более трудоемкой. Поэтому в настоящее время ведущие производители ПЛИС и реконфигурируемых вычислителей ориентируются на языки высокого уровня. Так, в новой среде разработки Vivado фирмы Xilinx добавлен новый инструмент проектирования Vivado HLS [6], основанный на языке высокого уровня, а компания Altera для своих ПЛИС предлагает инструментарий разработки Altera SDK (http://www.altera.com/literature/lit-opencl-sdk.jsp) для нового стандарта параллельного программирования гетерогенных систем OpenCL. В данных решениях используются трансляторы C-подобных языков, генерирующие из программы на аналоге языка программирования высокого уровня C код на языках описания аппаратуры на уровне регистровых передач (RTL-уровень, трансляторы C-to-RTL).

Несмотря на схожесть синтаксисов данных C-подобных языков с самим языком C, подобный подход вовсе не означает, что исходный код на языке C, написанный под персональный компьютер или кластерную вычислительную систему, будет понят трансляторами C-to-RTL. Выбор языка C в качестве основы обусловлен широкой распространенностью данного языка, что существенно упрощает освоение новых инструментариев разработки решений прикладных задач для ПЛИС.

Также при использовании трансляторов C-to-RTL весь код программы либо явно указанные процедуры транслируются в RTL-описания отдельных кристаллов ПЛИС. В подобных системах разработки отсутствует инструментарий, обеспечивающий автоматические разбиения параллельной программы на множество связанных кристаллов ПЛИС.

В Vivado HLS проект разрабатывается в рамках одного кристалла ПЛИС, и если программисту недостаточно аппаратного ресурса кристалла, то он вынужден самостоятельно распределять вычисления между несколькими проектами для каждой ПЛИС и синхронизировать управляющие и информационные потоки между ними.

Стандарт OpenCL используется компанией Nallatech (разработчик реконфигурируемых вычислителей) и подразумевает использование нескольких ПЛИС в одном проекте.

Программирование решений в кристаллах ПЛИС в данном случае осуществляется с помощью вызова функций из библиотек инструментария разработки Altera SDK, и в каждом задействованном кристалле ПЛИС выполняются вычисления, описанные отдельным участком кода. Таким образом, программа, написанная в стандарте OpenCL, представляет собой основной код, написанный под традиционные процессоры, и отдельные участки кода, написанные под ПЛИС, задействованные как сопроцессоры. В данном случае задача синхронизации данных также возлагается на самого программиста.

Еще одним известным средством программирования ПЛИС является комплекс, разработанный компанией Mitrionics, состоящий из виртуального процессора Mitrion Virtual Processor (MVP), программирование которого осуществляется на языке высокого уровня Mitrion-C, и библиотеки функций для построения хост-программ MithalAPI, входящих в пакет разработки Mitrion SDK [7]. Разрабатываемая на языке Mitrion-C программа должна быть полностью реализована на одном виртуальном процессоре MVP, что не позволяет программировать многокристальные РВС, и следовательно, существенно снижает эффективность использования программного комплекса компании Mitrionics. Для программирования многокристальных РВС со связями между ПЛИС программисту необходимо самому реализовать интерфейс (протокол) обмена данными между ПЛИС и решить вопросы, связанные с синхронизацией потоков данных. В этом случае программа для РВС вырождается в программу для кластера (множество процессоров MVP), реализованного на ПЛИС, что существенно снижает эффективность реализации задач на многокристальных РВС.

2.1. Язык COLAMO и комплекс средств программирования многокристальных РВС

Альтернативный подход к программированию РВС предлагается в НИИ МВС ЮФУ, занимающемся разработкой многокристальных реконфигурируемых вычислительных систем различных архитектуры и конфигурации уже более 15 лет.

Опыт решения различных классов задач в НИИ МВС ЮФУ показал, что для эффективного решения современных трудоемких задач программисту необходимы средства программирования, которые обеспечивают следующие основные функции:

- программирование на языке высокого уровня;
- поддержку многокристального программирования;
- обеспечение высокой частоты работы ПЛИС;
- высокую плотность заполнения кристалла ПЛИС;
- поддержку конвейерной и макроконвейерной организации вычислений.

В НИИ МВС ЮФУ разработан и широко используется программный комплекс, состоящий из следующих компонентов:

- транслятор языка программирования COLAMO, осуществляющего трансляцию исходного кода на COLAMO в информационный граф параллельной прикладной программы;
- синтезатор масштабируемых схемотехнических решений на уровне логических ячеек ПЛИС Fire!Constructor, осуществляющего отображение полученного от транслятора языка программирования COLAMO информационного графа на архитектуру РВС, размещение отображенного решения по кристаллам ПЛИС и автоматическую синхронизацию фрагментов информационного графа в разных кристаллах ПЛИС;

- библиотеки IP-ядер, соответствующих операторам языка COLAMO (функционально-законченных структурно-реализованных аппаратных устройств) для различных предметных областей, и интерфейсов для согласования скорости обработки информации и связи в единую вычислительную структуру;
- средства отладки и программ доступа и мониторинга состояния РВС.

Язык высокого уровня COLAMO предназначен для описания реализации параллельного алгоритма и создания на основе принципов структурно-процедурной организации вычислений [1, 8, 9] в архитектуре РВС специализированной вычислительной структуры, которая предполагает последовательную смену структурно (аппаратно) реализованных фрагментов информационного графа задачи, каждый из которых является вычислительным конвейером потока операндов.

Таким образом, приложение (прикладная задача) для РВС состоит из структурной составляющей, представленной в виде аппаратно реализованных фрагментов вычислений, и процедурной составляющей, представляющей собой единую для всех структурных фрагментов управляющую программу последовательной смены вычислительных структур и организации потоков данных.

Для представления такой организации вычислений в языке используется понятие «кадр». Кадром является программно-неделимый компонент, представляющий собой совокупность операторов, которые реализуются в виде арифметико-логических команд и команд чтения/записи, выполняемых на различных функциональных устройствах, соединенных между собой в соответствии с информационной структурой алгоритма.

В языке COLAMO отсутствуют явные формы описания параллелизма, а распараллеливание достигается с помощью объявления типов доступа к переменным и индексации элементов массивов, что характерно для языков потока данных. Для обращения к данным используется два основных метода доступа: параллельный доступ (задаваемый типом Vector) и последовательный доступ (задаваемый типом Stream). Степень параллелизма определяется по минимальному значению параметра распараллеливания. Для типа доступа Stream степень параллелизма равна 1, а для типа доступа Vector определяется наименьшим значением векторной составляющей каждого массива, участвующего в вычислениях. Для параллельного типа доступа возможна одновременная обработка всех размерностей массивов, заданных типом Vector, при этом повышается аппаратный ресурс на обработку, но снижается время обработки.

Многомерные массивы данных могут иметь множество измерений, каждое из которых может иметь последовательный или параллельный тип доступа, задаваемый ключевым словом Stream или Vector соответственно. Изменение типа доступа позволяет достаточно просто управлять как степенью распараллеливания вычислений на уровне описания структур данных, так и скоростью обработки и занимаемым ресурсом, что позволяет программисту описывать различные виды параллелизма в достаточно сжатом виде.

Помимо типа доступа, для переменной в языке COLAMO определены также и типы ее хранения: мемориальный (Mem), регистровый (Reg) и коммутационный (Com).

Мемориальной переменной называется величина, хранящаяся в ячейке распределенной памяти и, следовательно, сохраняющая свое значение до очередного переприсваивания. Для мемориальной переменной возможно одновременное выполнение только одного процесса. Поэтому в семантике языка COLAMO для мемориальной переменной в теле

кадра действуют правило однократного присваивания и правило единственной подстановки. Правило однократного присваивания указывает на то, что мемориальная переменная в кадре изменяет свое значение только один раз. Правило единственной подстановки определяет, что переменная в кадре может использоваться только для одного процесса чтения или записи.

Для описания связей между элементами информационного графа задачи в языке COLAMO предназначена коммутационная переменная. Поскольку коммутационная переменная описывает информационные связи, то она не требует никакого вычислительного аппаратного ресурса для своей реализации. Доступ к значению коммутационной переменной после выполнения кадра невозможен. Коммутационная переменная необходима транслятору для указания информационных зависимостей при построении вычислительной структуры задачи. На коммутационную переменную, как и на мемориальную переменную, действует правило однократного присваивания, но для нее не выполняется правило единственной подстановки. Использование коммутационных переменных позволяет легко разветвлять и дублировать потоки данных, но не позволяет реализовать рекурсию.

Для организации рекурсии в языке COLAMO используется регистровая переменная, которая представляет собой регистр на аппаратном уровне и используется для хранения промежуточных данных, полученных в процессе вычислений. Единственным ограничением для регистровой переменной в теле кадра является правило однократного присваивания.

Трансляция программы на языке высокого уровня COLAMO состоит в создании схемотехнической конфигурации вычислительной системы (структурной составляющей) и параллельной программы, управляющей потоками данных (потоковой и процедурной составляющих) [1, 8, 9]. Создание структурной составляющей заключается в построении вычислительного графа, соответствующего описанным на COLAMO информационным зависимостям между результатами вычислений.

При этом для каждой используемой в тексте программы операции подставляется специализированный вычислительный блок в зависимости от типа доступа к переменным, типа данных, их разрядности и т.д. Синтезированный информационный граф задачи передается в синтезатор Fire!Constructor для отображения на аппаратный ресурс многокристальной РВС [10].

Задача автоматического отображения параллельной программы на аппаратный ресурс многокристальной РВС состоит из трех подзадач: разбиения информационного графа на непересекающиеся подграфы, размещения сформированных подграфов в ПЛИС РВС и трассировки внешних связей размещённых подграфов в системе коммутаций РВС.

Результатом работы синтезатора Fire!Constructor являются файлы VHDL-описаний и файлы временных и топографических ограничений (User Constraints Files). Файлы VHDL описывают структурные реализации фрагментов параллельной программы. На основании этих файлов и библиотеки схемотехнических элементов формируются проекты для синтезатора ISE под каждую отдельную ПЛИС. Далее с помощью синтезатора ISE формируются конфигурационные файлы ПЛИС. Полученные конфигурационные файлы загружаются в РВС.

Программа на языке COLAMO разрабатывается в едином проекте и может быть

транслирована на любую РВС, описание которой и соответствующие библиотеки присутствуют в комплексе программирования РВС. В отличие от других существующих систем разработки решений прикладных задач на ПЛИС пользователю не требуется в тексте программы указывать, какие фрагменты программы в каких ПЛИС будут выполняться. Синтезатор Fire!Constructor обеспечивает автоматическое разбиение вычислительной структуры программы на языке COLAMO на несколько проектов в синтезаторе Xilinx ISE, при этом обеспечивается синхронизация информационных потоков как внутри кристалла ПЛИС, так и между ними.

Заключение

В данной статье рассматриваются ПЛИС в качестве элементной базы реконфигурируемых суперЭВМ. Согласно данным таблицы 3, ПЛИС обеспечивают устойчивый, близкий к линейному, рост производительности РВС, открывая новые перспективы по созданию суперкомпьютеров петафлопсной производительности. Можно утверждать, что конструктивные решения, положенные в основу перспективных вычислительных модулей на основе ПЛИС Xilinx Virtex UltraScale, в том числе на основе жидкостного охлаждения, позволят сосредоточить в пределах одной вычислительной стойки высотой 47U мощный вычислительный ресурс и обеспечить удельную производительность РВС на основе ПЛИС Xilinx Virtex UltraScale на уровне лучших мировых показателей для суперЭВМ с кластерной архитектурой.

Это позволяет рассматривать РВС на основе ПЛИС Xilinx Virtex UltraScale как основу для создания высокопроизводительных вычислительных комплексов нового поколения, обеспечивающих высокую эффективность вычислений и близкий к линейному рост производительности при наращивании вычислительного ресурса.

Работа выполнена при финансовой поддержке Министерства образования и науки РФ по Соглашению о предоставлении субсидии № 14.578.21.0006 от 05.06.2014, уникальный идентификатор RFMEFI57814X0006, гранту Южного федерального университета № 213.01-2014/014 и НИР №2257 базовой части государственного задания № 2014/174.

Литература

1. Каляев, И.А. Реконфигурируемые мультиконвейерные вычислительные структуры / И.А. Каляев, И.И. Левин, Е.А. Семерников, В.И. Шмойлов; под общ. ред. И.А. Каляева / Изд. 2-е, перераб. и доп. /Ростов-на-Дону: Изд-во ЮНЦ РАН, 2009. — 344 с.
2. Левин, И.И. Реконфигурируемые вычислительные системы с открытой масштабируемой архитектурой / И.И. Левин // Параллельные вычисления и задачи управления: Труды Пятой Международной конференции PACO'2010 (Москва, 26 октября – 28 октября 2010 г.). — М.: Учреждение Российской академии наук Институт проблем управления им. В.А. Трапезникова РАН, 2010. — С. 83–95.
3. Зотов, В.Ю. Проектирование цифровых устройств на основе ПЛИС фирмы XILINX в САПР WebPACK ISE / В.Ю. Зотов — М.: Горячая линия–Телеком. 2003. — 624 с.

4. Quartus II Handbook Version 10.1 Volume 1: Design and Synthesis. Altera Corporation 2010.
5. Libero IDE v9.1 User's Guide. Actel Corporation 2010.
6. Тарасов, И. Проектирование для ПЛИС Xilinx с применением языков высокого уровня в среде Vivado HLS / И. Тарасов // Компоненты и технологии, 2013. — № 12. — С. 33–36.
7. http://www.mitrionics.com/ (дата обращения 07.04.2015)
8. Kalyaev, I.A. Reconfigurable Computer Systems Based on Virtex-6 and Virtex-7 FPGAs / I.A. Kalyaev, I.I. Levin, A.I. Dordopulo, L.M. Slasten // IFAC Proceedings Volumes, Programmable Devices and Embedded Systems. — Vol. 12, Part № 1, 2013. — P. 210–214. DOI: 10.3182/20130925-3-cz-3023.00009.
9. Kalyaev, Igor A. FPGA-based Reconfigurable Computer Systems / Igor A. Kalyaev, Ilya I. Levin, Alexey I. Dordopulo, Liuba M. Slasten // Science and Information Conference (SAI) (London, UK, 7 Oct – 9 Oct 2013). — P. 148–155.
10. Gudkov, V.A. Multi-level Programming of FPGA-based Computer Systems with Reconfigurable Macroobject Architecture / V.A. Gudkov, A.A. Gulenok, V.B. Kovalenko, L.M. Slasten // IFAC Proceedings Volumes, Programmable Devices and Embedded Systems. — 2013. — Vol. 12, No. 1. — P. 204–209. DOI: 10.3182/20130925-3-cz-3023.00008.

Левин Илья Израилевич, доктор технических наук, профессор, директор, ООО «Научно-исследовательский центр супер-ЭВМ и нейрокомпьютеров» (Таганрог, Российская Федерация), levin@superevm.ru.

Дордопуло Алексей Игоревич, кандидат технических наук, старший научный сотрудник, Южный научный центр РАН (Ростов-на-Дону, Российская Федерация) scorpio@mvs.tsure.ru.

Каляев И.А., Научно-исследовательский институт многопроцессорных вычислительных систем, Южный федеральный университет (Таганрог, Российская Федерация), kaliaev@mvs.tsure.ru.

Доронченко Ю.И., технический директор, ООО «Научно-исследовательский центр супер-ЭВМ и нейрокомпьютеров» (Таганрог, Российская Федерация), doronchenko@superevm.ru.

Раскладкин М.К., начальник отдела, ООО «Научно-исследовательский центр супер-ЭВМ и нейрокомпьютеров» (Таганрог, Российская Федерация), raskladkin@mail.ru.

Поступила в редакцию 23 апреля 2015 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2015, vol. 4, no. 3, pp. 24–39

DOI: 10.14529/cmse150303

MODERN AND NEXT-GENERATION HIGH-PERFORMANCE COMPUTER SYSTEMS WITH RECONFIGURABLE ARCHITECTURE

I.I. Levin, Scientific Research Centre of Supercomputers and Neurocomputers (Taganrog, Russian Federation) levin@superevm.ru,
A.I. Dordopulo, Southern Scientific Centre of the RAS (Rostov-on-Don, Russian Federation) scorpio@mvs.tsure.ru,
I.A. Kaliaev, Academician A.V. Kalyaev SRI multiprocessor computer system at Southern Federal University (Taganrog, Russian Federation) kaliaev@mvs.tsure.ru,
Y.I. Doronchenko, Scientific Research Centre of Supercomputers and Neurocomputers (Taganrog, Russian Federation) doronchenko@superevm.ru,
M.K. Raskladkin, Scientific Research Centre of Supercomputers and Neurocomputers (Taganrog, Russian Federation) raskladkin@mail.ru

The paper covers the architecture and comparative specifications of reconfigurable computer systems (RCS) based on FPGAs of Xilinx Virtex-7 family: the computational modules 24V7-750 and Taygeta, which can be placed into a computational rack, and the reconfigurable desktop computational block Celaeno. Besides, the paper covers the architecture and assembly of the next-generation RCS based on liquid cooling, the results of prototyping of principal engineering solutions. The technology of implementation of applied tasks using the application development suit is considered. The next-generation RCS based on liquid cooling provide the performance up to 1 PFlops in a standard 47U computational rack with power consumption of 150 kWatts. It has a significant advantage of such technical and economical parameters as real and specific performance, power efficiency, mass and dimension parameters, etc. in comparison with other similar systems.

Keywords: reconfigurable computer systems, FPGAs, computational module, application software.

References

1. Kalyaev I.A., Levin I.I., Semernikov E.A., Shmoilov V.I. Rekonfiguriruyemiye multikonveyerniye vichislitelniye struktury [Reconfigurable multipipeline computing structures]/ Edited by Kalyaeva I.A. Rostov-on-Don, YUNTS RAN, 2009. 344 P.
2. Levin I.I. Rekonfiguriruyemiye vichislitelniye sistemy s otkrytoi masshtabiruyemoi arkhitekturoi [Reconfigurable computer systems with open scalable architecture] Parallelnuye vichisleniya i zadachi upravleniya (PACO'2010): Trudy V Mezhdunarodnoi konferentsii (Moskva, 26 oktyabrya – 28 oktyabrya 2010) [Parallel Computing and Control Problems: Proceedings of the Fifth International Conference (Moscow, Russia | October

26—28, 2010)]. M.: Uchrezhdeniye Rossiiskoi akademii nauk Institut problem upravleniya imeni V.A. Trapeznikova RAN, 2010. P. 83–95.

3. Zotov V.Y. Proijektirovanije tsifrovykh ustroistv na osnove PLIS firmy XILINX v SAPR WebPACK ISE [Design of digital devices based on Xilinx FPGAs using WebPACK ISE]. M.: Goryachaya liniya-Telekom, 2003. 624 P.
4. Quartus II Handbook Version 10.1 Volume 1: Design and Synthesis. Altera Corporation 2010.
5. Libero IDE v9.1 User's Guide. Actel Corporation 2010. 633 P.
6. Tarasov I. Proijektirovanije dlya PLIS Xilinx s primenenijem yazikov visokogo urovnya v srede Vivado HLS [Design for Xilinx FPGAs using high level languages and Vivado HLS]. Komponenty i tekhnologii [Components and technologies]. 2013. Vol. 12. P. 33–36.
7. http://www.mitrionics.com/(accessed: 07.04.2015)
8. Kalyaev I.A., Levin I.I., Dordopulo A.I., Slasten L.M. Reconfigurable Computer Systems Based on Virtex-6 and Virtex-7 FPGAs // Programmable Devices and Embedded Systems. IFAC Proceedings Volumes, Vol. 12, Part № 1. 2013. P. 210–214. DOI: 10.3182/20130925-3-cz-3023.00009.
9. Kalyaev I.A., Levin I.I., Dordopulo A.I., Slasten L.M. FPGA-based Reconfigurable Computer Systems // Science and Information Conference (SAI) (London, UK, Oct, 7 – 9, 2013). P. 148–155.
10. Gudkov V.A., Gulenok A.A., Kovalenko V.B., Slasten L.M. Multi-level Programming of FPGA-based Computer Systems with Reconfigurable Macroobject Architecture // Programmable Devices and Embedded Systems. IFAC Proceedings Volumes, 2013. Vol. 12, No. 1. P. 204–209. (ISSN 14746670). DOI: 10.3182/20130925-3-cz-3023.00008.

Received April 23, 2015.

УДК 004.72, 004.94 DOI: 10.14529/cmse150304

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ ПОДСЕТИ КОЛЛЕКТИВНЫХ ОПЕРАЦИЙ СЕТИ «АНГАРА»[1]

А.В. Мукосей, А.С. Семенов, А.С. Симонов

В ОАО «НИЦЭВТ» разрабатывается высокоскоростная коммуникационная сеть «Ангара» с топологией «многомерный тор». Для исследования и оценки производительности разрабатываемой сети при большом количестве используемых узлов создана параллельная потактовая имитационная модель сети. Сеть «Ангара» имеет аппаратную поддержку двух коллективных операций — broadcast и reduce. В статье описана реализация коллективных операций в имитационной модели, и представлены результаты оценки их производительности при помощи модели. Оценки производительности получены на базовых тестах broadcast и reduce, а также на прикладных задачах — умножение разреженной матрицы на вектор и численное решение нелинейного уравнения теплопроводности.

Ключевые слова: имитационное моделирование, «Ангара», многомерный тор, коммуникационная сеть, коллективные операции.

Введение

В настоящее время суперкомпьютеры содержат сотни тысяч вычислительных ядер. Эффективность одновременной работы ядер на задачах с интенсивным обменом данными между ними (задачи моделирования, задачи на графах и нерегулярных сетках, вычисления с использованием разреженных матриц) в основном определяется производительностью коммуникационной сети, соединяющей вычислительные узлы высокоскоростными каналами связи (линками).

В ОАО «НИЦЭВТ» разрабатывается высокоскоростная коммуникационная сеть «Ангара» с топологией «многомерный тор» [1–6]. В 2013 году выпущен кристалл маршрутизатора этой сети [7], на его основе в 2015 году ожидается построение суперкомпьютера.

Для некоторых прикладных задач требуется эффективное выполнение коллективных коммуникационных операций, в которых задействовано сразу много вычислительных узлов. В сети «Ангара» реализована аппаратная поддержка двух коллективных операций — broadcast и reduce [8]. Для этого добавлена виртуальная подсеть, состоящая из двух виртуальных каналов с особыми правилами маршрутизации. Виртуальная подсеть имеет топологию дерева, наложенную на «многомерный тор».

Для оценки производительности на тестовых программах и для исследования новых архитектур на языке Charm++ создана параллельная потактовая имитационная модель разрабатываемой коммуникационной сети [4]. Однако поддержка коллективных операций в модели отсутствовала.

Имитационное моделирование позволяет исследовать характеристики сетей и суперкомпьютеров, состоящих из большого числа узлов, что особенно важно для коллективных операций, эффективная реализация которых начинает проявляться при большом числе используемых узлов. Имитационное моделирование большого числа узлов важно из экономических соображений, так как большой суперкомпьютер-макет построить дорого по экономическим соображениям.

[1] Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии – 2015».

Целью данной работы является реализация подсети коллективных операций в имитационной модели, а также демонстрация производительности подсети коллективных операций при помощи разработанной модели. Статья организована следующим образом. Во втором разделе описывается архитектура поддержки коллективных операций в коммуникационной сети. Третий раздел посвящен реализации поддержки коллективных операций в потактовой имитационной модели сети. В четвертом разделе описываются оценки производительности коллективных операций, полученные при помощи имитационной модели сети. В заключении перечисляются основные результаты работы и планы дальнейших исследований.

1. Коллективные операции в маршрутизаторе коммуникационной сети «Ангара»

Коллективные операции используются для обмена данными между несколькими узлами системы. Их относят к основным примитивам взаимодействия вычислительных процессов в большинстве стандартов параллельного программирования, ориентированных на выполнение на системах с распределенной памятью (MPI [9], Shmem [10], PGAS-языки — UPC [11], X10 [12]); они могут составлять значительную часть коммуникационных обменов в процессе работы [13]. Реализация коллективных операций с использованием операций типа «точка-точка» имеет ряд недостатков, таких как большая доля дублирующего трафика, плохая масштабируемость [14], поэтому их аппаратная поддержка способствует повышению масштабируемости параллельной программы (см., например, [15]).

Высокоскоростная коммуникационная сеть «Ангара» с топологией «многомерный тор» поддерживает детерминированную и адаптивную передачу пакетов, неблокирующие записи, чтения, атомарные операции, отказоустойчивость на канальном уровне и обход отказавших каналов и узлов. В рамках данной сети реализована аппаратная поддержка двух коллективных операций — broadcast и reduce [8]. Для этого добавлена виртуальная подсеть, состоящая из двух виртуальных каналов с отдельными буферами и специальными правилами маршрутизации. Виртуальная подсеть имеет топологию дерева (рис. 1), построенного в торе. Выбирается корневой узел, от которого строится дерево с учетом порядка измерений: X, Y, Z, W (это позволяет предотвратить возможные дедлоки). В построенном дереве существует два направления движения: от корня и к корню. Каждому направлению соответствует свой виртуальный канал. В системе могут быть транзитные узлы, в них процессоры не посылают и не получают данных.

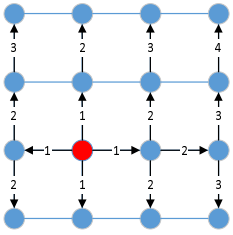


Рис. 1. Виртуальная подсеть коллективных операций на примере 2D-решетки. Стрелками обозначено направление движения от корня, цифрами обозначены этапы обхода дерева

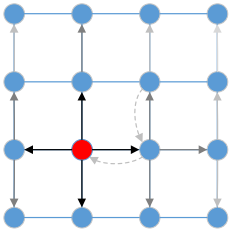


Рис. 2. Схема выполнения операции broadcast не из корневого узла на примере 2D-решетки. Пунктирной линией выделен путь до корня. Сплошной — распространение от корня

При выполнении операции broadcast каждый узел при получении пакета от узла, находящегося выше по дереву, рассылает его всем узлам, находящимся ниже по дереву (см. рис. 1). При инжекции пакета в сеть не в корневом узле сначала генерируется запрос на broadcast, который посылается корню (рис. 2).

При выполнении операции reduce (или варианта all reduce) узел ожидает пакеты от хоста, если узел не транзитный, и всех узлов, находящихся ниже по дереву; выполняет над ними указанную в пакете коммутативную ассоциативную бинарную операцию и отправляет готовый результат вверх к корню. В текущей реализации поддерживаются операции максимума, минимума и суммы целых чисел. Операция reduce в корне завершается аппаратной отправкой (без эжекции) результата заданному узлу посредством операции «точка-точка» (broadcast для all reduce).

Для определения направления к корню и от корня на каждом узле задается таблица маршрутизации коллективной подсети, при этом указываются следующие поля: направления на узлы ниже и выше по дереву; является ли узел транзитным или корневым. Для задания корректного дерева должны выполняться следующие критерии:

- корень ровно один;
- если в каком-то узле выставлено направление вниз по дереву, то в этом направлении должен находиться принадлежащий дереву узел, в котором направление вверх по дереву выставлено противоположным данному;
- направления на узлы ниже по дереву могут быть только: 1) по измерениям, следующим за направлением вверх по дереву в рамках порядка направлений, 2) по направлению, противоположному направлению вверх по дереву.

В рамках сети можно задавать различные пересекающиеся деревья. Поддерживается 16 деревьев, каждому соответствует свой идентификатор TreeId, по которому производится выборка из таблицы маршрутизации при принятии решения по маршрутизации пакета. Одновременно маршрутизатор поддерживает до 16 различных пакетов reduce по каждому TreeId. Каждый reduce, выполняющийся по данному дереву, имеет свой идентификатор ReduceId (от 0 до 15).

С точки зрения прикладного программиста, базовые версии коллективных операций — односторонние асинхронные операции. Процессор не блокируется после отправки сообщения, а результат записывается в память без активного участия принимающей стороны, это позволяет совмещать ожидание окончания операции со счетом. Для того, чтобы узнать, что коллективная операция завершилась и результат доступен вычислительным узлам, существуют механизмы синхронизации, основанные на коллективных операциях.

2. Реализация коллективных операций в параллельной имитационной модели

Для оценки производительности и исследования новых архитектур высокоскоростной коммуникационной сети разработана и используется параллельная потактовая имитационная модель [4]. Модель разработана на языке Charm++ и позволяет моделировать на вычислительном кластере конфигурации с большим количеством моделируемых узлов сети. Модель маршрутизатора устроена достаточно гибко и имеет большое количество конфигурационных параметров. Это позволяет подстраивать модель для различных типов сетей.

В модели реализованы следующие топологии: тор произвольной размерности, сеть Кэли, сеть Клоса.

