DISTRIBUTED ALGORITHM OF MULTIDIMENSIONAL DATA LATTICE DISTRIBUTION ON MULTIDIMENSIONAL MULTICOMPUTER IN THE LUNA FRAGMENTED PROGRAMMING SYSTEM

V. E. Malyshkin*, G. A. Schukin

Institute of Computational Mathematics and Mathematical Geophysics SB RAS,
630090, Novosibirsk, Russia
*Novosibirsk State University,
630090, Novosibirsk, Russia
Novosibirsk State Technical University
630073, Novosibirsk, Russia

The paper presents a scalable distributed algorithm for dynamic data allocation in LuNA fragmented programming system. The proposed algorithm is optimized for data lattice distribution on a grid network topology. The algorithm takes into account data structure of the application numerical model executed and enables dynamic load balancing.

Implementation of numerical models on multicomputers with large number of computing nodes is a challenging problem in the domain of high-performance parallel computing. Effective resources allocation and balancing strategies are necessary to achieve good efficiency and scalability of parallel programs. LuNA is a fragmented programming system which is being developed in Supercomputer Software Department of Institute of Computational Mathematics and Mathematical Geophysics SB RAS. LuNA's main objective is automation of construction of parallel programs, which implement large-scale numerical models for large-scale multicomputers.

In LuNA system an application algorithm is represented as two sets: a set of immutable data pieces (data fragments, DF) and a set of side-effect free computational processes (computational fragments, CF). All these fragments can be distributed over computational nodes of a multicomputer. Each DF is produced by one CF and can be used as an input by other CFs; each CF is executed only once and requires values of all its input DFs to be gathered on a single node for its execution. Execution of a program is managed by LuNA's runtime system. The runtime system performs distribution and migration of DFs and CFs over nodes of a multicomputer and deliverance of input DFs to their corresponding CFs to provide execution of all CFs in the program.

Efficiency of LuNA program execution (in terms of running time, memory consumption, communications amount, etc.) heavily depends on quality of CFs and DFs distribution. LuNA's runtime system may use different algorithms for data distribution and load balancing. First, algorithm Rope was developed, which supported distribution of a general data structures on a one-dimensional grid network. One of requirements of a quality data distribution is data locality, i.e. dependent data fragments should be allocated on the same or neighbor nodes to minimize communications. To improve data locality for multidimensional data lattices on multidimensional grid networks, new Patch algorithm was developed.

As in Rope algorithm, Patch uses mapping of data fragments to intermediate coordinates. The mapping is a constant function during execution of a program, thus it can be computed on any node without communications. Each coordinate is represented as a cell, and all coordinates are represented as a mesh of cells. That intermediate mapping is to account data dependencies: dependent („neighbor“) data fragments should be mapped to the same or neighbor cells; that way they will be distributed on

© V. E. Malyshkin, G. A. Schukin, 2018

the same or neighbor nodes. The cell mesh is split on cell domains, each node receives its own cell domain. Each data fragment is allocated on the node containing cell to which its is mapped. Each cell stores information about current location (node) of all cells adjacent to it in the cell mesh. With this information it is possible to find a location of any cell from any node, using only local communications between nodes. This search is used for initial data fragment allocation and also to serve requests for data fragments from computational fragments.

For the purpose of dynamic load balancing mapping of cells to nodes can be changed. Migration of workload is accomplished by migration of cells between cell domains. Migration of cells causes migration of actual data fragments, mapped to these cells, and subsequent change of workload. To organize exchange of workload, diffusion scheme is used. All nodes are organized into overlapping groups, each group consisting of main node and its neighbors. Each node knows its own workload and workload of its neighbors in a group. Formulas for workload evaluation can be different, for example value of workload can be proportional to a volume of memory occupied by data on the node. Workloads of neighboring nodes are compared periodically; if the main node is overloaded (its workload exceeds average workload of the group by some threshold), it migrates cells to underloaded nodes. Only cells on domains' borders are eligible for migration, i.e. ordering of cells' coordinates isn't changed after migration. That way data locality is preserved during dynamic load balancing.

Usage of only local communications for data allocation and load balancing makes proposed algorithm scalable to a large number of computational nodes, because no global synchronization (that can impede scalability) is used.

Testing of Patch algorithm has shown improvements over Rope algorithm, both in average length of communications between nodes in a topology and in average volume of transmitted data. Further development, testing and optimization of the algorithm are planned.

Key words: scalable distributed system algorithm, dynamic data allocation, distributed algorithms with local interactions, fragmented programming technology, fragmented programming system LuNA.

