ВЕСТНИК

ЮЖНО-УРАЛЬСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2019
Т. 8, № 1

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 автономное образовательное учреждение высшего 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
- Искусственный интеллект и машинное обучение
- 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

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

2019
vol. 8, no. 1

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
- Artificial intelligence and machine learning
- System software
- Advanced multiprocessor architectures
- Cloud computing
- Software engineering
- Computer graphics
- Internet technologies
- E-learning
- Database processing
- Data mining

Содержание

Contents

УДК 519.687.1 DOI: 10.14529/cmse190101

ОПТИМИЗАЦИЯ УТИЛИЗАЦИИ ПРИ ВЫДЕЛЕНИИ РЕСУРСОВ ДЛЯ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ С СЕТЬЮ АНГАРА*

© 2019 А.В. Мукосей, А.С. Семенов, А.С. Симонов

АО «Научно-исследовательский центр электронной вычислительной техники»
(117587 Москва, Варшавское шоссе, д. 125, стр. 15)
E-mail: mukosey@nicevt.ru, semenov@nicevt.ru, simonov@nicevt.ru

Поступила в редакцию: 22.07.2018

В данной работе рассматривается высокоскоростная вычислительная сеть Ангара с топологией «многомерный тор». Работа посвящена оптимизации фрагментации, возникающей в результате последовательного выделения вычислительных узлов в многоузловой системе при заданном требовании о том, что сетевой трафик разных пользовательских заданий не должен пересекаться. Данная работа является продолжение работы по оптимизации фрагментации ресурсов исследуемой вычислительной системы. В данной работе к учету фрагментации при выборе узлов добавлен метод запуска пользовательских заданий, основанный на политике выбора первого подходящего задания (First-Fit) в некотором рассматриваемом окне заданий. Исследование разработанного метода проводилось с помощью симулятора работы вычислительной системы. Рассмотрен набор различных вычислительных систем с трехмерными и четырехмерными топологиями, размер минимальной системы — 32 вычислительных узла, максимальной — 144 узла. Для каждой системы задана синтетическая очередь заданий, параметры которой приближены к реально возможной и основаны на данных, полученных с вычислительного кластера Desmos на базе сети Ангара. В качестве критерия качества метода выбора узлов рассматривается средняя утилизация ресурсов вычислительной системы и среднее время ожидания заданий в очереди. Исследованы различные размеры окон заданий. Исследование показало, что увеличение утилизации ресурсов для предложенного метода выбора узлов составило в среднем 7 % и на 36,6 % сокращает значение времени ожидания задания в очереди по сравнению с базовым методом.

Ключевые слова: коммуникационная сеть Ангара, многомерный тор, планирование ресурсов, фрагментация, выбор узлов.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Мукосей А.В., Семенов А.С., Симонов А.С. Оптимизация утилизации при выделении ресурсов для высокопроизводительных вычислительных систем с сетью Ангара // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2019. Т. 8, № 1. С. 5–19. DOI: 10.14529/cmse190101.

Введение

В АО «НИЦЭВТ» разработана высокоскоростная коммуникационная сеть Ангара [1, 2] с топологией «многомерный тор». В маршрутизаторе сети реализована бездедлоковая, адаптивная маршрутизация, основанная на правилах «пузырька» (Bubble flow control, [3]) и «порядка направлений» (Direction ordered routing, DOR, [4, 5]) с использованием битов направлений [5]. Благодаря алгоритму First Step/Last Step «нестандартного первого и последнего шага» [5] аппаратно поддерживается обход отказавших узлов или линков. Эффективность этого метода по поддержанию связности в сети с отказами была показана в статье [6].

*Работа рекомендована Программным комитетом международной конференции «Суперкомпьютерные дни в России»

В настоящий момент для сети Ангара разрабатываются алгоритмы по выделению ресурсов при условии отсутствия пересечения сетевого трафика различных заданий, при этом особое значение имеет проблема фрагментации, возникающая в результате последовательного выделения вычислительных узлов.

Для сетей с тороидальной топологией существует несколько стратегий выделения ресурсов [7]. Возможно разделение вычислительной системы на партиции, по которым размещаются задачи пользователей. Такая стратегия может снижать эффективность использования кластера из-за выделения большего числа узлов, чем требовалось, или невозможности выделить доступный набор узлов из разных партиций. Данная стратегия использовалась в суперкомпьютерах IBM Blue Gene/P, Blue Gene/Q [8, 9], в которых ее недостатки компенсировались большим числом не очень мощных по производительности вычислительных узлов и адекватным выбором размера партиции.

Вторая стратегия используется в серии суперкомпьютеров Cray XT/XE, где расположение выделенных узлов [10] не зависит от топологии. Такой способ выделения ресурсов может привести к деградации производительности ввиду наличия конкурирующего трафика.

Помимо возникающей фрагментации, на эффективное использование ресурсов влияют методы определения порядка запуска пользовательских задач. Такая задача является NP-сложной. Существует множество алгоритмов планирования, направленных на оптимизацию использования вычислительных ресурсов по разным параметрам. FCFS [11] (First Come First Served) — политика обработки очереди заданий в порядке, в котором задания поступили. Такая политика обеспечивает справедливость в отношении порядка поступления заданий, но может снижать утилизацию из-за простаивания ресурсов.

В работе [12] проводится сравнение различных вариантов алгоритмов, таких как: First Come First Served (FCFS, первым пришел — первым обслужен), Priority Queue (очередь с приоритетами), Shortest Job First (кратчайшая задача первая), Longest Job First (продолжительная задача первая) и других.

Одним из самых эффективных на данный момент алгоритмов многие авторы называют алгоритм Backfill [13] — алгоритм обратного заполнения, который является расширением политики FCFS. Для этого алгоритма необходимо наличие оценки о времени выполнения каждого задания. В алгоритме состояние системы по мере завершения работы запущенных заданий сопоставляется с очередью заданий. В статье рассматриваются консервативный вариант алгоритма, когда не допускается выполнения задания, если это повлияет на время запуска приоритетного задания, и агрессивный вариант, позволяющий запустить задание, если это не изменит времени запуска запланированного приоритетного задания.

При большом числе пользователей возникает задача справедливого распределения ресурсов, то есть избегания ситуации, когда один пользователь захватит все вычислительные ресурсы на длительное время. Различные варианты алгоритмов справедливого распределения ресурсов реализованы в программных системах управления ресурсами: PBS [14], Torque [15], MAUI [16], SGE [17], MBC-1000 [18], SLURM [19].

Данная статья является продолжением работы [20], в данной работе к методу выделения узлов с оценкой фрагментации добавлена возможность перестановки пользовательских заданий в очереди. Примененный алгоритм перестановки заданий в очереди основан на политике First-Fit (выбора первого подходящего задания); в алгоритме возможность перестановки ограничивается некоторым рассматриваемым окном заданий.

Стоит отметить, что для решения задачи планирования иногда используются генетические алгоритмы, например, в [21], однако предварительные результаты авторов данной статьи показывают, что использование данного механизма ведет к слишком большому времени работы процедуры выбора узлов.

Также в данной работе по сравнению с предыдущей работой авторов модифицирован принцип формирования очереди пользовательских заданий, который стал более приближенным к реальности и основан на данных полученных с вычислительного кластера Desmos [22].

Статья организована следующим образом. В разделе 1 приводятся необходимые формальные определения и постановка задачи. В разделе 2 описаны разработанные алгоритмы. В разделе 3 проведено исследование построенных алгоритмов.

Разработанный алгоритм выбора вычислительных узлов для потока пользовательских заданий, сокращающий фрагментацию вычислительной системы совместно с политикой поиска ресурсов, основанной на выборе первого подходящего задания в среднем дает прирост утилизации ресурсов 7 % и на 36,6 % сокращает значение времени ожидания задания в очереди по сравнению с базовым. Исследования проведены на симуляторе вычислительной системы на топологиях с числом узлов до 144. Разработан метод генерации синтетической очереди пользовательских заданий, параметры которой приближены к реально возможной.

1. Определения и постановка задачи

В данном разделе приводятся формальные определения, которые в дальнейшем будут использоваться в статье.

Рассмотрим вычислительную систему, узлы которой объединены в тороидальную топологию. Размерности тора обозначим $(d_1, d_2, ..., d_n)$, а множество всех узлов вычислительной системы обозначим $N = \{u | u = (u_1, ..., u_n), \forall i \ u_i \in \mathbb{Z}_{d_i}\}$, а общее число узлов — $|N|$. Расстояние на множестве N определим следующим образом: $L(u, v) = \sum\limits_{i=1}^{n} |u_i - v_i|, \ \forall u, v \in N$.

Состояние системы S можно описать множествами узлов, доступных и недоступных для выделения, обозначим эти множества N_{free} и N_{locked}, соответственно.

Будем называть *маршрутизируемым* множеством узлов в коммуникационной сети Ангара такое множество, что для каждого узла этого множества существует сетевой маршрут в любой другой узел множества, удовлетворяющий правилам маршрутизации сети Ангара, а также весь сетевой трафик узлов множества не выходит за его пределы.

Будем называть *заданием* W — число узлов W_{nodes}, запрашиваемое пользователем в момент времени W_{start} на время W_{time}, а *ресурсами для задания* — маршрутизируемое множество узлов, размер которого не меньше, чем W_{nodes}. *Потоком заданий* назовем множество различных заданий W.

Ранее в работе [23] авторами статьи решалась проблема поиска маршрутизируемого множества заданного размера в коммуникационной сети Ангара с учетом топологии и маршрутизации. Обозначим алгоритм, решающий эту проблему как $Find\_Systems(W, S)$. На вход этому алгоритму подается состояние системы S и задание W с требуемым числом вычислительных узлов W_{nodes}. Результатом работы алгоритма является набор вариантов ресурсов для задания. Необходимо заметить, что особенностью алгоритма является то, что все ресурсы для задания представляют собой многомерные прямоугольники.

Под *утилизацией* ресурсов U вычислительного кластера будем понимать среднее значение утилизации по всем вычислительным узлам:

$$U = \frac{\sum\limits_{i=1}^{|N|} U_i}{|N|}, U_i = \frac{T_i}{T},$$

где U_i — утилизация i-го вычислительного узла, T — время работы вычислительного кластера, T_i — полезное время работы i-го вычислительного узла.

Обозначим *значение времени нахождения задания в очереди относительно запрошенного времени* как $T^i_{delay} = \frac{Q^i}{W^i_{time}}$, где W^i_{time} — запрошенное время для задания W^i, Q^i — время ожидания задания W^i в очереди. За *среднее значение времени нахождения задания в очереди относительно запрошенного времени задания* примем $T_{mean} = \frac{\sum_{i=1}^{k} T^i_{delay}}{k} = \frac{1}{k}\sum_{i=1}^{k} \frac{Q^i}{W^i_{time}}$, где k — число различных заданий в потоке.

За оценку качества решения для потока пользовательских заданий возьмем утилизацию ресурсов вычислительного кластера и среднее значение времени нахождения задания в очереди относительно запрошенного времени. Эти характеристики используются по аналогии с работой [24].

Во введенных обозначениях проблема, которую решает данная статья, будет формулироваться следующим образом. Для заданного вычислительного кластера и последовательности заданий $Q = W^1, ..., W^k$ требуется найти ресурсы для всех заданий из последовательности, которые будут максимизировать утилизацию вычислительного кластера и минимизировать среднее значение времени нахождения задания в очереди относительно запрошенного времени.

2. Алгоритм выбора узлов

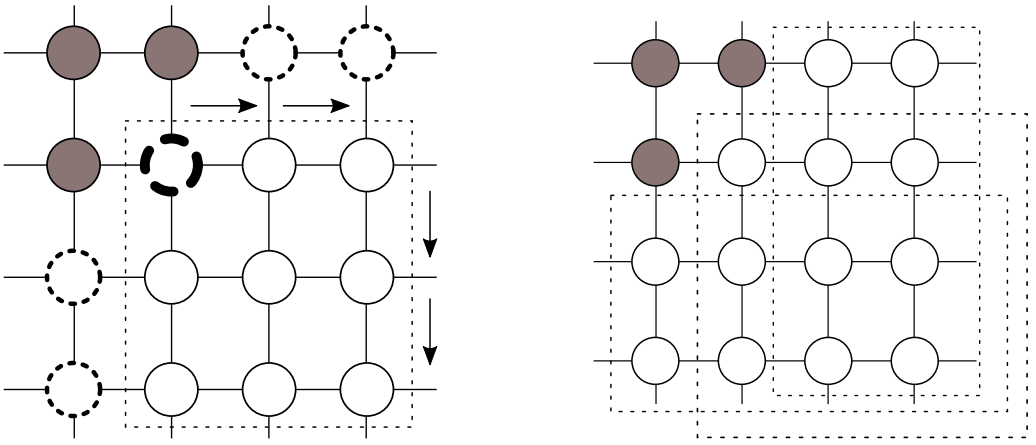
Задача упаковки контейнера является NP-полной задачей. В данной статье представлен алгоритм выбора узлов, основанный на методах, предложенных в работе [25], посвященной трехмерной упаковке контейнера. Идея алгоритма выбора узлов заключается в расположении задания таким образом, чтобы максимизировать оставшееся пространство в многомерном торе. Этот алгоритм предложен в работе авторов [20], однако для удобства восприятия приведен в данном тексте.

2.1. Алгоритм построения прямоугольников максимального размера

Назовем *прямоугольником максимально возможного размера* MSS ($MaxSpaceSize$) многомерный прямоугольник, состоящий только из узлов N_{free}, который нельзя расширить ни в одну из его сторон. Расширить прямоугольник может быть невозможно по двум причинам — либо по соответствующему измерению тора достигнуто максимальное количество узлов в кольце (расширение невозможно), либо сторона прямоугольника граничит с узлом из множества N_{locked}. Множество различных прямоугольников MSS характеризуют меру фрагментированности системы.

Алгоритм поиска различных прямоугольников MSS ($Find\_MSSs(S)$) реализован следующим образом. Из множества N_{free} выбирается узел $u_1 \in N_{free}$. Выбранный узел последовательно расширяется во все стороны, пока это возможно. Полученное множество узлов обозначим MSS_1. На следующем этапе выбирается узел $u_2 \in N_{free} \backslash MSS_1$ и аналогичным образом строится множество MSS_2. Алгоритм продолжается до тех пор,

пока множество $N_{free} \setminus \bigcup_{iter=1}^{Iters} MSS_{iter}$ не пусто, где $Iters$ — число итераций алгоритма. Псевдокод алгоритма представлен на рис. 2а. Обозначим множество различных MSS_{iter}, как $MSSs$. Важно отметить, что каждый прямоугольник строится независимо от остальных прямоугольников, в предположении доступности всех изначально свободных узлов N_{free}.



а) Построение прямоугольника максимально возможного размера

б) Все прямоугольники максимально возможного размера, построенные алгоритмом

Рис. 1. Выделение прямоугольников максимального размера

```
Input:
S -- массив, характеризующий состояние системы
S[u], может принимать следующие значения: 0 - free, 1 - locked, 2 - discovered
Output:
MSSs -- массив прямоугольников максимального размера
Find_MSS(S)
{
        Iter = 1
        dirs -- массив доступных направлений в торе, например, +x,-x,+y,-y...
        MSSs -- результирующий массив, изначально пуст
        for u in S {
                if S[u] == free {
                        MSSs[Iter].push_back(u)
                        for dir in dirs {
                                MSSs[Iter].extend(dir)
                        }
                        for v in MSSs[Iter] {
                                S[v] = discovered
                        }
                        Iter++
                }
        }
        return MSSs
}
```

Рис. 2. Псевдокод алгоритма Find_MSS поиска прямоугольников максимального размера

Иллюстрация работы алгоритма приведена на рис. 2а и рис. 2б, на которых в двумерной решетке узлы множества N_{locked} закрашены, а свободные узлы N_{free} — нет. Жирным контуром на рис. 2а выделен узел, из которого поочередным расширением построен двумерный прямоугольник, который обозначен пунктирной линией. Узлы, выделенные полужирным пунктиром, соответствуют множеству узлов N_{free}, не входящих в построенный прямоугольник. Из этих узлов будут строиться последующие прямоугольники. Все построенные прямоугольники MSS показаны на рис. 2б.

2.2. Оценка состояния вычислительной системы на основе прямоугольников максимального размера

Для оценки состояния вычислительной системы предложена функция φ, которая тем больше, чем большее число прямоугольников максимального размера имеется в системе:

$$\varphi(S) = N * MSS_{max}^{nnodes} + |MSS_{max}|,$$

где MSS_{max} — множество прямоугольников максимального размера, имеющих наибольшее число узлов MSS_{max}^{nnodes}, $|MSS_{max}|$ — число таких прямоугольников, S — текущее состояние системы.

Эта метрика была добавлена в алгоритм $Find\_Systems(W, S)$ поиска маршрутизируемого множества заданного размера. Для каждого найденного маршрутизируемого множества оценивается значение функции $\varphi(S')$, где S' – состояние вычислительной системы S после выделения узлов. Для увеличения утилизации вычислительного кластера требуется выбирать решения с наибольшим значением функции φ. Модифицированный алгоритм $Find\_Systems(W, S)$, в котором возможные варианты систем отсортированы с учетом значения функции φ, в дальнейшем будем обозначать $Find\_Systems_{MSS}(W, S)$.

2.3. Первоначальный алгоритм выбора узлов для кластеров с сетью Ангара

Алгоритм, который изначально работал на кластерах с сетью Ангара, устроен следующим образом. Для всего кластера строится таблица маршрутизации [23]. Для требуемого числа узлов W_{nodes} и допустимого числа транзитных узлов $N_{transit}$ строятся всевозможные разложения чисел $W_{nodes}, W_{nodes} + 1, ..., W_{nodes} + N_{transit}$ на n множителей, таких что $1 \leqslant p_i \leqslant d_i, \forall i \in 1..n$, где p_i — множитель разложения. Все такие разложения обозначим D. Эти разложения описывают всевозможные размеры прямоугольников, подходящих под решение задачи W. Средним диаметром прямоугольника, соответствующего разложению $D_j \in D$, назовем среднее арифметическое всех расстояний между узлами прямоугольника: $\frac{\sum_{u,v \in D_j, u!=v} L(u,v)}{|D_j|}$.

Следующий этап выбора узлов — поиск множества узлов вычислительного кластера, которое можно покрыть одним из найденных прямоугольников таким образом, чтобы в покрытии присутствовали только узлы из множества N_{free}, то есть доступные для выделения. Поиск покрытия начинается с разложений с наименьшим средним диаметром. При первом найденном решении алгоритм заканчивает свою работу.

3. Экспериментальное исследование

3.1. Симулятор вычислительного кластера

Для оценки утилизации ресурсов вычислительного кластера разработан симулятор очереди задач (заданий) и модель состояния кластера. На вход симулятору подается поток пользовательских задач $Q = W^1, ..., W^k$. На выходе выдается полное время работы всего кластера T, время работы каждого узла T_i и время предоставления ресурсов для каждого задания. Используя эти данные, можно вычислить утилизацию ресурсов вычислительного кластера U и среднее значение времени нахождения задания в очереди T_{mean}.

Введем некоторые формальные определения, необходимые для описания работы симулятора. *Очередью* симулятора Q_{now} назовем набор заданий из потока, для которых не выделялись ресурсы и время их запуска T_{start} меньше текущего симулируемого времени t. *Окном заданий* Q_{window} *размера* w назовем некоторое множество заданий таких, что $\forall W^i \in Q_{window}, i - i_{min} < w$, где i_{min} — минимальный индекс задания из множества Q_{now}. *Временем занятости узла* u системы S, назовем время, на которое узел u был выделен для некоторого задания W. В начальный момент времени $t = 0$ время занятости всех узлов равно 0. Операцией *выделения набора узлов* на время T_{alloc} назовем увеличение времени занятости для этих узлов на время T_{alloc}. *Временем изменения системы* T_S назовем время, через которое освободится хотя бы один из выделенных узлов. *Временем изменения очереди* T_{queue} назовем время, через которое хотя бы одно задание перейдет из потока заданий в очередь симулятора. Тогда *временем ожидания симулятора* T_{sleep} назовем минимальное время до изменения состояния симулятора: $T_{sleep} = min(T_S, T_{queue})$.

Алгоритм работы симулятора устроен следующим образом. Если окно заданий не пусто, симулятор выполняет процедуру поиска маршрутизируемого множества для каждого задания из окна по очереди. Если удалось найти решение, то симулятор выделяет найденные ресурсы на необходимое время, а также удаляет это задание из очереди. Если решение не было найдено, то симулятор выполняет процедуру поиска для следующего задания из окна. Если ни одно решение ни для одного задания из окна не было найдено, время симулятора сдвигается на время ожидания T_{sleep}, а время занятости каждого занятого узла u системы S уменьшается на T_{sleep}. Если очередь заданий пуста, а все узлы перешли в состояние свободных, то симулятор завершает свою работу.

3.2. Результаты исследования

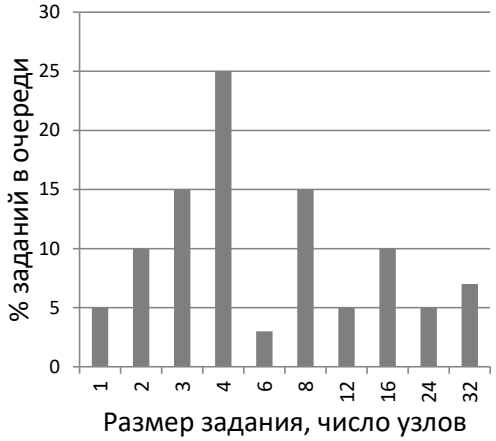
Исследование разработанного алгоритма проводилось на симуляторе для вычислительных систем, представленных в таблице.

Таблица

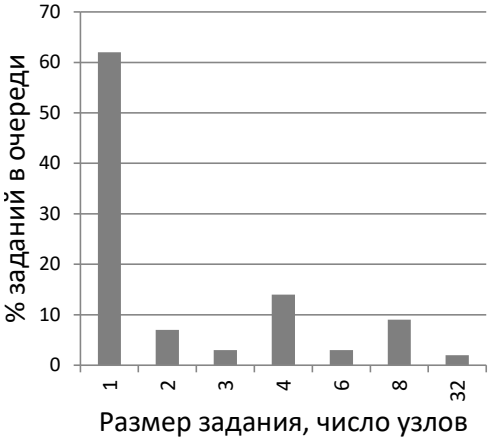
Моделируемые системы

Количество узлов	**3х-мерный тор**	**4х-мерный тор**
32	4x4x2	4x2x2x2
36	4x3x3	3x3x2x2
64	4x4x4	4x4x2x2
96	6x4x4	4x4x3x2
144	8x6x3	4x4x3x3

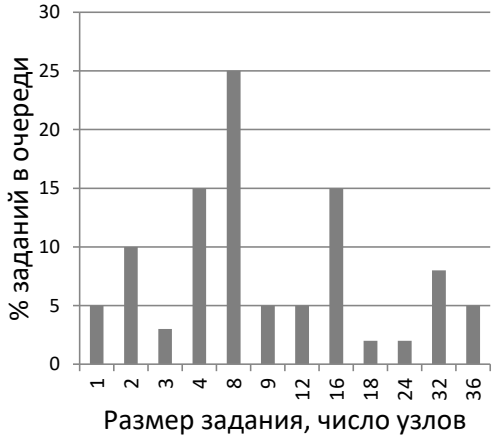
Поток заданий для каждой из систем характеризуется вероятностью появления задания для каждого числа узлов от 1 до максимального. Распределение таких вероятностей представлено на рис. 3. На рис. 2б представлено реальное распределение пользовательских задач (заданий) по количеству узлов, полученное с гибридного суперкомпьютера Desmos на базе сети Ангара, имеющую топологию 4х-мерный тор 4×2×2×2 [22]. Остальные распределения долей заданий по количеству узлов — синтетические, основанные на предположении о том, что чаще всего встречаются заданий с требуемым числом узлов, равным степеням двойки. Вероятности для остальных чисел узлов равны 0.



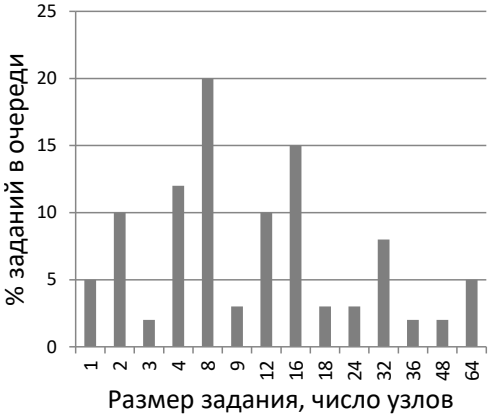
а) Для систем из 32 узлов



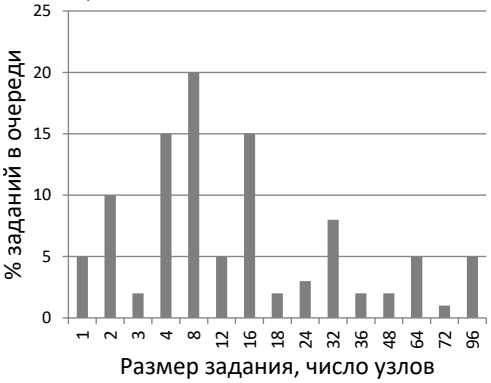
б) Для систем из 32 узлов, кластер Desmos



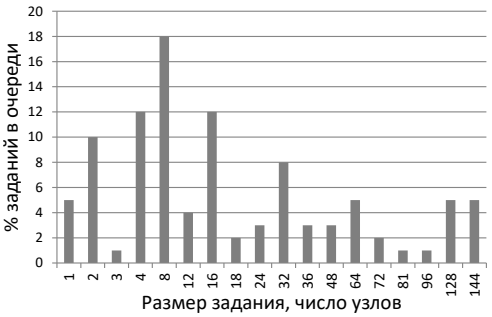
в) Для систем из 36 узлов



г) Для систем из 64 узлов



д) Для систем из 96 узлов



е) Для систем из 144 узлов

Рис. 3. Распределение заданий для систем с разным числом узлов

Задания равномерно случайно размещаются на временной шкале в диапазоне $\left[0; 60^2 * 24 * 30\right]$ для распределений полученных на кластере Desmos и на диапазоне $\left[0; 4 * 60^2 * 24 * 30\right]$ для остальных распределений вне зависимости от числа требуемых узлов.

Время продолжительности заданий соответствует распределению представленному на рис. 4, полученному в результате анализа запускаемых заданий на кластере Desmos. По оси y представлено отношение требуемого времени для задания к максимальному времени задания. На кластере Desmos максимальное время задания ограничено одними сутками. По оси x представлено процентное отношение заданий. Распределение разбито на две составляющие: на интервале $[0; 90]$ представляет из себя 10 в степени линейной функции, такой что в точке 0 оно принимает значение $0,01$, а в точке 90 — значение 99; на интервале $[90; 100]$ представляет линейную функцию и принимает значения от 99 до 100.

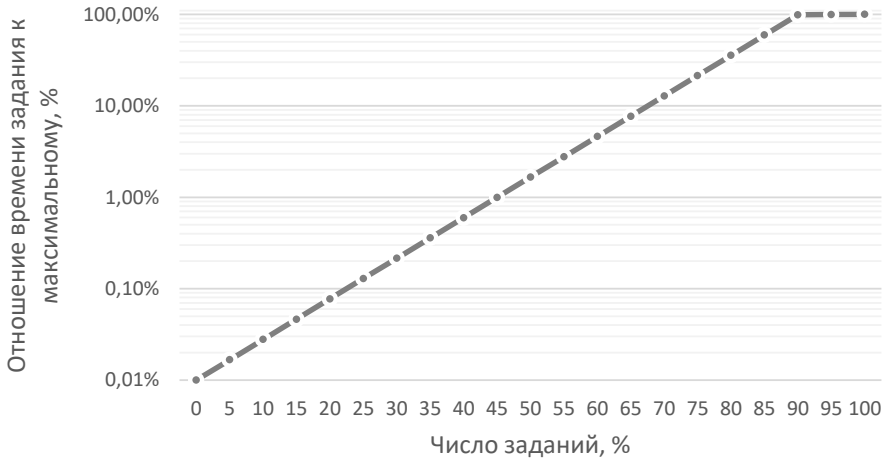


Рис. 4. Распределение времен заданий

Для исследования разработанного алгоритма поиск ресурсов для заданий производился тремя различными способами: методом $Find\_Systems$ без применения разработанной метрики ($Find\_Systems$); методом $Find\_Systems_{MSS}$ с применением разработанной метрики ($Find\_Systems + MSS$); методом, который изначально функционировал на кластерах с сетью Ангара ($base$).

Исследования проводились на окнах заданий размера $1, 2, 4, 8, 16, 32, 64, 128$.

На рис. 5 представлено среднее значение утилизации вычислительного кластера по всем системам в зависимости от размера окна. Разработанный алгоритм с применение разработанной метрикой оценки фрагментированности в данных условиях в среднем дает увеличение на $0,5$ % относительно алгоритма без учета фрагментированности системы и в среднем на 7 % относительно базового. С ростом размера окна утилизация во всех экспериментах увеличивается.

На рис. 6 представлено среднее значение времени нахождения задания в очереди по всем системам в зависимости от размера окна. Метод $Find\_Systems + MSS$ в среднем дал прирост на 2 % относительное метода $Find\_Systems$ и на $36,6$ % относительно $base$. С ростом размера окна значение времени ожидания задания в очереди в среднем значительно сокращается.

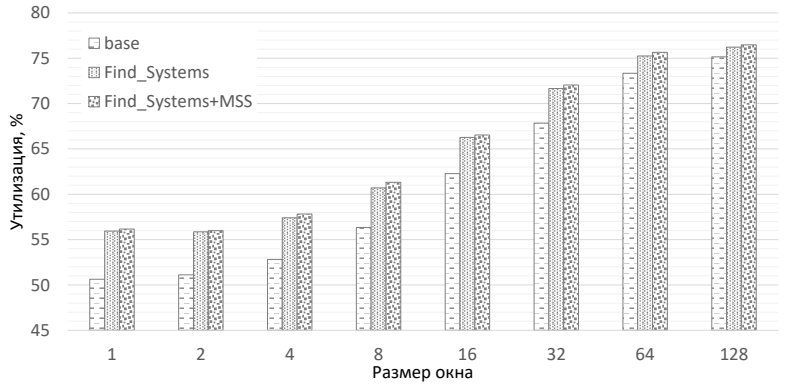


Рис. 5. Сравнение утилизации вычислительного кластера для различных систем, методов поиска и размеров окон

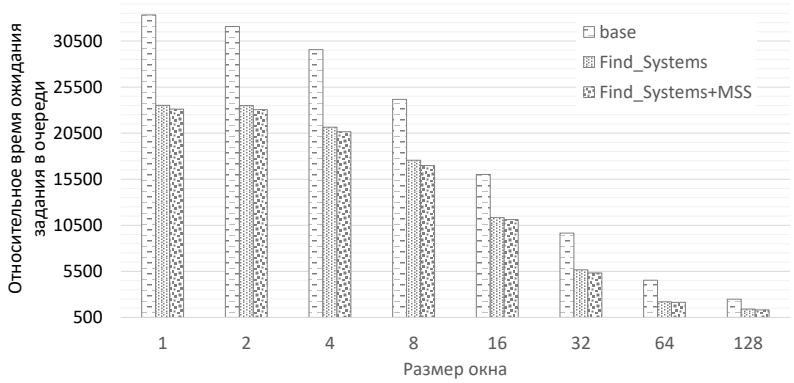


Рис. 6. Сравнение средних значений времени нахождения задания в очереди относительно запрошенного времени для различных систем, методов поиска и размеров окон

Заключение

В данной работе представлен метод выбора вычислительных узлов для потока пользовательских заданий, сокращающий фрагментацию вычислительной системы в сети Ангара с топологией многомерный тор. Метод основан на алгоритме выделения ресурсов для задания таким образом, чтобы максимизировать оставшееся пространство в многомерном торе. Это достигается построением прямоугольников максимального размера, которые возможно вписать в систему после размещения очередного пользовательского задания. Каждое множество узлов, подходящее для размещения задания, оценивается предложенной функцией, учитывающей размер и количество найденных прямоугольников максимального размера.

Проведено исследование применимости данного метода в политике поиска ресурсов для пользовательских заданий, основанной на выборе первого подходящего задания (First-Fit) в некотором рассматриваемом окне заданий.

Разработан метод генерации синтетической очереди пользовательских заданий, параметры которой приближены к реально возможной и основанны на данных, полученных с вычислительного кластера Desmos на базе сети Ангара.

Проведено экспериментальное исследование разработанного алгоритма для различных конфигураций вычислительных систем с топологией многомерный тор с общим числом узлов 32, 36, 64, 96 и 144, при этом рассмотрены 3х-мерные и 4х-мерные конфигурации топологий. Были рассмотрены различные варианты размера окна заданий.

Показана эффективность использования изменения порядка заданий для вычислительных систем с топологией многомерный тор с применением метода по оптимизации фрагментации. Разработанный метод с учет фрагментированности системы в среднем дает прирост утилизации 7 % и на 36, 6 % сокращает значение время ожидания задания в очереди по сравнению с базовым методом.

Добавление возможности изменения порядка заданий увеличивает утилизацию вычислительных ресурсов, время ожидания задания в очереди сокращается. Однако условия применения этого механизма требуют дальнейшего исследования.

Литература

1. Агарков А.А., Исмагилов Т.Ф., Макагон Д.В., Семенов А.С., Симонов А.С. Результаты оценочного тестирования отечественной высокоскоростной коммуникационной сети Ангара // Суперкомпьютерные дни в России: Труды международной конференции (Москва, 26–27 сентября 2016 г.). М.: Изд-во МГУ, 2016. С. 626–639.
2. Симонов А.С., Макагон Д.В., Жабин И.А., Щербак А.Н., Сыромятников Е.Л., Поляков Д.А. Первое поколение высокоскоростной коммуникационной сети «Ангара» // Наукоемкие технологии. 2014. Т. 15. № 1. С. 21–28.
3. Puente V., Beivide R., Gregorio J.A., Prellezo J.M., Duato J., Izu C. Adaptive Bubble Router: a Design to Improve Performance in Torus Networks // Proceedings of the International Conference Parallel Processing (ICPP). 1999. P. 58–67. DOI: 10.1109/ICPP.1999.797388.
4. Adiga N.R., Blumrich M., Chen D. Blue Gene/L Torus Interconnection Network // IBM Journal of Research and Development. 2005. Vol. 49. No. 2. P. 265–276. DOI: 10.1147/rd.492.0265.
5. Scott S.L. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. 1996.
6. Пожилов И.А., Семенов А.С., Макагон Д.В. Алгоритм определения связности сети с топологией «многомерный тор» с отказами для детерминированной маршрутизации // Программная инженерия. 2015. № 3. С. 13–19.
7. Lan Z., Tang W., Wang J., Yang X., Zhou Z., Zheng X. Balancing Job Performance with System Performance via Locality-aware Scheduling on Torus-connected Systems // 2014 IEEE International Conference on Cluster Computing (CLUSTER). 2014. P. 140–148. DOI: 10.1109/CLUSTER.2014.6968751.
8. IBM Redbooks Publication: IBM System Blue Gene Solution: Blue Gene/Q System Administration. 2013. 282 p.
9. Tang W., Lan Z., Desai N., Buettner D., Yu Y. Reducing Fragmentation on Torus-Connected Supercomputers // Proceedings of the 2011 IEEE International Parallel Distributed Processing Symposium (IPDPS'11). IEEE Computer Society, Washington, DC, USA. 2011. P. 828–839 DOI: 10.1109/IPDPS.2011.82.
10. Cray Document: Managing System Software for Cray XE and Cray XT Systems. 2010.
11. Schwiegelshohn U., Yahyapour R. Analysis of First-Come-First-Serve Parallel Job Scheduling // SODA. 1998. Vol. 98. P. 629–638.
12. Полежаев П.Н. Исследование алгоритмов планирования параллельных задач для кластерных вычислительных систем с помощью симулятора // Параллельные

вычислительные технологии (ПаВТ'2010): Труды международной конференции (Уфа, 29 марта–2 апреля 2010 г.). Челябинск: Издательский центр ЮУрГУ, 2010. С. 287–298.

13. Mu'alem A.W., Feitelson D.G. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling // IEEE Transactions on Parallel and Distributed Systems. 2001. Vol. 12. No. 6. P. 529–543. DOI: 10.1109/71.932708.

14. Henderson R.L. Job Scheduling Under the Portable Batch System // Workshop on Job Scheduling Strategies for Parallel Processing. Springer, Berlin, Heidelberg, 1995. P. 279–294.

15. Staples G. TORQUE Resource Manager // Proceedings of the 2006 ACM/IEEE Conf. on Supercomputing. ACM, 2006. P. 8.

16. Jackson D., Snell Q., Clement M. Core Algorithms of the Maui Scheduler // Workshop on Job Scheduling Strategies for Parallel Processing. Springer, Berlin, Heidelberg, 2001. P. 87–102.

17. Gentzsch W. Sun Grid Engine: Towards Creating a Compute Power Grid // Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on. IEEE, 2001. P. 35–36.

18. Баранов А.В., Смирнов С.В., Храмцов М.Ю., Шарф С.В. Модернизация СУПЗ МВС-1000 // Материалы Всероссийской научной конференции «Научный сервис в сети Интернет». Новороссийск, 2008.

19. SchedMD L. L. C. SLURM Workload Manager. 2018. https://slurm.schedmd.com/overview.html (дата обращения: 20.09.2018)

20. Мукосей А.В., Семенов А.С. Оптимизация фрагментации при выделении ресурсов для высокопроизводительных вычислительных систем с сетью Ангара // Параллельные вычислительные технологии (ПаВТ'2018): Труды международной научной конференции (Ростов-на-Дону, 2–6 апреля 2018 г.). Челябинск: Издательский центр ЮУрГУ, 2018. С. 310–318.

21. Woo S.H. Task Scheduling in Distributed Computing Systems with a Genetic Algorithm // High Performance Computing on the Information Superhighway. 1997. HPC Asia'97. IEEE. 1997. P. 301–305.

22. Вечер В.С., Кондратюк Н.Д., Смирнов Г.С., Стегайлов В.В. Гибридный суперкомпьютер на базе сети Ангара для задач вычислительного материаловедения // Суперкомпьютерные дни в России: Труды международной конференции (Москва, 25–26 сентября 2017 г.). М.: Изд-во МГУ, 2017. С. 557–571.

