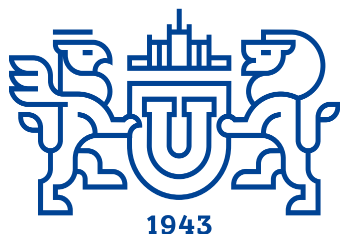


ВЕСТНИК

ЮЖНО-УРАЛЬСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2018
Т. 7, № 2

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

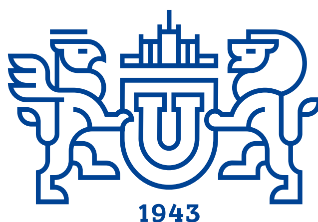
Учредитель — Федеральное государственное автономное образовательное учреждение высшего образования «Южно-Уральский государственный университет (национальный исследовательский университет)»

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

Редакционная коллегия
Л.Б. Соколинский, д.ф.-м.н., проф., *гл. редактор*
В.П. Танана, д.ф.-м.н., проф., *зам. гл. редактора*
М.Л. Цымблер, к.ф.-м.н., доц., *отв. секретарь*
Г.И. Радченко, к.ф.-м.н., доц.
Я.А. Краева, *техн. секретарь*

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



BULLETIN

OF THE SOUTH URAL STATE UNIVERSITY

2018
vol. 7, no. 2

SERIES
"COMPUTATIONAL MATHEMATICS AND SOFTWARE ENGINEERING"

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

Editorial Board
L.B. Sokolinsky, South Ural State University (Chelyabinsk, Russia)
V.P. Tanana, South Ural State University (Chelyabinsk, Russia)
M.L. Zymbler, South Ural State University (Chelyabinsk, Russia)
G.I. Radchenko, South Ural State University (Chelyabinsk, Russia)
Ya.A. Kraeva, South Ural State University (Chelyabinsk, Russia)

Editorial Council
S.M. Abdullaev, South Ural State University (Chelyabinsk, Russia)
A. Andrzejak, Heidelberg University (Germany)
V.I. Berdyshev, Institute of Mathematics and Mechanics, Ural Branch of the RAS (Yekaterinburg, Russia)
J. Dongarra, University of Tennessee (USA)
M.Yu. Khachay, Institute of Mathematics and Mechanics, Ural Branch of the RAS (Yekaterinburg, Russia)
D. Mallmann, Julich Supercomputing Centre (Germany)
A.V. Panyukov, South Ural State University (Chelyabinsk, Russia)
R. Prodan, University of Innsbruck (Innsbruck, Austria)
P. Shumyatsky, University of Brasilia (Brazil)
A. Tchernykh, CICESE Research Center (Mexico)
A.N. Tomilin, Institute for System Programming of the RAS (Moscow, Russia)
V.E. Tretyakov, Ural Federal University (Yekaterinburg, Russia)
V.I. Ukhobotov, Chelyabinsk State University (Chelyabinsk, Russia)
V.N. Ushakov, Institute of Mathematics and Mechanics, Ural Branch of the RAS (Yekaterinburg, Russia)
V.V. Voevodin, Lomonosov Moscow State University (Moscow, Russia)
S.V. Zykin, Sobolev Institute of Mathematics, Siberian Branch of the RAS (Omsk, Russia)

Содержание

Вычислительная математика

САМАЯ БЫСТРАЯ И ЭНЕРГОЭФФЕКТИВНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ПОИСКА В ШИРИНУ НА ОДНОУЗЛОВЫХ РАЗЛИЧНЫХ ПАРАЛЛЕЛЬНЫХ АРХИТЕКТУРАХ СОГЛАСНО РЕЙТИНГУ GRAPH500
А.С. Колганов 5

Дискретная математика и математическая кибернетика

ДИНАМИКА ИЗМЕНЕНИЯ ОБЛАСТЕЙ УСТОЙЧИВОСТИ ДИСКРЕТНЫХ МОДЕЛЕЙ НЕЙРОННЫХ СЕТЕЙ ТИПА SMALL WORLD ПРИ ИЗМЕНЕНИИ ЧИСЛОВЫХ ХАРАКТЕРИСТИК ГРАФА СЕТИ
С.А. Иванов, М.М. Кипнис 22

Информатика, вычислительная техника и управление

МОДЕЛЬ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ МНОГОПРОЦЕССОРНЫХ СИСТЕМ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ
Н.А. Ежова, Л.Б. Соколинский 32

ОПТИМИЗАЦИЯ ФРАГМЕНТАЦИИ ПРИ ВЫДЕЛЕНИИ РЕСУРСОВ ДЛЯ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ С СЕТЬЮ АНГАРА
А.В. Мукосей, А.С. Семенов 50

РАСПРЕДЕЛЕННЫЙ АЛГОРИТМ ОТОБРАЖЕНИЯ РАСПРЕДЕЛЕННЫХ МНОГОМЕРНЫХ ДАННЫХ НА МНОГОМЕРНЫЙ МУЛЬТИКОМПЬЮТЕР В СИСТЕМЕ ФРАГМЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ LUNA
Г.А. Щукин 63

© Издательский центр ЮУрГУ, 2018

Contents

Computational Mathematics

THE FASTEST AND ENERGY-EFFICIENT BREADTH-FIRST SEARCH ALGORITHM ON A SINGLE NODE WITH VARIOUS PARALLEL ARCHITECTURES ACCORDING TO GRAPH500
A.S. Kolganov 5

Discrete Mathematics and Mathematical Cybernetics

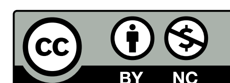
DYNAMICS OF STABILITY REGIONS OF DISCRETE MODELS OF NEURAL NETWORKS OF SMALL WORLD TYPE WHEN THE NUMERIC CHARACTERISTICS OF THE NETWORK GRAPH CHANGE
S.A. Ivanov, M.M. Kipnis 22

Computer Science, Engineering and Control

PARALLEL COMPUTATION MODEL FOR MULTIPROCESSOR SYSTEMS WITH DISTRIBUTED MEMORY
N.A. Ezhova, L.B. Sokolinsky 32

ALLOCATION OPTIMIZATION FOR REDUCING RESOURCE FRAGMENTATION IN ANGARA HIGH-SPEED INTERCONNECT
A.V. Mukosey, A.S. Semenov 50

DISTRIBUTED ALGORITHM FOR DISTRIBUTED DATA LATTICE MAPPING ON MULTIDIMENSIONAL MULTICOMPUTER IN THE LUNA FRAGMENTED PROGRAMMING SYSTEM
G.A. Schukin 63



This issue is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

УДК 004.021 DOI: 10.14529/cmse180201

САМАЯ БЫСТРАЯ И ЭНЕРГОЭФФЕКТИВНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ПОИСКА В ШИРИНУ НА ОДНОУЗЛОВЫХ РАЗЛИЧНЫХ ПАРАЛЛЕЛЬНЫХ АРХИТЕКТУРАХ СОГЛАСНО РЕЙТИНГУ GRAPH500*

© 2018 А.С. Колганов

Институт прикладной математики им. М.В. Келдыша РАН
(125047 Москва, Миусская пл., д. 4),
Московский государственный университет им. М.В. Ломоносова
(119991 Москва, ул. Ленинские горы, д. 1)
E-mail: alexander.k.s@mail.ru

Поступила в редакцию: 08.04.2018

Поиск в ширину является одним из основных алгоритмов обхода графа и базовым для многих алгоритмов анализа графов более высокого уровня. Поиск в ширину на графах является задачей с нерегулярным доступом к памяти и с нерегулярной зависимостью по данным, что сильно усложняет его распараллеливание на все существующие архитектуры. В статье будет рассмотрена реализация алгоритма поиска в ширину (основного теста рейтинга Graph500) для обработки больших графов на различных архитектурах: Intel x86, IBM Power8+, Intel KNL и NVidia GPU. Будет рассмотрены алгоритмы реализации поиска в ширину, такие как top-down обход, bottom-up обход и гибридный обход, содержащий в себе как top-down, так и bottom-up обходы. Будут описаны особенности реализации алгоритма на общей памяти, а также преобразования графа: локальная сортировка вершин графа, глобальная сортировка вершин графа, перенумерация всех вершин графа, сжатое представление вершин графа. Данные преобразования и гибридный алгоритм обхода позволяют достичь рекордных показателей производительности и энергоэффективности на данном алгоритме среди всех одноузловых систем рейтинга Graph500 и GreenGraph500.

Ключевые слова: параллельная обработка графов, BFS, CUDA, Power8, KNL, Graph500.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Колганов А.С. Самая быстрая и энергоэффективная реализация алгоритма поиска в ширину на одноузловых различных параллельных архитектурах согласно рейтингу Graph500 // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2018. Т. 7, № 2. С. 5–21. DOI: 10.14529/cmse180201.

Введение

В последнее время все большую роль играют графические ускорители (ГПУ) в неграфических вычислениях. Потребность их использования обусловлена их относительно высокой производительностью и более низкой стоимостью. Как известно, на ГПУ и центральных процессорах (ЦПУ) хорошо решаются задачи на структурных, регулярных сетках, где параллелизм так или иначе легко выделяется. Но есть задачи, которые требуют больших мощностей и используют неструктурированные сетки или данные. Примером таких задач являются: Single Shortest Source Path problem (SSSP) [1] — задача поиска кратчайших путей от заданной вершины до всех остальных во взвешенном графе, задача

*Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2018».

Breadth First Search (BFS) [2] — задача поиска в ширину в неориентированном графе, Minimum Spanning Tree (MST) — задача поиска сильно связанных компонент [3, 4] и другие.

Данные задачи являются базовыми в ряде алгоритмов на графах. На данный момент алгоритмы BFS и SSSP используются для ранжирования вычислительных машин в рейтингах Graph500 [5] и GreenGraph500 [6]. Алгоритм BFS (breadth-first search или поиск в ширину) является одним из наиболее важных алгоритмов анализа на графах. Он используется для получения некоторых свойств связей между узлами в заданном графе. В основном BFS используется как звено, например, в таких алгоритмах, как нахождение связных компонент [7], нахождение максимального потока [8], нахождение центральных компонент (betweenness centrality) [9, 10], кластеризация [11] и многие другие.

Алгоритм BFS имеет линейную вычислительную сложность $O(n + m)$, где n — количество вершин и m — количество ребер графа. Данная вычислительная сложность является наиболее оптимальной для последовательной реализации. Но такая оценка вычислительной сложности не применима для параллельной реализации, так как последовательная реализация (например, с помощью алгоритма Дейкстры [12]) имеет зависимости по данным, что препятствует ее распараллеливание. Также производительность алгоритма BFS ограничена производительностью памяти той или иной архитектуры. Поэтому наибольшее значение имеют оптимизации, направленные на улучшение работы с памятью всех уровней.

Данная работа направлена на исследование возможности отображения алгоритма поиска в ширину на больших графах на различные параллельные одноузловые архитектуры, такие как x86, Power8, NVidia, Intel KNL. Целью работы является разработка программы, реализующая алгоритм BFS, которая обладает следующими свойствами: единая программная реализация на сколько это возможно, одинаково эффективное выполнение одной и той же реализации на различных архитектурах, максимальная производительность и энергоэффективность по сравнению с существующими решениями в рейтингах Graph500 и GreenGraph500. На данный момент создать единую программную реализацию с использованием одной технологии, которая будет максимально эффективно работать на всех перечисленных архитектурах, невозможно. Поэтому для архитектуры графических ускорителей NVidia использовалась модель CUDA, а для всех остальных архитектур — модель OpenMP.

Статья организована следующим образом. В разделе 1 дается описание рейтингов Graph500 и GreenGraph500, а также анализ существующих решений. Раздел 2 описывает формат хранения графа и алгоритмы его преобразования. В разделе 3 содержится описание алгоритмов параллельной реализации для ЦПУ и ГПУ. В разделе 4 дается описание полученных результатов.

1. Обзор существующих решений и рейтинг Graph500

1.1. Graph500 и GreenGraph500

Рейтинг Graph500 был создан как альтернатива рейтингу Top500 [13]. Данный рейтинг используется для ранжирования вычислительных машин в приложениях, которые используют нерегулярный доступ к памяти, в отличие от последнего. Для тестируемого приложения в рейтинге Graph500 пропускная способность памяти и коммуникационной сети играют наиболее важную роль. Рейтинг GreenGraph500 является альтернативой рейтинга Green500 и используется в дополнении к Graph500.

В Graph500 используется метрика — количество обработанных ребер графа в секунду (TEPS — traversed edges per second), в то время как в GreenGraph500 используется метрика — количество обработанных ребер графа в секунду на один ватт. Таким образом, первый рейтинг ранжирует вычислительные машины по скорости вычисления, а второй — по энергоэффективности. Данные рейтинги обновляются каждые полгода.

1.2. Существующие решения

Алгоритм поиска в ширину был придуман более 50 лет назад. И до сих пор проводятся исследования для эффективной параллельной реализации на различных устройствах. Данный алгоритм показывает насколько хорошо организована работа с памятью и коммуникационной средой вычислителей. Существует достаточно много работ по распараллеливанию данного алгоритма на x86 системах [14–18] и на ГПУ [19, 20]. Также подробные результаты выполнения реализованных алгоритмов можно увидеть в рейтингах Green500 и GreenGraph500. К сожалению, алгоритмы многих эффективных реализаций не опубликованы в зарубежных источниках.

Описанная реализация алгоритма BFS в данной статье с использованием ускорителя Tesla P100 является лучшей по быстродействию и энергоэффективности среди всех одноузловых системы рейтинга Graph500 на графах с количеством вершин более 2^{25}. Более подробный анализ представлен в разделе 5.

2. Формат хранения графа

Для оценки производительности алгоритма BFS используются неориентированные RMAT графы [21]. RMAT графы хорошо моделируют реальные графы из социальных сетей, Интернета. В данном случае рассматриваются RMAT графы со средней степенью связности вершины 16, а количество вершин является степенью двойки. В таком RMAT графе имеется одна большая связная компонента и некоторое количество небольших связных компонент или висящих вершин. Сильная связность компонент не позволяет каким-либо образом разделить граф на такие подграфы, которые помещались бы в кэш память.

Для построения графа используется генератор, который предоставляется разработчиками рейтинга Graph500. Данный генератор создает неориентированный граф в формате RMAT, причем выходные данные представлены в виде набора ребер графа. Такой формат не очень удобен для эффективной параллельной реализации графовых алгоритмов, так как необходимо иметь агрегированную информацию по каждой вершине, а именно — какие вершины являются соседями для данной. Удобный для этого представления формат называется CSR (Compressed Sparse Rows) [22].

Данный формат получил широкое распространение для хранения разреженных матриц и графов. Для неориентированного графа с N вершинами и M ребрами необходимо два массива: X (массив указателей на смежные вершины) и (массив списка смежных вершин). Массив X размера $N+1$, а массив — $2*M$, так как в неориентированном графе для любой пары вершин необходимо хранить прямую и обратную дуги. В массиве X хранятся начало и конец списка соседей, находящиеся в массиве , то есть весь список соседей вершины J находится в массиве A с индекса $X[J]$ до $X[J+1]$, не включая его. Для иллюстрации на рис. 1 слева показан граф из 4 вершин, записанный с помощью матрицы смежности, а справа — в формате CSR.

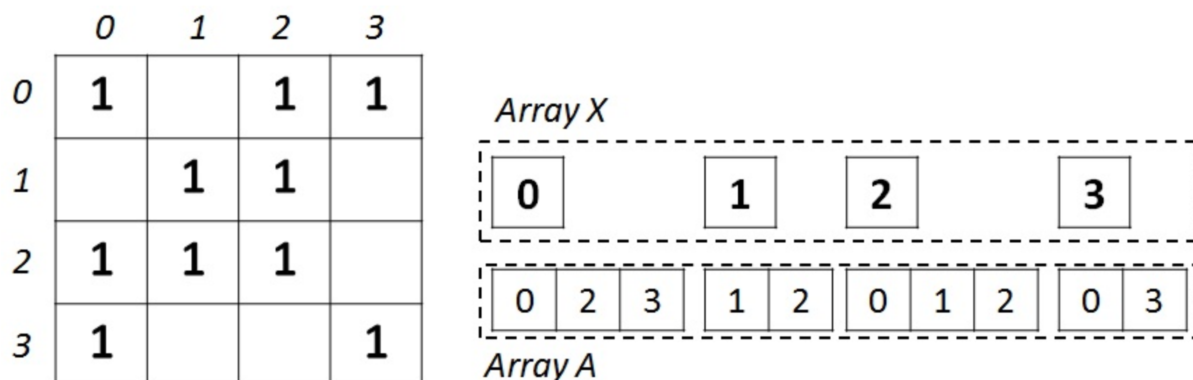


Рис. 1. Представление графа в виде матрицы смежности и в формате CSR

После преобразования графа в CSR-формат, необходимо проделать еще некоторую работу над входным графом для улучшения эффективности работы кэша и памяти вычислительных устройств. После выполнения описанных ниже преобразований, граф остается все в том же CSR-формате, но приобретет некоторые свойства, связанные с выполненными преобразованиями.

Введенные преобразования позволяют построить граф в оптимальном виде для большинства алгоритмов обхода графа в формате CSR. Добавление новой вершины в граф не приведет к выполнению всех преобразований заново, достаточно следовать введенным правилам и добавлять вершину так, чтобы общий порядок вершин не нарушался.

2.1. Локальная сортировка списка вершин

Для каждой вершины выполним сортировку по возрастанию ее списка соседей. В качестве ключа для сортировки будем использовать количество соседей для каждой сортируемой вершины. После выполнения данной сортировки, выполняя обход списка соседей у каждой вершины, мы будем обрабатывать сначала самые тяжелые вершины — вершины, имеющие большое количество соседей. Данную сортировку можно выполнять независимо для каждой вершины и параллельно. После выполнения данной сортировки, номера вершин графа в памяти не изменяются.

2.2. Глобальная сортировка списка вершин

Для списка всех вершин графа выполним сортировку по возрастанию. В качестве ключа будем использовать количество соседей для каждой из вершин. В отличие от локальной сортировки, данная сортировка требует перенумерации полученных вершин, так как меняется позиция вершины в списке. Процедура сортировки имеет сложность $O(N * log(N))$ и выполняется последовательно, а процедура перенумерации вершин может быть выполнена параллельно и по скорости работы сопоставима с временем копирования одного участка памяти в другой.

2.3. Перенумерация всех вершин графа

Занумеруем вершины графа таким образом, чтобы наиболее связные вершины имели наиболее близкие номера. Данная процедура устроена следующим образом. Сначала берется первая вершина из списка для перенумерации. Она получает номер 0. Затем все соседние вершины с рассматриваемой вершиной добавляются в очередь для перенумерации.

Следующая вершина из списка перенумерации получает номер 1 и так далее. В результате данной операции в каждой связной компоненте разница между максимальным и минимальным номером вершины будет наименьшей, что позволит лучшим образом использовать маленький объем кэша вычислительных устройств.

3. Реализация алгоритма

Алгоритм поиска в ширину в неориентированном графе устроен следующим образом. На входе подается начальная заранее неопределенная вершина в графе (корневая вершина для поиска). Алгоритм должен определить, на каком уровне, начиная от корневой вершины, находится каждая из вершин в графе. Под уровнем понимается минимальное количество ребер, которое необходимо преодолеть, чтобы добраться из корневой вершины в отличную от корневой вершину. Также для каждой из вершин, кроме корневой, необходимо определить вершину родителя. Так как у одной вершины может быть несколько родительских вершин, то в качестве ответа принимается любая из них.

Алгоритм поиска в ширину имеет несколько реализаций. Наиболее эффективная реализация — итерационный обход графа с синхронизацией по уровню. Каждый шаг представляет собой итерацию алгоритма, на которой информация с уровня J переносится на уровень $J+1$. Исходный код последовательного алгоритма представлен по ссылке [2].

Параллельная реализация базируется на гибридном алгоритме, состоящем из top-down (TD) и bottom-up (BU) процедур, который был предложен автором статьи по следующей ссылке [18]. Суть данного алгоритма заключается в следующем. Процедура TD позволяет обойти вершины графа в прямом порядке, то есть, перебирая вершины, мы рассматриваем связи $V1 \Rightarrow V2$ как родитель-потомок. Вторая процедура BU позволяет обойти вершины в обратном порядке, то есть, перебирая вершины, мы рассматриваем связи $V1 \Rightarrow V2$ как потомок-родитель.

Рассмотрим последовательную реализацию гибридного алгоритма TD-BU, исходный код которой показан на рис. 2. Для обработки графа вершин нам необходимо создать дополнительные два массива-очереди, которые будут содержать в себе набор вершин на текущем уровне — Q_{curr}, и набор вершин на следующем уровне — Q_{next}. Чтобы выполнять более быстрые проверки существования вершины в очереди, необходимо ввести массив посещенных вершин. Но так как нам в результате работы алгоритма необходимо получить информацию о том, на каком уровне располагается каждая из вершин, этот массив может быть использован и в качестве индикатора посещенных и размеченных вершин.

Основной цикл работы алгоритма состоит из последовательной обработки каждой вершины, находящейся в очереди Q_{curr}. Если в очереди Q_{curr} больше не остается вершин, то алгоритм останавливается и ответ получен.

В самом начале алгоритма начинает работать процедура TD, так как в очереди содержится всего одна вершина. В процедуре TD мы для каждой вершины V_i из очереди Q_{curr} просматриваем список соседних с этой вершиной V_k и добавляем в очередь Q_{next} тех из них, которые еще не были помечены как посещенные. Также все такие вершины V_k получают номер текущего уровня и родительскую вершину V_i. После завершения просмотра всех вершин из очереди Q_{curr} запускается процедура выбора следующего состояния, которая может либо остаться на процедуре TD для следующей итерации, либо сменить процедуру на BU.

```
void bfs_hybrid(G, N, M, Vstart) {
    Levels = (-1);  Parents = (N + 1); Qcurr+=Vstart; CountQ=lvl=1;
    while (CountQ) {
      CountQ = 0; vis = 0; inLvl = 0;
      if (state == TD)
        for (auto &Vi : Qcurr)
          for (auto &Vk : G.Edges(Vi)) {
            inLvl++;
            if (Levels[Vk] == -1)
              Qnext += Vk; Levels[Vk] = lvl; Parents[Vk] = Vi; vis++;
          }
      else if (state == BU)
        for (auto &Vi : G)
          if (Levels[Vi] == -1)
            for (auto &Vk : G.Edges(Vi)) {
              inLvl++;
              if (Levels[Vk] == lvl - 1)
                Qnext += Vi; Levels[Vi] = lvl;
                Parents[Vi] = Vk; vis++;
                break;
            }
      change_state(Qcurr, Qnext, vis, inLvl, G);
      Qcurr = Qnext; CountQ = Qnext.size();
    }
}
```

Рис. 2. Последовательный гибридный алгоритм BFS

В процедуре BU мы просматриваем вершины не из очереди Q_{curr}, а те вершины, которые еще не были помечены. Данная информация содержится в массиве уровней $Levels$. Если такие вершины V_i еще не были размечены, то мы проходим по всем ее соседям V_k и если эти вершины, которые являются родителями для V_i, находятся на предыдущем уровне, то вершина V_i попадает в очередь Q_{next}. В отличие от процедуры TD, в данной процедуре можно прервать просмотр соседних вершин V_k, так как нам достаточно найти любую родительскую вершину.

Если выполнять поиск только процедурой TD, то на последних итерациях алгоритма список вершин, который необходимо обработать, будет очень большим, а неразмеченных вершин будет достаточно мало. Тем самым процедура будет выполнять лишние действия и лишние обращения в память. Если выполнять поиск только процедурой BU, то на первых итерациях алгоритма будет достаточно много неразмеченных вершин и аналогично процедуре TD, будут выполнены лишние действия и лишние обращения в память.

Получается, что первая процедура эффективна на первых итерациях алгоритма BFS, а вторая — на последних. Ясно, что наибольший эффект будет достигнут, когда мы будем использовать обе процедуры. Для того, чтобы автоматически определять, когда необходимо выполнять переключение с одной процедуры на другую, воспользуемся алгоритмом (процедура change_state), который был предложен авторами той же статьи [18]. Данный алгоритм по информации о количестве обработанных вершин на двух соседних итерациях пытается понять характер поведения обхода. В алгоритме вводится два коэффициента,

которые позволяют настраивать переключение с одной процедуры на другую в зависимости от обрабатываемого графа.

```
state change_state(Qcurr, Qnext, vis, inLvl, G)
{
    new_state = old_state;
    factor = G.M / G.N / 2;
    if(Qcurr.size() < Qnext.size()) { // Growing phase
      if(old_state == TOP_DOWN) {
        if(inLvl < ((G.N - vis) * factor + G.N) / alpha)
          new_state = TOP_DOWN;
        else
          new_state = BOTTOM_UP;
      }
    } else { // Shrinking phase
      if (Qnext.size() < ((G.N - vis)*factor + G.N) / (factor*beta))
        new_state = TOP_DOWN;
      else
        new_state = BOTTOM_UP;
    }
    return new_state;
}
```

Рис. 3. Функция изменения состояния

Процедура смены состояния может переводить не только TD в BU, но и обратно BU в TD. Последняя смена состояния полезна в том случае, когда количество вершин, которые необходимо просмотреть, достаточно мало. Для этого вводится понятие нарастающего фронта и затухающего фронта размеченных и неразмеченных вершин. Следующий исходный код, представленный на рис. 3, выполняет смену состояния в зависимости от полученных характеристик на конкретной итерации обхода графа в зависимости от настроенных коэффициентов $alpha$ и $beta$. Данная функция может быть настроена на любой входной граф в зависимости от $factor$ (под $factor$ понимается средняя связность вершины графа).