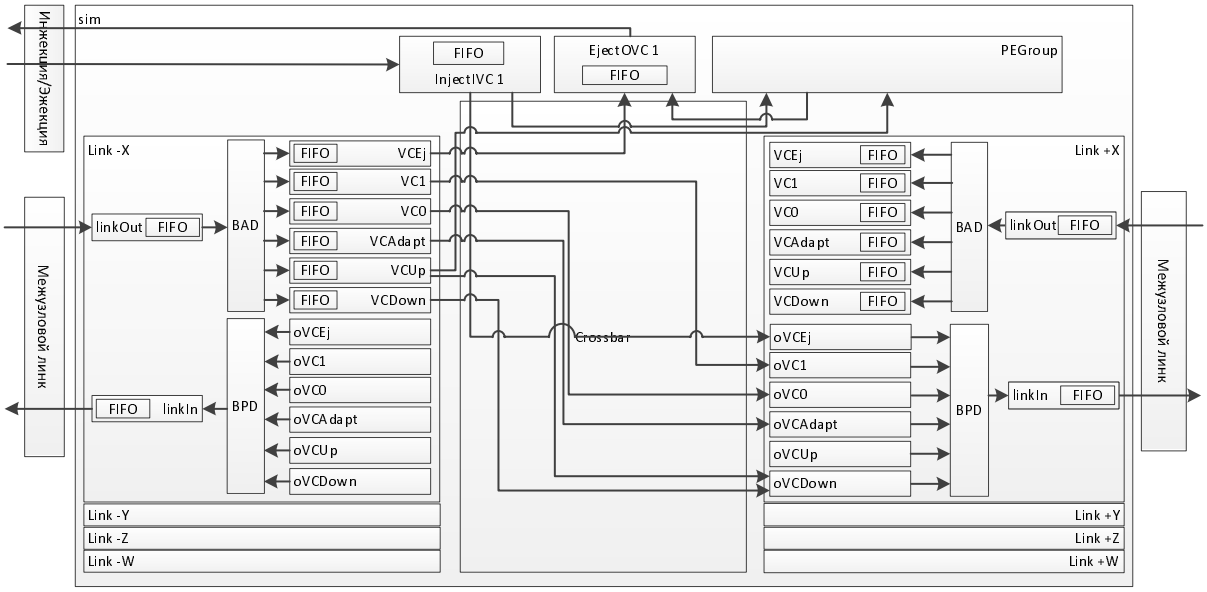


Рис. 3. Общая схема маршрутизатора коммуникационной сети с топологией «многомерный тор», реализованная в имитационной модели сети

На рис. 3 представлена общая схема маршрутизатора рассматриваемой сети, реализованная в имитационной модели.

Маршрутизатор имеет два типа входов: межузловые и инжекционные. Аналогично, имеется два типа выходов: межузловые и эжекционные. Межузловые входы (выходы) соединяются с выходами (входами) других узлов соответственно, посредством физических каналов, так называемых межузловых линков. Считается, что помехи отсутствуют. Инжекционные (эжекционные) каналы, так называемые процессорные, служат для связи процессора с маршрутизатором. Каждый процессор соединяется с маршрутизатором посредством одного или нескольких инжекционных и такого же количества эжекционных каналов. Каналы представляют собой FIFO-буфера, доступные только для чтения или только для записи. Минимальная длина передаваемых данных в сети — 128 бит (флит данных).

Для поддержки виртуальных подсетей в маршрутизаторе предусмотрены виртуальные каналы (VC). Виртуальные каналы представляют собой FIFO-буфера с блоком маршрутизации. Каждый пакет из линка через блок анализа данных BAD попадает в заданный типом пакета FIFO-буфер.

Для реализации подсети коллективных операций добавлены два виртуальных канала: для движения к корню (VCUp) и для движения к листьям (VCDown), а также блок PEGroup для эжекции, инжекции и обработки коллективных пакетов.

2.1. Виртуальные каналы VCUp и VCDown

За основу виртуальных каналов VCUp и VCDown взят детерминированный виртуальный канал (VCDet). При этом изменена маршрутизация, а также добавлен механизм выдачи копий пакета. Маршрутизация осуществляется по данным из таблицы маршрутизации и по заголовку пакета. Таблица маршрутизации заполняется на этапе инициализации модели. В таблице 16 строк, по строке на каждое дерево. У строк есть следующие поля: $TreeId$

— номер дерева подсети коллективных операций, $isRoot$ — определяет является ли узел корнем, $toRoot$ — направление вверх к корню, Pe — участвует ли узел в коллективной операции или он транзитный, $Dirs$ — направления вниз по дереву, Dir_sum — количество направлений (нужно для контроля выданных пакетов). Процесс прохождения пакета по виртуальным каналам коллективных операций состоит из следующих этапов (рис. 4):

- ожидание приема всего пакета в буфер виртуального канала;
- чтение таблицы маршрутизации, анализ головного флита пакета;
- в зависимости от виртуального канала и полученной выше информации составляется список направлений на передачу: детям, в вычислительный узел или к корню.

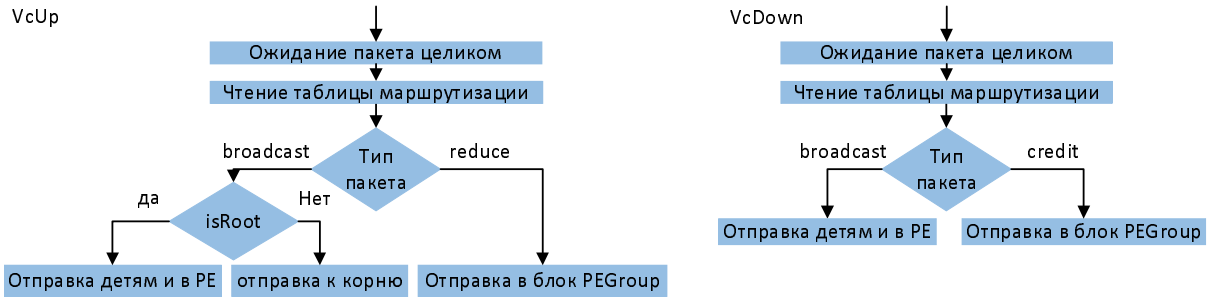


Рис. 4. Логика работы виртуальных каналов VCUp и VCDown

2.2. Блок PEGroup

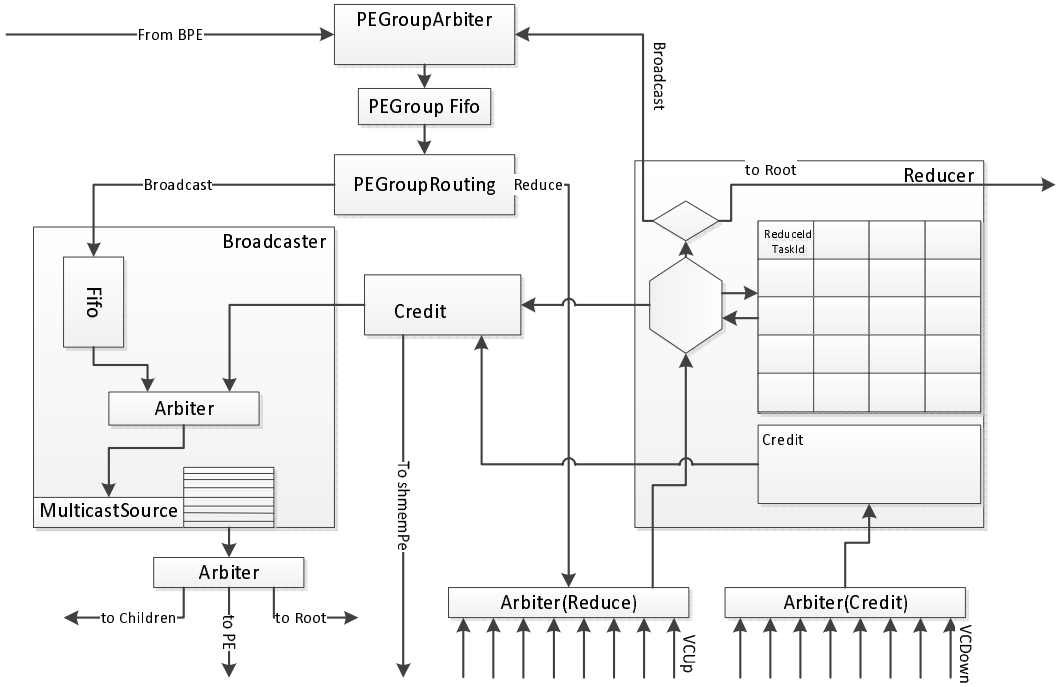


Рис. 5. Общая схема блока PEGroup

Блок PEGroup (рис. 5) предназначен для приема/передачи пакетов типа broadcast, reduce-пакетов и кредитных пакетов. Он разделен на три основные части: Broadcaster — блок рассылки broadcast пакетов, Reducer — блок для пакетов reduce и блок анализа кредитных пакетов. Инжектированные пакеты коллективных операций от процессора сначала попадают в очередь PEGroup Fifo блока PEGroup. После чего блок маршрутизации, ана-

лизируя пакеты, отправляет их в соответствующий блок. Reduce- и кредитные пакеты, пришедшие из виртуальных каналов, попадают в блок Reducer.

2.3. Блок Reducer

Блок Reducer принимает пакеты типа reduce. Над данными из пакета типа reduce блок Reducer производит заданную операцию (определяется по данным из головного флита пакета), сохраняет значения у себя в памяти для дальнейших операций или отправляет дальше по сети. Для каждого TreeId и ReduceId в блоке Reducer имеется изначально равный нулю счетчик принятых пакетов. Когда в Reducer из виртуального канала приходит пакет reduce, то его процесс прохождения по блоку Reducer будет следующим:

- Из заголовочного флита пакета считываются TreeId и ReduceId.
- Увеличивается на 1 значение счетчика принятых пакетов по TreeId и ReduceId.
- Считываются значения из памяти по TreeId и ReduceId (если это первый пакет, то ничего не происходит).
- Производится заданная арифметическая операция над каждым флитом данных из вновь пришедшего пакета и соответствующим значением из памяти (если это первый пакет, то ничего не происходит).
- Полученные значения вновь записываются в память по TreeId и ReduceId (если это первый пакет, то в качестве значений берутся флиты данных пакета).
- Если пришедший пакет был последним, то полученный результат выдается в блок маршрутизации. В кредитном блоке увеличивается на 1 значение счетчика выданных результирующих пакетов reduce по TreeId.

Если узел не был корневым, то маршрутизатор выставит запрос к корню вверх по дереву. Если узел корневой, то будет выставлен запрос на передачу в очередь PEGroup Fifo, для рассылки результата операции редукции (reduce убрать).

2.4. Блок анализа кредитных пакетов

Блок анализа кредитных пакетов создан для контроля количества одновременно обрабатываемых reduce пакетов в блоке Reducer; допустимо 16 операций по одному дереву. Блок отправляет кредитный пакет в PE и всем своим детям. Отправка детям происходит каждый раз, когда из узла было отправлено вверх по дереву или в процессор 8 пакетов с результатом редукции для каждого фиксированного дерева. Отправка в PE осуществляется каждые 128 отправленных пакетов с результатом редукции по всем TreeId или каждые 4096 тактов. В процессор отправляются значения всех счетчиков выполненных операций редукции по всем TreeId. Также блок анализирует приходящие кредитные пакеты. Каждый такой пакет означает, что узел-родитель может принять еще 8 пакетов по данному TreeId. Кредитный блок осуществляет передачу пакетов через блок Broadcaster.

2.5. Блок Broadcaster

Блок Broadcaster берет поочередно пакеты из очереди PEGroup Fifo и кредитного блока, анализирует заголовки пакетов и таблицу маршрутизации и принимает решение о маршрутизации: вниз по дереву, вверх к корню или на инжекцию в PE.

В модели реализован механизм построения временных диаграмм прохождения пакета по маршрутизатору. Для каждого реализованного блока с помощью этого механизма

построены временные диаграммы. Все диаграммы соответствую временным диаграммам аппаратной реализации. Пример диаграммы прохождения пакета по виртуальному каналу VCUp приведен на рис. 6.

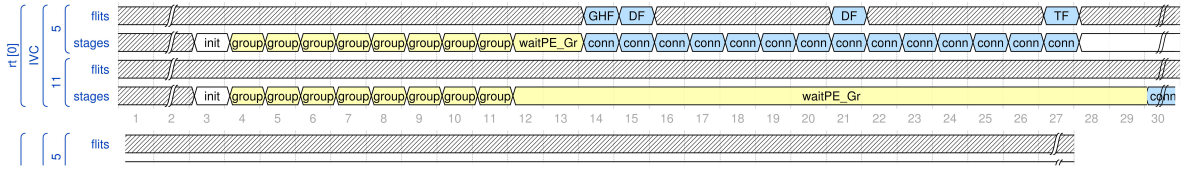


Рис. 6. Временная диаграмма прохождения пакета по каналу VCUp

3. Оценка производительности коллективных операций на базовых тестах и прикладных задачах

Оценка производительности коллективных операций проводилась в два этапа. На первом этапе исследовались базовые операции broadcast и all reduce, имеющие непосредственную аппаратную поддержку в подсети коллективных операций. Задача второго этапа — демонстрация производительности небольших прикладных тестов-задач. Выбраны следующие задачи:умножение разреженной матрицы на вектор, численное решение задачи с нелинейным уравнением теплопроводности.

Оценки производительности были получены при помощи имитационной модели сети, которая описана в третьем разделе. Параметры модели сети соответствуют значениям параметрам СБИС маршрутизатора сети «Ангара», выпущенного в 2013 году и приведены в таблице. Необходимые вычисления проводились на процессоре, параметры которого также приведены в таблице.

Таблица

Параметры оцениваемого суперкомпьютера

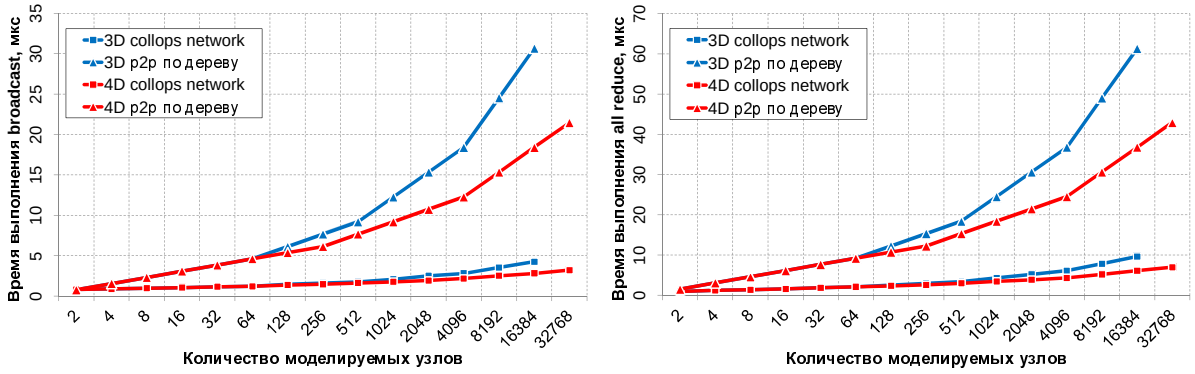
Параметр	**Значение**
Процессор	Intel Xeon E5-2660
Тактовая частота процессора, ГГц	2.2
Количество ядер в узле	8
Размер кэша L3, МБ	20
Пиковая производительность узла, Гфлопс	140.8
Тактовая частота маршрутизатора, МГц	500
Интерфейс процессора с сетью	PCIe gen2 x16
Задержка на инжекцию (эжекцию) пакета через PCIe, нс	300
Задержка передачи по линку, нс	80
Топология сети	3D/4D-тор

3.1. Базовые тесты broadcast и all reduce

На этапе тестирования базовых операций оценивалась производительность базовых операций broadcast и all reduce. Тесты заключались в посылке одного пакета максимального размера в 16 флитов (256 байт) при помощи операций broadcast и all reduce для разного количества узлов моделируемой сети.

На рис. 7 представлены результаты выполнения операций broadcast и all reduce. Синим и красным показано время выполнения (в мкс) коллективной операции для сети с топологией 3D- и 4D-тор соответственно. Квадратными маркерами отмечено время выполнения коллективной операции, реализованной с помощью сети коллективных операций, треугольными маркерами — при помощи коммуникационных операций «точка-точка» по тому же дереву, что и в подсети коллективных операций, но полученное аналитически.

Для операции broadcast в сети с топологией 3D-тор выигрыш для 8 узлов (тор 2x2x2) составляет 2,39 раз, а для 8096 узлов (тор 16x16x32) — 6,97 раз. Для топологии 4D-тор выигрыш для 8096 узлов (тор 8x8x8x16) составляет 6,18 раз. Превосходство аппаратной поддержки коллективных операций обусловлено отсутствием накладных расходов на эжекцию/инжекцию пакетов. Сети с топологией 4D-тор обладают меньшим диаметром по сравнению с сетями 3D-тор при одинаковом количестве узлов, поэтому для сетей с топологией 4D-тор выигрыш меньше. Для операции all reduce разница в производительности примерно такая же. Таким образом, реализация операций с помощью коллективной подсети дает значительный выигрыш по сравнению с использованием операций «точка-точка».



а) Операция broadcast б) Операция all reduce

Рис. 7. Время выполнения операций (в микросекундах) а broadcast и б all reduce

3.2. Умножение разреженной матрицы на вектор

Операция умножения разреженной матрицы на вектор лежит в основе метода сопряженных градиентов CG. CG — метод нахождения локального минимума функции на основе информации о ее значениях и градиенте. Один из способов распараллеливания этой операции заключается в следующем [16]: пусть дана разреженная матрица A и вектор x, требуется найти $y = Ax$. Строки матрицы блоками равномерно распределяются по узлам:

$$A = (A_1, A_2, \ldots, A_{np})^T \Rightarrow Ax = (A_1x, A_2x, \ldots, A_{np}x)^T, \quad (1)$$

где A_i — строки матрицы A, хранящиеся на i-ом узле. При умножении строк матрицы на вектор получаются результирующие части искомого вектора y: $y_j = A_jx$, где $j = 1, 2, \ldots, np$, np — количество вычислительных узлов. Для сборки результирующего век-

тора y на каждом из вычислительных узлов для последующих вычислений понадобится коллективная операция all gather.

Для реализации all gather через операцию «точка-точка» используется алгоритмом «рекурсивное удвоение» [17]. Алгоритм заключается в следующем: сначала все узлы попарно обмениваются друг с другом, затем пары обмениваются попарно таким образом, что каждый узел пары обменивается данными с другим узлом пары и т.д. Схема работы алгоритма изображена на рис. 8.

Для реализации all gather через коллективные операции используется broadcast — все узлы рассылают свою часть вектора, инициируя операцию broadcast. Используются все 16 деревьев, корни которых равномерно распределены по системе.

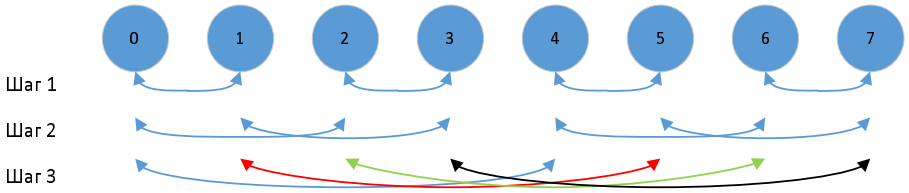


Рис. 8. Схема работы алгоритма «рекурсивное удвоение» для системы из 8 узлов

Для расчетов выбрана матрица с $d = 10$ ненулевыми элементами в строке и размером $N = 20480000$. Измерение времени умножения строк матрицы на плотный вектор проводилось на реальном процессоре Intel Xeon E5-2660 с использованием тестовой программы на C/OpenMP отдельно для каждого количества строк матрицы, попадающих на вычислительный узел при увеличении количества узлов сети и дроблении задачи. Производительность на одном узле составляет 591 Мфлопс. Время выполнения операции all gather получено с помощью имитационной модели сети.

Оценка точности имитационного моделирования проводилась для операции all gather, реализованной при помощи операций «точка-точка». Время выполнения all gather, полученное при помощи имитационной модели, сравнивалось со временем выполнения этой операции на тестовом кластере, параметры которого приведены в таблице, для 2, 4 и 8 узлов. Максимальная разница времен выполнения составляет 8,96 %.

На рис. 9 видно, что использование аппаратных коллективных операций дало максимальный прирост производительности 28 % на 2048 узлах для сети 4D-тор по сравнению с использованием коллективных операций, реализованных с помощью операций «точка-точка».

3.3. Задача с нелинейным уравнением теплопроводности

Рассматривается двухмерная нелинейная задача теплопроводности:

$$\frac{\partial u}{\partial t} = \sigma(x_1, x_2, t)\left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}\right) + f(x_1, x_2, t) \tag{2}$$

$$t\epsilon[0, t_k], x_1\epsilon[a, b], x_2\epsilon[c, d], \forall x_1, x_2, t : \sigma \geqslant 0 \tag{3}$$

Явная разностная схема:

$$\frac{U_{j,k}^{n+1} - U_{j,k}^n}{\Delta t} = \sigma_{j,k}^n \left(\frac{U_{j+1,k}^n - 2U_{j,k}^n + U_{j-1,k}^n}{h_j^2} + \frac{U_{j,k+1}^n - 2U_{j,k}^n + U_{j,k-1}^n}{h_k^2}\right) + f_{j,k}^h \tag{4}$$

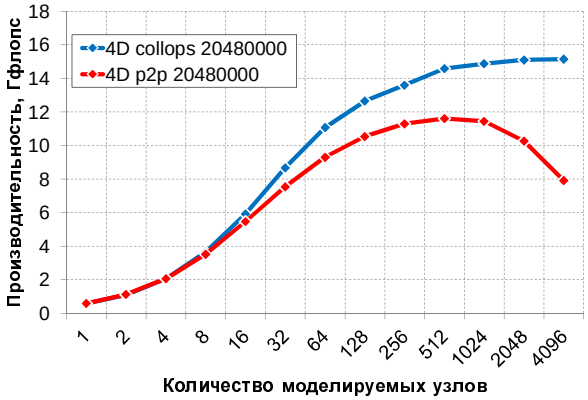


Рис. 9. Производительность задачи умножения разреженной матрицы на вектор ($N = 20480000$, $d = 10$) в зависимости от количества вычислительных узлов для сети с топологией 4D-тор

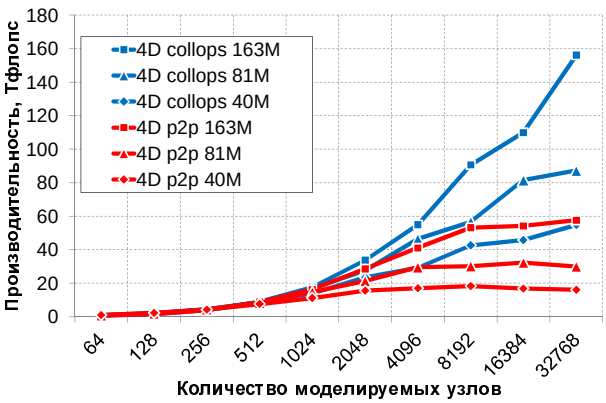


Рис. 10. Производительность двухмерной задачи теплопроводности на различных сетках в зависимости от количества вычислительных узлов

Условие устойчивости:

$$\Delta t \leqslant \frac{1}{2\sigma(x_1, x_2, t)\left(\frac{1}{h_1^2} + \frac{1}{h_2^2}\right)} \leqslant \frac{1}{2max[\sigma(x_1, x_2, t)]\left(\frac{1}{h_1^2} + \frac{1}{h_2^2}\right)} \tag{5}$$

Двухмерная область распределяется поровну по вычислительным узлам. В явной схеме для вычисления значения в ячейке на следующем слое по времени необходимы 9 значений ячеек с предыдущего слоя (см. рис. 11). Для этого на каждый узел требуется область больше на единицу в каждую сторону, чтобы была возможность рассчитать границы области. После каждого расчета области необходим обмен гранями между узлами и вычисление оптимального шага по времени. Для вычисления временного шага, требуется вычислить максимум коэффициента температуропроводности (усл. устойчивости 5). Вычисление максимума происходит при помощи коллективной операции all reduce. Общая схема параллельной реализации задачи представлена на рис. 12.

Производительность вычисляется по формуле: $\frac{N_{op}}{T_S + T_{max}} * 10^{-12}$ [Тфлопс], где N_{op} — количество операций с плавающей точкой. Рассматривались задачи с общим количеством ячеек, равным 40960000, 81920000 и 163840000 (на рис. 10 обозначены 40M, 81M и 163M соответственно). Время расчета области (T_S) измерялось на реальном процессоре Intel Xeon E5-2660 с использованием тестовой программы на C/OpenMP отдельно для каждого размера области, приходящейся на вычислительный узел при увеличении количества узлов сети и дроблении задачи. Производительность на одном узле составляет приблизительно 11 Гфлопс. Время вычисления максимума коэффициента температуропроводности (T_{max}) получено при помощи имитационного моделирования операции all reduce.

На данной задаче (см. рис. 10) аппаратная поддержка коллективных операций в сети 4D-тор сказывается, когда узлов в сети более 64. На большом числе вычислительных узлов (32768) производительность варианта с использованием подсети коллективных операций превышает производительность варианта с реализацией на основе операций «точка-точка» в 3 раза для задачи с 40M ячеек в сетке, для остальных размеров задач выигрыш составляет 2,7 раз.

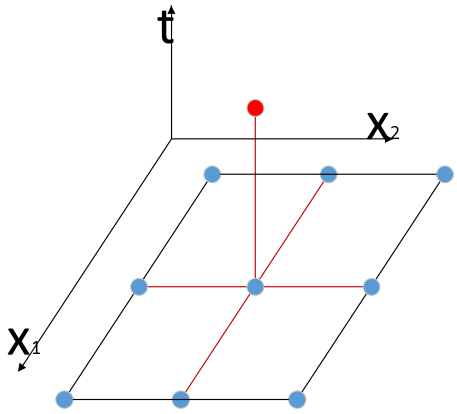


Рис. 11. Шаблон явной схемы двухмерной нелинейной задачи теплопроводности

Рис. 12. Схема параллельной реализации явной схемы для нелинейного уравнения теплопроводности.

Заключение

В данной работе описана реализация поддержки подсети коллективных операций в параллельной потактовой иммитационной модели сети «Ангара». С использованием имитационной модели получены демонстрационные оценки производительности базовых коллективных операций и небольших прикладных задач.

Оценки времени выполнения базовых коллективных операций reduce и broadcast показали значительный выигрыш (более 2-х раз для 8 узлов, более 6-ти раз для 8192 узлов) при использовании аппаратной подсети коллективных операций относительно программной реализации этих операций при помощи операции «точка-точка». С увеличением количества вычислительных узлов разница в производительности увеличивается.

Оценка производительности прикладных задач проводилась на имитационной модели на примере задачи умножения разреженной матрицы на вектор и задачи численного решения явной схемы нелинейного уравнения теплопроводности. Для задачи умножения матрицы на вектор выбраны следующие параметры: 20480000 — размер матрицы с 10 ненулевыми элементами в строке, задача теплопроводности рассматривалась на сетках с количеством ячеек 40960000, 81920000 и 163840000.

На задаче умножения разреженной матрицы на вектор, реализованной с помощью операции all gather, выигрыш от использования аппаратной поддержки коллективных операций по сравнению с реализацией на основе операции «точка-точка» небольшой, растет с увеличением количества вычислительных узлов и максимально составляет 28 % на 2048 узлах для сети с топологией 4D-тор.

На задаче численного решения явной схемы нелинейного уравнения теплопроводности на тысячах и десятках тысяч вычислительных узлов производительность варианта с использованием подсети коллективных операций превышает производительность варианта с реализацией на основе операций «точка-точка» от 2,7 до 3 раз. При увеличении количества вычислительных узлов разница в производительности растет.

Выполненная работа позволила получить оценки производительности тестов с использованием аппаратной поддержки коллективных операций в коммуникационной сети с то-

пологией «многомерный-тор» и показала возможность получения высокой производительности с использованием таких коллективных операций для выбранных задач. При этом выбраны конкретные размеры демонстрационных задач.

С использованием представленных в данной статье результатов планируется детальное исследование производительности коллективных операций с целью разработки новой архитектуры подсети коллективных операций для следующего поколения маршрутизатора сети «Ангара».