References

1. Malyshkin V. E., Perepelkin V. A., Schukin G. A. Scalable distributed data allocation in LuNA fragmented programming system // J. Supercomputing. 2017. V. 73. N 2. P. 726—732. Springer US.

2. Malyshkin V. E., Perepelkin V. A., Schukin G. A. Distributed Algorithm of Data Allocation in the Fragmented Programming System LuNA // PaCT 2015. LNCS, V. 9251. P. 80–85. Springer, Heidelberg.

3. Malyshkin V. E., Perepelkin V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // PaCT 2011, LNCS, V. 6873, P. 53–61. Springer, Heidelberg.

4. Malyshkin V. E., Perepelkin V. A. Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System // J. Supercomputing. 2012. V. 61, N 1, P. 235–248.

5. Malyshkin V. E., Perepelkin V. A. The PIC Implementation in LuNA System of Fragmented Programming // J. Supercomputing. 2014. V. 69. N 1. P. 89–97.

6. Kraeva M. A., Malyshkin V. E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers // J. Future Generation Computer Systems. 2001. V. 17. N 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 // J. IEEE Transactions on Parallel and Distributed Systems. 2014. V. 25. N 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. 12–12. USENIX Association, Berkeley, CA, USA.

9. Bikshandi G., Guo J., Hoeflinger D., Almasi G., Fraguela B. B., Garzaran 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. ACM New York, NY, USA.

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

11. Begau C., Sutmann G. Adaptive dynamic load-balancing with irregular domain decomposition for particle simulations // J. Computer Physics Communications. 2015. V. 190. P. 51-–61. Elsevier B. V.

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. V. 183. N 12. P. 2608–2615. Elsevier B. V.

13. Deng Y., Peierls R. F., Rivera C. An Adaptive Load Balancing Method for Parallel Molecular Dynamics Simulations // J. of Computational Physics. 2000. V. 161. N 1. P. 250–263. Elsevier B. V.

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. V. 74. N 4. P. 531—553. John Wiley & Sons, Ltd.

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. IEEE.

16. Rope and Patch algorithms demonstration page. [Electron. res.]: `http://ssd.sscc.ru/en/algorithms`.

РАСПРЕДЕЛЕННЫЙ АЛГОРИТМ РАСПРЕДЕЛЕНИЯ МНОГОМЕРНЫХ СЕТОК ДАННЫХ НА МНОГОМЕРНОМ МУЛЬТИКОМПЬЮТЕРЕ В СИСТЕМЕ ФРАГМЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ LUNA

В. Э. Малышкин*, Г. А. Щукин

Институт вычислительной математики и математической геофизики СО РАН,
630090, Новосибирск, Россия
*Новосибирский Государственный Университет,
630090, Новосибирск, Россия
Новосибирский Государственный Технический Университет,
630073, Новосибирск, Россия

УДК 004.021

В статье представлен распределенный алгоритм с локальными взаимодействиями Patch, использующийся для распределения данных и динамической балансировки нагрузки в системе фрагментированного программирования LuNA. Алгоритм разработан для случая распределения многомерной сетки данных на многомерной решетке вычислительных узлов, учитывает зависимости между данными и сохраняет локальность данных при динамической балансировке нагрузки. Произведено тестирование алгоритма, показывающее его преимущество над другими алгоритмами распределения данных в системе LuNA.

Ключевые слова: распределенный алгоритм, распределение данных, динамическая балансировка нагрузки, технология фрагментированного программирования, система фрагментированного программирования LuNA.

Введение. В настоящее время наблюдается широкое использование крупномасштабных численных моделей, особенно в науке. Для достижения хорошей производительности и масштабируемости параллельных программ численного моделирования на распределенных вычислительных системах требуются эффективные стратегии распределения ресурсов (данных) и динамической балансировки нагрузки. Ввиду затрат на реализацию таких стратегий для каждого приложения сложность прикладного параллельного программирования становится сопоставима со сложностью системного параллельного программирования. Для упрощения разработки параллельных программ численного моделирования разрабатывается система фрагментированного программирования LuNA ([1–6]).

Система LuNA автоматически конструирует параллельную программу 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. Каждому фрагменту данных присваивается значение только один раз, и каждый фрагмент вычислений исполняется только один раз. Фрагментированная структура программы сохраняется во время ее исполнения, что позволяет независимым фрагментам вычислений исполняться

© В. Э. Малышкин, Г. А. Щукин, 2018

параллельно, а всем фрагментам (данных и вычислений) — мигрировать между узлами мультикомпьютера для обеспечения динамической балансировки нагрузки.