Описанные выше концепции гибридной реализации алгоритма BFS применялись для параллельной реализации как для ЦПУ подобных систем, так и для ГПУ. Но есть некоторые отличия, которые будут рассмотрены далее.

3.1. Параллельная реализация на ЦПУ на общей памяти

Параллельная реализация для ЦПУ систем (Power 8+, Intel KNL и Intel x86) была выполнена с использованием OpenMP. Для запуска использовался один и тот же код, но для каждой платформы выполнялись свои настройки для директив OpenMP, например, задавались разные режимы балансировки нагрузки между нитями (schedule). Для реализации на ЦПУ с использованием OpenMP можно выполнить еще одно преобразование графа, а именно — сжатие списка вершин соседей.

Сжатие заключается в удалении незначащих нулевых битов каждого элемента из массива , причем данное преобразование делается отдельно для каждого диапазона $[X_i, X_{i+1})$. Происходит уплотнение элементов массива . Такое сжатие позволяет сократить

количество используемой памяти для хранения графа примерно на 30 % для больших графов порядка 2^{30} вершин и 2^{34} ребер. Для более маленьких графов экономия от такого преобразования пропорционально увеличивается в силу того, что уменьшается количество бит, которое занимает максимальный номер вершины в графе.

Такое преобразование графа накладывает некоторые ограничения на обработку вершин. Во-первых, все соседние вершины должны обрабатываться последовательно, так как они представляют собой сжатую, закодированную определенным образом последовательность элементов. Во-вторых, необходимо выполнять дополнительные действия по распаковке сжатых элементов. Данная процедура не является тривиальной и для ЦПУ Power8+ не позволила получить эффекта. Причиной может быть плохая оптимизация компилятора или отличная от Intel работа аппаратуры.

Для того, чтобы выполнять параллельно одну итерацию алгоритма, необходимо создать свои очереди Q_{th_next} для каждого потока OpenMP. А после выполнения всех циклов, выполнить объединение полученных очередей. Также необходимо локализовать все переменные, по которым есть редукционная зависимость. В качестве оптимизации процедура TD выполняется в последовательном режиме, если в очереди Q_{curr} количество вершин меньше заданного порога (например, меньше 300). Для графа разного размера, а также в зависимости от архитектуры, данный порог может иметь разные значения. Параллельные директивы располагались перед циклами $for\ (auto\ \&V_i\ in\ Q_{curr})$ в случае процедуры TD, и $for\ (auto\ \&V_i\ :\ G)$ в случае процедуры BU.

3.2. Параллельная реализация на ГПУ

Параллельная реализация для ГПУ была выполнена с использованием технологии CUDA. Реализация процедуры TD и BU существенно отличаются в случае использования ГПУ, так как существенно отличается алгоритм доступа к данным во время выполнения той или иной процедуры.

Процедура TD была реализована с помощью динамического параллелизма CUDA [23]. Данная возможность позволяет переложить некоторую работу, связанную с балансировкой нагрузки, на аппаратуру ГПУ. Каждая вершина V_i из очереди Q_{curr} может содержать абсолютно разное заранее не известное количество соседей. Из-за этого при прямом отображении всего цикла на набор нитей, возникает сильный дисбаланс нагрузки, так как CUDA позволяет использовать блок нитей фиксированного размера.

Описанная проблема решается путем использования динамического параллелизма. В начале запускаются мастер-нити. Данных нитей будет столько, сколько вершин содержится в очереди Q_{curr}. Затем каждая мастер-нить запускает столько дополнительных нитей, сколько соседей имеется у вершины V_i. Таким образом, количество используемых нитей формируется в динамике во время выполнения программы в зависимости от входных данных.

Данная процедура неудобна для выполнения на графическом процессоре из-за необходимости использовать динамический параллелизм. При большом суммарном количестве нитей, которые необходимо создать из мастер-нитей, возникают большие накладные расходы. Поэтому переключение на процедуру BU выполняется раньше, чем на ЦПУ.

Процедура BU является более благоприятной для выполнения на ГПУ, если проделать некоторые дополнительные преобразования данных. Данная процедура существенно

отличается от процедуры TD тем, что проход выполняется над всеми подряд идущими вершинами графа. Таким образом организованный цикл позволяет выполнить некоторую подготовку данных для хорошего доступа к памяти.

Преобразование заключается в следующем. Известно, что соседние нити одного варпа выполняют инструкции синхронно и параллельно. Для эффективного доступа к памяти требуется, чтобы соседние нити в варпе обращались к соседним ячейкам в памяти. Для примера положим количество нитей в варпе равным 2. Если каждой нити сопоставить одну вершину цикла $for\ (auto\ \&V_i\ :\ G)$, то во время обработки соседей в цикле $for\ (auto\ \&V_k\ :\ G.Edges(V_i))$ каждая нить будет обращаться в свою область памяти, что негативно скажется на производительности, так как соседние нити будут обрабатывать далеко расположенные ячейки в памяти. Для того, чтобы исправить положение, перемешаем элементы массива A таким образом, чтобы доступ к первым двум соседям из V_0 и V_1 осуществлялся наилучшим образом — соседние элементы располагались в соседних ячейках в памяти. Далее, в памяти таким же образом будут лежать вторые, третьи и т.д. элементы.

```
__global__ void bu_align( ... ) {
    idx = blockDim.x * blockIdx.x + threadIdx.x;
    countQNext = 0; inlvl = 0;
    for(i = idx; i < N; i += stride)
      if (levels[i] == 0) {
        start_k = rowsIndices[i];
        end_k = rowsIndices[i + N];
        for(k = start_k; k < start_k + 32 * end_k; k += 32) {
          inlvl++;
          vertex_id_t endk = endV[k];
          if (levels[endk] == lvl - 1) {
            parents[i] = endk;
            levels_out[i] = lvl;
            countQNext++;
            break;
          }
        }
      }
    atomicAdd(red_qnext, countQNext);
    atomicAdd(red_lvl, inlvl);
}
```

Рис. 4. Параллельное ядро для процедуры BU

Данное правило выравнивания применяется для группы нитей варпа (32 нити): выполняется перемешивание соседей — сначала располагаются первые 32 элемента, затем вторые 32 элемента и т.д. Так как граф отсортирован по убыванию количества соседей, то группы вершин, которые располагаются рядом, будут иметь достаточно близкое количество вершин соседей.

Перемешивать таким способом элементы можно не во всем графе, так как во время работы процедуры BU во внутреннем цикле есть досрочный выход. Описанные выше глобальная и локальная сортировки вершин позволяют выходить из данного цикла достаточно рано. Поэтому перемешивается только 40 % всех вершин графа. Данное преобразование требует дополнительной памяти для хранения перемешанного графа, зато

мы получаем заметный прирост в производительности. На рис. 4 представлен исходный код ядра для процедуры BU с использованием перемешанного расположения элементов в памяти.

4. Анализ полученных результатов

Тестированные реализованного алгоритма BFS производилось на четырех различных платформах: Intel Xeon Phi (Xeon KNL 7250) [24], Intel x86 (Xeon E5 2699 v3) [25], IBM Power8+ (Power 8+ s822lc) и GPU NVidia Tesla P100 [26]. Интересующие для сравнения характеристики данных устройств представлены в табл. 1 соответственно порядку их перечисления.

Таблица 1

Технические характеристики устройств

Ядер / Потоков	Частота, ГГц	RAM, GB/s	Макс. TDP, Вт	Транз., млрд
68 / 272	1,4	115 / 400	215	8
18 / 36	2,3	68	145	5,69
10 / 80	3,5	205	270	6
56 / 3584	1,4	40 / 700	300	15,3

В последнее время производители все больше задумываются о пропускной способности памяти. Как следствие этому, появляются различные решения проблемы медленного доступа к оперативной памяти. Среди рассматриваемых платформ две имеют двухуровневую структуру оперативной памяти.

Первая из них — Intel KNL, содержит быструю память на кристалле, скорость доступа к которой порядка 400 ГБ/с, и более медленную привычную нам DDR4, скорость доступа к которой не более 115 ГБ/с. Быстрая память имеет достаточно маленький размер — всего 16 ГБ, в то время как обычная память может быть до 384 ГБ. На тестируемом сервере было установлено 96 ГБ памяти такой памяти. Вторая платформа с гибридным решением — Power + NVidia Tesla. Данное решение базируется на новой технологии NVlink [27], которая позволяет иметь доступ к обычной памяти ЦПУ на скорости 40 ГБ/с, в то время как доступ к быстрой памяти осуществляется на скорости 700 ГБ/с. Количество быстрой памяти такое же, как и в Intel KNL — 16 ГБ.

Данные решения схожи с точки зрения организации — имеется быстрая память маленького размера, и медленная память большого размера. Сценарий использования двухуровневой памяти при обработки больших графов очевиден: быстрая память используется для хранения результата и промежуточных массивов, размеры которых достаточно малы по сравнению с входными данным, а исходный граф читается из медленной памяти.

С точки зрения реализации пользователю доступны следующие средства. Для Intel KNL достаточно использовать другие функции выделения памяти — hbm_malloc, вместо привычного malloc. Если программа использовала операторы malloc, то достаточно объявить один define для того, чтобы использовать данную возможность. Для NVidia Tesla необходимо использовать также другие функции выделения памяти — вместо cudaMalloc использовать cudaMallocHost. Данные модификации кода являются достаточными и не требуют каких-либо модификаций в вычислительной части программы.

Эксперименты проводились для графов разного размера, начиная от 2^{25} (4 ГБ) и заканчивая 2^{30} (128 ГБ). Средняя степень связности и тип графа брались из генератора графа для рейтинга Graph500. Данный генератор создает графы Кронекера со средней степенью связности 16 и коэффициентами $A = 0,57$, $B = 0,19$, $C = 0,19$.

Данного вида графы используются всеми участниками таблицы рейтинга, что позволяет корректно сравнивать реализации между собой. Значение производительности считается по метрике TEPS для таблицы Graph500 и TEPS / w для таблицы GreenGraph500. Для вычисления данной характеристики выполняется 64 запуска алгоритма BFS из разных стартовых вершин и берется среднее значение. Для вычисления потребления алгоритма берется текущее потребление системы в момент работы алгоритма с учетом потребления оперативной памяти.

Табл. 2 иллюстрирует полученную производительность в GTEPS на всех тестируемых графах. В таблице указываются два значения — минимальная / максимальная достигнутая производительность на каждом из графов. Также в случае использования Intel KNL были получены результаты выполнения алгоритма при использовании только памяти DDR4. К сожалению, даже при использовании всех алгоритмов сжатия данных, не удалось запустить на предоставленном сервере граф с 2^{30} вершинами на Intel KNL. Но учитывая стабильность работы Intel процессоров и технологичность Intel компиляторов, можно предположить, что производительность не изменится при увеличении размера графа (как это можно видеть для Intel Xeon E5).

Таблица 2

Полученная производительность в GTEPS

Размер графа	2^{25}	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}
Xeon KNL 7250	10,7/30,6	12,9/41	8,4/43,3	4,6/40,2	6,2/42,6	N/A
Xeon KNL 7250 DDR4	6,7/25,2	4,3/27	4,9/28,4	5,7/31,6	10,8/38,8	N/A
Xeon E5 2699 v3	11/16,5	11,8/17,3	12,7/18,5	13,1/18,3	12,1/18,0	12,4/21,1
IBM Power 8+ s822lc	8,8/22,5	9/23,3	7,9/23,4	10,4/23,7	10,1/24,6	7,59/14,8
NVidia Tesla P100	41/282	99/333	34/324	5/274	7,2/61	6,5/52

На рис. 5 отображена средняя производительность протестированных платформ. Можно заметить, что Power 8+ показал не очень хорошую стабильность при переходе с графа размером 64 ГБ на 128 ГБ. Возможно, это связано с тем, что использовался двух сокетный узел из двух аналогичных процессоров, причем у каждого процессора было по 128 ГБ памяти. И при обработке большего графа часть данных размещалось в памяти, не принадлежащей сокету. На графике также не отображена производительность Tesla P100 на более маленьких графах, так как разница между самым быстрым ЦПУ устройством и ГПУ составляет примерно 10 раз. Данное ускорение резко сокращается, когда графы становятся настолько большими, что не помещаются в кэш и доступ к графу осуществляется через NVlink. Но, несмотря на данное ограничение, производительность ГПУ все равно остается больше всех ЦПУ устройств. Такое поведение объясняется тем, что CUDA позволяет лучше контролировать вычисления и доступ к памяти, а также лучшей приспособленностью графических процессоров к параллельным вычислениям.

Табл. 3 иллюстрирует полученную производительность в GTEPS /w на всех тестируемых графах. В таблице указываются среднее энергопотребление при средней

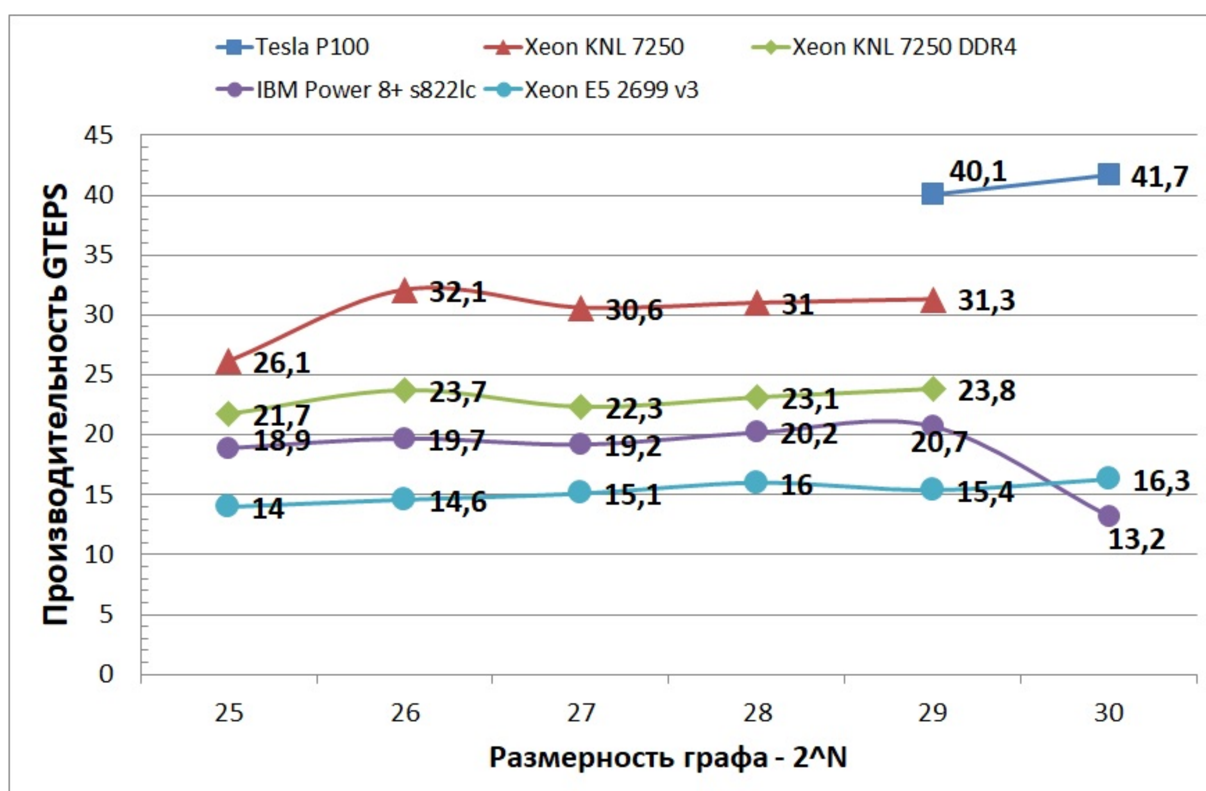


Рис. 5. Сравнение средней производительности

достигнутой производительности на каждом из графов. Резкое падение производительности и энергоэффективности при переходе с графа 2^{28} на граф 2^{29} на NVidia Tesla P100 объясняется тем, что быстрой памяти не хватает, чтобы поместить выровненную часть графа, к которой осуществляется наиболее частый доступ. В случае использования большего количества памяти (например, 32 ГБ) и увеличенного канала связи с ЦПУ NVlink 2.0 можно существенно повысить эффективность обработки графов большого размера.

Таблица 3

Полученная энергоэффективность в MTEPS / w

Размер графа	2^{25}	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}
Xeon KNL 7250	121,4	149,3	142,33	136,56	130,96	N/A
Xeon E5 2699 v3	95,56	98,65	100	101,9	91,12	84,46
IBM Power 8+ s822lc	93,8	97,04	93,2	95,28	92,41	53,23
NVidia Tesla P100	1228,57	1165,71	1235,96	1016,57	195,61	177,45

Заключение

В результате проделанной работы были реализованы два параллельных алгоритма BFS для ЦПУ подобных систем и для ГПУ. Было выполнено исследование производительности и энергоэффективности реализованных алгоритмов на различных платформах, таких как IBM Power8+, Intel x86, Intel Xeon Phi (KNL) и NVidia Tesla P100. Данные платформы имеют различные архитектурные особенности. Несмотря на это, первые три из них очень похожи по строению. Благодаря этому на этих платформах можно запускать OpenMP

приложения без каких-либо существенных изменений. Наоборот, архитектура ГПУ сильно отличается от ЦПУ подобных платформ и использует другую концепцию для реализации вычислительного кода — архитектуру CUDA.

Были рассмотрены графы, которые получаются после генератора для рейтинга Graph500. Была исследована производительность каждой из архитектур на двух классах данных. К первому классу относятся графы, которые помещаются в наиболее быструю память вычислителя. Ко второму классу относятся большие графы, которые нельзя поместить в быструю память целиком. Для демонстрации энергоэффективности использовались метрики GreenGraph500. Минимальный граф, который учитывается в рейтинге GreenGraph500 в классе больших данных, содержит в себе 2^{30} вершин и 2^{34} ребер и занимает в исходном виде 128 ГБ памяти. В классе же малых данных допускается граф любого размера до 2^{30} вершин и 2^{34} ребер, причем в качестве результата принимается наиболее большой граф, который удалось вместить в память.

На данный момент среди всех одноузловых систем в рейтинге Graph500 и GreenGraph500 полученная реализация на NVidia Tesla P100 занимает лидирующие позиции как в классе малых данных (с производительностью 220 GTEPS и эффективностью 1235,96 MTEPS/w), так и в классе больших данных (с производительностью 41,7 GTEPS и эффективностью 177,45 MTEPS/w). Такая высокая энергоэффективность и скорость работы на больших графах была достигнута благодаря новой технологии NVLink, которая связывает ГПУ и ЦПУ между собой и доступна на данный момент только в серверах компании IBM с ЦПУ Power8+.

В дальнейшем планируется исследовать возможность выполнения данного алгоритма на новой архитектуре NVidia Volta с использованием улучшенной технологии NVlink 2.0, а также планируется исследовать параллельную реализацию на нескольких ГПУ.

Литература

1. Cherkassky B.V., Goldberg A.V., Radzik T. Shortest Paths Algorithms: Theory and Experimental Evaluation // Math. Program. 1996. Vol. 73. P. 129–174. DOI: 10.1007/BF02592101.
2. Moore E.F. The Shortest Path through a Maze // Proceedings of the International Symposium on the Theory of Switching (2–5 April 1957). Harvard University Press, 1959. P. 285–292.
3. Chazelle B.A. Minimum Spanning Tree Algorithm with Inverse-ackermann Type Complexity // Journal of the ACM. 2000. Vol. 47, No. 6. P. 1028–1047. DOI: 10.1145/355541.355562.
4. Колганов А.С. Параллельная реализация алгоритма поиска минимальных остовных деревьев с использованием центрального и графического процессоров // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2016. Т. 5, № 3. С. 5–19. DOI: 10.14529/cmse160301.
5. Рейтинг Graph500. URL: `http://graph500.org/` (дата обращения 01.12.2017)
6. Рейтинг GreenGraph500. URL: `http://green.graph500.org/` (дата обращения 01.12.2017)
7. Cormen T., Leiserson, C., Rivest R. Introduction to Algorithms. MIT Press, Cambridge. 1990.
8. Edmonds J., Karp R.M. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems // Journal of the ACM. 1972. Vol. 19, No. 2. P. 248–264. DOI: 10.1007/3-540-36478-1_4.

9. Brandes U. A Faster Algorithm for Betweenness Centrality // J. Math. Sociol. 2001. Vol. 25, No. 2. P. 163–177. DOI: 10.1080/0022250X.2001.9990249.

10. Frasca M., Madduri K., Raghavan P. NUMA-Aware Graph Mining Techniques for Performance and Energy Efficiency // Proc. ACM/IEEE Int. Conf. High Performance Computing, Networking, Storage and Analysis (Salt Lake City, Utah, USA, November 10–16, 2012). P. 1–11. DOI: 110.1109/SC.2012.81.

11. Girvan M., Newman M.E. Community Structure in Social and Biological Networks // Proc. Natl. Acad. Sci. (USA, June 11, 2002). Vol. 99, No. 12. P. 7821–7826. DOI: 10.1073/pnas.122653799.

12. Dijkstra E.W. A Note on Two Problems in Connexion with Graphs // Numerische Mathematik. 1959. Vol. 1, No. 1. P. 269–271. DOI: 10.1007/BF01386390.

13. Рейтинг Top500. URL: https://www.top500.org/ (дата обращения 01.12.2017)

14. Bader D.A., Madduri K. Designing Multithreaded Algorithms for Breadth-first Search and St-connectivity on the Cray MTA-2. 2006. P. 523–530. DOI: 10.1109/ICPP.2006.34.

15. Korf R.E., Schultze P. Large-scale Parallel Breadth-first Search // AAAI. 2005. P. 1380–1385.

16. Yoo A., Chow E., Henderson K., McLendon W., Hendrickson B., Catalyurek U. A Scalable Distributed Parallel Breadth-first Search Algorithm on BlueGene/L // Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (Seattle, Washington, USA, November 12–18, 2005). DOI: 10.1109/SC.2005.4.

17. Zhang Y., Hansen E.A. Parallel Breadth-first Heuristic Search on a Shared-memory Architecture // AAAI Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications. 2006.

18. Yasui Y., Fujisawa K., Sato Y. Fast and Energy-efficient Breadth-First Search on a Single NUMA System // Lecture Notes in Computer Science. Vol. 8488. P. 365–381. DOI: 10.1007/978-3-319-07518-1_23.

19. Hiragushi T., Takahashi D. Efficient Hybrid Breadth-First Search on GPUs // Lecture Notes in Computer Science. Vol. 8286. P. 40–50. DOI: 10.1007/978-3-319-03889-6_5.

20. Merrill D., Garland M., Grimshaw A. Scalable GPU Graph Traversal // Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (New Orleans, Louisiana, USA, February 25–29, 2012). P. 117–128. DOI: 10.1145/2370036.2145832.

21. Chakrabarti D., Zhan Y., Faloutsos C. R-MAT: A Recursive Model for Graph Mining // Proceedings of the 2004 SIAM International Conference on Data Mining (Florida, USA, April 22–24, 2004). P. 442–446. DOI: 10.1137/1.9781611972740.43.

22. Pissanetzky S. Sparse Matrix Technology. Academic Press. 1984.

23. Динамический параллелизм в CUDA. URL: https://devblogs.nvidia.com/parallelforall/cuda-dynamic-parallelism-api-principles/ (дата обращения 01.12.2017)

24. Спецификация процессора Intel Xeon Phi 7250. URL: https://ark.intel.com/ru/products/94035/Intel-Xeon-Phi-Processor-7250-16GB-1_40-GHz-68-core (дата обращения 01.12.2017)

25. Спецификация процессора Intel Xeon E5 2699 v3. URL: https://ark.intel.com/ru/products/81061/Intel-Xeon-Processor-E5-2699-v3-45M-Cache-2_30-GHz (дата обращения 01.12.2017)

26. Спецификация процессоров IBM Power8 и NVidia Tesla P100. URL: https://www-03.ibm.com/systems/ru/power/hardware/s822lc-hpc/ (дата обращения 01.12.2017)

27. Технология NVLink. URL: http://www.nvidia.com/object/nvlink.html (дата обращения 01.12.2017)

Колганов Александр Сергеевич, младший научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация), аспирант, кафедра системного программирования, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

DOI: 10.14529/cmse180201

THE FASTEST AND ENERGY-EFFICIENT BREADTH-FIRST SEARCH ALGORITHM ON A SINGLE NODE WITH VARIOUS PARALLEL ARCHITECTURES ACCORDING TO GRAPH500

© 2018 A.S. Kolganov

Keldysh Institute of Applied Mathematics
(Miusskaya sq. 4, Moscow, 125047 Russia),
Lomonosov Moscow State Universisty
(GSP-1, Leninskie Gory 1, Moscow, 119991 Russia)
E-mail: alexander.k.s@mail.ru
Received: 08.04.2018

The breadth-first search algorithm is one of the basic algorithms for graph traversing and is used in many other algorithms of high-level graph analysis. BFS is characterized with intensive irregular memory accesses and a random data dependency, which greatly complicates its parallelization to all existing architectures. The paper considers the implementation of the BFS algorithm (the core benchmark of the Graph500 rating) for processing large graphs on different architectures: Intel x86, IBM Power8+, Intel KNL and NVidia GPU. Algorithms for implementing breadth-first search will be considered, such as top-down traverse, bottom-up traverse, and a hybrid traverse that includes both top-down and bottom-up traverses. The features of the algorithm implementation on shared memory will be shown, as well as graph transformations (local sorting of graph vertices, global sorting of graph vertices, renumbering of all graph vertexes, compressed representation of graph vertices), which allow achieving the best performance and energy-efficiency in this algorithm among all single-node systems of Graph500 and GreenGraph500 ratings.