Литература

1. Макагон, Д.В. Сети для суперкомпьютеров / Д.В. Макагон, Е.Л. Сыромятников // Открытые системы. СУБД. — 2011. — № 7. — С. 33–37.
2. Корж, А.А. Отечественная коммуникационная сеть 3D-тор с поддержкой глобально адресуемой памяти для суперкомпьютеров транспетафлопсного уровня производительности / А.А. Корж, Д.В. Макагон, И.А. Жабин, Е.Л. Сыромятников // Паралельные вычислительные технологии (ПаВТ'2010): Труды международной научной конференции (Уфа, 29 марта – 2 апреля 2010 г.). — Челябинск: Издательский центр ЮУрГУ, 2010. — С. 227–237.
   URL: `http://omega.sp.susu.ac.ru/books/conference/PaVT2010/full/134.pdf` (дата обращения: 29.04.2015).
3. Симонов, А.С. Разработка межузловой коммуникационной сети с топологией «многомерный тор» и поддержкой глобально адресуемой памяти для перспективных отечественных суперкомпьютеров / А.С. Симонов, И.А. Жабин, Д.В. Макагон // Научно-техническая конференция «Перспективные направления развития вычислительной техники» (Москва, 28 июня). — Москва: ОАО «Концерн «Вега», 2011. — С. 17–19.
4. Эйсымонт, Л.К. Моделирование российского суперкомпьютера «Ангара» на суперкомпьютере / Л.К. Эйсымонт, А.С. Семенов, А.А. Соколов, А.С. Фролов, А.Б. Шворин // В сборнике «Суперкомпьютерные технологии в науке, образовании и промышленности» под редакцией академика В.А. Садовничего, академика Г.И. Савина, чл.-корр. РАН Вл.В. Воеводина. — Москва: Издательство Московского университета, 2009. — С. 145–150.
5. Симонов, А.С. Первое поколение высокоскоростной коммуникационной сети «Ангара» / А.С. Симонов, Д.В. Макагон, И.А. Жабин А.Н. Щербак, Е.Л. Сыромятников, Д.А. Поляков // Наукоемкие технологии. — 2014. — Т. 15, № 1. — С. 21–28.
6. Слуцкин, А.И. Разработка межузловой коммуникационной сети ЕС8430 «Ангара» для перспективных суперкомпьютеров / А.И. Слуцкин, А.С. Симонов, И.А. Жабин, Д.В. Макагон, Е.Л. Сыромятников // Успехи современной радиоэлектроники. — 2012. — № 1. — С. 6–10.
7. Жабин, И.А. Кристалл для Ангары / И.А. Жабин, Д.В. Макагон, А.С. Симонов // Суперкомпьютеры. — Зима-2013. — С. 46–49.
8. Макагон, Д.В. Реализация аппаратной поддержки коллективных операций в маршрутизаторе высокоскоростной коммуникационной сети с топологией «многомерный тор» / Д.В. Макагон, Е.Л. Сыромятников, С.И. Парута, А.А. Румянцев // Успехи современной радиоэлектроники. — 2012. — № 1. — С. 11–15.
9. Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, 1995.

URL: http://www.mpi-forum.org/docs/mpi-1.1/mpi-11-html/node64.html (дата обращения: 29.04.2015).

10. Feind, K. Shared Memory Access (SHMEM) Routines. Cray Research, 1995. / K. Feind. URL: https://cug.org/5-publications/proceedings_attendee_lists/1997CD/S95PROC/303_308.PDF (дата обращения: 29.04.2015).
11. Wiebel, F. UPC Collective Operations Specifications. — 2003. / E. Wiebel, D. Greenberg, S. Seidel. URL: http://upc.gwu.edu/docs/UPC_Coll_Spec_V1.0.pdf (дата обращения: 29.04.2015).
12. Saraswat, V. X10 Language Specification. — 2011. / V. Saraswat, B. Bloom, I. Peshansky, O. Tardieu, D. Grove. URL: http://dist.codehaus.org/x10/documentation/languagespec/x10-latest.pdf (дата обращения: 29.04.2015).
13. Fox, G. Solving Problems on Concurrent Processors / G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, D. Walker // General techniques and regular problems. — V. 1, — Prentice-Hall Inc., 1998. — P. 592.
14. Bala, V. CCL: a Portable and Tunable Collective Communication Library for Scalable Parallel Computers / V. Bala, J. Bruck, R. Cypher, P. Elustondo, H. Ching-Tien, S. Kipnis, M. Snir // Parallel and Distributed System — 1995. — V. 6, — P. 154–164. DOI: 10.1109/71.342126.
15. Almasi, G. Efficient Implementation of Allreduce on BlueGene/L Collective Network / G. Almasi, G. Dozsa, C. Erway, B. Steinmacher-Burow // Recent Advances in Parallel Virtual Machine and Message Passing Interface. — 2005. — P.57–66. DOI: 10.1007/11557265_12.
16. Пожилов, И.А. Прогнозирование масштабируемости задачи умножения разреженной матрицы на вектор при помощи модели коммуникационной сети / И.А. Пожилов, А.С. Семенов, Д.В. Макагон // Вестник УГАТУ. — 2012. — Т. 16, № 6 (51). — С. 158–163.
17. Thakur, R. Optimization of Collective Communication Operations in MPICH. / R. Thakur, R. Rabenseifner, W. Gropp. URL: http://www.mcs.anl.gov/~thakur/papers/ijhpca-coll.pdf (дата обращения: 29.04.2015).

Мукосей Анатолий Викторович, ОАО «НИЦЭВТ» (Москва, Российская Федерация), mukav@mail.ru.

Семенов Александр Сергеевич, к.т.н., ОАО«НИЦЭВТ» (Москва, Российская Федерация), alxdr.semenov@gmail.com.

Симонов Алексей Сергеевич к.т.н., с.н.с., ОАО«НИЦЭВТ» (Москва, Российская Федерация), simonov@nicevt.ru.

Поступила в редакцию 10 апреля 2015 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2015, vol. 4, no. 3, pp. 40–55

DOI: 10.14529/cmse150304

SIMULATION OF COLLECTIVE OPERATIONS HARDWARE SUPPORT FOR «ANGARA» INTERCONNECT

A.V. Mukosey, JSC «NICEVT» (Moscow, Russian Federation) mukav@mail.ru,
A.S. Semenov, JSC «NICEVT» (Moscow, Russian Federation)
alxdr.semenov@gmail.com,
A.S. Simonov, JSC «NICEVT» (Moscow, Russian Federation) simonov@nicevt.ru

JSC NICEVT develops the Angara high-speed interconnect with multi-dimensional torus topology. To evaluate the performance of the interconnect on a large number of nodes a cycle-accurate simulator is used. Angara interconnect supports two types of collective operations: broadcast and reduce. The paper describes the implementation of collective operations in the simulator and presents an early performance evaluation. Performance benchmarks include some basic broadcast and all-reduce tests, as well as several well-known computational applications, specificaly, sparse matrix-vector multiplication and numerical solution of the nonlinear heat conduction equation.

Keywords: Keywords: simulation, Angara inteconnect, multi-dimensional torus, collectives operations.

References

1. Makagon D.V., Syromyatnikov E.L. Seti dlya superkomp'yuterov [Supercomputers Interconnect]. Otkrytyye sistemy. SUBD. [Open Systems. DBMS]. 2011. No. 7. P. 33–37.
2. Korzh A.A., Makagon D.V., Zhabin I.A., Syromyatnikov E.L. Otechestvennaya kommunikatsionnaya set' 3D-tor s podderzhkoy global'no adresuyemoy pamyati dlya superkomp'yuterov transpetaflopsnogo urovnya proizvoditel'nosti [Russian 3D-torus Interconnect with Support of Global Address Space Memory]. Parallelnye vychislitelnye tekhnologii (PaVT'2010): Trudy mezhdunarodnoj nauchnoj konferentsii (Ufa, 29 marta – 2 aprelya 2010) [Parallel Computational Technologies (PCT'2010): Proceedings of the International Scientific Conference (Ufa, Russia, March, 29 – April, 2, 2010)]. Chelyabinsk, Publishing of the South Ural State University, 2010. P. 527–237. URL: http://omega.sp.susu.ac.ru/books/conference/PaVT2010/full/134.pdf (accessed: 29.04.2015).
3. Simonov A.S., Zhabin I.A., Makagon D.V. Razrabotka mezhuzlovoy kommunikatsionnoy seti s topologiyey «mnogomernyy tor» i podderzhkoy global'no adresuyemoy pamyati dlya perspektivnykh otechestvennykh superkomp'yuterov [Development of the Multi-Dimensional Torus Topology Interconnectwith Support of Global Address Space Memory for Advanced National Supercomputers]. Nauchno-tekhnicheskaya konferentsiya «Perspektivnyye napravleniya razvitiya vychislitel'noy tekhniki» (Moskva, 28 iyunya) [Scientific and Technical Conference «Advanced Directions of the Computers Development Technology]. Moscow: JSC «Concern «Vega», 2011. P. 17–19

4. Eysymont L.K., Semenov A.S., Sokolov A.A., Frolov A.S., Shvorin A.B. Modelirovaniye rossiyskogo superkomp'yutera «Angara» na superkomp'yutere [Simulation of Angara Russian Supercomputer on the Supercomputer]. V sbornike «Superkomp'yuternyye tekhnologii v nauke, obrazovanii i promyshlennosti» pod redaktsiyey akademika V.A. Sadovnichego, akademika G.I. Savina, chl.-korr. RAN Vl.V. Voyevodina [Edited by Member of RAS V.A. Sadovnichy, Member of RAS G.I. Savin, Corresponding member of RAS V.V.Voevodin]. Moscow: Publishing of Moscow State University, 2009. P. 145–150.
5. Simonov A.S., Makagon D.V., Zhabin I.A., Shcherbak A.N., Syromyatnikov E.L., Polyakov D.A. Pervoye pokoleniye vysokoskorostnoy kommunikatsionnoy seti «Angara» [The First Generation of Angara High-Speed Interconnect]. Naukoyemkiye tekhnologii [Science Technologies]. 2014. Vol. 15, No. 1. P. 21–28.
6. Slutskin A.I., Simonov A.S., Zhabin I.A., Makagon D.V., Syromyatnikov E.L. Razrabotka mezhuzlovoy kommunikatsionnoy seti YES8430 «Angara» dlya perspektivnykh superkomp'yuterov [Development of ES8430 Angara Interconnect for Future Russian Supercomputers]. Uspekhi sovremennoy radioelektroniki [Progress of the Modern Radioelectronics]. 2012. No. 1. P. 6–10.
7. Zhabin I.A. Kristall dlya Angary [Angara Chip] / I.A. Zhabin, D.V. Makagon, A.S. Simonov Superkomp'yutery [Supercomputers]. Winter-2013. P. 46–49.
8. Makagon D.V., Syromyatnikov E.L., Paruta S.I., Rumyantsev A.A. Realizatsiya apparatnoy podderzhki kollektivnykh operatsiy v marshrutizatore vysokoskorostnoy kommunikatsionnoy seti s topologiyey «mnogomernyy tor» [The Implementation of Collective Operations Hardware Support in High Speed Interconnect with Multi-Dimensional Torus Topology]. Uspekhi sovremennoy radioelektroniki [Progress of the Modern Radioelectronics]. 2012. No. 1. P. 11–15.
9. Message Passing Interface Forum, MPI: A Message-Passing Interface Standard. 1995. URL: `http://www.mpi-forum.org/docs/mpi-1.1/mpi-11-html/node64.html` (accessed: 29.04.2015).
10. Feind K. Shared Memory Access (SHMEM) Routines. Cray Research. 1995. URL: `https://cug.org/5-publications/proceedings_attendee_lists/1997CD/S95PROC/303_308.PDF` (accessed: 29.04.2015).
11. Wiebe F., Wiebel E., Greenberg D., Seide S. UPC Collective Operations Specifications. 2003. URL: `http://upc.gwu.edu/docs/UPC_Coll_Spec_V1.0.pdf` (accessed: 29.04.2015).
12. Saraswat V., Bloom B., Peshansky I., Tardieu O., Grove D. X10 Language Specification. 2011. URL: `http://dist.codehaus.org/x10/documentation/languagespec/x10-latest.pdf` (accessed: 29.04.2015).
13. Fox G., Johnson M., Lyzenga G., Otto S., Salmon J., Walker D. Solving Problems on Concurrent Processors // General techniques and regular problems. Vol. 1, Prentice-Hall Inc., 1998. P. 592.
14. Bala V., Bruck J., Cypher R., Elustondo P., Ching-Tien H., Kipnis S., Snir M. CCL: a Portable and Tunable Collective Communication Library for Scalable Parallel Computers // Parallel and Distributed System. 1995. Vol. 6, P. 154–164. DOI: 10.1109/71.342126.
15. Almasi G., Dozsa G., Erway C., Steinmacher-Burow B. Efficient Implementation of Allreduce on BlueGene/L Collective Network // Recent Advances in Parallel Virtual Machine and Message Passing Interface. 2005. P. 57–66 DOI: 10.1007/11557265_12.
16. Pozhilov I.A., Semenov A.S., Makagon D.V. Prognozirovaniye masshtabiruyemosti zadachi

umnozheniya razrezhennoy matritsy na vektor pri pomoshchi modeli kommunikatsionnoy seti [Scalability Prediction of the Sparse Matrix-Vector Multiplication Using the Interconnection Network Simulator]. Vestnik UGATU. 2012. Vol. 16, No. 6 (51). P. 158–163.

17. Thakur R., Rabenseifner R., Gropp W. Optimization of Collective Communication Operations in MPICH. URL: `http://www.mcs.anl.gov/~thakur/papers/ijhpca-coll.pdf` (accessed: 29.04.2015).

Received April 10, 2015.

УДК 519.685 DOI: 10.14529/cmse150305

СТЕНД ДЛЯ ОТЛАДКИ И ТЕСТИРОВАНИЯ КАЧЕСТВА РАБОТЫ ЛОКАЛЬНЫХ СИСТЕМНЫХ РАСПРЕДЕЛЕННЫХ АЛГОРИТМОВ ДИНАМИЧЕСКОЙ БАЛАНСИРОВКИ НАГРУЗКИ[1]

В.А. Перепелкин, И.И. Сумбатянц

При параллельной реализации итерационных численных методов на регулярных сетках возникает необходимость в статической или динамической балансировке вычислительной нагрузки. Для исследования того или иного алгоритма балансировки нагрузки важно проводить его разносторонее тестирование на множестве различных задач рассматриваемого класса с различными конфигурациями вычислителя и различными наборами входных данных задач. В статье представлен тестовый стенд, предназначенный для автоматизации проведения такого тестирования. Он позволяет описать прикладную задачу и подключить реализацию алгоритма статической или динамической балансировки вычислительной нагрузки для последующего тестирования на мультикомпьютере. На выходе стенд предоставляет информацию о том, как происходило исполнение итерационного сеточного метода с точки зрения баланса вычислительной нагрузки. Приведен пример использования стенда для исследования диффузионного алгоритма динамической балансировки нагрузки на процессоры мультикомпьютера.

Ключевые слова: динамическая балансировка нагрузки, большие численные модели, автоматизация тестирования.

Введение

При реализации больших численных моделей на суперкомпьютерах, в частности, итерационных методов на регулярных сетках, встает проблема обеспечения равномерной загрузки во времени вычислительных узлов суперкомпьютера. Для решения этой проблемы используют алгоритмы статической и/или динамической балансировки вычислительной нагрузки. Эффективность таких алгоритмов (в смысле способности обеспечивать равномерность загрузки процессоров) часто сложно оценить теоретически. В значительной степени это обусловлено тем, что эффективность балансировки может существенно зависеть от конфигурации вычислителя и входных данных прикладной программы. Поэтому важную роль в исследовании алгоритмов балансировки нагрузки играет тестирование их эффективности. Такое тестирование должно охватывать широкий спектр ситуаций, чтобы получить представление о работе алгоритма балансировки в целом. В частности, при тестировании должны варьироваться такие параметры, как классы прикладных задач, входные данные задачи, размер задачи, количество вычислительных узлов, фоновая нагрузка на сеть, параметры самого алгоритма балансировки нагрузки (если имеются), и т.п.

Для автоматизации проведения таких тестов целесообразно создание отладочного тестового стенда (ОТС). ОТС — это программа, принимающая на вход реализацию некоторого алгоритма балансировки нагрузки и выполняющая серию тестов с различными

[1] Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии – 2015».

параметрами. В процессе тестирования производится реальное (не имитационное) исполнение задачи на мультикомпьютере. Результаты тестов позволяют судить о том, каковы качественные и количественные характеристики эффективности исследуемого алгоритма балансировки нагрузки на процессоры. Вследствие того, что в различных ситуациях требования к алгоритмам балансировки вычислительной нагрузки различаются, ОТС может быть разработан только для некоторой ограниченной предметной области. В настоящей работе предлагается ОТС, разработанный для исследования алгоритмов балансировки нагрузки на процессоры для итерационных методов на регулярных сетках, включая метод частиц-в-ячейках [3]. Так как в настоящей работе речь идет о численных моделях, реализуемых на суперкомпьютере, то ОТС должен быть ориентирован на локальные масштабируемые алгоритмы как прикладной задачи, так и балансировки нагрузки на процессоры.

Статья организована следующим образом. В разделе 1 представлена методика оценки алгоритмов балансировки нагрузки. В разделе 2 описывается предлагаемый тестовый стенд, в разделе 3 представлена его реализация, а в разделе 4 приводится пример использования стенда на практике. В заключении кратко изложены результаты и направление развития тестового стенда.

1. Методика оценки алгоритмов балансировки нагрузки

Для оценки и анализа работы балансировщика ОТС собирает данные о процессе исполнения численного метода. Такие данные должны быть достаточно полными и полезными для анализа. Так как стенд нацелен на практическое использование для определенного класса задач, то был исследован ряд статей, в которых были представлены алгоритмы статической и динамической балансировки нагрузки для характерного примера из целевого класса задач: итерационных сеточных методов и моделирования физических процессов методом частиц-в-ячейках.

Так было определено, что для анализа и тестирования алгоритмов балансировки нагрузки авторы использовали следующие данные:

- общее время исполнения и время моделирования [1, 2, 4];
- развертка максимума и минимума количества частиц на всех вычислительных узлах по времени моделирования [1];
- максимальное количество частиц отображенных на один вычислительный узел [2];
- развертка распределения вычислительной нагрузки по времени моделирования [6];
- время исполнения итерации на каждом узле [6].

Помимо этого существенными представляются следующие показатели:

- общее количество балансировок;
- развертка нагрузки на коммуникационную сеть по времени.

2. Структура ОТС

В соответствии с поставленной задачей, ОТС должен обеспечивать тестирование заданного алгоритма балансировки нагрузки. Соответственно, возникает потребность в унификации интерфейса модуля балансировки нагрузки (балансировщика). Кроме того, поведение прикладных программ с точки зрения баланса нагрузки на процессоры может быть очень многообразным. Как следствие, целесообразно не вкладывать все возможные ситуации в ОТС, а предоставить возможность закладывать в него различные приклад-

ные программы. Для этого в настоящей работе была предложена модель вычислений, в которой может быть представлен прикладной алгоритм из заданного класса.

2.1. Модель балансировщика

Балансировщик — это реализация локального распределенного алгоритма балансировки нагрузки на вычислительные узлы мультикомпьютера. Балансировщик должен устранять дисбаланс нагрузки и не допускать ситуаций, когда исполнение не может быть продолжено из-за отсутствия свободных ресурсов на узле. В силу своей распределенности балансировщик может опираться на каждом вычислительном узле на информацию о загруженности текущего узла и о загруженности узлов, находящихся в некоторой окрестности от него (в смысле сетевой инфраструктуры). Кроме того, балансировщик может иметь параметры (конфигурацию), определяемые статически и влияющие на его поведение (например, порог дисбаланса для диффузионных алгоритмов балансировки нагрузки).

Для ОТС балансировщик — это модуль, который содержит реализацию алгоритма балансировки и некоторый предикат, который позволяет определить, необходимо ли производить балансировку. В определенные моменты времени ОТС проверяет необходимость в балансировке, и после положительного результата инициирует процесс балансировки в некоторой окрестности текущего вычислительного узла. Значение предиката проверяется на каждом вычислительном узле и в случае необходимости балансировка асинхронно инициируется на одном из узлов. При этом возможно одновременное выполнение нескольких балансировок в разных частях мультикомпьютера.

2.2. Модель вычислений прикладной программы

Прикладной алгоритм представляется в виде графа $\langle F, N, op, R\rangle$, где F — множество вершин графа (далее — фрагментов), N — множество ребер, соединяющих вершины из множества F (отношение соседства), op — оператор, который применяется к каждому элементу F, R — оператор редукции, который применяется ко всем элементам F. Процесс вычислений происходит в дискретном времени $T = \{t_1 \dots t_n\}$, и в каждый момент времени t_i элемент $f \in F$ имеет состояние (значение) f_{t_i}. Каждое следующее состояние фрагмента $f_{t_{i+1}}$ зависит от его текущего состояния f_{t_i}, от текущих состояний соседних фрагментов $\left\{fn_{t_i} : (fn \in F)\&((fn, f) \in F)\right\}$ и от редукционных данных времени t_i. Тогда в общем виде процесс вычислений выглядит следующим образом:

$$\forall f \in F \left(f_{t_i} = op\left(f_{t_{i-1}}, \left\{ fn_{t_{i-1}} : (fn \in F)\&\big((fn, f) \in N\big)\right\}, R(t_{i-1})\right)\right).$$

Для примера рассмотрим метод частиц-в-ячейках [2, 3]. В этом методе имеется пространство моделирования, в котором движутся частицы, взаимодействуя с полем (полями). Поля дискретизуются на статичной регулярной сетке. Применение метода пространственной декомпозиции приводит к разделению пространства моделирования на домены, каждый из которых содержит часть сеточных значений и значения частиц, принадлежащих этому домену. Распределение частиц по доменам изменяется во времени. В этом примере фрагментом будет домен с его сеточными значениями и значениями частиц, оператор скрывает в себе вычисление новых координат частиц и новых сеточных значений, а редукционный оператор применяется для определения таких величин, как суммарная энергия системы.

2.3. Реализация вычислений на мультикомпьютере

Тестовый стенд реализует описанный прикладной вычислительный процесс на мультикомпьютере. Каждый узел мультикомпьютера может обмениваться сообщениями с ограниченным множеством других узлов. Конкретная топология соединений задается стендом. Далее будем называть такое множество *локальной окрестностью узла*, а узлы из этого множества — *соседними узлами*.

В ходе исполнения ОТС может предоставить фрагменту данные соседних фрагментов, передать данные от одного фрагмента другому и произвести редукцию данных между всеми фрагментами.

Начальное отображение фрагментов на узлы мультикомпьютера можно производить несколькими способами:

- статически определять место создания фрагментов до начала исполнения;
- создавать фрагменты на нескольких выделенных узлах после начала исполнения, для построения начального отображения с помощью балансировщика.

По требованию балансировщика система может перемещать фрагменты на соседние узлы мультикомпьютера в рамках локальной окрестности и создавать распределенную реализацию фрагмента, если на узле нет достаточного количества ресурсов для его хранения или обработки (в случае, если эта возможность поддерживается со стороны прикладной задачи).

3. Реализация ОТС

Предложенный ОТС был реализован в виде программного прототипа. Рассмотрим его. ОТС содержит интерфейс для подключения балансировщика, интерфейс для подключения модуля, реализующего прикладной алгоритм и систему, обеспечивающую исполнение прикладного алгоритма. Таким образом, на вход ОТС приниимает реализацию задачи и реализацию балансировщика.

3.1. Реализация задачи

В соответствии с моделью вычислений, прикладная задача — это множество фрагментов, отношение соседства на множестве фрагментов и операторы шага итерации и редукции. ОТС предоставляет интерфейсы для описания множества фрагментов, которые инкапсулируют в себе эти операторы и отношение соседства. ОТС и интерфейсы были реализованы на языке C++. Это язык был выбран по двум причинам: существует несколько реализаций MPI для C++, модель вычислений достаточно хорошо укладывается в объектно-ориентированную парадигму программирования. Таким образом, со стороны пользователя, прикладная задача — это реализация соответствующего интерфейса ОТС на языке C++. Таких реализаций для описания задачи может быть несколько, с разными поведением и функциями.

Пользователь ОТС может определять любые данные в реализации фрагмента, которые будут определять его состояние. Интерфейсы содержат управляемый пользователем внутренний счетчик, который определяет модельный момент времени (шаг модельного времени или номер итерации). Для упрощения описания процесса исполнения помимо счетчика модельного времени был введен счетчик прогресса исполенния на данном шаге итерации (подшаг итерации). Информация об отношении соседства на множестве фрагментов хранится распределенно: каждый объект из множества фрагментов

содержит информацию о его локальной окрестности (локальная окрестность задается пользователем в момент создания экземпляра фрагмента). Реализации операторов шага итерации и редукции — это реализации методов интерфейсов фрагмента. На рис. 1 представлена существенная часть исходного кода интерфейса для описания множества фрагментов.

```
class Fragment {
private:
    ID _vid; ///< ID of fragment
    ts::NodeID _vnodeID = 0; ///< Logic fragment location
    std::map<ID, ts::NodeID> _vneighboursLocation; ///< Fragment's
        /// neighbours locations

public:
    // General
    Fragment(ID id);
    virtual ~Fragment();
    ID id();
    void setNodeID(ts::NodeID);

    // Neighbours and their locations
    bool isNeighbour(const ID& id);
    void updateNeighbour(ID id, ts::NodeID node);
    void addNeighbour(ID id, ts::NodeID node);

    // Fragment steps defined by user
    virtual ReduceData* reduce() = 0;
    virtual ReduceData* reduce(ReduceData* data) = 0;
    virtual void reduceStep(ReduceData* data) = 0;
    virtual void runStep(std::vector<Fragment*> neighbours) = 0;

    // Flag setters
    void setEnd();
    void setUpdate();
    void setReduce();
    void setNeighbours(uint64_t iteration, uint64_t progress);

    // State setters
    void nextIteration();

    // State getters
    uint64_t iteration();
    uint64_t progress();
};
```

Рис. 1. Исходный кода интерфейса для описания множества фрагментов

Реализация задачи в терминах стенда — это множество объектов (фрагментов) из реализованного множества фрагментов, которые передаются ОТС в качестве входа. Начальное распределение фрагментов по узлам мультикомпьютера может быть задано статически: в процессе порождения фрагментов на мультикомпьютера, либо для этой цели может быть использован балансировщик: фрагменты порождаются на нескольких выделенных узлах, и балансировщик во время порождения начинает распределять фрагменты по соседним узлам мультикомпьютера.

ОТС скрывает в себе такие операции как: применение оператора к фрагменту, поиск соседних фрагментов для применения оператора, редукция данных, миграция фрагмента.

3.2. Исполнение задачи

ОТС содержит множество фрагментов, итерационные шаги которых необходимо исполнять. Исполнение производится по подшагам. Управление ходом исполнения производится с помощью установки флагов: при необходимости в редукции данных; необходимость в соседних фрагментах для применения оператора; необходимость в обновлении состояния фрагмента для глобального использования; для указания того, что на следующем подшаге будет следующая итерация. Флаги согласованно устанавливаются фрагментами в реализованном операторе.

ОТС передает фрагменты на исполнение в соответствии с выставленными флагами. Если фрагменту для данного подшага не нужны значения (состояния) соседних фрагментов, то он сразу передается на исполнение. В противном случае сначала производится поиск соответствующих фрагментов на данном узле, если не все фрагменты найдены, то данный фрагмент пропускается, пока на узел не придут значения всех соседних фрагментов. Во многих задачах для применения оператора к фрагменту нет необходимости знать полное состояние соседних фрагментов (например, в методе частиц-в-ячейках из сеточных значений соседей необходимы смежные грани сетки), поэтому для оптимизации скорости работы ОТС была введена такая абстракция как частичная реализация фрагмента, которая определяется пользователем. И именно частичная реализация фрагмента передается на узел с фрагментом, которому для применения оператора необходимы соседние фрагменты.

3.3. Балансировка

Балансировщик реализуется как внешняя динамически подключаемая к ОТС библиотека. Соответственно, балансировщик может быть реализован на любом языке программирования, который позволяет собрать динамическую библиотеку. Реализация алгоритма балансировки работает в терминах нагрузки: на вход принимает количество вычислительной нагрузки на локальный узел и на соседние узлы, а на выходе предоставляет информацию о том, как нагрузка должна быть распределена. После этого ОТС сам решает какие именно фрагменты необходимо передать. Величина нагрузки определяется ОТС в единицах, зависящих от задачи (и определяемых пользователем при описании прикладной задачи).

ОТС периодически обращается к балансировщику для определения необходимости инициации балансировки на узле.

Миграция фрагмента с узла i на узел j производится следующим образом: оповещается узел j о начале миграции; оповещаются все узлы, на которых есть соседние фрагменты, о том, что фрагмент будет перемещен на узел j; на узел j передается фрагмент и все частичные реализации фрагментов, которые были переданы ранее на узел i.

3.4. Ограничения реализации

Реализованный программный прототип ОТС имеет следующие ограничения:

- исполнение допустимо только на логической топологии «кольцо»;
- отсутствие поддержки распределенных реализаций фрагментов.

4. Пример использования

В качестве прикладной задачи для практического испытания ОТС была выбрана задача расчета изображения методом трассировки лучей. Эта задача была выбрана в качестве примера в связи с тем, что при ее параллельной реализации методом пространственной декомпозиции балансировка нагрузки на процессоры является сложной задачей, как правило требующей динамического решения.

Трассировка лучей — технология построения изображения трехмерных моделей, при которых отслеживается обратная траектория распространения луча [5]. Характерной особенностью этой задачи является различная вычислительная сложность обработки разных пикселей, которая определяется тем, сколько раз преломится/отразится луч, выпущенный через него. Это приводит к необходимости динамической балансировки нагрузки на процессоры.

Декомпозиция задачи была выполнена по пространству на блоки, каждый из которых вычисляется независимо. После обработки блоков полученные части изображения масштабируются и собираются в отдельном фрагменте.

С точки зрения модели вычислений каждый фрагмент содержит описание сцены, координаты камеры и экрана, границы экрана, до которых необходимо производить расчет. Процесс исполнения состоит из 3 шагов: вычисление пикселей части изображения, уменьшение полученной части изображения, передача полученного изображения специальному фрагменту. В качестве оценки вычислительной нагрузки каждого фрагмента использовалось количество пикселей, которые необходимо обработать. На каждом вычислителе порождается одинаковое количество фрагментов, после чего инициируется исполнение.