23. Мукосей А.В., Семенов А.С., Приближенный алгоритм выбора оптимального подмножества узлов в коммуникационной сети Ангара с отказами // Вычислительные методы и программирование. 2017. Т. 18. С. 53–64.

24. Баранов А.В., Киселёв Е.А., Ляховец Д.С. Квазипланировщик для использования простаивающих вычислительных модулей многопроцессорной вычислительной системы под управлением СУППЗ // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2014. Т. 3. № 4. С. 75–84. DOI: 10.14529/cmse140405.

25. Gonçalves J.F., Resende M.G.C. A Parallel Multi-population Biased Random-key Genetic Algorithm for a Container Loading Problem // Computers & Operations Research. February 2012. Vol. 39. No. 2. P. 179–190. DOI: 10.1016/j.cor.2011.03.009.

Мукосей Анатолий Викторович, научный сотрудник, сектор управления разработки вычислительной техники, акционерное общество «Научно-исследовательский центр электронной вычислительной техники» (Москва, Российская Федерация)

Семенов Александр Сергеевич, к.т.н., зам. начальника отдела архитектуры и программного обеспечения суперкомпьютеров, акционерное общество «Научно-исследовательский центр электронной вычислительной техники» (Москва, Российская Федерация)

Симонов Алексей Сергеевич, к.т.н., первый заместитель генерального директора АО «Научно-исследовательский центр электронной вычислительной техники» (Москва, Российская Федерация)

DOI: 10.14529/cmse190101

ALLOCATION OPTIMIZATION FOR REDUCING RESOURCE UTILIZATION IN ANGARA HIGH-SPEED INTERCONNECT

© 2019 A.V. Mukosey, A.S. Semenov, A.S. Simonov

JSC "NICEVT"

(Varshavskoye shosse 125, building 15, Moscow, 117587 Russia)

E-mail: mukosey@nicevt.ru, semenov@nicevt.ru, simonov@nicevt.ru

Received: 22.07.2018

This paper considers a high-speed interconnect with a multidimensional topology. The paper is devoted to the optimization of fragmentation resulting from sequential allocation of computing nodes in a supercomputer provided that network traffic from different user's tasks should not overlap. This paper is the continuation of resources fragmentation optimization work. In this work, the method for scheduling tasks based on the policy of choosing the first suitable task (First-Fit) in a certain task window has been added to the accounting for fragmentation when choosing nodes. A set of different computer systems with three-dimensional and four-dimensional topologies was considered. The minimum system size is 32 computing nodes, and the maximum is 144. A synthetic queue of tasks is set for each system. The parameters of the synthetic queues are close to real ones and are based on data received from the Desmos cluster equipped with Angara interconnect. The average utilization of the resources of the computer system and the average waiting time for the tasks in the queue is chosen as a method quality criterion. Various sizes of task windows have been evalauated. The study showed that the increase of the resources utilization for the proposed method averaged 7 % compared to the base method, and the average time spent in queue was reduced by 36.6 %.

Keywords: Angara interconnect, multidimensional torus, deterministic routing, direction ordered routing, fragmentation, allocation.

FOR CITATION

Mukosey A.V., Semenov A.S., Simonov A.S. Allocation Optimization for Reducing Resource Utilization in Angara High-speed Interconnect. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2019. vol. 8, no. 1. pp. 5–19. (in Russian) DOI: 10.14529/cmse190101.

References

1. Agarkov A.A., Ismagilov T.F., Makagon D.V. Performance Evaluation of the Angara Interconnect. *Superkomp'yuternye dni v Rossii: Trudy mezhdunarodnoi konferentsii (Moskva, 26–27 sentyabrya 2016)* [Russian Supercomputing Days: Proceedings of the International Scientific Conference (Moscow, Russia, September, 26–27, 2016)]. Moscow, Publishing of Moscow State University, 2016. pp. 626–639. (in Russian)

2. Simonov A.S., Makagon D.V., Zhabin I.A., Shcherbak A.N., Syromyatnikov E.L., Polyakov D.A. The First Generation of Angara High-Speed Interconnect. *Naukoemkie tekhnologii* [Science Intensive Technologies]. 2014. vol. 15. no. 1. pp. 21–28. (in Russian)

3. Puente V., Beivide R., Gregorio J.A., Prellezo J.M., Duato J., Izu C. Adaptive Bubble Router: a Design to Improve Performance in Torus Networks. Proceedings of the International Conference Parallel Processing (ICPP). 1999. pp. 58–67. DOI: 10.1109/ICPP.1999.797388.

4. Adiga N.R., Blumrich M., Chen D.. Blue Gene/L Torus Interconnection Network. IBM Journal of Research and Development. 2005. vol. 49. no. 2. pp. 265–276. DOI: 10.1147/rd.492.0265.

5. Scott S.L., et al. The Cray T3E Network: Adaptive Routing in a High. Performance 3D Torus. 1996.

6. Pozhilov I.A., Semenov A.S., Makagon D.V., Connectivity Problem Solution for Direction Ordered Deterministic Routing in nD Torus. Software Engineering. 2015. no. 3. pp. 13–19. (in Russian)

7. Lan Z., Tang W., Wang J., Yang X., Zhou Z., Zheng X. Balancing job Performance with System Performance via Locality-aware Scheduling on Torus-connected Systems. 2014 IEEE International Conference on Cluster Computing (CLUSTER). 2014. pp. 140–148. DOI: 10.1109/CLUSTER.2014.6968751.

8. IBM Redbooks Publication: IBM System Blue Gene Solution: Blue Gene/Q system administration. 2013. 282 p.

9. Tang W., Lan Z., Desai N., Buettner D., Yu Y. Reducing Fragmentation on Torus-Connected Supercomputers. In Proceedings of the 2011 IEEE International Parallel Distributed Processing Symposium (IPDPS'11). IEEE Computer Society, Washington, DC, USA. 2011. pp. 828–839 DOI: 10.1109/IPDPS.2011.82.

10. Cray Document: Managing System Software for Cray XE and Cray XT Systems. 2010.

11. Schwiegelshohn U., Yahyapour R. Analysis of First-Come-First-Serve Parallel Job Scheduling. SODA. 1998. vol. 98. pp. 629–638.

12. Polezhaev P.N. The Study of Parallel Job Scheduling Algorithms for Cluster Computing Systems Using a Simulator. *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. pp. 287—298. (in Russian)

13. Mu'alem A.W., Feitelson D.G. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. IEEE Transactions on Parallel and Distributed Systems. 2001. vol. 12. no. 6. pp. 529–543. DOI: 10.1109/71.932708.

14. Henderson R.L. Job Scheduling Under the Portable Batch System. Workshop on Job Scheduling Strategies for Parallel Processing. Springer, Berlin, Heidelberg, 1995. pp. 279–294.

15. Staples G. TORQUE Resource Manager. Proceedings of the 2006 ACM/IEEE conference on Supercomputing. ACM, 2006. pp. 8.

16. Jackson D., Snell Q., Clement M. Core Algorithms of the Maui Scheduler. Workshop on Job Scheduling Strategies for Parallel Processing. Springer, Berlin, Heidelberg, 2001. pp. 87–102.

17. Gentzsch W. Sun Grid Engine: Towards Creating a Compute Power Grid. Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on. IEEE, 2001. pp. 35–36.

18. Baranov A.V., Smirnov S.V., Khramtsov M.Yu., Sharf S.V. *Modernizatsiya SUPZ MVS-1000* [Modernization of the SUPZ MBS-1000]. *Materialy Vserossiiskoi nauchnoi konferentsii "Nauchnyi servis v seti Internet"* [Materials of the All-Russian Scientific Conference "Scientific Service on the Internet"]. Novorossiysk. 2008.

19. SchedMD L. L. C. SLURM Workload Manager. 2018. https://slurm.schedmd.com/overview.html (accessed: 20.09.2018)

20. Mukosey A.V., Semenov A.S. Allocation Optimization for Reducing Resource Fragmentation in Angara High-speed Interconnect. *Parallelnye vychislitelnye tekhnologii (PaVT'2010): Trudy mezhdunarodnoj nauchnoj konferentsii (Rostov-na-Donu, aprel' 2–6 2018)* [Parallel Computational Technologies (PCT'2018): Proceedings of the International Scientific Conference (Rostov-na-Donu, Russia, April, 2–6, 2018)]. Chelyabinsk, Publishing of the South Ural State University, 2018. pp. 310–318. (in Russian)

21. Woo S.H. Task Scheduling in Distributed Computing Systems with a Genetic Algorithm. High Performance Computing on the Information Superhighway. 1997. HPC Asia'97. IEEE. 1997. pp. 301–305.

22. Vecher V.S., Kondratyuk N.D., Smirnov G.S., Stegailov V.V. Angara-based hybrid supercomputer for efficient acceleration of computational materials science studies. *Superkomp'yuternye dni v Rossii: Trudy mezhdunarodnoj konferentsii (Moskva, sentyabr' 25–26 2017)* [Russian Supercomputing Days: Proceedings of the International Conference (Moscow, Russia, September, 25–26, 2017)]. Moscow, Publishing of Moscow State University, 2017. pp. 557–571. (in Russian)

23. Mukosey A.V., Semenov A.S. An Approximate Algorithm for Choosing the Optimal Subset of Nodes in the Angara Interconnect with Failures. Numerical methods and Programming. 2017. vol. 18. pp. 53–64. (in Russian)

24. Baranov A.V., Kiselev E.A., Lyakhovets D.S. The Quasi Scheduler for Utilization of Multiprocessing Computing System's Idle Resources Under Control of the Management System of the Parallel Jobs. *Vestnik Yuzho-Uralskogo gosudarstvennogo universiteta. Seriya "Matematicheskoe modelirovanie i programmirovanie"* [Bulletin of South Ural State University. Series: Mathematical Modeling, Programming & Computer Software]. 2014. vol. 3. no. 4. pp. 75–84. (in Russian) DOI: 10.14529/cmse140405.

25. Gonçalves J.F., Resende M.G.C. A Parallel Multi-Population Based Random-key Genetic Algorithm for a Container Loading Problem. Computers & Operations Research. February 2012. vol. 39. no. 2. pp. 179–190. DOI: 10.1016/j.cor.2011.03.009.

УДК 519.688, 004.272.2 **DOI: 10.14529/cmse190102**

ИССЛЕДОВАНИЕ МАСШТАБИРУЕМОСТИ АЛГОРИТМА ЧИММИНО ДЛЯ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ НЕРАВЕНСТВ НА КЛАСТЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ*

© 2019 И.М. Соколинская, Л.Б. Соколинский
Южно-Уральский государственный университет
(454080 Челябинск, пр. им. В.И. Ленина, д. 76)
E-mail: irina.sokolinskaya@susu.ru, leonid.sokolinsky@susu.ru
Поступила в редакцию: 05.05.2018

Работа посвящена исследованию масштабируемости алгоритма Чиммино для решения систем неравенств. Данный алгоритм является типичным представителем класса итерационных проекционных алгоритмов для решения систем линейных уравнений и неравенств. Для аналитического анализа масштабируемости используется модель параллельных вычислений BSF (Bulk Synchronous Farm). Дается представление алгоритма Чиммино в виде операций над списками с использованием функций высшего порядка *Map* и *Reduce*. Выводятся аналитические оценки границы масштабируемости алгоритма для многопроцессорных вычислительных систем с распределенной памятью. Приводятся данные о реализации алгоритма Чиммино над списками на языке C++ с использованием программного шаблона BSF и библиотеки параллельного программирования MPI. Демонстрируются результаты масштабных вычислительных экспериментов, выполненных на кластерной вычислительной системе. На основе экспериментальных результатов дается анализ адекватности оценок, полученных аналитическим путем с помощью стоимостных метрик модели BSF.

Ключевые слова: алгоритм Чиммино, система линейных неравенств, итерационный алгоритм, проекционный алгоритм, релаксация, модель параллельных вычислений BSF, оценка масштабируемости, эффективность распараллеливания, кластерные вычислительные системы.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Соколинская И.М., Соколинский Л.Б. Исследование масштабируемости алгоритма Чиммино для решения систем линейных неравенств на кластерных вычислительных системах // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2019. Т. 8, № 1. С. 20–35. DOI: 10.14529/cmse190102.

Введение

Задача решения системы линейных неравенств возникает во многих областях вычислительной математики. В качестве примеров можно привести линейное программирование [1, 2], восстановление изображений по проекциям [3], обработку изображений в магнитно-резонансной томографии [4], радиационную терапию с модулируемой интенсивностью [5]. В настоящее время известно достаточно много методов решения систем линейных неравенств, среди которых можно выделить класс «самоисправляющихся» итерационных методов, допускающих эффективное распараллеливание. Пионерскими работами здесь являются публикации [6, 7], в которых предложен метод релаксаций Агмона—Моцкина—Шоенберга для решения систем линейных неравенств. Метод релаксаций можно

* Работа рекомендована Программным комитетом международной научной конференции «Суперкомпьютерные дни в России 2018».

отнести к алгоритмам проекционного типа, использующим операцию ортогональной проекции на гиперплоскость в Евклидовом пространстве. Одним из первых итерационных алгоритмов проекционного типа был алгоритм Чиммино (Cimmino) [8], предназначенный для решения систем линейных уравнений и неравенств. Алгоритм Чиммино оказал большое влияние на развитие вычислительной математики [9]. Обобщениям и расширениям алгоритма Чиммино было посвящено значительное количество работ (см., например, [3, 10–13]).

Системы линейных неравенств, возникающие при решении практических задач, во многих случаях могут включать в себя сотни миллионов переменных, при этом количество неравенств составляет величину того же порядка [2]. В этом случае актуальным становится вопрос разработки масштабируемых параллельных алгоритмов для решения сверхбольших систем линейных неравенств на многопроцессорных системах с распределенной памятью. При создании параллельных алгоритмов для больших многопроцессорных систем важно уже на ранней стадии разработки алгоритма (до написания программы) получить аналитические оценки его масштабируемости. Для этой цели используются различные модели параллельных вычислений [14]. В настоящее время известно большое количество различных параллельных вычислительных моделей. Наиболее известными среди них являются модели PRAM [15], BSP [16] и LogP [17]. Указанные модели подверглись обобщениям и уточнениям, породив целые семейства, насчитывающие десятки параллельных вычислительных моделей (см., например, [18–20]). Задача разработки новых параллельных вычислительных моделей не утратила актуальности и в настоящее время. Это объясняется тем, что невозможно создать модель параллельных вычислений – «хорошую во всех отношениях». Поэтому необходимо ограничиваться определенными многопроцессорными архитектурами и определенными классами алгоритмов. В работе [21] была предложена модель параллельных вычислений BSF (Bulk Synchronous Farm), ориентированная на кластерные вычислительные системы и алгоритмы итерационного типа. Указанная модель позволяет с высокой точностью оценить границу масштабируемости параллельного итерационного алгоритма до написания программы. Пример использования модели BSF приводится в работе [22].

Целью настоящей статьи является исследование масштабируемости алгоритма Чиммино для решения больших систем линейных неравенств на многопроцессорных системах с распределенной памятью путем использования модели параллельных вычислений BSF. Статья имеет следующую структуру. В разделе 1 делается постановка задачи и дается описание алгоритма Чиммино для решения системы линейных неравенств. В разделе 2 строится представление алгоритма Чиммино в виде операций над списками с использованием функций высшего порядка *Map* и *Reduce*, определяемых формализмом Бёрда—Миртенса. Раздел 3 посвящен аналитическому исследованию масштабируемости алгоритма Чиммино над списками с помощью метрик модели BSF; приводятся итоговые формулы для оценки ускорения и эффективности распараллеливания; вычисляется граница масштабируемости алгоритма в зависимости от размера задачи. В разделе 4 дается информация о реализации алгоритма Чиммино над списками, выполненной на языке C++ с использованием программного каркаса BSF и библиотеки параллельного программирования MPI, и приводится сравнение результатов, полученных аналитическим и экспериментальным путем. В заключении суммируются полученные результаты и намечаются направления дальнейших исследований.

1. Алгоритм Чиммино для неравенств

Пусть в евклидовом пространстве $\mathbb{R}^n$ задана совместная система линейных неравенств

$$l_i(x) = \left\langle a_i, x \right\rangle - b_i \leq 0 \quad (i = 1,\ldots,m) \tag{1}$$

(угловыми скобками мы обозначаем скалярное произведение). Необходимо найти одно любое решение системы (1). Для решения этой задачи удобно использовать геометрический язык. В соответствии с этим $x = (x_1,\ldots,x_n)$ рассматривается как точка в n-мерном евклидовом пространстве $\mathbb{R}^n$, а каждое неравенство $l_i(x) \leq 0$ — как полупространство P_i. Таким образом, множество решений системы (1) можно представить как выпуклый n-мерный многогранник $M = \bigcap_{i=1}^{m} P_i$. Каждое уравнение $l_i(x) = 0$ определяет гиперплоскость H_i:

$$H_i = \left\{ x \in \mathbb{R}^n \mid \left\langle a_i, x \right\rangle = b_i \right\}. \tag{2}$$

Ортогональная проекция $\pi_{H_i}(x)$ точки x на гиперплоскость H_i вычисляется по следующей формуле:

$$\pi_{H_i}(x) = x + \frac{b_i - \left\langle a_i, x \right\rangle}{\left\| a_i \right\|^2} a_i, \tag{3}$$

где $\left\| \cdot \right\|$ — евклидова норма. Определим *ортогональное отражение* точки x относительно гиперплоскости H_i следующим образом

$$\rho_{H_i}(x) = \pi_{H_i}(x) - x = \frac{b_i - \left\langle a_i, x \right\rangle}{\left\| a_i \right\|^2} a_i. \tag{4}$$

Алгоритм Чиммино для неравенств с равными весовыми коэффициентами состоит из следующих шагов:

Шаг 1. $k := 0$; $x_0 := \mathbf{0}$.

Шаг 2. $x_{k+1} := x_k + \dfrac{\lambda}{m} \sum\limits_{i=1}^{m} \rho_{H_i}(x_k)$.

Шаг 3. Если $\left\| x_{k+1} - x_k \right\|^2 < \varepsilon$, перейти на шаг 5.

Шаг 4. $k := k + 1$; перейти на шаг 2.

Шаг 5. Стоп.

В качестве начального приближения x_0 в алгоритме Чиммино может быть взят произвольный вектор в $\mathbb{R}^n$. На шаге 1 в качестве начального приближения x_0 берется нулевой вектор. На шаге 2 организуется итерационный процесс:

$$x_{k+1} = x_k + \frac{\lambda}{m} \sum_{i=1}^{m} \rho_{H_i}(x_k). \tag{5}$$

Каждое новое приближение x_{k+1} получается путем добавления к текущему приближению x_k центра масс ортогональных отражений точки x_k относительно гиперплоскостей $H_1,\ldots,H_m$, умноженного на *коэффициент релаксации* λ. Известно [10], что при $0 < \lambda < 2$ итерационный процесс (5) сходится к точке, принадлежащей многограннику M.

2. Представление алгоритма Чиммино в виде операций над списками

Для получения аналитических оценок алгоритма в соответствии с метриками модели BSF он должен быть представлен в виде операций над списками с использованием функций высшего порядка *Map* и *Reduce*, определяемых формализмом Бёрда—Миртенса (Bird—Meertens formalism) [23]. Для заданных функции $F : \mathbb{A} \to \mathbb{B}$ и списка $[a_1,\ldots,a_m]$ функция высшего порядка *Map* формирует новый список той же длины путем применения функции F ко всем элементам списка $[a_1,\ldots,a_m]$:

$$Map(F,[a_1,\ldots,a_m]) = [F(a_1),\ldots,F(a_m)]\,. \tag{6}$$

Для заданных бинарной ассоциативной операции $\oplus : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ и списка $[b_1,\ldots,b_m]$ функция высшего порядка *Reduce* редуцирует список $[b_1,\ldots,b_m]$ к одному элементу путем многократного применения операции $\oplus$ к элементам списка:

$$\mathrm{Reduce}(\oplus,[b_1,\ldots,b_m]) = b_1 \oplus \ldots \oplus b_m\,. \tag{7}$$

В контексте алгоритма Чиммино определим список L_{Map} следующим образом:

$$L_{Map} = [i_1,\ldots,i_m], \tag{8}$$

где $i_k \in \{1,\ldots,m\}$ и $i_k \neq i_l$ при $k \neq l$ $(k,l = 1,\ldots,m)$. Другими словами, L_{Map} — список номеров неравенств системы (1), взятых в некотором порядке. Для произвольного $x \in \mathbb{R}^n$ определим функцию $F_x : \{1,\ldots,m\} \to \mathbb{R}^n$:

$$F_x(i) = \rho_{H_i}(x) \tag{9}$$

для всех $i \in \{1,\ldots,m\}$. Иначе говоря, функция $F_x(i)$ вычисляет ортогональное отражение точки x относительно гиперплоскости H_i. Для произвольного $x \in \mathbb{R}^n$ определим список $L^{(x)}_{Reduce} \subset \mathbb{R}^n$ следующим образом:

$$L^{(x)}_{Reduce} = [F_x(i_1),\ldots,F_x(i_m)]. \tag{10}$$

Список $L^{(x)}_{Reduce}$ включает в себя ортогональные отражения точки x на гиперплоскости $H_1,\ldots,H_m$, взятые в порядке, определяемом списком L_{Map}. Таким образом, список $L^{(x)}_{Reduce}$ получается из списка L_{Map} путем применения к нему функции высшего порядка *Map*, использующей в качестве параметра функцию F_x:

$$L^{(x)}_{Reduce} = Map(F_x, L_{Map})\,. \tag{11}$$

Определим бинарную ассоциативную операцию $\oplus : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ следующим образом:

$$x \oplus y = x + y \tag{12}$$

для любых $x, y \in \mathbb{R}^n$. В данном случае операция $\oplus$ выполняет обычное сложение векторов. Тогда сумма ортогональных отражений точки x может быть получена путем применения к списку $L^{(x)}_{Reduce}$ функции высшего порядка *Reduce*, использующей в качестве параметра операцию сложения векторов $\oplus$:

$$\sum_{i=1}^{m} \rho_{H_i}(x) = Reduce(\oplus, L^{(x)}_{Reduce}). \tag{13}$$

Теперь мы можем записать алгоритм Чиммино в виде операций над списками:

Шаг 1. $k := 0$; $x_0 := \mathbf{0}$; $L_{Map} := [1,\ldots,m]$.

Шаг 2. $L^{(x_k)}_{Reduce} := Map(F_{x_k}, L_{Map})$.

Шаг 3. $s := Reduce(\oplus, L^{(x_k)}_{Reduce})$.

Шаг 4. $x_{k+1} := x_k + \dfrac{\lambda}{m} \cdot s$

Шаг 5. Если $\left\| x_{k+1} - x_k \right\|^2 < \varepsilon$, перейти на шаг 7.

Шаг 6. $k := k + 1$; перейти на шаг 2.

Шаг 7. Стоп.

Модель BSF предполагает, что алгоритм выполняется вычислительной системой, состоящей из одного узла-мастера и K узлов-рабочих ($K > 0$). При этом шаг 1 выполняется и мастером, и рабочими в ходе инициализации итерационного процесса; шаг 2 (*Map*) выполняется только на узлах-рабочих; шаг 3 (*Reduce*) выполняется на узлах-рабочих и частично на узле-мастере; шаги 4–6 выполняются только на узле-мастере. В модели BSF предполагается, что все арифметические операции (сложение и умножение), а также операции сравнения над числами с плавающей точкой занимают одинаковое время τ_{op}.

3. Аналитическое исследование масштабируемости

Введем следующие обозначения для анализа масштабируемости алгоритма Чиммино:

c_s — количество вещественных чисел, передаваемых от мастера одному рабочему;

c_{Map} — количество арифметических операций, выполняемых на шаге *Map* (шаг 2 алгоритма);

c_a — количество арифметических операций, необходимых для выполнения одной операции $\oplus$;

c_r — количество вещественных чисел, передаваемых от одного рабочего мастеру;

c_p — количество арифметических операций и операций сравнения, выполняемых мастером на шагах 4 и 5 алгоритма.

Вычислим указанные значения. В начале каждой итерации мастер передает каждому рабочему очередное приближение x_k, являющееся вектором размерности n. Следовательно:

$$c_s = n. \tag{14}$$

Подсчитаем количество арифметических операций, выполняемых на шаге *Map*. Для каждого элемента списка L_{Map} вычисляется один вектор по формуле (4). Заметим, что величины $\|a_i\|^2$ ($i = 1,\ldots,m$) не зависят от x_k, и поэтому могут быть вычислены заранее на этапе инициализации. С учетом этого количество операций при вычислении одного ортогонального отражения точки x_k составляет $3n+1$. Умножив это число на количество неравенств, получаем

$$c_{Map} = m(3n+1). \tag{15}$$

При выполнении шага *Reduce* список L_{Reduce}, состоящий из m векторов, делится поровну между всеми рабочими. Везде далее мы предполагаем, что $K \leq m$. Для простоты мы будем считать, что m всегда кратно количеству рабочих K. Операция $\oplus$ в данном случае означает сложение двух векторов размерности n. Следовательно:

$$c_a = n. \tag{16}$$

Вектор, полученный каждым рабочим на шаге *Reduce*, пересылается мастеру. Значит:

$$c_r = n. \tag{17}$$

Выполнение шага 4 требует $2n$ операций (константу λ/m мы предполагаем вычисленной заранее). Выполнение шага 5 требует $3n-1$ арифметических операций и одну операцию сравнения. Отсюда получаем следующую формулу:

$$c_p = 5n. \tag{18}$$

Положим τ_{op} — время, затрачиваемое рабочим на выполнение одной арифметической операции, τ_{tr} — время, затрачиваемое на пересылку по сети одного вещественного числа без учета латентности. Тогда мы получаем следующие значения для стоимостных параметров модели BSF [21] в случае алгоритма Чиммино:

$$t_s = \tau_{tr} n; \tag{19}$$

$$t_{Map} = m(3n+1)\tau_{op}; \tag{20}$$

$$t_a = n\tau_{op}; \tag{21}$$

$$t_r = n\tau_{tr}; \tag{22}$$

$$t_p = 5n\tau_{op}. \tag{23}$$

Уравнение (19), полученное на основе (14), дает оценку времени t_s, затрачиваемого мастером на передачу сообщения одному рабочему без учета латентности. Уравнение (20), полученное с использованием (15), обозначает время, затрачиваемое одним рабочим на обработку всего списка *Map*. Формула (21) получена с использованием формулы (16). В соответствии со стоимостной метрикой модели BSF t_a обозначает время выполнения операции $\oplus$ (в данном случае – сложение векторов). Уравнение (22), полученное на основе (17), дает оценку времени t_r, затрачиваемого мастером на получение сообщения от одного

рабочего без учета латентности. Формула (23), полученная на основе (18), вычисляет время t_p, затрачиваемое мастером на вычисление очередного приближения и проверку условия завершения.

В соответствии с этим время решения задачи системой в составе одного мастера и одного рабочего ($K = 1$) может быть вычислено следующим образом:

$$\begin{aligned} T_1 &= 2L + t_s + t_r + t_p + t_{Map} + lt_a \\ &= 2(L + \tau_{tr} n) + \tau_{op}\big(5n + m(3n+1) + (m-1)n\big). \end{aligned} \tag{24}$$

Время решения задачи системой в составе одного мастера и K рабочих может быть вычислено по следующей формуле:

$$\begin{aligned} T_K &= K\big(2L + t_s + t_r + t_a\big) + \frac{t_{Map} + lt_a}{K} - t_a + t_p \\ &= 2K(L + \tau_{tr} n + \tau_{op} n) + \tau_{op}\left(\frac{m(3n+1) + (m-1)n}{K} + 4n\right). \end{aligned} \tag{25}$$

При $m \to \infty$ формулы (24) и (25) асимптотически приближаются к следующим оценкам:

$$T_1 = 2(L + \tau_{tr} n) + \tau_{op}\big(5n + m(3n+1) + mn\big); \tag{26}$$

$$T_K = 2K(L + \tau_{tr} n + \tau_{op} n) + \tau_{op}\left(\frac{m(3n+1) + mn}{K} + 4n\right). \tag{27}$$

На основании формул (26) и (27) мы можем записать формулу для ускорения a как функцию от K:

$$a(K) = \frac{T_1}{T_K} = \frac{2(L + \tau_{tr} n) + \tau_{op}\big(5n + m(3n+1) + mn\big)}{2K(L + \tau_{tr} n + \tau_{op} n) + \tau_{op}\left(\frac{m(3n+1) + mn}{K} + 4n\right)}. \tag{28}$$

Для определения границы масштабируемости алгоритма Чиммино в соответствии с методикой, описанной в [21], вычислим производную $a'(K)$ и решим уравнение

$$a'(K) = 0. \tag{29}$$

Из формулы (28) получаем следующую формулу для производной ускорения:

$$a'(K) = \frac{\big(2(L + \tau_{tr} n) + \tau_{op}\big(5n + m(3n+1) + mn\big)\big) \cdot \left(\frac{m(3n+1) + mn}{K^2}\tau_{op} - 2(L + n\tau_{tr}) - \tau_{op} n\right)}{\left(2K(L + \tau_{tr} n + \tau_{op} n) + \tau_{op}\left(\frac{m(3n+1) + mn}{K} + 4n\right)\right)^2}. \tag{30}$$

Решим уравнение

$$\frac{\big(2(L + \tau_{tr} n) + \tau_{op}\big(5n + m(3n+1) + mn\big)\big) \cdot \left(\frac{m(3n+1) + mn}{K^2}\tau_{op} - 2(L + n\tau_{tr}) - \tau_{op} n\right)}{\left(2K(L + \tau_{tr} n + \tau_{op} n) + \tau_{op}\left(\frac{m(3n+1) + mn}{K} + 4n\right)\right)^2} = 0. \tag{31}$$

Разделив обе части уравнения (31) на положительную величину

$$2(L + \tau_{tr} n) + \tau_{op}\big(5n + m(3n+1) + mn\big)$$

и умножив на положительную величину

$$\left(2K(L+\tau_{tr}n+\tau_{op}n)+\tau_{op}\left(\frac{m(3n+1)+mn}{K}+4n\right)\right)^2$$

переходим к уравнению

$$\frac{m(3n+1)+mn}{K^2}\tau_{op}-2(L+n\tau_{tr})-\tau_{op}n=0,$$

откуда получаем

$$K=\sqrt{\frac{\big(m(3n+1)+mn\big)\tau_{op}}{2(L+n\tau_{tr})+n\tau_{op}}}.$$

Таким образом, уравнение (29) имеет только один корень $K_0=\sqrt{\big(m(3n+1)+mn\big)\tau_{op}\big/\big(2(L+n\tau_{tr})+n\tau_{op}\big)}$ на интервале $[1,+\infty)$. Легко видеть, что $a'(K)$ принимает на интервале $[1,K_0)$ только положительные значения, а на интервале $(K_0,+\infty)$ только отрицательные значения. Следовательно, в точке K_0 достигается максимум ускорения. Таким образом, граница масштабируемости K_{max} алгоритма Чиммино в соответствии с моделью BSF определяется следующей формулой:

$$K_{max}=\sqrt{\frac{\big(m(3n+1)+mn\big)\tau_{op}}{2(L+n\tau_{tr})+n\tau_{op}}}. \tag{32}$$

Упростим формулу (32). При $n,m\to\infty$ имеем

$$\big(m(3n+1)+mn\big)\tau_{op}\approx O(mn) \tag{33}$$

и

$$2(L+n\tau_{tr})+n\tau_{op}\approx O(n). \tag{34}$$

Подставив правые части формул (33) и (34) в (32), получим

$$K_{max}=\sqrt{\frac{O(mn)}{O(n)}},$$

что равносильно

$$K_{max}=\sqrt{O(m)}. \tag{35}$$

Таким образом, граница масштабируемости алгоритма Чиммино над списками растет пропорционально квадратному корню из количества неравенств.

В заключение этого раздела напишем формулу для оценки параллельной эффективности e как функции от K. С учетом формулы (28) имеем

$$e(K)=\frac{a(K)}{K}=\frac{2(L+\tau_{tr}n)+\tau_{op}\big(5n+m(3n+1)+mn\big)}{2K^2(L+\tau_{tr}n+\tau_{op}n)+\tau_{op}\big(m(3n+1)+mn+4nK\big)}. \tag{36}$$

4. Вычислительные эксперименты

Для верификации результатов, полученных аналитическим путем, мы выполнили реализацию алгоритма Чиммино на языке C++ с использованием программного каркаса BSF и библиотеки параллельного программирования MPI. Данная реализация свободно доступна в сети Интернет по адресу https://github.com/leonid-sokolinsky/BSF-Cimmino. Систему неравенств мы заимствовали из модельной масштабируемой задачи линейного программирования *Model-n* размерности n, приведенной в работе [24]. Количество неравенств в этой системе равно $m = 2n + 2$. Мы исследовали ускорение и эффективность распараллеливания алгоритма Чиммино на суперкомпьютере «Торнадо ЮУрГУ» [25]. Тестирование проводилось для размерностей 1500, 5000, 10000 и 16000. Одновременно мы построили для этих же размерностей графики ускорения и эффективности с использованием формул (28) и (36). Для этого экспериментальным путем были определены величины в секундах: $L = 1.5 \cdot 10^{-5}$, $\tau_{op} = 2.9 \cdot 10^{-8}$ и $\tau_{tr} = 1.9 \cdot 10^{-7}$. Результаты приведены на рис. 1–8. Во всех случаях аналитические оценки оказались очень близки к экспериментальным. Кроме этого, проведенные эксперименты показывают, что граница масштабируемости программы растет пропорционально корню квадратному из количества неравенств, что было предсказано аналитически с помощью формулы (35).

Заключение

В статье выполнено аналитическое исследование масштабируемости и эффективности распараллеливания итерационного алгоритма Чиммино для решения больших систем линейных неравенств на многопроцессорных системах с распределенной памятью. Для этого использована модель параллельных вычислений BSF (Bulk Synchronous Farm), основанная на парадигме «мастер-рабочие». Описана реализация алгоритма Чиммино в виде операций над списками с использованием функций высшего порядка *Map* и *Reduce*, определяемых формализмом Бёрда—Миртенса. Получена оценка границы масштабируемости алгоритма Чиммино над списками. Данная оценка показывает, что граница масштабируемости растет пропорционально корню квадратному из количества неравенств. Также получены формулы для оценки ускорения и эффективности распараллеливания алгоритма Чиммино над списками. Выполнена реализация алгоритма Чиммино над списками на языке C++ с использованием программного каркаса BSF и библиотеки параллельного программирования MPI. На кластерной вычислительной системе проведены масштабные эксперименты для определения фактических кривых ускорения и параллельной эффективности для задач с количеством переменных 1500, 5000, 10000, 16000 и количеством неравенств 3002, 10002, 20002, 32002 соответственно. Результаты экспериментов показали, что модель BSF с высокой точностью предсказывает границу масштабируемости алгоритма Чиммино над списками. В рамках дальнейших исследований мы планируем решить следующие задачи:

1) применить алгоритм Чиммино для реализации фазы Qwest алгоритма NSLP [2], предназначенного для решения сверхбольших нестационарных задач линейного программирования;

2) провести вычислительные эксперименты по решению сверхбольших задач линейного программирования на кластерной вычислительной системе в условиях динамического изменения исходных данных.