Keywords: parallel graph processing, BFS, CUDA, Power8, KNL, Graph500.

FOR CITATION

Kolganov A.S. The Fastest and Energy-Efficient Breadth-First Search Algorithm on a Single Node with Various Parallel Architectures According to Graph500. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2018. vol. 7, no. 2. pp. 5–21. (in Russian) DOI: 10.14529/cmse180201.

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

References

1. Cherkassky B.V., Goldberg A.V., Radzik T. Shortest Paths Algorithms: Theory and Experimental Evaluation. *Math. Program.* 1996. vol. 73. pp. 129–174. DOI: 10.1007/BF02592101.
2. Moore E.F. The Shortest Path through a Maze. Proceedings of the International Symposium on the Theory of Switching (April 2–5, 1957). Harvard University Press, 1959. pp. 285–292.
3. Chazelle B.A., Minimum Spanning Tree Algorithm with Inverse-ackermann Type Complexity. *Journal of the ACM.* 2000. vol. 47, no. 6. pp. 1028–1047.
4. Kolganov A.S. Parallel Implementation of Minimum Spanning Tree Algorithm on CPU and GPU. *Vestnik Yuzho-Uralskogo gosudarstvennogo universiteta. Seriya: Vychislitel'naja matematika i informatika* [Bulletin of South Ural State University. Series: Computational Mathematics and Software Engineering]. 2016. vol. 5, no. 3. pp. 5–19. DOI: 10.14529/cmse160301. (in Russian)
5. Graph500. Available at: `http://graph500.org/` (accessed 01.12.2017)
6. GreenGraph500. Available at: `http://green.graph500.org/` (accessed 01.12.2017)
7. Cormen T., Leiserson C., Rivest R. Introduction to Algorithms. MIT Press, Cambridge. 1990.
8. Edmonds J., Karp R.M. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *Journal of the ACM.* 1972. vol. 19, no. 2. pp. 248–264. DOI: 10.1007/3-540-36478-1_4.
9. Brandes U. A Faster Algorithm for Betweenness Centrality. *J. Math. Sociol.* 2001. vol. 25, no. 2. pp. 163–177. DOI: 10.1080/0022250X.2001.9990249.
10. Frasca M., Madduri K., Raghavan P. NUMA-Aware Graph Mining Techniques for Performance and Energy Efficiency. Proc. ACM/IEEE Int. Conf. High Performance Computing, Networking, Storage and Analysis (Salt Lake City, Utah, USA, November 10–16, 2012). pp. 1–11. DOI: 110.1109/SC.2012.81.
11. Girvan M., Newman M.E. Community Structure in Social and Biological Networks. Proc. Natl. Acad. Sci. (USA, June 11, 2002). vol. 99, no. 12. pp. 7821–7826. DOI: 10.1073/pnas.122653799.
12. Dijkstra E.W. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik.* 1959. vol. 1, no. 1. pp. 269–271. DOI: 10.1007/BF01386390.
13. Top500. Available at: `https://www.top500.org/` (accessed 01.12.2017)
14. Bader D.A., Madduri K. Designing Multithreaded Algorithms for Breadth-first Search and St-connectivity on the Cray MTA-2. 2006. pp. 523–530. DOI: 10.1109/ICPP.2006.34.
15. Korf R.E., Schultze P. Large-scale Parallel Breadth-first Search. *AAAI.* 2005. pp. 1380–1385.
16. Yoo A., Chow E., Henderson K., McLendon W., Hendrickson B., Catalyurek U. A Scalable Distributed Parallel Breadth-first Search Algorithm on BlueGene/L. Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (Seattle, Washington, USA, November 12–18, 2005). DOI: 10.1109/SC.2005.4.

17. Zhang Y., Hansen E.A. Parallel Breadth-first Heuristic Search on a Shared-memory Architecture. AAAI Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications. 2006.
18. Yasui Y., Fujisawa K., Sato Y. Fast and Energy-efficient Breadth-First Search on a Single NUMA System. *Lecture Notes in Computer Science*. 2014. vol. 8488. pp. 365–381. DOI: 10.1007/978-3-319-07518-1_23.
19. Hiragushi T., Takahashi D. Efficient Hybrid Breadth-First Search on GPUs. *Lecture Notes in Computer Science*. 2013. vol. 8286. pp. 40–50. DOI: 10.1007/978-3-319-03889-6_5.
20. Merrill D., Garland M., Grimshaw A. Scalable GPU Graph Traversal. Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (New Orleans, Louisiana, USA, February 25–29, 2012). pp. 117–128. DOI: 10.1145/2370036.2145832.
21. Chakrabarti D., Zhan Y., Faloutsos C. R-MAT: A Recursive Model for Graph Mining. Proceedings of the 2004 SIAM International Conference on Data Mining (Florida, USA, April 22–24, 2004). pp. 442–446. DOI: 10.1137/1.9781611972740.43.
22. Pissanetzky S. Sparse Matrix Technology. Academic Press. 1984.
23. CUDA Dynamic Parallelism. Available at: `https://devblogs.nvidia.com/parallelforall/cuda-dynamic-parallelism-api-principles/` (accessed 01.12.2017)
24. Intel Xeon Phi 7250. Available at: `https://ark.intel.com/ru/products/94035/Intel-Xeon-Phi-Processor-7250-16GB-1_40-GHz-68-core` (accessed 01.12.2017)
25. Intel Xeon E5 2699 v3. Available at: `https://ark.intel.com/ru/products/81061/Intel-Xeon-Processor-E5-2699-v3-45M-Cache-2_30-GHz` (accessed 01.12.2017)
26. IBM Power8 and NVidia Tesla P100. Available at: `https://www-03.ibm.com/systems/ru/power/hardware/s822lc-hpc/` (accessed 01.12.2017)
27. Nvidia NVLink. Available at: `http://www.nvidia.com/object/nvlink.html` (accessed 01.12.2017)

УДК 519.6 DOI: 10.14529/cmse180202

ДИНАМИКА ИЗМЕНЕНИЯ ОБЛАСТЕЙ УСТОЙЧИВОСТИ ДИСКРЕТНЫХ МОДЕЛЕЙ НЕЙРОННЫХ СЕТЕЙ ТИПА SMALL WORLD ПРИ ИЗМЕНЕНИИ ЧИСЛОВЫХ ХАРАКТЕРИСТИК ГРАФА СЕТИ

© 2018 С.А. Иванов[1], М.М. Кипнис[2]

[1]*Южно-Уральский государственный университет*

(454080 Челябинск, пр. им. В.И. Ленина, д. 76)

[2]*Южно-Уральский государственный гуманитарно-педагогический университет*

(454080 Челябинск, пр. им. В.И. Ленина, д. 69)

E-mail: saivanov@susu.ru, mmkipnis@gmail.com

Поступила в редакцию: 17.10.2017

В статье дано описание дискретных моделей нейронных сетей со связями типа small world с вероятностью перенаправления связей внутри сети p, изменяющейся от 0 до 1. При значении $p = 0$ получим модель регулярной нейронной сети. Регулярной нейронной сетью выступает кольцевая нейронная сеть, в которой каждый нейрон взаимодействует с несколькими соседями по кольцу. При значении $p = 1$ получим модель, нейроны которой случайным образом соединены с другими нейронами сети без образования изолированных нейронов. Рассматриваемые нейронные сети имеют широкое применение при моделировании различные нейронных структур в живых организмах, например, гипокамп мозга млекопитающих. В работе проведено исследование динамики изменения областей устойчивости рассматриваемых нейронных сетей в случае изменения вероятности перенаправления связей, коэффициента кластеризации и длины кратчайшего пути в среднем графа нейронной сети. В ходе численных экспериментов были построены области устойчивости исследуемых моделей нейронных сетей для различных параметров сети и сделан вывод об увеличении области устойчивости при одновременном уменьшении длины кратчайшего пути в среднем и коэффициента кластеризации графа сети.

Ключевые слова: дискретные модели Ваттса—Строгаца, small world, устойчивость.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Иванов С.А., Кипнис М.М. Динамика изменения областей устойчивости дискретных моделей нейронных сетей типа small world при изменении числовых характеристик графа сети // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2018. Т. 7, № 2. С. 22–31. DOI: 10.14529/cmse180202.

Введение

Искусственная нейронная сеть — это математическая модель, построенная на основе организации и функционирования сетей нервных клеток живых организмов. В данной статье будет рассмотрена модель нейронной сети [1], в которой введена вероятность перенаправления связей между нейронами внутри сети p. В случае $p = 0$ модель является регулярной, в случае $p = 1$ нейронная сеть представляет собой полностью случайный граф без изолированных вершин. Связи типа small world могут быть представлены моделями нейронных сетей Ваттса—Строгаца при условии близких к нулю значений параметра p. Некоторые структуры внутри живых организмов можно представить в виде нейронных сетей типа small world [2, 3].

Использование моделей отделов головного мозга с использованием нейронных сетей типа small world находит свое применение в нейрохирургии [4]. В статье [2] сила воздействия нейрона на соседа и сила обратного воздействия любых двух нейронов в сети одинаковая. В связи с этим анализ устойчивости сети основан только на анализе графа связей в сети. Подобный анализ не дает ответ об устойчивости сетей с несимметричными силами воздействия нейронов между собой в сети. В статье [3] проведено исследование поведения нейронной сети со случайными силами взаимодействия, имеющими Гауссово распределение. Особо отметим отсутствие запаздывания в работах [2, 3]. На сегодняшний день проводится большое количество исследований, связанных с устойчивостью отделов головного мозга в целом и его частей [5, 6].

Гипокамп мозга млекопитающих играет важную роль в консолидации информации из кратковременной памяти в долговременную память, и в пространственной памяти, что позволяет осуществлять ориентацию в пространстве. Стремление мозга к устойчивому положению в ходе своей работы является необходимым требованием функционирования мозга. Таким образом возникает необходимость корректирования параметров нейронной сети для стремления всей сети к устойчивому положению.

Данная работа преследует целью построение областей устойчивости больших нейронных сетей, моделирующих структуру связей нейронов гипокампа человеческого мозга [7], и изучение динамики изменения этих областей при изменении числовых характеристик графа нейронной сети. Эту модель можно отнести к моделям нейронных сетей, построенных по типу small world. Рассматриваемая модель нейронной сети обладает несимметричными силами воздействия нейронов в сети: сила воздействия одного нейрона на другой равна a, а сила обратного воздействия равна b.

Интерпретацией знака коэффициентов a и b в исследуемой модели нейронной сети может служить возбуждающий или тормозящий сигнал идущий по нейрону. В исследуемой модели оказалось, что при одинаковом числе нейронов и общем числе связей в сети коэффициент кластеризации и длина кратчайшего пути в среднем графа оказывают влияние на область устойчивости сети. В ходе численных экспериментов было выявлено улучшение устойчивости нейронной сети при перенаправлении связей нейронной сети таким образом, чтобы полученный граф имел меньшую длину кратчайшего пути в среднем и меньший коэффициент кластеризации.

Статья имеет следующую структуру. Раздел 1 посвящен описанию исследуемой модели нейронной сети. В разделе 2 вводится уравнение и метод исследования устойчивости исследуемой нейронной сети. Вводятся определения числовых характеристик графа нейронной сети. Дано описание проведенных численных экспериментов диагностирования устойчивости модели, построенных областей устойчивости, основанных на алгоритмах и методах, ранее полученных автором, и формулируются выводы о динамике устойчивости сети при изменении числовых характеристик графа нейронной сети. В заключении резюмируется проведенная работа и обозначаются направления будущих исследований.

1. Построение модели

Большинство биологических нейронов имеют схожее строение и свойства с двигательными нейронами спинного мозга млекопитающих [8, 9]. Запаздывание в уравнение нейронной сети введено благодаря наличию разницы в скорости передачи нервных импульсов в зависимости от типа проводящих волокон.

Исследуемая модель дискретна и включает в себя запаздывание в реакции на соседних нейронов в сети. Модель имеет специальную матрицу, которая предназначена для описания системы связей внутри сети. В работе [10] показано, как формируются связи в модели.

Допустим, что каждому нейрону сети, состоящей из n нейронов, поставлен в соответствие номер $1, 2, \ldots, n$. Внутреннее состояние нейрона в сети с номером j ($1 \leq j \leq n$) в дискретный момент времени s ($s = 1, 2, \ldots$) описывается его выходным сигналом x_s^j. Состояние всей нейронной сети в момент времени s можно представить в виде вектора $x_s = (x_s^1, \ldots, x_s^n)^T$. Если в нейронной сети каждый нейрон окажется изолированным, то динамику изменения состояния нейронной сети можно описать уравнением $x_s = \alpha x_{s-r}$ ($-1 \leq \alpha \leq 1$), $r \in \mathbb{Z}_+$. Это позволяет гарантировать устойчивость нулевого стационарного состояния сети при условии отсутствия связей между любыми нейронами сети.

Коэффициент α будем называть коэффициентом демпфирования собственных колебаний нейронов в сети, число r — запаздыванием в реакции нейрона на свой собственный сигнал. Введем обозначение матрицы $B = (\beta_{ij})_{i,j=1}^n$ запаздывающих взаимодействий нейронов на соседей. Здесь значение β_{ij} это сила воздействия j-го нейрона на i-й. Граф исследуемой нейронной сети состоит из вершин с номерами $1, 2, \ldots, n$ и множество E составленное из направленных дуг, так что $(i, j) \in E$ если и только если $\beta_{ij} \neq 0$. Мы считаем равными нулю все диагональные элементы матрицы B, что гарантирует отсутствие петель в представленном графе.

Пример 1. Приведем пример матрицы запаздывающих взаимодействий B для сети, состоящей из 6 нейронов, нулевой вероятности перенаправления связей (регулярной сети), каждый нейрон которой соединен с двумя соседями по часовой стрелке и с двумя - против.

$$B = \begin{pmatrix} 0 & b & b & 0 & a & a \\ a & 0 & b & b & 0 & a \\ a & a & 0 & b & b & 0 \\ 0 & a & a & 0 & b & b \\ b & 0 & a & a & 0 & b \\ b & b & 0 & a & a & 0 \end{pmatrix}.$$

Положим, что любые два нейрона взаимодействуют между собой с запаздыванием на m тактов $s > m > r$. Силы взаимодействия между нейронами сети, реакция нейрона на самого себя и запаздывания в реакции нейронов в сети будут определять начальное состояние всей сети. В результате получили рекуррентное уравнение динамики исследуемой нейронной сети (см. [11, 12]).

$$x_s = \alpha\, x_{s-r} + B\, x_{s-m}, \quad s = 1, 2, \ldots \tag{1}$$

2. Исследование устойчивости нейронной сети

Уравнение динамики нейронной сети представляет собой линейное матричное разностное уравнение (1), которое зависит от дискретного времени s. Данное уравнение назовем устойчивым при любом значении s, если любое его решения ограничено, и асимптотически устойчивым, если любое его решение стремится к нулю при $s \to \infty$. Уравнение

$$\det(\lambda^m I - \alpha \lambda^{m-r} I - B) = 0 \tag{2}$$

степени $n \times m$ является характеристическим для (1). В (2) I — единичная матрица размера $n \times n$. Уравнение (1) назовем асимптотически устойчивым тогда и только тогда, когда все корни его характеристического уравнения лежат строго внутри единичного круга комплексной плоскости. Метод конусов устойчивости и алгоритмы определения устойчивости уравнения (1) представлены в работах [13, 14].

Важным вопросом является зависимость устойчивости исследуемой модели от параметров исследуемой модели p $(0 \leq p \leq 1)$, длины кратчайшего пути в среднем и коэффициента кластеризации. Метод конусов устойчивости лежит в основе проведенных численных экспериментов для поиска границы области устойчивости в пространстве параметров a, b.

Построение областей устойчивости нейронной сети в виде замкнутой линии в пространстве параметров a, b потребует определить массив точек, которые лежат на границе между неустойчивым и устойчивым состояниями сети. В работе [11] показано, что область устойчивости регулярной сети является замкнутой. Алгоритм построения областей нейронных сетей типа small world приведен в [10].

2.1. Числовые характеристики графа нейронной сети

Длина кратчайшего пути в среднем L измеряет расстояние между двумя узлами. L определяется числом ребер в кратчайшем пути между двумя вершинами. Чтобы посчитать длину кратчайшего пути в среднем во всей сети, нужно просуммировать кратчайшие длины путей между всеми парами вершин, а полученную сумму разделить на произведение $n(n-1)$, где n — число вершин в сети. Длина кратчайшего пути в среднем вычисляется по формуле:

$$L = \frac{1}{n(n-1)} \sum_{i,j \in n, i \neq j} d_{ij}, \tag{3}$$

где n — число нейронов в сети, d_{ij} — расстояние от узла i до узла j.

Коэффициент кластеризации C определяет, насколько близки соседи вершины к состоянию клики, т.е. подмножество вершин графа, являющихся соседями данной вершины, таково, что каждые две вершины в нем соединены ребром. Коэффициент кластеризации определяется следующим образом: предположим что вершина v имеет k_v соседей. Тогда число ребер между ними меньше или равно $\frac{(k_v(k_v-1))}{2}$, равенство достигается, когда все соседи вершины v соединены между собой. Обозначим как C_v число ребер, которые реально существуют из возможных между соседями, т.е. коэффициент кластеризации для вершины равен отношению числа реально существующих ребер между ее соседями к числу всех возможных ребер между ними и вычисляется по формуле

$$C = \frac{2C_v}{(k_v(k_v-1))}. \tag{4}$$

Тогда общий коэффициент кластеризации определим как среднее между коэффициентами кластеризации всех вершин.

2.2. Результаты численных экспериментов

Для генерации матриц запаздывающих взаимодействий и расчета длины кратчайшего пути в среднем L и коэффициента кластеризации C использовался программный продукт, предназначенный для построения и вычисления свойств графов типа small

world [15]. Данный программный продукт основан на алгоритме построения нейронных сетей со связями small world [1]. Данный программный продукт позволяет по заданным числу нейронов в сети n, числу ближайших соседей и вероятности перенаправления связей p генерировать матрицы смежности, инцидентности и матрицу запаздывающих взаимодействий для любого числа нейроннов в сети и производить расчет числовых характеристик графа нейронной сети. На основании полученных графов нейронных сетей для различных значений параметра p были записаны матрицы запаздывающих взаимодействий, в соответствии с рассматриваемой моделью нейронной сети, рассчитаны коэффициенты кластеризации C и длины кратчайшего пути в среднем L для каждой полученной матрицы B.

Алгоритмы построения областей устойчивости уравнения (1) были реализованы в математическом пакете MATLAB. При построении областей устойчивости предполагалось, что области являются замкнутыми и область, параметры которой обеспечивают устойчивость обязательно содержит точку $(0,0)$ [11]. Были проведены численные эксперименты с целью определения динамики изменений областей устойчивости исследуемой нейронной сети в случае изменения коэффициента кластеризации и длины кратчайшего пути в среднем и зафиксированными параметрами сети: коэффициент демпфирования α и запаздывания в работе сети m, r и число n нейронов в сети.

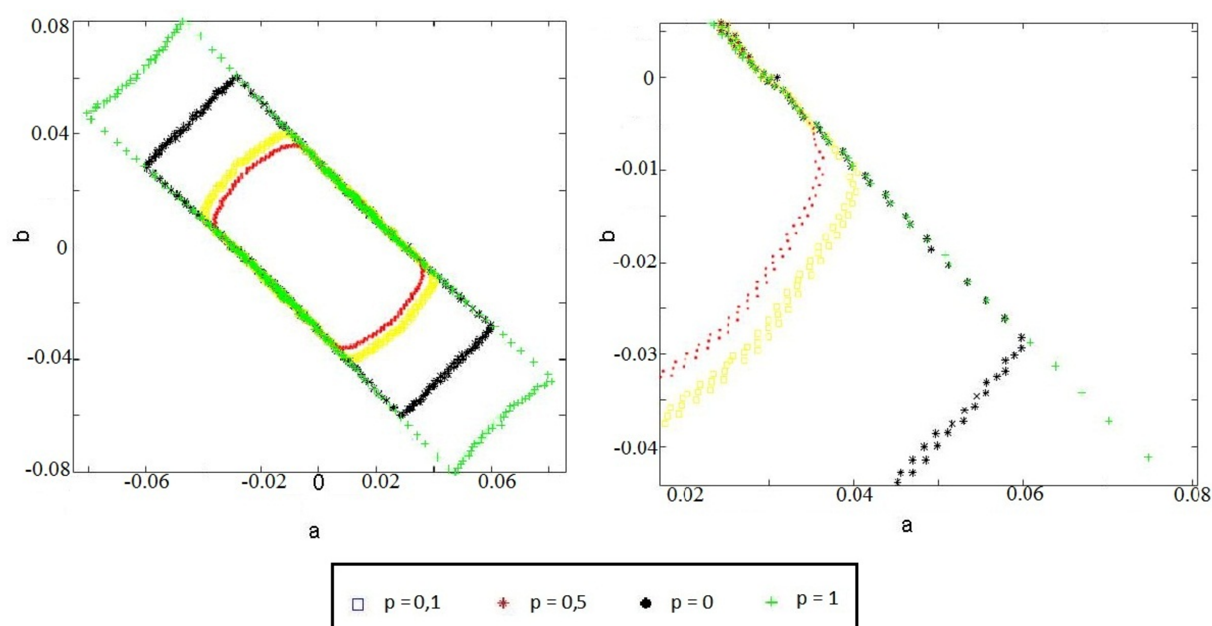
Для наглядности отображения результатов была выбрана сеть с параметрами $n = 400$, $k = 40$, $m = 3$, $r = 2$, $\alpha = 0,4$. Заметим, что параметры C и L оказались неразрывно связаны с параметром p, отвечающим за вероятность перенаправления связей при построении сети. При $p = 0$ мы имеем регулярная сеть, для нее $C = 0,7308$, $L = 5,4887$. Для каждого значения p с помощью алгоритмов, описанных в [10], и программного продукта [15] было сгенерировано десять различных матриц запаздывающих взаимодействий B и получены массивы точек лежащих на границах устойчивости исследуемых нейронных сетей.

Для визуализации результатов на рисунке области устойчивости являются усредненными для каждого значения p. В результате построения матриц B были получены усредненные значения C и L для каждого значения p. При $p = 0,1$ имеем $C = 0,5436$, $L = 2,2401$, при $p = 0,5$ имеем $C = 0,154$, $L = 1,92$, в случае случайной сети $p = 1$ имеем $C = 0,0968$, $L = 1,9128$.

На рисунке представлены области устойчивости сети с разным значением параметров p, C и L и регулярной нейронной сети.

Левая часть рисунка представляет собой области устойчивости нейронной сети целиком. Правая часть рисунка представляет собой увеличенный фрагмент областей устойчивости нейронной сети для понимания динамики изменения границ областей. При $ab > 0$ границы областей устойчивости нейронной сети изменились незначительно при изменении параметров p, C и L. При $ab < 0$ область устойчивости нейронной сети при значении параметра $p = 0$ является наименьшей из представленных на рисунке. В случае $p = 0,1$ область устойчивости несколько больше, чем при $p = 0$, но меньше области устойчивости при $p = 0,5$, которая, в свою очередь, меньше области при $p = 1$.

Численные эксперименты показывают, что наличие свободы в перенаправлении связей внутри сети без получения изолированных нейронов позволяет расширить область устойчивости сети в пространстве параметров a, b. Полученные результаты не противоречат полученным ранее результатам для регулярных сетей и сетей типа small world [10, 11, 16].



Границы областей устойчивости нейронной сети при разных значениях параметра p

Заключение

В рамках исследуемой модели рассмотрены нейронные сети типа со связями типа small world. Нейронные сети с большим значением коэффициента кластеризации и длины кратчайшего пути, соответствующие меньшим значениям параметра p, в среднем имеют меньшие области устойчивости по сравнениями с сетями с меньшими значениями параметров C и L, соответствующие большим значениям параметра p.

На основе результатов данной работы и предыдущих работ [10, 16] можно сделать следующий вывод. При $ab > 0$ изменения топологии нейронной сети не влечет значительного прироста области устойчивости. При $ab < 0$ изменение топологии сети с целью уменьшения параметров C и L графа нейронной сети с сохранением основных свойств сетей типа small world позволяет расширить область устойчивости нейронной сети в пространстве параметров a, b.

Для нейронных сетей со связями типа small world имеется возможность перевода нейронной сети из неустойчивого состояния в устойчивое с помощью перенаправления связей с целью уменьшения параметров C и L графа нейронной сети с сохранением основных параметров нейронной сети. Особенно это эффективно при наличии разных знаков у сил запаздывающих взаимодействий a и b.

Открытым остается вопрос улучшения численных алгоритмов проверки устойчивости моделей нейронных сетей. На данный момент алгоритмы не предназначены для расчетов на графических укорителях. Изменения алгоритмов для возможности использования графических ускорителей позволило бы значительно ускорить время проверки устойчивости дискретных моделей нейронных сетей.

Данная статья завершает цикл работ по устойчивости нейронных сетей со связями small world. Дальнейшим направлением исследований станет исследование возможности применения метода конусов устойчивости при обучении искусственных нейронных сетей для ускорения процесса обучения.

Работа выполнена при финансовой поддержке РФФИ в рамках научного проекта № 16-31-00343. Статья выполнена при поддержке Правительства РФ (Постановление № 211 от 16.03.2013 г.), соглашение № 02.A03.21.0011.

Второй автор был поддержан грантом Южно-Уральского государственного гуманитарно-педагогического университета по теме «Моделирование в теории и практике математического образования».