4.1. Балансировка нагрузки

Для задачи трассировки лучей был реализован локальный распределенный алгоритм балансировки нагрузки, который можно описать следующей схемой:

1. $load$ = нагрузка на текущем узле
2. Цикл по всем узлам i из 1-окрестности текущего узла:
    a. Если нагрузка на i ($nload$) меньше нагрузки $load$ на 20 % и более
        i. $diff = (load - nload) \,/\, 2$
        ii. Передать узлу i количество нагрузки $diff$
        iii. $load = load - diff$

Таким образом, балансировка нагрузки инициируется каждый раз, когда нагрузка на узле изменяется на $1/5$ часть, с момента начала исполнения или с момента последней балансировки.

4.2. Результаты

Предложенный алгоритм балансировки нагрузки был реализован и протестирован на ОТС. С помощью трассировки лучей отрисовывалось изображение размером 5000×5000 пикселей, которое было декомпозировано на 100 фрагментов и отображено на 4 и 10 вычислительных узлов. Фрагменты были отображены на вычислительные узлы таким образом, чтобы возникал дисбаланс вычислительной нагрузки. Для удобства визуализации нагрузка была нормирована. Для сравнения были также получены резуль-

таты работы задачи без динамической балансировки. Для начала приведем результаты тестирования.

По результатам измерения времени исполнения программы, приведенным в таблице, можно заметить, производительность увеличивается с увеличением количества узлов. Но балансировщик не дает значительного прироста производительности в данном случае.

Таблица

Время выполнения программы

Количество узлов	Без балансировки, с	С балансировкой, с
4	243	224
10	119	104

Для того, чтобы понять, почему это происходит, обратимся к разверткам нагрузки на вычислительные узлы по времени, полученным в процессе исполнения.

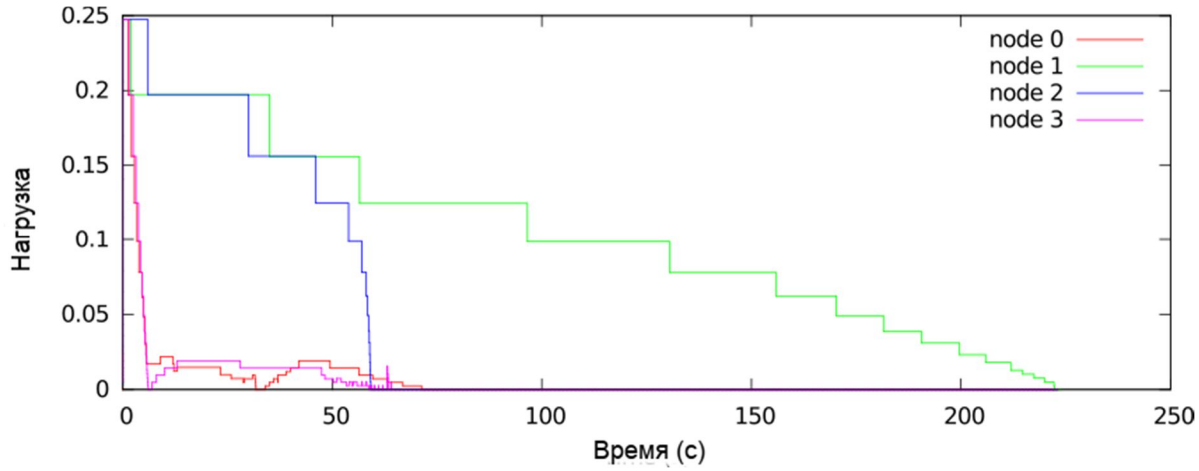


Рис. 2. Развертка нагрузки на вычислительные узлы по времени (4 узла)

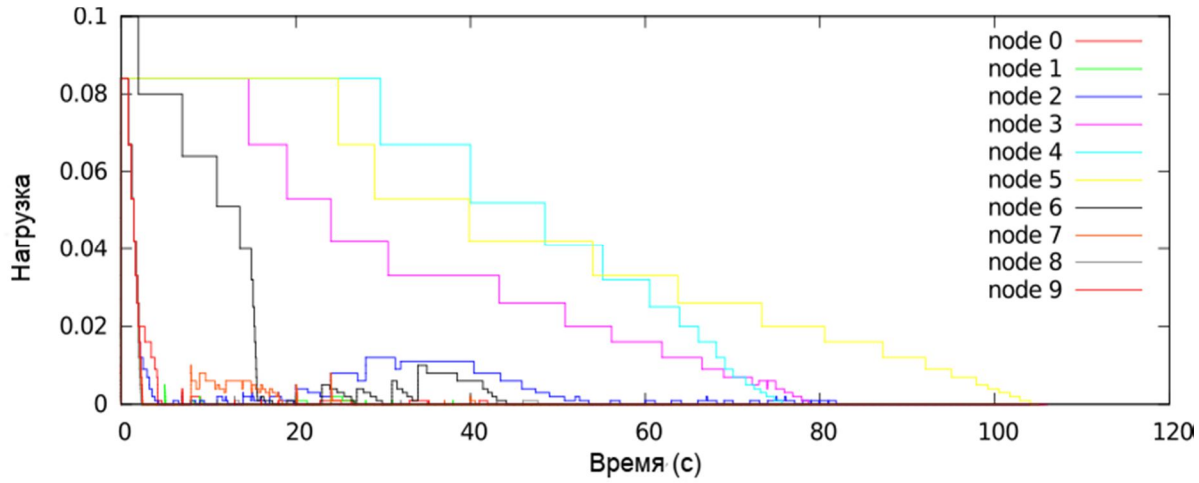


Рис. 3. Развертка нагрузки на вычислительные узлы по времени (10 узлов)

Как можно заметить из графиков (рис. 2 и 3), балансировщик действительно работает: нагрузка на некоторые узлы периодически увеличивается. Но основная проблема — это порог дисбаланса: балансировщик срабатывает при изменении нагрузки на 1/5

часть от текущей. Ближе к концу вычислений нагрузка не изменяется на такую величину, но, тем не менее, на узле остается большое количество пикселей, на которые приходится больше всего отражений. Из-за этого балансировщик работает неэффективно. Таким образом, другой проблемой исследуемого алгоритма балансировки является то, что оценка степени загруженности узла по числу пикселей является грубой, если средний вычислительный вес пикселей на разных узлах существенно отличается.

Данный пример наглядно демонстрирует процесс использования ОТС для исследования свойств алгоритма балансировки нагрузки на заданной задаче.

Заключение

В статье рассмотрен отладочный тестовый стенд, предназначенный для испытания различных характеристик алгоритмов балансировки нагрузки на процессоры. Стенд принимает на вход описание прикладной задачи на базе предложенной модели вычислений, а также реализацию алгоритма балансировки вычислительной нагрузки на базе предложенного интерфейса. Приведен пример исследования алгоритма динамической балансировки вычислительной нагрузки на процессоры в задаче построения изображения методом трассировки лучей. В будущем планируется продолжить развитие ОТС: реализовать поддержку описания виртуальной топологии сети, распределенной реализации фрагмента, и улучшить отдельные модули стенда.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта 14-01-31328 мол_а.

Литература

1. Ferraro, R.D. Dynamic load balancing for a 2D concurrent plasma PIC code / Robert D. Ferraro, Paulett C. Liewer, Viktor K. Decyk //Journal of computational physics. — 1993. — Vol. 109, N. 2. — P. 329–341. DOI: 10.1006/jcph.1993.1221.
2. Kraeva, M.A. Assembly technology for parallel realization of numerical models on MIMD-multicomputers. / M.A. Kraeva, V.E. Malyshkin // Future Generation Computer Systems. — 2001. — P. 755–765. DOI: 10.1016/s0167-739x(00)00058-3.
3. Kraeva, M.A. Implementation of PIC method on MIMD multicomputers with assembly technology / M.A. Kraeva, V.E. Malyshkin // High-Performance Computing and Networking. — 1997. — P. 541–549. DOI: 10.1007/bfb0031627.
4. Nakashima, H. OhHelp: a scalable domain-decomposing dynamic load balancing for particle-in-cell simulations / Hiroshi Nakashima, Yohei Miyake, Hideyuki Usui, Yoshiharu Omura // Proceedings of the 23rd international conference on Supercomputing. — 2009. — P. 90–99. DOI: 10.1145/1542275.1542293.
5. Ploeg, A.J. Interactive Ray Tracing — 2011 / A.J. van der Ploeg. URL: http://www.few.vu.nl/~kielmann/theses/avdploeg.pdf (дата обращения: 14.10.2014).
6. Wolfheimer, F. A parallel 3D particle-in-cell code with dynamic load balancing / Wolfheimer, Gjonaj, Weiland // Proceedings of the 8th International Computational Accelerator Physics Conference. — 2006. — Vol. 558, N. 1 — P. 202–204. DOI: 10.1016/j.nima.2005.11.003.

Перепелкин Владислав Александрович, м.н.с., лаборатория Синтеза параллельных программ, Институт вычислительной математики и математической геофизики СО РАН (Новосибирск, Российская Федерация), perepelkin@ssd.sscc.ru.

Сумбатянц Илья Ильич, магистрант, Новосибирский государственный университет (Новосибирск, Российская Федерация), ilya.sumb@gmail.com.

Поступила в редакцию 10 апреля 2015 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2015, vol. 4, no. 3, pp. 56–66

DOI: 10.14529/cmse150305

TEST BENCH FOR DISTRIBUTED DYNAMIC LOAD BALANCING ALGORITHMS WITH LOCAL COMMUNICATIONS

V.A. Perepelkin, Institute of Computational Mathematics and Mathematical Geophysics, Siberian Branch of Russian Academy of Sciences (Novosibirsk, Russian Federation) perepelkin@ssd.sscc.ru,

I.I. Sumbatyants, Novosibirsk State University (Novosibirsk, Russian Federation) ilya.sumb@gmail.com

Parallel implementation of iterative methods on regular meshes often requires static or dynamic load balancing. To study a load balancing algorithm it is important to perform versatile testing on a variety of application problems of given class, on different hardware configuration and input data sets. In the paper a software test bench is introduced. The purpose of the bench is to automate such testing. It allows to describe an application problem and to utilize user load balancing algorithm to perform tests on a multicomputer. The result of such testing is an information on the load algorithm's performance.

Keywords: dynamic load balancing, large-scale numerical modeling, performance testing automation.

References

1. Ferraro, R.D. Dynamic load balancing for a 2D concurrent plasma PIC code / Robert D. Ferraro, Paulett C. Liewer, Viktor K. Decyk //Journal of computational physics. — 1993. — Vol. 109, No. 2. — P. 329–341. DOI: 10.1006/jcph.1993.1221.
2. Kraeva, M.A. Assembly technology for parallel realization of numerical models on MIMD-multicomputers. / M.A. Kraeva, V.E. Malyshkin // Future Generation Computer Systems. — 2001. — P. 755–765. DOI: 10.1016/s0167-739x(00)00058-3.
3. Kraeva, M.A. Implementation of PIC method on MIMD multicomputers with assembly technology / M.A. Kraeva, V.E. Malyshkin // High-Performance Computing and Networking. — 1997. — P. 541–549. DOI: 10.1007/bfb0031627.

4. Nakashima, H. OhHelp: a scalable domain-decomposing dynamic load balancing for particle-in-cell simulations / Hiroshi Nakashima, Yohei Miyake, Hideyuki Usui, Yoshiharu Omura // Proceedings of the 23rd international conference on Supercomputing. — 2009. — P. 90–99. DOI: 10.1145/1542275.1542293.
5. Ploeg, A.J. Interactive Ray Tracing — 2011 / A.J. van der Ploeg. URL: http://www.few.vu.nl/~kielmann/theses/avdploeg.pdf (accessed: 14.10.2014).
6. Wolfheimer, F. A parallel 3D particle-in-cell code with dynamic load balancing / Wolfheimer, Gjonaj, Weiland // Proceedings of the 8th International Computational Accelerator Physics Conference. — 2006. — Vol. 558, N. 1 — P. 202–204. DOI: 10.1016/j.nima.2005.11.003.

Received April 10, 2015.

УДК 004.272.32, 004.832.25 DOI: 10.14529/cmse150306

СРАВНЕНИЕ ЭФФЕКТИВНОСТИ CPU И GPU РЕАЛИЗАЦИЙ НЕКОТОРЫХ КОМБИНАТОРНЫХ АЛГОРИТМОВ НА ЗАДАЧАХ ОБРАЩЕНИЯ КРИПТОГРАФИЧЕСКИХ ФУНКЦИЙ[1]

В.Г. Булавинцев

Проводится сравнение эффективности CPU и GPU реализаций некоторых комбинаторных алгоритмов, используемых в криптоанализе. В частности, анализируются причины, по которым не удается эффективно реализовать на GPU алгоритмы, осуществляющие «интеллектуальный перебор». Показывается, что применение специальных техник трансформации потока управления позволяет существенно компенсировать потери производительности, возникающие из-за неэффективного исполнения условных переходов на SIMD-устройстве. Однако ограничения, которые накладывают механизмы работы с памятью, применяемые в современных GPU, для рассматриваемых алгоритмов оказываются непреодолимыми. В качестве тестовых задач рассматриваются задачи обращения криптографических функций DES и A5/1.

Ключевые слова: GPU, CUDA, криптоанализ, DPLL, SAT, SIMD.

Введение

Современные GPU предоставляют выгодное соотношение цены, производительности и энергопотребления. Многие суперкомпьютерные кластеры имеют высокий рейтинг в ТОП500, благодаря использованию GPU [1]. Однако, будучи специализированными устройствами, рассчитанными на потоковую обработку однотипных данных, GPU на многих алгоритмах не показывают значимого преимущества перед процессорами традиционной архитектуры [2]. Это обусловлено тем, что современные GPU используют SIMD-архитектуру [3] и специально рассчитанные на работу с ней контроллеры памяти [4]. GPU являются «массивно-параллельными» процессорами, т.е. вычислительными устройствами, на которых одновременно выполняется большое (по сравнению с CPU) количество вычислительных потоков. Многие алгоритмы, применяемые в криптографии и криптоанализе, имеют возможность почти неограниченного масштабирования на параллельных вычислительных архитектурах. Криптоанализ методом прямого перебора, обслуживание криптовалют [5], построение rainbow-таблиц [6] — популярные алгоритмы демонстрирующие высокую эффективность на GPU. В то же время перспективным направлением в криптоанализе является применение алгоритмов, основанных на различных подходах к «интеллектуальному» сокращению перебора. В частности, в последние годы появился ряд работ, в которых для решения задач криптоанализа используются современные SAT-решатели [7, 8]. При реализации таких алгоритмов на GPU обнаруживается, что эта архитектура совершенно не приспособлена для их эффективного выполнения: малый объем кэш-памяти, проблемы с условными переходами, узкоспециальные механизмы работы с памятью — все эти факторы приводят к тому, что, даже

[1] Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии – 2015»

после тщательной адаптации алгоритма в соответствии с рекомендациями производителя, GPU сильно проигрывают CPU в скорости и эффективности их исполнения.

До сих пор основное внимание исследователей (за редким исключением, [2]) было сосредоточено на положительных аспектах применения GPU. Настоящая работа посвящена исследованию основных причин низкой производительности «интеллектуальных» комбинаторных алгоритмов на GPU на примере алгоритма DPLL.

Статья организована следующим образом. В первом разделе приводятся сведения о исследуемых в работе криптоалгоритмах DES и A5/1, и реализации атак на них методом полного перебора и при помощи SAT-подхода (алгоритма DPLL). Во втором и третьем разделах рассматриваются основные аппаратные особенности GPU, препятствующие эффективной реализации алгоритма DPLL. Демонстрируется метод, позволяющий частично преодолеть одно из этих препятствий — проблему неэффективной обработки условных переходов на SIMD-архитектурах. В четвертом разделе приведены результаты вычислительных экспериментов, исследующих эффективность рассмотренных алгоритмов. В заключении кратко описаны выводы, следующие из работы, обсуждаются перспективы развития платформы GPU.

1. Описание использованных алгоритмов

В качестве модельных задач для исследования мы выбрали криптоалгоритмы DES и A5/1. На сегодняшний день эти алгоритмы широко распространены и хорошо изучены. Мы реализовали атаку на них двумя способами: во-первых, классическим методом «прямого перебора» (т.н. brute-force, метод «грубой силы»), во вторых — с применением SAT-подхода. Многие задачи криптоанализа можно рассматривать в контексте более общей задачи обращения дискретных функций. Последняя задача эффективно сводится к задаче поиска выполняющего набора булевой формулы (т.н. SAT-задача).
Далее приведем описание реализаций на GPU брутфорс-атаки на DES и A5/1, а также реализации алгоритма DPLL, используемого для решения SAT-задач.

1.1. Криптоанализ генератора A5/1 на GPU методом полного перебора

Генератор потокового шифра A5/1 является стандартным для применения в современных сетях мобильной связи GSM. Он представляет собой [9] три независимых друг от друга регистра сдвига с линейной обратной связью (РСЛОС) длиной 19, 22 и 23 бит (всего 64 бит). РСЛОСы тактируются условно, по т.н. функции большинства («majority function»), взятой от одного бита из каждого регистра. Для получения ключевого потока выходы РСЛОСов смешиваются друг с другом. Секретным ключом является начальное заполнение РСЛОСов. Шифрование осуществляется побитовым смешиванием ключевого потока, порожденного генератором, с открытым текстом с помощью функции XOR. Протокол шифрования A5/1 продолжает использоваться, несмотря на то, что различными исследовательскими группами были продемонстрированы практические атаки на него [6, 9].

Далее мы рассматриваем задачу криптоанализа A5/1 на основе известного фрагмента ключевого потока [10]. Простейший метод криптоанализа в этом случае — полный перебор: для каждого из всех возможных вариантов секретного ключа генерируется соответствующий фрагмент ключевого потока, который сравнивается с заранее известным

образцом. Для генератора A5/1, чтобы однозначно определить секретный ключ достаточно фрагмента ключевого потока длиной 64 бит [10]. Таким образом, в худшем случае требуется для каждого из 2^{64} ключей-кандидатов получить 64 бита выхода генератора и сравнить их с известным образцом ключевого потока. Поскольку шифр является потоковым, нет необходимости генерировать сразу все 64 битов выхода. Достаточно генерировать ключевой поток по 1 биту и, в случае его несовпадения с образцом, прерывать проверку текущего ключа-кандидата.

Мы использовали специальную реализацию алгоритма A5/1. Поскольку РСЛОСы независимы друг от друга и их длина невелика (19–23 бит), для каждого из них можно сгенерировать соответствующую непериодическую часть порождаемой ими двоичной последовательности (т.н. *M*-последовательность). Для используемых в A5/1 РСЛОС получаемые последовательности занимают в сумме около 2 Мбайт оперативной памяти. Затем состояние любого бита РСЛОСа для любого его такта при любом начальном заполнении может быть получено выборкой n-ого значения из соответствующей последовательности по формуле: $n=(S+t+b)$, где S — смещение от начала последовательности, кодирующее начальное заполнение РСЛОСа; t — номер такта; b — номер бита в РСЛОСе. В таком случае вместо перебора начальных заполнений РСЛОСов в лексикографическом порядке, перебор проходит в порядке, заданном *M*-последовательностью. Поскольку значения в *M*-последовательности не повторяются, лишняя работа не производится. Эта версия алгоритма может быть реализована как на CPU, так и на GPU. Она не предъявляет специальных требований к возможностям оборудования[2]. При переносе на GPU ускорение достигается за счет одновременной проверки многих ключей-кандидатов. Каждый из тысяч вычислительных потоков GPU проверяет один вариант ключа. Данные по скорости реализации описанной атаки на GPU приведены табл 3.

1.2. Криптоанализ алгоритма DES на GPU методом полного перебора

Блочный шифр DES являлся федеральным стандартом США с 80-х годов и активно применялся вплоть до конца XX века. DES — симметричный шифр, построенный на сети Фейстеля. Длина ключа в DES составляет всего 56 бит, что и является главной его уязвимостью. С середины 90-х годов были неоднократно продемонстрированы практичные атаки на DES, использующие «метод грубой силы» (например, [11]).

Мы реализовали брутфорс-атаку на DES в условиях известного открытого текста. Для ускорения работы алгоритма мы применили технику bitslice, повышающую эффективность использования возможностей современных вычислительных архитектур [12].

Опишем вкратце технику bitslice. Шифр представляется в виде схемы, составленной из логических вентилей. Используя соответствующую этой схеме последовательность побитовых логических операций, можно одновременно проверять столько ключей-кандидатов (или шифровать столько блоков секретного текста), сколько двоичных разрядов содержит один регистр общего назначения (РОН) вычислительного устройства. Например, на 32-разрядной платформе в технике bitslice каждый РОН содержит в себе по 1 биту каждого из 32 ключей-кандидатов. Таким образом, на 32-разрядной платформе bitslice позволяет проверять одновременно 32 ключа. Следует отметить, что построение оптимальной схемы из вентилей является нетривиальной задачей, и исследователи

[2] Например, для эффективной реализации алгоритма A5 в технике bitslice требуется, чтобы устройство поддерживало аппаратную инструкцию bitselect или аналогичную ей [6].

до сих пор работают над улучшением такой схемы для DES. В нашей работе мы использовали лучшую известную на данный момент схему для DES на стандартных вентилях (*И, ИЛИ, НЕ, XOR*) [13].

Как и в случае A5/1, при переносе на GPU ускорение достигается за счет одновременной проверки многих ключей. Однако в данном случае каждый вычислительный поток GPU проверяет 32 ключа — за счет использования техники bitslice (для удобства сравнения версия для CPU работает в 1 поток и проверяет на нем за счет техники bitslice одновременно также 32 ключа). Данные по скорости перебора пространства ключей DES на GPU, достигнутой в результате применения описанного подхода, приведены в конце табл. 3.

1.3. Реализация алгоритма DPLL на GPU

Наиболее эффективные программы-решатели SAT-задач построены на базе классического алгоритма DPLL [14]. Он представляет собой направленный обход дерева, дополненный правилом распространения булевых ограничений (Boolean constraint propagation, BCP). Для ускорения процедуры BCP, которая занимает до 95 % времени работы решателя, применяются специальные «ленивые» структуры данных, т.н. «watched literals» [15]. Также, в современных решателях используется стратегия Conflict-Driven Clause Learning (CDCL): обнаружив конфликт в присвоении переменных, решатель анализирует его причины и добавляет информацию о них к основной базе ограничений в виде т.н. «конфликтного дизъюнкта». Это позволяет ускорить обход дерева, пропустив проверку переменных, не имеющих отношения к конфликту («backjumping») [16]. В дальнейшем конфликтный дизъюнкт может ускорить вывод по BCP.

Рассмотрев различные подходы к распараллеливанию алгоритма DPLL, мы пришли к выводу, что единственный способ полностью задействовать возможности GPU, не ориентируясь при этом на особенности SAT-задач, — это запускать в каждом вычислительном потоке отдельный экземпляр алгоритма. При этом общее дерево поиска расщепляется по значениям n отдельных переменных на 2^n подзадач, и каждый вычислительный поток GPU решает одну такую подзадачу. Когда поток заканчивает работу над своей подзадачей, он ищет другой поток, у которого еще есть работа, и расщепляет его подзадачу, забирая часть работы себе (при этом используется техника «work stealing» [17]).

1.4. Адаптация структур данных, используемых в DPLL, для GPU

Малый объем памяти современных GPU является одним из препятствий для реализации полноценной стратегии CDCL. Размер КНФ в задачах криптоанализа может достигать 10^4–10^6 литералов [8]. В то же время, при 2048 запущенных на GPU потоках, на каждый поток приходится не более 512 Кбайт из 1 Гбайт памяти устройства. Поэтому мы реализовали ограниченную версию CDCL, с хранением только тех дизъюнктов, которые необходимы для корректной работы нехронологического бэктрекинга [16].

Для представления дизъюнктов в памяти GPU мы разработали новые структуры данных, сокращающие потребление памяти решателем с 32 бит до 1 бита на каждый литерал КНФ. Требования к памяти GPU в нашем решателе могут быть определены по формуле: $M{=}M_o{+}n_nM_u$, где M_o — общие для всех потоков данные, M_u — индивидуальные рабочие данные потоков, n_n — число потоков. К примеру, для КНФ: A5/1 M_o=1160 Кбайт, M_u=114 Кбайт (с учетом требований выравнивания памяти для работы

на GPU). Эти 2048 потоков и общие данные КНФ при решении A5/1 (119700 литералов, 7817 переменных) занимают в памяти устройства около 270 Мбайт. Если бы мы применили схему, в которой каждый поток хранит все данные КНФ полностью, для размещения данных тех же 2048 потоков понадобилось бы не менее 1200 Мбайт. Кроме того, при запуске большого числа потоков примененная нами схема хранения является более «дружественной» к использованию кэш-памяти, т.к. до 50 % запросов к памяти в алгоритме DPLL составляют обращения к литералам. Поскольку общие для всех потоков данные занимают около 1 Мб, они могут почти полностью разместиться в кэш-памяти, быстрым доступом к которым могут воспользоваться все потоки. В случае традиционной схемы ресурсы кэш-памяти устройства были бы распределены между всеми потоками, что привело бы к значительному снижению эффективности кэширования.

2. Особенности обработки условных переходов на GPU

Современные GPU NVIDIA состоят из нескольких SIMD-ядер, называемых «мультипроцессорами». Каждый мультипроцессор обладает собственным набором регистровой памяти, кэшем L1, общей памятью, несколькими АЛУ, диспетчером команд, контроллером условных переходов и т.д. Мультипроцессоры независимы друг от друга и подключены к основной памяти устройства через кэш L2.

Для выполнения на мультипроцессоре вычислительные потоки GPU объединяются в SIMD-группы размером в 32 потока, именуемые «варпами» («warp»). Потоки в варпе выполняются в режиме «lockstep», т.е. «1 шаг — 1 одинаковая инструкция для всех потоков».

Условные переходы в SIMD обрабатываются особым образом. Если во время исполнения оператора условного перехода возникает ситуация, когда часть потоков в варпе должна выполнить один блок инструкций, а оставшиеся потоки — другой, происходит последовательное выполнение этих блоков (сериализация). Вначале будет выполнен блок, назначенный наибольшему числу потоков, затем — назначенный остальным. При сериализации потоки, не получившие инструкции, простаивают в бездействии (рис. 1). Это явление называется «warp divergence», оно является следствием экономии на управляющей логике в SIMD-архитектурах. К примеру, в результате последовательности из 5 вложенных условных переходов может возникнуть ситуация, когда из всех 32 потоков варпа в каждый момент времени будет выполняться только 1. Производительность мультипроцессора на таком участке кода при этом ухудшается пропорционально сериализации, до 32 раз. Чтобы уменьшить потери производительности от этого «SIMD-эффекта», можно применять различные техники программирования, позволяющие сократить в коде количество условных переходов и/или степень их вложенности. Далее мы описываем соответствующие техники, использованные нами при реализации на GPU нехронологического алгоритма DPLL.

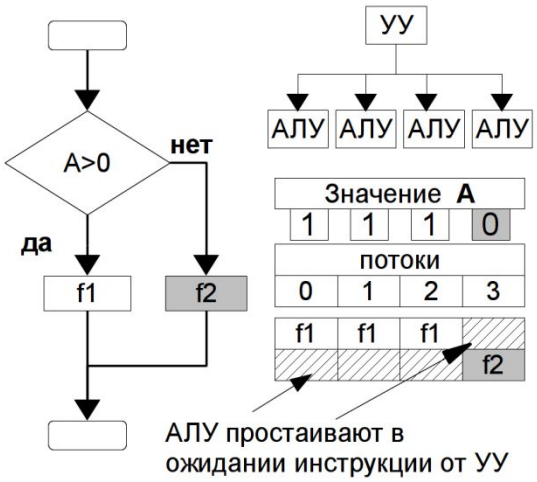


Рис. 1. Сериализация инструкций в результате условного перехода в SIMD

2.1. Увеличение эффективности обработки условных переходов в SIMD за счет реорганизации структуры вложенных циклов

Алгоритм DPLL можно условно рассматривать как конструкцию, состоящую из 4-х вложенных циклов:

1. цикл обхода дерева поиска;
2. цикл обработки очереди переменных в BCP;
3. цикл обработки дизъюнктов одной переменной;
4. цикл обработки литералов в отдельном дизъюнкте (цикл нижнего уровня).

Цикл обхода дерева поиска включает в себя процедуры угадывания новых переменных и backjumping'a после обнаружения конфликтного присвоения. Однако по сравнению с процедурой BCP эти процедуры используются редко — около 95 % всего времени занимает BCP.

Процедуры этих циклов оптимизированы так, чтобы выполнять только те итерации, которые действительно необходимы. Дизъюнкты могут иметь разную длину и невозможно определить заранее, сколько литералов из него понадобится просмотреть. Подмножество дизъюнктов, требующих проверки, также отличается для разных переменных и зависит от текущего их состояния и пути, пройденного решателем. Выведенные по BCP литералы добавляются в очередь, поэтому и ее длину невозможно предсказать заранее, она меняется динамически по ходу решения. Путь решателя в дереве также непредсказуем. Таким образом, каждый из 4-х циклов будет иметь непредсказуемую длину, зависящую от решаемой подзадачи, порядка угадывания переменных и т.д. Вычислительное время, необходимое для обработки любого из этих циклов также может варьироваться в широких пределах. Поскольку число итераций этих циклов заранее неизвестно, выход из них происходит по условию. В результате для DPLL глубина вложенности условных переходов составляет, как правило, 4 и более, что крайне негативно сказывается на производительности алгоритма на GPU, за счет усиления «SIMD-эффекта».