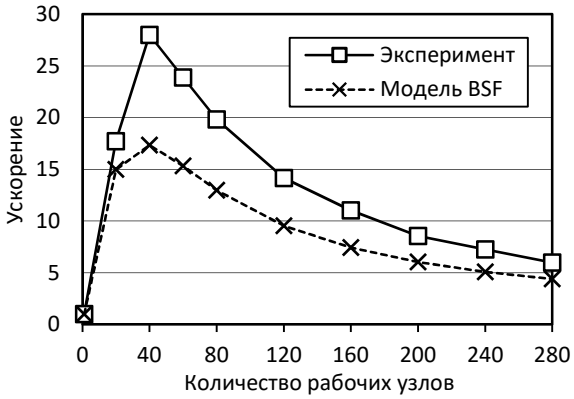


Рис. 1. Ускорение при $n = 1500$

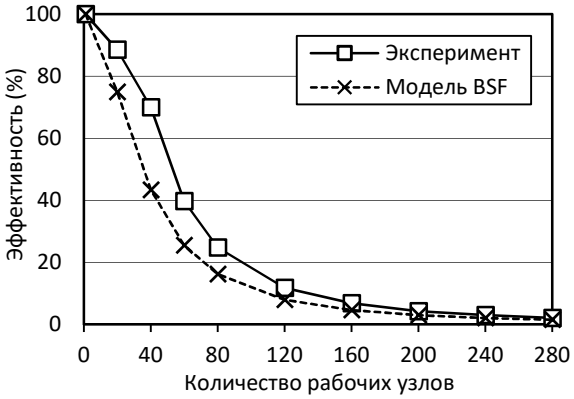


Рис. 2. Эффективность при $n = 1500$

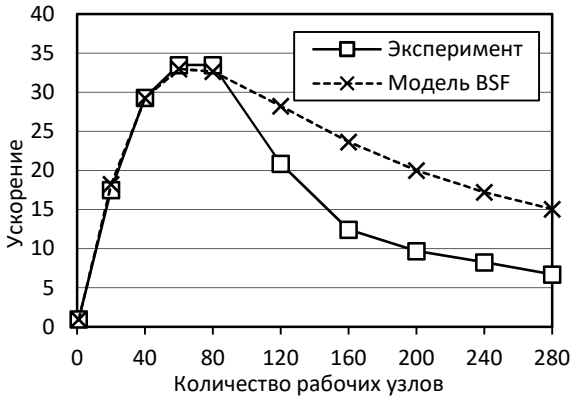


Рис. 3. Ускорение при $n = 5000$

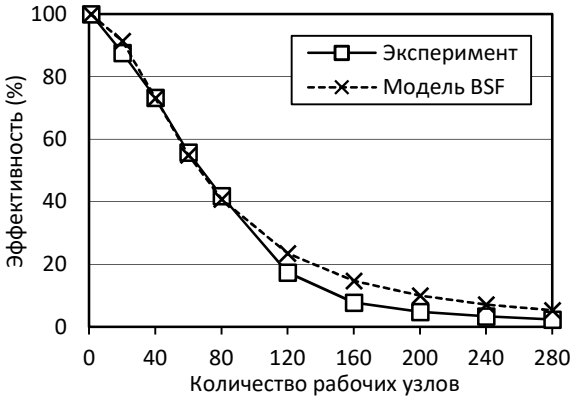


Рис. 4. Эффективность при $n = 5000$

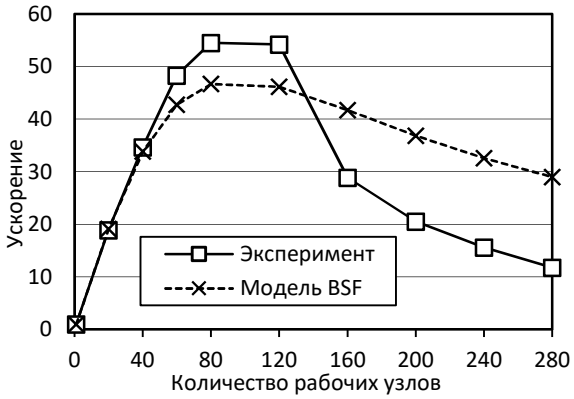


Рис. 5. Ускорение при $n = 10000$

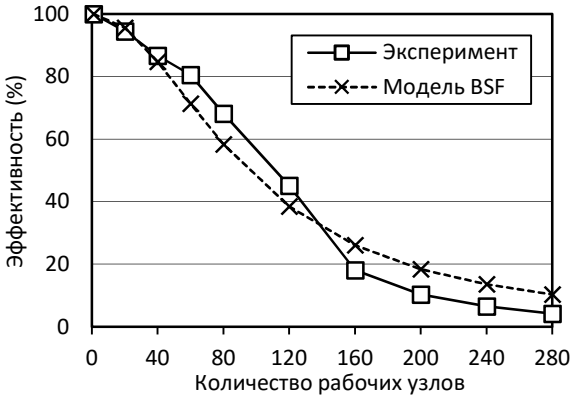


Рис. 6. Эффективность при $n = 10000$

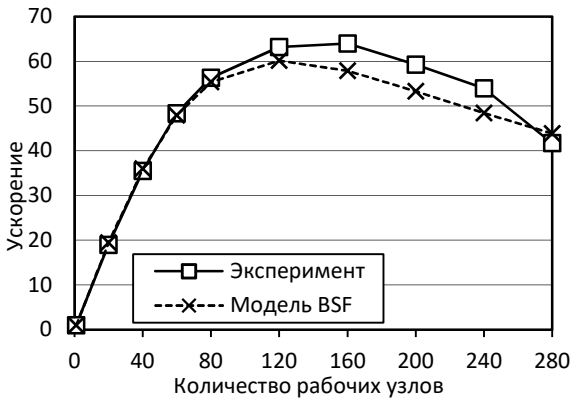


Рис. 7. Ускорение при $n = 16000$

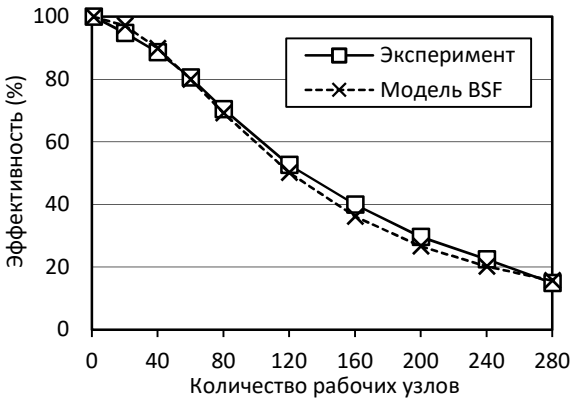


Рис. 8. Эффективность при $n = 16000$

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00352 а, Правительства РФ в соответствии с Постановлением №211 от 16.03.2013 г. (соглашение № 02.A03.21.0011) и Министерства образования и науки РФ (государственное задание 2.7905.2017/8.9).

Литература

1. Cottle R.W., Pang J.-S., Stone R.E. The Linear Complementarity Problem. Society for Industrial and Applied Mathematics, 2009. xxiv + 757 p. DOI: 10.1137/1.9780898719000.
2. Соколинская И.М., Соколинский Л.Б. О решении задачи линейного программирования в эпоху больших данных // Параллельные вычислительные технологии — XI международная конференция (ПаВТ'2017): Короткие статьи и описания плакатов (Казань, 3–7 апреля 2017 г.). Челябинск: Издательский центр ЮУрГУ, 2017. С. 471-484. URL: http://omega.sp.susu.ru/pavt2017/short/014.pdf.
3. Censor Y. et al. On Diagonally Relaxed Orthogonal Projection Methods // SIAM Journal on Scientific Computing. Society for Industrial and Applied Mathematics, 2008. Vol. 30, No. 1. P. 473–504. DOI: 10.1137/050639399.
4. Zhu J., Li X. The Block Diagonally-Relaxed Orthogonal Projection Algorithm for Compressed Sensing Based Tomography // 2011 Symposium on Photonics and Optoelectronics (SOPO). IEEE, 2011. P. 1–4. DOI: 10.1109/SOPO.2011.5780660.
5. Censor Y. Mathematical optimization for the inverse problem of intensity-modulated radiation therapy // Intensity-Modulated Radiation Therapy: The State of the Art / ed. Palta J.R., Mackie T.R. Madison, WI: Medical Physics Publishing, 2003. P. 25–49. DOI: 10.1118/1.1628279.
6. Agmon S. The relaxation method for linear inequalities // Canadian Journal of Mathematics. 1954. Vol. 6. P. 382–392. DOI: 10.4153/CJM-1954-037-2.
7. Motzkin T.S., Schoenberg I.J. The relaxation method for linear inequalities // Canadian Journal of Mathematics. 1954. Vol. 6. P. 393–404. DOI: 10.4153/CJM-1954-038-x.
8. Cimmino G. Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari // La Ricerca Scientifica, XVI, Series II, Anno IX, 1. 1938. P. 326–333.
9. Benzi M. Gianfranco Cimmino's Contributions to Numerical Mathematics // Atti del SemiSeminario di Analisi Matematica, Dipartimento di Matematica dell'Universit`a di Bologna (Ciclo di Conferenze in Ricordo di Gianfranco Cimmino, 2004). Bologna, Italy: Tecnoprint, 2005. P. 87–109.
10. Censor Y., Zenios S.A. Parallel Optimization: Theory, Algorithms, and Applications. New York: Oxford University Press, 1997.
11. Censor Y., Elfving T. New methods for linear inequalities // Linear Algebra and its Applications. North-Holland, 1982. Vol. 42. P. 199–211. DOI: 10.1016/0024-3795(82)90149-5.
12. Censor Y. Sequential and parallel projection algorithms for feasibility and optimization // Proc. SPIE 4553, Visualization and Optimization Techniques, (25 September 2001) / ed. Censor Y., Ding M. International Society for Optics and Photonics, 2001. P. 1–9. DOI: 10.1117/12.441550.

13. Kelley C.T. Iterative Methods for Linear and Nonlinear Equations. Philadelphia: Society for Industrial and Applied Mathematics, 1995. xiii + 156 p. DOI: 10.1137/1.9781611970944.
14. Bilardi G., Pietracaprina A. Models of Computation, Theoretical // Encyclopedia of Parallel Computing. Boston, MA: Springer US, 2011. P. 1150–1158. DOI: 10.1007/978-0-387-09766-4_218.
15. JaJa J.F. PRAM (Parallel Random Access Machines) // Encyclopedia of Parallel Computing. Boston, MA: Springer US, 2011. P. 1608–1615. DOI: 10.1007/978-0-387-09766-4_23.
16. Valiant L.G. A bridging model for parallel computation // Communications of the ACM. 1990. Vol. 33, No. 8. P. 103–111. DOI: 10.1145/79173.79181.
17. Culler D. et al. LogP: towards a realistic model of parallel computation // Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming, PPOPP '93. New York, New York, USA: ACM Press, 1993. P. 1–12. DOI: 10.1145/155332.155333.
18. Forsell M., Leppänen V. An extended PRAM-NUMA model of computation for TCF programming // Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2012. Washington, DC, USA: IEEE Computer Society, 2012. P. 786–793. DOI: 10.1109/IPDPSW.2012.97.
19. Gerbessiotis A. V. Extending the BSP model for multi-core and out-of-core computing: MBSP // Parallel Computing. Elsevier B.V., 2015. Vol. 41. P. 90–102. DOI: 10.1016/j.parco.2014.12.002.
20. Lu F., Song J., Pang Y. HLognGP: A parallel computation model for GPU clusters // Concurrency and Computation: Practice and Experience. 2015. Vol. 27, No. 17. P. 4880–4896. DOI: 10.1002/cpe.3475.
21. Ежова Н.А., Соколинский Л.Б. BSF: модель параллельных вычислений для многопроцессорных систем с распределенной памятью // Параллельные вычислительные технологии — XII международная конференция (ПаВТ'2018): Короткие статьи и описания плакатов (Ростов-на-Дону, 2–6 апреля 2018 г.). Челябинск: Издательский центр ЮУрГУ, 2018. С. 253–265. URL: http://omega.sp.susu.ru/pavt2018/short/001.pdf.
22. Sokolinskaya I., Sokolinsky L.B. Scalability evaluation of the NSLP algorithm for solving non-stationary linear programming problems on cluster computing systems // Суперкомпьютерные дни в России: Труды международной конференции (Москва, 25–26 сентября 2017 г.). М.: Изд-во МГУ, 2017. С. 319–332. URL: http://russianscdays.org/files/pdf17/319.pdf.
23. Cole M.I. Parallel programming with list homomorphisms // Parallel Processing Letters. 1995. Vol. 5, No. 2. P. 191–203. DOI: 10.1142/S0129626495000175.
24. Соколинская И.М., Соколинский Л.Б. Модифицированный следящий алгоритм для решения нестационарных задач линейного программирования на кластерных вычислительных системах с многоядерными ускорителями // Суперкомпьютерные дни в России: труды международной конференции (Москва, 26–27 сентября 2016 г.). М.: Изд-во МГУ, 2016. С. 294–306. URL: http://2016.russianscdays.org/files/pdf16/294.pdf.

25. Kostenetskiy P.S., Safonov A.Y. SUSU Supercomputer Resources // Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). CEUR Workshop Proceedings. 2016. Vol. 1576. P. 561–573.

Соколинская Ирина Михайловна, к.ф.-м.н., доцент, кафедра вычислительной математики и высокопроизводительных вычислений, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

Соколинский Леонид Борисович, д.ф.-м.н., профессор, проректор по информатизации, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

DOI: 10.14529/cmse190102

SCALABILITY EVALUATION OF CIMMINO ALGORITHM FOR SOLVING SYSTEMS OF LINEAR INEQUALITIES ON CLUSTER COMPUTING SYSTEMS

© 2019 I.M. Sokolinskaya, L.B. Sokolinsky

South Ural State University (pr. Lenina 76, Chelyabinsk, 454080 Russia)

E-mail: irina.sokolinskaya@susu.ru, leonid.sokolinsky@susu.ru

Received: 05.05.2018

The paper is devoted to a scalability study of Cimmino algorithm for linear inequality systems. This algorithm belongs to the class of iterative projection algorithms. For the analytical analysis of the scalability, the BSF (Bulk Synchronous Farm) parallel computation model is used. An implementation of the Cimmino algorithm in the form of operations on lists using higher-order functions Map and Reduce is presented. An analytical estimation of the scalability bound of the algorithm for cluster computing systems is derived. An information about the implementation of Cimmino algorithm on lists in C++ language using the BSF program skeleton and MPI parallel programming library are given. The results of large-scale computational experiments performed on a cluster computing system are demonstrated. A conclusion about the adequacy of the analytical estimations by comparing them with the results of computational experiments is made.

Keywords: Cimmino algorithm, system of linear inequalities, iterative algorithm, projection algorithm, parallel computation model, BSF, scalability estimation, speedup, parallel efficiency, cluster computing systems.

FOR CITATION

Sokolinskaya I.M., Sokolinsky L.B. Scalability Evaluation of Cimmino Algorithm for Solving Systems of Linear Inequalities on Cluster Computing Systems. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2019. vol. 8, no. 1. pp. 20–35. (in Russian) DOI: 10.14529/cmse190102.

References

1. Cottle R.W., Pang J.-S., Stone R.E. *The Linear Complementarity Problem*. Society for Industrial and Applied Mathematics, 2009. xxiv + 757 p. DOI: 10.1137/1.9780898719000.
2. Sokolinskaya I., Sokolinsky L.B. On the Solution of Linear Programming Problems in the Age of Big Data. *Parallel Computational Technologies. PCT 2017. Communications in Computer and Information Science*. Springer, 2017. vol. 753. pp. 86–100. DOI: 10.1007/978-3-319-67035-5_7.
3. Censor Y. et al. On Diagonally Relaxed Orthogonal Projection Methods. *SIAM Journal on Scientific Computing*. Society for Industrial and Applied Mathematics, 2008. vol. 30, no. 1. pp. 473–504. DOI: 10.1137/050639399.
4. Zhu J., Li X. The Block Diagonally-Relaxed Orthogonal Projection Algorithm for Compressed Sensing Based Tomography. *2011 Symposium on Photonics and Optoelectronics (SOPO)*. IEEE, 2011. pp. 1–4. DOI: 10.1109/SOPO.2011.5780660.

5. Censor Y. Mathematical optimization for the inverse problem of intensity-modulated radiation therapy. *Intensity-Modulated Radiation Therapy: The State of the Art*. WI: Medical Physics Publishing, 2003. pp. 25–49. DOI: 10.1118/1.1628279.
6. Agmon S. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*. 1954. vol. 6. pp. 382–392. DOI: 10.4153/CJM-1954-037-2.
7. Motzkin T.S., Schoenberg I.J. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*. 1954. vol. 6. pp. 393–404. DOI: 10.4153/CJM-1954-038-x.
8. Cimmino G. Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari. *La Ricerca Scientifica*, XVI, Series II, Anno IX, 1. 1938. pp. 326–333.
9. Benzi M. Gianfranco Cimmino's Contributions to Numerical Mathematics. *Atti del Semi-Seminario di Analisi Matematica, Dipartimento di Matematica dell'Universit`a di Bologna (Ciclo di Conferenze in Ricordo di Gianfranco Cimmino, 2004)*. Bologna, Italy: Tecnoprint, 2005. pp. 87–109.
10. Censor Y., Zenios S.A. *Parallel Optimization: Theory, Algorithms, and Applications*. New York: Oxford University Press, 1997.
11. Censor Y., Elfving T. New methods for linear inequalities. *Linear Algebra and its Applications*. North-Holland, 1982. vol. 42. pp. 199–211. DOI: 10.1016/0024-3795(82)90149-5.
12. Censor Y. Sequential and parallel projection algorithms for feasibility and optimization. *Proc. SPIE 4553, Visualization and Optimization Techniques*, (25 September 2001). Society for Optics and Photonics, 2001. pp. 1–9. DOI: 10.1117/12.441550.
13. Kelley C.T. *Iterative Methods for Linear and Nonlinear Equations*. Philadelphia: Society for Industrial and Applied Mathematics, 1995. xiii + 156 p. DOI: 10.1137/1.9781611970944.
14. Bilardi G., Pietracaprina A. Models of Computation, Theoretical. *Encyclopedia of Parallel Computing*. Boston, MA: Springer US, 2011. pp. 1150–1158. DOI: 10.1007/978-0-387-09766-4_218.
15. JaJa J.F. PRAM (Parallel Random Access Machines). *Encyclopedia of Parallel Computing*. Boston, MA: Springer US, 2011. pp. 1608–1615. DOI: 10.1007/978-0-387-09766-4_23.
16. Valiant L.G. A bridging model for parallel computation. *Communications of the ACM*. 1990. vol. 33, no. 8. pp. 103–111. DOI: 10.1145/79173.79181.
17. Culler D. et al. LogP: towards a realistic model of parallel computation. *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming, PPOPP '93*. New York, New York, USA: ACM Press, 1993. pp. 1–12. DOI: 10.1145/155332.155333.
18. Forsell M., Leppänen V. An extended PRAM-NUMA model of computation for TCF programming. *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2012*. Washington, DC, USA: IEEE Computer Society, 2012. pp. 786–793. DOI: 10.1109/IPDPSW.2012.97.
19. Gerbessiotis A. V. Extending the BSP model for multi-core and out-of-core computing: MBSP. *Parallel Computing*. Elsevier B.V., 2015. vol. 41. pp. 90–102. DOI: 10.1016/j.parco.2014.12.002.

20. Lu F., Song J., Pang Y. HLognGP: A parallel computation model for GPU clusters. *Concurrency and Computation: Practice and Experience*. 2015. vol. 27, no. 17. pp. 4880–4896. DOI: 10.1002/cpe.3475.
21. Sokolinsky L.B. Analytical Estimation of the Scalability of Iterative Numerical Algorithms on Distributed Memory Multiprocessors. *Lobachevskii Journal of Mathematics*. 2018. vol. 39, no. 4. pp. 571–575. DOI: 10.1134/S1995080218040121.
22. Sokolinskaya I., Sokolinsky L.B. Scalability Evaluation of NSLP Algorithm for Solving Non-Stationary Linear Programming Problems on Cluster Computing Systems. *Supercomputing. RuSCDays 2017. Communications in Computer and Information Science*. Cham: Springer, 2017. vol. 793. pp. 40–53. DOI: 10.1007/978-3-319-71255-0_4.
23. Cole M.I. Parallel programming with list homomorphisms. *Parallel Processing Letters*. 1995. vol. 5, no. 2. pp. 191–203. DOI: 10.1142/S0129626495000175.
24. Sokolinskaya I., Sokolinsky L. Revised Pursuit Algorithm for Solving Non-stationary Linear Programming Problems on Modern Computing Clusters with Manycore Accelerators. *Supercomputing. RuSCDays 2016. Communications in Computer and Information Science*. Cham: Springer, 2016. vol. 687. pp. 212–223. DOI: 10.1007/978-3-319-55669-7_17.
25. Kostenetskiy P.S., Safonov A.Y. SUSU Supercomputer Resources. *Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016)*. CEUR Workshop Proceedings. 2016. vol. 1576. pp. 561–573.

УДК 512.5, 519.1(075.8) **DOI: 10.14529/cmse190103**

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ ПОСТРОЕНИЯ A-ЦЕПЕЙ С УПОРЯДОЧЕННЫМ ОХВАТЫВАНИЕМ В ПЛОСКОМ СВЯЗНОМ 4-РЕГУЛЯРНОМ ГРАФЕ

© 2019 Т.А. Макаровских

Южно-Уральский государственный университет
(454080 Челябинск, пр. им. В.И. Ленина, д. 76)
E-mail: Makarovskikh.T.A@susu.ru

Поступила в редакцию: 24.07.2018

В CAD/CAM-системах технологической подготовки процессов раскроя встает задача построения маршрута движения режущего инструмента, при котором отрезанная от листа часть не требует дополнительных разрезаний и запрещены пересечения траектории резки (касания допускаются). Формально такая задача может быть сформулирована как задача построения самонепересекающейся цепи в плоском эйлеровом графе, представляющим гомеоморфный образ раскройного плана. В конечном счете задачи построения маршрутов, удовлетворяющих технологическим ограничениям, сводятся к нахождению A-цепи с упорядоченным охватыванием в плоском связном 4-регулярном графе. В статье предложен алгоритм нахождения такой цепи. Выполнение алгоритма состоит из двух этапов. На первом этапе выявляются и расщепляются точки сочленения ранга k. На втором этапе построение цепи начинается из произвольной вершины, инцидентной внешней грани; первым ребром цепи выбирается инцидентное данной вершине ребро максимального ранга; далее организуется итерационный процесс, где в качестве следующего ребра выбирается непройденное ребро максимального ранга, являющееся левым либо правым соседом текущего ребра. Показано, что для плоского связного 4-регулярного графа алгоритм строит маршрут с указанными свойствами за линейное время. Представленные алгоритмы реализованы в виде компьютерной программы. Приведены примеры решения ряда тестовых задач.

Ключевые слова: плоский граф, маршрут, раскройный план, полиномиальный алгоритм, CAD/CAM.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Макаровских Т.А. Программное обеспечение для построения A-цепей с упорядоченным охватыванием в плоском связном 4-регулярном графе // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2019. Т. 8, № 1. С. 36–53. DOI: 10.14529/cmse190103.

Введение

Лазерная резка является одной из основных современных технологий, используемых при обработке листового материала, что делает актуальной задачу определения траектории движения режущего инструмента. Задача определения траектории заключается в определении точной последовательности резов. Развитие автоматизации производства привело к появлению технологического оборудования с числовым программным управлением, используемого для резки листовых материалов. Новые технологии позволяют осуществлять вырезание по произвольной траектории с достаточной для практики точностью. Преимуществом при использовании лазерной резки является минимальность таких показателей как ширина реза и термические деформации. Целью задачи определения маршрута резки является поиск такого пути режущего инструмента, при котором выполняются условия предшествования, а время, затраченное на вырезание, минимально [1].

В терминах задачи лазерной резки под условием предшествования понимается требование к тому, чтобы отрезанная от листа часть не требовала дополнительных разрезаний (т.е. все элементы вложенного контура должны быть вырезаны прежде, чем внешний контур окажется полностью вырезанным).

Общее время, требуемое на выполнение резки, можно представить как суммарное время для осуществления всех вырезаний, времени на выполнение холостых переходов, времени на врезку и пр. Так как основным условием предшествования при выполнении лазерной резки является требование к отсутствию разрезаний для уже отрезанной части листа, то

- во-первых, все элементы внутренних контуров должны быть вырезаны прежде, чем будут полностью отрезаны от листа внешние контуры;
- во-вторых, необходимо учитывать термальные эффекты, например, стоит избегать пересечения имеющихся резов.

При технологической подготовке процессов раскроя с применением лазерной резки (для CAD/CAM систем) возникает задача нахождения маршрута для режущего инструмента, при котором отрезанная от листа часть не требует разрезаний, и не допускаются пересечения в траектории резки (касания допускаются) [1, 2]. В процессе лазерной резки происходит нагревание металлического листа. В связи с этим при определении траектории движения режущего инструмента необходимо учитывать как термические эффекты, так и возможные деформации. Это приводит к необходимости избегать пересечений резов.

В работах [1, 3] приводится классификация задач маршрутизации режущего инструмента и отмечается, что технологии ECP (Endpoint Cutting Problem) и ICP (Intermittent Cutting Problem) за счет возможности совмещения границ вырезаемых деталей позволяют сократить расход материала, длину резки и длину холостых проходов [3]. Проблемы уменьшения отходов материала и максимального совмещения фрагментов контуров вырезаемых деталей решаются на этапе составления раскройного плана.

Несмотря на очевидные преимущества технологий ECP и ICP, в настоящее время большинство отечественных [6–10] и зарубежных [1, 3–5] работ посвящено развитию технологии GTSP (General Travelling Salesman Problem), которая не предполагает совмещение контуров вырезаемых деталей. Таким образом, при использовани данной технологии длина траектории будет равна сумме периметров всех контуров, а количество точек врезки — количеству контуров. Однако при этом проблема выполнения отмеченных выше условий предшествования оказывается тривиальной.

Построению эффективных алгоритмов для раскройных планов, в которых допускаются совмещения границ контуров, посвящен ряд работ [11–13]. Алгоритм из работы [11] находит траекторию движения режущего инструмента и минимизирует число точек врезки. Предложенный в работе алгоритм применим только для раскройных планов, допускающих плоскую достройку до эйлерова графа. В статье [12] рассмотрена та же задача, что и в [11], задача формализована достаточно подробно, введено понятие маршрута с ограничениями в плоском графе (OE-маршрутов) и приведен не только алгоритм решения задачи для эйлерова графа, но и решена задача китайского почтальона (таким образом, приведен алгоритм решения задачи для плоского связного графа). Один из подходов к построению покрытия упорядоченной последовательностью реберно-непересекающейся последовательностью цепей рассмотрен и в работе [13]. Он требует решения задачи сельских почтальонов. Напомним, что в маршруте почтальона отсутствует требование, чтобы маршрут являлся реберно-непересекающимся.

Все отмеченные выше работы [11–13] не исключают построение траектории с самопересечениями.

О построении маршрута, не допускающего пересечения траекторий, впервые упоминается в работе [14]. В данной работе речь идет о самонепересекающейся цепи, тем не менее, в данной статье понятие самонепересекающейся цепи ошибочно сужено до понятия A-цепи [15]. В действительности, класс самонепересекающихся цепей шире класса A-цепей [17]. Под самонепересекающейся цепью понимают эйлеров цикл C в плоском графе G, гомеоморфный плоскому циклическому графу $\tilde{G}$ (представляющему жорданову кривую на плоскости), который может быть получен из графа G с помощью применения $O(|E(G)|)$ операций расщепления вершин. В работах [17, 18] показано, что задача построения самонепересекающейся цепи в графе G сводится к задаче нахождения A-цепи с упорядоченным охватыванием для плоского связного 4-регулярного графа $\tilde{G}$, полученного в результате введения в гомеоморфный образ раскройного плана фиктивных вершин и ребер. После стягивания всех фиктивных вершин и ребер в графе $\tilde{G}$ будет получена самонепересекающаяся цепь в графе G.

Таким образом, разработка эффективных алгоритмов для построения A-цепи с упорядоченным охватыванием для плоского связного 4-регулярного графа является актуальной задачей.

Статья организована следующим образом. В разделе 1 приводится представление гомеоморфного образа раскройного плана. В разделе 2 введены необходимые определения и предварительные результаты, используемые в дальнейших рассуждениях. Приведен полиномиальный алгоритм построения AOE-цепи в плоском связном 4-регулярном графе. Доказана его результативность. В разделе 3 приведены примеры тестовых задач, решенных с помощью разработанного алгоритма. В заключении перечислены полученные в работе результаты, отмечены направления дальнейших исследований.

1. Абстрактное представление раскройного плана в виде плоского графа

Для определения последовательности резки фрагментов раскройного плана не используется информация о форме детали, поэтому все кривые без самопересечений и соприкосновений на плоскости, представляющие форму деталей, интерпретируются в виде ребер плоского графа [19], а все точки пересечений и соприкосновений представляются в виде вершин графа. Для изложенных в работе алгоритмов необходимо представление графа в виде системы инцидентности. Чтобы система инцидентности с точностью до гомеоморфизма была эквивалентна раскройному плану, необходимо и достаточно в каждой вершине указать циклический порядок[1] на множестве инцидентных этой вершине ребер.

Рассмотрим подробнее представление плоского графа в виде его гомеоморфного образа, т.е. в виде системы инцидентности. Гомеоморфным образом раскройного плана является плоский граф G с внешней гранью f_0 на плоскости S. Для любой части J графа G (т.е. $J \subseteq G$) обозначим через $\mathsf{Int}\,(J)$ теоретико-множественное объединение его внутренних граней (объединение всех связных компонент $S \setminus J$, не содержащих внешней грани). Тогда $\mathrm{Int}(J)$ можно интерпретировать как отрезанную от листа часть. Множества вершин, ребер и граней графа J будем обозначать через $V(J)$, $E(J)$ и $F(J)$ соответственно.

[1]Циклическим порядком называется способ расположения объектов на окружности.

Топологическое представление плоского графа G на плоскости S с точностью до гомеоморфизма определяется заданием для каждого ребра $e \in E(G)$ следующих функций [2, 20]:

- $v_k(e)$, $k = 1, 2$ — вершины, инцидентные ребру e;
- $l_k(e)$, $k = 1, 2$ — ребра, полученные вращением ребра e против часовой стрелки вокруг вершины $v_k(e)$;
- $r_k(e)$, $k = 1, 2$ — ребра, полученные вращением ребра e по часовой стрелке вокруг вершины $v_k(e)$;
- $f_k(e)$ — грань, находящаяся справа при движении по ребру e от вершины $v_k(e)$ к вершине $v_{3-k}(e)$, $k = 1, 2$.

Иллюстрация введенных функций дана на рис. 1.

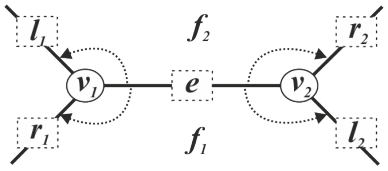


Рис. 1. Функции для представления плоского графа

Таким образом, используя известные координаты прообразов вершин графа G и размещения фрагментов раскройного плана, являющихся прообразами ребер графа G, любой маршрут в графе G можно интерпретировать как траекторию режущего инструмента.

Представление графа фактически задает ориентацию его ребер. Далее предполагается, что движение по ребру для определенности осуществляется от вершины $v_1(e)$ к вершине $v_2(e)$. Поскольку при задании графа G неизвестно, какое из ребер в каком направлении будет пройдено, то при выполнении алгоритма производится перестановка значений полей $v_k(e)$, $r_k(e)$ и $l_k(e)$, $f_k(e)$, $k = 1, 2$ некоторых ребер. В алгоритме данную процедуру выполняет функция `REPLACE`, функциональным назначением которой является перестановка индексов функций $v_k(e)$, $l_k(e)$ и $r_k(e)$ на $3 - k$, $k = 1, 2$ [20].

Далее будем считать, что все рассматриваемые плоские графы представлены указанными функциями. Пространственная сложность такого представления будет $O(|E(G)| \cdot \log_2 |V(G)|)$.

2. Построение AOE-цепи в плоском графе

2.1. Определения и обозначения

В дальнейшем будем использовать ряд понятий, определения которых имеются в работах [15, 21, 22]. Приведем их для полноты изложения.

Определение 1. *Графом разрешенных переходов* $T_G(v)$ вершины $v \in V(G)$ [21] будем называть граф, вершинами которого являются ребра, инцидентные вершине v, т.е. $V(T_G(v)) = E_G(v)$, а множество ребер — разрешенные переходы между ребрами.

Определение 2. *Системой разрешенных переходов* T_G [21] будем называть множество $\{T_G(v) \mid v \in V(G)\}$, где $T_G(v)$ — граф переходов в вершине v.

Определение 3. Путь $P = v_0, e_1, v_1, ..., e_k, v_k$ в графе G является T_G-*совместимым* [21], если $e_i,\ e_{i+1} \in E(T_G(v_i))$ для каждого i $(1 \le i \le k-1)$.

Определение 4. Пусть для цепи $T = v_0, k_1, v_1, \ldots, k_n, v_n,\ v_n = v_0$ в графе $G = (V, E)$ в каждой вершине $v \in V$ задан циклический порядок $O^{\pm}(v)$, определяющий систему переходов $A_G(v) \subset O^{\pm}(v)$. В случае, когда $\forall v \in V(G)\ A_G(v) = O^{\pm}(v)$, систему переходов $A_G(v)$ будем называть *полной системой переходов.*

В частности, маршруты, в которых цикл из пройденных ребер не охватывает еще непройденных (т.е. заданы условия предшествования: отрезанная от листа часть не требует дополнительных разрезаний) представляют класс OE-маршрутов.

Определение 5. Будем говорить, что цикл $C = v_1e_1v_2e_2\ldots v_k$ в плоском эйлеровом графе G имеет *упорядоченное охватывание* (называется OE-цепью), если для любой его начальной части $C_i = v_1e_1v_2e_2\ldots e_i$, $i \le |E(G)|$ выполнено условие $\mathrm{Int}\,(C_i) \cap G = \emptyset$.

Определение 6. Упорядоченная последовательность реберно-непересекающихся OE-цепей

$$\begin{aligned} C^0 &= v^0e_1^0v_1^0e_2^0...e_{k_0}^0v_{k_0}^0, \quad C^1 = v^1e_1^1v_1^1e_2^1...e_{k_1}^1v_{k_1}^1, \ldots, \\ C^{n-1} &= v^{n-1}e_1^{n-1}v_1^{n-1}e_2^{n-1}...e_{k_{n-1}}^{n-1}v_{k_{n-1}}^{n-1}, \end{aligned}$$

покрывающая граф G и такая, что

$$(\forall m :\ m < n), \quad \left(\bigcup\nolimits_{l=0}^{m-1} \mathrm{Int}(C^l)\right) \cap \left(\bigcup\nolimits_{l=m}^{n-1} C^l\right) = \emptyset$$

называется *маршрутом с упорядоченным охватыванием* (OE-маршрутом).

Определение 7. Эйлерову цепь T будем называть A-*цепью* [15], если она является A_G-совместимой цепью. Таким образом, последовательные ребра в цепи T (инцидентные вершине v) являются соседями в циклическом порядке $O^{\pm}(v)$.

Определение 8. Будем говорить, что цепь является AOE-*цепью*, если она одновременно является OE-цепью и A-цепью.

Определение 9. *Рангом ребра* $e \in E(G)$ будем называть значение функции $\mathrm{rank}(e) : E(G) \to \mathbb{N}$, определяемое рекурсивно:

- пусть $E_1 = \{e \in E :\ e \subset f_0\}$ — множество ребер, ограничивающих внешнюю грань f_0 графа $G(V, E)$, тогда $(\forall e \in E_1)\,(\mathrm{rank}(e) = 1)$;
- пусть $E_k(G)$ — множество ребер ранга 1 графа

$$G_k\left(V, E\backslash\left(\bigcup_{l=1}^{k-1} E_l\right)\right),$$

тогда $(\forall e \in E_k)\,(\mathrm{rank}(e) = k)$.

Ранг ребра определяет его удалённость от внешней грани и показывает, какое минимальное число граней необходимо пересечь, чтобы добраться от внешней грани f_0 до этого ребра. Это позволяет для определения ранга использовать граф G', топологически двойственный исходному графу G: множеством вершин графа G' является множество $F(G)$,

а ребрам графа G' соответствует наличие между двумя гранями границы ненулевой длины, т.е. ребра $e \in E(G)$.

Для всех $f \in F(G)$ расстояние в графе G' между f и внешней гранью f_0 можно определить, построив в графе G' дерево $T^{f_0}_{G'}$ кратчайших путей до вершины $f_0 \in F$. Наличие в представлении графа G функций $l_k : E(G) \to E(G)$, $k = 1, 2$ позволяет найти функции ранга за время, не превосходящее величины $O(|E(G)| \cdot \log_2 |V(G)|)$.

Различные способы вычисления значений функции $\mathrm{rank}(e)$ приведены в работах [22, 23].

Теорема 1. Если в плоском графе G графе существует A-цепь, то существует и AOE-цепь.

Доказательство. A-цепь в плоском эйлеровом графе представляет замкнутую жорданову кривую без самопересечений. Данную кривую можно получить следующим образом: рассмотрим граф G, в каждой вершине которого задана система переходов, соответствующая A-цепи, и граф G', полученный из графа G расщеплением вершин в соответствии с системой разрешенных переходов (рис. 2). Построенный таким образом граф G' представляет собой жорданову кривую без пересечений и в соответствии с теоремой Жордана [16] разбивает плоскость на внешнюю и внутреннюю компоненты связности.

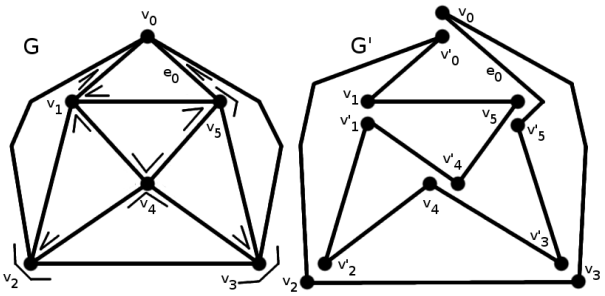


Рис. 2. Плоский граф G с заданной системой переходов A-цепи и соответствующий ему граф G', являющийся жордановой кривой на плоскости

Очевидно, что на полученной кривой найдется вершина v_0, инцидентная внешней грани графа G и ребру e_n: $\mathrm{rank}(e_n) = 1$. Если принять вершину v_0 за начало AOE-цепи, а направление обхода выбрать таким образом, чтобы ребро (e_n) было концом цепи C_n, то в силу отсутствия пересечений для внутренности любой начальной части цепи $C_i = v_0 e_1 v_1 e_2 \dots e_i$, $i \le |E(G)|$ будет выполнено условие

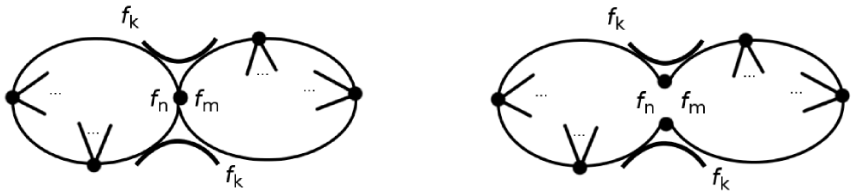
$$\mathrm{Int}\,(C_i) \cap G = \emptyset,$$

т.е. такая A-цепь C_n будет являться и OE-цепью. Действительно, предположение существования ребра $e \in \mathrm{Int}\,(C_i)$ сразу приводит к противоречию с тем, что в системе переходов цепи отсутствуют пересечения. *Теорема доказана.*

Теорема 2. В плоском связном 4-регулярном графе G существует AOE-цепь.

Доказательство. Доказательство существования A-цепи в связном 4-регулярном графе приведено в [15, Следствие VI.6]. Так как в этом графе существует A-цепь, то в силу теоремы 1 в нем существует и AOE-цепь. *Теорема доказана.*

Определение 10. Суграф G_k графа G, для которого $E(G_k) = \{e \in E(G) : \mathrm{rank(e)} \ge \mathrm{k}\}$ назовем *сугрaфом ранга* k.



а) Верная система переходов для расщепления точки сочленения суграфа G_k

б) Расщепление в соответствии с системой переходов в точке сочленения суграфа G_k

Рис. 3. Расщепление точек сочленения ранга k

Далее будет рассмотрен алгоритм `AOE-TRAIL`, позволяющий найти AOE-цепь в плоском связном 4-регулярном графе, в котором любой суграф G ранга k не имеет точек сочленения.

При поиске точек сочленения суграфа G_k используются следующие свойства 4-регулярных графов.

Предложение 1. Вершина, инцидентная четырем ребрам, смежным внешней грани, является точкой сочленения.

Предложение 2. Внешняя грань суграфа G_k является объединением всех граней ранга k в графе G.

Справедливость этих предложений очевидна в силу 4-регулярности графа.

Определение 11. *Рангом грани* $f \in F(G)$ будем называть значение функции rank : $F(G) \to \mathrm{Z}^{\geq 0}$:

$$\mathrm{rank}(f) = \begin{cases} 0, & \text{при } f = f_0, \\ \min_{e \in E(f)} \mathrm{rank}(e), & \text{в противном случае,} \end{cases}$$

где $E(f)$ — множество ребер инцидентных грани $f \in F$.

Определение 12. *Точкой сочленения ранга* k назовем вершину, инцидентную четырем ребрам, смежным внешней грани суграфа G_k.

