

РАСПРЕДЕЛЕННЫЕ АЛГОРИТМЫ С ЛОКАЛЬНЫМИ ВЗАИМОДЕЙСТВИЯМИ ДЛЯ УПРАВЛЕНИЯ ДАННЫМИ В СИСТЕМЕ ФРАГМЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ LuNA

Г. А. Щукин^{1,2}

¹ *Институт Вычислительной Математики и Математической Геофизики СО РАН, 630090, Новосибирск*

² *Новосибирский Государственный Технический Университет, 630073, Новосибирск*

УДК 004.021

В работе описаны распределенные алгоритмы с локальными взаимодействиями Rope и Patch для динамического распределения данных в системе фрагментированного программирования LuNA. Задача нового алгоритма Patch — уменьшить длину и объем коммуникаций между узлами во время исполнения фрагментированной программы по сравнению с алгоритмом Rope. Коммуникации включают в себя все взаимодействия по распределению, обработке, пересылке и поиску данных. Patch учитывает зависимости между данными и поддерживает локальность зависимых данных при их распределении и миграции.

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

Введение

В настоящее время крупномасштабное численное моделирование широко используется в науке. Для достижения хорошей производительности и масштабируемости при исполнении прикладных параллельных программ численного моделирования требуются эффективные стратегии распределения ресурсов, динамическая балансировка нагрузки и другие средства. В итоге, сложность прикладного параллельного программирования приближается к сложности системного параллельного программирования. Для упрощения разработки прикладных параллельных программ численного моделирования была разработана система LuNA [1–6].

Система LuNA автоматически конструирует параллельную программу из частей (фрагментов) данных и вычислений. Каждый фрагмент вычислений (ФВ) в ходе исполнения LuNA-программы определяет независимый процесс. Каждый ФВ вычисляет выходные фрагменты данных (ФД) из значений своих входных ФД. Каждый фрагмент данных получает свое значение только один раз и является иммутабельным. Фрагментированная структура программы сохраняется в ходе исполнения, что позволяет производить миграцию фрагментов данных и вычислений между узлами мультимикомпьютера и их параллельную обработку.

Эффективность исполнения фрагментированной LuNA-программы значительно зависит от качества распределения ресурсов. Ниже приведено описание распределенных алгоритмов с локальными взаимодействиями Rope и Patch, предназначенных для управления данными в системе LuNA.

1 Обзор родственных работ

Существует много методов декомпозиции данных для их дальнейшего распределения и динамической балансировки нагрузки при создании параллельных программ для вычислительных машин с распределенной памятью. Один из таких методов — “плиточные” массивы (tiled arrays, [7–10]), в котором массивы данных разбиваются на блоки — “плитки”, которые затем распределяются по узлам мультимикомпьютера и обрабатываются параллельно. В [7] и [9] используются иерархические массивы плиток — каждая плитка в свою очередь может быть разбита на плитки меньшего размера, что позволяет создавать вложенные структуры

над плитками. В [10] используются разнородные плитки — каждая плитка может иметь разный размер по каждому измерению и не обязательно должна быть выровнена относительно других плиток. В [8] описано, как пользователь может задавать произвольное разбиение массива данных на плитки. Ограничения “плиточных” массивов — прямоугольная форма плиток, которая в определенных случаях может помешать достижению баланса нагрузки между плитками. Также, разбиение массива на плитки зачастую подразумевается статическим, без возможности динамического изменения распределенным способом в ходе исполнения программы.

Декомпозиция области данных на подобласти часто используется при распараллеливании задач молекулярной динамики и моделирования поведения частиц ([11–15]). В [11] и [13] двух или трехмерная область данных (область с частицами) разбивается на сетку подобластей путем помещения в нее промежуточных внутренних вершин; каждая подобласть содержит группу частиц и распределяется на отдельный вычислительный узел. В целях балансировки нагрузки форма подобластей (и соответственно число частиц в них) может меняться путем сдвига внутренних вершин. Балансировка может производиться распределенным способом, при этом требуется синхронизация между всеми узлами, разделяющими одну и ту же внутреннюю вершину (до 8 узлов в трехмерном случае). Существует ограничение — в целях корректной работы подобласти должны иметь выпуклую форму.