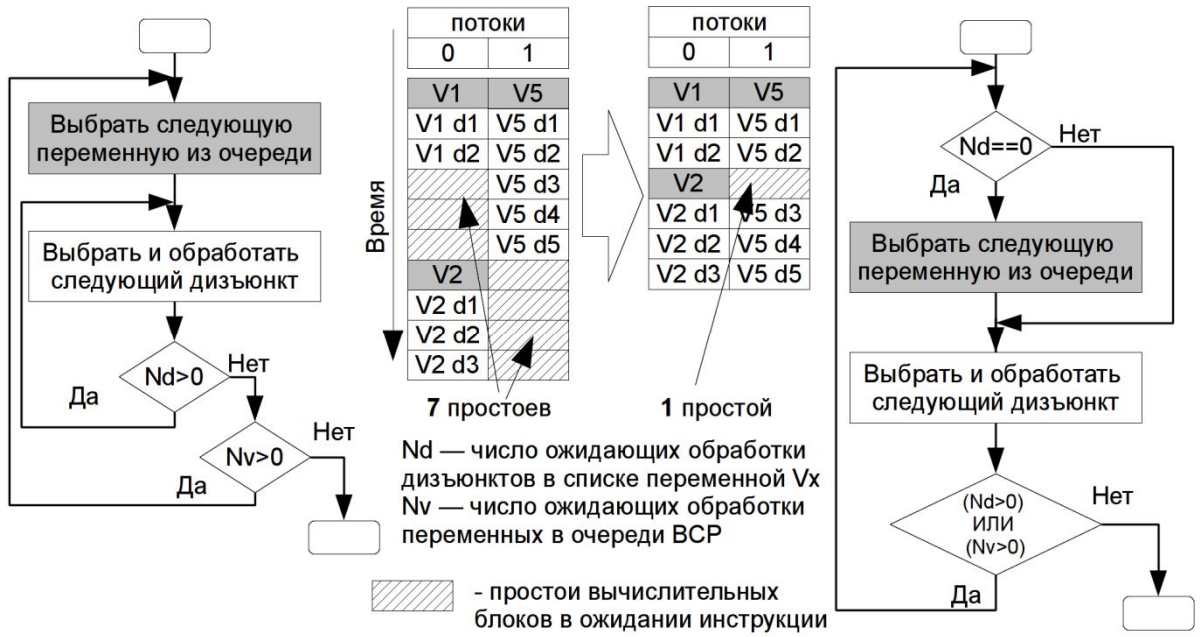


Рис. 2. Уменьшение степени вложенности циклов (на примере алгоритма DPLL)

Для того чтобы уменьшить потери производительности от SIMD-эффекта, можно реорганизовать вложенные циклы таким образом, чтобы уменьшить степень их вложенности. Пример такой трансформации для алгоритма DPLL приведен на рис. 2. Основная ее идея состоит в том, чтобы использовать обратную дугу внешнего цикла в качестве обратной дуги внутреннего цикла, тем самым оставив лишь одну обратную дугу и один естественный цикл. Проверка необходимости повтора внешнего цикла объединяется с той же проверкой для внутреннего цикла. Операции, относящиеся к циклу верхнего уровня, при этом переставляются под условный переход-предикат. Полученный в результате граф потока управления не будет эквивалентен исходному, но при выполнении алгоритма последовательность базовых блоков всегда будет совпадать с первоначальной. Эта трансформация должна положительно сказываться на производительности не только GPU, но и современных суперскалярных CPU (упрощается работа блока предсказания ветвлений).

В нашей реализации DPLL с помощью этой техники были объединены циклы обработки очереди переменных и цикл обработки дизъюнктов одной переменной. Это позволило повысить производительность DPLL на GPU на 30–50 %. В целом применение этой и других подобных техник реорганизации условных переходов позволило увеличить производительность DPLL на GPU в 2–5 раз (в зависимости от характера решаемой задачи).

2.2. Управление условными переходами в SIMD с помощью системы «голосований»

Объединение циклов может привести и к ухудшению производительности приложения на SIMD-архитектуре (рис. 3).

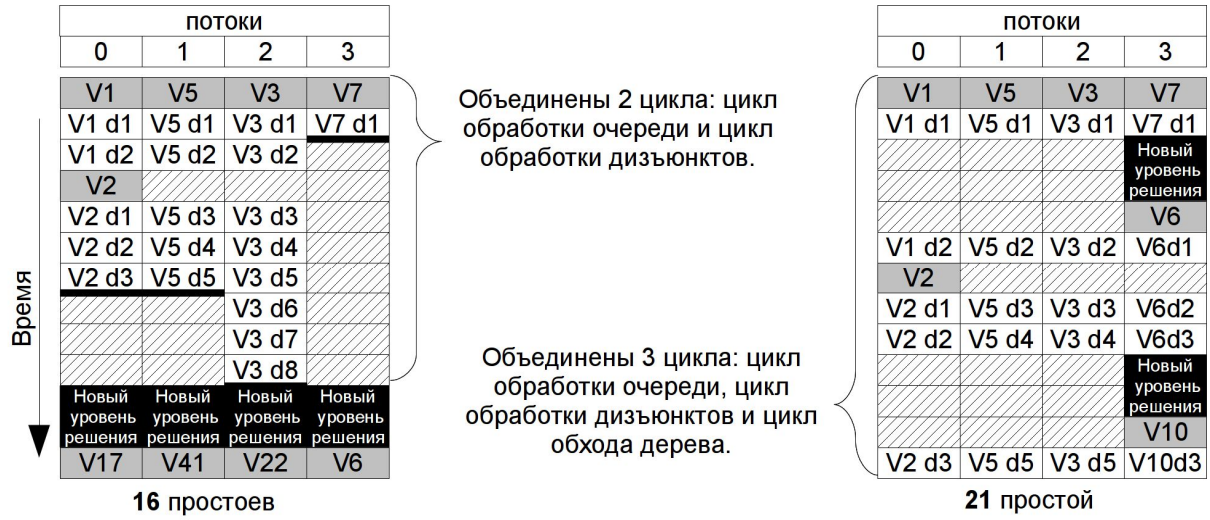


Рис. 3. Потери производительности из-за избыточного объединения циклов (на примере DPLL)

Преимущества от более полного использования АЛУ при обработке команд внутреннего цикла могут быть нивелированы простоями при обработке стоящих под предикатом команд внешнего цикла, если их выполнение требует слишком много времени. Выясним, в каких случаях одной SIMD-группе выгодно перейти к обработке внешнего цикла. Введем следующие обозначения:

T_a — время, выполнения инструкций внешнего цикла (цикла «a»), необходимых для обеспечения потоков заданием для внутреннего цикла;

T_b — время, необходимое для выполнения инструкций одной итерации внутреннего цикла (цикла «b»);

N — общее число потоков (в большинстве случаев равно размеру варпа);

n — число остановившихся потоков, закончивших задания во внутреннем цикле и готовых перейти ко внешнему циклу, чтобы получить новые задания;

$P_{iter}(b,N)$ — наиболее вероятное число итераций внутреннего цикла, после которого хотя бы у одного из N потоков закончится задание;

$T_b^{'} = T_b P_{iter}(b,N)$ — среднее время которое проведут потоки на внутреннем цикле, если у всех N потоков будут задания;

$(N-n)T_a$ — цена получения новых заданий остановившимися потоками (т.е. простои при переходе ко внешнему циклу тех потоков, у которых работа еще есть).

Прерывать обработку внутреннего цикла и выходить на внешний цикл имеет смысл только в том случае, если от этого будет выигрыш в работе:

$$nT_b^{'} - (N-n)T_a > 0.$$

Преобразовав это выражение, получим:

$$\frac{n}{N} > \frac{T_a}{T_a + T_b^{'}}.$$

Компилятор и диспетчер потоков мультипроцессора не знают ничего о средней продолжительности внутреннего цикла, которая, вообще говоря, зависит от решаемой задачи. Как правило, диспетчер действует по правилу «большинства голосов» — переходит

по той ветви условного перехода, которую требует в текущий момент большинство активных потоков. Очевидно, что такая стратегия не всегда будет наиболее эффективной.

Поэтому мы реализовали «ручное» управление выходом в цикл обхода дерева из цикла BCP в DPLL. Для этого мы применили стратегию «голосований» (рис. 4) с использованием встроенных инструкций GPU Fermi [4]. Результаты голосования внутри SIMD-группы сравниваются с пороговым значением, при достижении которого группа выходит на внешний цикл, чтобы простаивающие потоки могли получить новое задание. Голосование проводится каждый раз, когда в цикле BCP у одного из потоков заканчивается работа. Эксперименты с различными классами задач и значениями порога голосования показали, что порог должен подбираться индивидуально, в зависимости от характера решаемой задачи. В наших экспериментах в отдельных случаях удачно подобранное значение порога позволяло добиться ускорения работы решателя до 15 %. Интересным развитием этой техники может стать реализация динамического изменения порога, в зависимости от накопленной во время решения задачи статистики. По сути, управление значением порога является формой механизма предсказания ветвлений. Важно заметить, что метод голосования не является специфическим для DPLL, он может применяться в любых алгоритмах со сложной структурой вложенных циклов.

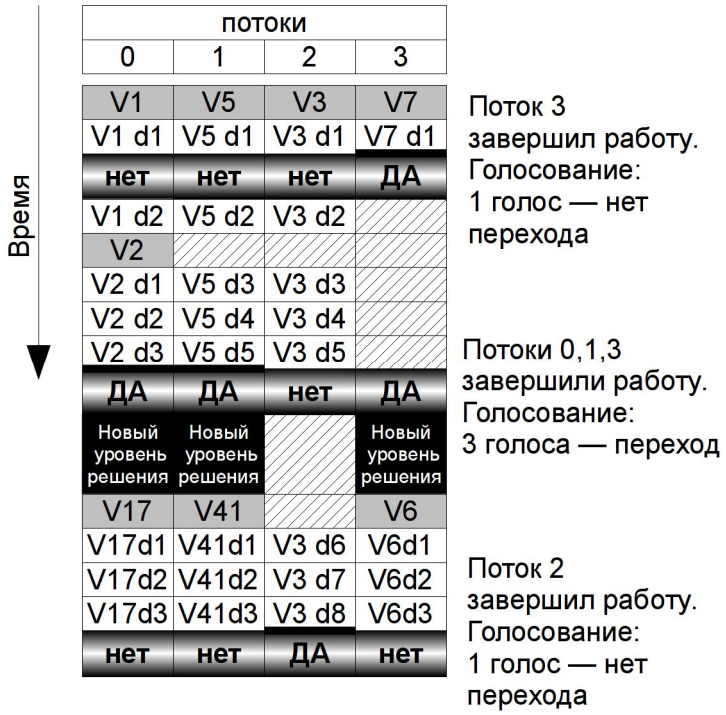


Рис. 4. Стратегия «голосований» за условный переход в SIMD (на примере DPLL)

3. Особенности доступа к памяти в GPU

Каждый мультипроцессор GPU способен обрабатывать несколько SIMD-групп одновременно, динамически переключаясь между ними. Это позволяет компенсировать латентность основной памяти устройства: в каждый момент времени обрабатывается та SIMD-группа, данные для исполнения команд которой уже доставлены из основной памяти устройства. Если данные для текущей группы еще не готовы, мультипроцессор переключается на обработку следующей группы, и т.д. Чем больше одновременно запущено на мультипроцессоре SIMD-групп — тем лучше удается скрыть латентность памя-

ти, тем полнее используется доступная пропускная способность памяти (ПСП). Возможность работы нескольких SIMD-групп на одном мультипроцессоре ограничена его ресурсами: регистровой памятью («registry file») и общей памятью («shared memory») [4].

3.1. Группировка запросов к памяти

В случае SIMD-архитектур обработка данных производится пакетами, состоящими из нескольких одинаковых элементов, предназначенных различным выч. потокам одной SIMD-группы. При этом запросы к памяти от всех 32-х потоков SIMD-группы должны укладываться в границы одной кэш-линии кэша L2 (128 байт для Fermi) — тогда они будут доставлены мультипроцессору в виде единого пакета данных [4]. В противном случае происходит сериализация обращений к памяти, создающая до 32 запросов вместо одного (рис. 5). Этот «эффект разреженного доступа» («uncoalesced access») приводит к кратному уменьшению эффективной ПСП.

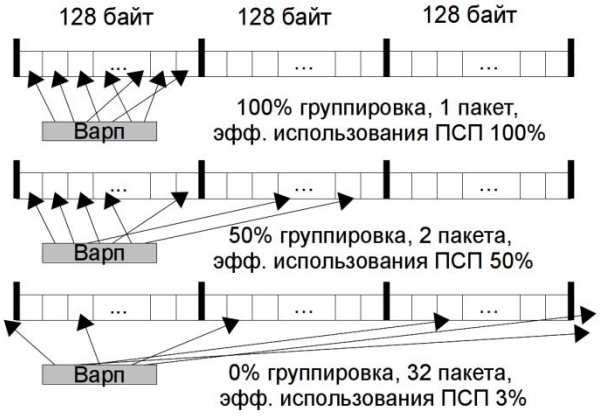


Рис. 5. Группировка запросов к памяти GPU

Для того чтобы запросы к памяти от одной SIMD-группы всегда выполнялись максимально эффективно, достаточно чтобы соблюдалось 2 условия:

Тип данных элементов массива должен быть выровнен в соответствии с требованиями устройства.

При выполнении инструкции потоки одной SIMD-группы должны обращаться к элементам массива, последовательно расположенным в памяти устройства.

Для описанных выше атак полного перебора (в применении к A5/1 и DES) эффект разреженного доступа не создает потерь эффективности, поскольку в этих алгоритмах рабочие данные потоков умещаются в регистрах и кэше L1, и не требуют хранения в основной памяти устройства.

В DPLL последовательность обращений к элементам массивов основной памяти заранее предсказать невозможно, поэтому разреженный доступ имеет вероятностный характер и может привести к весьма существенным потерям эффективности.

3.2. Сравнение характеристик кэш-памяти и основной памяти GPU и CPU

Для компенсации эффектов фон Неймановского ограничения [18] в современных CPU и GPU применяется кэш-память, расположенная непосредственно на кристалле

устройства. Объем, скорость и алгоритмы работы кэш-памяти в CPU и GPU сравнимы друг с другом, однако на практике кэш-память выполняет в них разные роли. Например, в GPU Nvidia, построенных на основе архитектуры Fermi, кэш-память 1-го уровня выделяется из расчета 16 Кбайт (или 48 Кбайт) на один блок (одну или несколько SIMD-групп, выполняемых на одном мультипроцессоре). Такой объем кэш-памяти 1-го уровня сравним с объемом кэш-памяти 1-го уровня в CPU Intel архитектуры Nehalem. Однако, в пересчете на 1 вычислительный поток (при 32 потоках в блоке) объем ее составит в лучшем случае 1,5 Кбайт. Этого достаточно лишь для того, чтобы частично компенсировать нехватку локальных регистров мультипроцессора, используемых для хранения промежуточных результатов вычислений (т.н. «registers spilling» [4]).

Кэш-память 2-го уровня в Fermi составляет 768 Кбайт и является общей для всего устройства. Ее объем сравним с объемом кэшей 2-го и 3-го уровней в Nehalem. Однако, в случае 2048 потоков, запущенных на GPU GTX 560Ti, на один поток придется всего 0,375 Кбайт этой кэш-памяти. Очевидно, что кэш-память 2-го уровня в архитектуре Fermi не может вместить «горячие» данные всех потоков. Она используется в основном для ускорения доступа к небольшому объему данных, одновременно используемых всеми потоками, и для обработки инструкций атомарного доступа к памяти. В случаях, когда обращения вычислительных потоков к глобальной памяти GPU не локализованы, наличие кэш-памяти L2 не дает почти никаких преимуществ.

Таблица 1

Сравнение подсистем памяти GPU и CPU[3]

Уровень подсистемы памяти	**CPU (Intel Nehalem)**			**GPU (Nvidia Fermi)**	
	Объем (байт)	**ПСП 128 бит (байт/с)**	**ПСП 32 бита (байт/с)**	**Объем (байт)**	**ПСП 32–1024 бита[4] (байт/с)**
Кэш L1	$32{\cdot}10^3$	$45{\cdot}10^9$	$11{\cdot}10^9$	$16/48{\cdot}10^3$	$\approx 8{\cdot}10^{12}$
Кэш L2	$256{\cdot}10^3$	$31{\cdot}10^9$	$8{\cdot}10^9$	$768{\cdot}10^3$	$16–520{\cdot}10^9$
Кэш L3	$8{\cdot}10^6$	$26{\cdot}10^9$	$6{\cdot}10^9$	–	–
Основная память	$1–24{\cdot}10^9$	$20{\cdot}10^9$	$5{\cdot}10^9$	$1–2{\cdot}10^9$	$4–128{\cdot}10^9$

Данные, представленные в табл. 1 показывают, что алгоритм, скорость которого ограничена ПСП в режиме передачи 32-битных слов и хранящий данные в основной памяти устройства, покажет 0,8–21-кратное ускорение при переносе с CPU Nehalem на GPU Fermi, в зависимости от того, насколько сгруппированными окажутся обращения к памяти при решении конкретной задачи. «Дружественность» структур данных к кэш-памяти и особенности конкретной задачи могут улучшить эту оценку.

До 95 % времени работы алгоритма DPLL занимает процедура BCP. В основном она состоит из непредсказуемой последовательности обращений к массивам литералов, watched-флагов и состояний переменных. Поэтому скорость DPLL ограничена, как пра-

[3] Вместительность и ПСП кэшей и основной памяти CPU и GPU приведены на примере архитектур Intel Nehalem и GPU Nvidia Fermi. ПСП для CPU приведены для 1 потока/ядра [19], для GPU — для всего устройства целиком (оценены по материалам [4]).

[4] В зависимости от степени выраженности описанного выше «эффекта разреженного доступа».

вило, латентностью памяти в режиме произвольного доступа. Если же латентность памяти не является сдерживающим фактором, производительность алгоритма будет зависеть от реальной ПСП устройства в режиме передачи коротких слов (1–32 бит).

В DPLL при 2048 потоках, запущенных на GPU занимаемый рабочими данными потоков, объем памяти устройств, составит около 300 Мбайт, в зависимости от числа литералов и переменных в КНФ. Поскольку «горячая область» памяти в каждой подзадаче/потоке будет разной, маловероятно, что наличие кэша L2 существенно повлияет на производительность устройства.

В описанных выше алгоритмах криптоанализа DES полным перебором кэш L1 активно используется для компенсации нехватки регистров. В случае же A5/1 наличие у GPU кэша L2 заметно влияет на производительность, поскольку он способен вместить большую часть данных, к которым обращаются все потоки во время работы.

4. Экспериментальная проверка производительности DPLL на GPU

Для того, чтобы сравнить эффективность механизмов использования памяти в CPU и GPU мы построили CPU-версию нашего GPU-решателя (с теми же структурами данных и алгоритмическими особенностями, которые были описаны выше). При его тестировании на SAT-задачах, кодирующих криптоанализ A5/1, было обнаружено, что все 100% запросов к памяти обслуживаются из кэша. Аналогичная картина наблюдается и для сторонних решателей (например, minisat [20]). При работе с памятью GPU на тех же самых тестовых задачах мы наблюдаем совершенно иную картину.

Основной мерой производительности для DPLL было выбрано число просмотров индивидуальных вхождений литералов в дизъюнкты. Эта мера совместима не только между различными аппаратными платформами, но и между различными решателями на базе DPLL и дает хорошее представление о базовой скорости работы алгоритма. Для обеспечения повторяемости результатов work stealing отключался. Чтобы обеспечить все потоки равномерной нагрузкой решатель принудительно выводился в область поиска, в которой ни одна из розданных потокам подзадач не решалась за время эксперимента.

В качестве меры производительности алгоритма прямого перебора было выбрано число ключей соответствующего шифра (DES или A5/1), проверяемых в секунду. Эта мера также является совместимой между различными аппаратными платформами и реализациями алгоритма прямого перебора.

4.1. Влияние SIMD-эффекта и эффекта разреженного доступа на производительность DPLL на GPU

Продемонстрируем, как группировка запросов к памяти и SIMD-эффект влияют на эффективность выполнения DPLL на GPU. Для этой цели мы специально построили различные тестовые версии решателя, в которых искусственно вводились негативные и позитивные факторы, влияющие на сериализацию запросов к памяти и инструкций в SIMD-группе, и, как следствие — на производительность приложения на GPU (табл. 2).

Опишем модификации, применявшиеся для получения тестовых вариантов приложения:

1) Нормальная/перемешанная индексация памяти. Если случайным образом перемешать номера вычислительных потоков между SIMD-группами, SIMD-группе при-

дется «собирать» данные из разбросанных областей памяти, что гарантированно спровоцирует эффект разреженного доступа.

2) Одинаковые/разные подзадачи в SIMD-группе. Отдавая всем потокам внутри SIMD-группы на решение одну и ту же подзадачу, можно гарантировать отсутствие SIMD-эффекта, т.к. последовательность выполнения потоками инструкций при одинаковых начальных условиях строго детерминирована.
3) Параллельное/последовательное выполнение инструкций в SIMD-группе. Для гарантированного получения 100 % SIMD-эффекта, достаточно сделать так, чтобы внутри одной SIMD-группы алгоритм, выполнялся потоками последовательно: вначале код алгоритма полностью выполняется только одним потоком, затем полностью выполняется другим потоком и т.д.

В табл. 2 представлены 8 тестовых вариантов приложения, реализующих все возможные сочетания описанных выше модификаций нашей реализации DPLL для GPU. Тестирование производилось на КНФ кодирующей задачу криптоанализа DES. Обмены задачами между потоками (work stealing) были выключены. Порядок угадывания переменных в различных подзадачах выбирался случайным образом, что соответствует состоянию решателя после продолжительной работы. В качестве меры производительности использовалась скорость просмотра литералов решателем в процедуре BCP.

Таблица 2

Производительность DPLL на GPU в зависимости от режима доступа к памяти и SIMD-эффекта.

Тестовый вариант приложения	**№ 1**	**№ 2**	**№ 3**	**№ 4**	**№ 5**	**№ 6**[5]	**№ 7**	**№ 8**
Индексация памяти: (+) нормальная, (−) перемешанная.	−	+	−	+	−	+	−	+
Подзадачи в SIMD-группе: (+) одинаковые, (−) разные.	−	−	+	+	−	−	+	+
Обр. инструкций: (+) паралл., (−) последовательная.	−	−	−	−	+	+	+	+
(+) 100 % эффективность доступа к памяти.	−	−	−	−	−	−	−	+
(+) 100 % отсутствие «SIMD-эффекта».	−	−	−	−	−	−	+	+
Производительность DPLL, млн. просмотров в с.	17,5	17,5	17,0	15,6	28,9	31,2	37,6	471,3

Из данных табл. 2 видно, что только при 100 % эффективности доступа к памяти (вариант № 8) DPLL демонстрирует высокий уровень производительности на GPU. Вариант № 8 в десятки раз опережает другие тестовые варианты, включая № 6, соответствующий рабочему режиму решателя. 100 % эффективность доступа к памяти в варианте № 8 является следствием полного отсутствия сериализации обращений к памяти в результате эффекта разреженного доступа, SIMD-эффекта, или по иным причинам. Если бы на производительность DPLL на GPU негативно влиял только «SIMD-эффект», разница в производительности между вариантами № 7 и № 8 не была бы столь велика.

[5] Тестовый вариант №6 соответствует рабочему режиму решателя.

Непредсказуемость пути поиска в DPLL, перемешанная индексация памяти, последовательное выполнение инструкций — любой из этих факторов приводит к 100 % сериализации обращений к памяти, что буквально обрушивает производительность GPU, снижая ее в десятки раз. Следует обратить внимание на то, что искусственное форсирование 100% сериализации описанными выше способами слабо влияет на скорость выполнения DPLL на GPU в рабочем режиме (переход от № 6 к № 7). Это говорит о том, что обращения к памяти в рабочем режиме DPLL совершенно хаотичны, вызывают сильнейший «эффект разреженного доступа» и, как следствие, обрабатываются почти на 100 % в серийном режиме. Фактически, это сводит на нет все преимущества SIMD-архитектуры.

Кроме того, в силу сложности алгоритма, скомпилированный для GPU код DPLL использует довольно много регистровой памяти. Из-за этого нам пришлось искать компромисс между запуском числа SIMD-групп, достаточного для компенсации латентности памяти (высокий показатель «occupancy»), и эффектом переполнения регистровой памяти («register spilling») [4]. Это означает, что требования DPLL к числу регистров слишком велики для GPU поколения Fermi.

Таким образом, архитектура памяти современных GPU совершенно не подходит для выполнения сложных комбинаторных алгоритмов, таких как DPLL, в силу следующих причин:

1) Латентность доступа к основной памяти GPU высока.
2) Низкое число регистров в мультипроцессорах не позволяет запускать на них достаточно SIMD-групп, чтобы компенсировать латентность доступа к памяти за счет переключения между ними.
3) Контроллер доступа к памяти в GPU рассчитан на выборку сплошных диапазонов памяти, лежащих в пределах одной кэш-линии и передачу этой информации мультипроцессору целиком. Однако, при решении реальных задач с помощью алгоритма DPLL вероятность того, что хотя бы 2 потока одной SIMD-группы обратятся одновременно к одной кэш-линии, крайне мала. На практике это приводит к очень высокому эффекту разреженного доступа и падению эффективной ПСП в десятки раз.
4) Малый объем кэш-памяти GPU не позволяет компенсировать проблемы с ПСП и латентностью памяти.

4.2. Сравнение производительности алгоритмов полного перебора и DPLL на CPU и GPU

В табл. 3 приведены результаты тестирования скорости криптоанализа генератора A5/1 и блочного шифра DES на GPU и CPU, реализованного с помощью алгоритмов прямого перебора и SAT-подхода (алгоритм DPLL). Для алгоритмов прямого перебора скорость оценивалась как число ключей, проверяемых за единицу времени («кл/с»).

В силу применения алгоритма work stealing поведение решателя на GPU не является строго детерминированным. Поэтому в качестве показателя скорости работы алгоритма DPLL, совместимого между GPU и CPU, использовалось число просмотров литералов в дизъюнктах в процедуре BCP за единицу времени («п/с»). При этом в версии DPLL для GPU число просмотров литералов суммировалось по всем потокам.

Для тестирования были использованы CPU Intel Core i7 930 и GPU Nvidia Geforce GTX560 Ti. На CPU многопоточность не использовалось.

Также, для оценки потенциальных вычислительных возможностей GPU, каждый алгоритм был протестирован в специальной версии, в которой все вычислительные потоки устройства выполняли одинаковую подзадачу, независимо друг от друга. В таком случае доступ к памяти организован полностью в соответствии с рекомендациями производителя, и потери производительности из-за условных переходов сведены к нулю (100 % «memory coalescence» и нулевой «warp divergence» в терминологии NVIDIA [4]).

Таблица 3

Сравнение скорости комбинаторных алгоритмов на CPU и GPU

Алгоритм	**CPU**	**GPU**	**Коэфф. ускорения**	**GPU тест, одинак. подзадачи.**	**Коэфф. ускорения**
Брутфорс A5/1	48 млн. кл/с	9761 млн. кл/с	203,3	12271 млн. кл/с	255,6
Брутфорс DES	12 млн. кл/с	1493 млн. кл/с	124,4	1908 млн. кл/с	159,0
DPLL A5/1	21 млн. п/с	84 млн. п/с	4,0	650 млн. п/с	30,9
DPLL DES	25 млн. п/с	195 млн. п/с	5,0	1053 млн. п/с	42,1

Данные, представленные в табл. 3 показывают, что алгоритм DPLL не выигрывает от переноса на GPU, в отличие от более простых способов криптоанализа.

Заключение

Архитектура современных GPU плохо приспособлена для выполнения DPLL и подобных ему сложных поисковых алгоритмов. Причины этого — проблемы с условными переходами и произвольным доступом к памяти. И хотя техники реорганизации вложенных циклов и «голосований», описанные нами в разделе 2, позволяют уменьшить негативные последствия «SIMD-эффекта», основным «узким местом» DPLL на GPU остается произвольный доступ к памяти. С другой стороны, простые комбинаторные алгоритмы, такие как метод прямого перебора, получают значительное преимущество от переноса на GPU.

Современные тенденции в развитии GPU включают: сохранение SIMD-архитектуры, использование векторных инструкций и соответствующих им «широких» типов данных, компенсация фон Неймановского ограничения за счет наращивания кэш-памяти, увеличение ПСП в ущерб латентности доступа к памяти. Таким образом, нет оснований предполагать, что указанные препятствия к реализации «интеллектуальных» комбинаторных алгоритмов на GPU будут в ближайшие годы устранены производителями оборудования.

Работа была поддержана грантом РФФИ № 14-07-00403-а.

