Параллельные алгоритмы построения изоповерхностей на больших сетках*

В.А. Киев, А.К. Кузин, С.Г. Орлов,
Б.Н. Четверушкин, Н.Н. Шабров, М.В. Якобовский

Рассматривается задача о построении редуцированной изоповерхности на большой (порядка 10^9 узлов) нерегулярной сетке тетраэдров. В каждом узле сетки задано значение непрерывного скалярного поля, изоповерхность которого требуется найти. Внутри каждого тетраэдра поле интерполируется линейно. Предложены реализации параллельных алгоритмов построения изоповерхностей, ориентированные на многоядерные вычислительные архитектуры; рассматривается также возможность использования GPU.

1. Введение

Исходная сетка разбита на домены, в каждом из которых — порядка 10^6 узлов (таким образом, имеется около 1000 доменов); имеется также соответствие между узлами границ, разделяющих соседние домены. Каждый домен может быть обработан отдельно, независимо от остальных. Результат обработки домена — кусок редуцированной изоповерхности на нём. Полученные на доменах куски изоповерхности попарно сшиваются и затем заново редуцируются.

Схема рассматриваемой программной реализации алгоритмов построения редуцированной изоповерхности представлена на рис. 1.

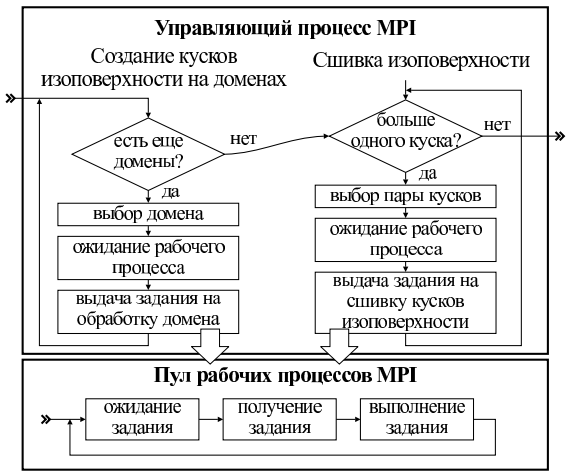

Рис. 1. Схема работы программы

Вначале создается пул процессов MPI, в одном из которых выполняется код планировщика заданий, а остальные ждут от него заданий. Планировщик выдает освободившимся рабочим процессам задания, сводящиеся к обработке отдельных доменов. Таким образом, обеспечивается динамическая балансировка загрузки CPU. Пока количество доменов превышает количество CPU, имеет место рост производительности с ростом количества CPU.

После того, как все домены обработаны, производится сшивка кусков и последующее редуцирование получающихся более крупных кусков. Процесс продолжается до тех пор, пока не останется единственный кусок. На этом этапе рабочим процессам выдаются задания, состоящие в сшивке и редуцировании пары кусков изоповерхности. Выбором пар

*Авторы работы благодарят РФФИ за поддержку исследований в рамках гранта № 09-07-12020-офи_м. Также авторы благодарны компании «Ниагара компьютерс» за предоставленное компьютерное оборудование.

занимается процесс планировщикаика заданий.

При размере домена порядка 10^6 узлов его обработка возможна как на CPU, так и на GPU, причем следует ожидать, что благодаря параллелизации внутри домена использование GPU позволит значительно сократить время обработки. Для эффективного использования GPU разработаны специализированные параллельные версии алгоритмов создания и редуцирования изоповерхности. Рабочие потоки MPI определяют, следует ли использовать CPU или GPU — в зависимости от того, имеется ли в системе свободный GPU. Для этого разработан распределитель ресурсов, отслеживающий занятость вычислительных ресурсов.

Таким образом, в программной реализации выделяются несколько отдельных частей: алгоритм генерации изоповерхности на одном домене (реализации для CPU и GPU); алгоритм редуцирования изоповерхности (реализации для CPU и GPU); алгоритм сшивки двух кусков изоповерхности (реализация для CPU); планировщик заданий; распределитель ресурсов.

2. Создание изоповерхности

Для алгоритма генерации изоповерхности существенно, что сетка состоит из тетраэдров, и скалярное поле линейно на каждом из них. Поэтому каждое ребро сетки пересекается с изоповерхностью не более, чем в одной точке. Пересечение изоповерхности с тетраэдром — либо треугольник, либо четырехугольник, так как часть изоповерхности внутри каждого тетраэдра — плоская.