В [14] для декомпозиции области данных на подобласти используется рекурсивная ортогональная бисекция. Для балансировки нагрузки плоскости бисекции могут быть сдвинуты. Сдвиг плоскостей требует рекурсивной обработки дерева бисекции и может привести к коммуникациям между неограниченным числом разных (не обязательно соседних) вычислительных узлов.

В [12] на область данных накладывается сетка ячеек Вороного, каждая подобласть состоит из связанного множества таких ячеек. Для балансировки загрузки ячейки могут перемещаться между соседними подобластями. В [15] прямоугольные ячейки используются схожим образом. Эти алгоритмы нацелены на класс задач молекулярной динамики и при расчете балансировки загрузки используют дополнительную информацию (такую как характеристики частиц), которая может быть недоступна в задачах другого класса.

Можно заключить, что желаемые свойства алгоритма распределения данных и балансировки нагрузки — распределенность, использование преимущественно локальных коммуникаций и применимость к широкому кругу задач.

2 Распределенные алгоритмы управления данными

Разработанный ранее алгоритм Rope [1, 2] был предназначен для обработки произвольных (с учетом выбора соответствующей функции отображения данных на одномерные координаты) структур данных на вычислительных машинах с распределенной памятью. В статье представлен новый алгоритм Patch, улучшающий обработку регулярных сеток на вычислительных машинах с топологией “многомерная решетка”. В следующих разделах представлено описание обоих алгоритмов и их сравнение.

2.1 Вспомогательные определения

В численных алгоритмах зачастую в качестве данных используются многомерные декартовы сетки (например, сетки в методе частиц-в-ячейках или в решении дифференциальных уравнений итерационными методами). Для декомпозиции сетка разбивается по нескольким измерениям на фрагменты данных (параллелепипеды), формируя сетку фрагментов данных. Фрагменты данных, находящиеся рядом в сетке по какой-либо декартовой координате, будем называть смежными. Два фрагмента данных называются соседними, если или значение одного из них вычисляется с использованием значения другого, или значения обоих используются для вычисления значения некоторого третьего фрагмента данных (т.е. такие фрагменты данных связаны информационными зависимостями). Во многих численных методах смежные фрагменты данных являются соседними и наоборот.

2.2 Алгоритм Rope

Алгоритм Rope использует отображение сетки фрагментов данных на одномерный числовой диапазон, например, с помощью пространственной кривой Гильберта (Рис. 1, [16]). Таким образом, каждый фрагмент данных получает свою (целочисленную) координату в диапазоне. Отображение на диапазон делается таким образом, чтобы соседние фрагменты данных отображались или на одну и ту же координату, или на близкие

координаты в диапазоне (использование кривой Гильберта обеспечивает это условие в некоторой степени). Отображение фиксируется перед началом исполнения фрагментированной программы и в ходе исполнения не меняется.

Для распределения фрагментов данных по узлам мультимпьютера диапазон разбивается на сегменты по числу вычислительных узлов, каждый узел получает свой сегмент. Предполагается, что узлы объединены в линейную топологию, тогда отображение сегментов на узлы выполняется один-в-один. Если соседние фрагменты данных находятся на разных узлах (их координаты находятся в разных сегментах), это приводит к коммуникациям между этими узлами.

Резиденцией фрагмента данных называется такой вычислительный узел, в сегмент которого попадает координата этого ФД. Каждый ФД распределяется на свой узел-резиденцию и хранится на нем, а также может быть запрошен (скопирован) с него. Поиск резиденции каждого ФД заключается в поиске сегмента, содержащего координату искомого фрагмента данных, и может быть произведен с использованием только локальных взаимодействий между вычислительными узлами.

Подробнее про алгоритм Rore см. в [1, 2].