Эффективность исполнения фрагментированной программы значительно зависит от качества распределения ресурсов, т. е. данных и вычислений. 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.

1. Обзор. Существует много алгоритмов декомпозиции данных и динамической балансировки нагрузки. Один из способов декомпозиции данных — „плиточные“ массивы ([7–10]). Массив данных сначала разбивается на прямоугольные блоки („плитки“), затем эти блоки распределяются по узлам мультикомпьютера. Разбиение на плитки может быть иерархическим, как показано, например, в работах [7] и [9]. Размер и расположение плиток относительно друг друга могут быть как регулярными, так и произвольными (см. [10]). Разбиение на плитки может производиться автоматически или контролироваться пользователем ([8]). Ограничением „плиточного“ подхода является прямоугольная форма 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 т. ч. распределенным способом.

Различные методы декомпозиции расчетной области широко используются в вычислительных методах, например, в приложениях молекулярной динамики, симуляциях частиц и т. п. ([11–15]). В статьях [11] и [13] расчетная область — поле частиц — разбивается на домены путем помещения специальных внутренних вершин внутрь области для создания сетки доменов; каждый домен изначально имеет прямоугольную форму и распределяется на отдельный узел мультикомпьютера. Для осуществления динамической балансировки нагрузки внутренние вершины могут изменять свое положение, в свою очередь изменяя форму домена, количество частиц в домене и нагрузку соответствующего узла мультикомпьютера. Балансировка может проходить распределенным способом, где каждый узел взаимодействует только со своими соседними узлами и их доменами. Тем не менее, для смещения одной внутренней вершины требуются коммуникации между разделяющими ее 8-ю соседними узлами (в трехмерном случае). Дополнительным ограничением является требование выпуклой формы доменов.

В статье [14] используется алгоритм рекурсивной ортогональной бисекции. Расчетная область рекурсивно делится плоскостями бисекции на домены, которые затем распределяются на вычислительные узлы. Для балансировки нагрузки плоскости бисекции смещаются для изменения размеров доменов. Недостатком является то, что сдвиг плоскостей бисекции может потребовать обхода всего дерева бисекции для изменения информации о доменах и привести к коммуникациям между неограниченным число узлов.

В статье [12] домены состоят из ячеек Воронова. Ячейки на границах доменов могут мигрировать между доменами для изменения их формы и достижения баланса нагрузки. В [15] аналогичным способом используются обычные прямоугольные ячейки. Для того, чтобы каждый узел взаимодействовал только с определенной группой соседних узлов, некоторые клетки обозначаются как постоянные и не могут покидать свой домен.

Большинство алгоритмов распределения данных и динамической балансировки нагрузки, приведенных в вышеописанных работах, нацелены на работу в определенных зада-

чах (например, методы молекулярной динамики и т. д.) и могут потребовать существенной переработки для их адаптации к более широкому кругу задач.

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

2. Распределенные алгоритмы распределения данных. Изначально для распределения данных и динамической балансировки нагрузки в системе LuNA был разработан алгоритм Rope ([1, 2]). Алгоритм Rope в общем случае поддерживал распределение произвольных структур данных на вычислительной сети с произвольной топологией. Тем не менее, он имел некоторые недостатки при распределении многомерных структур данных, для исправления которых был разработан алгоритм Patch. В дальнейших разделах представлены описание алгоритма Patch и его сравнение с алгоритмом Rope.

2.1. *Вспомогательные определения.* Численные методы зачастую оперируют с многомерными сетками данных (метод частиц-в-ячейках, итерационные методы решения дифференциальных уравнений с помощью конечно-разностной схемы и т. д.). При декомпозиции сетка разбивается на блоки (фрагменты), преобразуясь в сетку фрагментов данных. Фрагменты данных, смежные в сетке, будем называть смежными. Два фрагмента данных будем называть соседними, если они участвуют в одном и том же фрагменте вычислений: значение одного ФД вычисляется на основе значения другого ФД, либо значения обоих ФД используются для вычисления значения третьего ФД.

В системе LuNA каждый фрагмент данных и вычислений распределяется на вычислительный узел мультикомпьютера 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 для возможности их запроса фрагментами вычислений. Узел, на котором должен (например, в результате принятия решения по распределению) храниться фрагмент данных, будем называть резиденцией этого фрагмента данных. Если вычислена резиденция фрагмента данных, то этот фрагмент данных должен быть создан на этом узле (перемещен на него) 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.