Литература

1. Watts D., Strogatz S., Collective Dynamics of «Small-World» Networks // Nature. 1998. Vol. 393. P. 440–442.

2. Gray R.T., Fung C.K.C., Robinson P.A., Stability of Small-World Networks of Neural Populations // Neurocomputing. 2009. Vol. 72(7–9). P. 1565–1574. DOI: 10.1016/j.neucom.2008.09.006.

3. Sinha S. Complexity vs Stability in Small-World Networks // Physica A. 2005. Vol. 346. P. 147–153. DOI: 10.1016/j.physa.2004.08.062.

4. Hart M.G., Ypma R.J.F., Romero–Garcia R., Price S.J., Suckling J. Graph Theory Analysis of Complex Brain Networks: New Concepts in Brain Maping Aplied to Neurosurgery // Journal of Neurosurgery. 2016. Vol. 124, No. 6. P. 1665–1678. DOI: 10.3171/2015.4.jns142683.

5. Jobst B.M., Hindriks R., Laufs H., Tagliazucchi E., Hahn G., Ponce-Alvarez A., Stevner A.B.A., Kringelbach M.L., Deco G. Increased Stability and Breakdown of Brain Effective Connectivity During Slow-Wave Sleep: Mechanistic Insights from Whole-Brain Computational Modelling // Scientific Reports. 2017. Vol. 5, No 7(1). P. 1–16. DOI: 10.1038/s41598-017-04522-x.

6. Takesian A.E., Hensch T.K. Chapter 1 — Balancing Plasticity/Stability Across Brain Development // Progress in Brain Research. 2013. Vol. 207. P. 3–34. DOI: 10.1016/b978-0-444-63327-9.00001-1.

7. Netoff T.I., Clewley R., Arno S., Keck T., John A. White Epilepsy in Small-World Networks // The Journal of Neuroscience. 2004. Vol. 24(37). P. 8075–8083. DOI: 10.1523/jneurosci.1509-04.2004.

8. Arbib M.A., Érdi P., Szentágothai J. Neural Organization: Structure, Function, and Dynamics. Cambridge. MA: MIT Press. 1998. 420 p.

9. Arbib M. The Handbook of Brain Theory and Neural Networks. Cambridge. MA: MIT Press. 2003. 1308 p.

10. Иванов С.А. Вычисление областей устойчивости дискретных моделей больших нейронных сетей типа small world // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2016. Т. 5, № 3. С. 69–75. DOI: 10.14529/cmse160305.

11. Ivanov S.A., Kipnis M.M. Stability Analysis Discrete-time Neural Networks with Delayed interactions: Torus, Ring, Grid, Line // International Journal of Pure and Aplied Math. 2012. Vol. 78(5). P. 691–709.

12. Ivanov S.A., Kipnis M.M., Medina R. On the Stability of the Cartesian Product of a Neural Ring and an Arbitrary Neural Network // Advances in Difference Equations. 2014. Vol. 2014. P. 1–7. DOI: 10.1186/1687-1847-2014-176.

13. Kipnis M.M., Malygina V.V. The Stability Cone for a Matrix Delay Difference Equation // International Journal of Mathematics and Mathematical Sciences. 2011. Vol. 2011. P. 1–15. DOI: 10.1155/2011/860326.

14. Ivanov S.A., Kipnis M.M., Malygina V.V. The Stability Cone for a Difference Matrix Equation with Two Delays // ISRN Aplied Math. 2011. Vol. 2011. P. 1–19. DOI: 10.5402/2011/910936.

15. Блеес И.И., Иванов С.А. Свидетельство Роспатента об официальной регистрации программы для ЭВМ «Small world graph generator» номер 2016662698 от 21.11.2016, правообладатель: ФГБОУ ВПО «ЮУрГУ (НИУ)».

16. Ivanov S., Kipnis M. On the Stability of a Neural Network with Links Based on the Watts-Strogatz Model // International Journal of Pure and Applied Mathematics. 2015. Vol. 105, No. 3. P. 431–438. DOI: 10.12732/ijpam.v105i3.11.

Иванов Сергей Александрович, к.ф.-м.н., кафедра системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

Кипнис Михаил Маркович, д.ф.-м.н., профессор, кафедра математики и методики обучения математике, Южно-Уральский государственный гуманитарно-педагогический университет (Челябинск, Российская Федерация)

DOI: 10.14529/cmse180202

DYNAMICS OF STABILITY REGIONS OF DISCRETE MODELS OF NEURAL NETWORKS OF SMALL WORLD TYPE WHEN THE NUMERIC CHARACTERISTICS OF THE NETWORK GRAPH CHANGE

© 2018
S.A. Ivanov[1], M.M. Kipnis[2]

[1]*South Ural State University (pr. Lenina 76, Chelyabinsk, 454080 Russia),*

[2]*South Ural State Humanitarian Pedagogical University*

(pr. Lenina 69, Chelyabinsk, 454080 Russia)

E-mail: saivanov@susu.ru, mmkipnis@gmail.com

Received: 17.10.2017

The article gives the description of the discrete models of neural networks with constraints of the type of small world with probability redirecting connections within the network p varying from 0 to 1. At the value $p = 0$ we obtain a model of a regular neural network. A regular neural network is a ring neural network, in which each neuron interacts with several neighbors along the ring. At $p = 1$, we obtain a model, the neurons of which are randomly connected to other neurons of the network without formation of isolated neurons. The neural networks are widely used in modeling various neural structures in living organisms, for example, mammalian brain hypocampus. The paper studies the dynamics of the stability regions of the neural networks in case of changes in the probability of redirecting links, clustering coefficient and the length of the shortest path in the average for the graph of neural network. In a series of numerical experiments, the regions of stability of the studied neural network models for various network parameters were constructed, and the conclusion about increasing the stability region while reducing the length of the shortest path on average and the clustering coefficient of the network graph was drawn.

Keywords: Watts–Strogatz discrete models, small world, stability.

FOR CITATION

Ivanov S.A., Kipnis M.M. Dynamics of Stability Regions of Discrete Models of Neural Networks of Small World Type When the Numeric Characteristics of the Network Graph Change. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2018. vol. 7, no. 2. pp. 22–31. (in Russian) DOI: 10.14529/cmse180202.

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

References

1. Watts D., Strogatz S. Collective Dynamics of "Small-World" Networks. *Nature.* 1998. vol. 393. pp. 440–442.
2. Gray R.T., Fung C.K.C., Robinson P.A. Stability of Small-World Networks of Neural Populations. *Neurocomputing.* 2009. vol. 72(7–9). pp. 1565–1574. DOI: 10.1016/j.neucom.2008.09.006.
3. Sinha S. Complexity vs Stability in Small-World Networks. *Physica A.* 2005. vol. 346. pp. 147–153. DOI: 10.1016/j.physa.2004.08.062.
4. Hart M.G., Ypma R.J.F., Romero-Garcia R., Price S.J., Suckling J. Graph Theory Analysis of Complex Brain Networks: New Concepts in Brain Maping Aplied to Neurosurgery. *Journal of Neurosurgery.* 2016. vol. 124, no. 6. pp. 1665–1678. DOI: 10.3171/2015.4.jns142683.
5. Jobst B.M., Hindriks R., Laufs H., Tagliazucchi E., Hahn G., Ponce-Alvarez A., Stevner A.B.A., Kringelbach M.L., Deco G. Increased Stability and Breakdown of Brain Effective Connectivity During Slow-Wave Sleep: Mechanistic Insights from Whole-Brain Computational Modelling. *Scientific Reports.* 2017. vol. 5, no 7(1). pp. 1–16. DOI: 10.1038/s41598-017-04522-x.
6. Takesian A.E., Hensch T.K. Chapter 1 — Balancing Plasticity/Stability Across Brain Development. *Progress in Brain Research.* 2013. vol. 207. pp. 3–34. DOI: 10.1016/b978-0-444-63327-9.00001-1.
7. Netoff T.I., Clewley R., Arno S., Keck T., John A. White Epilepsy in Small-World Networks. *The Journal of Neuroscience.* 2004. vol. 24(37). pp. 8075–8083. DOI: 10.1523/jneurosci.1509-04.2004.
8. Arbib M.A., Érdi P., Szentágothai J. *Neural Organization: Structure, Function, and Dynamics.* Cambridge. MA: MIT Press. 1998. 420 p.
9. Arbib M. *The Handbook of Brain Theory and Neural Networks.* Cambridge. MA: MIT Press. 2003. 1308 p.
10. Ivanov S.A. Calculation of Stability Domains of Discrete Models of Big Size Small World Networks. [Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering]. 2016. vol. 5, no. 3. pp. 69–75. (in Russian) DOI: 10.14529/cmse160305.
11. Ivanov S.A., Kipnis M.M. Stability Analysis Discrete-Time Neural Networks with Delayed Interactions: Torus, Ring, Grid, Line. *International Journal of Pure and Aplied Math.* 2012. vol. 78(5). pp. 691–709.

12. Ivanov S.A., Kipnis M.M., Medina R. On the Stability of the Cartesian Product of a Neural Ring and an Arbitrary Neural Network. *Advances in Difference Equations*. 2014. vol. 2014. pp. 1–7. DOI: 10.1186/1687-1847-2014-176.
13. Kipnis M.M., Malygina V.V. The Stability Cone for a Matrix Delay Difference Equation. *International Journal of Mathematics and Mathematical Sciences*. 2011. vol. 2011. pp. 1–15. DOI: 10.1155/2011/860326.
14. Ivanov S.A., Kipnis M.M., Malygina V.V., The Stability Cone for a Difference Matrix Equation with Two Delays. *ISRN Aplied Math*. 2011. vol. 2011. pp. 1–19. DOI: 10.5402/2011/910936.
15. Blees I.I., Ivanov S.A. *Svidetel'stvo Rospatenta ob ofitsial'noi registratsii programmy dlya EVM "Small world graph generator"* [Certificate of Rospatent on official registration of computer programs "Small world graph generator"] number 2016662698 from 21.11.2016, right holder: FSBEIHE SUSU (NRU). (in Russian)
16. Ivanov S., Kipnis M. On the Stability of a Neural Network with Links Based on the Watts-Strogatz Model. *International Journal of Pure and Applied Mathematics*. 2015. vol. 105, no. 3. pp. 431–438. DOI: 10.12732/ijpam.v105i3.11.

УДК 004.051 DOI: 10.14529/cmse180203

МОДЕЛЬ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ МНОГОПРОЦЕССОРНЫХ СИСТЕМ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ*

© 2018 Н.А. Ежова[1], Л.Б. Соколинский[1,2]

[1]*Южно-Уральский государственный университет*
(454080 Челябинск, пр. им. В.И. Ленина, д. 76),
[2]*Институт математики и механики им. Н.Н. Красовского УрО РАН*
(620990 Екатеринбург, ул. С. Ковалевской, д. 16)
E-mail: ezhovana@susu.ru, leonid.sokolinsky@susu.ru

Поступила в редакцию: 12.03.2018

Появление мощных многопроцессорных вычислительных систем выдвигает на первый план вопросы, связанные с разработкой фреймворков (шаблонов), позволяющих создавать высокомасштабируемые параллельные программы, ориентированные на системы с распределенной памятью. При этом особенно важной является проблема разработки моделей параллельных вычислений, позволяющих на ранней стадии проектирования программы оценить ее масштабируемость. В статье приводятся общие требования к модели вычислений и строится новая высокоуровневая модель параллельных вычислений Bulk Synchronous Farm (BSF), являющаяся расширением модели BSP, и основанная на методе программирования SPMD и парадигме «мастер-рабочие». Модель BSF ориентирована на вычислительные системы с массовым параллелизмом на распределенной памяти, включающие в себя сотни тысяч процессорных узлов, и имеющие экзафлопный уровень производительности и на численные итерационные методы с высокой временной сложностью. Определяется архитектура BSF-компьютера и описывается структура BSF-программы. Описывается формальная стоимостная метрика, с помощью которой получаются верхние оценки масштабируемости параллельных BSF-программ применительно к вычислительным системам с распределенной памятью. Также выводятся формулы для оценки эффективности распараллеливания BSF-программ и даются аналитические оценки масштабируемости BSF-приложений.

Ключевые слова: параллельное программирование, модель параллельных вычислений, фреймворк «мастер-рабочие», модель BSF, временная сложность, Bulk Synchronous Farm, масштабируемость, многопроцессорные системы с распределенной памятью.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Ежова Н.А., Соколинский Л.Б. Модель параллельных вычислений для многопроцессорных систем с распределенной памятью // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2018. Т. 7, № 2. С. 32–49. DOI: 10.14529/cmse180203.

Введение

Суперкомпьютер TaihuLight с массово-параллельной архитектурой, занимающий первое место в списке TOP-500 [1] самых мощных суперкомпьютеров мира (ноябрь 2016), имеет 40 960 процессорных узлов, каждый из которых включает в себя 260 процессорных ядер. Общая оперативная память системы составляет 1.3 Петабайт, пиковая

* Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2018».

производительность превышает 120 петафлопс. Анализ динамики роста производительности суперкомпьютеров (рис. 1) показывает, что через 8–9 лет самый мощный суперкомпьютер становится рядовой системой, и что через 5–6 лет мы можем ожидать появление суперкомпьютера с экзафлопным уровнем производительности. Появление столь мощных многопроцессорных вычислительных систем выдвигает на первый план вопросы, связанные с разработкой фреймворков (шаблонов), позволяющих создавать высокомасштабируемые параллельные программы, ориентированные на системы с распределенной памятью. При этом особенно важной является проблема разработки моделей параллельных вычислений, позволяющих на ранней стадии проектирования программы оценить ее масштабируемость.

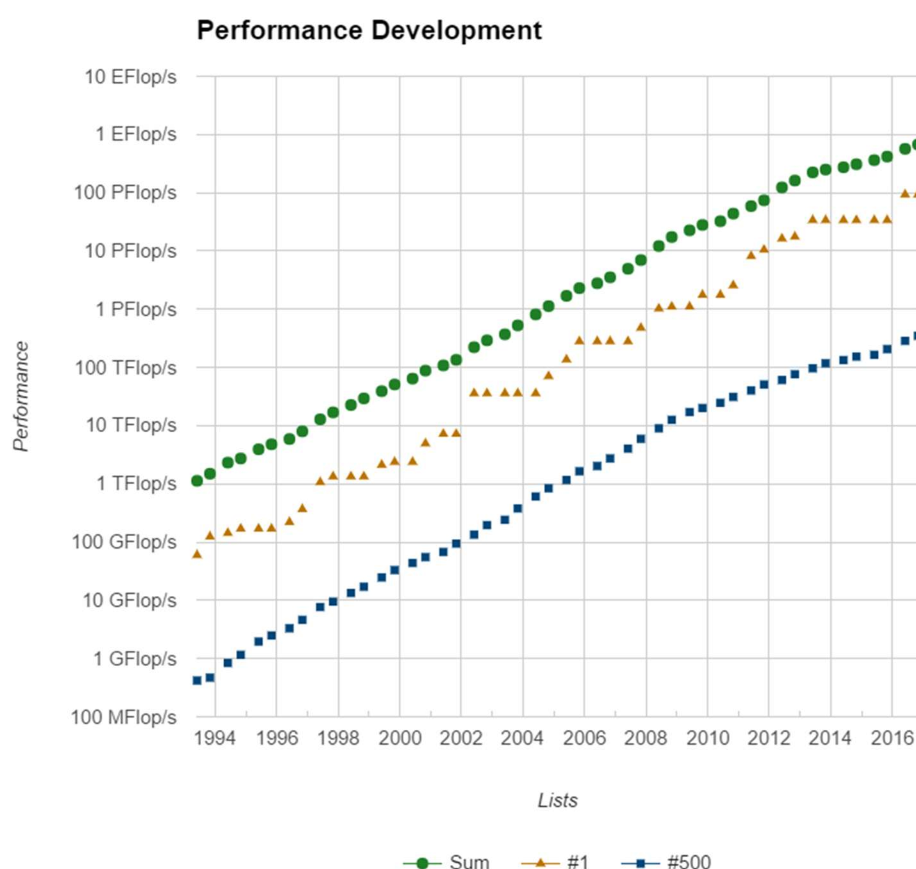


Рис. 1. Динамика роста производительности суперкомпьютеров в TOP500

В данной работе предлагается новая модель параллельных вычислений BSF (Bulk Synchronous Farm) — блочно-синхронная ферма, основанная на моделях «мастер-рабочие», BSP и SPMD. Модель BSF ориентирована на вычислительные системы с массовым параллелизмом на распределенной памяти, включающие в себя сотни тысяч процессорных узлов, и имеющие экзафлопный уровень производительности. Модель BSF включает в себя каркас (skeleton) для разработки параллельных программ и стоимостную метрику для оценки масштабируемости приложения. Статья имеет следующую структуру. В разделе 1 дается краткий обзор концептуальных моделей параллельных вычислений, лежащих в основе новой модели BSF. В разделе 2 приводятся общие требования к модели

вычислений. Раздел 3 посвящен описанию модели BSF. В разделе 4 строится стоимостная метрика, и даются аналитические оценки масштабируемости BSF приложений, а также исследуется вопрос эффективности распараллеливания BSF программ. В заключении суммируются полученные результаты и намечаются направления дальнейших исследований.

1. Модели параллельных вычислений

Одним из популярных фреймворков, используемых в параллельном и распределенном программировании, является *парадигма «мастер-рабочие»* [2–4]. В соответствии с этой парадигмой процессорные узлы вычислительной системы делятся на два множества: узлы-мастера и узлы-рабочие. Задача, решаемая на многопроцессорной системе, разбивается на независимые подзадачи. Каждая подзадача, в свою очередь, разбивается на три последовательные стадии: стадия предобработки, стадия вычислений и стадия постобработки. Стадии предобработки и постобработки выполняются узлами-мастерами, стадии вычислений выполняются узлами-рабочими. В каждый момент времени любой процессорный узел может выполнять только одну стадию. Предполагается, что количество подзадач совпадает с количеством узлов-рабочих и превышает количество узлов- мастеров. Решение задачи происходит следующим образом. Очередная подзадача назначается некоторому узлу-мастеру, который выполняет стадию предобработки. После этого он посылает задание (передает данные) узлу-рабочему, который должен выполнить стадию вычислений указанной подзадачи. После того, как узел-рабочий завершил требуемые вычисления, он посылает отчет (передает данные) тому узлу-мастеру, от которого получил задание. На этом выполнение подзадачи заканчивается. Задача считается выполненной целиком, когда выполнены все ее подзадачи. Фреймворк «мастер-рабочий» очень часто используется в параллельном программировании при реализации различных приложений, ориентированных на многопроцессорные системы с распределенной памятью (см., например, [5–10]). При этом наиболее популярна конфигурация, включающая одного мастера и множество рабочих (рис. 2). Основной проблемой модели «мастер-рабочие» является нахождение такого расписания вычислений, при котором время выполнения задачи будет минимальным. Известно, что указанная проблема является в общем случае NP-сложной [11]. В определенной мере эту проблему можно решить, используя модель *SPMD*.

SPMD (*Single Program Multiple Data*) [3, 12] — популярная парадигма параллельного программирования, в соответствии с которой все процессорные узлы выполняют одну и ту же программу, но обрабатывают различные данные. Какие именно данные необходимо обрабатывать тому или иному процессорному узлу, определяется его уникальным номером, который является параметром программы. Данный подход наиболее часто используется в сочетании с технологией MPI (Message Passing Interface) [13], которая де-факто является стандартом для параллельного программирования на распределенной памяти.

Модель BSP (*Bulk-Synchronous Parallelism*) была предложена Валиантом (Valiant) в работе [14]. Данная модель широко используется при разработке и анализе параллельных алгоритмов и программ. *BSP-компьютер* представляет собой систему из p процессоров, имеющих приватную память, и соединенных сетью, позволяющей передавать

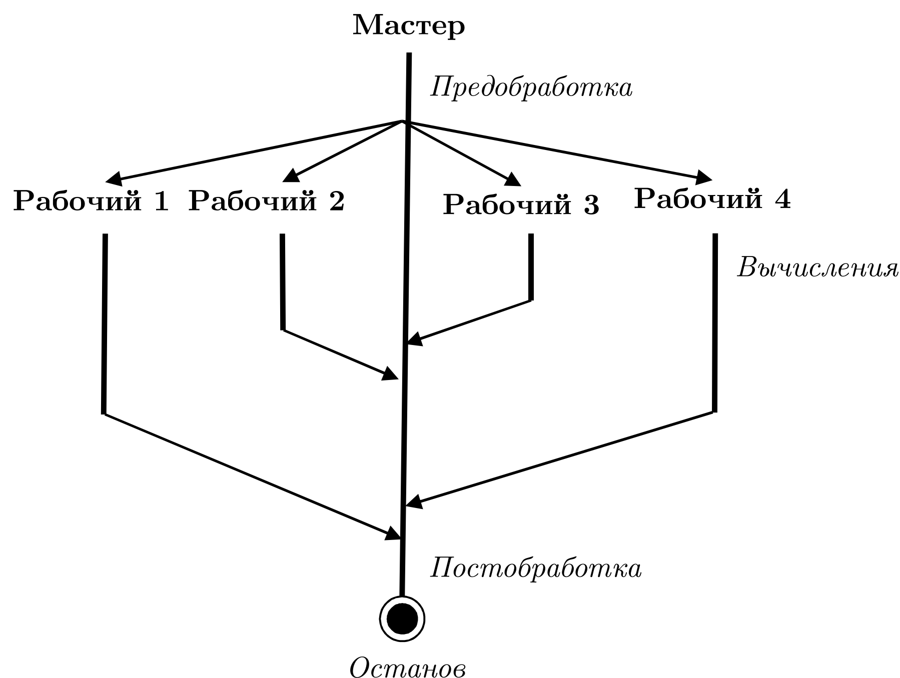


Рис. 2. Фреймворк «мастер-рабочие».
Полужирными линиями обозначены потоки вычислений,
тонкими линями со стрелками — потоки данных

пакеты данных фиксированного размера от одного процессора другому. Для соединительной сети вводятся следующие характеристики: g — время, необходимое для передачи по сети одного пакета; L — время, необходимое для инициализации передачи данных от одного процессора другому.

BSP-программа состоит из p потоков команд, каждый из которых назначается отдельному процессору, и делится на *супершаги*, которые выполняются последовательно относительно друг друга. Каждый *супершаг*, в свою очередь, включает в себя следующие четыре последовательных шага: 1) вычисления на каждом процессоре с использованием только локальных данных; 2) глобальная барьерная синхронизация; 3) пересылка данных от любого процессора любым другим процессорам; 4) глобальная барьерная синхронизация. Переданные данные становятся доступными для использования только после барьерной синхронизации. Пример *BSP*-программы приведен на рис. 3.

Стоимостная функция в модели *BSP* строится следующим образом [15]. Пусть *BSP*-программа состоит из S супершагов. Пусть w_i — максимальное время, затраченное каждым процессором на локальные вычисления, h_i — максимальное количество пакетов, посланных или полученных каждым процессором, на i-том супершаге. Тогда общее время t_i, затрачиваемое системой на выполнение i-того супершага, вычисляется по формуле

$$t_i = w_i + gh_i + L .$$

Время T выполнения всей программы определяется по формуле

$$T = W + Hg + LS , \tag{1}$$

где $W = \sum_{i=1}^{S} w_i$ и $H = g\sum_{i=1}^{S} h_i$.

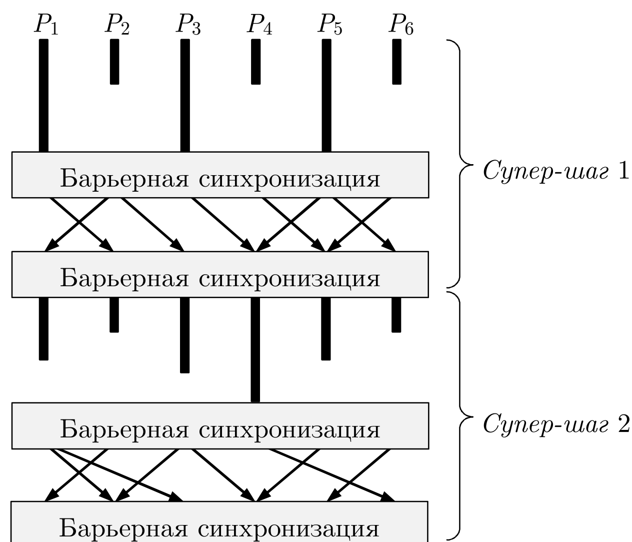


Рис. 3. *BSP*-программа из двух супершагов на шести процессорах.
Жирными линиями обозначены локальные вычисления,
тонкими линиями со стрелками — пересылка данных

Для того, чтобы облегчить программистам использование той или иной модели параллельных вычислении на практике, применяются *программные каркасы* [16]. Одним из наиболее популярных является *фермерный каркас* (*farm skeleton*), реализующий фреймворк «мастер-рабочие» [17]. Такой каркас представляет собой программную структуру, полностью реализующую фреймворк «мастер-рабочие», однако вместо эффективного кода и реальных данных она содержит заглушки, которые должны быть заменены по определенным правилам на фрагменты кода, реализующие целевую задачу.

2. Требования к модели параллельных вычислении

Модель параллельных вычислений в общем случае должна включать в себя следующие четыре компонента, некоторые из которых в определенных случаях могут быть тривиальны [18].

1. *Архитектурный компонент*, описываемый как помеченный граф, узлы которого соответствуют модулям с различной функциональностью, а дуги — межмодульным соединениям для передачи данных.
2. *Спецификационный компонент*, определяющий, что есть корректная программа.
3. *Компонент выполнения*, определяющий, как взаимодействуют между собой архитектурные модули при выполнении корректной программы.
4. *Стоимостный компонент*, определяющий одну или более стоимостных метрик для оценки времени выполнения корректной программы.