2.2. Расщепление точек сочленения ранга k

Для корректного построения AOE-цепи в плоском связном 4-регулярном графе необходимо предварительно «правильно» расщепить все точки сочленения суграфов G_k. В результате такого расщепления получим граф, у которого любой суграф G_k не содержит точек сочленения. Последовательность расщепления вершин не имеет значения, т.к. это локальная операция.

Под «правильным» будем понимать переход между дугами, соответствующими циклическому порядку и инцидентными различным парам граней (см. рис. 3.а)).

Результат расщепления приведен на рис. 3.б).

Рассмотрим алг. 1, используемый для выполнения операции расщепления точек сочленения всех рангов в плоском связном 4-регулярном графе.

Алгоритм 1 `CUT-POINT-SPLITTING`

Require: плоский связный 4-регулярный граф $G = (V, E)$, представленный для всех $e \in E(G)$ функциями v_s, l_s, r_s , $s = 1, 2$.

Ensure: гомеоморфный образ графа $G = (V, E)$, представленный для всех $e \in E(G)$ функциями v_s, l_s, r_s , $s = 1, 2$, в котором любой суграф G_k не имеет точек сочленения.

Промежуточные данные: $\forall v \in V(G)$: point(v) — массив указателей на одно из ребер, инцидентных вершине v, rank(v) — массив рангов вершин, count(v) — счетчик инцидентных ребер одного ранга для каждой вершины;
$\forall f \in F(G)$: массив rank(f).

```
1: Initiate():
2: for all v ∈ V(G) do                    ▷ Обнулить счетчик инцидентных вершине ребер одного ранга
3:     point(v) := 0; count(v) := 0
4: end for
5: Ranking(G)                             ▷ Определение ранга всех вершин, ребер и граней графа
6: Finding():                             ▷ Нахождение точек сочленения
7: for all e ∈ E(G) do
8:     point(v_1(e)) := e
9:     point(v_2(e)) := e
10: end for
11: for all v ∈ V(G) do                   ▷ Просмотреть последовательно все вершины
12:     e := point(v)
13:     k := rank(v)                      ▷ Сохранить значение ранга k инцидентного ребра e
14:     if v = v_1(e) then                ▷ Определить ориентацию ребра e
15:         s := 1
16:     else
17:         s := 2
18:     end if
19:     e := l_s(e)                       ▷ Подсчет количества инцидентных вершине v ребер ранга k
20:     for i = 1 up to 4 do
21:         if rank(e) = k then
22:             count(v) := count(v) + 1
23:             if i < 4 then
24:                 e := l_s(e)
25:             end if
26:         end if
27:     end for
28:     if count(v) = 4 then              ▷ Если найдена точка сочленения, расщепить вершину
29:         if (f_s(e) = f_s(l_s(e)) and f_{3−s}(e) = f_{3−s}(l_s(e))) or (f_s(e) = f_{3−s}(l_s(e)) and f_{3−s}(e) =
    f_s(l_s(e))) then
30:             e* := l_s(e), l_s(e) := r_s(e), r_s(r_s(e)) := e,
31:             r_s(e*) := l_s(e*), l_s(l_s(e*)) := e*
32:         else
33:             e* := r_s(e), r_s(e) := l_s(e), l_s(l_s(e) := e,
34:             l_s(e*) := r_s(e*), r_s(r_s(e*)) := e*
35:         end if
36:     end if
37: end for
```

Справедлива следующая теорема.

Теорема 3. Алгоритм `CUT-POINT-SPLITTING` определяет точки сочленения всех рангов плоского связного 4-регулярного графа и расщепляет их без нарушения связности за время не превосходящее $O(|E|)$.

Доказательство. На этапе инициализации (функция `Initiate()`, строки 1–4) для каждой вершины $v \in V(G)$ обнуляется счетчик $count(v)$ инцидентных ей ребер, ранг которых совпадает с рангом v. Затем вычисляются ранги всех вершин, ребер и граней графа (функция `Ranking()`, строка 5).

Далее осуществляется поиск и расщепление точек сочленения каждого ранга (строки 6–37). В первом цикле (строки 7–10) осуществляется инициализация массива $point(v)$: для каждой вершины определяется инцидентное ей ребро. В следующем цикле (строки 11–37) определяется ранг k текущей вершины (строка 13), уточняется ориентация текущего ребра (строки 14–18). В цикле (строки 19–27) подсчитывается количество инцидентных вершине v ребер, ранг которых совпадает с рангом k вершины. Если текущая вершина инцидентна четырем ребрам ранга k, то в соответствии с предложением 1, то она является точкой сочленения ранга k. В этом случае выполняется расщепление точки сочленения ранга k за счет соответствующей модификации указателей на смежные ребра (см. строки 28–37 алг. 1) в соответствии с рис. 3.

Итак, алг. 1 за время не превосходящее $O(|E|)$ осуществляет последовательный просмотр всех вершин графа G и в случае обнаружения точки сочленения ранга k выполняет расщепление найденной вершины на две вершины степени 2 без нарушения связности графа. Повторный просмотр вершин не требуется: алгоритм не изменяет ранги ребер, а только модифицирует их смежность. В результате каждой операции расщепления будет получено две вершины степени 2. Следовательно, в результате единственного последовательного просмотра всех вершин графа для каждого ранга k будут расщеплены все имеющиеся точки сочленения. Появление новых точек сочленения ранга k невозможно, т.к. значение функции rank(e) не изменяется. *Теорема доказана.*

2.3. Алгоритм построения AOE-цепи в плоском связном 4-регулярном графе без точек сочленения ранга k

Рассмотрим алг. 2, выполняющий построение AOE-цепи в плоском связном 4-регулярном графе без точек сочленения.

Алгоритм 2 AOE-`TRAIL`

Require: плоский связный 4-регулярный граф G без точек сочленения, представленный функциями v_k, l_k, r_k (рис. 1), $k = 1, 2$;

1: начальная вершина $v \in V(f_0)$.

Ensure: $ATrail$ — выходной поток, содержащий построенную алгоритмом AOE-цепь.

2: Initiate(G, v_0)

3: Ranking(G) ▷ Ранжирование

4: Constructing() ▷ Построение

Процедура инициализации (`Initiate()`) заключается в инциализации значением 0 счетчика `counter` количества ребер, включенных в результирующую цепь, а также в присваивании всем ребрам пометки `true`, соответствующей непройденному ребру.

Функциональное назначение процедуры `Ranking()` — определить для каждого ребра $e \in E(G)$ графа его ранг $\mathrm{rank}(e)$. Как отмечалось ранее, различные алгоритмы вычисления значений функции $\mathrm{rank}(e)$ приведены в работах [22, 23], их вычислительная сложность не превосходит величины $O(|E(G)| \cdot \log_2 |V(G)|)$.

Описание процедуры `Constructing()`, выполняющей построение AOE-цепи представлено ниже (см. алг. 3).

Алгоритм 3 `Constructing (v)`

1: $e = \arg\max_{e \in E(v)} \mathrm{rank}(e)$ ▷ Выбрать ребро максимального ранга, инцидентное вершине v
2: **repeat**
3: **if** $v \neq v_1(e)$ **then**
4: REPLACE(e)
5: **end if** ▷ При необходимости скорректировать нумерацию функций для ребра e
6: **Print**(v, e) ▷ Вывести ребро e в результирующую последовательность
7: mark(e) := **false**; counter:=counter+1; $v := v_2(e)$ ▷ Пометить ребро как пройденное
8: **if** ($\mathrm{rank}(r_2(e)) \geq \mathrm{rank}(l_2(e))$) **then** ▷ Выбрать следующим ребро максимального ранга
9: **if** mark($r_2(e)$) **then** ▷ Проверить, пройдено ли добавляемое в цепь ребро
10: $e := r_2(e)$ ▷ Для пройденных ребер значение в массиве $mark$ — ложь
11: **else**
12: $e := l_2(e)$
13: **end if**
14: **else**
15: **if** (mark($l_2(e)$)) **then** ▷ Выбрать непройденное ребро
16: $e := l_2(e)$
17: **else**
18: $e := r_2(e)$
19: **end if**
20: **end if**
21: **until** (counter $> |E(G)|$) ▷ Завершить цикл, когда просмотрены все ребра

В строке 1 предписано для заданной инцидентной внешней грани вершины v выбрать инцидентное ей ребро e максимального ранга.

В следующем далее цикле **repeat** — **until** (строки 2–21) осуществляется построение AOE-цепи:

- при необходимости корректируется нумерация функций для ребра e (строки 3–5), т.е. производится взаимообмен номеров инцидентных ребру e вершин, таким образом, чтобы вершина $v_1(e)$ посещалась при обходе раньше вершины $v_2(e)$;
- текущее ребро выводится в результирующую последовательность (строка 6);
- помечается как пройденное (mark(e)=**false**), в качестве текущей вершины v выбирается $v_2(e)$, увеличивается значение счётчика пройденных ребер (строка 7);
- в качестве следующего текущего ребра e выбирается инцидентное вершине v непройденное ребро (mark(e)=**true**), являющееся по возможности левым или правым соседом предыдущего ребра.

Теорема 4. Алгоритм `AOE-TRAIL` и процедура `Constructing(v)` строят AOE-цепь в плоском связном 4-регулярном графе G за время $O(|E(G)| \cdot \log_2 |V(G)|)$.

Доказательство. Результативность процедуры `Initiate()` очевидна. Результативность процедуры `Ranking()` доказана в [24]. Их совокупная вычислительная сложность не превосходит $O(|E(G)| \cdot \log_2 |V(G)|)$.

Основной цикл процедуры `Constructing()` состоит в выборе последующего непройденного ребра максимального ранга, смежного ребру e. Процедура выполняется до тех пор, пока все ребра не будут включены в результирующую цепь (их пометка будет изменена на `false`).

Докажем результативность процедуры `Constructing()`. Если для текущего ребра e вершина $v = v_2(e)$ посещается впервые, то выполнение тела цикла можно интерпретировать как расщепление вершины $v^* = v_2(e)$ в соответствии с системой переходов A-цепи. После расщепления гомеоморфный образ полученного графа уже не будет содержать вершины v^*. При повторном попадании алгоритма в вершину $v^* \in V(G)$ алгоритм продолжает формирование цепи в соответствии с гомеоморфным образом полученного ранее ребра.

Гомеоморфный образ полученного графа после выполнения тела цикла является плоским связным графом, так как ни один суграф ранга k не содержит точек сочленения.

После выполнения $|V(G)|$ расщеплений в соответствии с системой переходов A-цепи получим гомеоморфный образ графа, являющийся окружностью. В этом случае поток $ATrail$ будет содержать полученную AOE-цепь.

При выполнении расщеплений гомеоморфный образ остается связным графом, поэтому все ребра будут обработаны алгоритмом (получат пометку `false`). Процедура `Constructing()` имеет единственный цикл. Вычислительная сложность тела цикла не превосходит величины $O(\log_2 |V(G)|)$, а число итераций этого цикла равно $|E(G)|$. Следовательно, вычислительная сложность алгоритма не превосходит величины $O(|E(G)| \cdot \log_2 |V(G)|)$. *Теорема доказана.*

3. Примеры работы алгоритма

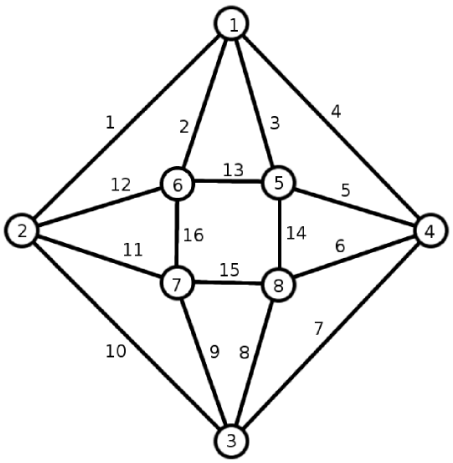
Представленные выше алгоритмы реализованы в виде компьютерной программы. Приведем примеры тестовых задач, решенных с помощью этой программы.

Во входном файле указывается общее число ребер, каждая последующая строка файла соответствует одному ребру. В ней приведены значения указанных в разделе 1 функций в следующем порядке: инцидентные вершины (функции $v_k(e)$, $k = 1, 2$); левые (функции $l_k(e)$, $k = 1, 2$) и правые (функции $r_k(e)$, $k = 1, 2$) соседние ребра; грани $f_k(e)$, $k = 1, 2$, инцидентные ребрам $l_k(e)$; последним числом в строке указывается ранг $\mathrm{rank}(e)$ соответствующего ребра. Плоский связный 4-регулярный граф с примером его кодирования приведены на рис. 4.

Разработанная программа в зависимости от номера начальной вершины (задаваемого пользователем) определяет AOE-цепь в графе, либо выводит сообщение, что задача не имеет решения.

Для тестирования разработанной программы достаточно рассмотреть следующие возможные случаи:

- граф, не имеющий точек сочленения (в этом случае не осуществляется вызов функции `CutPointSplitting`, пример такого графа представлен на рис. 4); найденные программой последовательности ребер, соответствующие AOE-цепям в указанном графе, приведены в табл. 1;




```
16
1 2 4 12 2 10 1 1
1 6 1 13 3 12 0 2
1 5 2 5 4 13 0 2
1 4 3 7 1 5 1 1
4 5 4 14 6 3 0 2
4 8 5 8 7 14 0 2
4 3 6 10 4 8 1 1
3 8 7 15 9 6 0 2
3 7 8 11 10 15 0 2
3 2 9 1 7 11 1 1
2 7 10 16 12 9 0 2
2 6 11 2 1 16 0 2
6 5 16 3 2 14 0 3
5 8 13 6 5 15 0 3
8 7 14 9 8 16 0 3
7 6 15 12 11 13 0 3
```

Рис. 4. Пример плоского связного 4-регулярного графа с помеченными вершинами и ребрами и данные входного файла для этого графа

Таблица 1

Найденные программой *AOE*-цепи для графа, приведенного на рис. 4

Вершина	*AOE*-цепь (ребра)
1	2–13–14–15–16–12–11–9–8–7–6–5–3–4–7–10–1
2	11–16–13–14–15–9–8–6–5–3–2–12–1–4–7–10
3	8–15–16–13–14–6–5–3–2–12–11–9–10–1–4–7
4	5–14–15–16–13–3–2–12–11–9–8–6–7–10–1–4

- граф, имеющий одну точку сочленения (см. рис. 5а); найденные программой последовательности ребер, соответствующие *AOE*-цепям в указанном графе, приведены в табл. 2;

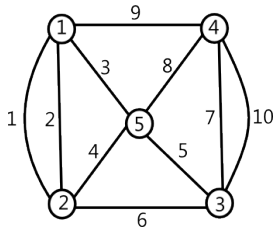
Таблица 2

Найденные программой *AOE*-цепи для графа, приведенного на рис. 5а

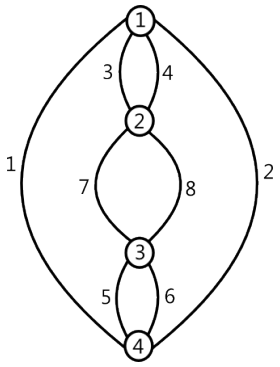
Вершина	*AOE*-цепь (ребра)
1	2–4–3–9–8–5–7–10–6–1
2	2–3–4–6–5–8–7–10–9–1
3	5–8–7–10–9–3–4–2–1–6
4	7–5–8–9–3–4–2–1–6–10

- граф, имеющий более одной точки сочленения, но все точки сочленения имеют один и тот же ранг (см. рис. 5б); найденные программой последовательности ребер, соответствующие *AOE*-цепям в указанном графе, приведены в табл. 3;
- граф, имеющий точки сочленения разных рангов (см. рис. 6); найденные программой последовательности ребер, соответствующие *AOE*-цепям в указанном графе, приведены в табл. 4.

Разработанное программное обеспечение в рассмотренных случаях корректно находит решение задачи для корректно заданных исходных данных. Так как других случаев



а) Пример плоского связного 4-регулярного графа с помеченными вершинами и ребрами, имеющего одну точку сочленения ранга 2 в вершине 5



б) Пример плоского связного 4-регулярного графа с помеченными вершинами и ребрами, имеющего две точки сочленения ранга 2 в вершинах 2 и 3

Рис. 5. Примеры графов, имеющих точки сочленения

Таблица 3

Найденные программой *AOE*-цепи для графа, приведенного на рис. 5б

Вершина	*AOE*-цепь (ребра)
1	3–4–2–6–8–7–5–1
4	5–7–8–6–2–4–3–1

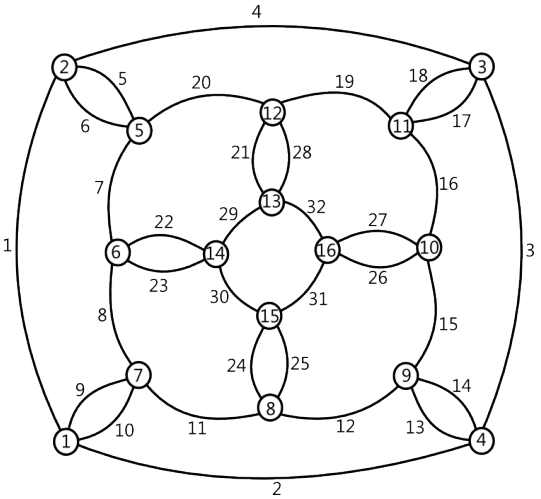


Рис. 6. Пример плоского связного 4-регулярного графа с помеченными вершинами и ребрами, имеющего точки сочленения ранга 2 в вершинах 5, 7, 9, 11 и ранга 3 в вершинах 13, 14, 15 и 16

Таблица 4

Найденные программой *AOE*-цепи для графа, приведенного на рис. 6

Вершина	*AOE*-цепь (ребра)
1	9–8–23–30–24–25–31–26–27–32–29–22–7–20–21–28–19–18–17–16–15–14–13–12–11 10–2–3–4–5–6–1
2	5–6–1–9–8–23–30–24–25–31–26–27–32–29–22–7–20–21–28–19–18–17–16–15–14–13–12–11 10–2–3–4
3	17–16–27–32–29–22–23–30–24–25–31–26–15–14–13–12–11–10–9–8–7–20–21–28–19–18–4–5–6–1–2–3
4	13–12–25–31–26–27–32–29–22–23–30–24–11–10–9–8–7–20–21–28–19–18–17–16–15–14–3–4–5–6–1–2

нахождения точек сочленения не существует, можно заключить, что для корректно заданных исходных данных программа корректно решает поставленную задачу поиска *AOE*-цепи в плоском связном 4-регулярном графе.

Заключение

Таким образом, предложенный в статье подход позволяет решить проблему маршрутизации при вырезании деталей, когда на маршрут движения режущего инструмента одновременно наложены такие технологические ограничения, как (1) отрезанная от листа часть не требует дополнительных разрезаний, (2) осуществляется переход по смежному контуру.

В статье предложен подход для построения *A*-цепи с упорядоченным охватыванием (*AOE*-цепи) в плоском связном 4-регулярном графе. Разработанный алгоритм позволяет решить задачу за полиномиальное время.

Изложенный алгоритм решения задачи маршрутизации в плоских графах решает задачу определения последовательности резки в соответствии с наложенными ограничениями по раскройному плану при использовании технологий ECP и ICP, допускающих возможность совмещения границ вырезаемых деталей. Применение этих технологий позволяет сократить расход материала, длину резки и длину холостых проходов.

В качестве направлений дальнейших исследований можно выделить разработку графического интерфейса для более удобного ввода исходных данных, создание библиотеки классов для решения задачи маршрутизации в плоских графах. Кроме того, открытой остается задача построения оптимального по длине холостых переходов покрытия несвязного графа цепями из классов *OE* и *AOE*.

Литература

1. Dewil R., Vansteenwegen P., Cattrysse D. A Review of Cutting Path Algorithms for Laser Cutters // International Journal Adv Manuf. Technol. 2016. Vol. 87. P. 1865–1884. DOI: 10.1007/s00170-016-8609-1

2. Makarovskikh T.A., Panyukov A.V., Savitsky E.A. Mathematical Models and Routing Algorithms for CAD Technological Preparation of Cutting Processes // Automation and Remote Control. 2017. Vol. 78, No. 4. P. 868–882.

3. Dewil R., Vansteenwegen P., Cattrysse D., Laguna M., Vossen T. An Improvement Heuristic Framework for the Laser Cutting Tool Path Problem // International Journal of Production

Research. 2015. Vol. 53, Issue 6. P. 1761–1776. DOI: 10.1080/00207543.2014.959268

4. Hoeft J., Palekar U. Heuristics for the Plate-cutting Travelling Salesman Problem. // IIE Transactions. 1997. Vol. 29(9). P. 719–731.

5. Dewil R., Vansteenwegen P., Cattrysse D. Construction Heuristics for Generating Tool Paths for Laser Cutters. // International Journal of Production Research. 2014. Vol. 52(20). P. 5965–5984.

6. Петунин А.А., Ченцов А.Г., Ченцов П.А. К вопросу о маршрутизации перемещений при листовой резке деталей // Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование. 2017. Т. 10, № 3. С. 25–39. DOI: 10.14529/mmp170303

7. Chentsov A.G., Grigoryev A.M., Chentsov A.A. Solving a Routing Problem with the Aid of an Independent Computations Scheme // Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование. 2018. Т. 11, № 1. С. 60–74. DOI: 10.14529/mmp180106

8. Petunin A., Stylios C. Optimization Models of Tool Path Problem for CNC Sheet Metal Cutting Machines. // 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016 Troyes, France, June 28–30, 2016. IFAC-PapersOnLine, 2016. Vol. 49. P. 23–28.

9. Chentsov A., Khachay M., Khachay D. Linear Time Algorithm for Precedence Constrained Asymmetric Generalized Traveling Salesman Problem // 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016, Troyes, France, June 28–30, 2016. IFAC-PapersOnLine, 2016. Vol. 49. P. 651–655.

10. Khachay M., Neznakhina K. Towards Tractability of the Euclidean Generalized Travelling Salesman Problem in Grid Clusters Defined by a Grid of Bounded Height // Communications in Computer and Information Science. 2018. Vol. 871. P. 68–77.

11. Manber U., Israni S. Pierce Point Minimization and Optimal Torch Path Determination in Flame Cutting // J. Manuf. Syst. Vol 3(1). 1984. P. 81–89. DOI: 10.1016/0278-6125(84)90024-4

12. Panyukova T.A. Constructing of OE-postman Path for a Planar Graph // Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование. 2014. Т. 7, № 4. С. 90–101. DOI: 10.14529/mmp140407

13. Garfinkel R.S., Webb I.R. On Crossings, the Crossing Postman Problem, and the Rural Postman Problem. // Networks. 1999. Vol. 34(3). P. 173–180.

14. Manber U., Bent S.W. On Non-intersecting Eulerian Circuits // Discrete Applied Mathematics. 1987. Vol. 18. P. 87–94. DOI: 10.1016/0166-218X(87)90045-X

15. Fleischner H. Eulerian Graphs and Related Topics. Part 1, Vol. 1.: Ann. Discrete Mathematics, 1990. № 45.

16. Филиппов А.Ф. Элементарное доказательство теоремы Жордана // Успехи математических наук. 1950. Т. 5, Вып. 5(39). С. 173–176.

17. Makarovskikh T., Panyukov A. The Cutter Trajectory Avoiding Intersections of Cuts // IFAC-PapersOnLine. 2017. Vol. 50, Issue 1. P. 2284-2289.

18. Makarovskikh T., Panyukov A. Development of Routing Methods for Cutting out Details // CEUR Workshop Proceedings. 2018. Vol. 2098. P. 249–263. URL: htpp://ceur-ws.org/Vol-2098 (дата обращения: 20.07.2018).

19. Зыков А.А. Основы теории графов. М.: Вузовская книга, 2004. 664 с.

20. Makarovskikh T.A., Panyukov A.V., Savitsky E.A. Mathematical Models and Routing Algorithms for CAM of Technological Support of Cutting Processes // ScienceDirect IFAC-PapersOnLine 49–12. 2016. P. 821–826. DOI: 10.1016/j.ifacol.2016.07.876

21. Szeider S. Finding Paths in Graphs Avoiding Forbidden Transitions // Discrete Applied Mathematics. 2003. No. 126. P. 261–273. DOI: 10.1016/S0166-218X(02)00251-2

22. Panyukova T. Chain Sequences with Ordered Enclosing // Journal of Computer and System Sciences International. 2007. Vol. 46, No. 1(10). P. 83–92. DOI: 10.1134/S1064230707010108

23. Савицкий Е.А. Использование алгоритма поиска в ширину для определения уровней вложенности ребер плоского графа // Информационные технологии и системы: Труды Третьей междунар. науч. конф. Челябинск: Изд-во ЧелГУ, 2014. С. 43–45.

24. Панюкова Т.А. Цепи с упорядоченным охватыванием в плоских графах // Дискретный анализ и исследование операций. Часть 2. 2006. Т. 13, № 2. С. 31–43.

Макаровских Татьяна Анатольевна, к.ф.-м.н., доцент, кафедра математического и компьютерного моделирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

DOI: 10.14529/cmse190103

SOFTWARE FOR CONSTRUCTING OF *A*-CHAINS WITH ORDERED ENCLOSING FOR A PLANE CONNECTED 4-REGULAR GRAPH

© 2019 T.A. Makarovskikh

South Ural State University (pr. Lenina 76, Chelyabinsk, 454080 Russia)

E-mail: Makarovskikh.T.A@susu.ru

Received: 24.07.2018

The task of constructing the cutting path of the tool when the part cut off from the sheet does not require additional cuts and the trajectories of cuts intersections are forbidden (the touches are allowed) may arise in CAD/CAM-systems of technological preparation of cutting processes. Formally, such a task can be formulated as the problem of constructing a self-non-intersecting chain in a plane Euler graph representing a homeomorphic image of a cutting plan. Finally, the tasks of constructing routes satisfying the technological constraints are reduced to defining an A-chain with ordered enclosing for a plane connected 4-regular graph. This article is devoted to the algorithm for finding such a chain. The execution of this algorithm consists of two stages. At the first stage, cut-vertices of rank k are identified and splitted. At the second stage, the construction of the chain starts from an arbitrary vertex incident to the outer face; the first edge of the chain is chosen to be an edge of maximal rank incident to a given vertex; next, an iterative process is organized where, as the next edge, an unpassed edge of maximum rank is chosen, which is the left or right neighbour of the current one. It is shown that the algorithm constructs a route with the indicated properties in linear time for a plane connected 4-regular graph. The considered algorithms are realized as a computer program. The examples of some test problems are given in this paper.

Keywords: plane graph, path, cutting plan, polynomial-time algorithm, CAD/CAM.

FOR CITATION

Makarovskikh T.A. Software for Constructing of *A*-chains with Ordered Enclosing for a Plane Connected 4-regular Graph. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2019. vol. 8, no. 1. pp. 36–53. (in Russian) DOI: 10.14529/cmse190103.

References

1. Dewil R., Vansteenwegen P., Cattrysse D. A Review of Cutting Path Algorithms for Laser Cutters. International Journal Adv Manuf. Technol. 2016. vol. 87. pp. 1865–1884. DOI: 10.1007/s00170-016-8609-1

2. Makarovskikh T.A., Panyukov A.V., Savitsky E.A. Mathematical Models and Routing Algorithms for CAD Technological Preparation of Cutting Processes. Automation and Remote Control. 2017. vol. 78, no. 4. pp. 868–882.

3. Dewil R., Vansteenwegen P., Cattrysse D., Laguna M., Vossen T. An Improvement Heuristic Framework for the Laser Cutting Tool Path Problem. International Journal of Production Research. 2015. vol. 53, iss. 6. pp. 1761–1776. DOI: 10.1080/00207543.2014.959268

4. Hoeft J., Palekar U. Heuristics for the Plate-cutting Travelling Salesman Problem. IIE Transactions. 1997. vol. 29(9). pp. 719–731.

5. Dewil R., Vansteenwegen P., Cattrysse D. Construction Heuristics for Generating Tool Paths for Laser Cutters. International Journal of Production Research. 2014. vol. 52(20). pp. 5965–5984.

6. Petunin A.A., Chentsov A.G., Chentsov P.A. On the Issue of Routing Movements in Sheet Cutting of Parts. *Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta. Seriya: Matematicheskoe modelirovanie i programmirovanie* [Bulletin of the South Ural State University, Series: Mathematical Modelling and Programming]. 2017. vol. 10, no. 3. pp. 25–39. DOI: 10.14529/mmp170303 (in Russian)

7. Chentsov A.G., Grigoryev A.M., Chentsov A.A. Solving a Routing Problem with the Aid of an Independent Computations Scheme. *Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta. Seriya: Matematicheskoe modelirovanie i programmirovanie* [Bulletin of the South Ural State University, Series: Mathematical Modelling and Programming]. 2018. vol. 11, no. 1. pp. 60–74. DOI: 10.14529/mmp180106 (in Russian)

8. Petunin A., Stylios C. Optimization Models of Tool Path Problem for CNC Sheet Metal Cutting Machines. 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016 Troyes, France, June 28–30, 2016. IFAC-PapersOnLine, 2016. vol. 49. pp. 23–28.

9. Chentsov A., Khachay M., Khachay D. Linear Time Algorithm for Precedence Constrained Asymmetric Generalized Traveling Salesman Problem. 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016, Troyes, France, June 28–30, 2016. IFAC-PapersOnLine, 2016. vol. 49. pp. 651–655.

10. Khachay M., Neznakhina K. Towards Tractability of the Euclidean Generalized Travelling Salesman Problem in Grid Clusters Defined by a Grid of Bounded Height. Communications in Computer and Information Science. 2018. vol. 871. pp. 68–77.

11. Manber U., Israni S. Pierce Point Minimization and Optimal Torch Path Determination in Flame Cutting. J. Manuf. Syst. 1984. vol. 3(1). pp. 81–89. DOI: 10.1016/0278-6125(84)90024-4

12. Panyukova T.A. Constructing of OE-postman Path for a Planar Graph. Bulletin of the South Ural State University, Series: Mathematical Modelling, Programming and Computer Software. 2014. vol. 7, no. 4. pp. 90–101. DOI: 10.14529/mmp140407

13. Garfinkel R.S., Webb I.R. On Crossings, the Crossing Postman Problem, and the Rural Postman Problem. Networks. 1999. vol. 34(3). pp. 173–180.

14. Manber U., Bent S.W. On Non-intersecting Eulerian Circuits. Discrete Applied Mathematics. 1987. vol. 18. pp. 87–94. DOI: 10.1016/0166-218X(87)90045-X

15. Fleischner H. *Eulerian Graphs and Related Topics. Part 1, Vol.1.* Ann. Discrete Mathematics. 1990. no. 45.

16. Filippov A.F. The Elementary Proof of Jordan Theorem *Uspekhi matematicheskikh nauk* [Successes of Mathematical Sciences]. 1950. vol. 5, no. 5(39). pp. 173–176. (in Russian)

17. Makarovskikh T., Panyukov A. The Cutter Trajectory Avoiding Intersections of Cuts. IFAC-PapersOnLine. 2017. vol. 50, issue 1. pp. 2284-2289.

18. Makarovskikh T., Panyukov A. Development of Routing Methods for Cutting out Details. CEUR Workshop Proceedings. 2018. vol. 2098. pp. 249–263. Available at: `htpp://ceur-ws.org/Vol-2098` (accessed: 20.07.2018).

19. Zykov A.A. *Osnovy teorii grafov* [The Basics of Graph Theory]. Moscow, High School Book, 2004. 664 p. (in Russian)

20. Makarovskikh T.A., Panyukov A.V., Savitsky E.A. Mathematical Models and Routing Algorithms for CAM of Technological Support of Cutting Processes. ScienceDirect IFAC-PapersOnLine 49–12. 2016. pp. 821–826. DOI: 10.1016/j.ifacol.2016.07.876

21. Szeider S. Finding Paths in Graphs Avoiding Forbidden Transitions. Discrete Applied Mathematics. 2003. no. 126. pp. 261–273. DOI: 10.1016/S0166-218X(02)00251-2

22. Panyukova T. Chain Sequences with Ordered Enclosing. Journal of Computer and System Sciences International. 2007. vol. 46, no. 1(10). pp. 83–92. DOI: 10.1134/S1064230707010108

23. Savitskiy E.A. Using of Wide-with Search Algorithm for Ranking of Edges of a Plane Graph. *Informacionnye tekhnologii i sistemy: tr. Tret'ej mezhdunar. nauch. konf.* [Information technologies and systems: Proceedings of the Third International conference]. Chelyabinsk, Publishing of Chelyabinsk State University. 2014. pp. 43–45. (in Russian)

24. Panyukova T.A. Chains with Ordered Enclosing for Plane Graphs. *Diskretnyj analiz i issledovanie operacij. Chast' 2.* [Discrete analysis and operation research. Part 2]. 2006. vol. 13, no. 2. pp. 31–43. (in Russian)

УДК 004.272.25, 004.421, 004.032.24 **DOI: 10.14529/cmse190104**

ПАРАЛЛЕЛЬНЫЙ ПОИСК ЧАСТЫХ НАБОРОВ НА МНОГОЯДЕРНЫХ УСКОРИТЕЛЯХ INTEL MIC

© 2019 М.Л. Цымблер

Южно-Уральский государственный университет
(454080 Челябинск, пр. им. В.И. Ленина, д. 76)
E-mail: mzym@susu.ru

Поступила в редакцию: 26.12.2018

Поиск ассоциативных правил предполагает нахождение устойчивых корреляций между наборами элементов в больших базах транзакционных данных и является одной из основных задач интеллектуального анализа данных. Ассоциативные правила генерируются на основе множества всех наборов, в которых элементы часто встречаются совместно. Алгоритм DIC (Dynamic Itemset Counting) является модификацией классического алгоритма Apriori поиска частых наборов. В отличие от предшественника DIC пытается сократить количество проходов по базе транзакций и сохранить при этом относительно небольшое количество наборов, поддержка которых подсчитывается в рамках одного прохода. В статье рассмотрена проблема ускорения алгоритма DIC на многоядерной архитектуре Intel Many Integrated Core (MIC) для случая, когда база транзакций помещается в оперативную память. Разработанная с помощью технологии OpenMP параллельная реализация алгоритма DIC использует битовое представление транзакций и наборов, что позволяет ускорить и векторизовать подсчет поддержки наборов, реализуемый посредством логических побитовых операций. Проведенные эксперименты с синтетическими и реальными данными подтвердили хорошую производительность и масштабируемость предложенного алгоритма.

Ключевые слова: интеллектуальный анализ данных, поиск ассоциативных правил, OpenMP, Intel Many Integrated Core.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Цымблер М.Л. Параллельный поиск частых наборов на многоядерных ускорителях Intel MIC // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2019. Т. 8, № 1. С. 54–70. DOI: 10.14529/cmse190104.

Введение

Поиск *ассоциативных правил (association rule mining)* предполагает нахождение часто повторяющихся зависимостей в заданном наборе объектов. Наглядным примером применения ассоциативных правил является *анализ рыночной корзины (market basket analysis)*, когда ставится задача нахождения наборов товаров в супермаркете, которые часто покупаются совместно. Ассоциативное правило представляет собой импликацию вида $A \rightarrow B$ $[support = x\ \%, confidence = y\ \%]$, где A и B — непустые и непересекающиеся наборы товаров, а $support$ и $confidence$ — меры полезности правила для аналитика. Поддержка (support) правила показывает, что в x % от общего числа покупок наборы A и B присутствовали одновременно. Достоверность (confidence) правила показывает, что y % покупателей, которые приобрели набор A, приобрели также набор B. Поиск ассоциативных правил применяется в медицине (например, нахождение побочных эффектов лекарств), генной инженерии (поиск часто повторяющихся цепочек ДНК) и других предметных областях.

Поиск ассоциативных правил может быть разбит на две последовательно выполняемые задачи: поиск всех частых наборов и генерация устойчивых ассоциативных правил на основе найденных частых наборов [2]. Формальное определение задачи поиска всех

частых наборов выглядит следующим образом. Пусть дано множество *объектов* $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, любое непустое его подмножество называют *набором*. Набор из k объектов ($1 \leqslant k \leqslant m$) называют *$k$-набором*. Пусть имеется множество *транзакций* $\mathcal{D}$, в котором каждая транзакция представляет собой пару $(tid; I)$, где tid — уникальный идентификатор транзакции, $I \subseteq \mathcal{I}$ — набор. *Поддержкой набора* $I \subseteq \mathcal{I}$ является доля транзакций $\mathcal{D}$, содержащих данный набор:

$$support(I) = \frac{|\{T \in \mathcal{D} \mid I \subseteq T.I\}|}{|\mathcal{D}|}. \quad (1)$$

Наперед задаваемый *порог поддержки* $minsup$ является параметром задачи. Набор, имеющий поддержку не ниже $minsup$, называют *частым*, иначе набор называют *редким*. Множество всех частых k-наборов обозначают $\mathcal{L}_k$. Решением задачи *поиска частых наборов* будет множество $\mathcal{L} = \cup_{k=1}^{k_{max}} \mathcal{L}_k$, где k_{max} — максимальное количество объектов в частом наборе.

В данной статье предлагается параллельный алгоритм поиска всех частых наборов для многоядерных ускорителей Intel Many Integrated Core (MIC) [10]. Ускорители Intel MIC основаны на архитектуре Intel x86 и поддерживают соответствующие модели и инструменты параллельного программирования. Устройства MIC обеспечивают большое количество энергоэффективных вычислительных ядер (до 72) с малой (относительно обычных многоядерных процессоров Intel) тактовой частотой, обладающих высокой пропускной способностью локальной памяти и 512-битными векторными регистрами. Системы MIC, как правило, дают наибольшую производительность в приложениях, предполагающих вычислительные операции над большими объемами данных (десятки миллионов элементов и более), векторизуемые компилятором [22]. Под векторизацией понимается замена нескольких скалярных операций в теле цикла с фиксированным количеством повторений на одну векторную операцию [3].

Остаток статьи организован следующим образом. В разделе 1 приведен обзор работ по тематике исследования. Описание последовательного алгоритма DIC приведено в разделе 2. Раздел 3 содержит описание предложенного параллельного алгоритма поиска частых наборов. Результаты экспериментального исследования разработанного алгоритма приведены в разделе 4. В заключении обсуждается область применения разработанного алгоритма и суммируются полученные результаты.