Особо следует отметить случай, когда значение поля в некотором узле в точности равно его значению на изоповерхности. При этом возникает негрубая ситуация, сильно усложняющая весь алгоритм. Разработанные версии алгоритма избегают этой проблемы, добавляя малые слагаемые к тем узловым значениям поля, которые в точности равны значению на изоповерхности.

Алгоритм создания изоповерхности состоит из следующих шагов.

1. Определение диапазона $[f_{\min},\ f_{\max}]$ узловых значений поля f.

2. Добавление малых слагаемых к узловым значениям, совпадающим с заданным на изоповерхности f_0. Величина слагаемого выбирается равной εf_0, если $f_0 \neq 0$ и $\varepsilon\,(f_{\max} - f_{\min})$, если $f_0 = 0$. Величина ε принимается равной 10^{-7}, так как вычисления производятся в числах с плавающей запятой с одинарной точностью. Такое изменение поля не сказывается на видимой геометрии изоповерхности, но существенно упрощает алгоритм, устраняя негрубые ситуации.

3. В цикле по всем ребрам сетки домена выясняется, пересекается ли ребро с изоповерхностью. Если это так, ребру ставится в соответствие очередной номер узла сетки изоповерхности. Также вычисляется параметр от 0 до 1, определяющий положение этого узла на ребре.

4. Создание треугольников изоповерхности. На этом этапе необходимо обойти все тетраэдры, пересекающиеся с изоповерхностью, и сгенерировать для каждого из них обходы одного или двух треугольников, являющихся частью изоповерхности в данном тетраэдре. Чтобы обеспечить согласованность ориентации треугольников на соседних тетраэдрах, достаточно располагать узловыми значениями поля в каждом отдельном тетраэдре (предполагается, однако, что ориентации всех тетраэдров исходной сетки согласованы).

5. Создание узлов изоповерхности. На этом этапе вычисляются координаты узлов изоповерхности, фактически найденных на шаге 3.

6. Построение соответствия между узлами на краю изоповерхности и ребрами сетки домена на его границе. Этот шаг необходим для последующей сшивки кусков изоповерхностей на соседних доменах. Указанное соответствие позволяет при сшивке определить соответствующие друг другу узлы алгебраическим путем, не сравнивая их координаты.

Отметим, что реализация этого алгоритма на GPU нетривиальна, но может быть полностью сведена к последовательности стандартных алгоритмов `for_each`, `transform`, `partition`, `sort`, `scan`, `gather`, `scatter`, `unique`, `remove_if`, `copy` [1]. Разработанное программное обеспечение использует библиотеку Thrust [3], предоставляющую параллельные реализации этих алгоритмов для GPU.

3. Редуцирование изоповерхности и сшивка кусков

Предлагаемый алгоритм позволяет существенно уменьшить размер сетки, представляющей изоповерхность и полученной на предыдущем этапе. Основная операция, производимая над сеткой изоповерхности — *удаление ребра* (рис. 2), то есть стягивание ребра в узел. Кроме этого, некоторые пары соседних треугольников заменяются другими парами, в которых общее ребро проходит иначе (рис. 3); отметим, что каждая такая пара идентифицируется некоторым ребром сетки изоповерхности. Вторую операцию будем называть *заменой ребра*. Возможность выполнения одной из двух указанных операций на ребре сет-

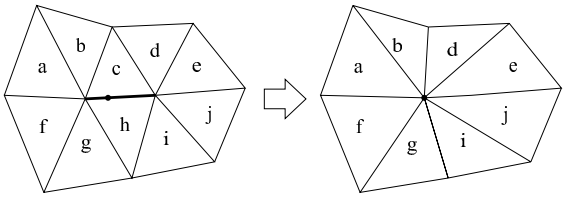

Рис. 2. Удаление ребра сетки

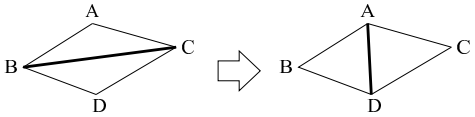

Рис. 3. Замена ребра сетки

ки определяется значением критерия Φ, вычисляемого на этом ребре (о нем речь пойдет ниже). Существенно, что выполнение операции на ребре влияет на значение критерия на близких ребрах. При разработке алгоритма предполагалась возможность его распараллеливания (с использованием GPU), и это наложило определенные требования на порядок выполняемых действий. Алгоритм выполняет последовательность итераций, на каждой из которых выполняются следующие шаги.