В качестве наиболее важных свойств модели параллельных вычислений обычно выделяют следующие [19]:

– *Юзабилити*, определяющая легкость описания алгоритма и анализа его стоимости средствами модели (модель должна быть легкой в использовании).
– *Адекватность*, выражающаяся в соответствии реального времени выполнения программ и временной стоимости, полученной аналитически с помощью стоимостных

метрик модели (программа, имеющая меньшую временную стоимость, должна выполняться быстрее).

– *Портируемость*, характеризующая широту класса целевых платформ, для которых модель оказывается применимой.

3. Модель BSF

Модель параллельных вычислений BSF (*Bulk Synchronous Farm*) — *блочно-синхронная ферма* ориентирована на многопроцессорные системы с кластерной архитектурой и архитектурой MPP. *BSF-компьютер* представляет собой множество однородных процессорных узлов с приватной памятью, соединенных сетью, позволяющей передавать данные от одного процессорного узла другому. Среди процессорных узлов выделяется один, называемый *узлом-мастером* (или кратко *мастером*). Остальные K узлов называются *узлами-рабочими* (или просто *рабочими*). В BSF компьютере должен быть по крайней мере один узел мастер и один рабочий ($K \geq 1$). Схематично архитектура BSF-компьютера изображена на рис. 4.

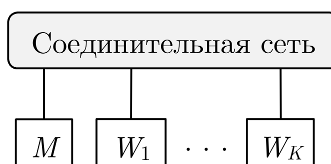


Рис. 4. *BSF*-компьютер.

M — мастер; $W_1, \ldots, W_K$ — рабочие

BSF компьютер работает по схеме SPMD. *BSF-программа* состоит из последовательности *супершагов* и глобальных барьерных синхронизаций, выполняемых мастером и всеми рабочими. Каждый супершаг делится на секции двух типов: *секции мастера*, выполняемые только мастером, и *секции рабочего*, выполняемые только рабочими. Относительный порядок секций мастера и рабочего в рамках супершага не существенен. Данные, обрабатываемые конкретным узлом-рабочим, определяются его номером, являющимся параметром среды исполнения.

BSF программа включает в себя следующие последовательные *разделы* (рис. 5):

– инициализация;
– итерационный процесс;
– завершение.

Инициализация представляет собой супершаг, в ходе которого мастер и рабочие считывают или генерируют исходные данные. Инициализация завершается барьерной синхронизацией. *Итерационный процесс* состоит в многократном повторении *тела итерационного процесса* до тех пор, пока не выполнено условие завершения, проверяемое мастером. В разделе *завершение* осуществляется вывод или сохранение результатов и завершение программы.

Тело итерационного процесса включает в себя следующие супершаги:

1) передача рабочим заданий от мастера;
2) выполнение задания (рабочими);
3) передача мастеру результатов от рабочих;
4) обработка полученных результатов мастером.

На первом супершаге мастер рассылает всем рабочим одинаковые задания. На втором супершаге происходит выполнение полученного задания рабочими (мастер при этом простаивает). Все рабочие выполняют один и тот же программный код, но обрабатывают

Рис. 5. Структура BSF-программы

различные данные, адреса которых определяются по номеру рабочего. Это означает, что все рабочие тратят на вычисления одно и то же время. Никаких пересылок данных при выполнении задания не происходит. Это является важным свойством модели BSF. На третьем супершаге все рабочие пересылают мастеру полученные результаты. Суммарный объем результатов является атрибутом задачи и не зависит от количества рабочих. После этого происходит глобальная барьерная синхронизация. В ходе четвертого супершага мастер производит обработку и анализ полученных результатов. Рабочие в это время простаивают. Время обработки мастером результатов, полученных от рабочих, является параметром задачи и не зависит от количества рабочих. Если после обработки результатов условие завершения оказывается истинным, то происходит выход из итерационного процесса, в противном случае осуществляется переход на первый супершаг итерационного процесса. На четвертом супершаге происходит вывод или сохранение результатов и завершение работы мастера и рабочих. Графическая иллюстрация работы BSF-программы приведена на рис. 6.

Областью применения модели BSF являются *масштабируемые* итерационные численные методы, имеющие высокую вычислительную сложность итерации при относительно невысокой стоимости коммуникаций. Под *масштабируемым итерационным методом* понимается метод, допускающий разбиение итерации на подзадачи, не требующие обменов данными. Пример такого метода можно найти в работе [20].

4. Исследование масштабируемости BSF-программ

Основной характеристикой масштабируемости является ускорение, вычисляемое как отношение времени выполнения программы на одном процессорном узле ко времени выполнения на K узлах. В данном разделе мы проведем аналитическое исследование масштабируемости модели *BSF*. Для этого нам понадобится оценка временных затрат на выполнение *BSF*-программы. Мы предполагаем, что временные затраты на инициализацию

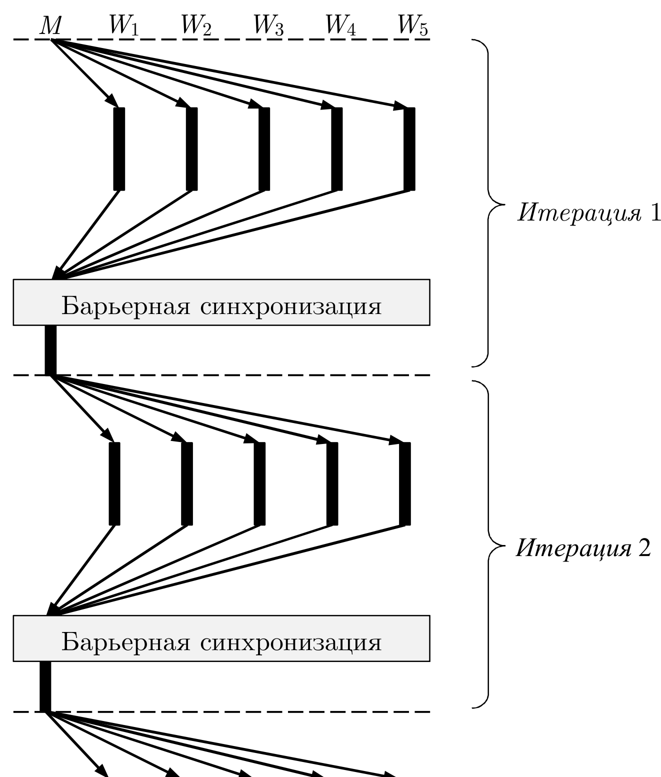


Рис. 6. Иллюстрация работы *BSF*-программы
с одним мастером M и пятью рабочими $W_1, \ldots, W_5$
(жирными линиями обозначены локальные вычисления,
тонкими линиями со стрелками — пересылка данных,
пунктирными линиями — границы одной итерации)

и завершение *BSF*-программы пренебрежимо малы по сравнению с затратами на выполнение итерационного процесса. Стоимость итерационного процесса получается как сумма стоимостей отдельных итераций. Поэтому для оценки времени выполнения *BSF*-программы нам достаточно получить оценку временной стоимости одной итерации.

Рассмотрим сначала конфигурацию вычислительной системы в составе мастера и одного рабочего. Пусть t_s — время, необходимое для посылки задания рабочему (без учета латентности); t_r — время, необходимое для передачи результата мастеру от рабочего (без учета латентности); t_p — время обработки мастером результатов, полученных от рабочего; L — затраты на инициализацию операции передачи сообщения (латентность); t_w — время выполнения задания бригадой из одного рабочего. Общее время T_1 выполнения итерации одним мастером и бригадой из одного рабочего может быть вычислено следующим образом:

$$T_1 = t_s + t_w + L + t_p + t_r + L, \tag{2}$$

что равносильно

$$T_1 = 2L + t_s + t_r + t_p + t_w. \tag{3}$$

Теперь рассмотрим конфигурацию вычислительной системы в составе одного мастера и K рабочих. Все рабочие получают от мастера одно и то же сообщение, поэтому общее время передачи сообщений от мастера рабочим составит $K(L+t_s)$. Все рабочие выполняют один и тот же код над своей частью данных, поэтому время выполнения всех вычислений K рабочими в рамках одной итерации будет равно t_w/K. Суммарный объем результатов, вычисленных рабочими, является параметром задачи и не зависит от K, поэтому общее время передачи сообщений мастеру от рабочих составит $K \cdot L + t_r$. Время обработки мастером результатов, полученных от рабочих, также является параметром задачи и не зависит от количества рабочих. Таким образом, общее время T_K выполнения итерации в системе с одним мастером и K рабочими может быть вычислено следующим образом:

$$T_K = K(2L + t_s) + t_r + t_p + t_w/K. \tag{4}$$

Из формул (3) и (4) получаем следующую формулу для ускорения a:

$$a(K) = \frac{T_1}{T_K} = \frac{2L + t_s + t_r + t_p + t_w}{K(2L + t_s) + t_r + t_p + t_w/K}. \tag{5}$$

Свяжем величины t_s и t_w через новую переменную v следующим уравнением:

$$v = \lg\left(t_w/t_s\right). \tag{6}$$

Тогда

$$t_s = 10^{-v} t_w. \tag{7}$$

Подставляя значение t_s из уравнения (7) в уравнение (5), получим

$$a(K) = \frac{T_1}{T_K} = \frac{2L + 10^{-v} t_w + t_r + t_p + t_w}{K(2L + 10^{-v} t_w) + t_r + t_p + t_w/K}. \tag{8}$$

Исследуем, как для некоторой фиксированной задачи выглядят графики зависимости ускорения от числа рабочих. Пусть имеется некоторая задача с решением в пространстве $\mathbb{R}^n$. Предположим, что

$$n = 10^4,\ t_w = n^3 = 10^{12},\ t_p = t_r = n = 10^4,\ L = 0.5. \tag{9}$$

Подставляя указанные значения в формулу (8), получаем

$$a(K) = \frac{1 + 10^{12-v} + 20^4 + 10^{12}}{(1 + 10^{12-v})K + 20^4 + 10^{12} / K} \approx \frac{10^{8-v} + 2 + 10^8}{10^{8-v} K + 2 + 10^8 / K}. \tag{10}$$

На рис. 7 приведены кривые ускорения a, вычисляемые по формуле (10) для различных значений параметра v.

Границами масштабируемости в каждом случае являются точки максимумов кривых ускорения, то есть — это точки, где производная ускорения равна нулю. Для определения таких точек вычислим производную по K для ускорения, вычисляемого по формуле (5):

$$a'(K) = \frac{(2L + t_s + t_r + t_p + t_w)(t_w/K^2 - 2L - t_s)}{\left(K(2L + t_s) + t_r + t_p + t_w/K\right)^2}. \tag{11}$$

Соответственно, для задачи (9), продифференцировав (10), получаем

$$a'(K) \approx \frac{\left(10^{8-v} + 2 + 10^8\right)\left(10^8 / K^2 - 10^{8-v}\right)}{\left(10^{8-v}K + 2 + 10^8 / K\right)^2}. \tag{12}$$

Примеры графиков производных ускорения a', вычисляемых по формуле (12), приведены на рис. 8.

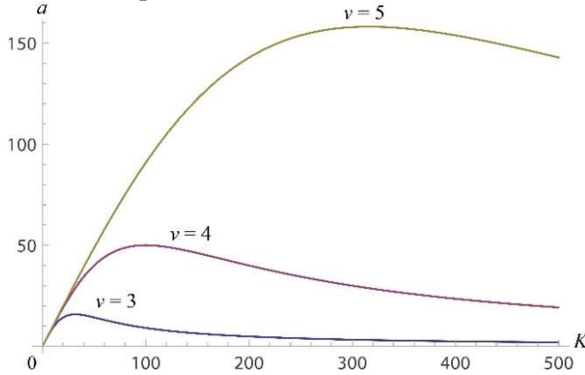


Рис. 7. Кривые ускорения для задачи (9) при различных v

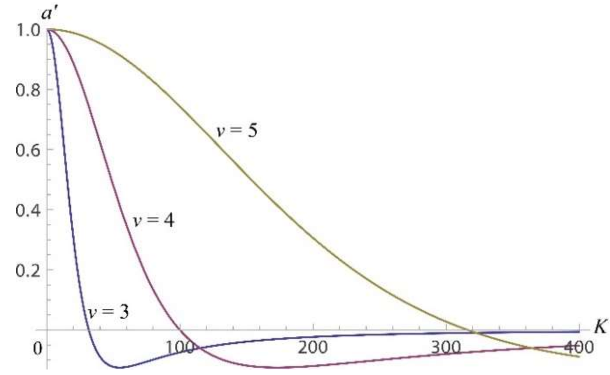


Рис. 8. Кривые производной ускорения для задачи (9) при различных v

Для того, чтобы вычислить нули производной (11), найдем корни уравнения

$$\frac{(2L + t_s + t_r + t_p + t_w)(t_w/K^2 - 2L - t_s)}{\left(K(2L + t_s) + t_r + t_p + t_w/K\right)^2} = 0\,. \tag{13}$$

Поделив обе части уравнения (13) на $(2L+t_s+t_r+t_p+t_w)$, и умножив их на $\left(K(2L+t_s)+t_r+t_p+t_w/K\right)^2$, получим

$$t_w/K^2 - 2L - t_s = 0\,, \tag{14}$$

что равносильно

$$\frac{K^2}{t_w} = \frac{1}{2L + t_s}, \tag{15}$$

то есть

$$K = \sqrt{\frac{t_w}{2L + t_s}}\,. \tag{16}$$

Таким образом, границы масштабируемости *BSF*-программы определяется следующим неравенством:

$$K \le \sqrt{\frac{t_w}{2L + t_s}}, \tag{17}$$

где K — количество узлов-рабочих; t_w — время выполнения всех вычислений в рамках итерации бригадой из одного рабочего; t_s — время, необходимое для посылки задания одному рабочему; L — затраты на инициализацию операции передачи сообщения. Примечательно то, что границы масштабируемости *BSF*-программы не зависят от затрат на пересылку результатов от рабочих мастеру и от времени обработки этих результатов на узле-мастере. Однако, как будет показано ниже, эти параметры оказывают существенное влияние на эффективность распараллеливания.

Продемонстрируем, как полученная оценка может применяться на практике. Пусть имеется некоторая *BSF*-программа, для которой параметр n (размерность задачи) характеризует объем исходных данных. Предположим, что затраты на посылку задания одному рабочему составляют $O(n)$, а суммарная временная стоимость вычислений, выполняемых рабочими, равна $O(n^3)$. Тогда по формуле (17) получаем $K \le \sqrt{O(n^3)/O(n)}$, то есть $K \le O(n)$. Это означает, что верхняя граница масштабируемости программы будет расти пропорционально росту размерности задачи, и, следовательно, мы можем характеризовать такую программу как *хорошо масштабируемую*.

Предположим теперь, что временная стоимость посылки задания одному рабочему по-прежнему составляет $O(n)$, а суммарная временная стоимость вычислений, выполняемых рабочими, равна $O(n^2)$. Тогда по формуле (17) получаем $K \le \sqrt{O(n^2)/O(n)}$, то есть $K \le \sqrt{O(n)}$. Это означает, что верхняя граница масштабируемости программы будет расти как корень квадратный от размерности задачи. Такую программу мы можем характеризовать как *ограниченно масштабируемую*. В заключение рассмотрим случай, когда в рамках одной итерации суммарная временная стоимость посылки задания рабочему пропорциональна суммарной временной стоимости вычислений, выполняемых рабочими, и равна $O(n)$. В этом случае из формулы (17) получаем $K \le \sqrt{O(n)/O(n)}$. Это означает, что верхняя граница масштабируемости программы ограничена некоторой константой, не зависящей от размерности задачи. Такую программу мы можем характеризовать как *плохо масштабируемую*.

Таким образом, мы можем сделать вывод, что *BSF*-программа будет обладать хорошей масштабируемостью, если временные затраты на посылку задания одному рабочему будут пропорциональны кубическому корню от суммарных временных затрат на решение задачи рабочими.

Оценим теперь эффективность e распараллеливания *BSF*-программы. Используя (3) и (4), имеем

$$e(K) = \frac{T_1}{K \cdot T_K} = \frac{2L + t_s + t_r + t_p + t_w}{K^2(2L + t_s) + K(t_p + t_r) + t_w} =$$

$$= \frac{2L + t_s}{K^2(2L + t_s) + K(t_p + t_r) + t_w} + \frac{t_p + t_r}{K^2(2L + t_s) + K(t_p + t_r) + t_w} +$$

$$+ \frac{t_w}{K^2(2L + t_s) + K(t_p + t_r) + t_w}.$$

В предположении $K \gg 1$ имеем

$$\frac{2L + t_s}{K^2(2L + t_s) + K(t_p + t_r) + t_w} \approx 0$$

и

$$\frac{t_p + t_r}{K^2(2L + t_s) + K(t_p + t_r) + t_w} \approx 0 \,.$$

Отсюда следует, что

$$e \approx \frac{t_w}{K^2(2L + t_s) + K(t_p + t_r) + t_w}$$

для $K \gg 1$. Поделив числитель и знаменатель на t_w, получаем итоговую формулу

$$e \approx \frac{1}{1 + \left(K^2(2L + t_s) + K(t_p + t_r)\right)\Big/t_w} \,. \qquad (18)$$

Подставляя значение t_s из уравнения (7), можно получить следующий вариант формулы (18):

$$e \approx \frac{1}{1 + \left(K^2(2L + 10^{-v}t_w) + K(t_p + t_r)\right)\Big/t_w} \,. \qquad (19)$$

Подсчитаем по формуле (19) эффективность распараллеливания задачи (9):

$$e \approx \frac{1}{1 + \left(K^2(1 + 10^{12-v}) + 2K \cdot 10^4\right)\Big/10^{12}} \,. \qquad (20)$$

На рис. 9 приведены графики эффективности распараллеливания задачи (9) для различных значений v, построенные с использованием формулы (20). Указанные графики показывают, что величина $v = \lg\left(t_w/t_s\right)$ оказывает существенное влияние на эффективность распараллеливания. Чем больше соотношение t_w / t_s, тем выше эффективность распараллеливания.

Сумма $t_r + t_p$ также оказывает существенное влияние на эффективность распараллеливания. Это можно увидеть на рис. 10, где приведены графики эффективности распараллеливания задачи (9) при различных значениях суммы $t_r + t_p$, указанных на кривых. Параметр t_s в этом случае имеет фиксированное значение для всех графиков: $t_s = 10^{-v}t_w = 10^{-5} \cdot 10^{12} = 10^7$. Можно видеть, что при $t_r + t_p = 2 \cdot 10^{11}$ эффективность распараллеливания на 20 процессорных узлах не превышает 20%, при этом, как показывает рис. 7, верхняя граница масштабируемости *BSF*-программы с такими параметрами лежит в районе 300 процессорных узлов.

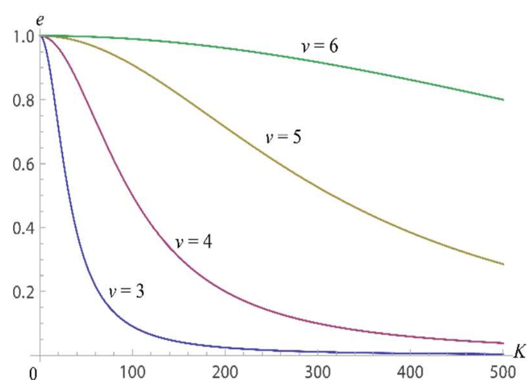


Рис. 9. Эффективность распараллеливания задачи (9) при различных v

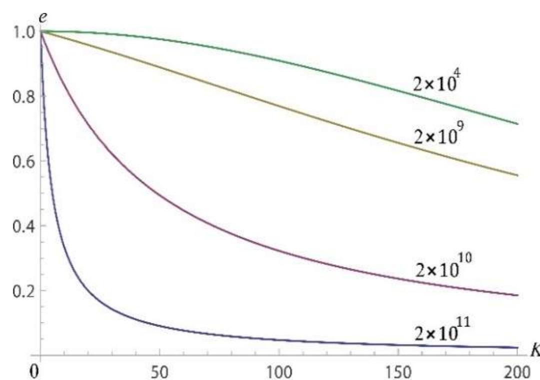


Рис. 10. Влияние t_p на эффективность распараллеливания $\left(v = 5\right)$

Заключение

В работе описана новая модель параллельных вычислений *BSF* (Bulk Synchronous Farm) — блочно-синхронная ферма, ориентированная на вычислительные системы с массовым параллелизмом, включающие в себя сотни тысяч процессорных узлов и имеющие экзафлопный уровень производительности. *BSF*-компьютер представляет собой множество однородных процессорных узлов с приватной памятью, соединенных сетью, позволяющей передавать данные от одного процессорного узла другому. Среди процессорных узлов выделяется один, называемый мастером. Остальные K узлов называются рабочими. *BSF*-компьютер работает по схеме *SPMD*. Описана структура *BSF*-программы. Построена стоимостная метрика для оценки времени выполнения *BSF*-программы. На основе предложенной стоимостной метрики получена оценка для верхней границы масштабируемости *BSF*-программ. Данная оценка позволяет сделать вывод, что *BSF*-программа будет обладать хорошей масштабируемостью, если временные затраты на посылку задания рабочему будут пропорциональны кубическому корню от суммарных временных затрат на решение задачи рабочими. Также получены формулы для оценки эффективности распараллеливания *BSF*-программ.

Для валидации теоретических результатов, полученных в этой статье, была выполнена *BSF*-реализация алгоритма *NSLP* [11], используемого для решения нестационарных сверхбольших задач линейного программирования на кластерных вычислительных системах. Реализация выполнялась на языке C++ с использованием библиотеки MPI. Тексты *BSF*-реализации алгоритма *NSLP* свободно доступны в сети Интернет по адресу: https://github.com/leonid-sokolinsky/*BSF-NSLP*. С использованием *BSF*-реализации алгоритма *NSLP* на вычислительном кластере «Торнадо ЮУрГУ» [21] были проведены вычислительные эксперименты по исследованию масштабируемости и эффективности этой параллельной программы. Результаты сравнения данных, полученных аналитически с помощью стоимостной метрики модели *BSF*, с данными, полученными в результате вычислительных экспериментов, приведены в статье [22]. Эти результаты показывают, что модель *BSF* позволяет с высокой точностью предсказывать верхнюю границу масштабируемости *BSF*-реализации алгоритма *NSLP*.

В работе [23] был построен эмулятор *BSF*-программ, моделирующий работу *BSF*-программы на реальной кластерной вычислительной системе. Эмулятор реализован

на языке C++ с использованием библиотеки MPI. В эмуляторе задаются числовые значения параметров модели *BSF*, приведенные в разделе 4. Эмулятор запускается на реальной кластерной вычислительной системе и имитирует работу *BSF*-программы, пересылая между матером и рабочими сообщения определенной длины, и симулируя вычислительную работу соответствующими циклами ожидания. Вычислительные эксперименты, проведенные на суперкомпьютере ЮУрГУ с использованием эмулятора *BSF*-программ, также показали хорошее соответствие показателей масштабируемости, полученных теоретически и экспериментально.

В рамках дальнейших исследований планируется решить следующие задачи:

1) разработать формализм для описания *BSF*-программ с использованием функций высшего порядка;
2) выполнить проектирование и реализацию каркаса для быстрой разработки *BSF*-программ на базе MPI (в виде библиотеки на языке C++);
3) провести вычислительные эксперименты на кластерной вычислительной системе с использованием известных итерационных методов для подтверждения адекватности модели *BSF*.

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. TOP500 Supercomputer Sites. URL: https://www.top500.org (дата обращения: 03.04.2017).
2. Sahni S., Vairaktarakis G. The Master-Slave Paradigm in Parallel Computer and Industrial Settings // Journal of Global Optimization. 1996. Vol. 9, No. 3–4. P. 357–377. DOI: 10.1007/BF00121679.
3. Silva L.M., Buyya R. Parallel Programming Models and Paradigms // High Performance Cluster Computing: Architectures and Systems. Vol. 2. 1999. P. 4–27.
4. Leung J.Y.-T., Zhao H. Scheduling Problems in Master-Slave Model // Annals of Operations Research. 2008. Vol. 159, No. 1. P. 215–231. DOI: 10.1007/s10479-007-0271-4.
5. Bouaziz R. et al. Efficient Parallel Multi-Objective Optimization for Real-Time Systems Software Design Exploration // Proceedings of the 27th International Symposium on Rapid System Prototyping — RSP'16, October 1–7, 2016, Pittsburgh, Pennsylvania, USA. P. 58–64. DOI: 10.1145/2990299.2990310.
6. Cantú-Paz E., Goldberg D.E. On the Scalability of Parallel Genetic Algorithms // Evolutionary Computation. 1999. vol. 7, no. 4. pp. 429–449. DOI: 10.1162/evco.1999.7.4.429
7. Depolli M., Trobec R., Filipič B. Asynchronous Master-Slave Parallelization of Differential Evolution for Multi-Objective Optimization // Evolutionary Computation. 2012. Vol. 21, No. 2. P. 1–31. DOI: 10.1162/EVCO_a_00076.
8. Mathias E.N. et al. DEVOpT: A Distributed Architecture Supporting Heuristic and Metaheuristic Optimization Methods // Proceedings of the ACM Symposium on Applied Computing, March 11–14, 2002, Madrid, Spain. ACM Press, 2002. P. 870–875. DOI: 10.1145/508791.508960.