1. Обзор работ

Классическим алгоритмом решения задачи поиска частых наборов является алгоритм *Apriori* [2]. Идея *Apriori* заключается в итеративной генерации множества *кандидатов* в частые наборы и последующем отборе кандидатов с подходящим значением поддержки. Итерация осуществляется по k, количеству объектов в наборах-кандидатах, начиная с 1. В алгоритме используется следующее свойство *антимонотонности поддержки* (принцип *a priori*), которое позволяет исключать из рассмотрения заведомо редкие наборы: если k-набор является редким, то содержащий его $(k+1)$-набор также является редким.

Алгоритм *Dynamic Itemset Counting (DIC)* [5] является модификацией *Apriori*. В отличие от предшественника *DIC* пытается сократить количество проходов по множеству транзакций и сохранить при этом относительно небольшое количество наборов, поддержка которых подсчитывается в рамках одного прохода. Авторы отметили большой потенциал

распараллеливания предложенного алгоритма посредством разделения транзакций между вычислителями.

Алгоритм *DIC-OPT* [19] является параллельной версией алгоритма *DIC* для вычислительных систем с распределенной памятью. Основная идея алгоритма заключается в том, что каждый вычислительный узел выполняет рассылку сообщений, содержащих значения поддержки наборов, всем остальным узлам по завершении обработки очередного блока из M транзакций. Авторы провели эксперименты на вычислительной системе из 12 узлов, где *DIC-OPT* показал сублинейное ускорение.

В работе [8] представлен алгоритм *APM*, который является модификацией алгоритма *DIC* для SMP-систем. Процессоры SMP-системы динамически генерируют наборы-кандидаты независимо друг от друга и не требуют синхронизации. Транзакции распределяются по узлам системы. Эксперименты, проведенные на вычислительной системе Sun Enterprise 4000 из 12 узлов, показали, что APM опережает параллельные реализации классического алгоритма Apriori. Однако, ускорение APM постепенно снижается до 4, когда количество задействованных узлов больше четырех.

Алгоритм *mcEclat* [21] является параллельной версией *Eclat* [23] для сопроцессора Intel Xeon Phi. *mcEclat* использует представление транзакций в виде вертикальных битовых карт: *tid* всех транзакций, в которых присутствует данный объект, преобразуются в битовую карту этого объекта, где в соответствующих позициях биты установлены в 1. Для вычисления поддержки набора над битовыми картами входящих в набор объектов выполняется логическая побитовая операция `AND` и затем подсчитывается количество битов результата, установленных в 1. Эксперименты показали ускорение алгоритма до 100 на 240 нитях сопроцессора, однако реализация не в полной мере использует возможности векторизации вычислений Xeon Phi и не опережает себя на платформе двухпроцессорной системы Intel Xeon.

В работах [6, 9, 14] предложены различные последовательные алгоритмы поиска частых наборов на основе использования битовых карт: *MAFIA*, *BitTableFI* и *BitwiseDIC* (версия алгоритма *DIC* [5]) соответственно.

2. Последовательный алгоритм поиска частых наборов

Алгоритм *DIC* (см. алг. 1) разбивает множества транзакций $\mathcal{D}$ на логические блоки, состоящие из $\lceil \frac{|\mathcal{D}|}{M} \rceil$ транзакций, где число транзакций в блоке M ($1 \leqslant M \leqslant |\mathcal{D}|$) является параметром алгоритма.

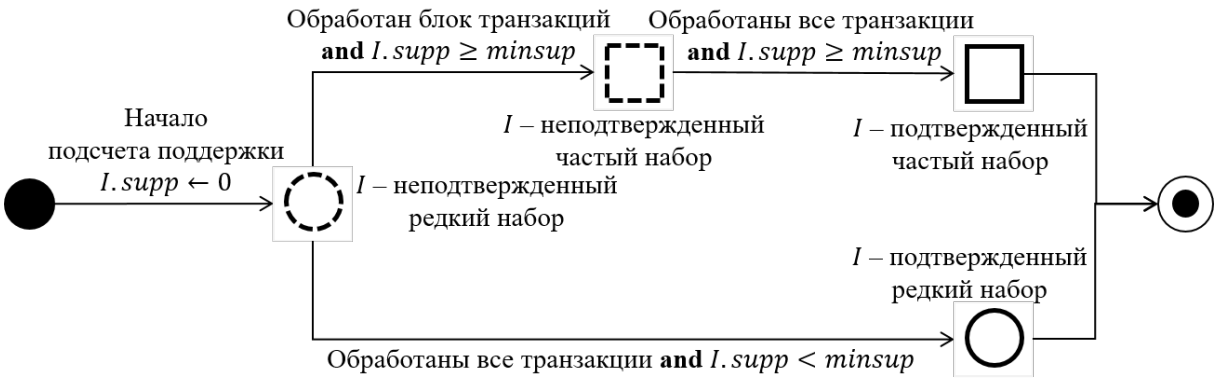


Рис. 1. Жизненный цикл набора в алгоритме *DIC*

Алгоритм 1 DIC(IN $\mathcal{D}$, $minsup$, M, OUT $\mathcal{L}$)

▷ Инициализация множеств наборов

1: $SolidBox \leftarrow \varnothing$; $SolidCircle \leftarrow \varnothing$; $DashedBox \leftarrow \varnothing$; $DashedCircle \leftarrow \mathcal{I}$
2: **while** $DashedCircle \cup DashedBox \neq \varnothing$ **do**

▷ Чтение транзакций

3: Read($\mathcal{D}, M, Chunk$)
4: **if** EOF($\mathcal{D}$) **then**
5: Rewind($\mathcal{D}$)
6: **for all** $T \in Chunk$ **do**

▷ Подсчет поддержки наборов

7: **for all** $I \in DashedCircle \cup DashedBox$ **do**
8: **if** $I \subseteq T$ **then**
9: $support(I) \leftarrow support(I) + 1$

▷ Генерация наборов-кандидатов

10: **for all** $I \in DashedCircle$ **do**
11: **if** $support(I) \geqslant minsup$ **then**
12: $DashedBox \leftarrow DashedBox \cup I$
13: **for all** $i \in \mathcal{I}$ **do**
14: $C \leftarrow I \cup i$
15: **if** $\forall s \subseteq C \ s \in SolidBox \cup DashedBox$ **then**
16: $DashedCircle \leftarrow DashedCircle \cup C$

▷ Проверка завершения полного прохода

17: **for all** $I \in DashedCircle \cup SolidBox$ **do**
18: **if** IsPassCompleted(I) **then**
19: **switch** Shape(I)
20: *dashed*: $DashedBox \leftarrow DashedBox \cup I$
21: *solid*: $SolidBox \leftarrow SolidBox \cup I$
22: $\mathcal{L} \leftarrow SolidBox$

DIC различает четыре вида наборов, для названия которых используются метафоры: *пунктирные окружности*, *пунктирные квадраты*, *сплошные окружности* и *сплошные квадраты*. Жизненный цикл набора в алгоритме *DIC* представлен на рис. 1. Для «пунктирных» наборов необходимо выполнить подсчет поддержки, в то время как для «сплошных» наборов подсчет поддержки закончен. «Квадрат» соответствует частому набору, «окружность» — редкому. В соответствии с этим определяются четыре непересекающихся множества наборов: *DashedCircle*, *DashedBox*, *SolidCircle* и *SolidBox*. Множества *DashedCircle* и *DashedBox* содержат неподтвержденные редкие и неподтвержденные частые наборы соответственно, а множества *SolidCircle* и *SolidBox* — подтвержденные редкие и подтвержденные частые наборы соответственно.

При инициализации множества *DashedBox*, *SolidCircle* и *SolidBox* полагаются пустыми, а множество *Dashed Circle* заполняется 1-наборами из множества объектов $\mathcal{I}$. При обработке транзакций блока *DIC* вычисляет поддержку «пунктирных» наборов из множеств *DashedCircle* и *DashedBox*. По завершении обработки блока наборы, поддержка которых сравнялась или превысила порог $minsup$ перемещаются из множества *DashedCircle* в *DashedBox*. В множество *DashedCircle* добавляется каждый набор, который является

надмножеством таких наборов из *DashedBox*, любое подмножество которых является «квадратом». После обработки последнего блока выполняется переход к первому блоку транзакций. Алгоритм завершается, когда множества *DashedCircle* и *DashedBox* становятся пустыми.

3. Поиск частых наборов на ускорителе Intel Xeon Phi

3.1. Проектирование структур данных

Параллельный алгоритм *PhiDIC* [24] использует *битовое представление* наборов и транзакций. Транзакция $T \subseteq \mathcal{D}$ (набор $I \subseteq \mathcal{I}$, соответственно) представляется в виде слова, в котором каждый $(p-1)$-й бит установлен в 1, если элемент $i_p \in T$ ($i_p \in I$, соответственно), и остальные биты установлены в 0. Количество битов в слове W зависит от используемой системы программирования и определяется как $W = \lceil \frac{m}{sizeof(byte)} \rceil$. В предлагаемой реализации используется система программирования C++, а набор и транзакция представлены значением типа данных `unsigned long long int`, таким образом, в данной реализации $W = 8$ и $m = 64$.

Введем *функцию битовой маски* $BitMask : \mathcal{I} \to \mathbb{N}$, которая возвращает натуральное число, являющееся битовым представлением указанной транзакции либо набора. Например, для набора $\{i_1, i_4, i_5\}$ функция $BitMask$ возвратит целое число 25, имеющее такое битовое представление, в котором 0-й, 3-й и 4-й биты установлены в 1, а остальные биты равны 0. Тогда *битовой картой* базы транзакций $\mathcal{D}$ назовем n-элементный одномерный массив $\mathcal{B}$, где $\mathcal{B}_j = BitMask(T_j)$, $1 \leqslant j \leqslant n$.

Использование битовых масок наборов и битовой карты базы транзакций упрощает подсчет поддержки наборов и обеспечивает векторизацию этой операции. Действительно, факт вхождения набора I в транзакцию T ($I \subseteq T$), установление которого необходимо при подсчете поддержки данного набора, может быть определен с помощью одной логической побитовой операции $BitMask(I)$ `AND` $BitMask(T)$=$BitMask(I)$, а циклическое выполнение данной операции может быть автоматически векторизовано компилятором.

Набор, таким образом, реализуется как структура со следующими полями:

- *mask* — битовая маска набора;
- k — количество элементов в наборе;
- *stop* — счетчик части транзакций, в которых проверено наличие данного набора;
- *supp* — поддержка данного набора;
- *shape* — вид данного набора (`BOX` или `CIRCLE`, либо `NULL`).

Множество наборов реализуется с помощью класса `vector` из стандартной библиотеки классов C++ (Standard Template Library) [16]. Данный класс реализует массив элементов, принадлежащих одному типу данных и обеспечивает операции добавления, удаления элементов, доступ к элементу по его индексу и итерацию элементов массива. Каждый из векторов `DASHED` и `SOLID` обеспечивает хранение двух множеств наборов: *DashedCircle*, *DashedBox* и *SolidCircle*, *SolidBox* соответственно. Хранение двух множеств в рамках одного вектора обеспечивает меньшие накладные расходы, чем выделение одного вектора на каждое множество элементов: для перемещения элемента из одного множества в другое достаточно изменить значение поля *shape*.

Алгоритм 2 PhiDIC(in $\mathcal{B}$, $minsup$, M; out $\mathcal{L}$)

```
1: SOLID.init(); DASHED.init()
2: for all i ∈ 0..m − 1 do
3:     I.shape ← NIL; I.mask ← 0; I.mask ← SetBit(I.mask, i)
4:     I.stop ← 0; I.supp ← 0; I.k ← 1
5:     SOLID.push_back(I)
6: k ← 1; stop ← 0; stop_max ← ⌈n/M⌉
7: while not DASHED.empty() do
8:     k ← k + 1; stop ← stop + 1
9:     if stop > stop_max then
10:        stop ← 1
11:    first ← (stop − 1) · M; last ← stop · M − 1
12:    CountSupport(DASHED)
13:    Prune(DASHED)
14:    MakeCandidates(DASHED)
15:    CheckFullPass(DASHED)
16: L ← {I | I ∈ SOLID ∧ I.shape = BOX}
```

3.2. Распараллеливание поиска частых наборов

Разработанная параллельная реализация поиска частых наборов представлена в алг. 2. Распараллеливанию подвергаются следующие стадии алгоритма: подсчет поддержки (см. алг. 3), отбрасывание заведомо редких наборов (см. алг. 4) и проверка завершения просмотра наборов-кандидатов по всей базе транзакций (см. алг. 5).

Реализация подсчета поддержки показана в алг. 3. Подсчет выполняется посредством двух вложенных циклов: внешний распараллеливается и выполняется по наборам-кандидатам, а внутренний — по транзакциям. Это позволяет избежать гонок данных при обновлении поддержки одного и того же набора разными нитями, которое может быть выполнено одновременно. Алгоритм различает два случая в зависимости от того, больше ли мощность множества наборов-кандидатов, чем количество нитей, или нет. В первом случае внешний цикл распараллеливается на все доступные нити. Во втором случае алгоритм активирует режим вложенного параллелизма, и внешний цикл распараллеливается на количество нитей, равное количеству наборов-кандидатов.

Внутренний цикл по транзакциям распараллеливается таким образом, что каждая нить внешнего цикла инициирует (*fork*) одинаковое количество подчиненных нитей, выполняющих подсчет поддержки. Указанная техника позволяет сбалансировать нагрузку нитей на финальной стадии алгоритма, когда общее количество наборов-кандидатов последовательно уменьшается, что позволяет повысить общую производительность алгоритма.

Реализация отбрасывания заведомо редких наборов представлена в алг. 4. Максимально возможная поддержка набора вычисляется путем сложения текущего значения поддержки данного набора с количеством транзакций, которые еще не были обработаны. Если вычисленная максимально возможная поддержка меньше, чем порог *minsup*, то данный набор заведомо редкий и может быть исключен из дальнейшего рассмотрения. Далее

Алгоритм 3 CountSupport(in out $DASHED$)

```
1: if DASHED.size() ⩾ num_of_threads then
2:     #pragma omp parallel for
3:     for all I ∈ DASHED do
4:         I.stop ← I.stop + 1
5:         for all T ∈ B_first .. B_last do
6:             if I.mask AND T = I.mask then
7:                 I.supp ← I.supp + 1
8: else
9:     omp_set_nested(true)
10:    #pragma omp parallel for num_threads(DASHED.size())
11:    for all I ∈ DASHED do
12:        I.stop ← I.stop + 1
13:        #pragma omp parallel for reduction(+:I.supp) num_threads(⌈num_of_threads / DASHED.size()⌉)
14:        for all T ∈ B_first .. B_last do
15:            if I.mask AND T = I.mask then
16:                I.supp ← I.supp + 1
```

Алгоритм 4 Prune(in out $DASHED$)

```
1: #pragma omp parallel for
2: for all I ∈ DASHED and I.shape = CIRCLE do
3:     if I.supp ⩾ minsup then
4:         I.shape ← BOX
5:     else
6:         supp_max ← I.supp + M · (stop_max − I.stop)
7:         if supp_max < minsup then
8:             I.shape ← NIL
9:             for all J ∈ DASHED and J.shape = CIRCLE do
10:                if I.mask AND J.mask = I.mask then
11:                    J.shape ← NIL
12: DASHED.erase({I | I.shape = NIL})
```

Алгоритм 5 CheckFullPass(in out $DASHED$)

```
1: #pragma omp parallel for
2: for all I ∈ DASHED do
3:     if I.stop = stop_max then
4:         if I.supp ⩾ minsup then
5:             I.shape ← BOX
6:         SOLID.push_back(I)
7:         I.shape ← NIL
8: DASHED.erase({I | I.shape = NIL})
```

в соответствии с принципом *a priori* отбрасывается каждый набор-кандидат, который является надмножеством отброшенного набора.

По завершении отбрасывания подпрограмма `MakeCandidates` выполняет генерацию наборов-кандидатов для их обработки на следующей итерации алгоритма. Новые кандидаты получаются посредством применения логической побитовой операции `OR` к каждой паре наборов из множества `DASHED`, помеченных как `BOX` (частый). Финальным шагом обработки наборов-кандидатов является проверка завершения просмотра этих наборов по всей базе данных транзакций (см. алг. 5). Если просмотр завершен, набора перемещается в множество `SOLID`. Если при этом набор имеет поддержку не ниже порогового значения, он помечается как `BOX` (частый). Цикл обработки наборов-кандидатов распараллеливается с помощью директивы компилятора `#pragma omp parallel for`. По завершении обработки все искомые частые наборы будут перемещены в множество `SOLID` и помечены как `BOX`.

4. Вычислительные эксперименты

4.1. Цели, аппаратная платформа и наборы данных

Для исследования эффективности разработанного алгоритма были проведены вычислительные эксперименты на вычислительном узле кластерной системы «Торнадо ЮУрГУ» [13], характеристики которого приведены в табл. 1.

Таблица 1

Аппаратная платформа экспериментов

Характеристика	Хост	Сопроцессор
Модель, Intel Xeon	X5680	Phi (KNC), SE10X
Количество физических ядер	2×6	61
Гиперпоточность	2	4
Количество логических ядер	24	244
Частота, ГГц	3,33	1,1
Размер VPU, бит	128	512
Пиковая производительность, TFLOPS	0,371	1,076

В работе [24] были проведены эксперименты, показавшие, что при поиске частых наборов в базах из сотен тысяч транзакций масштабируемость *PhiDIC* деградирует, поскольку такие объемы не обеспечивают достаточную вычислительную нагрузку на ускоритель при подсчете поддержки. В данной статье в качестве наборов данных для экспериментов использовались базы из десятков миллионов транзакций, представленные в табл. 2.

Таблица 2

Наборы данных для экспериментов

Набор данных	Вид	Транзакции			Частые наборы (при $minsup = 0.1$)	
		n	m	Ср. длина	Количество	k_{max}
20M	Синтетический	$2 \cdot 10^7$	64	40	4 606	6
Tornado20M	Реальный	$2 \cdot 10^7$	64	15	346	4

Синтетический набор данных 20M подготовлен с помощью генератора IBM Quest Data Generator [12], использованного в экспериментах с оригинальным алгоритмом *DIC* [5]. В итоге набор данных 20M содержит 4 606 частых набора, самый длинный из которых состоит из 6 элементов.

Набор Tornado20M представляет собой журнал с показаниями датчиков напряжения в вычислительных узлах суперкомпьютера «Торнадо ЮУрГУ» [13], снятых в течение одного месяца. Данный журнал используется для нахождения устойчивых ассоциативных правил, связующих вычислительные шкафы, полки, узлы суперкомпьютера и опасные значения напряжения. «Торнадо ЮУрГУ» состоит из 8 шкафов, каждый шкаф состоит из 8 полок, каждая полка состоит из 6 узлов. Рассматривается 4 возможных значения измеряемого напряжения, для каждого из которых различают 4 статуса («меньше нормы», «норма», «больше нормы», «ошибка измерения»). В соответствии с этим транзакция журнала может быть закодирована с помощью 64 бит (8 бит на номер шкафа, 8 бит на номер полки, 48 бит на 6 узлов, где каждая пара битов отражает статус измеренного напряжения). В итоге набор данных Tornado20M содержит 340 частых наборов, самый длинный из которых состоит из 4 элементов.

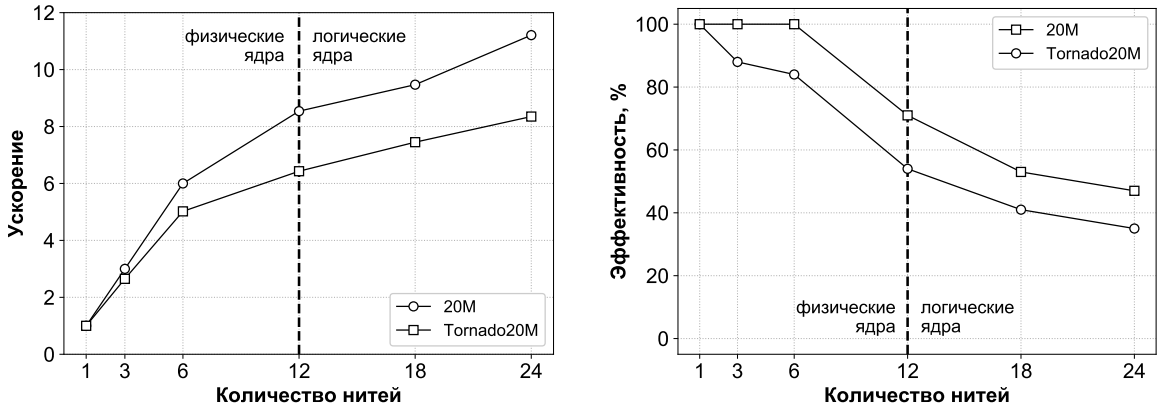
В экспериментах в качестве порога поддержки взято значение $minsup = 0,1$ как наиболее типичное. В качестве параметра количества транзакций в блоке взято значение $M = \lceil n/2 \rceil$, которое минимизирует накладные расходы на создание нитей для подсчета поддержки.

В экспериментах исследовались производительность, ускорение и параллельная эффективность алгоритма *PhiDIC*. Под *производительностью* понимается время работы алгоритма без учета времени загрузки данных в память и выдачи результата. *Ускорение* и *параллельная эффективность* параллельного алгоритма [1], запускаемого на k нитях, вычисляются как $s(k) = \frac{t_1}{t_k}$ и $e(k) = \frac{s(k)}{k}$ соответственно, где t_1 и t_k — время работы алгоритма на одной и k нитях соответственно.

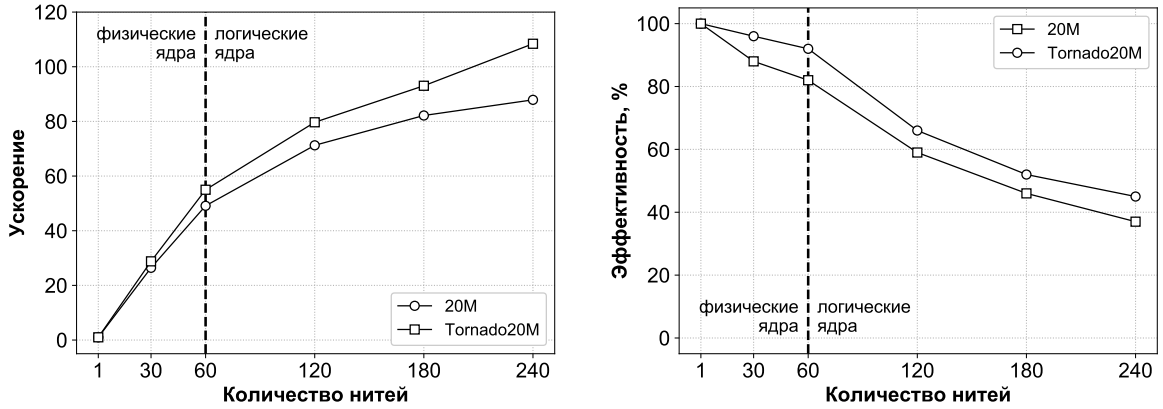
4.2. Результаты и обсуждение

Результаты экспериментов по исследованию ускорения и параллельной эффективности алгоритма *PhiDIC* представлены на рис. 2. На платформе Intel Xeon Phi алгоритм *PhiDIC* показывает ускорение, близкое к линейному и параллельную эффективность, близкую к 100 %, когда количество нитей, на которых запущен алгоритм, совпадает с количеством физических ядер процессора. Если при запуске алгоритм использует более одной нити на физическое процессорное ядро, то ускорение становится сублинейным (его значение уменьшается до 88 и 108 соответственно для наборов данных 20M и Tornado20M), а параллельная эффективность снижается соответствующим образом (до 37 % и 45 % для соответствующих наборов данных). На платформе вычислительного узла с двумя процессорами Intel Xeon наблюдается схожая картина, хотя и с несколько более скромными результатами для набора данных Tornado20M. Ускорение и параллельная эффективность алгоритма в экспериментах на наборе данных Tornado20M уменьшаются до значений 8 и 35 % соответственно, когда алгоритм запускается на максимальном количестве физических ядер.

Результаты экспериментов по исследованию производительности алгоритма *PhiDIC* представлены на рис. 3. Можно видеть, что алгоритм *PhiDIC* работает до полутора раз быстрее на платформе многоядерного процессора Intel Xeon Phi, чем на платформе

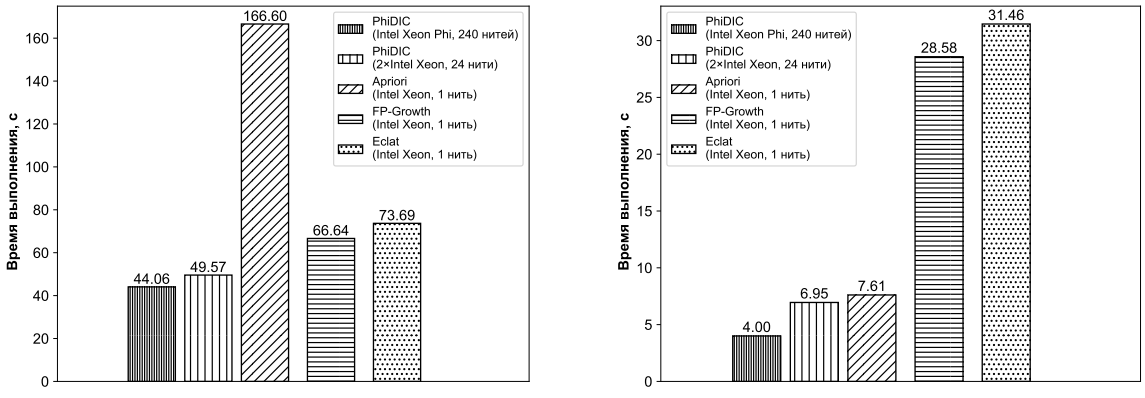


а) Платформа 2×Intel Xeon



б) Платформа Intel Xeon Phi

Рис. 2. Ускорение и параллельная эффективность алгоритма



а) Набор 20M б) Набор Tornado20M

Рис. 3. Производительность алгоритма

вычислительного узла, состоящего из двух обычных процессоров Intel Xeon. Алгоритм *PhiDIC* на платформе многоядерного процессора Intel Xeon Phi опережает в два раза лучшие результаты последовательных алгоритмов-конкурентов (*Apriori* [2], *Eclat* [23] и FP-Growth [11] в реализации [4]) на платформе процессора Intel Xeon.

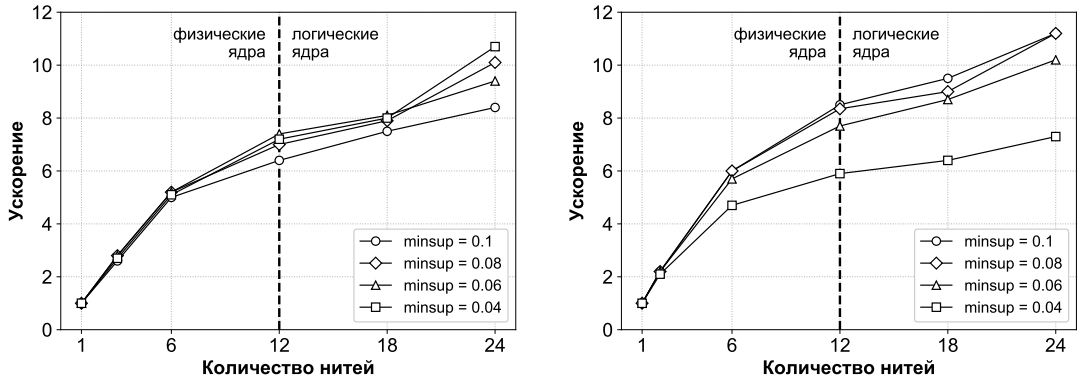
В табл. 3 представлены результаты экспериментов с набором данных Tornado20M, показывающих выгоду векторизации вычислений. Можно видеть, что производительность алгоритма *PhiDIC* на платформе Intel Xeon Phi увеличивается в два раза, если обеспечивается векторизация вычислений.

Таблица 3

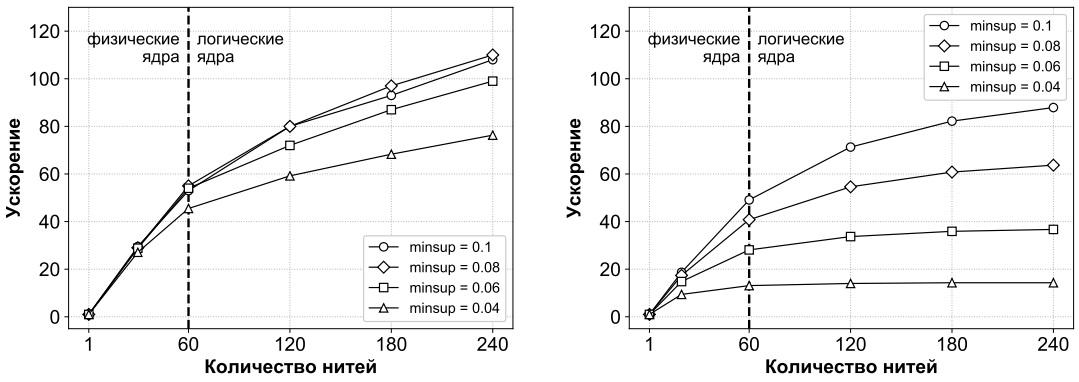
Влияние векторизации на производительность

Платформа	Время выполнения алгоритма, с	
	векторизация включена	векторизация отключена
Intel Xeon Phi	4,00	10,36
Intel Xeon	6,95	8,55

На рис. 4 показано влияние порога *minsup* на ускорение алгоритма. На обеих платформах и для обоих наборов данных ускорение алгоритма ожидаемо страдает от уменьшения значения *minsup*, поскольку это существенно увеличивает количество наборов-кандидатов для вычисления поддержки. Алгоритм *PhiDIC* все еще показывает лучшее ускорение, когда задействованы только физические ядра, и лучшее ускорение на платформе Intel Xeon Phi, чем на двухпроцессорной системе Intel Xeon.



а) Платформа 2×Intel Xeon



б) Платформа Intel Xeon Phi

Рис. 4. Влияние параметра *minsup* на ускорение алгоритма (слева — набор Tornado20M, справа — набор 20M)

Заключение

В статье предложен параллельный алгоритм решения задачи поиска частых наборов *PhiDIC* для многоядерного ускорителя архитектуры Intel MIC. Алгоритм предполагает битовое представление транзакций и наборов. Использование битовых масок наборов и битовой карты базы транзакций упрощает подсчет поддержки наборов и обеспечивает векторизацию этой операции.

С помощью технологии OpenMP распараллеливаются следующие стадии алгоритма: подсчет поддержки, отбрасывание заведомо редких наборов-кандидатов и проверка завершения просмотра наборов-кандидатов по всей базе транзакций. Подсчет поддержки выполняется посредством двух вложенных циклов: внешний распараллеливается и выполняется по наборам-кандидатам, а внутренний — по транзакциям, что позволяет избежать гонок данных. Если мощность множества наборов-кандидатов больше, чем количество нитей, внешний цикл распараллеливается на все доступные нити. В противном случае алгоритм активирует режим вложенного параллелизма, и внешний цикл распараллеливается на количество нитей, равное количеству наборов-кандидатов, а каждая нить внешнего цикла инициирует одинаковое количество подчиненных нитей, выполняющих подсчет поддержки. Указанная техника позволяет сбалансировать нагрузку нитей на финальной стадии алгоритма, когда общее количество наборов-кандидатов последовательно уменьшается, и повысить общую производительность алгоритма.

Результаты проведенных вычислительных экспериментов на синтетических и реальных наборах, состоящих из десятков миллионов транзакций, показывают, что разработанный алгоритм демонстрирует ускорение, близкое к линейному и параллельную эффективность, близкую к 100 %, когда количество нитей, на которых запущен алгоритм, совпадает с количеством физических ядер процессора.

Битовое представление наборов и транзакций, используемое в алгоритме *PhiDIC*, означает, что каждые набор и транзакция, представляемые целым положительным числом, не могут состоять из более чем 64 объектов. Данное ограничение, очевидно, неприемлемо для поиска частых наборов в транзакциях покупок в супермаркете [5], ДНК-микрочипов [7] и др. Однако алгоритм *PhiDIC* потенциально применим для поиска шаблонов в медицинских данных, что подтверждается следующими научными публикациями. В работе [15] описан поиск шаблонов риска заболевания у пациентов клиники, где количество атрибутов не превышает 30. В работах [17, 18] описан механизм трансформации данных медицинских карт в формат транзакций для последующего поиска частых наборов. В экспериментах авторы взяли не более 25 атрибутов из более чем 100 атрибутов медицинской карты пациента, поскольку выбранные атрибуты могут дать полную картину течения болезни. Кроме того, опыт авторов показал, что шаблоны, в которые входит более чем пять медицинских атрибутов, трудно поддаются интерпретации врачами. В работе [20] описан поиск шаблонов в базе данных больницы с более чем 2,5 млн. транзакций с данными пациентов, включая атрибуты, касающиеся демографии, диагностики и употребления лекарств.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 17-07-00463), Правительства РФ в соответствии с Постановлением № 211 от 16.03.2013 (соглашение № 02.A03.21.0011) и Министерства образования и науки РФ (государственное задание 2.7905.2017/8.9).

Литература

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.
2. Agrawal R., Srikant R. Fast Algorithms for Mining Association Rules in Large Databases // VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago de Chile, Chile. 1994. P. 487–499.
3. Bacon D.F., Graham S.L., Sharp O.J. Compiler Transformations for High-Performance Computing // ACM Computing Surveys. 1994. Vol. 26, No. 4. P. 345–420. DOI: 10.1145/197405.197406.
4. Borgelt C. Frequent Item Set Mining // Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2012. Vol. 2, No. 6, P. 437–456. DOI: 10.1002/widm.1074.
5. Brin S., Motwani R., Ullman J.D., Tsur S. Dynamic Itemset Counting and Implication Rules for Market Basket Data // SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13–15, 1997, Tucson, Arizona, USA. 1997. P. 255–264. DOI: 10.1145/253260.253325.
6. Burdick D., Calimlim M., Flannick J. *et al.* MAFIA: A Maximal Frequent Itemset Algorithm // IEEE Transactions on Knowledge and Data Engineering. 2005. Vol. 17, No. 11. P. 1490–1504. DOI: 10.1109/TKDE.2005.183.
7. Chang Y., Chen J., Tsai Y. Mining Subspace Clusters from DNA Microarray Data Using Large Itemset Techniques // Journal of Computational Biology. 2009. Vol. 16, No. 5. P. 745–768. DOI: 10.1089/cmb.2008.0161.
8. Cheung D.W., Hu K., Xia S. An Adaptive Algorithm for Mining Association Rules on Shared-Memory Parallel Machines // Distributed and Parallel Databases. 2001. Vol. 9, No. 2. P. 99–132. DOI: 10.1023/A:1018951022124.
9. Dong J., Han M. BitTableFI: An Efficient Mining Frequent Itemsets Algorithm // Knowledge-Based Systems. 2007. Vol. 20, No. 4. P. 329–335. DOI: j.knosys.2006.08.005.
10. Duran A., Klemm M. The Intel Many Integrated Core Architecture // 2012 International Conference on High Performance Computing and Simulation, HPCS 2012, Madrid, Spain, July 2–6, 2012. 2012. P. 365–366. DOI: 10.1109/HPCSim.2012.6266938.
11. Han J., Pei J., Yin Y. Mining Frequent Patterns without Candidate Generation // Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16–18, 2000, Dallas, Texas, USA. P. 1–12. DOI: 10.1145/342009.335372.
12. IBM Quest Synthetic Data Generator. URL: https://ibmquestdatagen.sourceforge.io/ (дата обращения: 03.11.2018).
13. Kostenetskiy P., Safonov A. SUSU Supercomputer Resources // PCT'2016, International Scientific Conference on Parallel Computational Technologies, Arkhangelsk, Russia, March 29–31, 2016. CEUR Workshop Proceedings. Vol. 1576, CEUR-WS.org, 2016. P. 561–573.
14. Kumar P., Bhatt P., Choudhury R. Bitwise Dynamic Itemset Counting Algorithm // Proceedings of the ICCIC 2015, IEEE International Conference on Computational Intelligence and Computing Research, December 10–12, 2015, Madurai, India. 2015. P. 1–4. DOI: 10.1109/ICCIC.2015.7435752.

15. Li J., Fu A.W., Fahey P. Efficient Discovery of Risk Patterns in Medical Data // Artificial Intelligence in Medicine. 2009. Vol. 45, No. 1. P. 77–89. DOI: 10.1016/j.artmed.2008.07.008.

16. Lischner R. STL Reference: Containers, Iterators, and Algorithms. O'Reilly, 2003. 120 p.

17. Ordonez C., Ezquerra N.F., Santana C.A. Constraining and Summarizing Association Rules in Medical Data // Knowledge and Information Systems. 2006. Vol. 9, No. 3. P. 1–2. DOI: 10.1007/s10115-005-0226-5.

18. Ordonez C., Santana C.A., de Braal L. Discovering Interesting Association Rules in Medical Data // 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Dallas, Texas, USA, May 14, 2000. 2000. P. 78–85.

19. Paranjape-Voditel P., Deshpande U. A DIC-based Distributed Algorithm for Frequent Itemset Generation // Journal of Software. 2011. Vol. 6, No. 2. P. 306–313. DOI: 10.4304/jsw.6.2.306-313.

20. Pattanaprateep O., McEvoy M., Attia J., Thakkinstian A. Evaluation of Rational Nonsteroidal Anti-inflammatory Drugs and Gastro-protective Agents Use; Association Rule Data Mining Using Outpatient Prescription Patterns // BMC Medical Informatics and Decision Making. 2017. Vol. 17, No. 1. P. 96:1–96:7. DOI: 10.1186/s12911-017-0496-3.

21. Schlegel B., Karnagel T., Kiefer T., Lehner W. Scalable Frequent Itemset Mining on Many-core Processors // Proceedings of the 9th International Workshop on Data Management on New Hardware, DaMoN 2013, New York, NY, USA, June 24, 2013. 2013. P. 3. DOI: 10.1145/2485278.2485281.

22. Sokolinskaya I., Sokolinsky L. Revised Pursuit Algorithm for Solving Non-stationary Linear Programming Problems on Modern Computing Clusters with Manycore Accelerators // Supercomputing. RuSCDays 2016. Communications in Computer and Information Science. 2016. Vol. 687. P. 212–223. DOI: 10.1007/978-3-319-55669-7_17.

23. Zaki M.J. Scalable Algorithms for Association Mining // IEEE Transactions on Knowledge and Data Engineering. 2000. Vol. 12, No. 3. P. 372–390. DOI: 10.1109/69.846291.

