МЕТОД ПАРАЛЛЕЛЬНОЙ ГЕНЕРАЦИИ ТРЕХМЕРНЫХ НЕСТРУКТУРИРОВАННЫХ СЕТОК ДЛЯ ЗАДАЧ МОДЕЛИРОВАНИЯ И ВИЗУАЛИЗАЦИИ НА СУПЕРКОМПЬЮТЕРАХ

К.Н. Киселёв, О.В. Джосан

В настоящее время проведение научных расчетов на высокопроизводительных многопроцессорных системах и их визуализация стали довольно популярны из-за широкой распространенности и доступности последних. Как следствие, появляется необходимость в разработке некоторого инструментария ученого, с помощью которого он без приложения дополнительных усилий на организацию данных и процесса вычислений мог бы получать наиболее точные и полные результаты в изучаемой области, тем самым, сконцентрировавшись на изучении проблемы.

Чаще всего подобное может быть полезным при решении задач, связанных с моделированием физических объектов или явлений. Модель может иметь такую природу, что, к примеру, некоторые области имеет смысл визуализировать с высокой степенью точности, в то время как другие могут быть представлены на небольшом числе примитивов. Таким образом, становится важным наиболее рационально распределить ресурсы для хранения структуры подходящей для задачи решетки.

Рациональность может достигаться путем использования нерегулярной (неструктурированной) решетки. Регулярной называется решетка, каждую ячейку (примитив) которой можно адресовать алгебраическим путем. Нерегулярная (неструктурированная) же решетка представляет собой неориентированный граф, заданный, в общем случае, списком ребер. В случае использования последней не составляет труда добавить в сетку, например, еще один узел в некоторую ее часть без увеличения разрешения остальной части решетки [2]. В результате данное представление получается достаточно гибким и в тоже время емким.

Визуализация подобных результатов может проводиться на высокопроизводительных многопроцессорных системах, в связи с чем, важным аспектом становится правильная организация и распределение данных между узлами для достижения эффективного распараллеливания. Существует два основных подхода к генерации неструктурированных решеток: метод распространяющегося фронта и метод на основе свойств триангуляции Делоне.

Метод распространяющегося фронта [3] заключается в последовательном построении следующего слоя тетраэдров (треугольников в двумерном случае) на основании текущего активного слоя. На каждом шаге активный слой представляет собой множество примитивов, являющихся границей заполняемой в данный момент области. Каждый примитив достраивается до тетраэдра (треугольника) путем добавления нового узла, либо уже принадлежащего решетке. То какой узел будет оптимален для добавления, определяется по введенной оценочной функции. Эта функция выбирается чаще всего на основе каких-либо физических особенностей модели, но может быть и универсальной, например, когда нужно всего лишь ограничить средний объем (или другие геометрические характеристики примитивов) окрестностью некоторой величины. В данном подходе важна аккуратность в выборе нового узла, так как это не должно привести к пересечениям примитивов решетки.

Триангуляция множества точек S называется триангуляцией Делоне, если окружность(сфера), описанная около любого из треугольников(тетраэдров) на точках S, не содержит в себе других точек этого множества [1]. Триангуляция Делоне обладает следующими важными свойствами: 1) минимальный угол среди всех треугольников максимален (данное свойство полезно, так как наличие в решетке треугольников с очень маленьким углом, но большой площадью, сильно понижает равномерность распределения нагрузки при распараллеливании генерации); 2) однозначно соответствует диаграмме Вороного (если исходные точки S смежных областей диаграммы соединить, то получится триангуляция Делоне, причем существуют алгоритмы построения диаграммы за O(NlogN)), где N – это количество точек.

Целью данной работы является реализация параллельной версии генератора трехмерных неструктурированных сеток и анализ эффективности его применения к моделям, имеющим порядка 10^6 узлов. Реализация подобного генератора с применением распараллеливания существующих методов построения решетки может значительно повысить эффективность научных расчетов, за счет уменьшения временных затрат на визуализацию. При создании нужно стремиться сделать генератор как можно более масштабируемым, а, кроме того, адаптивным. Последнее означает независимость процесса построения решетки и процесса вычислений, проводимых пользователем, то есть универсальность по отношению к расчетной задаче и процессу визуализации.