2.2. *Алгоритм Rope.* Для целей распределения фрагментов данных алгоритм Rope использует отображение сетки фрагментов данных на одномерный числовой диапазон, например, с помощью пространственной кривой Гильберта (рис. 1), [16]). 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 позволяет обеспечить выполнение этого условия в той или иной мере). Отображение фиксируется перед началом исполнения фрагментированной программы и в ходе исполнения не меняется.

Для распределения фрагментов данных по узлам мультикомпьютера диапазон разбивается на сегменты по числу вычислительных узлов, каждый узел получает свой сегмент. Предполагается, что узлы объединены в линейную топологию, что позволяет провести отображение сегментов на узлы один-в-один. Резиденцией ФД становится 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 произведен с помощью прохода по линейке узлов, с использованием только локальных взаимодействий между вычислительными узлами.

Отображение соседних ФД на одну и ту же или близкие координаты позволяет обеспечить их распределение на один и тот же или близкие в топологии узлы, таким образом добиваясь локальности коммуникаций. Подробнее про алгоритм Rope см. в [1, 2].

2.3. *Алгоритм Patch.* Из-за отображения фрагментов данных на одномерный диапазон алгоритм Rope не позволяет полностью сохранять соседство фрагментов данных по всем измерениям для многомерных сеток фрагментов данных: некоторые соседние фрагменты данных будут отображены на далеко отстоящие друг от друга координаты и, следовательно, распределены на далеко отстоящие друг от друга в линейной топологии узлы. Для устранения этого недостатка требуется отображение на многомерную область координат, которая позволит учитывать соседство фрагментов данных по многим координатам одновременно. В свою очередь, многомерная область координат должна отображаться на многомерную решетку вычислительных узлов, размерностью не большей чем сама область. Такое отображение было реализовано в новом алгоритме Patch.

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

— невозможность ошибок округления при вычислении принадлежности координаты домену;

— возможность изменения размера и формы домена путем локальных взаимодействий с фиксированным числом соседних узлов и их доменов, т. е. без коммуникаций со всеми узлами, выделенным центральным узлом или большой группой узлов.

С учетом этих требований был выбран следующий подход: область представляется в виде регулярной декартовой сетки ячеек, каждая ячейка имеет свою n-мерную целочисленную координату. Фрагменты данных отображаются на ячейки. Соседние фрагменты данных отображаются на одну и ту же или смежные ячейки. Предполагая, что вычислительные узлы объединены в топологию „решетка“, сетка ячеек разбивается на домены ячеек, по одному домену на узел (рис. 2), [16]. Каждый домен — связная группа ячеек. Изначально домены имеют форму параллелепипедов, но в дальнейшем могут принимать любую форму.

Резиденцией ФД является узел, которому в данный момент принадлежит ячейка этого ФД. Для возможности поиска резиденции фрагмента данных с каждого узла каждая ячейка хранит текущее местоположение всех смежных с ней ячеек — номер узла, на котором в данный момент находятся смежные ячейки. Используя эту информацию о смежности, возможно определить резиденцию любого фрагмента данных, начиная с любого узла, за конечное число шагов. Для определения резиденции 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, учитывая, что координаты упорядочены, с помощью информации о смежности можно определить соседний узел в направлении искомой ячейки и продолжить поиск с этого соседнего узла.

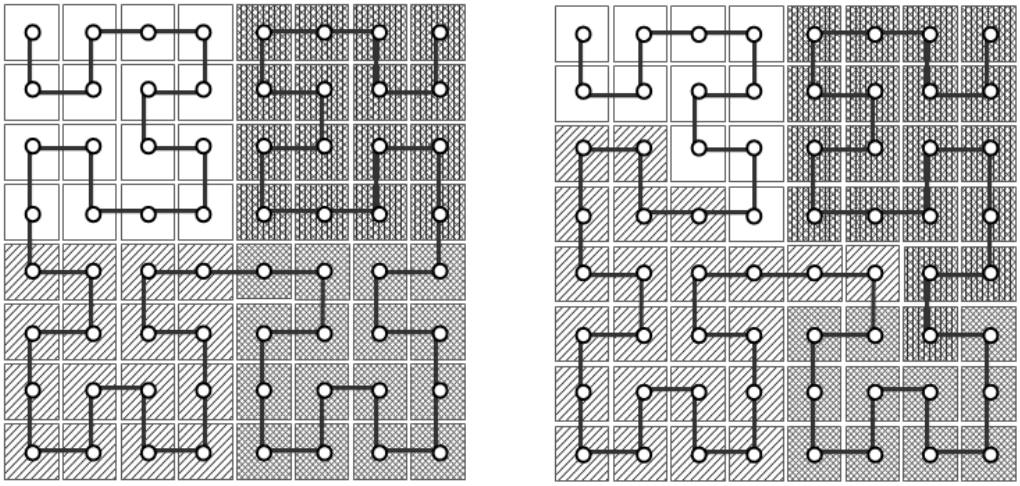


Рис. 1. Распределение данных по вычислительным узлам в алгоритме Rope с помощью кривой Гильберта, изначальное (слева) и после возможной балансировки нагрузки (справа). Разные цвета обозначают разные вычислительные узлы

