УДК 532.529

БАЛАНСИРОВКА НАГРУЗКИ ПРОЦЕССОРОВ ПРИ РЕШЕНИИ КРАЕВЫХ ЗАДАЧ МЕХАНИКИ ЖИДКОСТИ И ГАЗА СЕТОЧНЫМИ МЕТОДАМИ

К. Н. Волков[1]

Численное решение задач механики жидкости и газа на многопроцессорных вычислительных системах состоит в геометрической декомпозиции расчетной области, обработке каждым процессором своей подобласти и коммуникациях между процессорами для получения полного решения. Сбалансированность нагрузки процессоров определяется равномерностью распределения сетки по процессорам и затратами на передачу данных между ними. Объем передачи данных между процессорами зависит от числа связей между подобластями, распределенными по процессорам. Обсуждаются подходы к обеспечению статической и динамической балансировки нагрузки процессоров при решении задач механики жидкости и газа на многопроцессорных вычислительных системах. Приводится описание различных этапов и методов статической (методы половинного деления, комбинаторные методы, комбинированные подходы) и динамической (диффузный алгоритм, метод потенциала, многоуровневые подходы) балансировки, а также сравниваются показатели их эффективности. Проводится сравнение диффузного метода и метода потенциала для области простой геометрической конфигурации при решении задачи на адаптивной сетке.

Ключевые слова: параллельный алгоритм, балансировка нагрузки, декомпозиция, сетка, механика жидкости и газа.

1. Введение. Многие задачи механики жидкости и газа требуют для своего решения ресурсов, имеющихся в распоряжении высокопроизводительных вычислительных систем (High Performance Computing, HPC). В задачах механики жидкости и газа повышенные требования к производительности и памяти обусловливаются сложными нелинейными моделями среды, описываемыми дифференциальными уравнениями в частных производных (уравнения Эйлера или Навье–Стокса), пространственным характером задачи и нестационарностью протекающих процессов [1].

При параллельном решении задачи выделяют несколько этапов. На этапе 1 (декомпозиция, partitioning) производится разбиение задачи на минимальные независимые подзадачи. Этап 2 (коммуникации, communication) связан с определением информационных связей между подзадачами. На этапе 3 (объединение, agglomeration) производится объединение подзадач в группы для минимизации информационных связей между ними. На этапе 4 (отображение, mapping) группы подзадач назначаются конкретным процессорам.

При моделировании газодинамических процессов широко используются сеточные методы, основанные на разбиении области на ячейки (контрольные объемы) и выполнении ряда вычислений по обработке данных, связанных с ячейками сетки. На каждой итерации происходит моделирование ряда физических показателей в текущий момент времени. Каждый из физических показателей моделируется при помощи процедуры с явными или неявными зависимостями. В случае явных схем метод расчета в каждой из ячеек основан на получении результатов расчетов в предыдущий момент времени от смежных ячеек и выполнении вычислений по обработке текущей и смежных ячеек. Для неявных схем используется метод расщепления по координатным направлениям. Вычислительная нагрузка, связанная с обработкой различных ячеек, отличается и изменяется во времени.

Для распараллеливания вычислений расчетная область разделяется на несколько подобластей по числу процессоров [2]. При использовании регулярных сеток, топологически эквивалентных индексному прямоугольнику или параллелепипеду, задача декомпозиции решается разбиением области на заданное число подобластей, в каждой из которых содержится одинаковое число узлов, плоскостями, перпендикулярными индексным плоскостям. Подобласти имеют ряд фиктивных ячеек (ghost cell), которые перекрываются с ячейками соседних подобластей и хранят граничные значения соседних блоков (рис. 1). Шаблон

[1] Балтийский государственный технический университет “Военмех” им. Д. Ф. Устинова, физико-механический факультет, 1-я Красноармейская ул., 1, 190005, Санкт-Петербург; доцент, e-mail: dsci@mail.ru

разностной схемы определяет, какие и сколько внешних узлов оказываются необходимыми для получения корректного решения. В случае трехточечного шаблона для сшивки решений используются функции из одного, а в случае пятиточечного — из двух соседних узлов с каждой стороны. Каждая подобласть обрабатывается одним процессором, а обмен данными между процессорами требуется только при переходе к следующему временно́му слою. Связь подобластей осуществляется при помощи копирования значений искомых функций в фиктивные ячейки [3].