Общая схема работы параллельного генератора неструктурированных сеток.

Первым шагом работы генератора является разбор исходных данных и построение внутреннего представления решетки поверхности, которая дается на вход. Далее идет процесс разбиения поверхности на домены (способ разбиения описывается далее). После достижения требуемой глубины в дереве разбиения

начинается процесс генерации решетки для внутренней области каждого домена. Последняя осуществляется с помощью триангуляции Делоне в пространстве, то есть разбиения нужной нам области на оптимальные тетраэдры. Этот этап выполняется уже параллельно для каждого из доменов на разных процессорах. Общая схема представлена на Рис.1.

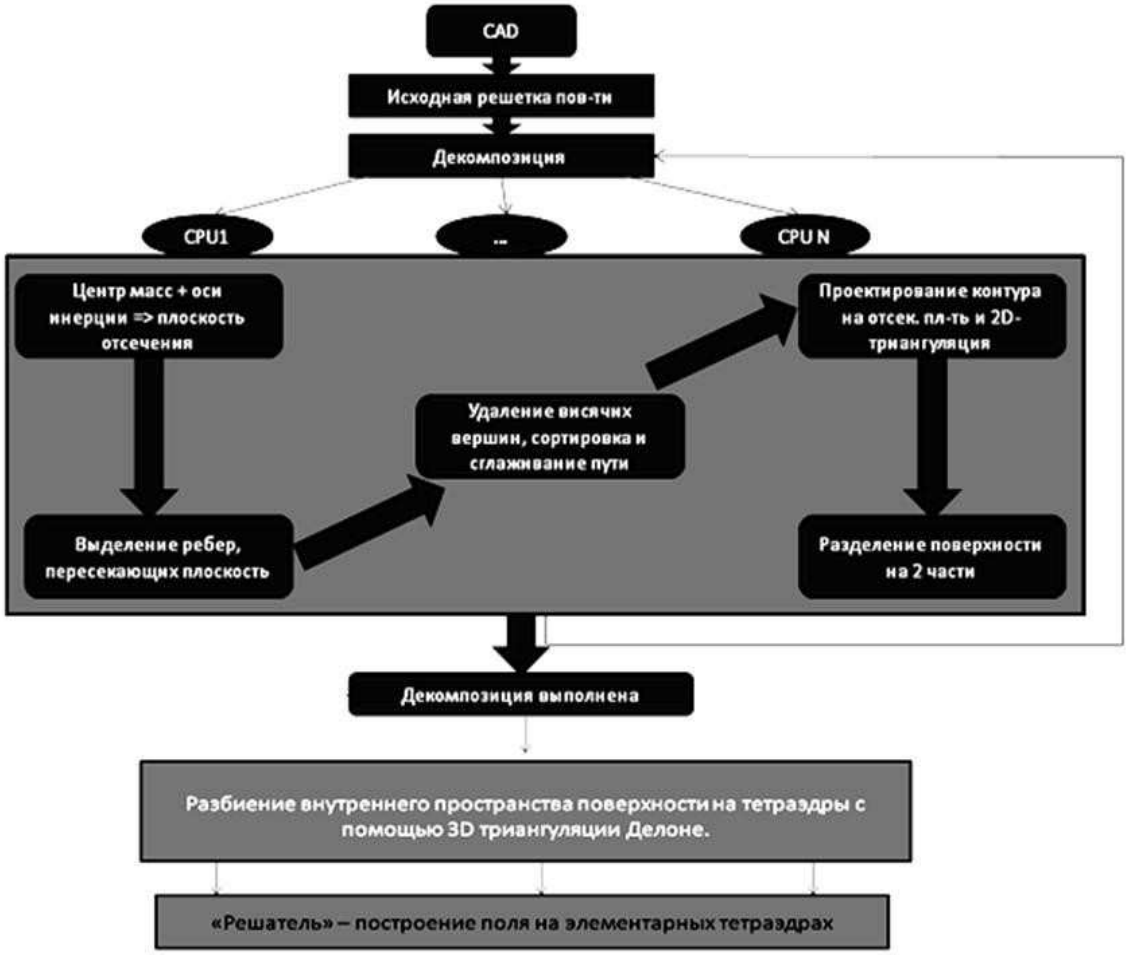

Рис.1 Общая схема работы алгоритма параллельного генератора

Декомпозиция поверхности на домены для выполнения параллельной триангуляции.