2.3 Алгоритм Patch

Из-за отображения фрагментов данных на одномерный диапазон алгоритм Rore не позволяет полностью сохранять соседство фрагментов данных по всем измерениям в случае многомерных сеток (некоторые соседние фрагменты данных будут распределены на далеко отстоящие друг от друга в топологии узлы). Для устранения этого недостатка был разработан новый алгоритм Patch, использующий отображение k -мерной сетки фрагментов данных на n -мерную область координат. Область представляется в виде регулярной декартовой сетки ячеек, каждая ячейка имеет свою n -мерную координату. Отображение фрагментов данных на n -мерную сетку ячеек позволяет соседним фрагментам данных быть отображенными на одну и ту же или смежные ячейки, лучше сохраняя отношение соседства, чем в Rore. Как и в Rore, отображение фиксируется на все время исполнения фрагментированной программы.

Предполагая, что вычислительные узлы объединены в топологию “решетка”, сетка ячеек разбивается на домены ячеек, по одному домену на узел (Рис. 2, [16]). Как и в случае с Rore, отображение доменов на узлы выполняется один-в-один. В общем случае, домены ячеек могут быть любой формы (не обязательно параллелепипеды), но должны учитываться следующие ограничения:

- Не допускается пустых доменов (не содержащих ни одной ячейки)
- Площадь границы доменов должна быть минимальна, чтобы уменьшить объем коммуникаций между доменами (и соответственно между соседними узлами)
- Домен каждого вычислительного узла должен быть смежен (иметь смежные ячейки) с доменами соседних с ним в топологии вычислительных узлов

Эти ограничения служат для обеспечения корректности распределения и поиска фрагментов данных, о которых будет рассказано дальше.

В системе LuNA каждый фрагмент данных и вычислений распределяется по вычислительным узлам мультимпьютера динамически, т.е. по ходу исполнения программы. Т.к. распределение фрагментов данных по узлам не статическое, а может меняться со временем (например, из-за динамической балансировки нагрузки), требуется способ динамического определения текущего местоположения фрагментов данных. Узел, которому в данный момент принадлежит ячейка, на которую отображается фрагмент данных, будем называть резиденцией этого фрагмента данных. Если известна резиденция фрагмента данных (узел), то этот фрагмент данных должен храниться (быть создан) на этом узле, 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.

Для возможности определения резиденции фрагмента данных каждая ячейка хранит информацию о смежности — текущее местоположение (номер узла) всех смежных с ней ячеек в сетке ячеек. Используя эту информацию, возможно определить резиденцию любого фрагмента данных, начиная с любого узла, за конечное число шагов. Сначала вычисляется координата ячейки, на которую отображен искомый фрагмент данных. Ввиду неизменности отображения фрагментов данных на ячейки, эта координата — константа. Затем проверяется, содержит ли текущий узел ячейку с нужной координатой. Если ячейка найдена, поиск

резиденции завершен. Иначе на узле находится ячейка, ближайшая к искомой, и поиск продолжается с узла, смежного с этой ячейкой в направлении искомой ячейки.

Преимущества алгоритма Patch:

- Сохранение отношения соседства фрагментов данных по всем измерениям (соседние фрагменты данных находятся или на одном и том же, или соседних узлах, что приводит к сокращению объема и длины коммуникаций)
- Хорошо подходит для вычислительных систем с реальной топологией “решетка”
- Худшее время распределения или поиска фрагмента данных пропорционально диаметру вычислительной сети, в то время как у Rore оно пропорционально числу вычислительных узлов