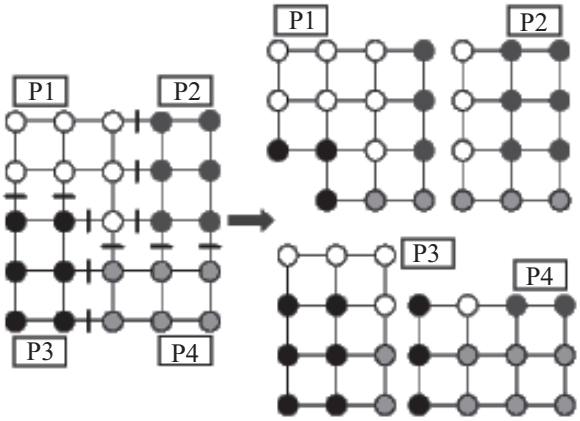


Рис. 1. Геометрическая декомпозиция расчетной области (4 процессора)

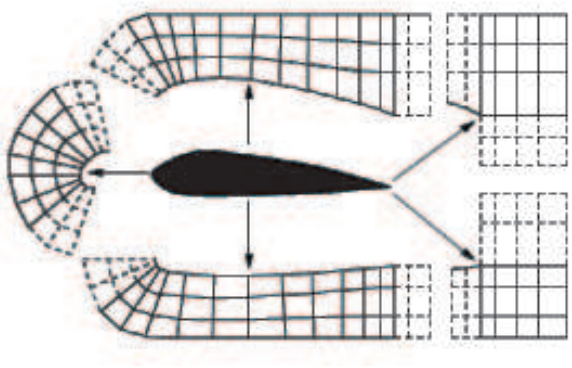


Рис. 2. Пример декомпозиции расчетной области (5 процессоров)

Пример геометрической декомпозиции расчетной области для задачи обтекания профиля приводится на рис. 2 (используется сетка типа C). Пунктирные линии соответствуют ячейкам сетки, через которые осуществляются коммуникации между процессорами (используется два слоя фиктивных ячеек).

Задача рационального распределения по процессорам нерегулярных сеток сводится к разбиению на компактные подобласти графа, веса вершин которого отражают объемы вычислений в различных узлах сетки, а веса ребер — объемы обменов данными, передаваемых в процессе счета между соответствующими узлами.

К параллельным вычислительным алгоритмам и их программной реализации предъявляется ряд требований, связанных с обеспечением изменяемого поведения математической модели и адаптивной точности вычислений [1, 2]. На итоговую производительность одновременно оказывают влияние распределенная вычислительная нагрузка и коммуникационные затраты между процессорами. Важной проблемой является оптимальное отображение узлов сетки на параллельную вычислительную систему.

Распределенное приложение представляет собой совокупность логических процессов, которые взаимодействуют друг с другом, посылая сообщения. Логические процессы распределяются по разным вычислительным узлам и функционируют параллельно. При выполнении распределенного приложения возникает конфликт между сбалансированным распределением объектов по процессорам и низкой скоростью обменов сообщениями между процессорами. При распределении логических процессов между процессорами таким образом, чтобы издержки на коммуникации между ними сводились к нулю, некоторые процессоры простаивают, в то время как другие являются перегруженными. Под балансировкой нагрузки (load balancing) подразумевается такое распределение подзадач по процессорам, которое обеспечивает примерно равную вычислительную загрузку процессоров и минимальные затраты на передачу данных между ними.

Статическая балансировка (static balancing) выполняется до начала выполнения распределенного приложения. Для статической балансировки нагрузки требуется равномерное распределение вычислительной нагрузки между процессорами (подобласти имеют равное число узлов, минимальное число общих граней, минимальную связность, минимальную ширину матрицы системы разностных уравнений, обрабатываемых каждым процессором, оптимальную обусловленность локальных матриц). В ряде случаев требуется обеспечить соответствие топологии и размеров подобластей возможностям сеточных генераторов.