1) Нахождение разбивающей плоскости

Главным шагом в предлагаемом методе генерации является удачная декомпозиция исходной поверхности. Существует множество вариантов разбиения последней на отдельные области, важным же здесь в первую очередь будет являться балансировка вычислительной нагрузки на процессоры. Под нагрузкой понимается количество примитивов, обрабатываемых отдельным процессором.

Самым простым является так называемое разбиение «по длине». Другой способ разбиения – это разбиение «по объему», которое заключается в разделении поверхности на куски, имеющие одинаковый объем. В данной работе предлагается разбивать поверхность на области, опираясь на понятия центра масс и главных осей инерции. Для этого каждому узлу решетки присваивается единичный вес, далее рассчитывается центр масс и главные оси инерции. По узлам строится матрица следующего вида (здесь V – множество узлов, xg, yg, zg – координаты центра масс):

$$\begin{pmatrix} \sum_{v\in V}(y_v-y_g)^2+(z_v-z_g)^2 & -\sum_{v\in V}(x_v-x_g)(y_v-y_g) & -\sum_{v\in V}(x_v-x_g)(z_v-z_g) \\ -\sum_{v\in V}(x_v-x_g)(y_v-y_g) & \sum_{v\in V}(z_v-z_g)^2+(x_v-x_g)^2 & -\sum_{v\in V}(z_v-z_g)(y_v-y_g) \\ -\sum_{v\in V}(x_v-x_g)(z_v-z_g) & -\sum_{v\in V}(z_v-z_g)(y_v-y_g) & \sum_{v\in V}(y_v-y_g)^2+(x_v-x_g)^2 \end{pmatrix}$$

Собственные векторы этой матрицы вместе с центром масс будут задавать главные оси инерции. Разбиение проводится плоскостью, проходящей через центр масс и перпендикулярной главной оси инерции, соответствующей минимальному собственному значению [5], на две части, которые далее будем называть «левой» и «правой». Вследствие того, что мы учитываем и «маленькие» и «большие» примитивы, так как не ориентируемся на их геометрические размеры, на каждом шаге в левой и правой части будет оставаться примерно равное количество узлов сетки поверхности, вне зависимости от распределения разрешения. Таким образом, достигается равномерность нагрузки на процессоры, относительно количества обрабатываемых данных.

2) Выделение замкнутого пути из ребер

После того как разбивающая плоскость найдена, нужно разделить список ребер на 2 части «левую» и «правую». На первом шаге просто ищем множество ребер, пересекающихся с разделяющей плоскостью (рис.6-а). Эта операция выполняется за O(E), где E – это общее число ребер в решетке поверхности. На следующем шаге нужно выкинуть из полученного множества висячие вершины, тогда в результате мы получим замкнутый цикл. Далее список сортируется в порядке обхода цикла, эту операцию очень просто выполнить за O(E'logE'), где E' – это количество ребер в цикле, с помощью обычного обхода в глубину на отсортированном по первой вершине и расставленными ссылками списке. На последнем шаге происходит «сглаживание цикла». Этот этап заключается в том, что каждые два последовательных ребра в отсортированном цикле, принадлежащие одному треугольнику исходной решетки, заменяются на третье ребро этого треугольника. Таким образом, за один проход каждое ребро в списке будет принадлежать только одному примитиву.

3) Непосредственное разбиение на «левую» и «правую» части

Осталось всего лишь разделить исходное множество ребер поверхности на две части относительно разбивающего цикла. Пометим вершины разбивающего пути (рис.6). Теперь, запустив обход графа в глубину из любой вершины решетки, не принадлежащей срезу, мы разобьем исходную сетку на две компоненты связности, если в обходе в глубину не будем идти дальше помеченных вершин.

4) Триангуляция среза

После того как поверхность разбита на 2 части, нужно триангулировать срез, который после разбиения представляет собой неплоский многоугольник. Для этого все точки среза проектируются на отсекающую плоскость, далее внутрь многоугольника, в зависимости от требуемого разрешения, помещается некоторое число дополнительных точек, находящихся в случайных местах и полученное множество разбивается на треугольники с помощью триангуляции Делоне. Затем точки отображаются обратно на «левую» и «правую» части и мы получаем 2 новых поверхности, готовых к разбиению.