Для обеспечения корректности распределения и поиска фрагментов данных должны поддерживаться следующие ограничения:

— не допускается пустых доменов (не содержащих ни одной ячейки);

— домен каждого вычислительного узла должен быть смежен (иметь смежные ячейки) с доменами соседних с ним в топологии вычислительных узлов.

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

— 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“;

— время распределения или поиска фрагмента данных в худшем случае пропорционально диаметру топологии вычислительной сети, в то время как у Rope оно пропорционально числу всех узлов.

2.4. *Динамическая балансировка нагрузки в алгоритме Patch.* Для динамической балансировки нагрузки в алгоритме Patch, как и в алгоритме Rope, используется принцип диффузии. 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 одновременно может быть смежным узлом в других группах, что обеспечивает взаимодействие между группами и позволяет передавать нагрузку через все множество узлов, о чем будет сказано далее. Для каждого узла рассчитывается значение его нагрузки. В алгоритме Rope 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. Соседние узлы обмениваются значениями своей нагрузки, так что каждому узлу известны значения его нагрузки и нагрузки его соседей

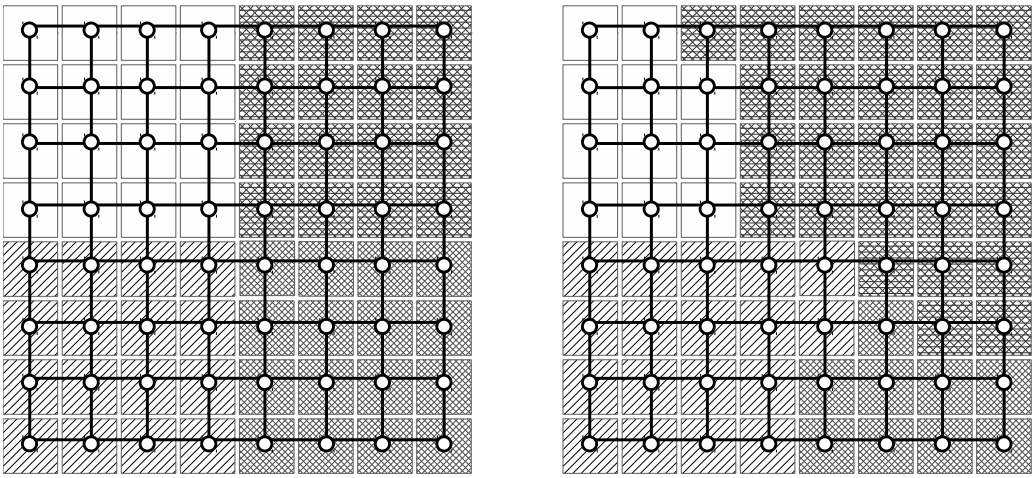


Рис. 2. Распределение данных (ячеек) по вычислительным узлам в алгоритме Patch, изначальное (слева) и после возможной балансировки нагрузки (справа). Разные цвета обозначают разные вычислительные узлы

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.

Передача нагрузки производится путем перераспределения координат между сегментами/доменами. В алгоритме Rope для изменения сегментов использовался сдвиг границы между смежными сегментами. В алгоритме Patch используется миграция ячеек между смежными доменами, с перегруженного узла на недогруженный. Для миграции выбирается группа ячеек на границе доменов. При миграции ячеек происходит изменение формы доменов. Фрагменты, отображенные на мигрировавшие ячейки, также мигрируют на новый узел, что и приводит к выравниванию нагрузки между узлами. Стоит отметить, что порядок координат не меняется, что позволяет соседним фрагментам данных оставаться на одном или соседних вычислительных узлах после балансировки.

При выборе группы ячеек для миграции должны учитываться следующие критерии, обеспечивающее дальнейшее эффективное функционирование алгоритма:

— суммарная величина нагрузки группы ячеек должна быть примерно равна величине нагрузки, которую нужно отдать на недогруженный узел;

— так как площадь границы домена зачастую пропорциональна объему коммуникаций с соседними узлами, эта площадь должна оставаться минимальной после миграции ячеек;

— после миграции домен должен продолжать представлять собой связную группу ячеек и иметь смежные ячейки с доменами соседних в топологии узлов.