Динамическая балансировка (dynamic balancing) предусматривает перераспределение вычислительной нагрузки на узлы во время выполнения приложения. Программное обеспечение, реализующее динамическую балансировку, определяет нагрузку вычислительных узлов, пропускную способность линий связи, а также частоту обменов сообщениями между логическими процессами распределенного приложения. На

основе собранных данных принимается решение о том, на каком вычислительном узле следует выполнять вычисления, связанные с новым заданием, а также о переносе логических процессов с наиболее загруженных на менее загруженные узлы (migration). Для динамической балансировки используются как методы статической балансировки, применяемые на каждом шаге по времени [4], так и специальные подходы, а для повышения их эффективности — параллельные версии последовательных алгоритмов [5, 6]. Обзор и сравнение различных методов динамической балансировки дается в работе [7].

Во многих приложениях нагрузка каждого процессора остается неизменной в течение определенного числа итераций, поэтому балансировка нагрузки требуется только после перестроения сетки (квазидинамическая балансировка, quasi-dynamic balancing).

Основная трудность при использовании динамической балансировки в разработанных комплексах параллельных программ связана с затратами на доработку существующих кодов. Методы статической балансировки сравнительно просто внедряются в существующие программные системы. В моменты разбалансировки системы работа текущего алгоритма счета останавливается, данные сбрасываются в специальные хранилища, а затем заново загружаются с учетом балансировки.

В случае неоднородной вычислительной нагрузки использование методов динамической балансировки нагрузки процессоров позволяет снизить время простоя отдельных процессоров и повысить эффективность распараллеливания. Методы динамической балансировки классифицируются по времени обмена информацией о текущей нагрузке между процессорами (процессоры обмениваются информацией периодически или при нагрузке меньше или больше допустимой), по месту принятия решения о перераспределении нагрузки (централизованный или децентрализованный), по времени перераспределения нагрузки (синхронный или асинхронный), а также в соответствии с каким подходом производится перераспределение нагрузки (детерминированный, стохастический, по инициативе получателя или отправителя).

В настоящей статье рассматриваются методы статической и динамической балансировки нагрузки процессоров (геометрические методы, методы разделения графов, многоуровневые подходы), проводится анализ их функциональных возможностей, а также обсуждаются и сравниваются показатели эффективности некоторых из подходов.

2. Стратегия балансировки. При использовании адаптивных сеток, число узлов которых изменяется в процессе решения, требуется динамическая балансировка нагрузки для того, чтобы сохранить эффективность параллельного алгоритма [8]. Адаптивные методы представляют собой итерационные алгоритмы, использующие полученное приближенное решение для повышения точности последующих приближений. Вычисления на адаптивных неструктурированных сетках относятся к классу задач, имеющих нерегулярный доступ к данным, а также низкую пространственную и временну́ю локализацию обращений к памяти. Специфические особенности адаптивных алгоритмов приводят к возникновению дисбаланса вычислительной нагрузки, приходящейся на один процессор.

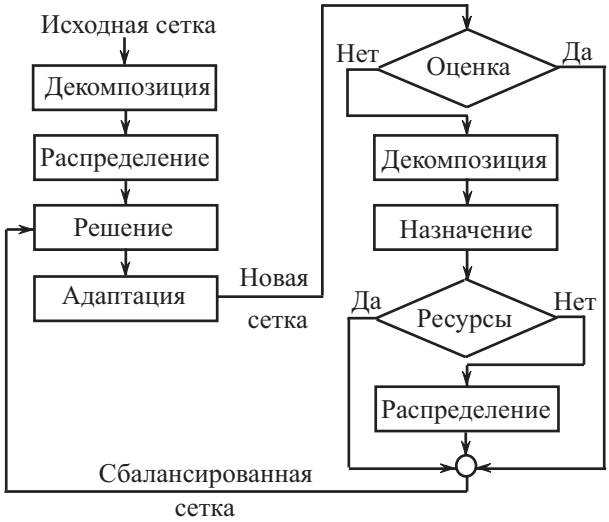


Рис. 3. Решение задачи на адаптивной сетке

Добавление новых узлов сетки изменяет характерный размер и число ячеек в соответствии с локальными критериями перестроения, основанными на оценке погрешности решения [9]. Локальное перестроение сеточной модели с увеличением числа ячеек приводит к разбалансировке вычислительной нагрузки, для устранения которой требуется перераспределять сеточные данные между процессорами динамическим образом. В связи с этим, обеспечение балансировки нагрузки процессоров является одной из наиболее важных характеристик параллельных методов декомпозиции [10].