Эти 4 этапа выполняются до тех пор, пока не будет получена нужная глубина дерева разбиений, которая напрямую связанна с количеством используемых процессоров.

5) Параллельное разбиение на тетраэдры

Заключительным и самым трудоемким этапом работы генератора является разбиение внутренней области поверхности на тетраэдры. Разбиение выполняется параллельно на всех процессорах, за счет чего достигается значительное ускорение. За основной метод разбиения берется разбиение, удовлетворяющее свойствам триангуляции Делоне в 3-х мерном случае. Существует множество решений, реализующих разбиение области, ограниченной решеткой поверхности, на тетраэдры. Одним из них является свободная библиотека TetGen[4], которая использовалась при реализации параллельного генератора, предложенного в данной работе.

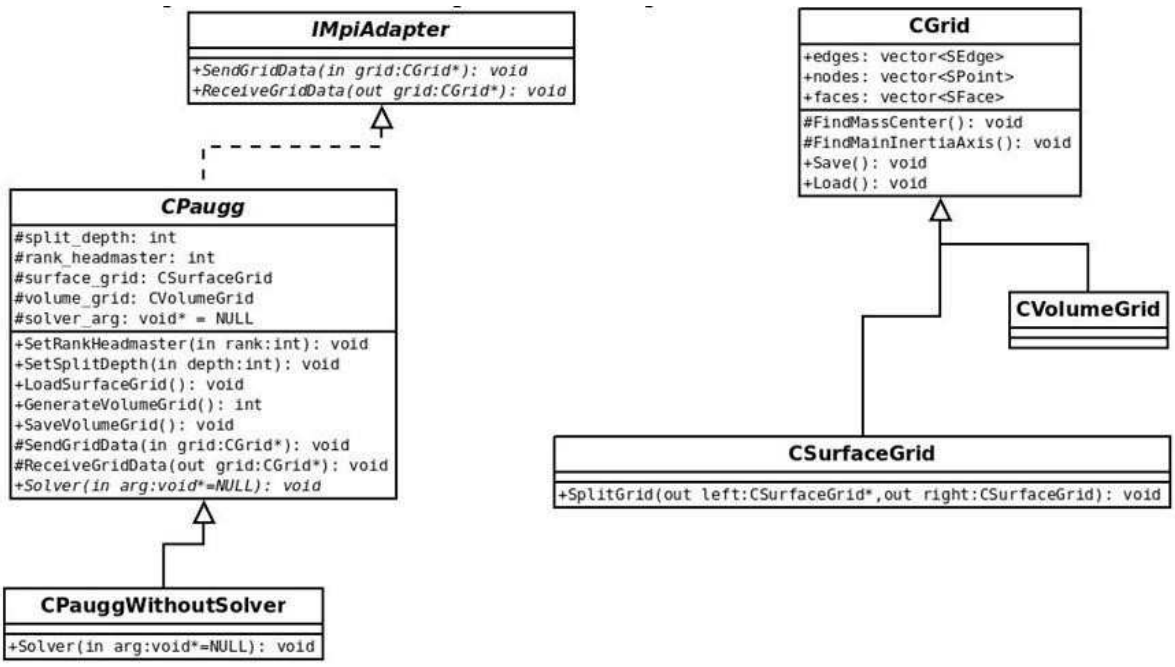

Рис.2. UML диаграмма классов библиотеки PAUGG

Одним из поставленных критериев выступала адаптивность реализуемого генератора, поэтому оптимальным был выбран способ представления не в виде отдельной программы, а в форме библиотеки классов (далее PAUGG – Parallel Adaptive Unstructured Grids Generator), позволяющих при несложном подключении и настройке иметь возможность использования параллельной генерации неструктурированных сеток. Для некоторых частей библиотеки, таких как нахождение собственных векторов матрицы и разбиение на тетраэдры с сохранением свойств триангуляции Делоне, были использованы готовые решения: в первом случае свободная

библиотека LAPACK (Linear Algebra Package), во втором свободная библиотека TetGen. На Рис.2 представлена UML диаграмма классов реализованной библиотеки PAUGG.

