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:

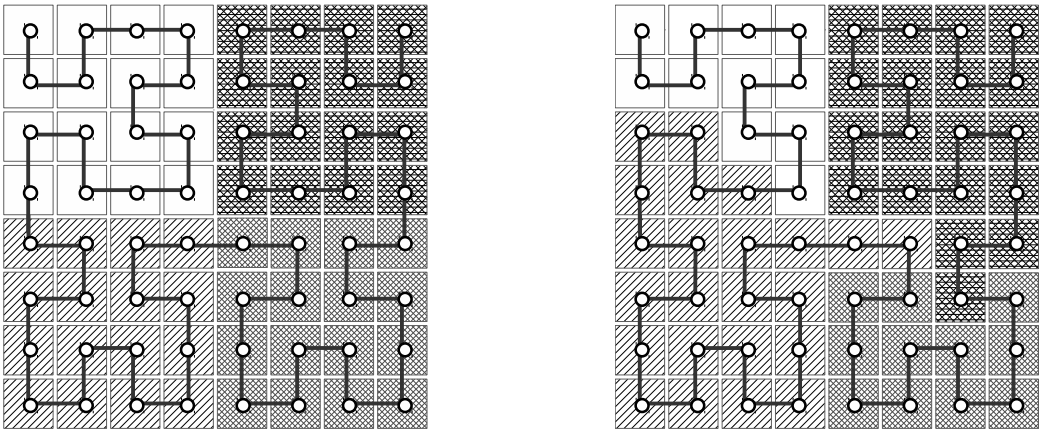
- не допускается пустых доменов, т.е. не содержащих ни одной ячейки;
- домен каждого вычислительного узла должен быть смежен (иметь смежные ячейки) с доменами соседних с ним в топологии вычислительных узлов.

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).

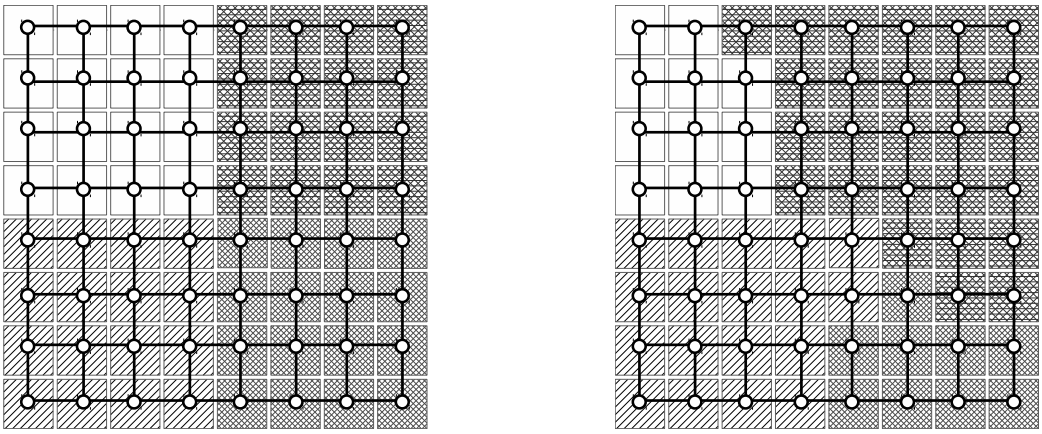
3.4. Динамическая балансировка нагрузки в алгоритме Patch

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

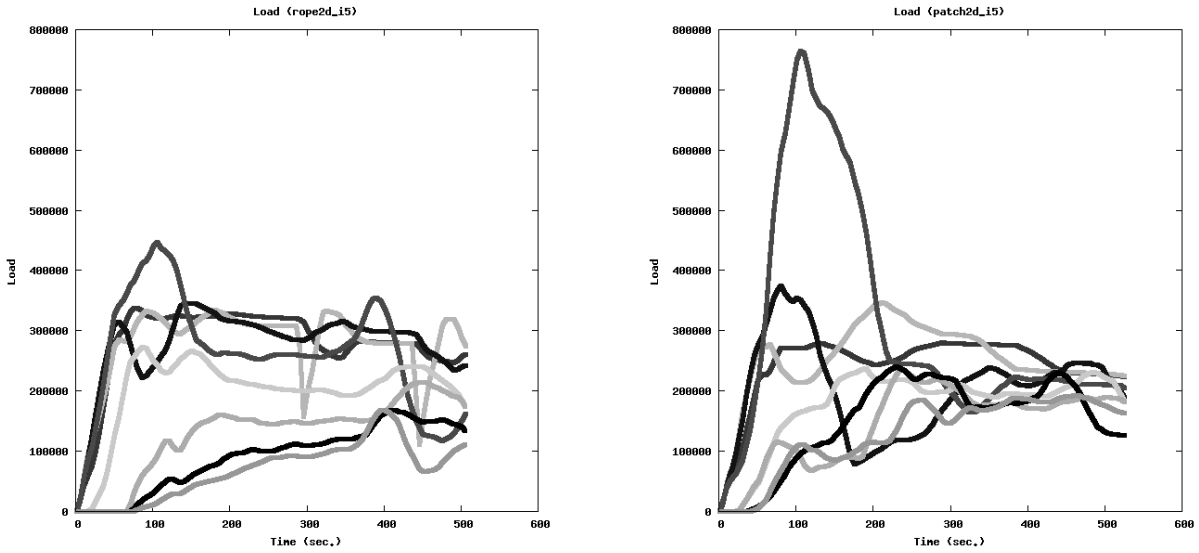
Заключение

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.

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).