24. Zymbler M. Accelerating Dynamic Itemset Counting on Intel Many-core Systems // Proceedings of the 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO'2017, Opatija, Croatia, May 22–26, 2017. IEEE, 2017. P. 1575–1580. DOI: 110.23919/MIPRO.2017.797363.

Цымблер Михаил Леонидович, к.ф.-м.н., доцент, кафедра системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

DOI: 10.14529/cmse190104

PARALLEL FREQUENT ITEMSET MINING ON THE INTEL MIC ACCELERATORS

© 2019 M.L. Zymbler

South Ural State University (pr. Lenina 76, Chelyabinsk, 454080 Russia)

E-mail: mzym@susu.ru

Received: 26.12.2018

Association rule mining is one of the basic problems of data mining, which supposes finding strong correlations between itemsets in large transaction database. Association rules are generated from frequent itemsets (itemset is frequent if its items frequent occur together in transactions). The DIC (Dynamic Itemset Counting) algorithm is modification of the classical Apriori algorithm of finding frequent itemsets. DIC tries to reduce the number of passes made over the transaction database while keeping the number of itemsets counted in a pass relatively low. The paper addresses the task of accelerating DIC on the Intel MIC (Many Integrated Core) systems in the case when the transaction database fits into the main memory. The paper presents a parallel implementation of DIC based on OpenMP technology and thread-level parallelism. We exploit the bit-based internal layout for transactions and itemsets. This technique simplifies the support count via logical bitwise operation, and allows for vectorization of such a step. Experiments with large synthetic and real databases showed good performance and scalability of the proposed algorithm.

Keywords: data mining, frequent itemset counting, OpenMP, Intel Many Integrated Core.

FOR CITATION

Zymbler M.L. Parallel Frequent Itemset Mining on the Intel MIC Accelerators. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2019. vol. 8, no. 1. pp. 54–70. (in Russian) DOI: 10.14529/cmse190104.

References

1. Voevodin V.V., Voevodin Vl.V. *Parallellnye vachisleniya* [Parallel Computing]. SPb: BHV-Petersburg, 2002. 608 p.

2. Agrawal R., Srikant R. Fast Algorithms for Mining Association Rules in Large Databases. VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago de Chile, Chile. 1994. pp. 487–499.

3. Bacon D.F., Graham S.L., Sharp O.J. Compiler Transformations for High-Performance Computing. ACM Computing Surveys. 1994. vol. 26, no. 4. pp. 345–420. DOI: 10.1145/197405.197406.

4. Borgelt C. Frequent Item Set Mining. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2012. vol. 2, no. 6, pp. 437–456. DOI: 10.1002/widm.1074.

5. Brin S., Motwani R., Ullman J.D., Tsur S. Dynamic Itemset Counting and Implication Rules for Market Basket Data. SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13–15, 1997, Tucson, Arizona, USA. 1997. P. 255–264. DOI: 10.1145/253260.253325.

6. Burdick D., Calimlim M., Flannick J. *et al.* MAFIA: A Maximal Frequent Itemset Algorithm. IEEE Transactions on Knowledge and Data Engineering. 2005. vol. 17, no. 11. pp. 1490–1504. DOI: 10.1109/TKDE.2005.183.

7. Chang Y., Chen J., Tsai Y. Mining Subspace Clusters from DNA Microarray Data Using Large Itemset Techniques. Journal of Computational Biology. 2009. vol. 16, no. 5. pp. 745–768. DOI: 10.1089/cmb.2008.0161.

8. Cheung D.W., Hu K., Xia S. An Adaptive Algorithm for Mining Association Rules on Shared-Memory Parallel Machines. Distributed and Parallel Databases. 2001. vol. 9, no. 2. pp. 99–132. DOI: 10.1023/A:1018951022124.

9. Dong J., Han M. BitTableFI: An Efficient Mining Frequent Itemsets Algorithm. Knowledge-Based Systems. 2007. vol. 20, no. 4. pp. 329–335. DOI: j.knosys.2006.08.005.

10. Duran A., Klemm M. The Intel Many Integrated Core Architecture. 2012 International Conference on High Performance Computing and Simulation, HPCS 2012, Madrid, Spain, July 2–6, 2012. 2012. pp. 365–366. DOI: 10.1109/HPCSim.2012.6266938.

11. Han J., Pei J., Yin Y. Mining Frequent Patterns without Candidate Generation. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16–18, 2000, Dallas, Texas, USA. pp. 1–12. DOI: 10.1145/342009.335372.

12. IBM Quest Synthetic Data Generator. URL: https://ibmquestdatagen.sourceforge.io/ (accessed: 03.11.2018).

13. Kostenetskiy P., Safonov A. SUSU Supercomputer Resources. PCT'2016, International Scientific Conference on Parallel Computational Technologies, Arkhangelsk, Russia, March 29–31, 2016. CEUR Workshop Proceedings. vol. 1576, CEUR-WS.org, 2016. pp. 561–573.

14. Kumar P., Bhatt P., Choudhury R. Bitwise Dynamic Itemset Counting Algorithm. Proceedings of the ICCIC 2015, IEEE International Conference on Computational Intelligence and Computing Research, December 10–12, 2015, Madurai, India. 2015. pp. 1–4. DOI: 10.1109/ICCIC.2015.7435752.

15. Li J., Fu A.W., Fahey P. Efficient Discovery of Risk Patterns in Medical Data. Artificial Intelligence in Medicine. 2009. vol. 45, no. 1. pp. 77–89. DOI: 10.1016/j.artmed.2008.07.008.

16. Lischner R. STL Reference: Containers, Iterators, and Algorithms. O'Reilly, 2003. 120 p.

17. Ordonez C., Ezquerra N.F., Santana C.A. Constraining and Summarizing Association Rules in Medical Data. Knowledge and Information Systems. 2006. vol. 9, no. 3. pp. 1–2. DOI: 10.1007/s10115-005-0226-5.

18. Ordonez C., Santana C.A., de Braal L. Discovering Interesting Association Rules in Medical Data. 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Dallas, Texas, USA, May 14, 2000. 2000. pp. 78–85.

19. Paranjape-Voditel P., Deshpande U. A DIC-based Distributed Algorithm for Frequent Itemset Generation. Journal of Software. 2011. vol. 6, no. 2. pp. 306–313. DOI: 10.4304/jsw.6.2.306-313.

20. Pattanaprateep O., McEvoy M., Attia J., Thakkinstian A. Evaluation of Rational Nonsteroidal Anti-inflammatory Drugs and Gastro-protective Agents Use; Association Rule Data Mining Using Outpatient Prescription Patterns. BMC Medical Informatics and Decision Making. 2017. vol. 17, no. 1. pp. 96:1–96:7. DOI: 10.1186/s12911-017-0496-3.

21. Schlegel B., Karnagel T., Kiefer T., Lehner W. Scalable Frequent Itemset Mining on Many-core Processors. Proceedings of the 9th International Workshop on Data Management on New Hardware, DaMoN 2013, New York, NY, USA, June 24, 2013. 2013. pp. 3. DOI: 10.1145/2485278.2485281.

22. Sokolinskaya I., Sokolinsky L. Revised Pursuit Algorithm for Solving Non-stationary Linear Programming Problems on Modern Computing Clusters with Manycore Accelerators. Supercomputing. RuSCDays 2016. Communications in Computer and Information Science. 2016. vol. 687. pp. 212–223. DOI: 10.1007/978-3-319-55669-7_17.

23. Zaki M.J. Scalable Algorithms for Association Mining. IEEE Transactions on Knowledge and Data Engineering. 2000. vol. 12, no. 3. pp. 372–390. DOI: 10.1109/69.846291.

24. Zymbler M. Accelerating Dynamic Itemset Counting on Intel Many-core Systems. Proceedings of the 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO'2017, Opatija, Croatia, May 22–26, 2017. IEEE, 2017. pp. 1575–1580. DOI: 110.23919/MIPRO.2017.797363.

УДК 577.151, 004.382.2 DOI: 10.14529/cmse190105

МОЛЕКУЛЯРНАЯ ДИНАМИКА В СИЛОВОМ ПОЛЕ FF14SB В ВОДЕ TIP4P-EW, И В СИЛОВОМ ПОЛЕ FF15IPQ В ВОДЕ SPC/E_B: СРАВНИТЕЛЬНЫЙ АНАЛИЗ НА GPU И CPU*

© 2019 Д.А. Суплатов, Я.А. Шарапова, Н.Н. Попова, К.Е. Копылов, Вл.В. Воеводин, В.К. Швядас

Московский государственный университет имени М.В. Ломоносова
(119991 Москва, ул. Ленинские Горы, д. 1, стр. 40)
E-mail: d.a.suplatov@belozersky.msu.ru, sharapova@belozersky.msu.ru, popova@cs.msu.su, kopylov@mail.chem.msu.ru, voevodin@parallel.ru, vytas@belozersky.msu.ru
Поступила в редакцию: 03.08.2018

Проведен сравнительный анализ вычислительной эффективности и масштабируемости молекулярной динамики (МД), реализованной в пакете AMBER, на реальных биологических системах с применением классического силового поля FF14SB с 4-центровой моделью воды TIP4P-Ew, а также нового многообещающего поля FF15IPQ с 3-центровой моделью воды SPC/E_b. Были использованы классические процессоры Intel Xeon E5-2697 v3, а также GPU ускорители Tesla K40 (архитектура Kepler) и P100 (Pascal). Уменьшение количества атомов в ячейке на 25–31 % в результате использования 3-центровой модели растворителя ускоряет расчет МД до 63% и ухудшает масштабируемость до 11 %. При этом полученные результаты могут качественно отличаться, что говорит о необходимости совместного использования разных силовых полей при изучении биологических систем. Использование GPU-ускорителей как альтернативы классическим CPU позволяет существенно увеличить длину траектории в повседневной практике.

Ключевые слова: классическая молекулярная динамика, AMBER, силовые поля FF14SB и FF15IPQ, модели воды TIP4P-Ew, TIP3P и SPC/E_b, GPU-ускорители Kepler и Pascal.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Суплатов Д.А., Шарапова Я.А., Попова Н.Н., Копылов К.Е., Воеводин Вл.В., Швядас В.К. Молекулярная динамика в силовом поле FF14SB в воде TIP4P-Ew, и в силовом поле FF15IPQ в воде SPC/E_b: сравнительный анализ на GPU и CPU // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2019. Т. 8, № 1. С. 71–88. DOI: 10.14529/cmse190105.

Введение

Молекулярная динамика — собирательное название семейства компьютерных алгоритмов для изучения структуры и функции биологических макромолекул, которые образуют один из ключевых методов современной биологии [1, 2]. Несмотря на ряд ограничений этого подхода не вызывает сомнений его полезность при решении широкого спектра научных задач — изучения стабильности, конформационных перестроек в белках, взаимодействия белков с лигандами, процесса формирования нативной пространственной структуры (фолдинга) белка и др. Роль молекулярной динамики в науке была окончательно признана в 2013 году присуждением Нобелевской премии Мартину

*Статья рекомендована к публикации программным комитетом Международной конференции «Суперкомпьютерные дни в России — 2018».

Карплусу, Майклу Левитту и Арье Уоршелу за «разработку многомерных моделей сложных химических систем».

Применение метода молекулярной динамики требует мощного вычислительного ресурса. Бурное развитие компьютерных технологий и совершенствование самого метода в последние годы способствовало его распространению и повседневному использованию в научной практике. Нужно отметить, что речь идет не столько об увеличении мощности компьютеров в целом, сколько о появлении специальных вычислительных устройств, ориентированных на решение конкретных научных задач. В 2008 году частная американская компания D. E. Shaw Research объявила о создании специализированного суперкомпьютера Anton, спроектированного на аппаратном уровне для решения единственной вычислительной задачи — расчета длинных молекулярно-динамических траекторий [3]. В 2014 году появилась улучшенная версия этой машины — Anton 2 [4]. С другой стороны, революцией стала разработка GPU-ускорителей (от англ. graphics processing unit) как эффективной альтернативы классическим процессорам при решении широкого спектра научных задач. Разработка и имплементация компьютерных алгоритмов для исполнения на GPU ускорителях, прежде всего ре-имплементация ранее предложенных классических CPU кодов полезных алгоритмов, постепенно становится преобладающей стратегией для ускорения сложных вычислений в компьютерной биологии. Эта тенденция, которая раньше продвигалась в основном разработчиками GPU-оборудования, в последние годы твердо закрепилась в фундаментальной биологической науке [5]. Преимущества GPU заключаются в сочетании аппаратной универсальности, высокой мощности вычислительного устройства и относительно невысокой стоимости. Современные видеокарты могут быть установлены даже в индивидуальные рабочие станции и предоставляют значительное ускорение расчетов в ежедневной работе. К настоящему моменту все основные методы молекулярного моделирования, в том числе наиболее распространенный подход классической молекулярной динамики в периодической системе с явно заданным растворителем, частично или полностью реализованы на GPU [5].

Потребность молекулярного моделирования в больших ресурсах имеет конкретные причины. При использовании молекулярной динамики важна не столько скорость получения результатов, сколько их «правильность», т.е. способность метода адекватно описывать реальные процессы, происходящие в живых системах на уровне отдельных атомов. Повышение точности расчетов может быть достигнуто благодаря установке температуры, оптимальной для работы белков; увеличению времени наблюдения за системой (длины траектории); использованию более сложной молекулярной модели воды, которая могла бы более правильно воспроизводить физико-химические характеристики растворителя (например, 4-центровой модели TIP4P-Ew вместо 3-центровой TIP3P); уменьшению шага интегрирования уравнения движения; усложнению электростатических расчетов (увеличению радиуса учета нековалентных взаимодействий и отключению псевдоструктур, которые представляют типовые химические блоки в упрощенном виде, например, редуцируют сопряженные многоатомные системы до нескольких псевдоатомов); увеличению отступа между поверхностью белка и краем молекулярной ячейки для исключения артефактов в периодической системе и др. Однако применение этих настроек приводит к существенному увеличению вычислительной сложности задачи. Поэтому для ускорения молекулярной динамики приходилось жертвовать точностью

вычислений. Так, использование экстремальных температур (100–1000°C) позволяло ускорить структурные изменения в белках и уменьшить время наблюдения, однако существенно повышало вероятность появления артефактов. Именно поэтому появление доступных GPU-ускорителей и совместимых приложений молекулярной динамики было встречено с таким энтузиазмом — они позволяют увеличить время наблюдения за биомакромолекулярной системой при адекватных параметрах алгоритма (в том числе, физиологических значений температуры).

Перед тем, как исследовать свой объект с использованием молекулярной динамики необходимо определиться по следующим ключевым вопросам: (1) выбрать программу (метод), (2) силовое поле и (3) модель растворителя для проведения вычислительного эксперимента. Конечно, есть и другие параметры, значения которых необходимо подбирать под конкретную задачу (например, размер ячейки с растворителем, о чем будет более подробно сказано ниже), однако, указанные три пункта будут иметь наиболее существенное влияние на вычислительную сложность эксперимента и биологическую интерпретацию его результатов.

В настоящий момент существует множество пакетов программ для молекулярной динамики, наиболее известными из которых являются AMBER, CHARMM, GROMACS и NAMD. Каждый из этих продуктов обладает своими преимуществами и существенно отличается от других — по набору доступных методов, параметров (в том числе силовых полей), средств анализа результатов, доступности исходного кода, и др. При том, что выбор программного продукта для молекулярной динамики диктуется конкретной задачей пакет AMBER [6, 7], возможно, является оптимальным с точки зрения сочетания доступности и функциональности для решения широкого круга задач. Пакет программ AMBER богат на реализации различных биологических алгоритмов — классической и ускоренной молекулярной динамики, метадинамики, молекулярной динамики в вакууме, неявно заданном растворителе, методов QM/MM (сочетание классической динамики и методов квантовой химии) и т.д. Некоторые из них доступны в двух версиях — для исполнения на GPU или CPU. Важным преимуществом AMBER является наличие дополнительного пакета AMBERTOOLS с широким набором силовых полей, моделей растворителя, а также различных программ для подготовки биологической системы к молекулярной динамике и анализа результатов. В частности, пользователям AMBER доступно программное обеспечение для «параметризации» новых молекул, которые отсутствуют в силовом поле по умолчанию (а это практически все низкомолекулярные соединения), например, субстратов ферментативных реакций или ингибиторов ключевых ферментов метаболизма патогенных бактерий, что значительно расширяет возможности метода молекулярной динамики для изучения живых систем.

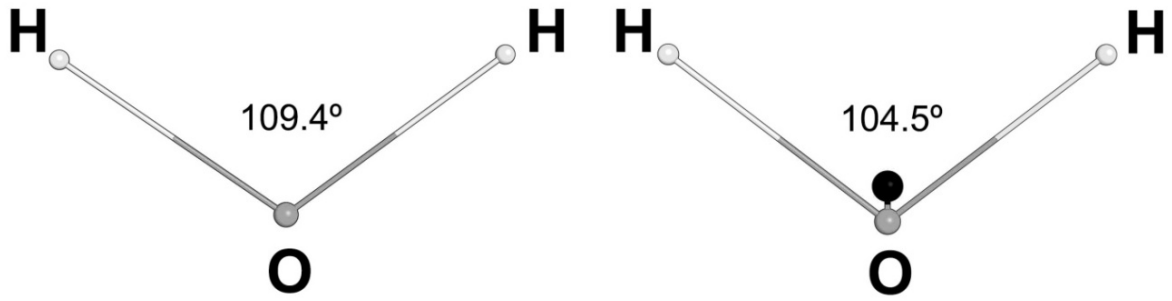
Силовые поля — это функции расчета энергии движения и взаимодействия атомов, а также наборы параметров для этих функций, характеризующих, в том числе, массу и заряд атомов, длины, углы и двугранные углы ковалентных взаимодействий атомов друг с другом, и т.д. Силовые поля AMBER предназначены для оригинальных программ из пакета AMBER и, пожалуй, являются наиболее известными и широко используемыми в повседневной практике. Нужно отметить, что силовые поля AMBER в ограниченном виде поддерживаются программными продуктами сторонних производителей — например, пакет GROMACS версии 2018 года поддерживает устаревшие силовые поля AMBER, разработанные до 2003 года (FF94, FF96, FF99, FF99SB, FF99SB-ILDN,

FF03 и FFGS) [8]. Силовые поля являются сложными аппроксимациями, предназначенными для описания динамических процессов в биологических системах, которые постоянно развиваются и обновляются для повышения точности расчетов. На момент написания статьи силовое поле FF14SB [9] является основным рекомендуемым полем для изучения динамики белков в пакете программ AMBER [10]. Для изучения биологических макромолекул небелковой природы (нуклеиновых кислот, липидов и т.д.) специально разработаны отдельные силовые поля, которые можно использовать в комбинации с полями для белков. Поле FF14SB является улучшенной (оптимизированной) версией силового поля FF99SB, полученного «классическим» способом. Процесс создания силовых полей заключается в выборе таких значений всех параметров, характеризующих сложную биологическую систему (зарядов, масс атомов, длин, углов и жесткости всех возможных связей и т.д.), которые позволяют получать результаты, в наибольшей степени согласующиеся с экспериментальными данными. Проблема в том, что таких параметров очень много. В этой связи, как правило, использовалась итеративная оптимизация только определенного подмножества параметров, например, ре-оптимизации параметров, характеризующих торсионные углы при заранее заданных и фиксированных зарядах атомов. Результат такого «классического» подхода, очевидно, зависит от точности неоптимизированных (фиксированных) параметров. Для преодоления этих ограничений были предложены новые подходы, позволяющие в полуавтоматическом режиме оптимизировать большое количество параметров силового поля. Одним из таких подходов является метод Implicitly Polarized Charge (IPolQ), с помощью которого было разработано поле FF14IPQ и его более новая версия FF15IPQ [11]. Разработчиками было отмечено, что FF15IPQ позволяет более точно моделировать ионные взаимодействия между противоположно заряженными аминокислотными остатками, вторичные структуры и глобулярные укладки белков, а также в целом предназначено для изучения биологических систем при расчете длинных траекторий. Поскольку одним из ключевых преимуществ современных GPU-ускорителей является возможность существенно увеличить время наблюдения за системой в повседневной практике, заявленные возможности силового поля FF15IPQ для изучения белков представляются интересными и актуальными.

Наконец, необходимо определиться со средой, в которой будет проведен эксперимент по моделированию. Она может отсутствовать, в таком случае расчет будет проводиться в вакууме. Такой подход используют при создании сложных моделей белков — «прижимая» две части структуры в вакууме можно индуцировать комплементарность между ними, чтобы реконструировать, например, интерфейс взаимодействия между доменами. Расчет молекулярной динамики без растворителя можно исполнить, не имея значительных вычислительных ресурсов, и его результаты могут быть полезны в отдельных случаях, однако для описания работы белков в физиологических условиях этот вариант, очевидно, не подходит. Существует отдельный класс методов для молекулярного моделирования в неявно заданном растворителе, которые применяют для ускорения расчетов при изучении очень больших биомолекулярных систем, например, белков в составе липидной мембраны, однако эти подходы имеют ряд существенных ограничений [12]. Наиболее распространенным подходом к изучению динамики биологических макромолекул является использование явно заданного растворителя, в роли которого, как правило, выступает вода. Растворитель играет важную роль в процессах, происходящих в живой клетке, поэтому модели воды для молекулярной динамики, как и сило-

вые поля, постоянно дорабатываются для повышения точности расчетов. Существует множество моделей воды, которые предназначены для использования в разных силовых полях, по-разному воспроизводят экспериментально изученные физико-химические свойства воды, и отличаются геометрией, зарядами, массами атомов, а также их количеством. Наиболее распространенными сегодня являются 3-центровые модели воды (пример приведен на рис. 1а). Так, вместе с силовым полем FF14SB чаще всего используют модель воды TIP3P, состоящую, как и настоящая вода, из трех атомов — кислорода (O) и двух атомов водорода (H). В попытке улучшить точность моделирования были предложены 4-центровые модели воды, содержащие один дополнительный псевдоатом. Для расчетов в периодической системе с явно заданным растворителем была разработана модель воды TIP4P-Ew (рис. 1б). Было показано, что она лучше описывает вращательное движение белков по сравнению с обычно используемой 3-центровой моделью TIP3P [13], а также более корректно воспроизводит физические свойства жидкого растворителя при физиологических значениях температуры и давления [14]. Добавление псевдоатомов в модель растворителя может помочь более точно оценить поведение белка в воде, однако существенно увеличивает вычислительную сложность эксперимента. В этом контексте представляет интерес 3-центровая модель воды SPC/E_b (рис. 1а), которая совместима с ранее упомянутым перспективным силовым полем FF15IPQ, а по своим характеристикам не уступает либо превосходит доступные аналоги [15]. Существуют также 2-, 5- и 6-центровые модели воды, однако на практике используются значительно реже.

В этой работе метод классической молекулярной динамики в периодической системе с явно заданным растворителем, реализованный в пакете AMBER, был применен для анализа реальных биологических систем — с применением классического силового поля FF14SB с 4-центровой моделью воды TIP4P-Ew, а также нового перспективного силового поля FF15IPQ с 3-центровой моделью воды SPC/E_b. Проведен сравнительный анализ результатов — как с точки зрения скорости и масштабируемости молекулярной динамики на GPU и CPU, так и их биологической интерпретации.



а) 3-центровая модель воды SPC/E_b

б) 4-центровая модель воды TIP4P-Ew. Псевдоатом показан черным цветом

Рис. 1. Расположение атомов в разных моделях воды. Приведено значение угла H-O-H

1. Методы

1.1. Биологические системы и размеры ячеек

Метод классической молекулярной динамики применяли к четырем белкам: MAPK — митоген-активируемая протеинкиназа человека (PDB код 1R3C); NanC — нейраминидаза C из *Streptococcus pneumoniae* (PDB код 4YW2); LDH — лактатдегидрогеназа из *Staphylococcus aureus* (PDB код 3D4P); NanA — нейраминидаза A из *Streptococcus pneumoniae* (PDB коды 2YA8 и 4ZXK). Перечисленные белки представляют интерес в качестве мишеней для разработки лекарств от различных заболеваний человека и существенно отличаются по размеру и геометрии. Так, NanA — модульный белок, состоящий из доменов, которые не образуют стабильной глобулярной структуры, а соединены между собой гибким междоменным линкером (рис. 2). Для динамики этой макромолекулы в периодической системе необходимо поместить ее в «ячейку» таких размеров, чтобы белок в процессе движения и конформационных перестроек не взаимодействовал со своей копией в соседней ячейке при заданном радиусе учета нековалентных взаимодействий (в этой работе — 10 Å). В случае с NanA, исходя из длины междоменного линкера, использовали ячейку с минимальным отступом между белком и краем ячейки в 30 Å (рис. 3а). В остальных случаях (MAPK, NanC, LDH) белки представляли собой устойчивые глобулярные структуры, что позволяло использовать стандартный отступ — не менее 12 Å (рис. 3б). Для каждого случая были подготовлены две модели белка в двух силовых полях – FF14SB и FF15IPQ. Ячейки заполняли водой двух типов — для поля FF14SB использовали модель воды TIP4P-Ew, для поля FF15IPQ использовали модель воды SPC/E_b. Размеры соответствующих биологических систем приведены в табл. 1. В колонке «изменение» указан коэффициент уменьшения количества атомов в системе с 3-центровой моделью воды SPC/E_b по сравнению с 4-центровой моделью воды TIP4P-Ew. «Количество атомов воды в ячейке» указано в процентах от общего количества атомов. Для контрольных экспериментов также были подготовлены ячейки с MAPK в силовом поле FF14SB, заполненные водой TIP3P и SPC/E_b, а также ячейки с LDH в силовом поле FF14SB, заполненные водой TIP3P и SPC/E_b. Можно добавить, что силовое поле FF15IPQ совместимо только с моделью воды SPC/E_b.

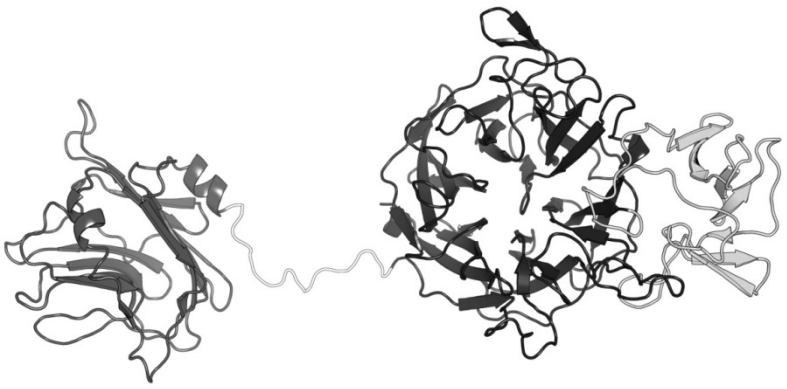
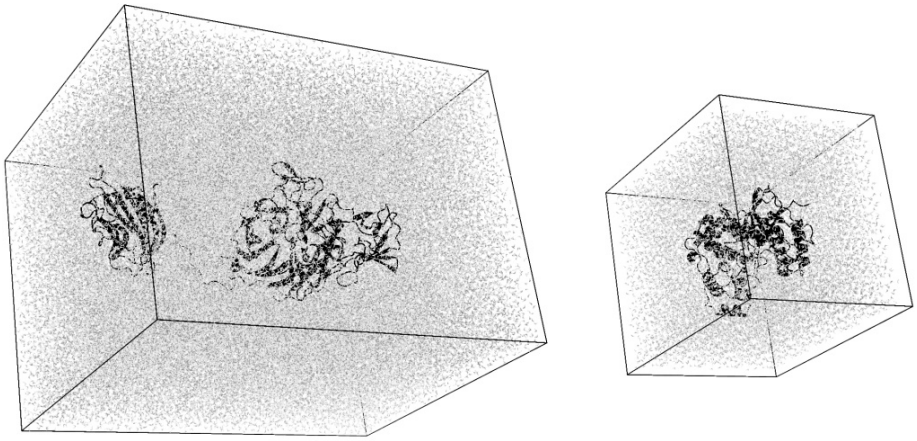


Рис. 2. Модель структуры нейраминидазы NanA из *Streptococcus pneumoniae* [18]. Отдельные домены и междоменный линкер окрашены в разные тона серого



а) NanA из *Streptococcus pneumoniae* б) p38α MAP киназа человека

Рис. 3. Ячейки биологических макромолекулярных систем в воде

Таблица 1

Размеры исследованных биологических систем

Название системы	**Количество атомов в ячейке**			**Количество атомов воды в ячейке**	
	FF14SB TIP4P-Ew	**FF15IPQ SPC/E_b**	**Изменение**	**FF14SB TIP4P-Ew**	**FF15IPQ SPC/E_b**
MAPK	65941	51884	×1,27	91,4%	89,0%
NanC	119709	93438	×1,28	91,3%	88,9%
LDH	140069	111932	×1,25	86,3%	82,8%
NanA	347145	265232	×1,31	96,9%	95,9%

1.2. Вычислительный эксперимент

Метод классической молекулярной динамики в периодической системе с явно заданным растворителем применяли в соответствии с ранее описанным протоколом [16–18] с использованием реализаций алгоритма в пакете AMBER для GPU и CPU. Для изучения скорости и масштабируемости на разном числе видеокарт одного и нескольких узлов задачи запускали на 60 минут в режиме GPU или CPU и оценивали среднюю скорость расчета траектории в нс/день. Для получения биологически значимых результатов системы изучали в течение длительного периода времени (от 100 до 1000 нс).

1.3. Оборудование

Минимизацию энергии, нагрев и уравновешивание систем выполняли на локальном GPU-ускорителе GeForce GTX 980 Ti, а собственно оценку скорости молекулярной динамики на стадии «свободной динамики» (т.е. без применения позиционных ограничений) выполняли на двух разделах — «Compute» и «Pascal» — уникальной научной установки суперкомпьютере «Ломоносов-2». Один узел раздела «Compute» оснащен одним процессором Intel Xeon E5-2697 v3 (14 физических ядер), 64 Гб оперативной памяти и одной видеокартой Tesla K40 (архитектура Kepler). Один узел раздела «Pascal» оснащен одним процессором Intel Xeon Gold 6126 (12 физических ядер), 96 Гб оперативной памяти и двумя видеокартами Tesla P100 (архитектура Pascal). Основная сеть и сеть I/O — Infiniband FDR.

1.4. Программное обеспечение

Для расчета классической молекулярной динамики использовали программы пакета AMBER версий 14 и 16. Программы версии 14 использовали для вычислительного эксперимента на CPU и GPU раздела «Compute» суперкомпьютера «Ломоносов-2» (видеокарты Tesla K40 архитектуры Kepler), а программы версии 16 исполняли на GPU ускорителях раздела «Pascal» суперкомпьютера «Ломоносов-2» (видеокарты Tesla P100 архитектуры Pascal). Сборку программ осуществляли из исходного кода с использованием следующих компиляторов и библиотек: Intel icc/icpc/ifort версии 15.0.3, MKL версии 11.1.3, CUDA версий 6.5 (для сборки AMBER версии 14) и 8.0 (для сборки AMBER версии 16), а также OpenMPI версий 1.10.7 и 2.1.1.

2. Результаты

2.1. Сравнение скорости и масштабируемости

Скорость вычислений в силовом поле FF15IPQ с 3-центровой моделью воды SPC/E_b во всех случаях была существенно выше, чем в силовом поле FF14SB с 4-центровой моделью воды TIP4P-Ew. При использовании GPU-ускоренной молекулярной динамики на видеокартах Tesla K40 (архитектура Kepler) уменьшение количества атомов в ячейке на 25–31 % в результате использования FF15IPQ+SPC/E_b вместо FF14SB+TIP4P-Ew ускоряет расчет в ×1,25–×1,63 раза в зависимости от размера биологической системы и количества использованных узлов (табл. 2, рис. 4). Наилучшее ускорение в ×1,44–×1,63 раза наблюдалось для системы NanA, самой большой как по общему количеству атомов, так и по процентному содержанию воды; наихудшее — в ×1,25–×1,43 раза — для самой маленькой системы MAPK. Для всех изученных систем наилучшее ускорение в результате использования FF15IPQ+SPC/E_b вместо FF14SB+TIP4P-Ew наблюдалось при запуске на двух узлах — в ×1,43–×1,63 раза, наихудшее — в ×1,25–×1,44 раза при запуске на восьми узлах. Можно также в целом отметить ухудшение масштабируемости вычислений при использовании FF15IPQ+SPC/E_b вместо FF14SB+TIP4P-Ew — до 11 % (табл. 3). Эффективность работы GPU-имплементации молекулярной динамики, собранной с использованием OpenMPI версий 1.10.7 и 2.1.1, различалась незначительно.

Результаты, полученные при использовании молекулярной динамики на GPU (Tesla K40) и CPU (Intel Xeon E5-2697 v3) узлах раздела «Compute», существенно различались по двум показателям — абсолютным значениям скорости, а также масштабируемости вычислений на большом числе узлов. При использовании только классических процессоров ускорение расчетов в результате использования FF15IPQ+SPC/E_b было эквивалентно значениям, полученным для GPU — до ×1,49 раза. Однако скорость на GPU была существенно выше скорости на CPU — до 67 нс/день и 21 нс/день, соответственно. Максимальная масштабируемость в большинстве случаев была достигнута на 4 узлах (рис. 4 и 5, табл. 3). Однако для систем с процентным содержанием воды в ячейке более 91,3 % (MAPK, NanC, и NanA в FF14SB+TIP4P-Ew, а также NanA в FF15IPQ+SPC/E_b) производительность молекулярной динамики продолжала расти даже на большем числе узлов. Эффективность работы приложения, собранного разными версиями OpenMPI на 6–8 узлах при использовании только CPU принципиально различалась — масштабируемость сборки MPI компилятором версии 2.1.1 деградировала, в

Таблица 2

Скорость расчета МД на GPU ускорителях Tesla K40 (в нс/день)

Название системы	N1			N2		
	FF14SB TIP4P-Ew	**FF15IPQ SPC/E_b**	**Ускорение**	**FF14SB TIP4P-Ew**	**FF15IPQ SPC/E_b**	**Ускорение**
MAPK	30,54	41,95	×1,37	33,89	48,43	×1,43
NanC	16,56	23,61	×1,43	19,17	28,26	×1,47
LDH	13,31	18,85	×1,42	17,13	24,42	×1,43
NanA	5,75	8,87	×1,54	6,86	11,16	×1,63
Название системы	**N4**			**N6**		
	FF14SB TIP4P-Ew	**FF15IPQ SPC/E_b**	**Ускорение**	**FF14SB TIP4P-Ew**	**FF15IPQ SPC/E_b**	**Ускорение**
MAPK	48,91	67,26	×1,38	50,07	63,72	×1,28
NanC	26,80	39,15	×1,46	27,95	38,19	×1,37
LDH	23,64	30,30	×1,28	23,62	30,95	×1,31
NanA	9,21	14,38	×1,56	9,96	15,27	×1,53

Название системы	N8		
	FF14SB TIP4P-Ew	**FF15IPQ SPC/E_b**	**Ускорение**
MAPK	51,67	64,52	×1,25
NanC	30,09	38,61	×1,28
LDH	23,44	30,40	×1,30
NanA	10,39	14,97	×1,44

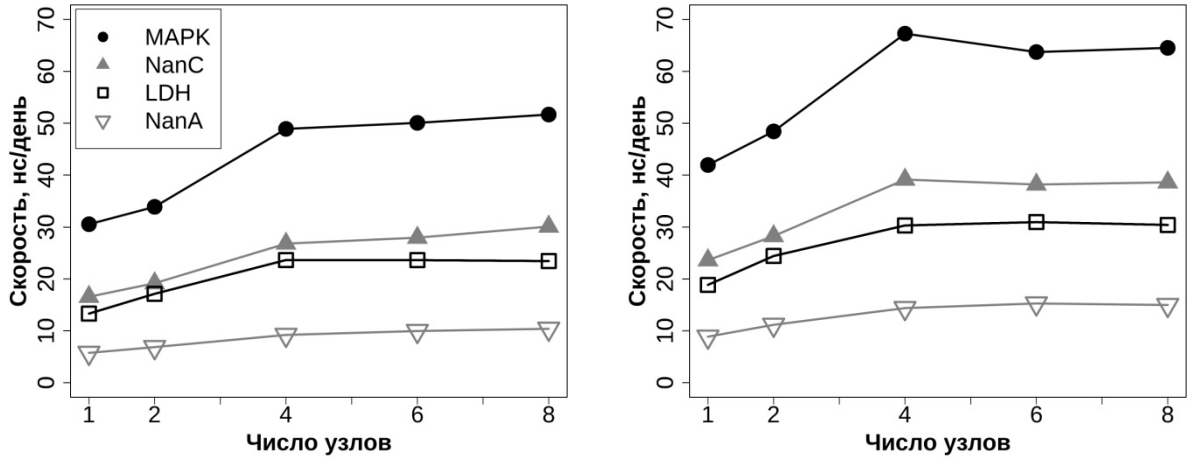
Таблица 3

Ускорение в параллельном режиме на GPU Tesla K40

Название системы		**GPU**			
		N2	N4	N6	N8
MAPK	FF14SB + TIP4P-Ew	1,11	1,60	1,64	1,69
	FF15IPQ + SPC/E_b	1,15	1,60	1,52	1,54
NanC	FF14SB + TIP4P-Ew	1,16	1,62	1,69	1,82
	FF15IPQ + SPC/E_b	1,20	1,66	1,62	1,64
LDH	FF14SB + TIP4P-Ew	1,29	1,78	1,77	1,76
	FF15IPQ + SPC/E_b	1,30	1,61	1,64	1,61
NanA	FF14SB + TIP4P-Ew	1,19	1,60	1,73	1,81
	FF15IPQ + SPC/E_b	1,26	1,62	1,72	1,69

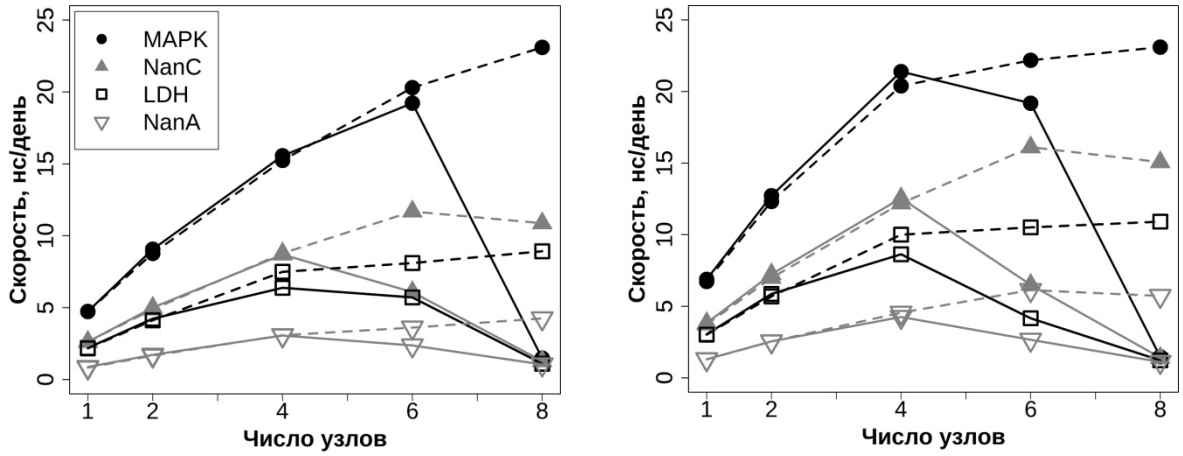
то время как производительность сборки MPI компилятором версии 1.10.7 в некоторых случаях продолжала расти (рис. 5).