9. Ершова А.В., Соколинская И.М. Параллельный алгоритм решения задачи сильной отделимости на основе фейеровских отображений // Вычислительные методы и программирование. 2011. Т. 12. С. 423–434.
10. Бурцев А.П. Параллельная обработка данных сейсморазведки с использованием расширенной модели Master-Slave // Суперкомпьютерные дни в России: Труды международной конференции (26–27 сентября 2016 г., г. Москва). М.: Изд-во МГУ, 2016. С. 887–895.
11. Sahni S. Scheduling Master-Slave Multiprocessor Systems // IEEE Transactions on Computers. 1996. Vol. 45, No. 10. P. 1195–1199. DOI: 10.1109/12.543712
12. Darema F. et al. A Single-Program-Multiple-Data Computational Model for EPEX/FORTRAN // Parallel Computing. 1988. Vol. 7, No. 1. P. 11–24. DOI: 10.1016/0167-8191(88)90094-4.
13. Gropp W., Lusk E., Skjellum A. Using MPI: Portable Parallel Programming with the Message-Passing Interface. Second Edition. MIT Press, 1999.
14. 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.
15. Goudreau M. et al. Towards Efficiency and Portability: Programming with the BSP Model // Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures — SPAA'96. New York, NY, USA: ACM Press, 1996. P. 1–12. DOI: 10.1145/237502.237503.
16. Cole M.I. Algorithmic Skeletons: Structured Management of Parallel Computation. Cambridge, MA, USA: MIT Press, 1991. 170 p.
17. Poldner M., Kuchen H. On Implementing the Farm Skeleton // Parallel Processing Letters. 2008. Vol. 18, No. 1. P. 117–131. DOI: 10.1142/S0129626408003260.
18. Padua D. et al. 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.
19. Bilardi G., Pietracaprina A., Pucci G. A Quantitative Measure of Portability with Application to Bandwidth-Latency Models for Parallel Computing // Euro-Par'99 Parallel Processing, Toulouse, France, August 31 – September 3, 1999. Proceedings. Lecture Notes in Computer Science, vol. 1685. Springer, Berlin, Heidelberg, 1999. P. 543–551. DOI: 10.1007/3-540-48311-X_76.
20. Соколинская И.М., Соколинский Л.Б. О решении задачи линейного программирования в эпоху больших данных // Параллельные вычислительные технологии – XI международная конференция, ПаВТ'2017, г. Казань, 3–7 апреля 2017 г. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2017. С. 471–484. http://omega.sp.susu.ru/pavt2017/short/014.pdf.
21. Костенецкий П.С., Сафонов А.Ю. Суперкомпьютерный комплекс ЮУрГУ // Параллельные вычислительные технологии (ПаВТ'2016): труды международной научной конференции (28 марта – 1 апреля 2016 г., г. Архангельск). Челябинск: Издательский центр ЮУрГУ, 2016. С. 561–573. http://ceur-ws.org/Vol-1576/119.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 (25–26 сентября 2017 г., г. Москва). М.: Изд-во МГУ, 2017. С. 319–332. http://russianscdays.org/files/pdf17/319.pdf.
23. Ezhova N. Verification of BSF Parallel Computational Model // 3rd Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists (Ural-PDC). CEUR

Workshop Proceedings. Vol-1990. CEUR-WS.org. P. 30–39. http://ceur-ws.org/Vol-1990/paper-04.pdf.

Ежова Надежда Александровна, аспирант, кафедра системного программирования, 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), ведущий научный сотрудник, отдел математического программирования Института математики и механики им. Н.Н. Красовского УрО РАН (Екатеринбург, Российская Федерация)

DOI: 10.14529/cmse180203

PARALLEL COMPUTATION MODEL FOR MULTIPROCESSOR SYSTEMS WITH DISTRIBUTED MEMORY

© 2018 N.A. Ezhova[1], L.B. Sokolinsky[1,2]

[1]*South Ural State University (pr. Lenina 76, Chelyabinsk, 454080 Russia),*
[2]*N.N. Krasovskii Institute of Mathematics and Mechanics (IMM UB RAS)*
(Kovalevskaya St. 16, Ekaterinburg, 620990 Russia)
E-mail: ezhovana@susu.ru, leonid.sokolinsky@susu.ru
Received: 12.03.2018

The emergence of powerful multiprocessor computing systems brings to the fore issues related to the development of frameworks (templates) that allow creating high-scalable parallel programs oriented to systems with distributed memory. In this case, the most important problem is the development of parallel computing models that allow us to assess its scalability at an early stage of the program design. General requirements for computational models are described and a new high-level parallel computing model called BSF is derived, which is an extension of the BSP model, and is based on the SPMD programming method and the «master-workers» framework. The BSF-model is oriented to computational systems with massive parallelism on distributed memory, including hundreds of thousands of processor nodes, and having an exaflop level of performance and numerical iterative methods with high time complexity. The BSF-computer architecture is defined, and the structure of the BSF-program is described. The formal cost metric, which provides parallel BSF-programs scalability upper bounds in context of distributed memory computing systems, is described. Also, formula for BSF-programs parallel efficiency are derived.

Keywords: parallel programming, parallel computation model, master-workers framework, BSF-model, time complexity, Bulk Synchronous Farm, scalability, multiprocessor systems with distributed memory.

FOR CITATION

Ezhova N.A., Sokolinsky L.B. Parallel Computational Model for Multiprocessor Systems with Distributed Memory. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2018. vol. 7, no. 2. pp. 32–49. (in Russian) DOI: 10.14529/cmse180203.

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cites.

References

1. TOP500 Supercomputer Sites. Available at: https://www.top500.org (accessed: 03.04.2017).
2. Sahni S., Vairaktarakis G. The Master-Slave Paradigm in Parallel Computer and Industrial Settings. Journal of Global Optimization. 1996. vol. 9, no. 3–4. pp. 357–377. DOI: 10.1007/BF00121679.
3. Silva L.M., Buyya R. Parallel Programming Models and Paradigms. High Performance Cluster Computing: Architectures and Systems. vol. 2. 1999. pp. 4–27.
4. Leung J.Y.-T., Zhao H. Scheduling Problems in Master-Slave Model. Annals of Operations Research. 2008. vol. 159, no. 1. pp. 215–231. DOI: 10.1007/s10479-007-0271-4.
5. Bouaziz R. et al. Efficient Parallel Multi-Objective Optimization for Real-Time Systems Software Design Exploration. Proceedings of the 27th International Symposium on Rapid System Prototyping — RSP'16, October 1–7, 2016, Pittsburgh, Pennsylvania, USA. pp. 58-64. DOI: 10.1145/2990299.2990310.
6. Cantú-Paz E., Goldberg D.E. On the Scalability of Parallel Genetic Algorithms. Evolutionary Computation. 1999. vol. 7, no. 4. pp. 429–449. DOI: 10.1162/evco.1999.7.4.429.
7. Depolli M., Trobec R., Filipič B. Asynchronous Master-Slave Parallelization of Differential Evolution for Multi-Objective Optimization. Evolutionary Computation. 2012. vol. 21, no. 2. pp. 1–31. DOI: 10.1162/EVCO_a_00076.
8. Mathias E.N. et al. DEVOpT: A Distributed Architecture Supporting Heuristic and Metaheuristic Optimization Methods. Proceedings of the ACM Symposium on Applied Computing, March 11–14, 2002, Madrid, Spain. ACM Press, 2002. pp. 870–875. DOI: 10.1145/508791.508960.
9. Ershova A.V., Sokolinskaya I.M. Parallel Algorithm for Solving the Strong Separability Problem on the Basis of Fejér Mappings. Numerical Methods and Programming. 2011. vol. 12. pp. 423–434. (in Russian)
10. Burtsev A.P. Parallel Processing of Seismic Data Using the Extended Master-Slave Model. *Supercomputernie dni v Rossii: Trudy meghdunarodnoj konferentsii (Moscva, 26–27 sentyabra 2016)* [Russian Supercomputing Days: Proceedings of the International Scintific Conference (Moscow, Russia, September 26 –27, 2016)]. Moscow, Publishing House of Moscow State University, 2016. pp. 887–895. (in Russian)
11. Sahni S. Scheduling Master-Slave Multiprocessor Systems. IEEE Transactions on Computers. 1996. vol. 45, no. 10. pp. 1195–1199. DOI: 10.1109/12.543712.
12. Darema F. et al. A Single-Program-Multiple-Data Computational Model for EPEX/FORTRAN. Parallel Computing. 1988. vol. 7, no. 1. pp. 11–24. DOI: 10.1016/0167-8191(88)90094-4.
13. Gropp W., Lusk E., Skjellum A. Using MPI: Portable Parallel Programming with the Message-Passing Interface. Second Edition. MIT Press, 1999.
14. 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.
15. Goudreau M. et al. Towards Efficiency and Portability: Programming with the BSP Model. Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures — SPAA'96. New York, NY, USA: ACM Press, 1996. pp. 1–12. DOI: 10.1145/237502.237503.

16. Cole M.I. Algorithmic Skeletons: Structured Management of Parallel Computation. Cambridge, MA, USA: MIT Press, 1991. 170 p.
17. Poldner M., Kuchen H. On Implementing the Farm Skeleton. Parallel Processing Letters. 2008. vol. 18, no. 1. pp. 117–131. DOI: 10.1142/S0129626408003260.
18. Padua D. et al. 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.
19. Bilardi G., Pietracaprina A., Pucci G. A Quantitative Measure of Portability with Application to Bandwidth-Latency Models for Parallel Computing. Euro-Par'99 Parallel Processing, Toulouse, France, August 31 – September 3, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1685. Springer, Berlin, Heidelberg, 1999. pp. 543–551. DOI: 10.1007/3-540-48311-X_76.
20. Sokolinskaya I.M., Sokolinsky L.B. On the Solution of the Problem of Linear Programming in the Era of Large Data. *Parallel'nye vychislitel'nye tekhnologii — XI mezhdunarodnaya konferentsiya, PaVT'2017, g. Kazan, 3–7 aprelya 2017 g. Korotkie stat'i i opisaniya plakatov.* [Short Articles and Posters of the XI International Conference on Parallel Computational Technologies (PCT'2017), Kazan, 3–7 April 2017]. Chelyabinsk, Publishing Center of the South Ural State University, 2017. pp. 471–484. http://omega.sp.susu.ru/pavt2017/short/014.pdf. (in Russian)
21. Kostenetskiy P.S., Safonov A.Y. SUSU Supercomputer Resources. Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). Arkhangelsk, Russia, March 29–31, 2016. CEUR Workshop Proceedings. 2016. vol. 1576. pp. 561–563. http://ceur-ws.org/Vol-1576/119.pdf. (in Russian)
22. Sokolinskaya I., Sokolinsky L.B. Scalability Evaluation of the *NSLP* Algorithm for Solving Non-Stationary Linear Programming Problems on Cluster Computing Systems. *Supercomputernie dni v Rossii: Trudy meghdunarodnoj konferentsii (Moscva, 25–26 sentyabra 2017)* [Russian Supercomputing Days: Proceedings of the International Scintific Conference (Moscow, Russia, September 25–26, 2017)]. Moscow, Publishing House of Moscow State University, 2017. pp. 319–332. http://russianscdays.org/files/pdf17/319.pdf.
23. Ezhova N. Verification of BSF Parallel Computational Model. 3rd Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists (Ural-PDC). CEUR Workshop Proceedings. Vol-1990. CEUR-WS.org. pp. 30–39. http://ceur-ws.org/Vol-1990/paper-04.pdf.

УДК 519.687.1 DOI: 10.14529/cmse180204

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

© 2018 А.В. Мукосей, А.С. Семенов

АО «НИЦЭВТ»

(117587 Москва, Варшавское шоссе, д. 125, стр. 15)

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

Поступила в редакцию: 10.05.2018

В данной работе рассматривается высокоскоростная вычислительная сеть 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. В данной работе на основе идей из задачи о многомерной упаковке контейнера предложен метод поиска узлов с оценкой фрагментированности системы. Для такой оценки введено понятие 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. 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. В качестве критерия качества метода выбора узлов рассматривается средняя утилизация ресурсов вычислительной системы и среднее время ожидания заданий в очереди. Исследование показало, что увеличение утилизации ресурсов для предложенного метода выбора узлов составило в среднем 11 % по сравнению с базовым методом, а среднее значение времени нахождения задания в очереди сокращенно на 45,3 %.

Ключевые слова: коммуникационная сеть Ангара, многомерный тор, правило порядка направлений, фрагментация вычислительной системы, выбор узлов.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Мукосей А.Б., Семенов А.С. Оптимизация фрагментации при выделении ресурсов для высокопроизводительных вычислительных систем с сетью Ангара // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2018. Т. 7, № 2. С. 50–62. DOI: 10.14529/cmse180204.

Введение

Фрагментация, возникающая в результате последовательного выделения вычислительных узлов в многоузловой системе играет критическую роль в эффективности использования суперкомпьютера. Особенное значение эта проблема имеет для сетей с топологией типа тор. На данный момент существует две стратегии выделения ресурсов, используемых в суперкомпьютерах с тороидальной сетью [1]. В суперкомпьютерах IBM Blue Gene/P, Blue Gene/Q предоставление ресурсов основано на разделение системы на партиции, по которым размещаются задачи пользователей [2, 3]. Такая стратегия может

*Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2018»

порождать фрагментацию, возникающую в результате выделения большего числа узлов, чем требовалось или невозможности выделить доступный набор узлов из разных партиций.

Вторая стратегия используется в серии суперкомпьютеров Cray XT/XE, где расположение выделенных узлов [4] не зависит от топологии. Такой способ выделения ресурсов может привести к деградации производительности ввиду наличия конкурирующего трафика. В статье [1] алгоритм оптимизации упаковки заданий использует линейную нумерацию узлов, однако это неудобно для сети Ангара.

В АО «НИЦЭВТ» разработана высокоскоростная коммуникационная сеть Ангара [5, 6] с топологией «многомерный тор». В маршрутизаторе сети реализована бездедлоковая, адаптивная маршрутизация, основанная на правилах «пузырька» (Bubble flow control, [7]) и «порядка направлений» (Direction ordered routing, DOR, [8, 9]) с использованием битов направлений [9]. Благодаря алгоритму First Step/Last Step «нестандартного первого и последнего шага» [9] аппаратно поддерживается обход отказавших узлов или линков. Эффективность этого метода по поддержанию связности в сети с отказами была показана в статье [10].

Для сети Ангара необходимо разработать алгоритм оптимизации фрагментации при условии отсуствия пересечения трафика различных задач. Такая задача носит название упаковки контейнера и является NP-полной задачей. Существуют различные способы решения такой задачи: авторы статьи [11] предлагают решать такую задачу муравьиным алгоритмом — полиномиальный алгоритмов метаэвристической оптимизации для нахождения приближенных решений, который основан на использовании модели поведения муравьев, ищущих пути от колонии к источнику питания. Авторы статьи [12] разработали различные алгоритмы упаковки с использованием генетического алгоритма.

В данной статье предлагается алгоритм выбора узлов для суперкомпьютеров с топологией «многомерный тор» с учетом расположения заданий в топологии в коммуникационной сети. Разработанный алгоритм на рассмотренных системах и предложенных очередях в среднем увеличил утилизацию ресурсов на 11 %, а значение времени нахождения заданий в очереди сократил на 57 % по сравнению с первоначальным вариантом алгоритма, использовавшемся для выбора узлов.

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. В заключении приводится краткая сводка результатов, полученных в работе, и намечаются направления дальнейших исследований.

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 [13] 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 $T_{mid} = \frac{\sum_{i=1}^{k} T^i_{delay}}{k} = \frac{1}{k}\sum_{i=1}^{k}\frac{Q^i}{W^i_{time}}$, 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 [14].

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.

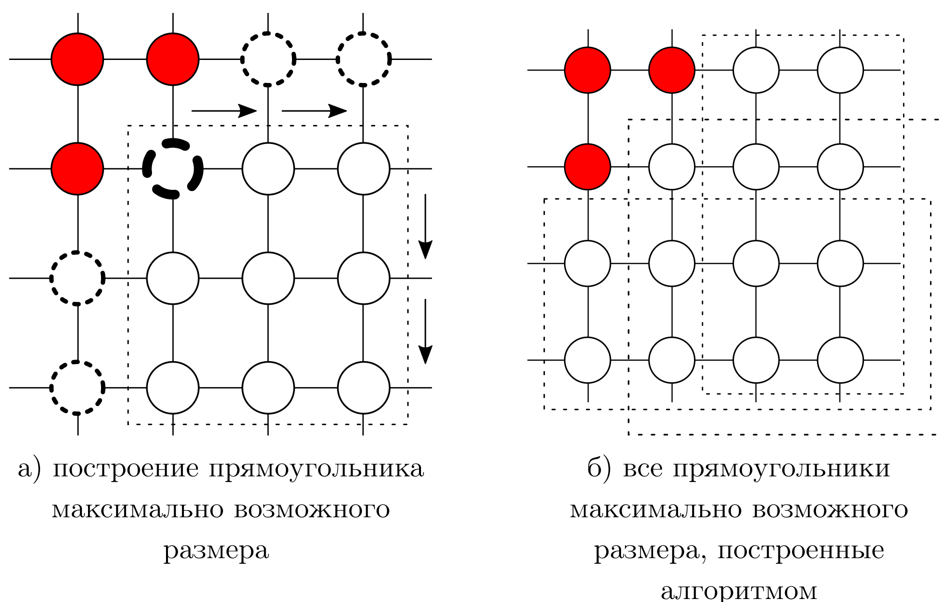
2. Алгоритм выбора узлов

Для решения поставленной проблемы в статье предложен алгоритм выбора узлов, основанный на методах, предложенных в работе [12], посвященной трехмерной упаковке контейнера. Идея алгоритма выбора узлов заключается в расположении задания таким образом, чтобы максимизировать оставшееся пространство в многомерном торе.

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 (расширять некуда), 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 $N_{free} \backslash \bigcup\limits_{iter=1}^{Iters} MSS_{iter}$ не пусто, где $Iters$ — число итераций алгоритма. Псевдокод алгоритма представлен на рис. 2. 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