Литература

1. TOP500 Supercomputer Site URL: http://www.top500.org (дата обращения: 01.01.2015).

2. Lee, V.W. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU / V.W. Lee et al // ACM SIGARCH Computer Architecture News. — ACM, 2010. — Vol. 38, No. 3. — P. 451–460. DOI: 10.1145/1816038.1816021.
3. Flynn, M. Some computer organizations and their effectiveness / M. Flynn // Computers, IEEE Transactions on. — 1972. — Vol. 100, No. 9. — P. 948–960. DOI: 10.1109/tc.1972.5009071.
4. CUDA C Best Practices Guide — CUDA SDK v.6.0 — NVIDIA corp. — 2014. URL: http://docs.nvidia.com/cuda (дата обращения: 15.07.2014).
5. Percival, C. The scrypt Password-Based Key Derivation Function. — IETF Draft — 2012 / C. Percival, S. Josefsson URL: http://tools.ietf.org/html/josefsson-scrypt-kdf-00.txt (дата обращения: 30.11.2012).
6. Nohl, K. Attacking phone privacy / K. Nohl // Black Hat USA. — 2010. URL: https://srlabs.de/blog/wp-content/uploads/2010/07/Attacking.Phone_.Privacy_Karsten.Nohl_1.pdf (дата обращения: 01.01.2015).
7. Mironov, I. Applications of SAT solvers to cryptanalysis of hash functions / I. Mironov, L. Zhang // Theory and Applications of Satisfiability Testing-SAT 2006. — Springer, 2006. — P. 102–115. DOI: 10.1007/11814948_13.
8. Semenov, A. Parallel logical cryptanalysis of the generator A5/1 in BNB-Grid system / A. Semenov et al. // Parallel Computing Technologies. — Springer, 2011. — P. 473–483. DOI: 10.1007/978-3-642-23178-0_43.
9 Biryukov, A. Real Time Cryptanalysis of A5/1 on a PC / A. Biryukov, A. Shamir, D. Wagner // Fast Software Encryption. — Springer, 2001. — P. 1–18. DOI: 10.1007/3-540-44706-7_1.
10 Golic, J.D. Cryptanalysis of alleged A5 stream cipher / J.D. Golic // Advances in Cryptology—EUROCRYPT'97. — Springer, 1997. — P. 239–255. DOI: 10.1007/3-540-69053-0_17.
11 Kumar, S. Breaking ciphers with COPACOBANA — a cost-optimized parallel code breaker / S. Kumar et al. // Cryptographic Hardware and Embedded Systems-CHES 2006. — Springer, 2006. — P. 101–118. DOI: 10.1007/11894063_9.
12 Kwan, M. Reducing the Gate Count of Bitslice DES / M. Kwan // IACR Cryptology ePrint Archive. — 2000. — Vol. 2000. — P. 51.
13 John the Ripper password cracker — 2013 URL: http://www.openwall.com/john (дата обращения: 02.07.2014).
14 Davis, M. A machine program for theorem-proving / M. Davis, G. Logemann, D. Loveland // Communications of the ACM. — 1962. — Vol. 5, No. 7. — P. 394–397. DOI: 10.1145/368273.368557.
15 Moskewicz, M.W. Chaff: Engineering an efficient SAT solver / M.W. Moskewicz et al. // Proceedings of the 38th annual Design Automation Conference. — ACM, 2001. — P. 530–535. DOI: 10.1109/dac.2001.935565.
16 Marques-Silva, J.P. GRASP: A search algorithm for propositional satisfiability / J.P Marques-Silva, K.A. Sakallah // Computers, IEEE Transactions on. — 1999. — Vol. 48, No. 5. — P. 506–521. DOI: 10.1109/12.769433.
17 Blumofe, R.D. Scheduling multithreaded computations by work stealing / R.D. Blumofe, C.E. Leiserson // Journal of the ACM (JACM). — 1999. — Vol. 46, No. 5. — P. 720–748. DOI: 10.1145/324133.324234.

18 Backus, J. Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs / J. Backus // Communications of the ACM. — 1978. — Vol. 21, No. 8. — P. 613–641. DOI: 10.1145/359576.359579.

19 Molka, D. Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system / D. Molka et al. // PACT'09. 18th International Conference on Parallel Architectures and Compilation Techniques. — IEEE, 2009. — P. 261–270. DOI: 10.1109/pact.2009.22.

20 Een, N. MiniSat: A SAT solver with conflict-clause minimization / N. Een, N. Sörensson // Proceedings of the International Symposium on the Theory and Applications of Satisfiability Testing (2005). — 2005. — Vol. 5. — P. 55.

Булавинцев Вадим Германович, программист, лаборатория дискретного анализа, Институт динамики систем и теории управления им. В.М. Матросова СО РАН (Иркутск, Российская Федерация), v.g.bulavintsev@gmail.com.

Поступила в редакцию 10 апреля 2015 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2015, vol. 4, no. 3, pp. 67–84

DOI: 10.14529/cmse150306

AN EVALUATION OF CPU VS. GPU PERFORMANCE OF SOME COMBINATORIAL ALGORITHMS FOR CRYPTOANALYSIS

V.G. Bulavintsev, Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences (Irkutsk, Russian Federation)
v.g.bulavintsev@gmail.com

In this work we assess performance of CPU and GPU implementations of some widely-used cryptanalytic combinatorial algorithms. In particular, we analyze obstacles for effective GPU implementation of "smart" combinatorial algorithms. Next, to alleviate performance problems arising from inefficient processing of conditional expressions in SIMD-devices we devise some special control flow graph transformation techniques. Finally, we demonstrate that contemporary GPU's memory access schemes are incompatible with typical memory access patterns of "smart" combinatorial algorithms studied. We use DES and A5/1 cryptographic functions as test cases.

Keywords: GPU, CUDA, cryptoanalysis, DPLL, SAT, SIMD.

References

1. TOP500 Supercomputer Site URL: http://www.top500.org (accessed: 01.01.2015).
2. Lee V.W. et al. Debunking the 100X GPU vs. CPU Myth: an Evaluation of Throughput Computing on CPU and GPU // ACM SIGARCH Computer Architecture News. ACM, 2010. Vol. 38, No. 3. P. 451–460. DOI: 10.1145/1816038.1816021.

3. Flynn M. Some Computer Organizations and Their Effectiveness // Computers, IEEE Transactions on. 1972. Vol. 100, No. 9 P. 948–960. DOI: 10.1109/tc.1972.5009071.
4. CUDA C Best Practices Guide CUDA SDK v.6.0 NVIDIA corp. 2014. URL: http://docs.nvidia.com/cuda (accessed: 15.07.2014).
5. Percival C., Josefsson S. The scrypt Password-Based Key Derivation Function. IETF Draft URL: http://tools.ietf.org/html/josefsson-scrypt-kdf-00.txt (accessed: 30.11.2012).
6. Nohl K. Attacking phone privacy // Black Hat USA. 2010. URL: https://srlabs.de/blog/wp-content/uploads/2010/07/ Attacking.Phone_.Privacy_Karsten.Nohl_1.pdf (accessed: 01.01.2015).
7. Mironov I., Zhang L. Applications of SAT Solvers to Cryptanalysis of Hash Functions // Theory and Applications of Satisfiability Testing-SAT 2006. Springer, 2006. P. 102–115. DOI: 10.1007/11814948_13.
8. Semenov A. et al. Parallel Logical Cryptanalysis of the Generator A5/1 in BNB-Grid System // Parallel Computing Technologies. Springer, 2011. P. 473–483. DOI: 10.1007/978-3-642-23178-0_43.
9. Biryukov A., Shamir A., Wagner D. Real Time Cryptanalysis of A5/1 on a PC // Fast Software Encryption. Springer, 2001. P. 1–18. DOI: 10.1007/3-540-44706-7_1.
10. Golić J.D. Cryptanalysis of Alleged A5 Stream Cipher // Advances in Cryptology—EUROCRYPT'97. Springer, 1997. P. 239–255. DOI: 10.1007/3-540-69053-0_17.
11. Kumar S. et al. Breaking Ciphers With COPACOBANA — a Cost-Optimized Parallel Code Breaker // Cryptographic Hardware and Embedded Systems-CHES 2006. Springer, 2006. P. 101–118. DOI: 10.1007/11894063_9.
12. Kwan M. Reducing the Gate Count of Bitslice DES IACR Cryptology ePrint Archive. 2000. Vol. 2000. P. 51.
13. John the Ripper password cracker 2013 URL: http://www.openwall.com/john/ (accessed: 02.07.2014).
14. Davis M., Logemann G., Loveland D. A Machine Program for Theorem-Proving // Communications of the ACM. 1962. Vol. 5, No. 7. P. 394–397. DOI: 10.1145/368273.368557.
15. Moskewicz M.W. et al. Chaff: Engineering an Efficient SAT Solver // Proceedings of the 38th Annual Design Automation Conference. ACM, 2001. P. 530–535. DOI: 10.1109/dac.2001.935565.
16. Marques-Silva J.P., Sakallah K.A. GRASP: A Search Algorithm for Propositional Satisfiability // Computers, IEEE Transactions on. 1999. Vol. 48, No. 5. P. 506–521. DOI: 10.1109/12.769433.
17. Blumofe R.D., Leiserson C. E. Scheduling Multithreaded Computations by Work Stealing // Journal of the ACM (JACM). 1999. Vol. 46, No. 5. P. 720–748. DOI: 10.1145/324133.324234.
18. Backus J. Can Programming Be Liberated from the von Neumann Style?: a Functional Style and Its Algebra of Programs // Communications of the ACM. 1978. Vol. 21, No. 8. P. 613–641. DOI: 10.1145/359576.359579.
19. Molka D. et al. Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System // PACT'09. 18th International Conference on. Parallel Architectures and Compilation Techniques. IEEE, 2009. P. 261–270. DOI: 10.1109/pact.2009.22.
20. Een N., Sörensson N. MiniSat: A SAT Solver with Conflict-Clause Minimization // Proceedings of the International Symposium on the Theory and Applications of Satisfiability Testing (2005). 2005. Vol. 5. P. 55.

Received April 10, 2015.

УДК 519.63 DOI: 10.14529/cmse150307

О ПРОГРАММНЫХ КОМПОНЕНТАХ МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ[1]

В.П. Ильин

Бурный рост мировых суперкомпьютерных мощностей неизбежно ведет к активизации применений математического моделирования в процессе реиндустриализации экономики. Расширение сфер использования и одновременное появление все новых вычислительных методов, сопровождающиеся эволюцией компьютерных архитектур — все это ведет к драматическому увеличению совокупного прикладного программного обеспечения, стоимость которого сравнима с ценами на многопроцессорные вычислительные системы (МВС) постпетафлопсного уровня. Такие количественные изменения влекут за собой качественный пересмотр концепции и технологических структур при создании массовых программных продуктов. Мы рассматриваем подходы к решению возникающих проблем в применении к базовой системе моделирования, ориентированной на интегрированную поддержку всех основных этапов крупномасштабного вычислительного эксперимента. Многообразие технологических вопросов включает обеспечение взаимодействия многоязыковых и кросс-платформенных программных компонент, внутренние и внешние интерфейсы с конвертацией множественных представлений данных, переиспользование внешних продуктов, длительный жизненный цикл системы с эволюцией функционального наполнения и адаптацией к модернизации МВС, а также масштабируемый параллелизм алгоритмов и программ.

Ключевые слова: базовая система моделирования, компонентные технологии, высокопроизводительные вычисления, масштабируемое распараллеливание, многоязыковость, кросс-платформенность, прямые и обратные междисциплинарные задачи.

Введение

Математическое моделирование реальных процессов и явлений актуализируется, с одной стороны, бурным ростом мировых суперкомпьютерных ресурсов, знаменующим наступление эпохи постиндустриализации, а с другой, менее заметной,— появлением новой вычислительной математики и информатики, открывающей замечательные перспективы в принципах получения фундаментальных знаний и достижения технологического прогресса. При этом, как отмечалось в [1, 2] и цитируемых там работах, характерная тенденция развития современного прикладного и программного обеспечения заключается в его «глобализации», т.е. в переходе от разработок пакетов прикладных программ (ППП, например, ANSYS [3]) или библиотек, ориентированных на конкретные типы задач или представляющих собой наборы вспомогательных алгоритмических инструментов, к интегрированным программным окружениям для решения широкого класса прикладных проблем, поддерживающим все основные стадии математического моделирования и рассчитанным на длительный жизненный цикл с непрерывным пополнением состава моделей, методов и технологий, на адаптацию к эволюции архитектур и конфигураций гетерогенных МВС, а также на скоординированную реализацию различными группами разработчиков. Примерами таких проектов являются FOAM [4], DUNE [5] и БСМ [1]. Если же говорить о частных предметных областях, то здесь следует отметить унифицированный подход, реализованный в базовых программных системах для задач линейной алгебры BLAS и SPARSE BLAS, средства которых успешно используются в огромном количестве прикладных разработок.

[1] Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии – 2015».

С прикладной, или предметной, точки зрения, особенностью проблемы является, с одной стороны, нацеленность на очень широкий класс прямых и обратных междисциплинарных задач, а с другой — избыточное многообразие включаемых в функциональное наполнение вычислительных методов и технологий, что позволяет выбрать в конкретном приложении наиболее подходящий или даже оптимальный алгоритм, устраняя или хотя бы ослабляя неизбежный конфликт между универсальностью и эффективностью.

Особенностью таких разработок является очень большой объем программных кодов и их сложная внутренняя структура, многоверсионность, а также необходимость многократных реконфигураций, гибкого управления вычислительным процессом и поддержки как межмодульных интерфейсов, так и взаимодействия с внешними продуктами. Очевидно, что такие обстоятельства требуют неизбежных временных издержек, как при разработке программных продуктов, так и при их эксплуатации, но при этом нужно сохранить достаточно высокую производительность алгоритмов и программ на гетерогенных МВС, со сложной иерархией общей и распределенной памяти.

Существующие уже несколько десятилетий широко используемые компонентные технологии, такие как CORBA (Common Object Reques Broker Architecture, [6]) и COM/DCOM [7] (Distributed Component Object Management), которые в определенном смысле представляют развитие объектно-ориентированного программирования, но в силу своих исторических рамок, недостаточно акцентированы на обеспечение высокой производительности наукоемких вычислений с масштабируемым параллелизмом, сложными типами данных и разноязычными программными модулями. С целью преодоления этих ограничений в 1997 г. был организован Common Component Architecture (CCA) Forum [8], поставивший задачей определение основных стандартов, инструментов и технологий. Департамент энергетики (DOE) в США создал Центр компонетных технологий для программного обеспечения терамасштабного моделирования (Center for Component Technology for Terascale Simulation Software, CCTTSS [9]), который совместно с ведущими Национальными лабораториями создали ряд инструментальных средств в рамках спецификаций CCA. Среди таких разработок можно отметить язык описания компонентных интерфейсов SIDL (Scientific Interface Definition Language [10]), являющейся обобщением языка IDL из проекта CORBA и послуживший основой поддержки многоязыковости, включая F77, F90/95, C, C++, Python, реализованный в группе Babel [11], а также инструментальный комплекс CCAFFEINE [12] для осуществления технологических операций в иерархической распределенной памяти. Имееется уже ряд публикаций по результатам успешного использования компонентных технологий для серьезных приложений. Таким примером может служить модернизация в методологии CCA пакета программ SPARSKIT [13] для решения задач линейной алгебры, а также разработанные в национальной лаборатории SANDIA (США) интерфейсные стандарты для проблем решения уравнений [14].

Целью данной работы является рассмотрение подходов к применению компонентных технологий для построения базовой системы моделирования [1], ориентированной на методологическую поддержку различного типа прямых и обратных междисциплинарных задач. Статья организована следующим образом. В разделе 2 мы анализируем структурные свойства ядра БСМ и ее основных функциональных блоков, с учетом классификации прикладного программного обеспечения в различных «координатных плоскостях»: по типу решаемых математических задач, по применяемым вычислительным методам, по характеру отраслевых приложений, а также по типу пользователей и способу эксплуатации. Нако-

нец, в разделе 3 формулируются характерные методологические требования к системному наполнению с точки зрения конечной эффективности и производительности создаваемого интегрируемого программного окружения, а также с учетом такого прагматического критерия, как отношение «цена/качество», подразумевая в числителе общую стоимость разработки, а в знаменателе — производительность (в некоторых условных единицах) результирующего программного кода. В заключении формулируются некоторые проблемы, обусловленные необходимостью внедрения компонентных технологий.

1. Архитектурные особенности базовой системы моделирования

Несмотря на бесчисленное разнообразие возможных вычислительных проблем и алгоритмов, в целом функциональное наполнение математического и программного обеспечения задач наукоемкого экстремального моделирования формально можно представить следующим единообразным, неформальным определением.

Данная категория включает пространственно-временные процессы и явления, описываемые начально-краевыми задачами для систем дифференциальных, интегральных и/или дискретных уравнений, а также соответствующими обобщенными или вариационными постановками. Если требуется найти решение при полностью заданных граничных и начальных условиях в расчетной области с известными материальными свойствами сред, то такая задача относится к классу прямых задач. Более сложными являются обратные задачи, в которых любые из исходных данных могут быть определены с помощью неизвестных параметров, оптимальные значения которых надо найти по условиям минимизации задаваемого целевого функционала от решения, при дополнительных линейных или нелинейных ограничениях, которые также могут описываться какими-то функционалами. Решение обратных задач подразумевает в общем случае многократное нахождение решений прямых задач, при направленном поиске необходимых значений параметров с помощью методов оптимизации. Здесь, в свою очередь, требуется разделить задачи локальной минимизации (если соответствующий изолированный в некоторой окрестности параметров минимум существует) и проблемы глобальной минимизации, т.е. нахождение всех экстремальных точек.

Под наукоемкими и экстремальными можно понимать задачи, требующие для своего понимания большого объема математических и других знаний, решение которых описывается громоздкими формулами высокой логической сложности и требующими для своей реализации сверхбольших вычислительных ресурсов (в отношении существующего на текущий момент парка компьютеров). К задачам повышенной вычислительной сложности относятся междисциплинарные и разномасштабные проблемы с контрастными материальными свойствами, в которых достижение гарантированной точности расчетов требует огромного числа степеней свободы неизвестных (порядка $10^8 - 10^{10}$).

Функциональное наполнение вычислительного ядра БСМ, в соответствии с основными стадиями математического моделирования, составляется из достаточно автономных блоков, каждый из которых представляет собой расширяемую подсистему, взаимодействующую с остальными через гибкие согласованные, допускающие множественные представления со взаимной конвертацией, структуры данных. В целом рассматриваемые этапы и архитектура ядра могут быть описаны следующим образом.

- Геометрическое и функциональное моделирование, определяющее уровень автоматизации построения математических моделей и интеллектуальности пользовательского интерфейса, а также взаимодействие с внешними, графическими и САПРовскими продуктами. Итоговая информация обо всех математических объектах, а также об их взаимосвязях, однозначно определяющая постановку исходной задачи, представляется геометрической и функциональной структурами данных (ГСД и ФСД), обеспечивающими внутренние интерфейсы с остальными компонентами ядра БСМ и взаимную конвертацию форматов, используемых внешними программными разработками.
- Генерация адаптивных неструктурированных многомерных сеток в областях со сложной конфигурацией кусочно-гладких границ, поддерживающая разнообразные типы конечных элементов, контроль качества, современные многосеточные и декомпозиционные подходы и являющаяся ключевой технологией для успешного вычислительного эксперимента в реальных задачах со сложными конфигурациями областей и контрастными свойствами материальных сред. Подчеркнем, что на данной стадии допускается применение как собственных, так и внешних генераторов сеток, а результирующая информация представляется сеточной структурой данных (ССД), которая в совокупности с ГСД и ФСД отображает свойства решаемой задачи уже на дискретном уровне.
- Аппроксимация решаемых систем дифференциальных и/или интегральных уравнений — наиболее теоретизированная стадия, отвечающая за точность численных результатов и представляющая бурно развивающуюся область математики, кардинально программистское решение которой заключается в реализации машинных символьных вычислений, что значительно повышает уровень автоматизации построения алгоритмов и имеет большое значение для повышения производительности программного труда, особенно при реализации актуальных методов высокой точности. Конечным результатом данной стадии является алгебраическая структура данных (АСД).
- Численное решение сверхбольших систем получаемых линейных и нелинейных алгебраических уравнений — наиболее ресурсоемкий этап, поскольку здесь трудоемкость алгоритмов сильно увеличивается с ростом порядка. В данном случае для обеспечения масштабируемого распараллеливания расчетов требуется эффективное отображение структуры алгоритмов на архитектуру ЭВМ, предполагающее тонкое знание особенностей работы арифметических и коммуникационных устройств.
- Оптимизационные алгоритмы условной минимизации функционалов для решения обратных задач. Эта область вычислительной математики претерпела революционные изменения в последние десятилетия, и инновационные программистские решения здесь особенно важны, поскольку каждый эксперимент связан с решениями множества прямых задач.
- Постобработка и визуализация результатов отвечает за обратную связь компьютера с человеком, важность которой трудно переоценить. При необходимости качественного графического анализа данный этап является экстремально ресурсоемким и требует массивного параллелизма. К счастью, здесь успеха решения зачастую можно достичь путем грамотного использования уже существующих профессиональных графических систем, на создание которых требуется изрядное количество человеко-лет.

Перечисленные пункты можно дополнить подсистемой принятия решений по результатам вычислительного эксперимента, что является венцом математического моделирования и должно определять его практическую востребованность.

Важно отметить, что каждый из рассмотренных блоков ядра является формально методо-ориентированным и проблемно-независимым в том смысле, что он может применяться для различных типов приложений, т.е. имеет междисциплинарный характер. В силу этого он должен представлять собой мета-алгоритм, или библиотеку программ, для решения однотипных подзадач определенного класса (например, генерации каких-либо сеток, аппроксимации различных видов и порядков для дифференциальных уравнений, решения задач вычислительной алгебры и т.д.). В проекте БСМ соответствующие программные блоки преддставлены как библиотеки вычислительных инструментариев DELAUNAY [15], CHEBYSHEV [16] и KRYLOV [17]. Важно подчеркнуть, что каждая из этих компонент — это свободно распространяемаое и расширяемое математическое окружение, включающее не только «свои» оригинальные разработки, но также открытое для использования внешних продуктов. Рассматриваемые алгоритмы могут применяться к самым разным приложениям, классифицируемым, с одной стороны, по физико-математическим признакам задачи электромагнетизма, теплофизики, упруго-пластичности, гидро-газодинамики и т.д., — описываемые соответствующими уравнениями Максвелла, массо-теплопроводности, Ламэ и Навье—Стокса. Эти уравнения могут решаться в различных расчетных областях, при разных начальных и краевых условиях, и все такое многообразие исходных математических постановок должно описываться средствами геометрического и функционального моделирования, составляющих первый из вышеперечисленных блоков ядра БСМ. С другой стороны, решаемые задачи отличаются по техническому назначению моделируемых устройств (электроника, машиностроение, металлургия, гео-электроразведка и др.). Данные спецификации определяют такую важную характеристику программного продукта, как интерфейс конечного пользователя, качество которого может оказаться решающим для востребованности и успешного практического применения программного проекта.

2. Основные требования и принципы компонентной архитектуры БСМ

Вычислительное ядро базовой системы моделирования представляет собой инструментальное окружение, с помощью которого средствами внутреннего конфигурационного управления для конкретного класса задач (и пользователя) формируется приложение, или пакет прикладных программ. Конструируемый ППП может или отторгаться, или эксплуатироваться в составе БСМ. Эти операции требуют соответствующей системной поддержки, и здесь необходимо выделить такую важную методологическую проблему, как автоматизация тестирования задач и алгоритмов, требующая создания презентативного банка пробных примеров.

Другая важная функция системного наполнения — обеспечение внутренних межмодульных интерфейсов, включая проблему поддержки многоязыковости. Связанная с этим, но совершенно независимая проблема — внешние интерфейсы вычислительных инструментариев БСМ с окружающим миром, что подразумевает как кросс-платформенность проекта, так и взаимодействие со сторонними прикладными продуктами, среди которых, в первую очередь, надо иметь разнообразные САПРовские разработки, имеющие многолетний опыт использования в промышленности. Интерфейсные аспекты имеют и третью важную сторону — обеспечение комфортных контактов с пользователями, под которыми надо подразумевать как разработчиков модулей, так и лиц отраслевых профессий, эксплуати-

рующие ППП, и от успеха которых зависит достижение конечной цели математиками и программистами. Очевидно, что качество решения всех интерфейсных проблем в значительной степени определяет уровень интеллектуальности разработки, а главное направление деятельности в этом направлении — это множественность структур данных и создание достаточно содержательного набора проблемно-ориентированных языков (Domestic Specific Language) для обеспечения взаимной конвертации информационных форматов.

Еще одна функция системного наполнения заключается в расширении и модернизации состава моделей и алгоритмов в ядре БСМ. Здесь существенно разнятся ситуации, когда новый включаемый модуль создан в «родной» инструментальной обстановке или привносится извне. Пополнение и обновление функционального наполнения ядра БСМ — это залог длительного жизненного цикла проекта, необходимого для его внедренческого и практического успеха. В определенном смысле данная характеристика означает «инновации в квадрате», поскольку само применение БСМ к решению конкретных практических проблем подразумевает существенный экономический эффект в соответствующей отрасли. Важнейшая сторона данной проектной деятельности — это производительность труда прикладного математика-программиста, от которой и зависит скорость и качество пополнение состава вычислительных модулей. Залогом успеха в данном случае является высокий уровень автоматизации построения алгоритмов.

К рассмотренной только что проблематике тесно примыкает другая технологическая задача — адаптация к перманентной эволюции архитектуры компьютерных систем. Последняя может означать как количественное наращивание производственных параметров МВС (количество вычислительных узлов, процессоров и ядер или объемы памяти на разных ступенях иерархии), так и кардинальное изменение структурных элементов компьютера, что происходит, вообще говоря, достаточно редко. Возникающий в данной области программистской деятельности круг вопросов имеет такую достаточно сложившуюся и «вечно актуальную» формулировку, как отображение алгоритмов на архитектуру МВС (раньше говорили — ЭВМ). Первоочередное требование, предъявляемой при этом к разработке, — это отсутствие программных ограничений на число степеней свободы решаемой задачи и на количество используемых процессоров и/или ядер. А с точки зрения производительности прикладного продукта ключевым является обеспечение масштабируемого параллелизма в сильном или слабом смысле. Первое означает линейное уменьшение времени решения фиксированной задачи при увеличении числа вычислительных элементов, а второе — примерное сохранение времени счета при одинаковом увеличении размерности задачи и количества арифметических устройств.

Большой философский смысл всегда имеет проблема антагонизма между универсальностью и экономичностью большой программной системы. В отношении БСМ выбрана единственно правильная, по-видимому, стратегия множественности состава алгоритмов для решения каждой достаточно ресурсоемкой задачи или подзадачи. Именно она позволяет выбрать на каждом конкретном этапе оптимальный или почти оптимальный алгоритм из уже имеющегося состава библиотек БСМ. Конечно, такая концепция существенно увеличивает объем работ по созданию вычислительных модулей, но в стратегическом плане такая игра стоит свеч.

В данной проблеме имеется один очень важный момент. Чем больше решаемая задача и чем сложнее ее структура со множеством подзадач, и чем богаче множество алгоритмов для их решения, тем труднее и дороже становится сама проблема поиска оптимального

метода. А поскольку зачастую процесс оптимизации приходится вести почти перебором, то такая процедура может оказаться более трудоемкой, чем само решение исходной задачи каким-то подходящим из общих естественных соображений методом. Истина находится обычно где-то посередине, и палиативным, или соломоновым, решением чаще всего является «квазиоптимизация» метода.

Разработка сверхбольших программных комплексов, каковым планируется быть БСМ, естественным образом предполагает уменьшение общих трудозатрат при выработке общего стиля и технологий работ. Однако, с другой стороны, необходимость широкой кооперации различных групп разработчиков со своими производственными условиями делает обязательным создание развитой системы поддержки компонентных технологий, обеспечивающих многоверсионность и пополняемость состава вычислительных модулей, а также закрывающих проблемы разноязычности, кросс-платформенности и другую техническую «кухню» от математиков-программистов.

Аналогичные вопросы очень актуальны и при формировании интеллектуальных интерфейсов для генерируемых в рамках БСМ пакетов прикладных программ, ориентированных на конечных пользователей с разной профессиональной подготовкой. Здесь особо следует отметить учебные и демонстрационные версии ППП, в том числе для студентов и преподавателей, а также для тренинга и переподготовки специалистов.

Важно еще выделить такую компоненту БСМ, актуальную и для разработчиков алгоритмов, и для конечных пользователей, как систему автоматического тестирования вычислительных методов и технологий, включающую презентативные каталоги пробных задач для каждой из стадий моделирования.

Заключение

В заключение можно сказать, что рассмотренные вопросы относятся к проблеме реиндустриализации прикладного программного обеспечения, с целью доведения его в идеале до уровня технологичности использования компиляторов и операционных систем. В определенном смысле мы подтверждаем ту истину, что «новое — это хорошо забытое старое», поскольку данные аспекты активно обсуждались еще в 80-е годы (естественно, на своем уровне развития методологии, см., например, [18]).

Резюме данной работы можно сформулировать таким образом, что качественное проектирование интегрированного программного окружения для обеспечения математического моделирования широкого круга реальных процессов и явлений, основанного на решении междисциплинарных прямых и обратных задач и ориентированного на длительный жизненный цикл с адаптацией на расширение состава вычислительных моделей, методов и алгоритмов, а также на адаптацию к эволюции суперкомпьютерных архитектур, требует безусловного развития и применения компонентных технологий. Данный подход, в свою очередь, должен предваряться всесторонней классификацией и структуризацией прикладных разработок по различным координатным направлениям: решаемые математические задачи, вычислительные методы, программно-информационные технологии, предметно-ориентированные приложения, отрасли и типы пользователей.