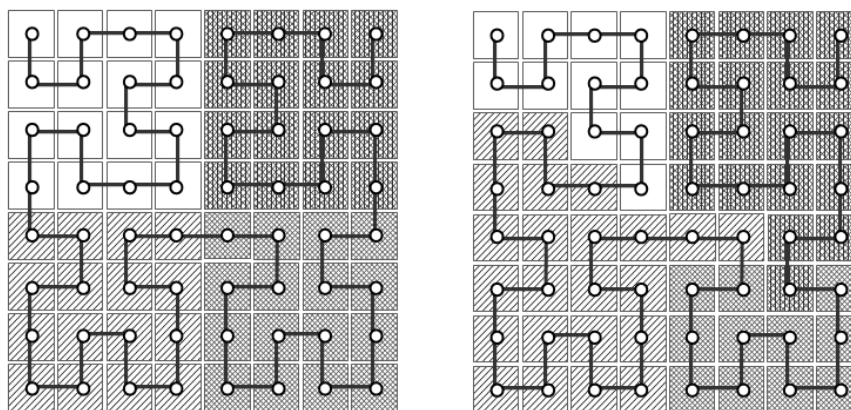


Рис. 1: Распределение данных по вычислительным узлам в алгоритме Rore с помощью кривой Гильберта, изначальное (слева) и после возможной балансировки нагрузки (справа). Разные цвета обозначают разные вычислительные узлы.

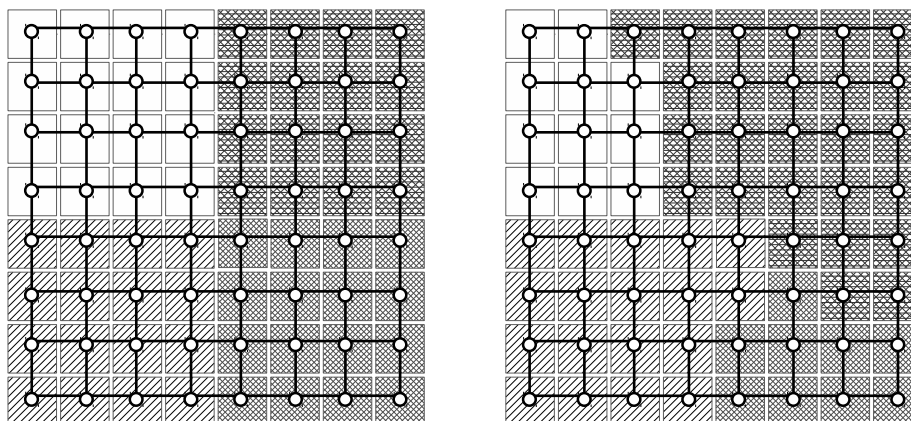


Рис. 2: Распределение данных (ячеек) по вычислительным узлам в алгоритме Patch, изначальное (слева) и после возможной балансировки нагрузки (справа). Разные цвета обозначают разные вычислительные узлы.

2.4 Динамическая балансировка нагрузки в алгоритме Patch

Для динамической балансировки нагрузки в алгоритмах Rore и Patch используется диффузионный подход. Все вычислительные узлы мультикомпьютера разбиваются на перекрывающиеся группы, каждая группа состоит из центрального узла группы и всех смежных с ним узлов в топологии. Центральный узел каждой группы может одновременно быть смежным узлов в других группах, что обеспечивает взаимодействие

между группами. Для каждого узла рассчитывается значение его нагрузки. В алгоритме Rore нагрузка рассчитывается для каждой координаты из сегмента на узле, в Patch — для каждой ячейки из домена на узле. Значение нагрузки каждого узла равно сумме значений нагрузки его координат/ячеек. Для расчета значения нагрузки координаты/ячейки могут использоваться различные формулы и критерии, например, суммарный текущий объем фрагментов данных на узле, отображенных на эту координату/ячейку. Каждому узлу известны значения его нагрузки и нагрузки его соседей в группе. Узел считается перегруженным, если значение его нагрузки больше значения средней нагрузки в группе (с учетом некоторого порога дисбаланса), и недогруженным, если меньше. Если центральный узел группы перегружен, он должен передать свою избыточную нагрузку недогруженным узлам в группе.

В алгоритме Rore для балансировки нагрузки использовался сдвиг границы между смежными сегментами. В алгоритме Patch используется миграция ячеек между смежными доменами, от перегруженного узла на недогруженный. Ячейки для миграции выбираются таким образом, чтобы, во-первых, их суммарное значение нагрузки было примерно равно значению нагрузки, которую нужно отдать на недогруженный узел, и, во-вторых, число граничных клеток между доменами оставалось минимальным (т.к. это число прямо пропорционально объему возможных коммуникаций между доменами и, соответственно, узлами). Для этого используется жадный алгоритм, выбирающий группу смежных ячеек на границе доменов, соответствующую вышеуказанным критериям.