1. Вычисление значения критерия Φ_i на i-том ребре, для всех ребер сетки изоповерхности. Эта операция может быть выполнена параллельно (в частности, в реализации на GPU для каждого ребра используется отдельный поток выполнения). Ребра, на которых Φ_i превышает некоторое Φ^*, определяющее качество редуцированной изоповерхности, потенциально подходят для удаления (или замены, что для каждого ребра определяется отдельным флажком). Будем обозначать символом G^* множество ребер g_i, на которых $\Phi_i > Phi^*$. Отметим, что в случае удаления ребра также вычисляется параметр, определяющий положение узла, которое заменит это ребро.

2. Выбор подмножества G_1^* ребер, которые можно удалить или заменить одновременно, из G^*. Возможность одновременного удаления таких ребер подразумевает, что удаление любого из ребер, принадлежащих G_1^*, не влияет на значение критерия на любом другом ребре этого подмножества. Для выбранного нами критерия это, в свою очередь, означает, что в графе, образованном узлами и ребрами сетки изоповерхности, длина пути от любого узла, принадлежащего ребру из G_1^*, до любого узла, принадлежащего другому узлу из G_1^*, должна быть не менее двух. В настоящее время реализован лишь последовательный алгоритм выбора такого подмножества.

3. Выполнение операции удаления или замены для всех ребер из G_1^*. Эта операция может быть произведена параллельно.

Итерации продолжаются до тех пор, пока G_1^* не станет пустым, или же пока не возникнет двух идущих подряд итераций, на которых не удалено ни одно ребро.

Для вычисления критерия Φ_i на ребре g_i используется информация обо всех гранях изоповерхности, содержащих узлы этого ребра. Назовем *листом* с центром в k-том узле, L_k, множество всех граней изоповерхности, содержащих этот узел. Значение Φ_i определяется объединением листов $L_{g_{i,1}} \cup L_{g_{i,2}}$, где $g_{i,1}$ и $g_{i,2}$ — номера узлов на концах ребра g_i (рис. 4). Обозначим также через $e_{g_{i,1}}$ и $e_{g_{i,2}}$ номера граней, содержащих ребро g_i. Отметим, что алгоритм создания изоповерхности гарантирует, что каждое ребро принадлежит либо двум граням, либо (для ребер на краю сетки) одной.

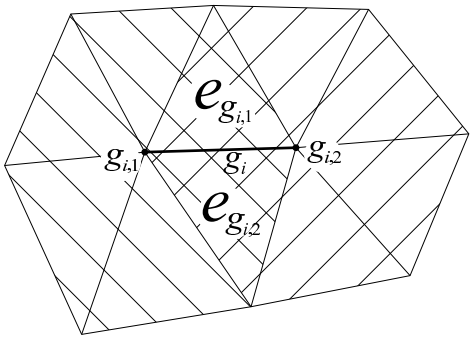

Рис. 4. Листы с центрами в узлах ребра; грани, содержащие ребро

Процедура вычисления критерия Φ_i на ребре g_i довольно разветвленная. Наиболее часто значение вычисляется на основании оценки локальной кривизны листов $L_{g_{i,1}}$ и $L_{g_{i,2}}$. Рассмотрим для примера внутреннее ребро — оно принадлежит двум граням. Единичные нормали к этим граням обозначим $\boldsymbol{n}_1$ и $\boldsymbol{n}_2$. Каждый из листов $L_{g_{i,j}}$ $(j = 1, 2)$ разделяется на две части $L_{g_{i,j},1}$ и $L_{g_{i,j},2}$ следующим образом. В первую часть попадает грань $e_{g_{i,1}}$. Далее обходятся соседние грани этого листа, причем в такую сторону, чтобы следующей за $e_{g_{i,1}}$ гранью была *не* грань $e_{g_{i,2}}$. В первую часть попадут все идущие подряд грани, нормали к которым ближе к $\boldsymbol{n}_1$, нежели к $\boldsymbol{n}_2$. Остальные грани попадут во вторую часть. Обозначим множества единичных нормалей к граням из первой и второй частей как $\boldsymbol{n}_{g_{i,j},1}$, $\boldsymbol{n}_{g_{i,j},2}$. Для каждого листа вычисляется

$$\Phi_{i,j} = \min_{s=1,2} \left\{ \min_{\boldsymbol{n} \in \boldsymbol{n}_{g_{i,j},s}} \{\boldsymbol{n}_s \cdot \boldsymbol{n}\} \right\}, \qquad j = 1, 2$$