Основная часть работы возлагается на класс CPaugg, который обеспечивает высокоуровневую работу по генерации неструктурированной решетки. Распараллеливание реализовано прозрачно, то есть чтобы пользователь не задумывался об использовании дополнительного MPI кода.

Для корректного запуска генератора, требуется в каждом процессе создать объект класса CPauggWithoutSolver, загрузить исходную решетку методом LoadSurfaceGrid, а затем всего лишь вызвать метод GenerateVolumeGrid(). После этого решетку можно сохранить с помощью метода SaveVolumeGrid().

Можно воспользоваться альтернативным подходом и написать собственный класс-наследник CPaugg, в котором переопределить функцию Solver(void *arg). Это позволит сделать нужные расчеты на ходу, то есть параллельно на процессорах-листьях дерева разбиения. Таким образом, можно еще больше ускорить процесс вычислений. Но в этом способе придется внимательно подойти к реализации метода конечных элементов, чтобы корректно учесть граничные условия – правильно «склеить» и интерполировать модель.

Для анализа полученных результатов было принято решение использовать 8-ядерную машину. В качестве тестовой модели [6] была выбрана модель сердца с числом узлов примерно 700000. За показатель эффективности принималось значение ускорения T1 / TN, где T1 – время выполнения генерации решетки на 1 процессоре, а TN - время генерации решетки для тех же исходных данных, но на N процессорах с использованием предложенного метода распараллеливания. Результаты эксперимента представлены на Рис.3а. Характер графика свидетельствует о том что, с ростом числа процессоров время генерации уменьшается даже в большее число раз, чем увеличивается количество процессов. Это можно объяснить тем, что затраты на передачу данных и непосредственно сам процесс разбиения поверхности с большим (порядка 106) числом узлов становятся малы, по сравнению со временем, затраченным на разбиение внутренней области на тетраэдры.

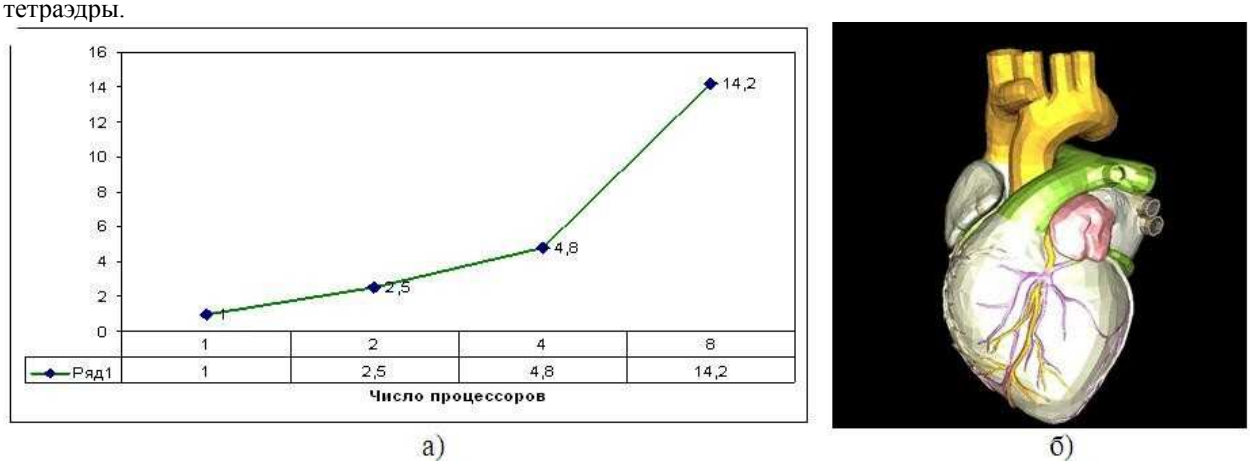

Рис.3.а) Зависимость показателя эффективности от числа процессоров; б) Визуализация построенной модели сердца[6].

Организация данных на системе с распределенной памятью.

Такого рода организация данных естественным образом снижает производительность любого алгоритма, её использующего. Но в виду того, что альтернативного подхода применить не имеется возможности (количество данных огромное — более 10^9 примитивов), рассматриваемая концепция является достаточно полезной.