Для синхронизации одновременной пересылки ячеек между многими узлами используется механизм транзакций. Каждая транзакция включает в себя пересылку одной группы ячеек от одного узла другому. В каждый момент времени каждый узел может выполнять только одну транзакцию (прием или отправка ячеек). Для определения порядка исполнения транзакций и предотвращения дедлоков каждой транзакции назначается случайный приоритет; транзакция с максимальным приоритетом на узле выполняется в первую очередь. Так как миграция ячеек может потребовать изменение информации о смежности в ячейках на многих узлах (не только участвующих в приеме и отправке ячеек), таким узлам отправляются специальные сообщения для поддержания корректности этой информации.

Передача единичных групп ячеек вместо, например, сдвига всей границы домена по какой-либо координате обеспечивает более высокую точность балансировки нагрузки. Т.к. каждая миграция ячеек требует синхронизации (блокирования) только двух узлов, множество миграций может проводиться параллельно, ускоряя процесс балансировки. Для динамической балансировки нагрузки требуются только локальные взаимодействия между соседними вычислительными узлами, что делает алгоритм балансировки масштабируемым на большое число узлов.

3 Тесты

Для сравнения алгоритмов использовалась LuNA-реализация решения дифференциального уравнения Пуассона на трехмерной сетке итерационным методом Якоби. Тесты проводились на кластере МВС-10П МСКЦ РАН с двумя процессорами Intel Xeon E5-2690 на вычислительный узел и транспортной сетью Infiniband FDR. Для компиляции программы использовался компилятор C++ GCC 5.3 и MPI библиотека MPICH 3.2.

3.1 Результаты тестирования

Для тестирования регулярная сетка размером 512^3 была разбита на 32^2 трехмерных фрагмента по двум координатам. Измерения проводились до 256 процессов (N). До 4-х процессов могло помещаться на одном вычислительном узле кластера. Для алгоритма Rore процессы были объединены в линейную топологию, для Patch использовалась топология “двумерная решетка”.

Измерялись следующие характеристики исполнения фрагментированной программы: общее время выполнения (ET, секунды), средняя дистанция пересылки фрагмента данных процессом (AvgSD, процессы), средний суммарный объем фрагментов данных, отправленных процессом (AvgSD, мегабайты) и среднеквадратичное отклонение среднего времени полезных вычислений на процессе (AvgCTD, секунды).

Таблица 1 показывает результаты для случая равномерной загрузки процессов данными и вычислениями. Алгоритм Patch демонстрирует практически одинаковое с Rore общее время работы, и, ожидаемо, показывает намного меньшую среднюю дистанцию пересылки и суммарный объем отправленных фрагментов данных. Превосходящие результаты Patch объясняются лучшим учетом соседства фрагментов данных чем в Rore.

Таблица 1: Сетка 512^3 , 32^2 фрагментов, равномерное распределение.

N	1	2	4	8	16	32	64	128	256
ET, Rope	773.8	407.4	217.8	111.9	57.9	31.7	19.7	14.7	17.4
ET, Patch	769.0	409.4	221.3	113.4	68.8	31.7	19.4	12.3	13.6
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

Таблица 2: Сетка 512^3 , 32^2 фрагментов, неравномерное распределение.