Значение $\Phi_{i,j}$ окажется тем больше, чем ближе нормали граней из $L_{g_{i,j},1}$ к $\boldsymbol{n}_1$, а граней из $L_{g_{i,j},2}$ — к $\boldsymbol{n}_2$. Оно достигает единицы для плоских и для «согнутых» листов (рис. 5). Наконец, значение критерия Φ_i на ребре вычисляется по формуле

$$\Phi_i = \Phi_{i,1} t_i + \Phi_{i,2}(1 - t_i), \qquad t_i = \frac{2}{\pi} \operatorname{arctg} \frac{\Phi_{i,2}}{\Phi_{i,1}}.$$

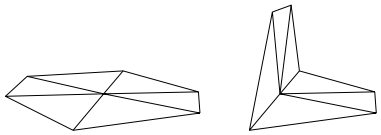

Рис. 5. Плоский и «согнутый» листы

Параметр t_i определяет положение узла, который заменит ребро при его удалении.

Использование указанных формул для вычисления Φ_i позволяет редуцировать поверхности, имеющие углы, удаляя ребра, лежащие на этих углах (а не только на «плоской» части сетки).

При вычислении значений критерия на ребрах иногда реализуются и другие ветки процедуры, позволяющие удалить очень короткие ребра; особо рассмотреть ребра с листами из трех граней; особо рассмотреть ребра, лежащие на границе; заменить длинные ребра между очень узкими треугольникам. В данной статье не представляется возможным описать все эти ветки, однако их наличие позволяет существенно повысить качество и уменьшить количество граней редуцированной поверхности. Кроме того, производится ряд тестов, отбраковывающих ребра, удаление которых привело бы к нарушению топологии сетки или появлению почти вырожденных граней. Благодаря этому гарантируется гомеоморфизм исходной и редуцированной поверхностей.

Отметим еще, что алгоритм редуцирования предусматривает возможность запретить удаление произвольного подмножества узлов на краю поверхности. Она требуется для обеспечения возможности сшивки кусков изоповерхности на соседних доменах.

Сшивка пары кусков изоповерхности сводится к созданию единой нумерации узлов и граней с учетом того, что часть узлов, принадлежащая интерфейсной границе между доменами принадлежит обоим кускам. Кроме того, после сшивки производится повторное редуцирование, цель которого — проредить сетку в месте, где проходит «шов».

4. Балансировка и планирование заданий

При разработке приложения для мультипроцессорной системы неминуемо возникает задача балансировки загрузки созданных процессов. В настоящей задаче это достигается простым и в то же время эффективным механизмом динамического распределения работы между MPI процессами. Один MPI процесс — управляющий, его задача состоит только в распределении работы между остальными процессами. Рабочий процесс оповещает о своей готовности управляющего и в ответном сообщении получает задание. По выполнении задания он отсылает главному процессу результат и опять ждет сообщение с новым заданием, либо команду завершения. Управляющий же процесс по запросу раздает работу из очереди заданий. В силу специфики рассматриваемой задачи такой подход весьма эффективен и позволяет свести практически к нулю время простоя вычислительных мощностей.

Однако использование гетерогенных вычислительных систем, таких как кластер с установленными на узлах GPU создает дополнительную проблему распределения заданий между процессорами для получения максимального быстродействия. Для достижения этой цели можно было бы создать менеджер ресурсов, позволяющий процессу на узле принять решение, какое устройство (GPU или CPU) следует использовать для минимизации времени выполнения текущей задачи; менеджер предсказывал бы время выполнения операции на основании накопленной статистики о ранее завершенных заданиях, а также вел бы учет загрузки устройств узла. В случае нехватки ресурсов рабочий процесс переводится в состояние ожидания до их освобождения.

Описанный вариант менеджера ресурсов был разработан, однако в рассматриваемой нами задаче от него пришлось отказаться в пользу более простой модификации, оказавшейся

в то же время более эффективной. Работа менеджера сводится к назначению фиксированного ресурса для каждого рабочего процесса, так что одни процессы используют только CPU, а другие — только GPU, причем количество последних не превышает количества GPU в системе. Более высокая эффективность упрощенного менеджера ресурсов связана с отсутствием необходимости перевода рабочих процессов в режим ожидания.