Работа поддержана грантом Российского научного фонда № 14-11-00485.

Литература

1. Ильин, В.П. Технологии вычислительного программирования / В.П. Ильин, И.Н. Скопин // Программирование. 2011. — № 4. — С. 53–72.
2. Ильин, В.П. Стратегии и тактики «заоблачного» математического моделирования / В.П. Ильин // Труды Международной конференции ПАВТ'2014. — Челябинск: Издательский центр ЮУрГУ, 2014. — С. 99–107.
3. ANSYS – Simulation Driven Product Development: URL: www.ansys.com (дата обращения: 10.09.2014).
4. OpenFOAM®R – The Open Source computational Fluid Dynamics (CFD) Toolbox: URL: www.openfoam.com (дата обращения: 10.09.2014).
5. DUNE Numerics. URL: www.dune-project.org (дата обращения: 10.09.2014).
6. Object Management Group. «CORBA Components». URL: http://www.omg.org (дата обращения: 10.09.2014).
7. Maloney, J. Distributed COM Application Development Using Visual C++ /J.Maloney // Prentice Hall, N.Y. – 1999.
8. CCA–Forum. The DOE common component architecture project. URL: http://www.CCA-forum.org (дата обращения: 10.09.2014).
9. CCTTSS. DOE SciDAC Center for Component Technology for Terascale Simulation Software. URL: http://www.cca-forum.org/ccttss (дата обращения: 01.10.2014).
10. Kohn, S., Kumfert G., Painter J., Ribben C. Divorcing language dependencies from a scientific software library. www:LLNL document UCRL-JC-140349. (дата обращения: 10.09.2014).
11. Babel Team. The DOE Babel Project. URL: http://www.llnl.gov/case/components/babel (дата обращения: 10.09.2014).
12. Allan, B. The CCA core specification in a distributed memory SPMD framework. Concurrency Computat: / B. Allan, R. Armstrong, A. Wolfe, J. Ray, D. Bernholdt, J. Kohl // Practice and Experience. – 2002. – Vol. 14 .– P. 323 – 345. DOI: 10.1002/cpe.651.
13. Jones, J. Component–based iterative methods for sparse linear systems / J. Jones, M. Sosonkina, Y. Saad // Concurency Computat. Pract. Exper. — 2007. — Vol. 19. — P. 625–635. DOI: 10.1002/cpe.1057.
14. The Equation Solver Interface Standards Forum. URL: http: //z.ca.sandia.gov/esi (дата обращения: 01.10.2014).
15. Ильин, В.П. DELAUNAY: технологическая среда генерации сеток/ В.П. Ильин // СибЖИМ. — 2013. — Т. 16, № 2(54). — С. 83–97.
16. Бутюгин, Д.С. CHEBYSHEV: принципы автоматизации построения алгоритмов в интегрированной среде для сеточных аппроксимаций начально-краевых задач / Д.С. Бутюгин, В.П. Ильин // Труды Международной конференции ПАВТ 2014. — Челябинск: Издательский центр ЮУрГУ, 2014. — С. 42–50.
17. Бутюгин, Д.С. Библиотека параллельных алгебраических решателей KRYLOV / Д.С. Бутюгин, Я.Л. Гурьева, В.П. Ильин, Д.В. Перевозкин // Труды конференции ПАВТ-2013. — Челябинск: Издательский центр ЮУрГУ, 2013. — С. 76–86.
18. Ершов, А.П. Пакеты программ — технология решения прикладных задач / А.П. Ершов, В.П. Ильин // Новосибирск, Препринт ВЦ СО АН СССР. – № 121; ВЦ СО АН СССР. – 1978.

Ильин Валерий Павлович, д.ф.-м.н., г.н.с., Институт вычислительной математики и математической геофизики СО РАН, профессор, Новосибирский государственный университет (Новосибирск, Российская Федерация), ilin@sscc.ru.

Поступила в редакцию 27 февраля 2015 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2015, vol. 4, no. 3, pp. 85–94

DOI: 10.14529/cmse150307

ON THE PROGRAM COMPONENTS OF THE MATHEMATICAL MODELLING

V.P. Il'in, Institute of Computational Mathematics and Mathematical Geophysics of Siberian Branch of the Russian Academy of Sciences, Novosibirsk State University (Novosibirsk, Russian Federation) ilin@sscc.ru

The fast growth of the world supercomputer power obviously directs to the aktive applications of the mathematical modelling in the processes of economics re-industrialization. The extension of applications and simultaneous appearance of the new numerical methods, accompaining by the computer architecture evolution conducts to dramatic increasing of the total applied software, the cost of which is comparable with the expensis of the multi-processor computationsl systems (MPCS). Such quantitative changes leads to qualitive re-considering of the conception and technological structures of the software products. We consider the approaches for solving the arising problems in adapting to the basic system of modelling which is oriented to the integrative support of the all main stages of the large scale computational experiment. The variety of the technological issues includes the providnce of interaction of multi-language and cross-platform program components, internal and external interfaces with convertation of the multi-fold data presentation, reusing the external products, long life circle of the system, evolution of the functionality and adaptation to the MPCS modernisation, as well as scalable parallelezation of the algorithms and programs.

Keywords: basic system of modelling, component technologies, high performance computing, scalable parallelism, mult-languagesness, cross-platform, direct and inverse inter-disciplinary problems.

References

1. Il'in V.P., Skopin I.N. Computational Programming Technologies // Programming and Computer Software. 2011. Vol. 37, № 4. P. 210–222.
2. Il'in V.P. Strategii i tactici «zaoblachnogo» modelirovania [Strategies and tactics of «overclouding» modelling]. Parallelnye vychislitelnye tekhnologii (PaVT'2014): Trudy mezhdunarodnoj nauchnoj konferentsii (Rostov-na-Donu, 1 – 3 aprelya 2014) [Parallel Computational Technologies (PCT'2014): Proceedings of the International Scientific Conference (Rostov-on-Don, Russia, April, 1 – 3, 2014)]. Chelyabinsk: Publishing of the South Ural State University, 2014. — P. 99–107.
3. ANSYS – Simulation Driven Product Development: URL: www.ansys.com (accessed: 10.09.2014)

4. OpenFOAM®R – The Open Source computational Fluid Dynamics (CFD) Toolbox: URL: www.openfoam.com (accessed: 10.09.2014).
5. DUNE Numerics: URL: www.dune-project.org (accessed: 10.09.2014).
6. Object Management Group. «CORBA Components». URL: http://www.omg.org (accessed: 10.09.2014).
7. Maloney J. Distributed COM Application Development Using Visual C++ // Prentice Hall, N.Y. – 1999.
8. CCA–Forum. The DOE common component architecture project. URL: http://www.CCA-forum.org (accessed: 10.09.2014)
9. CCTTSS. DOE SciDAC Center for Component Technology for Terascale Simulation Software. http://www.cca-forum.org/ccttss (accessed: 01.10.2014).
10. Kohn S., Kumfert G., Painter J., Ribben C. Divorcing language dependencies from a scientific software library. www:LLNL document UCRL-JC-140349. (accessed: 10.09.2014).
11. Babel Team. The DOE Babel Project. URL: http://www.llnl.gov/case/components/babel (accessed: 10.09.2014).
12. Allan B., Armstrong R., Wolfe A., Ray J., Bernholdt D., Kohl J. The CCA core specification in a distributed memory SPMD framework. Concurrency Computat // Practice and Experience. 2002. Vol. 14. P. 323–345. DOI: 10.1002/cpe.651.
13. Jones J., Sosonkina M., Saad Y. Component–based iterative methods for sparse linear systems // Concurency Computat. Pract. Exper. 2007. Vol. 19. P. 625–635. DOI: 10.1002/cpe.1057.
14. The Equation Solver Interface Standards Forum. http://z.ca.sandia.gov/esi (accessed: 01.10.2014).
15. Il'in V.P. DELAUNAY: technologicheskaya sreda generatsii setok [DELAUNAY: technology environment of the grid generation]. SibJIM. 2013. Vol. 16, № 2(54). P. 83–97.
16. Butyugin D.S., Il'in V.P. CHEBYSHEV: principy automatizatsii postroenia algoritmov v integrirovannoi srede dlya setochnyh approximatsiy nachalno-kraevyh zadach [CHEBYSHEV: the principles of automatical construction of the algorithms in the integrated environment for grid approximation of the initial boundary value problems].Parallelnye vychislitelnye tekhnologii (PaVT'2014): Trudy mezhdunarodnoj nauchnoj konferentsii (Rostov-na-Donu, 1 – 3 aprelya 2014) [Parallel Computational Technologies (PCT'2014): Proceedings of the International Scientific Conference (Rostov-on-Don, Russia, April, 1 – 3, 2014)]. Chelyabinsk: Publishing of the South Ural State University, 2014. – P. 42–50.
17. Butyugin D.S., Gurieva Y.L., Il'in V.P. i dr. Biblioteka algebraicheskih reshatelei KRYLOV [The library of algebraic solvers KRYLOV]. Parallelnye vychislitelnye tekhnologii (PaVT'2014): Trudy mezhdunarodnoj nauchnoj konferentsii (Rostov-na-Donu, 1 – 3 aprelya 2014) [Parallel Computational Technologies (PCT'2014): Proceedings of the International Scientific Conference (Rostov-on-Don, Russia, April, 1 – 3, 2014)]. Chelyabinsk: Publishing of the South Ural State University, 2014. P. 76–86.
18. Ershov A.P., Il'in V.P. Pakety programm — tehnologia resheniya prikladnyh zadach [The software packages — the technology of the solving of applied problems]. Novosibirsk, Preprint, Computing Center, SB AS USSR. № 121. 1978.

Received February 27, 2015.

УДК 519.1, 004.832.25 DOI: 10.14529/cmse150308

ПОИСК ПАР ОРТОГОНАЛЬНЫХ ДИАГОНАЛЬНЫХ ЛАТИНСКИХ КВАДРАТОВ ПОРЯДКА 10 В ПРОЕКТЕ ДОБРОВОЛЬНЫХ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ SAT@HOME[1]

О.С. Заикин, С.Е. Кочемазов

В статье рассматривается подход к решению задач поиска систем ортогональных латинских квадратов, основанный на сведении этих задач к проблеме булевой выполнимости. Была построена соответствующая кодировка для задачи поиска пар ортогональных диагональных латинских квадратов порядка 10. С помощью построенной кодировки в проекте добровольных распределенных вычислений SAT@home были найдены 17 новых пар. На основе 17 найденных пар, а также 3 ранее известных пар, были построены псевдотройки диагональных латинских квадратов порядка 10. Построение псевдотроек было осуществлено на вычислительном кластере, для этого была сделана параллельная реализация алгоритма генерации диагональных латинских квадратов порядка 10.

Ключевые слова: латинские квадраты, добровольные распределенные вычисления, задача о булевой выполнимости, проект SAT@home.

Введение

Комбинаторные задачи, связанные с латинскими квадратами, интересуют математиков со времен Леонарда Эйлера (см., например, обзорную статью [1]). Латинский квадрат порядка n — это квадратная таблица размера $n \times n$, заполненная элементами некоторого множества $M, |M| = n$ таким образом, что в каждой строке и в каждом столбце таблицы каждый элемент из M встречается в точности один раз. Изначально Леонард Эйлер в качестве множества M использовал множество букв латинского алфавита, именно поэтому такие квадраты были названы латинскими. В дальнейшем в настоящей статье в качестве M будет использовано множество $\{0, \ldots, n-1\}$.

Пара латинских квадратов одинакового порядка называется ортогональной, если различны все упорядоченные пары чисел (a, b), где a — число в некоторой ячейке первого латинского квадрата, а b — число в ячейке с тем же номером второго латинского квадрата. Если имеется набор из m различных латинских квадратов, любая пара которых ортогональна, то говорят о системе из m попарно ортогональных латинских квадратов. Одной из самых известных нерешенных задач, касающейся латинских квадратов, является следующая: определить, существует ли система из трех попарно ортогональных латинских квадратов порядка 10 [2].

Задачи поиска комбинаторных структур можно решать различными подходами. В настоящей статье развивается SAT-подход к решению таких задач. SAT-подход состоит в сведении исходной задачи к проблеме булевой выполнимости (SAT [3]). Привлекательность SAT-подхода обусловлена тем, что к проблеме булевой выполнимости можно эффективно свести обширный класс задач криптографии, биоинформатики и поиска

[1] Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии – 2015»

комбинаторных структур [4]. Все известные алгоритмы решения SAT-задач экспоненциальны в худшем случае, т.к. SAT является NP-трудной задачей. Несмотря на это, современные SAT-решатели (в том числе параллельные и распределенные) успешно справляются с решением целого ряда задач из упомянутых выше областей. Большинство из современных быстрых SAT-решателей основано на алгоритме Conflict-Driven Clause Learning (CDCL), базовые принципы которого рассмотрены в обзорной статье [5].

Для использования SAT-подхода необходимо перейти от исходной постановки к булевому уравнению вида «КНФ=1» (КНФ — конъюнктивная нормальная форма). Такой переход называется пропозициональным кодированием исходной проблемы. Попытки применить SAT-подход к задаче поиска систем ортогональных латинских квадратов регулярно предпринимаются с середины 90-х годов XX века. Много полезной информации о применении SAT-решателей к поиску различных систем ортогональных латинских квадратов содержится в обзорной статье [6]. Кроме всего прочего, автор статьи [6] пробовал решить упомянутую выше задачу поиска тройки попарно ортогональных квадратов порядка 10 с использованием SAT-решателя PSATO в течение более 10 лет в грид-системе из 40 персональных компьютеров (окончательный ответ так и не был получен).

Кратко опишем структуру статьи. В первом разделе будет представлена техника пропозиционального кодирования задач поиска систем ортогональных латинских квадратов, а также результат применения этой техники к построению кодировки задачи поиска пар ортогональных диагональных латинских квадратов порядка 10. Во втором разделе будет описан эксперимент по поиску пар ортогональных диагональных латинских квадратов порядка 10 в проекте добровольных распределенных вычислений SAT@home. В третьем разделе приведены результаты использования вычислительного кластера для поиска псевдотроек ортогональных диагональных латинских квадратов порядка 10. В заключении мы подводим краткие итоги проведенного исследования, а также описываем наши планы на будущее.

1. Пропозициональное кодирование задач поиска систем ортогональных латинских квадратов

В силу того, что система попарно ортогональных латинских квадратов как комбинаторная структура эквивалентна ряду других комбинаторных структур, задачу поиска такой системы можно переформулировать через эквивалентные объекты. Это позволяет использовать для решения задачи различные пропозициональные кодировки. Так, например, можно кодировать задачу поиска пары ортогональных латинских квадратов при помощи ортогональных массивов или записывать ее через трансверсали. Вообще говоря, даже если использовать пропозициональную кодировку, основанную на латинских квадратах, возможны различные варианты формулирования необходимых условий. В описанных далее вычислительных экспериментах мы использовали так называемую «наивную» кодировку задачи поиска набора ортогональных латинских квадратов, аналогичную приведенной, например, в [6, 7].

Приведем краткое описание данной кодировки. Рассматриваем две матрицы $A = \|a_{ij}\|$ и $B = \|b_{ij}\|$, $i,j = \overline{1,\dots,n}$. Содержимое каждой ячейки любой из матриц кодируется n булевыми переменными. Для кодирования всей матрицы, таким образом, требуется n^3 булевых переменных. Будем использовать запись $x(i,j,k)$ и $y(i,j,k)$ для обозначения переменных, кодирующих элементы матриц A и B, соответственно. При этом

переменная $x(i,j,k)$ принимает значение «истина» тогда и только тогда, когда в ячейке, находящейся в строке с номером i и столбце с номером j, стоит число k. Чтобы матрицы A и B представляли латинские квадраты, необходимо наложить на соответствующие им переменные ряд условий. Данные условия рассмотрены далее на примере матрицы A. Они естественным образом записываются в виде конъюнкций дизъюнктов.

В каждой ячейке матрицы стоит ровно одно целое число от 0 до $n-1$:

$\wedge_{i=1}^{n}\wedge_{j=1}^{n}\vee_{k=1}^{n} x(i,j,k)$;

$\wedge_{i=1}^{n}\wedge_{j=1}^{n}\wedge_{k=1}^{n-1}\wedge_{r=k+1}^{n}\left(\neg x(i,j,k)\vee\neg x(i,j,r)\right)$.

Каждое число от 0 до $n-1$ появляется в каждой строке ровно один раз:

$\wedge_{j=1}^{n}\wedge_{k=1}^{n}\vee_{i=1}^{n} x(i,j,k)$;

$\wedge_{j=1}^{n}\wedge_{k=1}^{n}\wedge_{i=1}^{n-1}\wedge_{r=i+1}^{n}\left(\neg x(i,j,k)\vee\neg x(r,j,k)\right)$.

Каждое число от 0 до $n-1$ появляется в каждом столбце ровно один раз:

$\wedge_{i=1}^{n}\wedge_{k=1}^{n}\vee_{j=1}^{n} x(i,j,k)$;

$\wedge_{i=1}^{n}\wedge_{k=1}^{n}\wedge_{j=1}^{n-1}\wedge_{r=j+1}^{n}\left(\neg x(i,j,k)\vee\neg x(i,r,k)\right)$.

Аналогичным образом записываются условия, кодирующие матрицу B. После этого необходимо закодировать условие ортогональности. Например, это можно сделать следующим образом:

$\wedge_{i=1}^{n}\wedge_{j=1}^{n}\wedge_{k=1}^{n}\wedge_{p=1}^{n}\wedge_{q=1}^{n}\wedge_{r=1}^{n}\left(\neg x(i,j,k)\vee\neg y(i,j,k)\vee\neg x(p,q,r)\vee\neg y(p,q,r)\right)$.

В результате применения всех описанных выше условий для задачи поиска пар ортогональных латинских квадратов порядка 10 мы получили КНФ, состоящую из 432600 дизъюнктов над множеством из 2000 булевых переменных. Соответствующий файл с КНФ в формате DIMACS занимает 9,5 мегабайт.

Рассмотрим задачу поиска пар ортогональных диагональных латинских квадратов. Латинский квадрат является диагональным, если для него справедливо следующее условие: главная и побочная диагонали содержат все числа от 0 до $n-1$, где n — это порядок латинского квадрата. Таким образом, условие уникальности элементов накладывается не только на строки и столбцы квадрата, но и на главную и побочную диагонали. Известно, что пары ортогональных диагональных латинских квадратов порядка 10 существуют — в 1992 году была опубликована статья [8], в которой были представлены три такие пары. В открытых источниках нам не удалось найти другие пары, кроме этих трех. Поэтому, на наш взгляд, интересным представляется поиск новых пар ортогональных диагональных латинских квадратов порядка 10. Решению именно этой задачи, в основном, посвящена настоящая статья. Для получения пропозициональной кодировки задачи поиска систем таких квадратов необходимо добавить в описанную выше кодировку дизъюнкты, соответствующие условию диагональности. Полученная в результате КНФ состоит из 2000 переменных и 434440 дизъюнктов, файл с КНФ в формате DIMACS занимает 10 мегабайт.

2. Поиск систем ортогональных латинских квадратов в проекте добровольных распределенных вычислений SAT@home

Задачи поиска систем ортогональных латинских квадратов при помощи SAT-подхода хорошо подходят для организации масштабных экспериментов в распределенных вычислительных средах, в частности, в проектах добровольных распределенных

вычислений. Это объясняется тем, что SAT-задачи сами по себе допускают естественные стратегии крупноблочного распараллеливания. Авторами статьи в сотрудничестве с коллегами из ИППИ РАН был разработан проект SAT@home [9–11], функционирующий с сентября 2011. Данный проект предназначен для решения трудных экземпляров SAT-задач из различных предметных областей. При создании проекта была использована открытая платформа BOINC [12, 13].

По состоянию на 10 марта 2015 года к проекту SAT@home подключено 3214 активно работающих ПК участников со всего мира, обеспечивающих среднюю производительность около 10 TFLOPs. Отметим, что отечественных проектов добровольных распределенных вычислений относительно немного: OPTIMA@home [9], Gerasim@home [14] и NetMax@home [15]. При этом у проекта SAT@home самая высокая производительность.

Схема решения SAT-задач с помощью проекта SAT@home представлена на рис. 1. Предварительно осуществляется поиск «хорошей» декомпозиции SAT-задачи с помощью метода Монте-Карло на вычислительном кластере [10, 16]. Необходимость использования кластера исходит из того, что на данном этапе используются интенсивные межпроцессорные обмены.

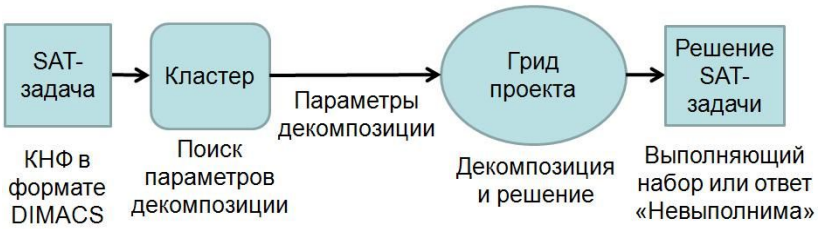


Рис. 1. Схема решения SAT-задач с помощью проекта SAT@home

В мае 2012 в проекте SAT@home был успешно завершен полугодовой эксперимент, направленный на решение 10 задач криптоанализа генератора ключевого потока A5/1 [10], которые не решаются с помощью известных rainbow-таблиц [17]. В каждой задаче нужно было найти неизвестное начальное заполнение регистров генератора.

На следующем этапе в SAT@home был запущен поиск пар ортогональных диагональных латинских квадратов порядка 10 с использованием пропозициональной кодировки, описанной в предыдущем разделе. В качестве ядра клиентского приложения (которое запускается на ПК участников) использовался CDCL-решатель MiniSat 2.2 [18], в который были внесены небольшие изменения, направленные на уменьшение объема используемой оперативной памяти. Первая строка первого квадрата была изначально зафиксирована и равнялась «0 1 2 3 4 5 6 7 8 9». Это было сделано потому, что любую пару ортогональных латинских квадратов путем перестановок, не нарушающих условия ортогональности и диагональности, можно преобразовать к паре, в которой первая строка первого квадрата будет именно такой.

Выбор способа декомпозиции SAT-задач — нетривиальная проблема, в решении которой нужно использовать свойства исходных постановок [10, 16]. Декомпозиция задачи поиска пар ортогональных диагональных латинских квадратов была осуществлена следующим образом. Варьировались значения первых 8 ячеек второй и третьей строки первого квадрата. Всего оказалось около 230 миллиардов возможных вариантов значений этих 16 ячеек, не нарушающих условие, что квадрат является диагональным латинским квадратом. Было решено сформировать для решения в SAT@home первые 20 миллионов подзадач из 230 миллиардов (т.е. всего около 0,0087 % описанного выше пространства

поиска). В итоге каждая подзадача формировалась в результате подстановки в первый квадрат 8 первых ячеек второй и третьей строки (при фиксированной первой строке). Значения остальных 74 ячеек первого квадрата и всех 100 ячеек второго квадрата были неизвестны — их должен был найти SAT-решатель. На каждую подзадачу был установлен лимит в 2600 рестартов решателя MiniSat 2.2, что примерно соответствует 5 минутам работы одного ядра процессора Intel Core 2 Duo E8400. При достижении лимита вычисления прерывались. Каждое задание, которое получал участник SAT@home, состояло из 20 таких подзадач. Формирование именно такого количества подзадач в задании было продиктовано необходимостью обеспечить в задании около 2 часов работы на одном ядре процессора (такая продолжительность выполнения заданий хорошо подходит для BOINC-проектов). На обработку 20 миллионов подзадач (в виде 1 миллиона заданий), сгенерированных для данного эксперимента, потребовалось около 9 месяцев работы проекта (с сентября 2012 по май 2013). Вычисления практически для всех подзадач были прерваны при достижении лимита, но решение 17 подзадач закончилось успешно — в результате были найдены 17 новых пар ортогональных диагональных латинских квадратов порядка 10 (мы сравнивали их с тремя ранее известными парами квадратов из статьи [8]). Все найденные пары размещены на сайте проекта SAT@home [11] в разделе «Найденные решения». На рис. 2 приведена первая пара ортогональных диагональных латинских квадратов порядка 10, найденная в проекте SAT@home.

$$\begin{bmatrix} 0&1&2&3&4&5&6&7&8&9\\ 1&2&0&4&3&7&9&8&5&6\\ 7&3&5&9&0&4&8&6&2&1\\ 3&5&6&8&9&0&4&1&7&2\\ 4&9&7&2&6&8&1&5&0&3\\ 5&8&4&6&7&1&3&2&9&0\\ 8&4&9&1&2&3&7&0&6&5\\ 6&7&3&0&1&2&5&9&4&8\\ 9&0&1&5&8&6&2&4&3&7\\ 2&6&8&7&5&9&0&3&1&4 \end{bmatrix} \begin{bmatrix} 0&1&2&3&4&5&6&7&8&9\\ 7&5&1&9&2&8&0&4&6&3\\ 1&0&3&4&6&7&5&2&9&8\\ 9&8&4&7&5&2&1&0&3&6\\ 6&7&9&0&8&3&2&1&5&4\\ 4&6&5&1&0&9&8&3&2&7\\ 2&3&8&5&1&6&4&9&7&0\\ 5&2&7&8&3&4&9&6&0&1\\ 3&4&6&2&9&0&7&8&1&5\\ 8&9&0&6&7&1&3&5&4&2 \end{bmatrix}$$

Рис. 2. Первая пара ортогональных диагональных латинских квадратов порядка 10, найденная в проекте SAT@home

Как уже было отмечено в предыдущем разделе, есть много различных вариантов формирования пропозициональной кодировки задачи поиска пар ортогональных диагональных латинских квадратов. При этом встает вопрос сравнения различных кодировок по эффективности. Как показала практика, количество переменных и дизъюнктов в КНФ не всегда адекватно указывает на скорость ее обработки с помощью SAT-решателей. Мы предполагаем в ближайшем будущем использовать найденные пары ортогональных диагональных латинских квадратов (а также ранее известные пары) для оценки эффективности возможных кодировок рассмотренной задачи. Сравнение можно проводить по набору КНФ (каждая КНФ соответствует известной паре ортогональных диагональных латинских квадратов порядка 10), в каждой из которых подставлено правильное означивание нескольких строк первого квадрата из соответствующей пары. Также с помощью такого тестового набора задач можно выбрать наиболее подходящие для этой предметной области SAT-решатели.