Жадный алгоритм используется для выбора ячеек для миграции, в целях оптимизации времени работы этой процедуры. Сначала выбирается одна стартовая ячейка для миграции, затем к ней по одной добавляются смежные ячейки, пока не будет набрана нужная группа ячеек, при этом контролируется соблюдение вышеописанных критериев.

Распределенная природа алгоритма дозволяет параллельное осуществление многих миграций ячеек между разными узлами. Для синхронизации приема и отправки ячеек между узлами используется механизм транзакций. Каждая транзакция заключает в се-

бя пересылку одной группы ячеек от одного узла другому. В каждый момент времени каждый узел может выполнять только одну транзакцию (прием или отправка ячеек), при этом выполнение других транзакций от других узлов приостанавливается. Для определения порядка исполнения транзакций и предотвращения взаимных блокировок, каждой транзакции назначается случайный приоритет; каждый узел выполняет транзакции в порядке убывания их приоритета. Так как миграция ячеек может потребовать изменение информации о смежности в ячейках на нескольких узлах (не только участвующих непосредственно в приеме и отправке ячеек), домены которых смежны с изменяемым доменом, таким узлам отправляются сообщения для поддержания корректности этой информации.

Описанный алгоритм балансировки является распределенным, параллельным и масштабируемым на большое число вычислительных узлов, т. к. каждый узел взаимодействует только с конечным числом соседних в топологии узлов. Изменяя, автоматически или вручную, такие параметры алгоритма как частота вызова балансировки, порог балансировки, величина передаваемой нагрузки и т. д., можно регулировать скорость схождения всех узлов к сбалансированному состоянию, нивелируя возможный недостаток „диффузии“ в виде медленного схождения к глобальному балансу из-за использования только локальных коммуникаций.

3. Тесты. Для сравнения алгоритмов Patch и Rope использовалась фрагментированная реализация решения уравнения Пуассона в трехмерной области с помощью явной конечно-разностной схемы. Тестирование проводилось на кластере 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.

3.1. *Результаты тестов.* Для тестирования в задаче использовалась регулярная сетка размером 512^3, разбитая на 32^2 фрагмента данных по двум координатам. До 256 процессов использовалось для запуска. Конфигурация кластера позволяла поместить до 4-х процессов на один вычислительный узел. Для алгоритма Rope процессы были объединены в топологию „одномерная решетка“, для Patch — в топологию „двумерная решетка“.

Отслеживались следующие характеристики исполнения программы: средняя дистанция пересылки фрагмента данных (AvgSD, процессы) и средний суммарный объем переданных фрагментов данных (AvgSD, мегабайты).

Табл. 1 показывает результаты для случая равномерной загрузки процессов данными и вычислениями. Алгоритм Patch ожидаемо показывает меньшую среднюю дистанцию пересылки, чем Rope, что объясняются лучшим учетом соседства фрагментов данных в Patch: коммуникации происходят только между соседними процессами, поэтому дистанция пересылки никогда не становится больше 1. Одинаковый объем пересылаемых данных для Rope и Patch в этом тесте объясняется регулярностью задачи, но следует учесть, что в случае Rope те же данные пересылаются на большую дистанцию, чем в случае Patch, что приводит в итоге к большим затратам на коммуникации для Rope.

Для тестирования динамической балансировки нагрузки был создан изначальный дисбаланс путем распределения данных и вычислений только на половину работающих процессов. Целью балансировки было динамически перераспределить данные и вычисления на все процессы для их равномерной загрузки. Результаты исполнения программы показаны в табл. 2, а график нагрузки — на рис. 3. В данном тесте Patch выигрывает у Rope как по средней дистанции пересылки, так и по среднему суммарному объему отправленных данных, что опять же объясняется лучшей локальностью данных. Уменьшение средней дистанции пересылки и увеличение объема отправленных данных, по сравнению со