Иллюстрация работы алгоритма приведена на рис. 1а и рис. 1б, на которых в двумерной решетке узлы множества N_{locked} закрашены, а свободные узлы N_{free} — нет. Жирным контуром на рис. 1а 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. 1б.

```
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

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 [13]. 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. Заметим, что в данной работе не рассматривалась возможность обгона заданий, то есть предоставление ресурсов заданию, имеющему более позднее время W_{time}, если заданию с ранним временем W_{time} не были выделены ресурсы. 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_{mid}.

Введем некоторые формальные определения, необходимые для описания работы симулятора. *Очередью* симулятора назовем набор заданий из потока, для которых не выделялись ресурсы и время их запуска T_{start} меньше текущего симулируемого времени t. *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 очередь симулятора не пуста, симулятор выполняет процедуру поиска маршрутизируемого множества для каждого задания из очереди 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, 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 [15], представленных в таблице.

Таблица

Моделируемые системы

Количество узлов вычислительной системы	**Топология 3х-мерный тор**	**Топология 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 вероятностей таких вероятностей представлено на рис. 3. На рис. 3б представлено реальное распределение пользовательских задач (заданий) по количеству узлов, полученное с гибридного суперкомпьютера Desmos на базе сети Ангара, имеющую топологию 4х-мерный тор 4x2x2x2. В дальнейшем системы с данной очередью будем помечать символом s — 4x2x2x2s и 4x4x2s. 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.

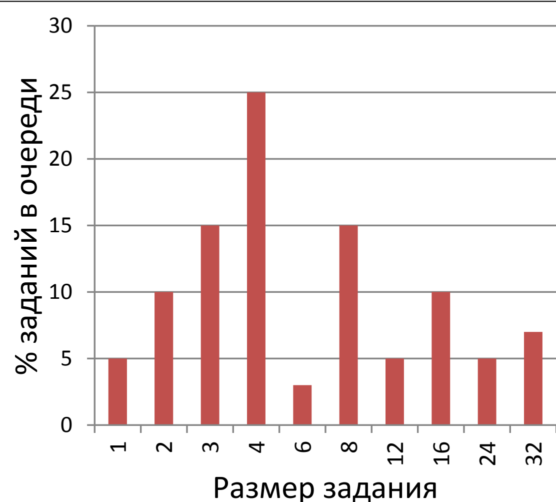
Вероятности для остальных числов узлов равны 0. Задания равномерно случайно размещаются на временной шкале в диапазоне $[0; 40500]$ секунд вне зависимости от числа требуемых узлов. Время продолжительности заданий равномерно распределено по диапазону $[50; 100]$.

Для исследования разработанного алгоритма поиск ресурсов для заданий производился четырьмя 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 ($base$).

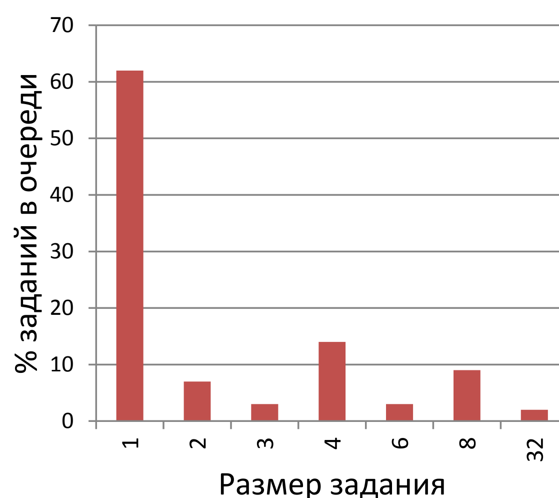
Диаграмма сравнения утилизации для различных методов и систем представлена на рис. 4а. Сравнение средних значений времени нахождения задания в очереди относительно запрошенного времени (T_{mid}) для различных методов и систем представлено на рис. 4б.

Разработанный алгоритм $Find_Systems$ из статьи [13] сократил в среднем значение времени нахождения заданий в очереди на $45,3$ %, а утилизация ресурсов кластера в среднем увеличилась на $8,4$ % относительно базового метода $base$.

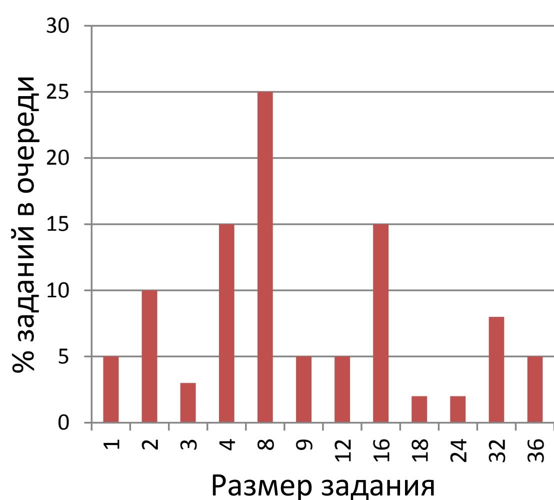
Добавление в $Find_Systems$ учета фрагментированности вычислительной системы позволило еще сократить значение времени нахождения заданий в очереди до 57 % относительно метода $base$, что на $11,7$ % больше чем у метода $Find_Systems$. Утилизация



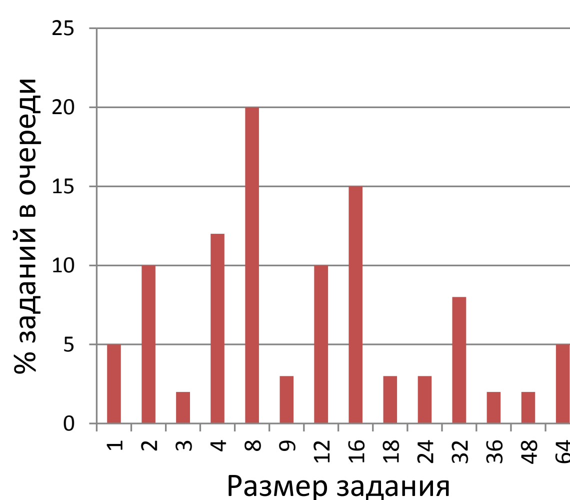
а) для систем из 32 узлов



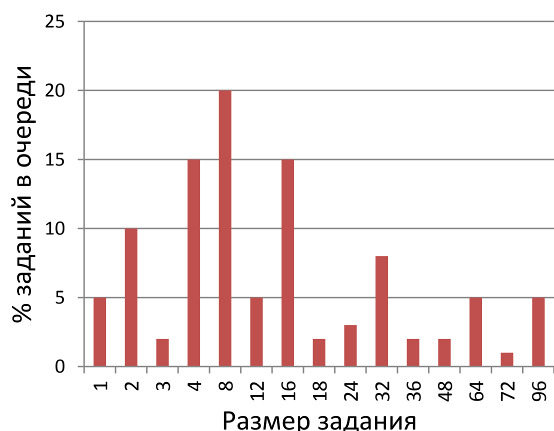
б) для систем из 32 узлов, кластер Desmos



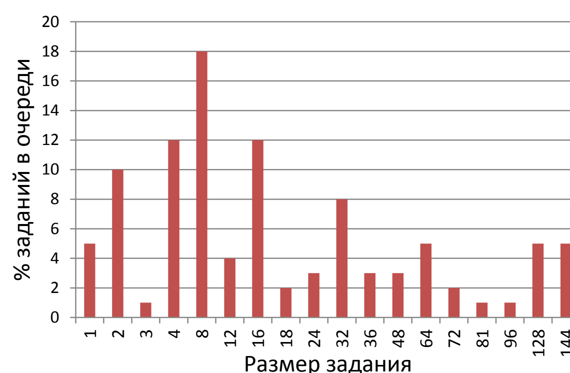
в) для систем из 36 узлов



г) для систем из 64 узлов



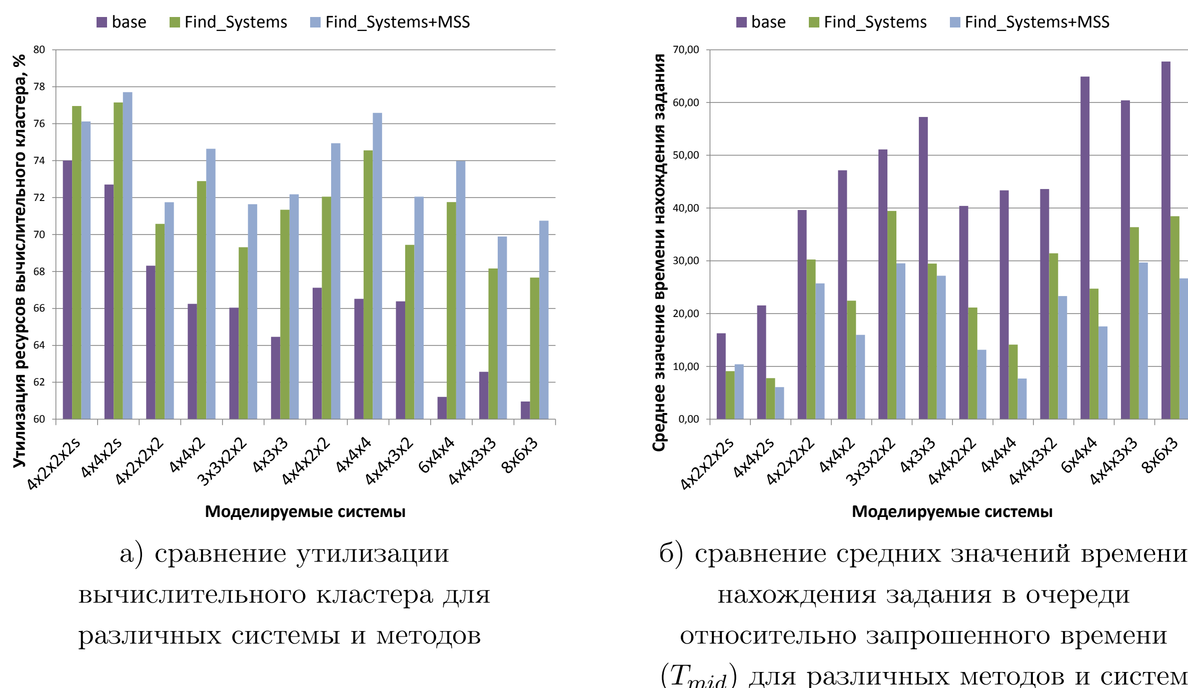
д) для систем из 96 узлов



е) для систем из 144 узлов

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

ресурсов при этом увеличивается до 11 % относительно метода *base*, что на 2,6 % больше чем у метода *Find_Systems*.



а) сравнение утилизации вычислительного кластера для различных системы и методов

б) сравнение средних значений времени нахождения задания в очереди относительно запрошенного времени (T_{mid}) для различных методов и систем

Рис. 4. Выделение прямоугольников максимального размера

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 экспериментальное исследование разработанного алгоритма для различных 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. Эксперимент проводился при помощи разработанного симулятора вычислительной системы. Для каждой исследуемой вычислительной системы была создана синтетическая очередь заданий, параметры которой приближены к реальным.

Средняя утилизация ресурсов для разработанного алгоритма выбора узлов увеличилась на 11 %, а значение времени нахождения заданий в очереди сократилось на 57 % по сравнению с первоначальным вариантом алгоритма.

В дальнейшем необходимо провести исследование разработанного алгоритма для систем большого размера, а также детально исследовать и провести возможную оптимизацию времени работы алгоритма. Также возможно провести исследование о влиянии перестановки задач в очереди для улучшения характеристик использования вычислительной системы.

Литература

1. 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) (Madrid, Spain, September 22–26, 2014), 2014. P. 140–148. DOI: 10.1109/CLUSTER.2014.6968751.

2. IBM Redbooks Publication: IBM System Blue Gene Solution: Blue Gene/Q System Administration. 2013. 282 p.

3. 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) (Anchorage, Alaska, USA, May 16–20, 2011), 2011. P. 828–839. DOI: 10.1109/IPDPS.2011.82.

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

5. 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 (26–27 сентября 2016 г., г. Москва). М.: Изд-во МГУ, 2016. С. 626–639.

6. 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.

7. 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 (Wakamatsu, Japan, September 21–24, 1999), 1999. P. 58–67. DOI: 10.1109/ICPP.1999.797388.

8. 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.

9. Scott S.L. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. 1996.

10. 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.

11. Кагиров Р.Р. Многомерная задача о рюкзаке: новые методы решения // Вестник Сибирского государственного аэрокосмического университета им. академика М.Ф. Решетнева. 2007. № 3. С. 16–20.

12. 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.

13. 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, № 1. С. 53–64.

14. 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.

15. Полежаев П.Н. Симулятор вычислительного кластера и его управляющей системы, используемый для исследования алгоритмов планирования задач // Вестник ЮУрГУ. Серия: Математическое моделирование и программирование. 2010. Т. 6, № 35(211). С. 79–90.

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)

DOI: 10.14529/cmse180204

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

© 2018 A.V. Mukosey, A.S. Semenov

JSC «NICEVT»

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

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

Received: 10.05.2018

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 compute nodes in a supercomputer provided that network traffic from different user's tasks should not overlap. This paper proposes a method for searching nodes with an evaluation of the fragmentation of the system based on ideas from multidimensional container loading problem. For such an evaluation, the concept of rectangles is introduced, which can be inscribed into the system after placing the next user task. Each set of nodes that is suitable for placing the task is evaluated by the proposed function taking into account the size and the number of found rectangles of maximum size. The proposed method was evaluated using computer system model. A set of different computer systems with three-dimensional and four-dimensional topologies was considered. The minimum system size is 32 compute 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 a real ones. 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. The study showed that the increase of the resources utilization for the proposed method averaged 11 % compared to the base method, and the average time spent in queue is reduced by 45,3 %.

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

FOR CITATION

Mukosey A.V., Semenov A.S. Allocation Optimization for Reducing Resource Fragmentation in Angara High-speed Interconnect. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2018. vol. 7, no. 2. pp. 50–62. (in Russian) DOI: 10.14529/cmse180204.

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

References

1. 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) (Madrid, Spain, September 22–26, 2014). 2014. pp. 140–148. DOI: 10.1109/CLUSTER.2014.6968751.
2. IBM Redbooks Publication: IBM System Blue Gene Solution: Blue Gene/Q system administration. 2013. 282 p.
3. 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) (Anchorage, Alaska, USA, May 16–20, 2011), 2011. pp. 828–839. DOI: 10.1109/IPDPS.2011.82.
4. Cray Document: Managing System Software for Cray XE and Cray XT Systems. 2010.
5. Agarkov A.A., Ismagilov T.F., Makagon D.V. Performance Evaluation of the Angara Interconnect. *Supercomputernie dni v Rossii: Trudy meghdunarodnoj konferentsii (Moscva, 26–27 sentyabra 2016)* [Russian Supercomputing Days: Proceedings of the International Scintific Conference (Moscow, Russia, September 26–27, 2016)]. Moscow, Publishing House of Moscow State University, 2016. pp. 626–639. (in Russian)
6. 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 technologii* [Science Technologies]. 2014. vol. 15, no. 1. pp. 21–28. (in Russian)
7. 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 (Wakamatsu, Japan, September 21–24, 1999), 1999. pp. 58–67. DOI: 10.1109/ICPP.1999.797388.
8. 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.
9. Scott S.L., et al. The Cray T3E Network: Adaptive Routing in a High. Performance 3D Torus. 1996.
10. Pozhilov I.A., Semenov A.S., Makagon D.V. Connectivity Problem Solution for Direction Ordered Deterministic Routing in nD Torus. *Programmnaya inzheneriya* [Software Engineering]. 2015. no. 3. pp. 13–19. (in Russian)
11. Kagirov R.R. Multiple Knapsack Problem: New Solving Methods. *Vestnik SibGAU* [Vestnik of the Reshetnev Siberian State Aerospace University]. 2007. no. 3. pp. 16–20. (in Russian)
12. 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.
13. Mukosey A.V., Semenov A.S. An Approximate Algorithm for Choosing the Optimal Subset of Nodes in the Angara Interconnect with Failures. *Vyichislitelnyie metodyi i programmirovanie* [Numerical methods and Programming]. 2017. vol. 18, no. 1. pp. 53–64. (in Russian)

14. 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 Yuzhno-Ural'skogo Gosudarstvennogo Universiteta. Seriya "Vychislitelnaya Matematika i Informatika"* [Bulletin of South Ural State University. Series: Computational Mathematics and Software Engineering]. 2014. vol. 3, no. 4. pp. 75–84. DOI: 10.14529/cmse140405. (in Russian)

15. Polezhaev P.N. Simulator of Computer Cluster and Its Management System Used for Research of Job Scheduling Algorithms. *Vestnik Yuzhno-Ural'skogo Gosudarstvennogo Universiteta. Seriya "Matematicheskoe Modelirovanie i Programmirovanie"* [Bulletin of South Ural State University. Series: Mathematical Modeling, Programming & Computer Software]. 2010. vol. 6, no. 35(211). pp. 79–90. (in Russian)

УДК 004.021 DOI: 10.14529/cmse180205

РАСПРЕДЕЛЕННЫЙ АЛГОРИТМ ОТОБРАЖЕНИЯ РАСПРЕДЕЛЕННЫХ МНОГОМЕРНЫХ ДАННЫХ НА МНОГОМЕРНЫЙ МУЛЬТИКОМПЬЮТЕР В СИСТЕМЕ ФРАГМЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ LUNA*

© 2018 Г.А. Щукин

Институт вычислительной математики и математической геофизики СО РАН
(630090 Новосибирск, пр. Академика Лаврентьева, д. 6),
Новосибирский государственный технический университет
(630073 Новосибирск, пр. К.Маркса, д. 20)
E-mail: schukin@ssd.sscc.ru

Поступила в редакцию: 22.02.2018

В статье рассматривается распределенный алгоритм с локальными взаимодействиями Patch, предназначенный для управления распределением данных и динамической балансировки нагрузки в системе фрагментированного программирования LuNA. Система LuNA используется для упрощения создания параллельных реализаций крупномасштабных численных моделей для распределенных вычислительных систем. Фрагментированная программа в системе LuNA выполняется под управлением исполнительной системы, которая использует различные алгоритмы распределения данных и вычислений для обеспечения эффективного (в плане времени исполнения и потребления ресурсов) исполнения программы. Разработанный для использования в системе LuNA распределенный алгоритм Patch предназначен для случая распределения многомерных сеток данных на многомерной решетке вычислительных узлов. Алгоритм использует отображение данных на многомерную решетку ячеек (координат), которые затем распределяются между вычислительными узлами мультикомпьютера. Такое отображение позволяет алгоритму учитывать зависимости между данными и сохранять локальность данных при динамической балансировке нагрузки. Тестирование алгоритма Patch на фрагментированной реализации реальной вычислительной задачи показало его преимущество над использовавшимся ранее в системе LuNA алгоритме Rope, в виде уменьшения суммарного объема и дальности коммуникаций между вычислительными узлами в ходе исполнения программы.

Ключевые слова: распределенные алгоритмы, распределение данных, динамическая балансировка нагрузки, технология фрагментированного программирования, система фрагментированного программирования LuNA.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Щукин Г.А. Распределенный алгоритм отображения распределенных многомерных данных на многомерный мультикомпьютер в системе фрагментированного программирования LuNA // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2018. Т. 7, № 2. С. 63–76. DOI: 10.14529/cmse180205.

Введение

В настоящее время с увеличением производительности распределенных вычислительных систем наблюдается рост использования крупномасштабных численных моделей, особенно в науке. Для достижения хорошей производительности и масштабируемости параллельных программ численного моделирования на вычислительных

*Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2018».

системах, состоящих из сотен и тысяч вычислительных узлов и процессорных ядер, требуются эффективные стратегии распределения ресурсов (данных и вычислений) и динамической балансировки нагрузки. Реализация таких стратегий для каждого прикладного параллельного приложения обладает высокой сложностью, сопоставимой со сложностью системного параллельного программирования. Ввиду этого, была разработана система фрагментированного программирования LuNA [1–6], позволяющая упростить разработку параллельных программ численного моделирования, могущих показать хорошую производительность на распределенных вычислительных системах.

В системе LuNA параллельная программа конструируется из элементов (фрагментов) данных и вычислений, которые описываются прикладным программистом. Каждый фрагмент вычислений (ФВ) определяет независимый процесс, вычисляющий выходные фрагменты данных (ФД) из входных фрагментов данных. Каждый фрагмент данных получает свое значение только один раз и каждый фрагмент вычислений исполняется только один раз. Представление программы в виде множества фрагментов сохраняется во время ее исполнения, что позволяет распределять фрагменты вычислений по вычислительным узлам и исполнять их параллельно, а также обеспечивать динамическую балансировку нагрузки путем миграции фрагментов данных и вычислений между узлами мультикомпьютера.

Эффективность исполнения фрагментированной программы значительно зависит от качества распределения ресурсов, т.е. данных и вычислений. Качественное распределение ресурсов должно обеспечивать равномерную нагрузку вычислительных узлов и минимизировать коммуникации между узлами. В статье описывается распределенный алгоритм с локальными взаимодействиями Patch, предназначенный для распределения ресурсов в системе LuNA и оптимизированный для случая распределения многомерных сеток данных на многомерной решетке вычислительных узлов.

Статья организована следующим образом. В разделе 1 представлен обзор родственных работ и алгоритмов. В разделе 2 представлено исполнение фрагментированной программы в системе LuNA. Раздел 3 содержит описание разработанных алгоритмов, подраздел 3.2 посвящен описанию алгоритма Rope, 3.3 — алгоритму Patch. Описание динамической балансировки нагрузки в разработанных алгоритмах приведено в подразделе 3.4. Раздел 4 содержит результаты тестирования алгоритмов Rope и Patch. Достигнутые результаты и направления дальнейшего развития исследования приведены в заключении.

1. Обзор родственных работ

Существует много способов декомпозиции данных. Один из них — «плиточные» массивы (tiled arrays) [7–10]. Массив данных сначала разбивается на прямоугольные блоки («плитки», тайлы), затем эти блоки распределяются по узлам мультикомпьютера. Разбиение на тайлы может быть иерархическим, как показано, например, в работах [7] и [9], регулярным (все тайлы имеют одинаковый размер и выровнены относительно друг друга) или произвольным (см. [10]). Некоторые системы, использующие «плиточные» массивы, позволяют пользователю задавать свой алгоритм разбиения на тайлы (см. [8]). Одним из ограничений «плиточного» подхода является требование прямоугольной формы плиток, что в некоторых случаях может помешать осуществить сбалансированное разбиение данных на плитки. Другим ограничением является предполагаемая статичность разбиения, что

подразумевает невозможность его динамического изменения в ходе исполнения программы, в т.ч. распределенным способом.

Различные методы декомпозиции расчетной области широко используются в вычислительных методах, например, в приложениях молекулярной динамики и симуляциях частиц [11–15]. В работах [11] и [13] расчетная область — поле частиц — разбивается на домены путем помещения специальных внутренних вершин внутрь области, на регулярном расстоянии друг от друга, для создания сетки доменов; каждый домен имеет изначально прямоугольную форму и назначается на отдельный узел мультикомпьютера. Для осуществления динамической балансировки нагрузки каждая внутренняя вершина может сдвигаться, изменяя форму смежных доменов, что приводит к изменению количества частиц в доменах и соответственно к изменению нагрузки соответствующих узлов мультикомпьютера. Балансировка может производиться распределенным способом, когда каждый узел взаимодействует только со своими соседними узлами в топологии и их доменами. Тем не менее, в случае трехмерной решетки доменов каждая внутренняя вершина может принадлежать до 8-и смежным доменам одновременно, что в свою очередь приводит к необходимости синхронизации между 8-ю вычислительными узлами для сдвига этой вершины. Дополнительным ограничением является требование выпуклой формы доменов.

В работе [14] применяется алгоритм рекурсивной ортогональной бисекции. Расчетная область рекурсивно разбивается на домены плоскостями бисекции, получившиеся домены затем назначаются на разные вычислительные узлы. Для балансировки нагрузки плоскости бисекции сдвигаются, чтобы изменить размер доменов. Недостатком является то, что сдвиг плоскостей бисекции может потребовать рекурсивного обхода всего дерева бисекции и привести к коммуникациям между неограниченным числом узлов.

В статье [12] описываются домены, состоящие из ячеек Воронова. Ячейки Воронова на границах доменов могут мигрировать между доменами для изменения их формы и достижения баланса нагрузки. В [15] аналогичным образом используются обычные прямоугольные ячейки. Некоторые ячейки обозначаются как постоянные и не могут покидать свой домен, что сделано для того, чтобы каждый узел взаимодействовал только с фиксированным числом соседних узлов и картина коммуникаций не менялась.

Минусом алгоритмов распределения данных и динамической балансировки нагрузки, описанных в вышеописанных работах, является их направленность на определенный круг задач (методы молекулярной динамики и т.п.). Для их адаптации к более широкому кругу задач может потребоваться существенная переработка.

В итоге можно заключить, что основными чертами, которые желательно требовать от алгоритмов распределения данных и балансировки нагрузки, являются их распределенность, использование локальных коммуникаций и применимость ко многим классам вычислительных задач.

2. Исполнение фрагментированной программы в системе LuNA

Фрагментированная программа представляет из себя множество фрагментов данных и вычислений. Фрагменты вычислений подразделяются на атомарные, которые принимают на вход входные фрагменты данных и вычисляют по ним выходные фрагменты данных, с

помощью вызовы заданного программистом кода, и структурированные, представляющие из себя множество других атомарных или структурированных ФВ.

Исполнение фрагментированной программы в системе LuNA проходит в режиме полуинтерпретации. Исполнительная система LuNA выбирает готовые фрагменты вычислений — те, для которых вычислены их входные фрагменты данных — и запускает их. При запуске структурированного ФВ порождаются содержащиеся в нем дочерние ФВ, при запуске атомарного — выходные ФД. Для каждого порожденного фрагмента данных или вычислений исполнительная система должна решить, на каком узле хранить этот фрагмент данных или на каком узле исполнить этот фрагмент вычислений. Для этого исполнительная система LuNA может использовать различные алгоритмы распределения ресурсов, основные два из которых будут описаны в следующих секциях.

3. Распределенные алгоритмы распределения данных

Распределенный алгоритм Rope [1, 2] был изначально разработан для распределения данных и динамической балансировки нагрузки в системе LuNA. Алгоритм Rope поддерживает распределение произвольных структур данных на вычислительной сети с произвольной топологией. Тем не менее из-за своей универсальности он обеспечивал не самое эффективное распределение для случая многомерных структур данных. Для исправления недостатков алгоритма Rope был разработан алгоритм Patch. В следующих разделах представлено описание и сравнение этих двух алгоритмов.

3.1. Вспомогательные определения

Численные методы зачастую работают с многомерными массивами (сетками) данных (метод частиц-в-ячейках, итерационные методы решения дифференциальных уравнений с помощью конечно-разностной схемы и т.д.). При декомпозиции данных сетка разбивается на блоки — фрагменты, в итоге получается сетка фрагментов данных. Фрагменты данных, смежные в сетке, будем называть смежными фрагментами данных. Два фрагмента данных называются соседними, если значение одного ФД вычисляется на основе значения другого ФД, либо значения обоих ФД используются для вычисления значения третьего ФД, т.е. соседние фрагменты данных связаны зависимостью через некоторый фрагмент вычислений. В численных методах зачастую смежные фрагменты данных являются соседними и наоборот.

В системе LuNA каждый фрагмент данных и вычислений распределяется на вычислительный узел распределенной вычислительной системы динамически, т.е. по ходу исполнения программы. Ввиду того, что распределение фрагментов данных не статическое и может меняться со временем (например, из-за динамической балансировки нагрузки), требуется способ определения текущего местоположения фрагментов данных для возможности их запроса фрагментами вычислений на любом вычислительном узле. Узел, на котором должен храниться фрагмент данных, будем называть резиденцией этого фрагмента данных. При создании фрагмента данных для него определяется его резиденция, после чего он должен быть перемещен на этот узел и храниться на нем, а также может быть запрошен (скопирован) с него при необходимости. Таким образом, проблема распределения и поиска фрагмента данных сводится к проблеме определения его резиденции с любого узла. Стоит отметить, что при «хорошем» распределении фрагментов данных и вычислений число запросов фрагментов данных с других узлов должно быть минимальным.

3.2. Алгоритм Rope

Для распределения фрагментов данных алгоритм Rope [1, 2] использует отображение сетки фрагментов данных на одномерный числовой диапазон, например, с помощью пространственной кривой Гильберта (рис. 1, [16]). Каждому фрагменту данных назначается его (целочисленная) координата в диапазоне. Отображение фрагментов на диапазон делается таким образом, чтобы соседние фрагменты данных отображались на одну и ту же координату или на близкие координаты в диапазоне (использование кривой Гильберта позволяет обеспечить выполнение этого условия в той или иной мере). Отображение фиксируется перед началом исполнения фрагментированной программы и не меняется в ходе ее исполнения.