3. Поиск псевдотроек ортогональных диагональных латинских квадратов на вычислительном кластере

Большой научный интерес представляет знаменитая задача о существовании тройки попарно ортогональных латинских квадратов порядка 10. Однако, данная задача крайне трудна. Поэтому активно исследуется задача нахождения таких троек латинских квадратов, что условие ортогональности выполняется не для всех пар квадратов (например, только для одной или для двух пар из трех возможных). Далее подобные системы латинских квадратов называются псевдотройками. Главная характеристика псевдотроек, которая будет рассматриваться далее — это количество упорядоченных пар элементов, по которым выполняется условие ортогональности одновременно для всех трех пар квадратов (A и B, A и C, B и C), составляющих псевдотройку. На данный момент рекордный по этому показателю набор — это опубликованная в статье [19] псевдотройка с характеристикой 91, при этом для двух пар квадратов из трех условие ортогональности выполняется полностью. На рис. 3 представлена данная псевдотройка, в ней квадрат A ортогонален квадратам B и C, при этом квадраты B и C ортогональны только по 91 упорядоченной паре элементов из 100 возможных.

$$A = \begin{bmatrix} 0&8&9&7&5&6&4&2&3&1 \\ 9&1&4&6&2&7&3&8&0&5 \\ 7&4&2&5&1&3&8&6&9&0 \\ 8&6&5&3&9&2&1&0&4&7 \\ 6&2&1&8&4&0&9&5&7&3 \\ 4&9&3&2&7&5&0&1&6&8 \\ 5&3&7&1&0&8&6&9&2&4 \\ 3&5&0&9&8&4&2&7&1&6 \\ 1&7&6&0&3&9&5&4&8&2 \\ 2&0&8&4&6&1&7&3&5&9 \end{bmatrix} B = \begin{bmatrix} 0&7&8&9&1&2&3&4&5&6 \\ 9&0&6&1&8&3&2&5&4&7 \\ 7&2&0&4&3&9&1&8&6&5 \\ 8&5&3&0&2&1&7&6&9&4 \\ 6&9&5&3&0&7&4&2&1&8 \\ 4&1&7&6&5&0&8&9&3&2 \\ 5&4&2&8&9&6&0&3&7&1 \\ 3&6&1&7&4&8&5&0&2&9 \\ 1&8&4&2&6&5&9&7&0&3 \\ 2&3&9&5&7&4&6&1&8&0 \end{bmatrix} C = \begin{bmatrix} 0&7&8&9&1&2&3&4&5&6 \\ 6&4&2&8&9&5&1&3&7&0 \\ 4&9&5&3&2&7&6&0&1&8 \\ 5&1&7&6&4&3&8&9&0&2 \\ 3&2&9&0&7&1&5&6&8&4 \\ 1&0&3&7&6&8&2&5&4&9 \\ 2&8&0&1&3&4&9&7&6&5 \\ 9&5&4&2&8&6&0&1&3&7 \\ 7&3&6&5&0&9&4&8&2&1 \\ 8&6&1&4&5&0&7&2&9&3 \end{bmatrix}$$

Рис. 3. Рекордная псевдотройка ортогональных латинских квадратов порядка 10 из статьи [19]

Мы рассмотрели похожу по постановке задачу, состоящую в поиске псевдотроек диагональных латинских квадратов порядка 10. В неявном виде данная постановка присутствует в статье [8]. В этой статье приведены 3 пары ортогональных диагональных латинских квадратов порядка 10, при этом в первой и второй паре фигурирует один и тот же квадрат (второй в обеих парах). Упомянутые наборы квадратов изображены на рис. 4 и рис. 5 соответственно.

$$A = \begin{bmatrix} 0&9&4&6&1&7&5&8&2&3 \\ 7&1&9&4&5&3&8&0&6&2 \\ 4&6&2&8&3&1&7&5&9&0 \\ 6&0&7&3&2&8&4&9&1&5 \\ 5&3&6&7&4&2&9&1&0&8 \\ 8&4&1&2&9&5&0&6&3&7 \\ 2&5&3&0&8&9&6&4&7&1 \\ 3&2&8&9&0&4&1&7&5&6 \\ 9&7&5&1&6&0&3&2&8&4 \\ 1&8&0&5&7&6&2&3&4&9 \end{bmatrix} B = \begin{bmatrix} 0&8&5&1&7&3&4&6&9&2 \\ 5&1&7&2&9&8&0&3&4&6 \\ 1&7&2&9&5&6&8&0&3&4 \\ 9&6&4&3&0&2&7&1&5&8 \\ 3&0&8&6&4&1&5&9&2&7 \\ 4&3&0&8&6&5&9&2&7&1 \\ 7&2&9&5&1&4&6&8&0&3 \\ 6&4&3&0&8&9&2&7&1&5 \\ 2&9&6&4&3&7&1&5&8&0 \\ 8&5&1&7&2&0&3&4&6&9 \end{bmatrix}$$

Рис. 4. Первая пара ортогональных диагональных латинских квадратов порядка 10 из статьи [8]

$$A = \begin{bmatrix} 0&4&1&9&8&2&7&3&5&6 \\ 3&1&6&8&2&9&4&5&0&7 \\ 6&5&2&4&9&0&3&8&7&1 \\ 1&8&5&3&7&4&9&0&6&2 \\ 9&2&0&5&4&7&8&6&1&3 \\ 8&6&3&7&1&5&0&9&2&4 \\ 4&0&7&2&5&3&6&1&9&8 \\ 2&9&4&1&6&8&5&7&3&0 \\ 7&3&9&6&0&1&2&4&8&5 \\ 5&7&8&0&3&6&1&2&4&9 \end{bmatrix} B = \begin{bmatrix} 0&8&5&1&7&3&4&6&9&2 \\ 5&1&7&2&9&8&0&3&4&6 \\ 1&7&2&9&5&6&8&0&3&4 \\ 9&6&4&3&0&2&7&1&5&8 \\ 3&0&8&6&4&1&5&9&2&7 \\ 4&3&0&8&6&5&9&2&7&1 \\ 7&2&9&5&1&4&6&8&0&3 \\ 6&4&3&0&8&9&2&7&1&5 \\ 2&9&6&4&3&7&1&5&8&0 \\ 8&5&1&7&2&0&3&4&6&9 \end{bmatrix}$$

Рис. 5. Вторая пара ортогональных диагональных латинских квадратов порядка 10 из статьи [8]

На основе этих двух пар ортогональных диагональных латинских квадратов порядка 10 можно естественным образом построить псевдотройку порядка 10 (см. рис. 6).

$$A = \begin{bmatrix} 0&8&5&1&7&3&4&6&9&2 \\ 5&1&7&2&9&8&0&3&4&6 \\ 1&7&2&9&5&6&8&0&3&4 \\ 9&6&4&3&0&2&7&1&5&8 \\ 3&0&8&6&4&1&5&9&2&7 \\ 4&3&0&8&6&5&9&2&7&1 \\ 7&2&9&5&1&4&6&8&0&3 \\ 6&4&3&0&8&9&2&7&1&5 \\ 2&9&6&4&3&7&1&5&8&0 \\ 8&5&1&7&2&0&3&4&6&9 \end{bmatrix} B = \begin{bmatrix} 0&9&4&6&1&7&5&8&2&3 \\ 7&1&9&4&5&3&8&0&6&2 \\ 4&6&2&8&3&1&7&5&9&0 \\ 6&0&7&3&2&8&4&9&1&5 \\ 5&3&6&7&4&2&9&1&0&8 \\ 8&4&1&2&9&5&0&6&3&7 \\ 2&5&3&0&8&9&6&4&7&1 \\ 3&2&8&9&0&4&1&7&5&6 \\ 9&7&5&1&6&0&3&2&8&4 \\ 1&8&0&5&7&6&2&3&4&9 \end{bmatrix} C = \begin{bmatrix} 0&4&1&9&8&2&7&3&5&6 \\ 3&1&6&8&2&9&4&5&0&7 \\ 6&5&2&4&9&0&3&8&7&1 \\ 1&8&5&3&7&4&9&0&6&2 \\ 9&2&0&5&4&7&8&6&1&3 \\ 8&6&3&7&1&5&0&9&2&4 \\ 4&0&7&2&5&3&6&1&9&8 \\ 2&9&4&1&6&8&5&7&3&0 \\ 7&3&9&6&0&1&2&4&8&5 \\ 5&7&8&0&3&6&1&2&4&9 \end{bmatrix}$$

Рис. 6. Псевдотройка ортогональных диагональных латинских квадратов порядка 10, которую можно построить на основе первых двух пар квадратов из статьи [8]

Нетрудно посчитать, что характеристика такой псевдотройки равна 60. Напомним, что под характеристикой псевдотройки понимается число упорядоченных пар элементов, по которым одновременно выполняется условие ортогональности для всех трех пар квадратов псевдотройки. На рис. 7 представлены соответствующие 60 упорядоченных пар элементов для данной псевдотройки.

$$\begin{bmatrix} 00&01&02&-&-&05&06&-&08&- \\ 10&11&-&13&-&15&16&-&18&- \\ -&21&22&-&24&25&-&27&-&29 \\ -&-&32&33&-&-&36&37&-&39 \\ -&41&-&-&44&45&46&-&48&49 \\ 50&-&52&53&-&55&-&57&58&59 \\ 60&61&-&-&-&65&66&-&-&69 \\ -&-&72&73&74&75&-&77&-&79 \\ -&-&-&83&84&85&-&87&88&- \\ 90&91&-&93&94&-&96&97&98&99 \end{bmatrix}$$

Рис. 7. Набор из 60 упорядоченных пар элементов, по которым ортогональны все три пары квадратов псевдотройки, изображенной на рис. 6

Мы рассмотрели следующую постановку задачи: найти псевдотройку ортогональных диагональных латинских квадратов порядка 10 с характеристикой больше, чем у псевдотройки из статьи [8]. Для решения поставленной задачи было решено использовать пары ортогональных диагональных латинских квадратов порядка 10 (3 пары из статьи [8] и 17 пар, найденных в результате эксперимента, описанного в предыдущем разделе).

Идея состоит в генерации диагональных латинских квадратов порядка 10 с последующим формированием двадцати псевдотроек на основе известных пар и каждого сгенерированного квадрата. Для каждой псевдотройки можно быстро вычислить ее характеристику. Таким образом, можно выбрать лучшую псевдотройку (из двадцати) для конкретного сгенерированного квадрата, а также лучшую псевдотройку для всех сгенерированных квадратов.

Для генерации диагональных латинских квадратов был использован алгоритм поиска с возвратом. Основная идея этого алгоритма заключается в последовательной подстановке допустимых значений в ячейки таблицы, которая изначально пуста. Заполнение происходит слева направо сверху вниз по строкам таблицы. Поиск завершается, если найдено заполнение таблицы, которое соответствует некоторому диагональному латинскому квадрату. После подстановки каждого нового значения в некоторую ячейку осуществляется проверка, нарушает ли текущее заполнение таблицы условия, накладываемые на элементы в диагональном латинском квадрате. Если заполнение не прошло проверку, происходит возврат до ближайшей ячейки, элемент которой нарушает условия, после чего этой ячейке назначается другое допустимое значение. В алгоритме хранится история уже использованных неудачных значений для каждой ячейки таблицы. Это нужно для того, чтобы исключить повтор вариантов, которые приводят к недопустимым вариантам заполнений. Если для некоторой ячейки требуется поменять значение, но исчерпано множество допустимых вариантов, то происходит возврат на предыдущую ячейку.

Ключевой особенностью описанного алгоритма является то, что для увеличения количества сгенерированных диагональных латинских квадратов за единицу времени следует прерывать текущий поиск после достижения некоторого лимита времени. Это связано с тем, что при выборе «неудачных» значений в первых строках таблицы алгоритму может потребоваться очень большое время для возврата к этим начальным значениям. Если же начальные значения выбраны «удачно», то искомый квадрат может быть найден очень быстро. Опытным путем было установлено, что наиболее подходящий лимит по времени для программы, реализующей данный алгоритм — это 0,001 секунды. При этом фактически этот временной порог лишь немного превышает минимальное время, необходимое для «достройки» диагонального латинского квадрата при «удачно» выбранных начальных значениях.

На основе описанного алгоритма был создана MPI программа [20] на языке программирования C++. Управляющий процесс этой программы получает сообщения обо всех рекордных псевдотройках, найденных в процессе работы параллельной программы. Вычислительные процессы генерируют диагональные латинские квадраты порядка 10 с использованием приведенного выше алгоритма. При нахождении очередной рекордной псевдотройки вычислительный процесс отправляет соответствующее сообщение на управляющий процесс.

Созданная MPI программа была запущена на 1 сутки на 30 узлах вычислительного кластера «Академик В.М. Матросов» Иркутского суперкомпьютерного центра СО РАН. На этих 30 узлах было задействовано 60 16-ядерных процессоров AMD Opteron 6276, т.е. суммарно 960 процессорных ядер. В результате работы MPI программы была найдена псевдотройка с характеристикой 62, что является рекордом в сравнении с рассмотренной выше псевдотройкой из статьи [8]. Рекордная псевдотройка была найдена через 7

часов 20 минут после старта работы программы. Всего же за сутки было сгенерировано около 20 миллионов разных диагональных латинских квадратов порядка 10, соответственно было сформировано около 400 миллионов псевдотроек. Описанные результаты аргументирует применение для рассмотренной задачи высокопроизводительных вычислений, ведь на обычном ПК для проведения таких расчетов потребовалось бы несколько месяцев. На рис. 8 представлена найденная нами рекордная псевдотройка, а на рис. 9 представлены соответствующие ей 62 упорядоченные пары элементов. Отметим, что данная псевдотройка основана на одной из 17 пар ортогональных диагональных латинских квадратов порядка 10, найденных в проекте SAT@home.

$$A = \begin{bmatrix} 0&1&2&3&4&5&6&7&8&9\\ 1&2&0&4&6&7&9&5&3&8\\ 6&9&8&5&1&3&2&0&4&7\\ 9&0&3&1&2&6&5&8&7&4\\ 3&4&5&0&7&1&8&9&6&2\\ 7&3&9&2&8&4&0&1&5&6\\ 8&5&6&7&9&2&3&4&0&1\\ 5&7&4&9&3&8&1&6&2&0\\ 4&6&1&8&5&0&7&2&9&3\\ 2&8&7&6&0&9&4&3&1&5 \end{bmatrix} B = \begin{bmatrix} 0&1&2&3&4&5&6&7&8&9\\ 6&7&4&5&9&3&0&8&1&2\\ 7&6&1&0&3&4&8&2&9&5\\ 8&5&7&9&1&2&3&6&4&0\\ 9&2&6&1&8&7&5&4&0&3\\ 2&8&3&4&0&6&9&5&7&1\\ 4&9&5&6&7&0&2&1&3&8\\ 1&0&8&2&5&9&4&3&6&7\\ 3&4&0&7&2&8&1&9&5&6\\ 5&3&9&8&6&1&7&0&2&4 \end{bmatrix} C = \begin{bmatrix} 3&6&8&5&9&4&2&0&7&1\\ 5&9&6&0&8&1&4&3&2&7\\ 1&8&7&2&0&6&9&5&3&4\\ 6&3&0&4&1&7&8&2&9&5\\ 2&1&3&8&5&9&0&7&4&6\\ 8&4&2&1&6&0&7&9&5&3\\ 9&2&5&3&7&8&6&4&1&0\\ 0&7&4&9&2&5&3&1&6&8\\ 4&0&1&7&3&2&5&6&8&9\\ 7&5&9&6&4&3&1&8&0&2 \end{bmatrix}$$

Рис. 8. Псевдотройка ортогональных диагональных латинских квадратов порядка 10, найденная с помощью MPI программы

$$\begin{bmatrix} - & 01 & 02 & 03 & 04 & 05 & 06 & 07 & 08 & - \\ 10 & 11 & - & 13 & 14 & 15 & 16 & - & - & - \\ - & 21 & - & - & - & - & 26 & 27 & 28 & 29 \\ 30 & - & 32 & - & 34 & 35 & 36 & - & 38 & - \\ 40 & 41 & - & 43 & - & - & - & - & - & 49 \\ 50 & - & 52 & 53 & 54 & 55 & - & - & 58 & - \\ 60 & - & 62 & 63 & 64 & 65 & 66 & - & 68 & - \\ 70 & 71 & - & - & - & 75 & - & 77 & 78 & 79 \\ 80 & - & 82 & - & - & 85 & 86 & 87 & - & 89 \\ - & 91 & 92 & 93 & 94 & - & 96 & 97 & 98 & 99 \end{bmatrix}$$

Рис. 9. Набор из 62 упорядоченных пар элементов, по которым ортогональны все три пары квадратов, входящих в псевдотройку, представленную на рис. 8

Заключение

В статье описано успешное применение проекта добровольных распределенных вычислений SAT@home для поиска пар ортогональных диагональных латинских квадратов порядка 10. Найдено 17 таких пар, все они являются новыми (в сравнении с тремя ранее известными). На основе найденных пар с помощью параллельной реализации алгоритма генерации диагональных латинских квадратов была найдена псевдотройка с рекордным количеством упорядоченных пар элементов, по которым ортогональны все три пары квадратов из псевдотройки. В дальнейшем мы планируем разработать и реализовать новые пропозициональные кодировки для приведенных в статье задач, а также ускорить работу используемого SAT-решателя на задачах поиска систем ортогональных латинских квадратов. Мы благодарим А.А. Семенова за ценные советы, которые позволили улучшить качество данной статьи, М.А. Посыпкина, Н.Н. Храпова и И.И. Курочкина за помощь в сопровождении проекта SAT@home, а также добровольцев за предоставленные вычислительные ресурсы.

Работа была поддержана РФФИ (гранты № 14-07-00403-а, 14-07-31172-мол-а и 15-07-07891-а) и Советом по грантам Президента РФ (стипендия № СП-1184.2015.5 и грант для поддержки ведущих научных школ № НШ-5007.2014.9).

Литература

1. Тужилин, М.Э. Об истории исследований латинских квадратов / М.Э. Тужилин // Обозрение прикладной и промышленной математики. — 2012. — Т. 19, Вып. 2. — С. 226–227.
2. Colbourn, C.J. Handbook of Combinatorial Designs. Second Edition / C.J. Colbourn, J.H. Dinitz. — Chapman&Hall, — 2006. — 984 p. DOI: 10.1201/9781420010541.
3. Biere, A. Handbook of Satisfiability / A. Biere, V. Heule, H. van Maaren, T. Walsh (eds.). — IOS Press. — 2009.
4. Семенов, А.А. Применение алгоритмов решения проблемы булевой выполнимости (SAT) к комбинаторным задачам / А.А. Семенов, О.С. Заикин, И.В. Отпущенников, С.Е. Кочемазов // Труды XII Всероссийского совещания по проблемам управления. — Изд-во ИПУ РАН. — 2014. — С. 7361–7374.
5. Семенов, А.А. Технологии решения многомерных задач логического поиска / А.А. Семенов, Д.В. Беспалов // Вестник Томского государственного университета. —2005. — № 14. — С. 61–73.
6. Zhang, H. Combinatorial Designs by SAT Solvers / H. Zhang. — In: Biere A., Heule V., van Maaren H., Walsh T. (eds.) Handbook of Satisfiability. — IOS Press. — 2009. — P. 533–568.
7. Gomes, C. Completing quasigroups or Latin squares: A structured graph coloring problem / C. Gomes, D. Shmoys // Proceedings of the Computational Symposium on Graph Coloring and Generalizations. — 2002. — P. 22–39.
8. Brown, J.W. Completion of the spectrum of orthogonal diagonal Latin squares / J.W. Brown, F. Cherry, L. Most, M. Most, E.T. Parker, W.D. Wallis // Lecture notes in pure and applied mathematics. — 1992. — Vol. 139. — P. 43–49.
9. Заикин, О.С. Опыт организации добровольных вычислений на примере проектов OPTIMA@home и SAT@home / О.С. Заикин, М.А. Посыпкин, А.А. Семенов, Н.П. Храпов // Вестник Нижегородского университета им. Н.И. Лобачевского. — 2012. — № 5-2. — С. 340–347.
10. Заикин, О.С. Процедуры построения декомпозиционных множеств для распределенного решения SAT-задач в проекте добровольных вычислений SAT@home / О.С. Заикин, А.А. Семенов, М.А. Посыпкин // Управление большими системами. — М.: ИПУ РАН. — Вып. 43. — 2013. — С. 138–156.
11. SAT@home: проект добровольных распределенных вычислений для решения крупномасштабных SAT-задач. URL: http://sat.isa.ru/pdsat/ (дата обращения 20.03.2015).
12. Anderson, D.P. BOINC: A System for Public-Resource Computing and Storage / D.P. Anderson // In: Buyya, R. (ed.) GRID, IEEE Computer Society. — 2004. — P. 4–10. DOI: 10.1109/grid.2004.14.
13. Ивашко, Е.Е. Использование BOINC-грид в вычислительноемких научных исследованиях / Е.Е. Ивашко, Н.Н. Никитина // Вестник Новосибирского государственного

университета. Серия: Информационные технологии. — 2013. — Т. 11, № 1. — С. 53–57.

14. Ватутин, Э.И. Использование добровольных распределенных вычислений на платформе BOINC для анализа качества разбиений граф-схем параллельных алгоритмов / Э.И. Ватутин, В.С. Титов // Труды шестой международной конференции «Параллельные вычисления и задачи управления» PACO'2012. — Институт проблем управления им. В.А. Трапезникова РАН, Москва. — 2012. — С. 37–54.
15. Гринберг, Я.Р. Алгоритм кластеризации элементов сетей передачи данных / Я.Р. Гринберг, И.И. Курочкин, А.В. Корх // Информационные технологии и вычислительные системы. — 2012. — № 3. — С. 18–30.
16. Заикин О.С. Применение метода Монте-Карло к прогнозированию времени параллельного решения проблемы булевой выполнимости / О.С. Заикин, А.А. Семенов // Вычислительные методы и программирование: новые вычислительные технологии. — 2014. — Т. 15, № 1. — С. 22–35.
17. Rainbow-таблицы для шифра A5/1. URL: http://opensource.srlabs.de/projects/a51-decrypt (дата обращения 30.11.2014).
18. Een, N. An Extensible SAT-solver / N. Een, N. Sorensson // Lecture notes in computer science. — 2003. — Vol. 2919. — P. 502–518. DOI: 10.1007/978-3-540-24605-3_37.
19. Egan, J. Enumeration of MOLS of small order / Egan J., Wanless I.M. // Cornell University Library. arXiv:1406.3681v2 [math.CO].
20. Гришагин, В.А. Параллельное программирование на основе MPI / В.А. Гришагин, А.Н. Свистунов. — Изд-во ННГУ им. Н.И. Лобачевского. — 2005. — 93 с.

Заикин Олег Сергеевич, к.т.н., научный сотрудник, Институт динамики систем и теории управления им. В.М. Матросова СО РАН (Иркутск, Российская Федерация), zaikin.icc@gmail.com.

Кочемазов Степан Евгеньевич, программист, Институт динамики систем и теории управления им. В.М. Матросова СО РАН (Иркутск, Российская Федерация), veinamond@gmail.com.

Поступила в редакцию 27 марта 2015 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2015, vol. 4, no. 3, pp. 95–108

DOI: 10.14529/cmse150308

THE SEARCH FOR PAIRS OF ORTHOGONAL DIAGONAL LATIN SQUARES OF ORDER 10 IN THE VOLUNTEER COMPUTING PROJECT SAT@HOME

O.S. Zaikin, Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences (Irkutsk, Russian Federation) zaikin.icc@gmail.com,
S.E. Kochemazov, Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences (Irkutsk, Russian Federation)
veinamond@gmail.com

In this paper we consider the approach to search for systems of orthogonal Latin Squares, that is based on reducing the corresponding problems to Boolean Satisfiability problem. We constructed the SAT encoding for finding orthogonal diagonal Latin Squares of order 10. Using this encoding and the resources provided by the volunteer computing project SAT@home we managed to find 17 previously unknown pairs. Based on these 17 pairs and on 3 previously published pairs we constructed the pseudotriples of diagonal Latin Squares of order 10. To construct pseudotriples we employed the computing cluster. The last step required us to make parallel implementation of the algorithm for generating diagonal Latin Squares of order 10.

Keywords: Latin squares, volunteer computing, Boolean satisfiability problem, SAT@home project.

References

1. Tuzhilin M.E. Ob istorii issledovany latinskikh kvadratov [On the history of the study of Latin squares] // Obozreniye prikladnoy i promyshlennoy matematiki [Review of Applied and Industrial Mathematics]. 2012. Vol. 19, Issue 2. P. 226–227.
2. Colbourn C.J., Dinitz J.H. Handbook of Combinatorial Designs. Second Edition. — Chapman&Hall, — 2006. — 984 p. DOI: 10.1201/9781420010541.
3. Biere A., Heule V., van Maaren H., Walsh T. (eds.). Handbook of Satisfiability. IOS Press. 2009.
4. Semenov A.A., Zaikin O.S., Otpushchennikov I.V., Kochemazov S.E. Primeneniye algoritmov resheniya problemy bulevoy vypolnimosti (SAT) k kombinatornym zadacham [Using algorithms for solving Boolean satisfiability problem (SAT) to combinatorial problems] // Trudy XII Vserossyskogo soveshchaniya po problemam upravleniya [Proceedings of the XII National Conference on Conrtol Problems]. Moskow, Publishing of the Institute of Control Sciences RAS, 2014. P. 7361–7374.
5. Semenov A.A., Bespalov D.V. Tekhnologii resheniya mnogomernykh zadach logicheskogo poiska [Technology for solving multidimensional problems of logical search] // Vestnik Tomskogo gosudarstvennogo universiteta [Tomsk State University Journal]. 2005. No. 14. P. 61–73.
6. Zhang H. Combinatorial Designs by SAT Solvers. In: Biere A., Heule V., van Maaren H., Walsh T. (eds.) Handbook of Satisfiability. IOS Press. 2009. P. 533–568.

7. Gomes C., Shmoys D. Completing quasigroups or Latin squares: A structured graph coloring problem // Proceedings of the Computational Symposium on Graph Coloring and Generalizations. 2002. P. 22–39.
8. Brown J.W., Cherry F., Most L., Most M., Parker E.T., Wallis W.D. Completion of the spectrum of orthogonal diagonal Latin squares // Lecture notes in pure and applied mathematics. 1992. Vol. 139. P. 43–49.
9. Zaikin O.S., Posyipkin M.A., Semenov A.A., Hrapov N.P. Opyit organizatsii dobrovolnyih vyichisleniy na primere proektov OPTIMA@home i SAT@home [Experience of organizing volunteer computing project on the examples of OPTIMA@home and SAT@home] // Vestnik Nizhegorodskogo universiteta im. N.I. Lobachevskogo [Lobachevsky State Unversity of Nizhni Novgorod Journal]. 2012. No. 5-2. P. 340–347.
10. Zaikin O.S., Semenov A.A., Posyipkin M.A. Protseduryi postroeniya dekompozitsionnyih mnozhestv dlya raspredelennogo resheniya SAT-zadach v proekte dobrovolnyih vyichisleniy SAT@home [Procedures for constructing decomposition sets for distributed SAT solving in the volunteer computing project SAT@home] // Large-scale systems control. Moskow, Publishing of the Institute of Control Sciences RAS, 2013. Issue 43. P. 138–156.
11. SAT@home: volunteer computing project for solving hard SAT problems. URL: http://sat.isa.ru/pdsat/ (accessed: 20.03.2015).
12. Anderson D.P. BOINC: A System for Public-Resource Computing and Storage. In: Buyya, R. (ed.) GRID, IEEE Computer Society. 2004. P. 4–10. DOI: 10.1109/grid.2004.14.
13. Ivashko E.E., Nikitina N.N. Ispolzovanie BOINC-grid v vyichislitelnoemkih nauchnyih issledovaniyah [Using BOINC-grid in resource-intensive scientific researches] // Vestnik Novosibirskogo gosudarstvennogo universiteta. Seriya: Informatsionnyie tehnologii [Novosibirsk State University Journal. Series information technologies]. 2013. Vol. 11, No. 1. P. 53–57.
14. Vatutin E.I., Titov V.S. Ispolzovanie dobrovolnyih raspredelennyih vyichisleniy na platforme BOINC dlya analiza kachestva razbieniy graf-shem parallelnyih algoritmov [Use of volunteer computing platform BOINC to analyze the quality of partitions of a graph-schemes of parallel algorithms] // Trudyi shestoy mezhdunarodnoy konferentsii "Parallelnyie vyichisleniya i zadachi upravleniya" PACO'2012 [Proceedings of the Sixth International Conference "Parallel Computations and Control Problems" PACO'2012]. Moscow, Publishing of the Institute of Control Sciences RAS, 2012. P. 37–54.
15. Grinberg Y.R., Kurochkin I.I., Korh A.V. Algoritm klasterizatsii elementov setey peredachi dannyih [Algorithm for clustering elements of data networks] // Informatsionnyie tehnologii i vyichislitelnyie sistemyi [Information technology and computer systems]. 2012. No. 3. P. 18–30.
16. Zaikin O.S., Semenov A.A. Primenenie metoda Monte-Karlo k prognozirovaniyu vremeni parallelnogo resheniya problemyi bulevoy vyipolnimosti [Application of the Monte Carlo method to predict time of parallel solving of the Boolean satisfiability problem] // Vyichislitelnyie metodyi i programmirovanie: novyie vyichislitelnyie tehnologi [Numerical methods and programming : new computational technologies]. 2014. Vol. 15, No. 1. P. 22–35.
17. Rainbow tables for the A5/1 cipher. URL: http://opensource.srlabs.de/projects/a51-decrypt (accessed: 30.11.2014).

18. Een N., Sorensson N. An Extensible SAT-solver // Lecture notes in computer science. 2003. Vol. 2919. P. 502–518. DOI: 10.1007/978-3-540-24605-3_37.
19. J. Egan, I.M. Wanless. Enumeration of MOLS of small order // Cornell University Library. arXiv:1406.3681v2 [math.CO].
20. Grishagin V.A., Svistunov A.N. Parallelnoye programmirovaniye na osnove MPI [Parallel programming on the base of MPI]. Nizhny Novgorod, Publishing of the Lobachevsky State University of Nizhny Novgorod, 2005. 93 p.

Received March 27, 2015.

СВЕДЕНИЯ ОБ ИЗДАНИИ

Научный журнал «Вестник ЮУрГУ», серия «Вычислительная математика и информатика» основан в 2012 году.

Свидетельство о регистрации ПИ ФС77-57377 выдано 24 марта 2014 г. Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций.

Журнал включен в Реферативный журнал и Базы данных ВИНИТИ; индексируется в библиографической базе данных РИНЦ. Сведения о журнале ежегодно публикуются в международной справочной системе по периодическим и продолжающимся изданиям «Ulrich's Periodicals Directory».

Подписной индекс научного журнала «Вестник ЮУрГУ», серия «Вычислительная математика и информатика»: 10244, каталог «Пресса России». Периодичность выхода — 4 выпуска в год (февраль, май, август и ноябрь).

ПРАВИЛА ДЛЯ АВТОРОВ

1. Правила подготовки рукописей и пример оформления статей можно загрузить с сайта серии `http://vestnikvmi.susu.ru`. **Статьи, оформленные без соблюдения правил, к рассмотрению не принимаются.**
2. Адрес редакции научного журнала «Вестник ЮУрГУ», серия «Вычислительная математика и информатика»:

   Россия 454080, г. Челябинск, пр. им. В.И. Ленина, 76, ЮУрГУ, факультет ВМИ, кафедра ИТ, ответственному секретарю Цымблеру М.Л.
3. Адрес электронной почты редакции: `vestnikvmi@susu.ru`
4. **Плата с авторов за публикацию рукописей не взимается, и гонорары авторам не выплачиваются.**