Таблица 1

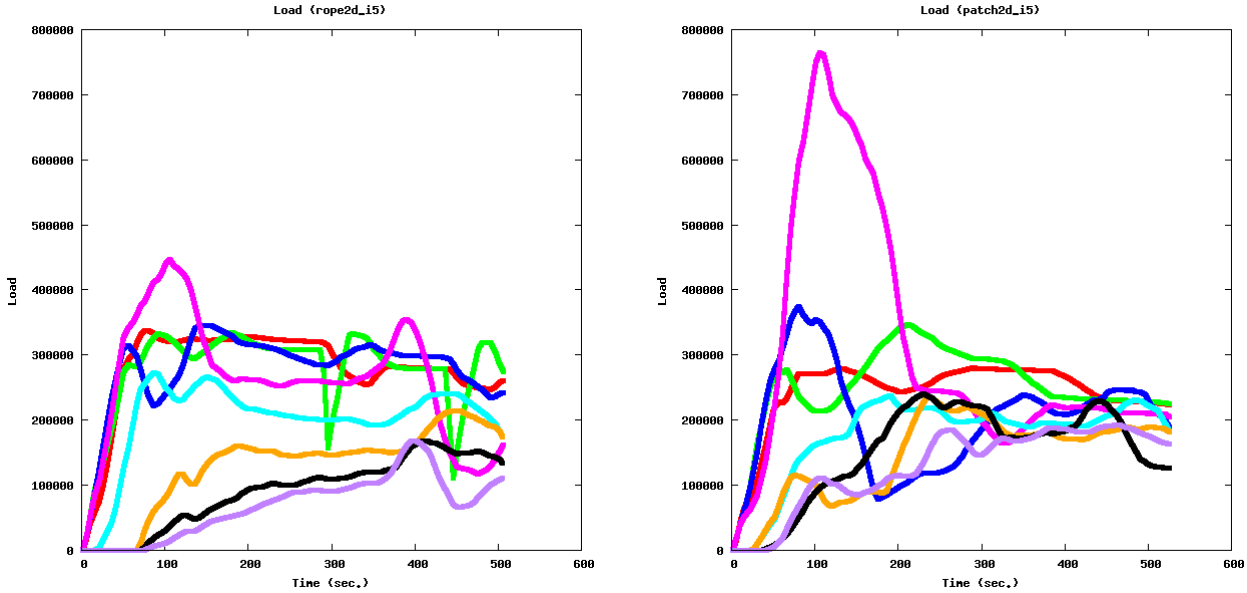
Сетка 512^3, 32^2 фрагментов, равномерное распределение

N	1	2	4	8	16	32	64	128	256
AvgSD, Rope	0	1,00	1,50	1,80	2,50	3,24	4,84	6,41	9,66
AvgSD, Patch	0	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
AvgSS, Rope	0	20,20	20,20	20,20	15,10	12,60	8,80	6,90	4,70
AvgSS, Patch	0	20,20	20,20	20,20	15,10	12,60	8,80	6,90	4,70

Таблица 2

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

N	2	4	8	16	32	64	128	256
AvgSD, Rope	1,00	1,19	1,42	1,48	1,46	1,80	1,30	1,44
AvgSD, Patch	1,00	1,03	1,18	1,27	1,31	1,24	1,16	1,04
AvgSS, Rope	7787,60	6246,70	2821,60	1032,20	1192,60	598,00	566,90	392,00
AvgSS, Patch	9917,90	4955,90	2225,30	1226,20	714,60	444,10	193,80	91,80



а) б)

Рис. 3. Динамическая балансировка нагрузки в алгоритмах Rope (слева) и Patch (справа). Нагрузка разных процессов показана разным цветом

случаем равномерного распределения, связано с учетом вклада от миграции фрагментов данных при балансировке, которая происходила между соседними процессами.

Рис. 3 показывает график нагрузки для 8-и процессов в ходе динамической балансировки нагрузки для алгоритмов Rope и Patch. Нагрузка узла измерялась как текущий объем активных данных на узле. В виду того, что число одновременно взаимодействующих узлов в алгоритме Patch больше, чем в Rope, алгоритму Patch быстрее удается привести данные к сбалансированному состоянию.

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 входят дальнейшая 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 // J. Supercomputing. 2017. V. 73. N 2. P. 726—732. Springer US.

2. Malyshkin V. E., Perepelkin V. A., Schukin G. A. Distributed Algorithm of Data Allocation in the Fragmented Programming System LuNA // PaCT 2015. LNCS, V. 9251. P. 80–85. Springer, Heidelberg.

3. Malyshkin V. E., Perepelkin V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // PaCT 2011, LNCS, V. 6873, P. 53–61. Springer, Heidelberg.

4. Malyshkin V. E., Perepelkin V. A. Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System // J. Supercomputing. 2012. V. 61, N 1, P. 235–248.

5. Malyshkin V. E., Perepelkin V. A. The PIC Implementation in LuNA System of Fragmented Programming // J. Supercomputing. 2014. V. 69. N 1. P. 89–97.