N	2	4	8	16	32	64	128	256
AvgCTD, Rope	53.4	15.99	55.68	47.21	25.18	13.51	7.44	3.93
AvgCTD, Patch	58.11	23.7	56.49	33.33	21.39	12.56	7.01	3.71
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

Для тестирования динамической балансировки нагрузки был создан дизбаланс путем начального распределения данных и вычислений только на половину работающих процессов. Целью балансировки было постараться динамически перераспределить данные и вычисления на все процессы для их равномерной загрузки. Результаты показаны в таблице 2. Алгоритм Patch в общем демонстрирует меньшее отклонение времени полезных вычислений, чем Rope, что означает, что ему удалось равномернее загрузить процессы данными и вычислениями. Опять же из-за лучшего учета отношения соседства Patch показывает меньшие среднюю дистанцию пересылки и суммарный объем отправленных фрагментов данных, чем Rope. Значительное уменьшение средней дистанции пересылки и увеличение объема отправленных данных, по сравнению с результатами в таблице 1, связано с учетом вклада от миграции фрагментов данных при балансировке, которая происходила между соседними узлами.

Заключение

Предложен распределенный алгоритм с локальными взаимодействиями Patch для динамического распределения данных и балансировки нагрузки в системе фрагментированного программирования LuNA. Проведено сравнительное тестирование алгоритма на реальной вычислительной задаче. Показаны преимущества разработанного алгоритма. В направления дальнейшей работы входит оптимизация и тестирование алгоритма на вычислительных задачах разных классов.

Список литературы

- [1] Malyshkin V.E., Perepelkin V.A., Schukin G.A. Scalable distributed data allocation in LuNA fragmented programming system // The Journal of Supercomputing. 2017. V. 73, iss. 2. P. 726–732.
- [2] Malyshkin V.E., Perepelkin V.A., Schukin G.A. Distributed Algorithm of Data Allocation in the Fragmented Programming System LuNA // PaCT 2015, LNCS. 2015. V. 9251. P. 80–85.
- [3] Malyshkin V.E., Perepelkin V.A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // PaCT 2011, LNCS. 2011. V. 6873. P. 53–61.
- [4] Malyshkin V.E., Perepelkin V.A. Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System // The Journal of Supercomputing. 2012. V. 61, iss. 1. P. 235–248.

- [5] Malyshkin V.E., Perepelkin V.A. The PIC Implementation in LuNA System of Fragmented Programming // The Journal of Supercomputing. 2014. V. 69, iss. 1. P. 89–97.
- [6] Kraeva M.A., Malyshkin V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers // Journal Future Generation Computer Systems. 2001. V. 17, iss. 6. P. 755–765.
- [7] Gonzalez-Escribano A., Torres Y., Fresno J., Llanos D.R. An Extensible System for Multilevel Automatic Data Partition and Mapping // Journal IEEE Transactions on Parallel and Distributed Systems. 2014. V. 25, iss. 5. P. 1145–1154.
- [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. 2010. P. 12.
- [9] Bikshandi G., Guo J., Hoefflinger D., Almasi G., Fraguera 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. 2006. P. 48–57.
- [10] Furtado P., Baumann P. Storage of Multidimensional Arrays Based on Arbitrary Tiling // 15th International Conference on Data Engineering. 1999. P. 480–489.
- [11] Begau C., Sutmann G. Adaptive dynamic load-balancing with irregular domain decomposition for particle simulations // Journal Computer Physics Communications. 2015. V. 190. P. 51–61.
- [12] Fattbert J.-L., Richards D.F., Glosli J.N. Dynamic load balancing algorithm for molecular dynamics based on Voronoi cells domain decompositions // Journal Computer Physics Communications. 2012. V. 183, iss. 12. P. 2608–2615.
- [13] Deng Y., Peierls R.F., Rivera C. An Adaptive Load Balancing Method for Parallel Molecular Dynamics Simulations // Journal of Computational Physics. 2000. V. 161, iss. 1. P. 250–263.
- [14] Fleissner F., Eberhard P. Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection // International Journal for Numerical Methods in Engineering. 2008. V. 74, iss. 4. P. 531–553.
- [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. 2000. P. 85–92.
- [16] Веб-страница с демонстрацией работы алгоритмов Rope и Patch: <http://ssd.sccc.ru/en/algorithms>.

*Георгий Анатольевич Щукин — мл. науч. сотр. Института
вычислительной математики и математической геофизики СО РАН;
e-mail: schukin@ssd.sccc.ru.*

Дата поступления — 31 мая 2017 г.