Менеджер ресурсов — один на узел, поэтому он обслуживает все рабочие процессы, запущенные на этом узле.

5. Тестирование программной реализации

Программная реализация алгоритмов построения редуцированной изоповерхности тестировалась на узле с 12 Гб оперативной памяти, 16 ядрами CPU и двумя GPU Tesla с 4 Гб памяти на каждом.

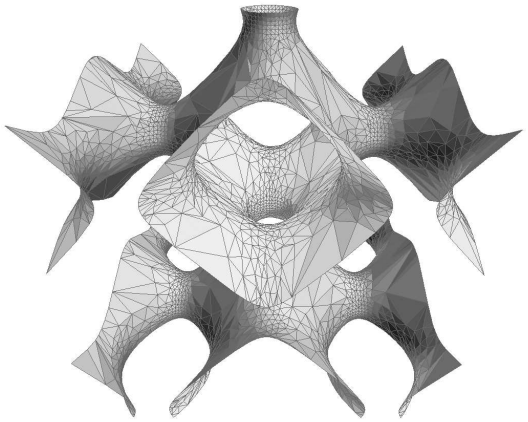

Рис. 6. Фрагмент тестовой изоповерхности

В качестве теста рассмотрена область в виде куба с размерами $5 \times 5 \times 5$ с заданной на ней сеткой $500 \times 500 \times 500$ узлов и полем

$$f(x, y, z) = 2\cos(10x) + 2\sin(10y) + \cos(10z);$$

изоповерхность строилась для уровня $f = 0,5$; её фрагмент представлен на рис. 6. Проводились три серии тестов, отличающихся размерами домена:

- 250 доменов по $5 \cdot 10^5$ узлов;
- 125 доменов по 10^6 узлов;
- 63 домена по $2 \cdot 10^6$ узлов.

Благодаря специальному выбору поля f в каждом из этих случаев в домены попали примерно одинаковые по размеру куски изоповерхности.

Приведенные далее графики относятся к серии 125 доменов по 10^6 узлов; отметим, что зависимость быстродействия программы от размера домена слабая; с ростом размера домена она незначительно увеличивается.

Были рассмотрены зависимости времени обработки всех доменов от количества рабочих MPI-процессов и от размера домена. Их графики представлены на рис. 7. Выигрыш при использовании GPU очевиден, особенно это заметно при малом количестве рабочих процессов. Отметим, что время сшивки изоповерхности во всех случаях невелико и не превышает 10% от времени обработки доменов.

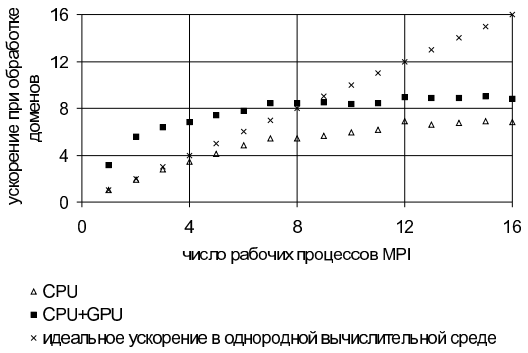

Рис. 7. Ускорение расчетов в зависимости от количества рабочих процессов MPI

6. Заключение

Тестирование разработанной реализации алгоритмов построения редуцированной изоповерхности показало её работоспособность и масштабируемость, пока число процессоров остается значительно меньше числа доменов.

Анализ результатов тестирования показал, что уменьшение прироста производительности с ростом числа используемых CPU скорее всего связано с пропускной способностью подсистемы оперативной памяти. Чем больше запущено процессов, тем больше нагрузка на нее.

Использование имеющихся в системе GPU позволяет значительно увеличить скорость расчета. Алгоритм редуцирования изоповерхности выполняется на GPU намного быстрее, чем на CPU, хотя и не в десятки раз. Отчасти это связано с необходимостью частых обменов данных с GPU, отчасти — с дополнительными операциями, требующимися в GPU-версии алгоритма.

Список литературы

1. S. Gorlatch, C. Lengauer. (De)Composition for Parallel Scan and Reduction. mppm, pp.23, Massively Parallel Programming Models, 1997.

2. Burkhard Wuensche. A Survey and Evaluation of Mesh Reduction Techniques. Proceedings of IVCNZ '98, Auckland University, Auckland, November 1998, pages 393–398.

3. Thrust — a CUDA library of parallel algorithms. http://code.google.com/p/thrust/