6. Kraeva M. A., Malyshkin V. E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers // J. Future Generation Computer Systems. 2001. V. 17. N 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 // J. IEEE Transactions on Parallel and Distributed Systems. 2014. V. 25. N 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. 12–12. USENIX Association, Berkeley, CA, USA.

9. Bikshandi G., Guo J., Hoeflinger D., Almasi G., Fraguela B. B., Garzaran 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. ACM New York, NY, USA.

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

11. Begau C., Sutmann G. Adaptive dynamic load-balancing with irregular domain decomposition for particle simulations // J. Computer Physics Communications. 2015. V. 190. P. 51-–61. Elsevier B. V.

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. V. 183. N 12. P. 2608–2615. Elsevier B. V.

13. Deng Y., Peierls R. F., Rivera C. An Adaptive Load Balancing Method for Parallel Molecular Dynamics Simulations // J. of Computational Physics. 2000. V. 161. N 1. P. 250–263. Elsevier B. V.

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. V. 74. N 4. P. 531—553. John Wiley & Sons, Ltd.

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. IEEE.

16. Веб-страница с демонстрацией работы алгоритмов Rope и Patch. [Electron. res.]: `http://ssd.sscc.ru/en/algorithms`.

Малышкин Виктор Эммануилович — д-р технич. наук, профессор ИВМ и МГ СО РАН, зав. лабораторией синтеза параллельных программ, e-mail: `malysh@ssd.sscc.ru`, тел.: +7 (383) 330-89-94.

В. Э. Малышкин получил степень магистра математики в Томском государственном университете (1970), степень кандидата физико-математических наук в Вычислительном центре СО АН СССР (1984), степень доктора технических наук в Новосибирском государственном университете (1993). В настоящее время является заведующим лабораторией синтеза параллельных программ в Институте вычислительной математики и математической геофизики СО РАН. Он также основал и в настоящее время возглавляет кафедру Параллельных вычислений в Национальном исследовательском университете Новосибирска и кафедру Параллельных вычислительных технологий в НГТУ. Является одним из организаторов международной конференции PaCT (Parallel Computing Technologies), проводимой каждый нечетный год в России. Имеет более 110 публикаций по параллельным и распределенным вычислениям, синтезу параллельных программ, суперкомпьютерному программному обеспечению и приложениям, параллельной реализации крупномасштабных численных моделей. В настоящее время область его интересов включает параллельные вычислительные технологии, языки и системы параллельного программирования, методы и средства параллельной реализации крупномасштабных численных моделей, технология активных знаний.

Victor Malyshkin received his M. S. degree in Mathematics from the State University of Tomsk (1970), Ph. D. degree in Computer Science from the Computing Center of the Russian Academy of Sciences (1984), Doctor of Sciences degree from the State University of Novosibirsk (1993). He is presently the head of Supercomputer Software Department of the Institute of Computational Mathematics and Mathematical Geophysics, Russian Academy of Sciences. He also found and presently heading the Chair of Parallel Computing at the National Research University of Novosibirsk. He is one of the organizers of the PaCT (Parallel Computing Technologies) series of international conferences that are held each odd year in Russia. He published over 110 papers on parallel and distributed computing, parallel program synthesis, supercomputer software and applications, parallel implementation of large scale numerical models. His current research interests include parallel computing technologies, parallel programming languages and systems, methods and tools for parallel implementation of large scale numerical models, active knowledge technology.

Щукин Георгий Анатольевич — ИВМ и МГ СО РАН, младш. науч. сотр., e-mail: `schukin@ssd.sscc.ru`, тел.: +7 (913) 706-54-74.

Родился в Новосибирске в 1986 году. В 2009 окончил Новосибирский государственный технический университет со степенью магистра прикладной математики и информатики. Текущее место работы: Институт вычислительной математики и математической геофизики, младший научный сотрудник. Участвует в разработке системы фрагментированного программирования LuNA, а также других проектах лаборатории синтеза па-

раллельных программ. Профессиональные интересы: параллельное и системное программирование, языки программирования, компьютерная графика.

Born in Novosibirsk in 1986. In 2009 graduated from Novosibirsk State Technical University as master of Applied Mathematics and Informatics. Nowadays works as junior researcher at Institute of Computational Mathematics and Mathematical Geophysics SB RAS. Participates in development of LuNA fragmented programming system and other projects of Supercomputer Software Department. Professional interests: parallel and system programming, programming languages, computer graphics.

Дата поступления — 30.11.2017