Отдельно было проведено изучение вычислительной эффективности и масштабируемости молекулярной динамики на видеокартах Tesla P100 (GPU-архитектура Pascal), которые появились на суперкомпьютере Ломоносов-2 после обновления в марте 2018 года (раздел «Pascal»). Вычислительная мощность одной Tesla P100 существенно выше, чем у Tesla K40, что принципиально сказывается на производительности и масштабируемости молекулярной динамики. Скорость расчета на двух видеокартах Tesla P100 од-



а) FF14SB+TIP4P-Ew б) FF15IPQ+SPC/E_b

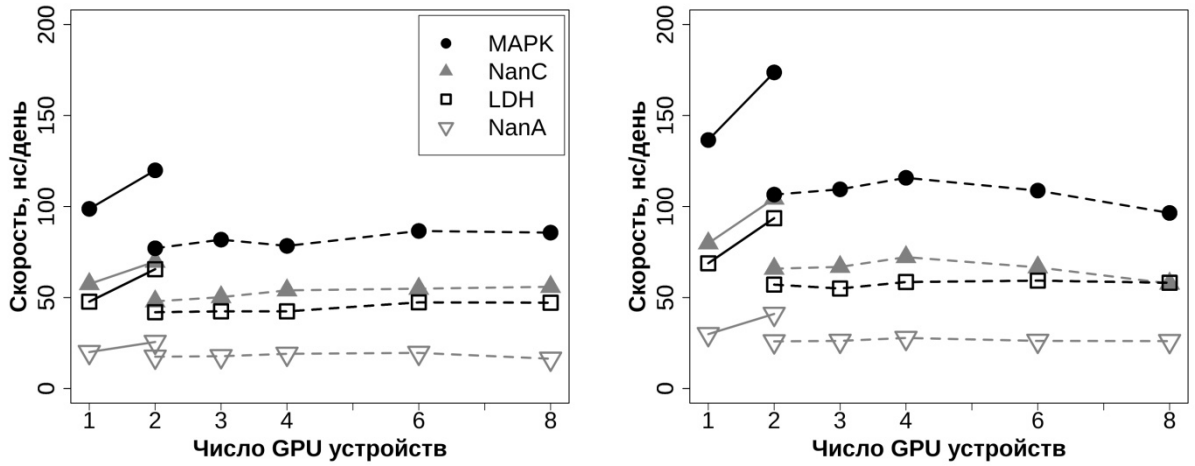
Рис. 4. Скорость расчета молекулярной динамики на GPU Tesla K40 (один узел — одна видеокарта)



а) FF14SB+TIP4P-Ew б) FF15IPQ+SPC/E_b

Рис. 5. Скорость расчета молекулярной динамики на CPU. Сплошной линией показаны результаты работы приложения, собранного с использованием OpenMPI версии 2.1.1, пунктирной линией — OpenMPI версий 1.10.7

ного узла раздела «Pascal» (содержит две видеокарты на каждом узле) была существенно выше лучшей скорости расчетов, полученной на 4–8 узлах с Tesla K40 раздела «Compute» (содержит одну видеокарту на каждом узле) — в ×2,32–×2,77 и ×2,57–×3,02 для набора параметров FF14SB+TIP4P-Ew и FF15IPQ+SPC/E_b, соответственно (рис. 6). Ускорение на двух Tesla P100 одного узла по сравнению с одной Tesla P100 составило ×1,21–×1,37, однако производительность на двух и более узлах во всех случаях была существенно ниже производительности на одной видеокарте, что объясняется различиями в механизме обмена данными между устройствами, установленными в одном и разных узлах. Две видеокарты одного узла способны обмениваться данными в режиме peer-to-peer, т.е. непосредственно через соединение PCI-E минуя центральный процессор, что снижает время ожидания и в итоге приводит к ускорению. Однако вычислительная мощность Tesla P100 настолько велика, что обмен данными становится скорость-лимитирующей стадией расчета при совместном использовании видеокарт на разных узлах, что делает невозможным дальнейшее ускорение. Ускорение на Tesla P100 при ис-



а) FF14SB+TIP4P-Ew б) FF15IPQ+SPC/E_b

Рис. 6. Скорость расчета молекулярной динамики на GPU Tesla P100 (один узел — две видеокарты). Сплошной линией показаны результаты запуска приложения на одном или двух GPU ускорителях одного узла (в режиме peer-to-peer), пунктирной линией — на двух и более GPU ускорителях двух и более узлов

пользовании FF15IPQ+SPC/E_b вместо FF14SB+TIP4P-Ew были эквивалентны значениям, полученным на Tesla K40 — ×1,38–×1,49 и ×1,43–×1,60 на одной и двух видеокартах одного узла, соответственно. Эффективность работы на Tesla P100 приложения, собранного с использованием OpenMPI версий 1.10.7 и 2.1.1, различалась незначительно.

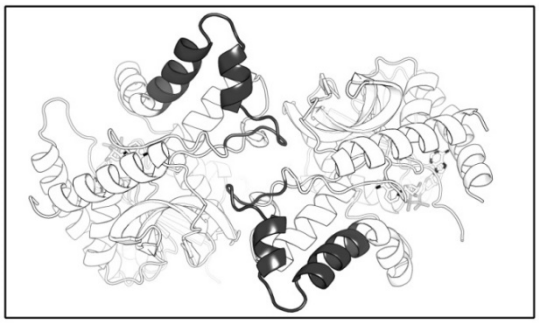
Для изучения влияния конкретной модели растворителя на вычислительную эффективность молекулярной динамики была дополнительно проведена серия контрольных экспериментов с системой MAPK, параметризованной в силовом поле FF14SB с разными 3-центровыми моделями растворителя. Сравнение показало, что скорость и масштабируемость не зависели от конкретного силового поля или модели воды, а определялись количеством центров в растворителе.

2.2. Биологическая интерпретация результатов

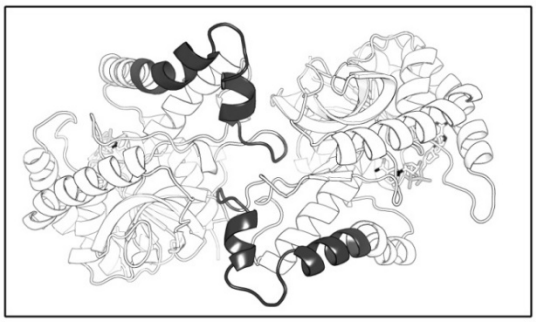
Результаты молекулярной динамики для MAPK, NanC и NanA в FF15IPQ+SPC/E_b и FF14SB+TIP4P-Ew были эквивалентны, однако при использовании первого набора параметров систематически наблюдалось более значительное среднеквадратичное отклонение (RMSD) структуры белка от первоначального положения в процессе длительной симуляции. Более подробно интерпретация результатов моделирования этих белков с биологической точки зрения изложена в отдельных публикациях [17, 18].

Существенные различия были выявлены при моделировании системы LDH — лактатдегидрогеназы из *Staphylococcus aureus*. В структурах многих бактериальных лактатдегидрогеназ (например, из *Bifidobacterium longum*, *Bacillus subtilis*, *Geobacillus stearothermophilus*) на границе между субъединицами гомотетрамера есть аллостерический сайт, в котором связывается аллостерический активатор фруктозо-1,6-дифосфат (FBP). Считается, что эти ферменты существуют в растворе в виде ансамбля конформеров с разной функциональностью и соотношение между этими структурными формами зависит от концентрации аллостерического эффектора. Для многих лактатдегидрогеназ присутствие FBP критически необходимо для сборки четвертичной структуры фермента. Однако существуют и неаллостерические прокариотические лак-

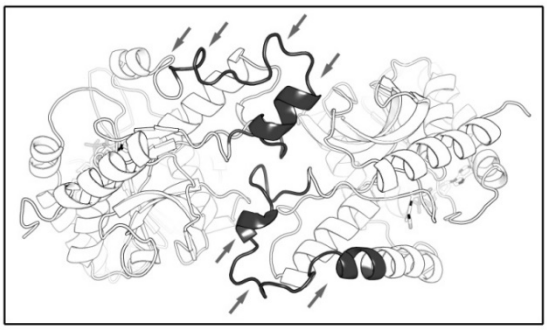
татдегидрогеназы, для которых роль FBP не описана, например, фермент из *Lactobacillus pentosus*, в котором стабилизация интерфейса между субъединицами достигается за счет сети солевых мостиков. Аллостерия в лактатдегидрогеназе из *Staphylococcus aureus* (объекте этого исследования) пока не описана и споры по этому поводу продолжаются [19]. Молекулярное моделирование этого фермента в FF15IPQ+SPC/E_b выявило конформационную перестройку участка структуры, содержащего остатки 193-218, в течение 100 нс симуляции при температуре 300 K (рис. 7). Использование набора параметров FF14SB+TIP4P-Ew, а также FF14SB+TIP3P и FF14SB+SPC/E_b, не выявило никаких структурных изменений в этой области белка в идентичных условиях, что в целом согласуется с представлением о том, что классическим силовым полям AMBER, в том числе полю FF14SB, свойственна переоценка некоторых типов взаимодействий, в том числе возникающих в элементах вторичной структуры, а также солевых мостиках, приводящая к артефактам стабилизации структуры [2, 11]. Участок 193-218 в структуре лактатдегидрогеназы из *Staphylococcus aureus* расположен на границе между субъединицами гомотетрамера и входит в состав аллостерического центра связывания FBP в родственных лактатдегидрогеназах. Роль этого участка и его конформационных перестроек в структуре и функции фермента требует дальнейшего изучения, однако полученный результат моделирования представляет интерес в свете все более активного в последние годы изучения аллостерии и выявления новых путей регуляции в ферментах, для которых аллостерия еще не описана [20].



а) Кристаллографическая структура LDH из банка данных PDB (код 3D4P)



б) Структура LDH через 100 нс молекулярной динамики в силовом поле FF14SB в воде TIP4P-Ew



в) Структура LDH через 100 нс молекулярной динамики при температуре 300 K в силовом поле FF15IPQ в воде SPC/E_b

Рис. 7. Существенные различия результатов молекулярной динамики структуры лактатдегидрогеназы из *Staphylococcus aureus* (LDH) в двух силовых полях. Участок структуры, содержащий остатки 193-218, показан черным цветом

Заключение

Использованные в работе силовые поля FF14SB/FF15IPQ и модели воды TIP4P-Ew/TIP3P/«SPC/E_b» представляют собой разные наборы параметров, однако уравнения, в которые значения этих параметров подставляются для вычисления энергии взаимодействия атомов и возникающего движения частиц, идентичны. Отличия в вычислительной эффективности молекулярной динамики с разными силовыми полями и моделями воды, в принципе, могут быть связаны с отличиями в геометрии, как самого растворителя, так и белка. Средние значения и жесткость углов/длин связей отличаются в разных моделях воды и силовых полях, что может сказаться на самой вычислительно сложной части алгоритма — расчете нековалентных взаимодействий каждого атома с другими атомами, расположенными внутри заданного радиуса. Однако геометрия силовых полей и моделей воды различается не до такой степени, чтобы существенно повлиять на скорость. Поэтому полученные в работе результаты ускорения и масштабируемости при использовании FF15IPQ+SPC/E_b вместо FF14SB+TIP4P-Ew связаны, прежде всего, с разным количеством атомов в моделях растворителя. Использование 3-центровой модели растворителя вместо 4-центровой модели (т.е. удаление одного псевдоатома из каждой молекулы воды в ячейке) приводит к уменьшению количества атомов в одинаковом объеме — изменяется как количество атомов в ячейке, так и плотность атомов (количество атомов в заданном радиусе), что существенно сказывается на расчете нековалентных взаимодействий и позволяет достичь ускорения до $\times$1,63 раз на GPU и до $\times$1,49 раз на CPU. Кроме того, в некоторых случаях использование нового перспективного силового поля FF15IPQ с единственной совместимой с ним моделью воды SPC/E_b позволяет не только существенно ускорить расчет, но и получать качественно иные результаты, более интересные с точки зрения изучения конформационной подвижности в структурах белков. Однако не следует забывать, что поля серии IPQ (в том числе FF15IPQ) все еще находятся в процессе разработки и развития, в то время как FF14SB — основное рекомендованное поле для молекулярной динамики белков с помощью программ пакета AMBER. Таким образом, полученные результаты говорят о необходимости совместного использования разных силовых полей при изучении биологических систем для того, чтобы оценить зависимость полученных результатов от использованных параметров.

Расчет на видеокартах позволяет существенно увеличить длину траектории в повседневной практике — в среднем, с десятков наносекунд до нескольких микросекунд — при сохранении адекватных настроек метода. Это означает, что для изучения даже больших и сложно организованных белков, таких как нейраминидаза А из *Streptococcus pneumoniae* (до 350 000 атомов в ячейке с водой) больше не нужно жертвовать качеством в угоду скорости. Уникальная научная установка суперкомпьютер «Ломоносов-2», оснащенная мощными GPU-вычислителями Tesla K40 (архитектура Kepler) и Tesla P100 (архитектура Pascal), открывает принципиально новые горизонты для научной работы в области структурной биологии и биоинформатики.

Работа выполнена при поддержке грантов РФФИ №17-07-00751 и РНФ №15-14-00069-П с использованием оборудования Центра коллективного пользования сверхвысокопроизводительными вычислительными ресурсами МГУ имени М.В. Ломоносова [21].

Литература

1. Суплатов Д.А., Попова Н.Н., Копылов К.Е., Шегай М.В., Воеводин Вл.В., Швядас В.К. Гибридные вычислительные кластеры для изучения структуры, функции и регуляции белков // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2017. Т. 6, № 4. С. 74–90. DOI: 10.14529/cmse170406.
2. Godwin R.C., Melvin R., Salsbury F.R. Molecular Dynamics Simulations and Computer-Aided Drug Discovery «Computer-Aided Drug Discovery» (Wei Zhang) Springer, New York. 2016. P. 1–31. DOI: 10.1007/7653_2015_41.
3. Shaw D.E. et al. Anton, a Special-Purpose Machine for Molecular Dynamics Simulation // Communications of the ACM. July 2008. Vol. 51, No. 7. P. 91–97. DOI: 10.1145/1250662.1250664.
4. Shaw D.E. et al. Anton 2: Raising the Bar for Performance and Programmability in a Special-Purpose Molecular Dynamics Supercomputer // International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14): Proceedings of the International Conference (New Orleans, LA, USA, November 16–21, 2014). IEEE Press. 2014. pp. 41–53. DOI: 10.1109/SC.2014.9.
5. Nobile M.S. et al. Graphics Processing Units in Bioinformatics, Computational Biology and Systems Biology // Briefings in Bioinformatics. 2016. Vol. 18, No. 5. P. 870–885. DOI: 10.1093/bib/bbw058.
6. Salomon-Ferrer R. et al. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald // Journal of Chemical Theory and Computation. 2013. Vol. 9, No. 9. P. 3878–3888. DOI: 10.1021/ct400314y.
7. Götz A.W. et al. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 1. Generalized Born // Journal of Chemical Theory and Computation. 2012. Vol. 8, No. 5. P. 1542–1555. DOI: 10.1021/ct200909j.
8. Документация к программному обеспечению GROMACS. URL: http://manual.gromacs.org/documentation/2018/user-guide/force-fields.html (дата обращения: 25.07.2018).
9. Maier J.A. et al. FF14SB: Improving the Accuracy of Protein Side Chain and Backbone Parameters from FF99SB // Journal of Chemical Theory and Computation. 2015. Vol. 11, No. 8. P. 3696–3713. DOI: 10.1021/acs.jctc.5b00255.
10. Документация к программному обеспечению AMBER. URL: http://ambermd.org/doc12/Amber17.pdf, P. 33 (дата обращения: 25.07.2018).
11. Debiec K.T. et al. Further Along the Road Less Traveled: AMBER FF15ipq, an Original Protein Force Field Built on a Self-Consistent Physical Model // Journal of Chemical Theory and Computation. 2016. Vol. 12, No. 8. P. 3926–3947. DOI: 10.1021/acs.jctc.6b00567.
12. Onufriev A. Implicit Solvent Models in Molecular Dynamics Simulations: A Brief Overview // Annual Reports in Computational Chemistry. 2008. Vol. 4. P. 125–137. DOI: 10.1016/S1574-1400(08)00007-8.
13. Wong V., Case D.A. Evaluating Rotational Diffusion From Protein MD Simulations // The Journal of Physical Chemistry B. 2008. Vol. 112, No. 19. P. 6013–6024. DOI: 10.1021/jp0761564.

14. Horn H.W. et al. Development of an Improved Four-Site Water Model for Biomolecular Simulations: TIP4P-Ew // The Journal of Chemical Physics. 2004. Vol. 120, No. 20. P. 9665–9678. DOI: 10.1063/1.1683075.
15. Takemura K., Kitao A. Water Model Tuning for Improved Reproduction of Rotational Diffusion and NMR Spectral Density // The Journal of Physical Chemistry B. 2012. Vol. 116, No. 22. P. 6279–6287. DOI: 10.1021/jp301100g.
16. Pierce L. et al. Routine Access to Millisecond Time Scale Events with Accelerated Molecular Dynamics // Journal of Chemical Theory and Computation. 2012. Vol. 8, No. 9. P. 2997–3002. DOI: 10.1021/ct300284c.
17. Suplatov D., Kopylov K., Sharapova Y., Švedas V. Human p38α Mitogen-Activated Protein Kinase in the Asp168-Phe169-Gly170-in (DFG-in) State Can Bind Allosteric Inhibitor Doramapimod // Journal of Biomolecular Structure and Dynamics. 2018. DOI: 10.1080/07391102.2018.1475260.
18. Sharapova Y., Suplatov D., Švedas V. Neuraminidase A from Streptococcus Pneumoniae Has a Modular Organization of Catalytic and Lectin Domains Separated by a Flexible Linker // The FEBS Journal. 2018. Vol. 285, No. 13. P. 2428–2445. DOI: 10.1111/febs.14486.
19. Crooke A.K. et al. CcpA-Independent Glucose Regulation of Lactate Dehydrogenase 1 in Staphylococcus Aureus // PLoS One. 2013. Vol. 8, No. 1. P. e54293. DOI: 10.1371/journal.pone.0054293.
20. Суплатов Д.А., Швядас В.К. Изучение функциональных и аллостерических сайтов в суперсемействах белков // Acta Naturae. 2015. Т. 7, №. 4. С. 39–52.
21. Воеводин Вл.В., Жуматий С.А., Соболев С.И., Антонов А.С., Брызгалов П.А., Никитенко Д.А., Стефанов К., Воеводин В.В. Практика суперкомпьютера «Ломоносов» // Открытые системы. 2012. Т. 7, С. 36–39.

Суплатов Дмитрий Андреевич, к.х.н., с.н.с. НИИ физико-химической биологии имени А.Н. Белозерского, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

Шарапова Яна Александровна, аспирант факультета биоинженерии и биоинформатики, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

Попова Нина Николаевна, к.ф-м.н., доцент факультета вычислительной математики и кибернетики, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

Копылов Кирилл Евгеньевич, аспирант факультета биоинженерии и биоинформатики, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

Воеводин Владимир Валентинович, чл.-корр. РАН, д.ф.-м.н., зав. кафедрой суперкомпьютеров и квантовой информатики факультета вычислительной математики и кибернетики, зам. директора Научно-исследовательского вычислительного центра, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

Швядас Витас Каятоно, д.х.н., профессор факультета биоинженерии и биоинформатики и НИИ физико-химической биологии имени А.Н. Белозерского, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

DOI: 10.14529/cmse190105

MOLECULAR DYNAMICS IN THE FORCE FIELD FF14SB IN WATER TIP4P-EW, AND IN THE FORCE FIELD FF15IPQ IN WATER SPC/E_B: A COMPARATIVE ANALYSIS ON GPU AND CPU

© 2019 D.A. Suplatov, Ya.A. Sharapova, N.N. Popova, K.E. Kopylov, Vl.V. Voevodin, V.K. Švedas

Lomonosov Moscow State University

(Leninskye gory 1-40, Moscow, 119991 Russia)

E-mail: d.a.suplatov@belozersky.msu.ru, sharapova@belozersky.msu.ru, popova@cs.msu.su, kopylov@mail.chem.msu.ru, voevodin@parallel.ru, vytas@belozersky.msu.ru

Received: 03.08.2018

A comparative analysis of computational efficiency and scalability of molecular dynamics (MD) implemented in the AMBER package was carried out on real biological systems using the classical force field FF14SB with the 4-site water model TIP4P-Ew, as well as the new promising force field FF15IPQ with the 3-site water model SPC/E_b. The Intel Xeon E5-2697 v3 processors, as well as GPU accelerators Tesla K40 (Kepler architecture) and P100 (Pascal) were used. Reduction of the number of atoms in a cell by 25–31 % as a result of implementing a 3-site solvent model speeds up the MD calculations by up to 63 % and decreases scalability by about 11 %. The obtained results can be qualitatively different, what indicates the need for joint use of different force fields at studying biological systems. The use of GPU-accelerators as an alternative to classical CPUs provides an opportunity to significantly increase the length on MD trajectories in the daily laboratory practice.

Keywords: classical molecular dynamics, AMBER, force fields FF14SB and FF15IPQ, water models TIP4P-Ew, TIP3P and SPC/E_b, GPU-accelerators Kepler and Pascal.

FOR CITATION

Suplatov D.A., Sharapova Y.A., Popova N.N., Kopylov K.E., Voevodin Vl.V., Švedas V.K. Molecular Dynamics in the Force Field FF14SB in Water TIP4P-Ew, and in the Force Field FF15IPQ in Water SPC/E_b: a Comparative Analysis on GPU and CPU. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2019. vol. 8, no. 1. pp. 71–88. (in Russian) DOI: 10.14529/cmse190105.

References

1. Suplatov D.A., Popova N.N., Kopylov K.E., Shegay M.V., Voevodin Vl.V., Švedas V.K. Hybrid Computing Clusters to Study Protein Structure, Function and Regulation. *Vestnik Yuzho-Uralskogo gosudarstvennogo universiteta. Seriya "Matematicheskoe modelirovanie i programmirovanie"* [Bulletin of South Ural State University. Series: Mathematical Modeling, Programming & Computer Software]. 2017. vol. 6, no. 4. pp. 74–90. (in Russian) DOI: 10.14529/cmse170406.

2. Godwin R.C., Melvin R., Salsbury F.R. *Molecular Dynamics Simulations and Computer-Aided Drug Discovery* "Computer-Aided Drug Discovery" (Wei Zhang) Springer, New York. 2016. pp. 1–31. DOI: 10.1007/7653_2015_41.
3. Shaw D.E. et al. Anton, a Special-Purpose Machine for Molecular Dynamics Simulation. *Communications of the ACM*. July 2008. vol. 51, no. 7. pp. 91–97. DOI: 10.1145/1250662.1250664.
4. Shaw D.E. et al. Anton 2: Raising the Bar for Performance and Programmability in a Special-Purpose Molecular Dynamics Supercomputer. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14): Proceedings of the International Conference (New Orleans, LA, USA, November 16-21, 2014)*. IEEE Press. 2014. pp. 41–53. DOI: 10.1109/SC.2014.9.
5. Nobile M.S. et al. Graphics Processing Units in Bioinformatics, Computational Biology and Systems Biology. *Briefings in Bioinformatics*. 2016. vol. 18, no. 5. pp. 870–885. DOI: 10.1093/bib/bbw058.
6. Salomon-Ferrer R. et al. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald. *Journal of Chemical Theory and Computation*. 2013. vol. 9, no. 9. pp. 3878–3888. DOI: 10.1021/ct400314y.
7. Götz A.W. et al. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 1. Generalized Born. *Journal of Chemical Theory and Computation*. 2012. vol. 8, no. 5. pp. 1542–1555. DOI: 10.1021/ct200909j.
8. *Dokumentacija k programmnomu obespecheniju GROMACS* [GROMACS Software Manual]. Available at: http://manual.gromacs.org/documentation/2018/user-guide/force-fields.html (accessed: 25.07.2018).
9. Maier J.A. et al. FF14SB: Improving the Accuracy of Protein Side Chain and Backbone Parameters from FF99SB. *Journal of Chemical Theory and Computation*. 2015. vol. 11, no. 8. pp. 3696–3713. DOI: 10.1021/acs.jctc.5b00255.
10. *Dokumentacija k programmnomu obespecheniju AMBER* [AMBER Software Manual]. Available at: http://ambermd.org/doc12/Amber17.pdf, P. 33 (accessed: 25.07.2018).
11. Debiec K.T. et al. Further Along the Road Less Traveled: AMBER FF15ipq, an Original Protein Force Field Built on a Self-Consistent Physical Model. *Journal of Chemical Theory and Computation*. 2016. vol. 12, no. 8. pp. 3926–3947. DOI: 10.1021/acs.jctc.6b00567.
12. Onufriev A. Implicit Solvent Models in Molecular Dynamics Simulations: A Brief Overview. *Annual Reports in Computational Chemistry*. 2008. vol. 4. pp. 125–137. DOI: 10.1016/S1574-1400(08)00007-8.
13. Wong V., Case D.A. Evaluating Rotational Diffusion from Protein MD Simulations. The *Journal of Physical Chemistry B*. 2008. vol. 112, no. 19. pp. 6013–6024. DOI: 10.1021/jp0761564.
14. Horn H.W. et al. Development of an Improved Four-Site Water Model for Biomolecular Simulations: TIP4P-Ew. *The Journal of Chemical Physics*. 2004. vol. 120, no. 20. pp. 9665–9678. DOI: 10.1063/1.1683075.
15. Takemura K., Kitao A. Water Model Tuning for Improved Reproduction of Rotational Diffusion and NMR Spectral Density. *The Journal of Physical Chemistry B*. 2012. vol. 116, no. 22. pp. 6279–6287. DOI: 10.1021/jp301100g.

16. Pierce L. et al. Routine Access to Millisecond Time Scale Events with Accelerated Molecular Dynamics. *Journal of Chemical Theory and Computation*. 2012. vol. 8, no. 9. pp. 2997–3002. DOI: 10.1021/ct300284c.
17. Suplatov D., Kopylov K., Sharapova Y., Švedas V. Human p38α Mitogen-Activated Protein Kinase in the Asp168-Phe169-Gly170-in (DFG-in) State Can Bind Allosteric Inhibitor Doramapimod. *Journal of Biomolecular Structure and Dynamics*. 2018. DOI: 10.1080/07391102.2018.1475260.
18. Sharapova Y., Suplatov D., Švedas V. Neuraminidase A from Streptococcus Pneumoniae Has a Modular Organization of Catalytic and Lectin Domains Separated by a Flexible Linker. *The FEBS Journal*. 2018. vol. 285, no. 13. pp. 2428–2445. DOI: 10.1111/febs.14486.
19. Crooke A.K. et al. CcpA-Independent Glucose Regulation of Lactate Dehydrogenase 1 in Staphylococcus Aureus. *PLoS One*. 2013. vol. 8, no. 1. pp. e54293. DOI: 10.1371/journal.pone.0054293.
20. Suplatov D., Švedas V. Study of Functional and Allosteric Sites in Protein Superfamilies. *Acta Naturae*. 2015. vol. 7, no. 4, pp. 34–45.
21. Voevodin Vl.V., Zhumatiy S.A., Sobolev S.I., Antonov A.S., Bryzgalov P.A., Nikitenko D.A., Stefanov K.S., Voevodin Vad.V. Practice of the "Lomonosov" Supercomputer. *Otkrytye Sistemy* [Open Systems]. 2012. vol. 7, pp. 36–39 (in Russian).

УДК 004,432 **DOI: 10.14529/cmse190106**

РЕШЕНИЕ ПРИКЛАДНЫХ ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ DVM-СИСТЕМЫ*

© 2019 В.А. Бахтин, Д.А. Захаров, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула

ИПМ им. М.В. Келдыша РАН

(125047 Москва, Миусская пл., д. 4)

E-mail: bakhtin@keldysh.ru, s123-93@mail.ru, alexander.k.s@mail.ru, krukov@keldysh.ru, konov@keldysh.ru, pritula@keldysh.ru

Поступила в редакцию: 21.06.2018

DVM-система предназначена для разработки параллельных программ научно-технических расчетов на языках C-DVMH и Fortran-DVMH. Эти языки используют единую модель параллельного программирования (DVMH-модель) и являются расширением стандартных языков Си и Фортран спецификациями параллелизма, оформленными в виде директив компилятору. DVMH-модель позволяет создавать эффективные параллельные программы для гетерогенных вычислительных кластеров, в узлах которых в качестве вычислительных устройств наряду с универсальными многоядерными процессорами могут использоваться ускорители (графические процессоры или сопроцессоры Intel Xeon Phi). В статье описывается опыт использования DVM-системы для распараллеливания различных прикладных программ. Рассматривается метод инкрементального или частичного распараллеливания, возможности системы для работы с неструктурированными сетками, новые средства для отображения MPI-программ на многоядерные процессоры и ускорители. Исследуется эффективность выполнения параллельных DVMH-программ на гетерогенных вычислительных кластерах K-10, K-100, Ломоносов и MVS-10P. Описаны основные преимущества DVM-подхода при разработке параллельных программ. Представлены основные возможности инструментов DVM-системы для анализа производительности и функциональной отладки параллельных программ. Определяются направления для дальнейшего развития DVM-системы.

Ключевые слова: автоматизация разработки параллельных программ, DVM-система, спецификации параллелизма, ускоритель, графический процессор, сопроцессор, Фортран, Си.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Бахтин В.А., Захаров Д.А., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н. Решение прикладных задач с использованием DVM-системы // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2019. Т. 8, № 1. С. 89–106. DOI: 10.14529/cmse190106.

Введение

В последнее время все большее распространение получают вычислительные кластеры, в узлах которых установлены ускорители различной архитектуры. Внедрение таких вычислительных систем серьезно усложнило процесс разработки прикладных программ. Для высокопроизводительных вычислений на современных кластерах широко используются три модели программирования — MPI, OpenMP и CUDA. При этом пока вполне можно обходиться комбинацией MPI/OpenMP или MPI/CUDA, поскольку графические ускорители (далее ГПУ) используют для тех программ, для которых они значительно эффективнее, чем многоядерные универсальные процессоры (далее ЦПУ), и, следовательно, неполной загрузкой многоядерных процессоров можно пренебречь. Если будут

*Статья рекомендована к публикации программным комитетом Международной конференции «Суперкомпьютерные дни в России — 2017».

появляться программы, в которых только часть вычислений выгодно производить на ГПУ, а оставшиеся вычисления лучше оставить на ЦПУ, то придется использовать и комбинацию из трех перечисленных моделей.

Технически объединять низкоуровневые модели программирования, реализованные через библиотеки, проще, чем высокоуровневые модели, реализуемые посредством языков и соответствующих компиляторов. Но программировать, отлаживать, сопровождать, переносить на другие ЭВМ такие программы гораздо сложнее. Например, при переходе от ГПУ фирмы NVIDIA к ГПУ фирмы AMD придется заменить CUDA на OpenCL. Поэтому важно использовать высокоуровневые модели и языки программирования.

Среди высокоуровневых моделей программирования особое место занимают модели, реализуемые посредством добавления в программы на стандартных последовательных языках спецификаций, управляющих отображением этих программ на параллельные машины. Эти спецификации, оформляемые в виде комментариев в Фортран-программах или директив компилятору (прагм) в программах на языках Си и Си++, не видны для обычных компиляторов, что значительно упрощает внедрение новых моделей параллельного программирования. Такими моделями для кластеров являются HPF и XcalableMP, для мультипроцессоров — OpenMP, для ускорителей — OpenACC, для гибридных вычислительных кластеров с ускорителями — DVMH.

Система автоматизации разработки параллельных программ (DVM-система) существенно упрощает процесс разработки параллельных DVMH-программ. Получаемые программы без каких-либо изменений могут выполняться на кластерах, использующих многоядерные процессоры, графические ускорители и сопроцессоры Intel Xeon Phi. Это достигается за счет различных оптимизаций, которые выполняются как статически, при компиляции DVMH-программ, так и динамически. Получаемые параллельные программы могут настраиваться при запуске на выделенные для их выполнения ресурсы — количество узлов кластера, ядер, ускорителей и их производительность [1].

Данные оптимизации позволяют добиться высокой эффективности выполнения различных тестовых программ (например, программ из пакета NAS NPB [2, 3]) и реальных приложений (получены результаты при использовании более 1000 ГПУ [4]).

DVM-система активно развивается, появляются новые возможности, которые расширяют область применимости системы. Например, позволяют распараллеливать не только задачи на структурированных сетках (для которых DVM-система была предназначена изначально), но и задачи на неструктурированных сетках, а также использовать средства DVM для добавления новых уровней параллелизма в уже существующие MPI-программы.

В статье представлен опыт разработки параллельных программ с использованием DVM-системы. В разделе 1 представлены результаты инкрементального распараллеливания программы моделирования многокомпонентной фильтрации при разработке месторождений нефти и газа. В разделе 2 описан опыт распараллеливания программы, решающей двумерную задачу теплопроводности в шестиграннике. В разделе 3 демонстрируется использование средств DVM для дополнительного распараллеливания MPI-программы, решающей по явной схеме систему гиперболических уравнений в двумерной области сложной формы. В разделе 4 перечислены основные возможности DVM-системы, сформулированы преимущества DVM-подхода при разработке параллельных программ. В заключении обобщаются основные результаты исследования и определяются направления для дальнейшего развития DVM-системы.

1. Распараллеливание прикладных задач, использующих структурированные сетки

Распараллеливание программы в модели DVMH можно разделить на следующие этапы:

1. Распределение данных (массивов) и вычислений (параллельных циклов).
2. Определение и спецификация удаленных данных (данные, которые вычисляются на одном процессоре, а используются на других).
3. Определение вычислительных регионов (последовательности операторов, циклов) для выполнения на ускорителях.
4. Управление перемещением данных между памятью ЦПУ и памятью ускорителей.

Основная сложность разработки параллельной программы для кластера — необходимость принятия глобальных решений по распределению данных и вычислений с учетом свойств всей программы, а затем выполнения кропотливой работы по модификации программы и ее отладке. Большой объем программного кода, многомодульность, многофункциональность затрудняет принятие решений по согласованному распределению данных и вычислений.

Для решения данной проблемы может использоваться метод инкрементального, или частичного распараллеливания. Идея этого метода заключается в том, что распараллеливанию подвергается не вся программа целиком, а ее части (области распараллеливания) — в них заводятся дополнительные экземпляры требуемых данных, производится распределение этих данных и соответствующих вычислений. Данные области могут быть построены на основе времени, полученного с помощью профилирования последовательной программы.

Для взаимодействия с теми частями программы, которые не подвергались распараллеливанию, используются операции копирования исходных (нераспределенных) данных в дополнительные (распределенные) данные и обратно. Конечно, операции копирования могут снизить или вообще ликвидировать эффект от распараллеливания. Кроме того, до полного распараллеливания (пока не все массивы программы будут распределены) программе будет требоваться память и для распределенных, и для нераспределенных массивов, что может ограничивать размер решаемой задачи.

К достоинствам инкрементального распараллеливания можно отнести:

1. Возможность распараллелить не всю программу, а ее времяемкие фрагменты, упрощает работу программиста, так как существенно сокращается объем кода программы для анализа и распараллеливания.
2. Отказ от распараллеливания сложных фрагментов программы позволяет с большей вероятностью найти хорошие решения для выделенных областей распараллеливания.
3. Найденные решения могут быть использованы в качестве подсказки при распараллеливании других частей программы на следующих этапах.

Данный метод был успешно применен для распараллеливания программы моделирования многокомпонентной фильтрации при разработке месторождений нефти и газа.

1.1. Многокомпонентная многофазная изотермическая фильтрация

Композиционные модели фильтрации используются при подробном моделировании залежей, содержащих легкие углеводороды (конденсат и газ) в том случае, когда необходимо тщательно описывать массообмен между фазами, либо когда пластовые флюиды содержат ценные неуглеводородные компоненты. Эти модели также часто используются для изучения методов увеличения нефтеотдачи при закачке газов высокого давления, азота, углекислого газа и других агентов.

Последовательная версия программы «Композит» для решения задач многокомпонентной многофазной изотермической фильтрации была разработана в НИИСИ РАН [5]. Параллельная версия программы была разработана на языке Fortran-DVMH специалистами ИПМ РАН.

Для определения времяемких фрагментов программы, требующих распараллеливания, использовалось профилирование. В табл. 1 показано время выполнения различных процедур программы «Композит».

Таблица 1

Время выполнения различных процедур программы «Композит»

Процедура	**Время, в секундах**
INDA	0,0872
INCON	0,0204
GRID	0,3002
DECL	0,3088
CONDIN	0,3032
INSATZ	3,0720
TRANSM	0,0331
TRANSMOD	0,0035
REMAPI	9,2835
REMAPR	25,9266
RESERVS	0,8419
WELLIN	2,4177
COEF	2,5344
SXYN	476,3081
CFPR	2,9180
PWELL	0,0661
LSOR	12,4683
CWELL	4,2935
COMPOZ	45,4745
SXYDEF	313,9402
Вся программа	**910,2972**

Данные были получены на суперкомпьютере К-100 [6] с использованием анализатора производительности, который входит в состав DVM-системы. Механизм интервалов, реализованный в анализаторе, позволяет получить временные характеристики выполнения

программы с различной степенью подробности (например, для каждого цикла, для каждой процедуры). Если не учитывать процедуры, в которых выполняется ввод/вывод — REMAPI и REMAPR, то основное время занимают процедуры: COMPOZ, SXYN, SXYDEF и LSOR. Эти процедуры и были распараллелены на первом этапе.