Одним из наиболее приоритетных решений для подобного рода задач является использование высокопроизводительных структур данных, таких как деревья поиска. За основу берется то, что всю информацию, какой бы природы она ни была, возможно упорядочить посредством задания определенного ключа, по которому в дальнейшем будет производиться поиск.

Производительность предлагаемого далее метода напрямую зависит от исходной задачи. Это связанно со степенью упорядоченности данных, то есть с тем, какое число элементов структуры соответствует одному ключу. Естественно, реализация подобного дерева для абстрактной задачи будет предполагать неэффективный поиск по ключу исходных данных. Если же рассматривать конкретную специфику задачи, то можно достичь лучших результатов.

По отношению к генерации неструктурированных решеток такого рода подход более чем приемлем. Данный представлены в хорошо упорядочиваемом виде (используются координаты узлов и соответствующие им номера вершин для хранения графа), поэтому организация дерева поиска напрашивается сама по себе.

Предлагается использовать декартово дерево. Будем сопоставлять каждой вершине ключ и случайную высоту, а также ссылки на левого и правого потомков в виде пар <номер процесса; локальный идентификатор на процессе>. Главный процесс будет выступать в виде корневой вершины. Для получения нужных данных «заинтересованный» процесс будет обращаться к главному узлу. Поиск в декартовом дереве выполняется за O(logN), где N – это число элементов, хранящихся в структуре, таким образом, будет проделано порядка

O(logN) взаимодействий между процессорами через MPI. Для ускорения работы межпроцессного взаимодействия предлагается использовать односторонние операции в MPI-2. Перестроение такого дерева выполняется также за O(logN) элементарных операций взаимодействия узлов. Организация исходного дерева требует O(NlogN) операций.

Таким образом для представленной выше задачи построения неструктурированных сеток становится возможным хранение базы вершин в виде подобной структуры. Важно заметить, что неотъемлемой частью представленного менеджера памяти является реализация системы транзакций для безопасного поиска и изменения дерева.

В работе предложена общая схема параллельной генерации неструктурированных решеток на основе метода разбиения пространственной области на тетраэдры с сохранением свойств 3-мерной триангуляции Делоне. Выполнена реализация библиотеки, осуществляющей параллельную генерацию неструктурированных сеток на многопроцессорной системе. Получены и проанализированы результаты и эффективность предложенного подхода по сравнению с последовательным алгоритмом, реализующим данный метод. На основании полученных результатов можно заключить, что параллельная генерация нерегулярной решетки, с помощью метода декомпозиции на домены, может значительно ускорить процесс визуализации моделей с большим числом узлов. В качестве развития данного подхода предполагается усовершенствование предложенного метода на случай решеток с числом узлов порядка 10^9. В этом случае для реализации на системе с распределенной памятью придется рассмотреть возможность применения распределенного хранения данных о сетке.

Данная работа выполнена при поддержке Федеральной целевой программы "Научные и научно-педагогические кадры инновационной России" на 2009 - 2013 годы и грантов РФФИ 08-07-00445-а, 09-07-12068-офи_м.

ЛИТЕРАТУРА:

1. E.G. Ivanov, H. Andrä, A.N. Kudryavtsev, “Automatic Parallel Generation of Tetrahedral Grids by Using a Domain Decomposition Approach”, Proceedings of the Conference on “Numerical Geometry, Grid Generation and High Performance Computing”, Moscow, Russia, July 4-7, 2006, pp.115-124.
2. Б. Н. Четверушкин, Н. Н. Шабров, М. В. Якобовский «Параллельные алгоритмы построения изоповерхностей на больших сетках» // ПАВТ 2010, Россия, Уфа
3. Steven P. Callahan «Progressive volume rendering of large unstructured grids» // IEEE Transactions on Visualization and Computer Graphics, pp. 1307-1314 , NJ, USA, 2006
4. Библиотека TetGen: http://tetgen.berlios.de
5. БиблиотекаLAPACK – Linear Algebra PACKage: http://www.netlib.org/lapack/
6. База данных группы INRIA Gamma team: http://www-roc.inria.fr/gamma/