Для распределения фрагментов данных по узлам мультикомпьютера диапазон разбивается на непересекающиеся смежные сегменты по числу вычислительных узлов, каждый узел получает свой сегмент. Предполагается, что узлы объединены в линейную топологию, что позволяет осуществить отображение сегментов на узлы один-в-один. Резиденцией ФД является тот узел, сегменту которого в данный момент принадлежит координата этого ФД. Таким образом, поиск резиденции каждого ФД заключается в поиске узла с нужной координатой. Так как координаты упорядочены по узлам, поиск сегмента может быть осуществлен с помощью прохода по линейке узлов, с использованием только локальных взаимодействий между вычислительными узлами.

Если соседние ФД отображаются на одну и ту же или близкие координаты, они распределяются на один и тот же или близкие в топологии узлы, таким образом обеспечивая локальность коммуникаций. К сожалению, не всегда удается построить такое отображение, поэтому для таких случаев был разработан новый алгоритм Patch.

3.3. Алгоритм Patch

Алгоритм Rope не позволяет полностью сохранить соседство фрагментов данных по всем измерениям для многомерных сеток фрагментов данных ввиду использования одномерного диапазона для отображения: некоторые соседние фрагменты данных будут отображены на далеко отстоящие друг от друга координаты в диапазоне, и, следовательно, будут распределены на далеко отстоящие друг от друга в линейной топологии вычислительные узлы. В этом случае требуется отображение на многомерную область координат, которая позволит учитывать соседство фрагментов данных по многим координатам одновременно. Такая многомерная область координат должна отображаться на многомерную решетку вычислительных узлов, размерностью не большей чем сама область. В итоге все соседние фрагменты данных будут расположены на одном и том же или соседних в топологии вычислительных узлах. Новый алгоритм Patch реализует такое отображение.

Было рассмотрено несколько способов представления многомерной области координат и ее декомпозиции на домены: помещение контрольных вершин внутри области, использование плоскостей разбиения и т.д. Учитывались следующие критерии:

- однозначное определение принадлежности координаты домену (без возможности ошибок округления);
- возможность изменения размера и формы домена путем локальных взаимодействий с фиксированным числом соседних узлов и их доменов, т.е. без коммуникаций со всем множеством узлов или выделенным центральным узлом.

С учетом этих требований был выбран следующий подход: область представляется в виде регулярной декартовой n-мерной сетки ячеек, каждая ячейка имеет свою n-мерную целочисленную координату. Фрагменты данных отображаются на ячейки (с каждым фрагментом связана координата его ячейки), соседние фрагменты данных отображаются на одну и ту же или смежные ячейки. Отображение фрагментов на ячейки задается перед стартом программы и далее не меняется. Предполагая, что вычислительные узлы объединены в топологию «решетка», сетка ячеек разбивается на домены ячеек, каждый домен назначается на отдельный узел (рис. 2, [16]). Каждый домен представляет собой связную группу ячеек. Благодаря использованию целочисленных координат всегда можно безошибочно вычислить, какому домену принадлежит какая ячейка и, соответственно, какие фрагменты данных.

Резиденцией ФД является узел, домену которого в данный момент принадлежит ячейка (координата) этого ФД. Для распределения ФД или его запроса должна быть возможность определить его резиденцию с любого узла, как в алгоритме Rope. Для этого каждая ячейка хранит для каждой смежной с ней ячейки ее местоположение — номер узла, на котором в данный момент находится смежная ячейка. Эта информация позволяет, начав из любого домена (узла), за конечное число шагов найти нужную ячейку (т.е. определить резиденцию ФД), переходя между смежными доменами (узлами). При поиске учитывается тот факт, что искомая координата константна, т.к. отображение ФД на координаты не меняется в ходе исполнения программы, и все координаты глобально упорядочены по узлам, что позволяет определить направление движения.

Ввиду использования информации о смежности ячеек, для корректности определения резиденции ФД должны учитываться следующие ограничения:

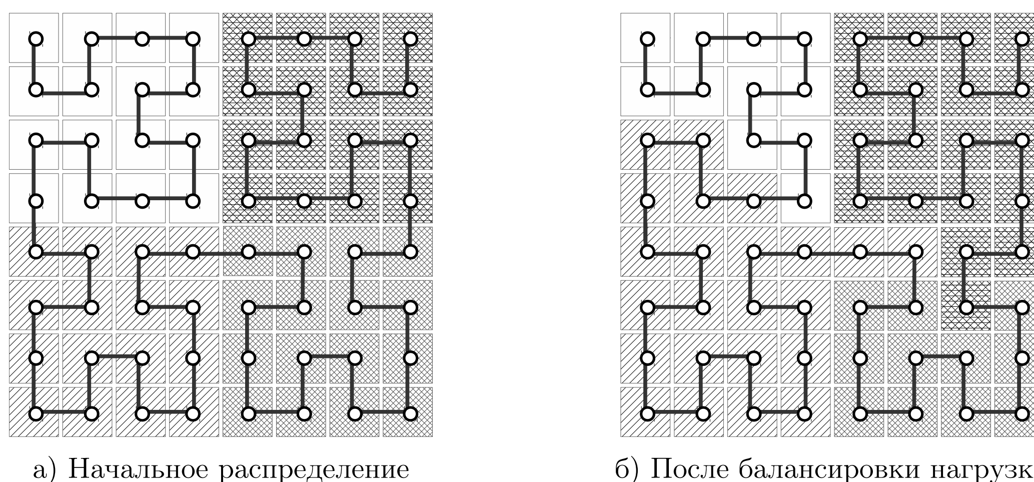
- не допускается пустых доменов, т.е. не содержащих ни одной ячейки;
- домен каждого вычислительного узла должен быть смежен (иметь смежные ячейки) с доменами соседних с ним в топологии вычислительных узлов.

Следует отметить следующие преимущества алгоритма Patch по сравнению с алгоритмом Rope:

- сохранение отношения соседства фрагментов данных по всем измерениям: соседние фрагменты данных находятся на одном и том же или соседних узлах, что приводит к сокращению объема и дистанции коммуникаций между узлами;
- время распределения или поиска фрагмента данных в худшем случае пропорционально диаметру топологии вычислительной сети (для Rope — пропорционально числу всех узлов).

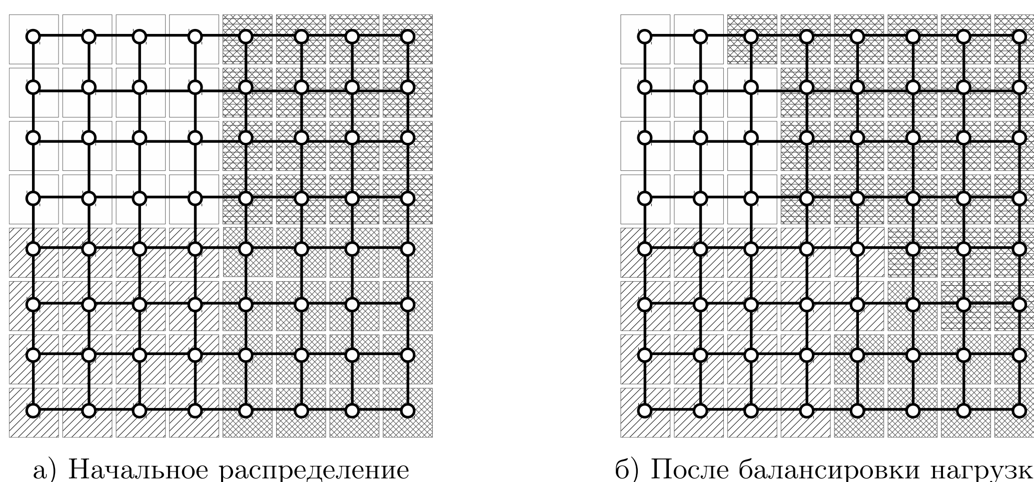
3.4. Динамическая балансировка нагрузки в алгоритме Patch

Для динамической балансировки нагрузки в алгоритме Patch, как и в алгоритме Rope, применяется диффузионный подход. Все вычислительные узлы распределяются между перекрывающимися группами, каждая группа узлов включает в себя центральный узел группы и все смежные с ним узлы в топологии; центральный узел каждой группы может быть одновременно смежным узлом в других группах, что обеспечивает взаимодействие между группами. В алгоритме Rope для каждой координаты может быть рассчитано значение приходящейся на нее нагрузки (исходя из отображенных на эту координату фрагментов), в алгоритме Patch нагрузка вычисляется для каждой ячейки. Для расчета значения нагрузки координаты/ячейки могут использоваться различные



а) Начальное распределение б) После балансировки нагрузки

Рис. 1. Распределение данных по вычислительным узлам в алгоритме Rope. Разные цвета обозначают принадлежность данных разным вычислительным узлам



а) Начальное распределение б) После балансировки нагрузки

Рис. 2. Распределение данных по вычислительным узлам в алгоритме Patch. Разные цвета обозначают принадлежность данных разным вычислительным узлам

формулы и критерии, например, суммарный текущий объем фрагментов данных на узле, отображенных на эту координату/ячейку. Значение нагрузки каждого узла равно сумме значений нагрузки находящихся на нем в данный момент координат/ячеек. В каждой группе все узлы посылают значения своей нагрузки центральному узлу группы. Узел считается перегруженным, если значение его нагрузки больше значения средней нагрузки в группе (с учетом некоторого порога дисбаланса), и недогруженным в противном случае. Если обнаруживается, что центральный узел группы перегружен, избыточная нагрузка с него должна быть передана недогруженным узлам в группе. Такая передача нагрузки происходит распределенно во всех группах, что позволяет перераспределять нагрузку по всему полю вычислительных узлов.

Передача нагрузки производится путем миграции координат между сегментами/доменами. В алгоритме Rope для миграции координат используется сдвиг границы между смежными сегментами (рис. 1б). В алгоритме Patch ячейки непосредственно мигрируют между смежными доменами (рис. 2б), с перегруженного узла на недогруженный. Фрагменты, отображенные на мигрировавшие ячейки, также

мигрируют на новый узел, что и приводит к выравниванию нагрузки между узлами. Для миграции выбирается связная группа ячеек на границе доменов, причем после миграции порядок координат не меняется, что позволяет соседним фрагментам данных оставаться на одном и том же или смежных в топологии вычислительных узлах.

При выборе группы ячеек для миграции в алгоритме Patch должны учитываться следующие критерии:

- для обеспечения точности балансировки суммарная величина нагрузки группы ячеек должна быть близка к величине нагрузки, которую нужно отдать на недогруженный узел;
- площадь границы домена должна оставаться минимальной, так как она зачастую пропорциональна объему коммуникаций с соседними узлами;
- при миграции не должна нарушаться связность ячеек внутри домена.

В целях оптимизации времени работы процедуры выбора ячеек для миграции используется жадный алгоритм. В этом алгоритме сначала выбирается одна стартовая ячейка, затем по одной добавляются смежные ячейки, пока не будет набрана нужная группа ячеек, при этом контролируется соблюдение вышеописанных критериев.

Ввиду распределенности алгоритма балансировки нагрузки миграция ячеек может происходить одновременно между многими вычислительными узлами, причем один и тот же узел может как принимать, так и отдавать ячейки. Для синхронизации приема и отправки ячеек используется механизм транзакций. Каждая транзакция представляет из себя передачу одной группы ячеек от одного узла другому. В каждый момент времени каждый узел может выполнять только одну транзакцию (прием или отправка ячеек), при этом другие транзакции на выполнение ставятся в очередь на ожидание. Для принятия решения, какую транзакцию начать исполнять, и чтобы не допустить взаимную блокировку узлов, каждой транзакции назначается случайный приоритет; каждый узел выполняет транзакции в порядке убывания их приоритета. Тем узлам, домены которых смежны с изменяемым доменом, но не участвующим непосредственно в миграции, отправляются специальные сообщения для поддержания корректности информации о смежности ячеек.

Описанный распределенный алгоритм балансировки масштабируем на большое число вычислительных узлов, т.к. каждый узел взаимодействует только с конечным числом смежных в топологии узлов. Возможный недостаток «диффузионного» подхода в виде медленного схождения к глобальному балансу из-за использования только локальных коммуникаций можно нивелировать, изменяя, автоматически или вручную, такие параметры алгоритма, как частота вызова балансировки, порог балансировки и объем передаваемой нагрузки между узлами.

4. Тесты

Для тестирования алгоритмов Rope и Patch использовалась фрагментированная реализация решения уравнения Пуассона в трехмерной области с помощью явной конечно-разностной схемы. Тестирование проводилось на кластере МВС-10П МСЦ РАН с двумя процессорами Intel Xeon E5-2690 на узле и коммуникационной сетью Infiniband FDR. Использовались компилятор C++ GCC 5.3 и библиотека MPI MPICH 3.2.

Для тестирования использовалась регулярная сетка размером 512^3, разбитая на 32^2 фрагмента данных по двум координатам; для запуска использовалось до 256 процессов. Конфигурация кластера позволяла поместить до 4-х процессов на один вычислительный

узел. Для алгоритма Rope процессы объединялись в топологию «одномерная решетка», для Patch — в топологию «двумерная решетка».

Для сравнения алгоритмов отслеживались следующие характеристики исполнения программы: средняя дистанция пересылки фрагмента данных (AvgSD, процессы) и средний суммарный объем переданных фрагментов данных (AvgSS, мегабайты).

Табл. 1 показывает результаты для случая равномерной загрузки процессов данными и вычислениями. Алгоритм Patch показывает меньшую среднюю дистанцию пересылки, чем Rope, что объясняются лучшим учетом соседства фрагментов данных: коммуникации происходят только между смежными в топологии узлами, поэтому дистанция пересылки всегда равна 1. Одинаковый объем пересылаемых данных объясняется регулярностью задачи, но следует учесть что алгоритме Rope те же данные пересылаются на большую дистанцию, чем в Patch.

Для тестирования динамической балансировки нагрузки изначальный дисбаланс был создан путем распределения данных и вычислений только на половину работающих процессов. Целью балансировки было равномерно загрузить процессы данными и вычислениями путем их динамического перераспределения. Результаты показаны в табл. 2, а график нагрузки — на рис. 3. Patch выигрывает у Rope как и по средней дистанции пересылки, так и по среднему суммарному объему отправленных данных, что опять объясняется лучшей локальностью данных. Уменьшение средней дистанции пересылки и увеличение объема отправленных данных, по сравнению со случаем равномерного распределения, связано с учетом вклада от коммуникаций на миграцию фрагментов данных при балансировке.

Рис. 3 показывает график нагрузки для 8-и процессов в ходе динамической балансировки нагрузки для алгоритмов Rope и Patch. Нагрузка процесса измерялась как текущий объем активных данных в этом процессе. В виду того, что в алгоритме Patch каждый процесс одновременно взаимодействует с большим, чем в Rope, числом соседних процессов в решетке процессов, данные перераспределяются быстрее и алгоритму Patch удается быстрее достичь сбалансированного состояния.

Таблица 1

Сетка 512^3, 32^2 фрагментов, равномерное распределение

N	1	2	4	8	16	32	64	128	256
AvgSD, Rope	0	1	1,5	1,8	2,5	3,24	4,84	6,41	9,66
AvgSD, Patch	0	1	1	1	1	1	1	1	1
AvgSS, Rope	0	20,2	20,2	20,2	15,1	12,6	8,8	6,9	4,7
AvgSS, Patch	0	20,2	20,2	20,2	15,1	12,6	8,8	6,9	4,7

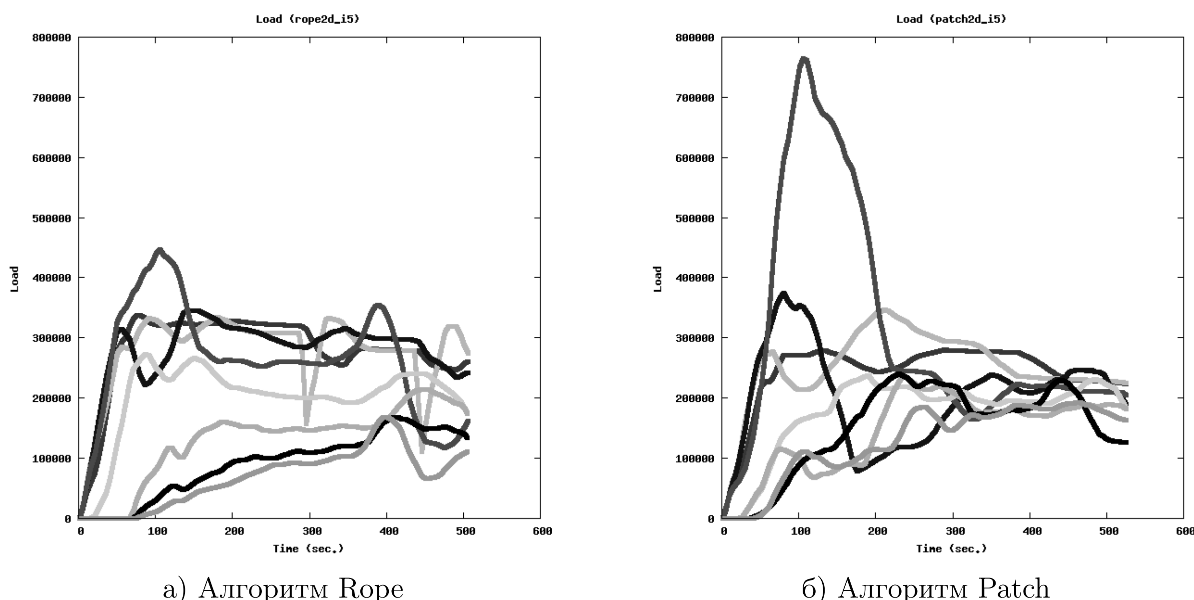
Заключение

Были разработаны и исследованы распределенные алгоритмы управления данными. Предложен распределенный алгоритм с локальными взаимодействиями Patch для динамического распределения данных и балансировки нагрузки в системе фрагментированного программирования LuNA. Выполнено тестирование алгоритма на реальной вычислительной задаче и его сравнение с алгоритмом Rope, использовавшимся до этого в системе LuNA. Было показано, что алгоритм Patch обеспечивает снижение общего

Таблица 2

Сетка 512^3, 32^2 фрагментов, неравномерное распределение

N	2	4	8	16	32	64	128	256
AvgSD, Rope	1	1,19	1,42	1,48	1,46	1,80	1,30	1,44
AvgSD, Patch	1	1,03	1,18	1,27	1,31	1,24	1,16	1,04
AvgSS, Rope	7787,6	6246,7	2821,6	1032,2	1192,6	598,0	566,9	392,0
AvgSS, Patch	9917,9	4955,9	2225,3	1226,2	714,6	444,1	193,8	91,8



а) Алгоритм Rope б) Алгоритм Patch

Рис. 3. Изменение нагрузки вычислительных узлов в алгоритмах Rope и Patch

объема и дистанции коммуникаций в ходе исполнения фрагментированной программы, тем самым повышая ее эффективность. В дальнейшем планируется доработка и оптимизация алгоритма Patch и его тестирование на вычислительных задачах различных классов.

Литература

1. Malyshkin V.E., Perepelkin V.A., Schukin G.A. Scalable Distributed Data Allocation in LuNA Fragmented Programming System // J. Supercomputing, 2017. Vol. 73, No. 2. P. 726—732. DOI: 10.1007/s11227-016-1781-0.

2. Malyshkin V.E., Perepelkin V.A., Schukin G.A. Distributed Algorithm of Data Allocation in the Fragmented Programming System LuNA // PaCT 2015, 13th International Conference on Parallel Computing Technologies, August 31 – September 4, 2015, Petrozavodsk, Russia. Springer, LNCS, 2015. Vol. 9251. P. 80–85. DOI: 10.1007/978-3-319-21909-7_8.

3. Malyshkin V.E., Perepelkin V.A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // PaCT 2011, 11th International Conference on Parallel Computing Technologies, September 19–23, 2011, Kazan, Russia. Springer, LNCS, 2011. Vol. 6873. P. 53–61. DOI: 10.1007/978-3-642-23178-0_5.

4. Malyshkin V.E., Perepelkin V.A. Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System // J. Supercomputing, 2012. Vol. 61, No. 1. P. 235–248. DOI: 10.1007/s11227-011-0649-6.

5. Malyshkin V.E., Perepelkin V.A. The PIC Implementation in LuNA System of Fragmented Programming // J. Supercomputing, 2014. Vol. 69, No. 1. P. 89–97. DOI: 10.1007/s11227-014-1216-8.
6. Kraeva M.A., Malyshkin V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers // J. Future Generation Computer Systems, 2001. Vol. 17, No. 6. P. 755–765. DOI: 10.1016/S0167-739X(00)00058-3.
7. Gonzalez-Escribano A., Torres Y., Fresno J., Llanos D.R. An Extensible System for Multilevel Automatic Data Partition and Mapping // J. IEEE Transactions on Parallel and Distributed Systems, 2014. Vol. 25, No. 5. P. 1145–1154. DOI: 10.1109/TPDS.2013.83.
8. Chamberlain B.L., Deitz S.J., Iten D., Choi S.-E. User-Defined Distributions and Layouts in Chapel: Philosophy and Framework // HotPar'10, 2nd USENIX conference on Hot topics in Parallelism. USENIX Association, Berkeley, CA, USA, 2010. P. 12–12.
9. Bikshandi G., Guo J., Hoeflinger D., Almasi G., Fraguela B.B., Garzarán M.J., Padua D., von Praun C. Programming for Parallelism and Locality with Hierarchically Tiled Arrays // PPoPP '06, 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM New York, NY, USA, 2006. P. 48–57. DOI: 10.1145/1122971.1122981.
10. Furtado P., Baumann P. Storage of Multidimensional Arrays Based on Arbitrary Tiling // 15th International Conference on Data Engineering. IEEE, 1999. P. 480–489. DOI: 10.1109/ICDE.1999.754964.
11. Begau C., Sutmann G. Adaptive Dynamic Load-balancing with Irregular Domain Decomposition for Particle Simulations // J. Computer Physics Communications, 2015. Vol. 190. P. 51–61. DOI: 10.1016/j.cpc.2015.01.009.
12. Fattebert J.-L., Richards D.F., Glosli J.N. Dynamic Load Balancing Algorithm for Molecular Dynamics Based on Voronoi Cells Domain Decompositions // J. Computer Physics Communications, 2012. Vol. 183, No. 12. P. 2608–2615. DOI: 10.1016/j.cpc.2012.07.013.
13. Deng Y., Peierls R.F., Rivera C. An Adaptive Load Balancing Method for Parallel Molecular Dynamics Simulations // J. of Computational Physics, 2000. Vol. 161, No. 1. P. 250–263. DOI: 10.1006/jcph.2000.6501.
14. Fleissner F., Eberhard P. Parallel Load-balanced Simulation for Short-range Interaction Particle Methods with Hierarchical Particle Grouping Based on Orthogonal Recursive Bisection // Int. J. for Numerical Methods in Engineering, 2008. Vol. 74, No. 4. P. 531—553. DOI: 10.1002/nme.2184.
15. Hayashi R., Horiguchi S.: Efficiency of Dynamic Load Balancing Based on Permanent Cells for Parallel Molecular Dynamics Simulation // IPDPS 2000, 14th International Parallel and Distributed Processing Symposium. IEEE, 2000. P. 85–92. DOI: 10.1109/IPDPS.2000.845968.
16. Веб-страница с демонстрацией работы алгоритмов Rope и Patch. URL: http://ssd.sscc.ru/en/algorithms (дата обращения: 01.02.2018).

Щукин Георгий Анатольевич, м.н.с., лаборатория синтеза параллельных программ, институт вычислительной математики и математической геофизики СО РАН (Новосибирск, Российская Федерация), ассистент, кафедра параллельных вычислительных технологий, Новосибирский государственный технический университет (Новосибирск, Российская Федерация)

DOI: 10.14529/cmse180205

DISTRIBUTED ALGORITHM FOR DISTRIBUTED DATA LATTICE MAPPING ON MULTIDIMENSIONAL MULTICOMPUTER IN THE LUNA FRAGMENTED PROGRAMMING SYSTEM

© 2018 G.A. Schukin

Institute of Computational Mathematics and Mathematical Geophysics SB RAS
(pr. Akademika Lavrentieva 6, Novosibirsk, 630090 Russia),
Novosibirsk State Technical University
(pr. K. Marksa 20, Novosibirsk, 630073 Russia)
E-mail: schukin@ssd.sscc.ru
Received: 22.02.2018

Distributed algorithm with local interactions Patch is presented in the paper. Patch is intended for data distribution and dynamic load balancing in the LuNA fragmented programming system. LuNA system is used for creation of parallel implementations of large-scale numerical models for distributed memory systems. Execution of a fragmented program is controlled by LuNA run-time system, which uses different data and computation distribution algorithms to enable efficient use of resources and minimize total execution time of the program. Patch algorithm, developed to be used in the LuNA system, enables distribution of multidimensional data meshes on a multidimensional lattice of computational nodes of a supercomputer. The algorithm uses mapping of data to multidimensional lattice of cells (coordinates), which in their turn are mapped to computational nodes. That mapping makes it possible to account for data dependencies and preserve data locality during dynamic load balancing. Patch algorithm was compared with another LuNA data distribution algorithm Rope, fragmented realisation of a real numerical problem was used for experiments. Experiments showed that Patch algorithm provides a general reduction in total computational volume and distances, as compared to Rope algorithm.

Keywords: distributed algorithms, data distribution, dynamic load balancing, fragmented programming technology, LuNA fragmented programming system.

FOR CITATION

Schukin G.A. Distributed Algorithm for Distributed Data Lattice Mapping on Multidimensional Multicomputer in the LuNA Fragmented Programming System. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2018. vol. 7, no. 2. pp. 63–76. (in Russian) DOI: 10.14529/cmse180205.

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

References

1. Malyshkin V.E., Perepelkin V.A., Schukin G.A. Scalable Distributed Data Allocation in LuNA Fragmented Programming System. *J. Supercomputing.* 2017. vol. 73, no. 2. pp. 726—732. DOI: 10.1007/s11227-016-1781-0.

2. Malyshkin V.E., Perepelkin V.A., Schukin G.A. Distributed Algorithm of Data Allocation in the Fragmented Programming System LuNA. PaCT 2015, 13th International Conference on Parallel Computing Technologies, August 31 – September 4, 2015, Petrozavodsk, Russia. Springer, LNCS, 2015. vol. 9251. pp. 80–85. DOI: 10.1007/978-3-319-21909-7_8.

3. Malyshkin V.E., Perepelkin V.A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. PaCT 2011, 11th International Conference on Parallel Computing Technologies, September 19–23, 2011, Kazan, Russia. Springer, LNCS, 2011. vol. 6873. pp. 53–61. DOI: 10.1007/978-3-642-23178-0_5.

4. Malyshkin V.E., Perepelkin V.A. Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System. *J. Supercomputing*. 2012. vol. 61, no. 1. pp. 235–248. DOI: 10.1007/s11227-011-0649-6.

5. Malyshkin V.E., Perepelkin V.A. The PIC Implementation in LuNA System of Fragmented Programming. *J. Supercomputing*. 2014. vol. 69, no. 1. pp. 89–97. DOI: 10.1007/s11227-014-1216-8.

6. Kraeva M.A., Malyshkin V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers. *J. Future Generation Computer Systems*. 2001. vol. 17, no. 6. pp. 755–765. DOI: 10.1016/S0167-739X(00)00058-3.

7. Gonzalez-Escribano A., Torres Y., Fresno J., Llanos D.R. An Extensible System for Multilevel Automatic Data Partition and Mapping. *J. IEEE Transactions on Parallel and Distributed Systems*. 2014. vol. 25, no. 5. pp. 1145–1154. DOI: 10.1109/TPDS.2013.83.

8. Chamberlain B.L., Deitz S.J., Iten D., Choi S.-E. User-Defined Distributions and Layouts in Chapel: Philosophy and Framework. HotPar'10, 2nd USENIX conference on Hot topics in Parallelism. USENIX Association, Berkeley, CA, USA, 2010. pp. 12–12.

9. Bikshandi G., Guo J., Hoeflinger D., Almasi G., Fraguela B.B., Garzarán M.J., Padua D., von Praun C. Programming for Parallelism and Locality with Hierarchically Tiled Arrays. PPoPP '06, 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM New York, NY, USA, 2006. pp. 48–57. DOI: 10.1145/1122971.1122981.

10. Furtado P., Baumann P. Storage of Multidimensional Arrays Based on Arbitrary Tiling. 15th International Conference on Data Engineering. IEEE, 1999. pp. 480–489. DOI: 10.1109/ICDE.1999.754964.

11. Begau C., Sutmann G. Adaptive Dynamic Load-balancing with Irregular Domain Decomposition for Particle Simulations. *J. Computer Physics Communications*. 2015. vol. 190. pp. 51–61. DOI: 10.1016/j.cpc.2015.01.009.

12. Fattebert J.-L., Richards D.F., Glosli J.N. Dynamic Load Balancing Algorithm for Molecular Dynamics Based on Voronoi Cells Domain Decompositions. *J. Computer Physics Communications*. 2012. vol. 183, no. 12. pp. 2608–2615. DOI: 10.1016/j.cpc.2012.07.013.

13. Deng Y., Peierls R.F., Rivera C. An Adaptive Load Balancing Method for Parallel Molecular Dynamics Simulations. *J. of Computational Physics*. 2000. vol. 161, no. 1. pp. 250–263. DOI: 10.1006/jcph.2000.6501.

14. Fleissner F., Eberhard P. Parallel Load-balanced Simulation for Short-range Interaction Particle Methods with Hierarchical Particle Grouping Based on Orthogonal Recursive Bisection. Int. *J. for Numerical Methods in Engineering*. 2008. vol. 74, no. 4. pp. 531–553. DOI: 10.1002/nme.2184.

15. Hayashi R., Horiguchi S.: Efficiency of Dynamic Load Balancing Based on Permanent Cells for Parallel Molecular Dynamics Simulation. IPDPS 2000, 14th International Parallel and Distributed Processing Symposium. IEEE, 2000. pp. 85–92. DOI: 10.1109/IPDPS.2000.845968.

16. Rope and Patch algorithms demonstration web–page. URL: http://ssd.sscc.ru/en/algorithms (accessed: 01.02.2018).

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 следующим отраслям и группам специальностей: 05.13.00 – информатика, вычислительная техника и управление; 25.00.00 – науки о Земле.

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, гонорары авторам не выплачиваются.**