В табл. 2 показано время выполнения (в секундах) частично распараллеленной программы «Композит» на суперкомпьютере К-100 при использовании различного числа вычислительных узлов.

Таблица 2

Время выполнения частично распараллеленной программы на К-100

Процедура	**1 ядро**	**1 узел**	**2 узла**	**3 узла**	**4 узла**	**8 узлов**
INDA	0,0498	0,0372	0,0588	0,1095	0,1157	0,0879
INCON	0,0185	0,0243	0,0233	0,0191	0,0190	0,0249
GRID	0,3114	0,3201	0,3197	0,3197	0,3204	0,3164
DECL	0,3229	0,3225	0,3208	0,3200	0,3205	0,3367
CONDIN	0,3083	0,9399	0,9389	0,8835	0,8824	0,9561
INSATZ	3,1876	0,4715	0,4179	0,3903	0,3925	0,4238
TRANSM	0,0314	0,0307	0,0309	0,0315	0,0314	0,0316
TRANSMOD	0,0042	0,0026	0,0055	0,0027	0,0023	0,0060
REMAPI	9,0168	8,9658	8,9520	8,9647	8,9968	9,1267
REMAPR	26,7028	26,3997	26,3024	26,0393	25,9841	26,0864
RESERVS	0,8317	0,8338	0,8359	0,8340	0,8343	0,8663
WELLIN	2,4394	37,5341	37,5114	37,3192	37,2989	37,4203
COEF	2,7969	3,0714	3,0623	3,0590	3,0536	3,0149
SXYN	436,1784	39,6501	20,4180	14,1194	10,8128	10,9114
CFPR	2,786	2,9011	2,8950	2,8890	2,8966	2,8845
PWELL	0,0654	0,0676	0,0659	0,0656	0,0680	0,0656
LSOR	12,3597	2,2833	2,1432	2,3122	2,3465	2,7084
CWELL	4,2860	4,4267	4,4218	4,4199	4,4250	4,4209
COMPOZ	67,2481	18,0482	15,1819	14,3580	14,0307	14,3647
SXYDEF	329,4486	41,5750	35,3501	33,2744	32,4631	34,5857
Вся программа	**907,8578**	**161,9112**	**133,4614**	**123,7524**	**119,7692**	**122,4489**

При использовании 4-х вычислительных узлов К-100 программа ускоряется в 7,5 раз по сравнению с выполнением программы на одном ядре (время счета сокращается с 908 до 120 секунд). Дальнейшее увеличение числа используемых узлов приводит к замедлению программы.

Основная причина — рост накладных расходов на копирование данных из распределенных массивов в исходные нераспределенные массивы, которое выполняется после завершения выполнения параллельной версии процедуры (табл. 3). При увеличении числа используемых узлов, время выполнения операций копирования массивов начинает существенно превышать время выполнения самой процедуры (например, процедура SXYDEF на 1 узле выполняется 29,35 секунд, время копирований составляет 12 секунд; на 4-х узлах — время счета составляет 7,57 секунд, а время копирования уже 24,74 секунды). Избавиться от операций копирования позволяет лишь полное распараллеливание программы.

Таблица 3

Накладные расходы на копирование данных между исходными данными и их копиями

Интервал	1 ядро	1 узел	2 узла	3 узла	4 узла	8 узлов
Копирование до выполнения SXYN	5,7900	1,1122	0,5898	0,4751	0,3694	0,2567
Процедура SXYN	429,7672	38,1114	19,2957	12,8864	9,6858	9,6749
Копирование после выполнения SXYN	0,0700	0,2091	0,3135	0,4028	0,4483	0,4869
Копирование до выполнения LSOR	0,8583	0,3204	0,1805	0,1302	0,1062	0,0870
Процедура LSOR	11,3605	1,5575	1,2807	1,2538	1,3051	1,5619
Копирование после выполнения LSOR	0,1229	0,3321	0,6065	0,7633	0,8358	0,8358
Копирование до выполнения COMPOZ	18,9040	3,3755	1,8916	1,3577	1,3577	0,8558
Процедура COMPOZ	44,4443	8,7812	4,8451	3,4420	2,4817	2,2543
Копирование после выполнения COMPOZ	2,1775	5,2959	8,2695	9,2633	10,3155	11,2422
Копирование до выполнения SXYDEF	10,6708	1,8021	1,0436	0,7518	0,6409	0,4999
Процедура SXYDEF	312,9793	29,3578	15,0164	10,0026	7,5699	7,5578
Копирование после выполнения SXYDEF	4,9856	10,2023	19,2134	22,0208	24,0996	26,3115

Если распараллеливание основных процедур (2700 из 13844 строк) программы фактически не требовало изменения текста последовательной программы и было выполнено достаточно быстро, то полное распараллеливание потребовало серьезного изменения структуры программы: инлайн подстановки процедур (для фрагментов программы, выполняемых на ускорителях), преобразования операторов ввода/вывода (DVMH-модель накладывает некоторые ограничения на использование распределенных массивов в операторах ввода/вывода), перехода на динамические массивы (вместо их моделирования) и др. Найденные при распараллеливании основных процедур решения по распределению данных были использованы при распараллеливании других частей программы.

Полное распараллеливание программы позволяет запускать ее в различных режимах. Режим работы DVMH-программы, количество используемых нитей, графических процессоров задается при помощи переменных окружения и не требует перекомпиляции программы. Переменная DVMH_PPN задает количество MPI-процессов, которые будут запускаться на каждом узле суперкомпьютера. Переменная DVMH_NUM_THREADS определяет количество рабочих нитей, которые будут созданы для выполнения программы на ядрах ЦПУ, для каждого из процессов. Переменная DVMH_NUM_CUDAS определяет количество CUDA-ускорителей для каждого из процессов. Переменные DVMH_CPU_PERF и DVMH_CUDAS_PERF позволяют задать условную производительность (вес) ЦПУ- и CUDA-устройств. С учетом этой производительности будут распределяться данные между различными вычислительными устройствами узла. Если ка-

кая-то из перечисленных переменных не будет задана, то ее оптимальное значение определяет система поддержки выполнения DVMH-программ, которая обеспечивает эффективное использование всех ресурсов узла.

В табл. 4 и 5 показано время выполнения (в секундах) 100 итераций программы «Композит» в режиме MPI/OpenMP на суперкомпьютерах MVS-10P [7], K-100 и Ломоносов [8] при использовании от 1 до 8 вычислительных узлов для вариантов расчета при закачке в пласт сухого газа и газа, обогащенного промежуточными фракциями.

Таблица 4

Время выполнения 100 итераций параллельной программы для варианта расчета с закачкой сухого газа на разном числе узлов

Суперкомпьютер	**1 MPI**	**2 MPI**	**4 MPI**	**8 MPI**	**16 MPI**
MVS-10P	140,69	55,57	30,58	17,26	10,83
K-100	165,57	81,53	48,78	27,05	16,85
Ломоносов	290,06	142,00	75,63	42,04	25,42

Таблица 5

Время выполнения 100 итераций параллельной программы для варианта расчета с закачкой жирного газа на разном числе узлов

Суперкомпьютер	**1 MPI**	**2 MPI**	**4 MPI**	**8 MPI**	**16 MPI**
MVS-10P	143,27	56,30	31,22	17,94	11,50
K-100	165,96	82,28	44,49	25,21	17,14
Ломоносов	288,56	140,77	74,96	40,88	24,32

На каждом узле суперкомпьютера запускался 1 или 2 MPI-процесса, каждый из которых создавал 8 (для MVS-10P), 6 (для K-100) или 4 OpenMP-нити. Всего для расчетов использовалось до 128 ядер ЦПУ. При увеличении в 16 раз количества используемых ядер расчет ускоряется в 10–13 раз.

В табл. 6 показано время выполнения полного расчета программы «Композит» для сетки 261x261x6 на суперкомпьютерах K-100 и K-10 [6] при использовании графических ускорителей.

Таблица 6

Время выполнения программы «Композит» в различных режимах на одном узле суперкомпьютера K-100 и K-10

Система	**Последовательная программа**	**1 MPI**			**2 MPI**		
K-100	11698	12 нитей	1 GPU	12 нитей + 1 GPU	6 нитей	1 GPU	6 нитей + 1 GPU
		1152	2518	979	1179	1258	845
K-10	9961,8	16 нитей	1 GPU	16 нитей + 1 GPU	8 нитей	1 GPU	8 нитей + 1 GPU
		912	1745	816,2	945	969,8	742

Использование 1-го графического ускорителя позволяет ускорить выполнение программы в 4,6–5,7 раз по сравнению с выполнением на 1-ом ядре. Но при этом программа выполняется медленнее чем, при использовании всех ядер ЦПУ узла. Основная причина — не удается сбалансировать нагрузку нитей (вычисления в различных точках сетки, в которых находятся добывающие, нагнетательные скважины, существенно отличаются). Получить ускорение за счет использования графических ускорителей удается за счет совмещения вычислений на ЦПУ и ГПУ. В таком режиме программа ускоряется с 11698 секунд до 845 секунд на К-100; с 9961,8 секунд до 742 секунд на К-10. Таким образом, при использовании всех вычислительных устройств 1 узла суперкомпьютера ускорение выполнения программы составляет 13,84 раза для К-100 и 13,42 раза для К-10 по сравнению с выполнением программы на 1-м ядре.

2. Распараллеливание прикладных задач, использующих неструктурированные сетки

Для удовлетворения желания увеличения точности вычислений, исследователям-вычислителям приходится значительно измельчать расчетную сетку. Это приводит к пропорциональному росту потребления памяти ЭВМ и увеличению времени расчетов. Частично с этой проблемой позволяет справиться переход с использования структурированных сеток на неструктурированные. В этом случае появляется возможность варьировать подробность сетки по расчетной области, тем самым сократив и время на излишне точный обсчет некоторых областей, и оперативную память, освобожденную от хранения невостребованно подробных полей величин. Также привлекательной чертой является абстрагирование численных методов от геометрии расчетной области и практическое снятие требований к ней.

В 2016 году были сформулированы основные предложения по расширению DVMH-модели для задач, использующих неструктурированные сетки [9]:

- введены новые правила для распределения данных (поэлементное — косвенное и производное);
- новые возможности для построения согласованных распределений (блочных или поэлементных);
- новый способ для задания произвольных по содержанию буферов удаленных элементов с эффективным однородным доступом к ним и обновлением;
- возможность реорганизации данных — оптимизации шаблона доступа к памяти путем изменения порядка хранения локальных элементов;
- сохранение быстрого доступа к распределенным массивам с помощью механизмов перехода на локальную индексацию.

На данный момент не все возможности, описанные выше, были реализованы в DVM-системе. Однако некоторые виды программ на нерегулярных сетках удается распараллелить имеющимися средствами уже сейчас.

2.1. Двумерная задача теплопроводности в шестиграннике

Рассмотрим двумерную задачу теплопроводности с постоянным, но разрывным коэффициентом в шестиграннике (рис. 1). Область состоит из двух материалов с различными коэффициентами температуропроводности.

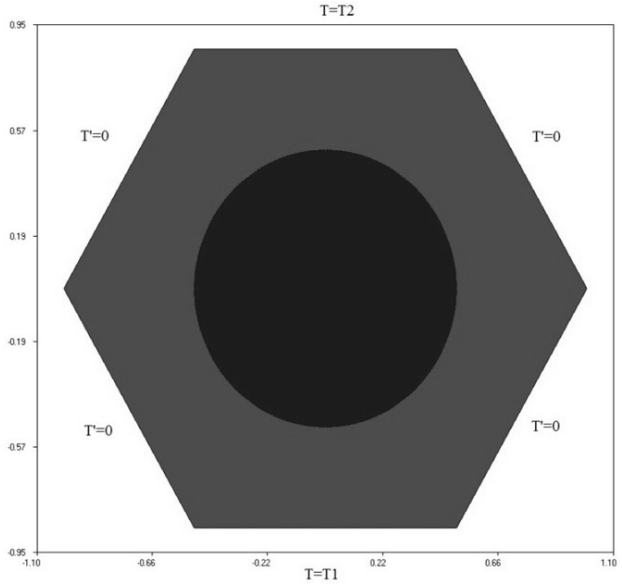


Рис. 1. Расчетная область и граничные условия

Рассмотрим фрагмент программы на языке Фортран, реализующий одну итерацию по времени для расчета по явной схеме (рис. 2):

```
      do i = 1, np2
        nn = ii(i)
        nb = npa(i)
        if (nb.ge.0) then
          s1 = FS(xp2(i),yp2(i),tv)
          s2 = 0d0
          do j = 1, nn
            j1 = jj(j,i)
            s2 = s2 + aa(j,i) * tt1(j1)
          enddo
          s0 = s1 + s2
          tt2(i) = tt1(i) + tau * s0
        else if (nb.eq.-1) then
          tt2(i) = vtemp1
        else if (nb.eq.-2) then
          tt2(i) = vtemp2
        endif
        s0 = (tt2(i) - tt1(i)) / tau
        gt = DMAX1(gt,DABS(s0))
      enddo
      do i = 1, np2
        tt1(i) = tt2(i)
      enddo
```

Рис. 2. Фрагмент Фортран-программы на нерегулярной сетке

Как видно из этого фрагмента, массивы величин `tt1` и `tt2` являются одномерными несмотря на то, что расчетная область двумерная. Также видно, что количество соседей у каждой ячейки не является константой, а читается из массива `ii`, а сами номера соседних ячеек читаются из массива `jj`, с которым связан также массив `aa`, описывающий каждую такую связь с соседом (в данном примере — просто коэффициент при суммировании). Существующая версия DVM-системы позволяет распараллелить такую программу при следующем ограничении: существует такое небольшое M, при котором все

соседи ячейки с номером n имеют номера, отличающиеся от n не более, чем на M. Ключевое слово здесь «небольшое», т.к. очевидно, что иначе в качестве такого M подойдет общее количество элементов сетки. Допустимая величина M — вещь субъективная, однако, учитывая описанный ниже способ распараллеливания, предлагается считать допустимым такое M, при котором M/N стремится к нулю при измельчении сетки (N — общее количество ячеек в сетке).

При распараллеливании в модели DVMH данной программы, были распределены:

- массивы искомых величин `tt1` и `tt2`;
- массивы описания элементов сетки `npa`, `xp2`, `yp2`;
- вспомогательные топологические массивы `ii` и `jj`;
- массив описания связей `aa`.

В результате, все вычисления производились только над распределенными данными с использованием только распределенных данных. Наиболее существенным при таком распараллеливании было вычисление необходимой ширины теневых граней для распределенного массива `tt1`.

Размеры теневых граней у `tt1` должны обеспечить присутствие (в локальной части или в теневой грани) на процессоре, владеющим элементом с индексом `i` также и всех элементов с индексами `jj(1:ii(i),i)`. Очевидно, что любое число M из приведенного выше ограничения является подходящей шириной теневой грани для массива `tt1`. Так как эта характеристика становится известной только во время выполнения, то был применен следующий подход:

1. Сначала делалось блочное распределение одномерных массивов без теневых граней.
2. Затем, анализируя результаты этого распределения и данные о связях, каждый процессор вычислял минимальную ширину теневой грани, покрывающую все его нужды (ссылки).
3. Затем производилось объединение этих требований со взятием максимального из них.

Учитывая, что теневые грани — это то, что будет пересылаться каждую итерацию по времени между процессорами, видно, что в этом подходе имеется сразу 2 места, ведущие к завышению пересылаемых объемов за счет ненужных данных.

Во-первых, ширина теневых граней в модели DVMH в настоящее время является универсальной величиной, не зависящей от номера процессора, т.е. нельзя одному процессору иметь ширину теневых граней 5, а другому — 10. Поэтому, чтобы удовлетворить требования всех процессоров, приходится задавать максимальную среди минимально необходимых отдельным процессорам ширину теневых граней.

Во-вторых, в теневые грани возможно включать только непрерывные участки распределенного массива, непосредственно примыкающие к локальной части процессора и невозможно включать отдельные элементы распределенных массивов. Это ограничение заставляет иметь теневую грань достаточно широкую для того, чтобы объять все элементы массива, на которые имеются ссылки при вычислении собственных элементов.
Однако в рассматриваемой задаче размеры ячеек сетки были примерно одинаковыми, а также отсутствовали «вытянутые» элементы (с диаметром, значительно отличающимся от среднего), что позволило удачным для распараллеливания в модели DVMH способом упорядочить сеточные элементы с целью минимизации включения лишних элементов в

теневые грани распределенных массивов. А именно, было применено геометрическое упорядочивание снизу–вверх, слева–направо.

Такой порядок нумерации сеточных элементов в сочетании с блочным распределением одномерных массивов величин (индексируемых номерами сеточных элементов) приводит к распределению расчетной области по процессорам слоями (горизонтальными полосами). Одну из расчетных сеток можно видеть на рис. 3, на котором изображен результат расчета (стационарное распределение температуры).

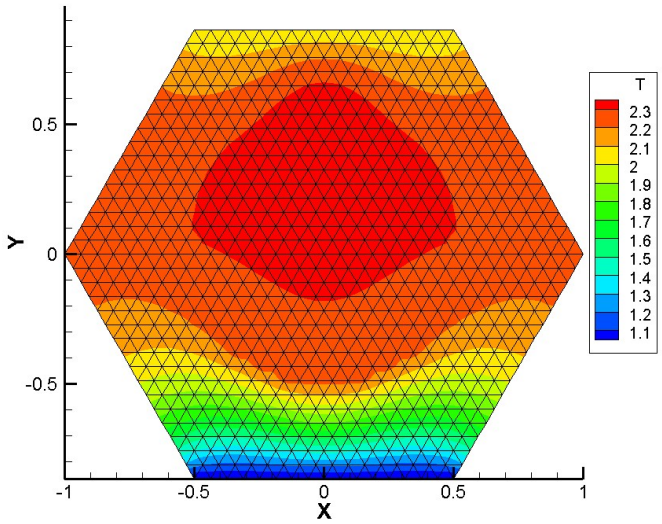


Рис. 3. Стационарное распределение температуры

В табл. 7 и 8 показаны время (в секундах) и ускорения параллельных версий программы для решения задачи теплопроводности (явная и неявная схемы) при использовании различного числа ЦПУ и ГПУ кластера К-100 для сетки 8 млн. узлов.

Таблица 7

Параллельное выполнение программы на ЦПУ

Схема	Посл	Параллельное выполнение, на разном количестве ядер ЦПУ											
		2		4		8		12		24		96	
	Время	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.
Явная	174	87,2	2	50	3,49	32	5,45	21,7	8,02	11,1	15,7	2,75	63,3
Неявная	928	728	1,27	611	1,52	449	2,07	309	3	155	6	42,8	21,7

Таблица 8

Параллельное выполнение программы на ЦПУ

Схема	Посл	Параллельное выполнение, на разном количестве ядер ЦПУ											
		1		2		3		6		12		24	
	Время	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.	Время	Уск.
Явная	174	7	24,8	3,81	46	2,76	63	1,5	116	0,87	200	0,55	316
Неявная	928	77	12	42,3	22	29,8	31	16,3	57	9,64	96	6,55	142

Полная реализация всех предложений по расширению DVMH-модели для задач, использующих неструктурированные сетки, должна повысить эффективность выполнения данных программ и улучшить полученные на данном этапе ускорения.

3. Дополнительное распараллеливание MPI-программ

В настоящее время, когда параллельные машины уже не одно десятилетие эксплуатируются для проведения расчетов, имеется множество программ, которые уже распараллелены на кластер, однако не позволяют эффективно использовать многоядерные процессоры и ускорители.

Традиционно в DVM-подходе весь процесс программирования (или распараллеливания имеющихся последовательных программ) начинается с распределения массивов, а затем отображения на них параллельных вычислений. Это означает, что для использования средств DVM-системы распараллеленные, например, на MPI, программы приходится превращать обратно в последовательные и заменять распределенные вручную данные и вычисления на описанные на DVM-языке распределенные массивы и параллельные циклы.

Однако, во-первых, автору не всегда хочется отказываться от своей параллельной программы, а во-вторых, не всегда удается перевести исходную схему распределения данных и вычислений на DVM-язык. Одним из способов избавиться от обеих проблем стал новый режим работы DVM-системы, в котором ее возможности используются локально в каждом MPI-процессе для организации работы на многоядерных процессорах и ускорителях. Данный режим включается заданием специально созданной MPI-библиотеки при сборке DVM-системы.

Кроме такого режима, в компиляторы C-DVMH [10] и Fortran-DVMH [11] введено понятие нераспределенного параллельного цикла, для которого нет необходимости задавать отображение на распределенный массив. Например, трехмерный параллельный цикл может выглядеть так (рис. 4).

```
#pragma dvm parallel(3) reduction (max(eps))
for (int i = L1; i <= H1; i++)
    for (int j = L2; j <= H2; j++)
        for (int k = L3; k <= H3; k++)
...
!DVM$ PARALLEL(I,J,K) REDUCTION (MAX(EPS))
      DO I = L1,H1
         DO J = L2, H2
            DO K = L3, H3
...
```

Рис. 4. Новый режим распределения вычислений

По определению такой цикл выполняется всеми процессорами текущей многопроцессорной системы, но в описанном новом режиме такая конструкция не приводит к размножению вычислений, а только лишь позволяет использовать параллелизм внутри одного процесса (ЦПУ или ГПУ). Как следствие, появляется возможность не задавать ни одного распределенного в терминах модели DVMH массива и в то же время пользоваться возможностями DVM-системы:

– добавление параллелизма в общей памяти (ядра ЦПУ): с использованием OpenMP или без, возможность задания привязки нитей;

– использование ГПУ: не только «наивное» портирование параллельного цикла на ускоритель, но и выполнение автоматической реорганизации данных, упрощенное управление перемещениями данных;
– подбор оптимизационных параметров;
– удобные средства отладки производительности.

Такой режим может быть использован в том числе для получения промежуточных результатов в процессе проведения полноценного распараллеливания программы в модели DVMH. Он позволяет быстро и заметно проще получить программу для многоядерного ЦПУ и ГПУ, а также оценить перспективы ускорения целевой программы на кластере с многоядерными ЦПУ и ускорителями.

В качестве примера приведем программу, являющуюся частью большого развитого комплекса вычислительных программ (В.А. Гасилов, А.С. Болдарев, ИПМ им. М.В. Келдыша РАН). Будучи ориентированным на решение по явной схеме систем гиперболических уравнений (в основном, газовой динамики) в двумерных областях сложной формы с использованием неструктурированных сеток, этот код был написан на C++ с очень широким использованием объектно-ориентированного подхода для обеспечения максимальной универсальности и простоты дальнейшего развития.

Так как эта программа является частью целого комплекса, код ее основан на богатой платформе базовых понятий и структур данных. Это приводит к значительным размерам (39 тыс. строк) и сложности всей программы, если ее рассматривать целиком.

Полноценное распараллеливание программы в модели DVMH вряд ли возможно без рассмотрения и модификации всей программы. Новые возможности позволили выполнить «локальное» распараллеливание вычислительноемких частей программы. Модификации подверглись лишь 3 из 39 тыс. строк программы.

В результате такого распараллеливания на 12 ядрах ЦПУ с использованием OpenMP-нитей было получено ускорение в 9,83 раза относительно последовательной версии, а на ГПУ NVIDIA GTX Titan — в 18 раз относительно последовательной версии. Данные результаты подтверждают эффективность отображения рассматриваемой программы DVM-системой на ускорители и многоядерные ЦПУ и дают основания продолжить распараллеливание программы уже с использованием распределенных массивов в модели DVMH.

4. Преимущества DVM-подхода для разработки параллельных программ

DVM-система автоматизирует процесс разработки параллельных программ.

Анализатор производительности позволяет определить времяемкие фрагменты или циклы программы, требующие распараллеливания. Механизм интервалов, реализованный в анализаторе, позволяет получить временные характеристики выполнения программы с различной степенью подробности. Пользователь DVM-системы может управлять разбиением программы на интервалы при ее компиляции. Для каждого такого фрагмента собирается информация, которая позволяет оценить потери, возникающие при распараллеливании программы, например, потери из-за выполнения межпроцессорных обменов, потери из-за перемещений данных из памяти ЦПУ в ГПУ, потери из-за простоев процессоров, на которых выполнение программы завершилось раньше, чем на остальных и.т.п. Данная

информация очень востребована при распараллеливании программы и важна при ее дальнейшей отладке эффективности.

DVM-отладчики автоматизируют процесс обнаружения ошибок, возникающих в процессе распараллеливания программы. Например, сравнительная отладка позволяет быстро обнаружить ошибки в распараллеливаемой программе, возникающие в результате переноса программы из ОС Windows в ОС Linux, или различия при выполнении программы на ЦПУ и ГПУ.

DVMH-модель параллельного программирования рассчитана на работу со структурированными и неструктурированными сетками, базируется на модели параллелизма по данным и модели параллелизма по управлению, что позволяет ее использовать для дополнительного распараллеливания в MPI-программах.

DVMH-компиляторы преобразуют входную программу в параллельную программу использующую стандартные технологии программирования MPI, OpenMP и CUDA, выполняя при этом различные анализы (например, режим автоматического определения приватных переменных для параллельных циклов) и оптимизации (например, реорганизацию данных для повышения эффективности доступа к глобальной памяти ГПУ).

Таким образом, использование DVM-системы позволяет существенно упростить процесс разработки параллельных программ для гетерогенных вычислительных кластеров.

Заключение

Новые возможности, которые появились в DVM-системе в последнее время, существенно расширили класс программ, которые могут быть распараллелены с помощью DVM-подхода.

Новые правила для распределения данных (косвенное, производное, согласованное) позволили использовать DVMH-модель для распараллеливания задач, использующих неструктурированные сетки.

Новый режим работы DVM-системы, в котором ее возможности используются локально в каждом MPI-процессе, позволяет отображать существующие MPI-программы на многоядерные процессоры и графические ускорители с использованием высокоуровневых DVM-спецификаций параллелизма.

Новые конструкции для распределения вычислений (например, нераспределенный параллельный цикл) существенно упрощают процесс инкрементального распараллеливания программ в DVM-модели.

В новой версии компилятора C-DVMH на базе Clang и инфраструктуры LLVM реализована начальная поддержка подмножества конструкций языка C++.

С использованием данных возможностей были разработаны параллельные версии программ:

- для моделирования многокомпонентной фильтрации при разработке месторождений нефти и газа,
- решения двумерной задачи теплопроводности в шестиграннике,
- решения по явной схеме системы гиперболических уравнений в двумерной области сложной формы.

Полученные DVMH-программы эффективно выполняются на гетерогенных кластерах, использующих многоядерные процессоры и графические ускорители.

В качестве следующего шага в развитии DVM-системы планируется расширить язык C-DVMH и компилятор с него для полноценной поддержки наиболее востребованных конструкций языка C++ (язык CDVMH++), а также исследовать подходы для интеграции модулей, разработанных с помощью DVM, в MPI-программы пользователей.

Работа подготовлена при поддержке программы президиума РАН №26 «Фундаментальные основы создания алгоритмов и программного обеспечения для перспективных сверхвысокопроизводительных вычислений».

Литература

1. Бахтин В.А., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н. Методы динамической настройки DVMH-программ на кластеры с ускорителями // Суперкомпьютерные дни в России: Труды международной конференции (Москва, 28–29 сентября 2015 г.). М.: Изд-во МГУ, 2015. С. 257–268.
2. Алексахин В.Ф., Бахтин В.А., Жукова О.Ф., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Савицкая О.А., Шуберт А.В. Распараллеливание на графические процессоры тестов NAS NPB 3.3.1 на языке Fortran DVMH // Вестник Уфимского государственного авиационного технического университета. 2015. Т. 19, №1(67). С. 240–250.
3. Алексахин В.Ф., Бахтин В.А., Жукова О.Ф., Колганов А.С., Крюков В.А., Островская И.П., Поддерюгина Н.В., Притула М.Н., Савицкая О.А. Распараллеливание на языке Fortran-DVMH для сопроцессора Intel Xeon Phi тестов NAS NPB3.3.1 // Параллельные вычислительные технологии (ПаВТ'2015): труды международной научной конференции (Екатеринбург, 31 марта–2 апреля 2015 г.). Челябинск: Издательский центр ЮУрГУ, 2015. С. 19–30.
4. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Смирнов А.А. Использование языка Fortran DVMH для решения задач гидродинамики на высокопроизводительных гибридных вычислительных системах // Вестник Южно-Уральского государственного университета, серия «Вычислительная математика и информатика». Челябинск: Издательский центр ЮУрГУ, 2013. Т. 2, № 3. С. 106–120. DOI: 10.14529/cmse130308.
5. Бахтин В.А., Королев А.В., Поддерюгина Н.В. Использование параллельных вычислений для моделирования многокомпонентной фильтрации при разработке месторождений нефти и газа // Международная конференция «Математика и информационные технологии в нефтегазовом комплексе»: тезисы докладов (Сургут, 16–20 мая 2016 г.). Сургут: ИЦ СурГУ, 2016. С. 164–166.
6. ИПМ им. М.В. Келдыша РАН. Вычислительные ресурсы. URL: http://www.kiam.ru/MVS/resourses/ (дата обращения: 18.05.2018).
7. Межведомственный Суперкомпьютерный Центр. Вычислительные системы. URL: http://www.jscc.ru/scomputers.html (дата обращения: 18.05.2018).
8. Суперкомпьютерный комплекс МГУ имени М.В. Ломоносова. URL: http://parallel.ru/cluster/ (дата обращения: 18.05.2018).
9. Бахтин В.А., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Поляков С.В., Притула М.Н. Расширение DVMH-модели для работы с нерегулярными сетками // Па-

раллельные вычислительные технологии (ПаВТ'2016): труды международной научной конференции (Архангельск, 28 марта–1 апреля 2016 г.). Челябинск: Издательский центр ЮУрГУ, 2016. С. 757.

10. Язык C-DVMH. C-DVMH компилятор. Компиляция, выполнение и отладка CDVMH-программ. URL: http://dvm-system.org/static_data/docs/CDVMH-reference-ru.pdf (дата обращения: 18.05.2018).
11. Язык Fortran-DVMH. Fortran-DVMH компилятор. Компиляция, выполнение и отладка DVMH-программ. URL: http://dvm-system.org/static_data/docs/FDVMH-user-guide-ru.pdf (дата обращения: 18.05.2018).

Бахтин Владимир Александрович, к.ф.-м.н., ведущий научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация)

Захаров Дмитрий Александрович, младший научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация)

Колганов Александр Сергеевич, младший научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация)

Крюков Виктор Алексеевич, д.ф.-м.н., профессор, главный научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация)

Поддерюгина Наталия Викторовна, к.ф.-м.н., старший научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация)

Притула Михаил Николаевич, старший научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация)

DOI: 10.14529/cmse190106

DEVELOPMENT OF PARALLEL APPLICATIONS USING DVM-SYSTEM

© 2019 V.A. Bakhtin, D.A. Zaharov, A.S. Kolganov, V.A. Krukov, N.V. Podderyugina, M.N. Pritula

Keldysh Institute of Applied Mathematics
(Miusskaya sq., 4, Moscow, 125047 Russia)
E-mail: bakhtin@keldysh.ru, s123-93@mail.ru, alexander.k.s@mail.ru, krukov@keldysh.ru, konov@keldysh.ru, pritula@keldysh.ru

Received: 21.06.2018

DVM-system was designed to create parallel programs of scientific-technical computations in C-DVMH and Fortran-DVMH languages. These languages use the same model of parallel programming (DVMH-model) and are the extensions of standard C and Fortran languages by parallelism specifications, implemented as compiler directives. DVMH-model allows creating efficient parallel programs for heterogeneous computational clusters, the nodes of which use as computing devices not only universal multi-core processors but also can use attached accelerators (GPUs or Intel Xeon Phi coprocessors). This article describes the experience of parallelizing various application programs using DVM-system. The method of incremental or partial parallelization, the system's capabilities for working with unstructured grids, new tools for mapping MPI-programs to multi-core processors and accelerators are considered. The efficiency of parallel DVMH-programs on heterogeneous computing clusters K-10, K-100, Lomonosov and MVS-10P is investigated. The main advantages of DVM-approach for the development of parallel programs are described. The main features of DVM-system tools for performance analysis and functional debugging of parallel programs are presented. The directions for further development of DVM-system are determined.

Keywords: automation the development of parallel programs, DVM-system, parallelism specification directives, accelerator, GPU, coprocessor, Fortran, C.

FOR CITATION

Bakhtin V.A., Zaharov D.A., Kolganov A.S., Krukov V.A., Podderyugina N.V., Pritula M.N. Development of Parallel Applications Using DVM-system. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2019. vol. 8, no. 1. pp. 89–106. (in Russian) DOI: 10.14529/cmse190106.

References

1. Bakhtin V.A., Kolganov A.S., Krukov V.A., Podderugina N.V., Pritula M.N. Methods of Dynamic Tuning of DVMH Programs on Clusters with Accelerators. *Superkompjuternye dni v Rossii: Trudy mezhdunarodnoj konferencii (Moskva, 28–29 sentjabrja 2015)* [Russian Supercomputing Days: Proceedings of the International Scientific Conference (Moscow, Russia, September 28–29, 2015)]. Moscow, Publishing of the Moscow State University, 2015. pp. 257–268. (in Russian)
2. Aleksahin V.F., Bakhtin V.A., Kolganov A.S., Krukov V.A., Podderugina N.V., Pritula M.N., Savitskaya O.A., Shubert A.V., Zhukova O.F. GPU Parallelization of NAS NPB3.3.1 Benchmarks Using Fortran DVMH Programming Language. *Vestnik Ufimskogo gosudarstvennogo aviacionnogo tehnicheskogo universiteta* [Bulletin of the Ufa State Aviation Technical University]. Ufa, 2015. vol. 19, no. 67. pp. 240–250. (in Russian)
3. Aleksahin V.F., Bakhtin V.A., Kolganov A.S., Krukov V.A., Ostrovskaya I.P., Podderugina N.V., Pritula M.N., Savitskaya O.A., Zhukova O.F. Parallelization of NAS NPB3.3.1 Tests on Fortran-DVMH for Intel Xeon Phi Coproccessor. *Parallelnye vychislitelnye tehnologii (PaVT'2015): trudy mezhdunarodnoj nauchnoj konferencii (Ekaterinburg, 31 marta–2 aprelja 2015)* [Parallel computational technologies 2015: Proceedings of the International Scientific Conference (Ekaterinburg, Russia, March, 31–April, 2, 2015)]. Chelyabinsk, Publishing of the South Ural State University, 2015. pp. 19–30. (in Russian)
4. Bakhtin V.A., Klinov M.S., Krukov V.A., Podderugina N.V., Pritula M.N., Smirnov A.A. Usage of Fortran DVMH Language for Solving Hydrodynamics Problems on Hybrid Computing Systems. *Vestnik Yuzho-Uralskogo gosudarstvennogo universiteta. Seriya "Vychislitelnaja matematika i informatika"* [Bulletin of the South Ural State University. Computation Mathematics and Software Engineering]. Chelyabinsk, Publishing of the South Ural State University, 2013. vol. 2, no. 3. pp. 106–120. (in Russian). DOI: 10.14529/cmse130308.
5. Bakhtin V.A., Korolev A.V., Podderugina N.V. Parallel Computations for Compositional Flow Simulation During Oil and Gas Fields Development. *Mezhdunarodnaja konferencija "Matematika i informacionnye tehnologii v neftegazovom komplekse": tezisy dokladov (Surgut, 16–20 maja 2016)* [Mathematics and Informational Technologies for Oil and Gas Industry: Abstracts of the International Scientific Conference (Surgut, Russia, May, 16–20, 2016)]. Surgut: Publishing center SurSU, 2016. pp. 164–166. (in Russian)
6. Keldysh Institute of Applied Mathematics. Computing Resources. Available at: http://www.kiam.ru/MVS/resourses/ (accessed: 18.05.2018).

7. Joint Supercomputer Center of the Russian Academy of Sciences. Available at: http://www.jscc.ru/scomputers.html (accessed: 18.05.2018).
8. Supercomputing Center of Lomonosov Moscow State University. Available at: http://parallel.ru/cluster/ (accessed: 18.05.2018).
9. Bakhtin V.A., Kolganov A.S., Krukov V.A., Podderugina N.V., Polyakov S.V., Pritula M.N. DVMH Model Extension for Operation with Irregular Grids. *Parallelnye vychislitelnye tehnologii (PaVT'2016): trudy mezhdunarodnoj nauchnoj konferencii (Arhangelsk, 28 marta–1 aprelja 2016)* [Parallel computational technologies 2015: Proceedings of the International Scientific Conference (Archangelsk, Russia, March, 28–April, 1, 2016)]. Chelyabinsk, Publishing of the South Ural State University, 2016. pp. 757. (in Russian)
10. C-DVMH Language, C-DVMH Compiler, Compilation, Execution and Debugging of DVMH Programs. Available at: http://dvm-system.org/static_data/docs/CDVMH-reference-en.pdf (accessed: 18.05.2018).
11. Fortran-DVMH Language. Fortran-DVMH Compiler. The Compilation, Execution and Debugging of DVMH-programs. Available at: http://dvm-system.org/static_data/docs/FDVMH-user-guide-en.pdf (accessed: 18.05.2018)

СВЕДЕНИЯ ОБ ИЗДАНИИ

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

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

Главный редактор — Л.Б. Соколинский.

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. Журнал размещен в открытом доступе на Всероссийском математическом портале MathNet. Сведения о журнале ежегодно публикуются в международной справочной системе по периодическим и продолжающимся изданиям «Ulrich's Periodicals Directory».

Решением Президиума Высшей аттестационной комиссии Министерства образования и науки Российской Федерации журнал включен в «Перечень рецензируемых научных изданий, в которых должны быть опубликованы основные научные результаты на соискание ученой степени кандидата наук, на соискание ученой степени доктора наук» по научным специальностям и соответствующим им отраслям науки: 05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей (физико-математические науки), 05.13.17 – Теоретические основы информатики (физико-математические науки).

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

Адрес редакции, издателя: 454080, г. Челябинск, проспект Ленина, 76, Издательский центр ЮУрГУ, каб. 32.

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 редакционной коллегии научного журнала «Вестник ЮУрГУ», серия «Вычислительная математика и информатика»:

   Россия 454080, г. Челябинск, пр. им. В.И. Ленина, 76, ЮУрГУ, кафедра СП,

   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.**