Блок-схема решения задачи на адаптивной сетке с применением параллельных вычислительных технологий приводится на рис. 3. Для решения задачи требуется вычислительный модуль (flow solver) и сеточный модуль, обеспечивающий адаптацию сетки к решению (mesh adaptor). Предполагается, что, в отличие от вычислительного модуля, процедура адаптации сетки не требует балансировки. Компонентами сеточного модуля являются подмодуль, обеспечивающий разделение сетки по процессорам (partitioning), и подмодуль, который назначает группы узлов сетки процессорам (mapping). После работы вычислительного модуля и получения решения задачи на нескольких итерациях (flow solution) и оценки по-

грешности решения производится вызов сеточного модуля и построение новой сетки или сгущение узлов существующей сетки (adaption). После проверки условия сбалансированности нагрузки процессоров управление либо передается вычислительному модулю, либо производится разделение новой сетки на подобласти (repartitioning). При необходимости производится назначение новых подзадач процессорам вычислительной системы таким образом, чтобы минимизировать число коммуникаций между процессорами (reassignment). Оценка стоимости коммуникаций позволяет использовать имеющееся отображение сетки на структуру многопроцессорной системы и передать управление вычислительному модулю (новое распределение данных оказывается дорогим с точки зрения числа обменов данными) или провести новое распределение подзадач между процессорами (remapping), если это не требует большого числа обменов данными.

Примером задачи, требующей для своего эффективного решения динамической балансировки, является моделирование горения предварительно неперемешанного топлива (non-premixed combustion). Для решения задачи обычно используется метод расщепления по физическим процессам. Уравнения газовой динамики описывают диффузию и перенос компонентов газовой смеси, а уравнения химической кинетики — химические процессы, протекающие в расчетной области. Узлы сетки, в которых протекают интенсивные химические реакции, локализуются на фронте пламени и непосредственно за ним. Время обработки узлов, расположенных вблизи фронта пламени, существенно больше, чем узлов, соответствующих области, удаленной от фронта пламени. Положение фронта пламени меняется с течением времени, что приводит к миграции соответствующих узлов между процессорами. Скорость химической реакции зависит от температуры (при низкой температуре скорость реакции близка к нулю), поэтому время решения уравнений химической кинетики изменяется от узла к узлу в пространстве и на разных шагах по времени.

При моделировании газодинамических процессов с увеличением числа процессоров число обменов между ними возрастает, приводя к снижению эффективности расчета. Начиная с некоторого числа процессоров, добавление новых процессоров не приводит к росту производительности (эффект насыщения). При решении уравнения химической кинетики расчеты проводятся локально в каждом узле, а необходимости обмена данными между процессорами нет. Причины ограничения производительности, связанные с обменами данными между обрабатывающими процессорами, отсутствуют. Эффективность расчета снижается из-за неравномерной нагрузки процессоров, вызванной различием времени счета в различных узлах и из-за затрат на перераспределение узлов.

В механике твердого тела применение динамической балансировки нагрузки обусловливается использованием h-версий (добавление или удаление элементов) и p-версий (различные степени аппроксимации для отдельных элементов сетки) метода конечных элементов.

Решение задачи балансировки нагрузки состоит из оценки нагрузки вычислительных узлов, инициации балансировки нагрузки, принятии решения о балансировке и перемещении объектов.

На этапе 1 осуществляется приблизительная оценка нагрузки каждого процессора (load evaluation). Полученная информация о нагрузке используется для определения возникновения дисбаланса и нового распределения объектов путем вычисления объема работ, необходимого для их перемещения.

На этапе 2 определяется момент возникновения дисбаланса нагрузки и степень необходимости балансировки путем сравнения возможной пользы от ее проведения (profitability). При синхронном определении дисбаланса все процессоры прерывают работу в определенные моменты синхронизации и определяют дисбаланс нагрузки путем сравнения нагрузки отдельного процессора с общей средней нагрузкой. При асинхронном определении дисбаланса каждый процессор хранит историю своей нагрузки. В этом случае момент синхронизации для определения степени дисбаланса отсутствует. Вычислением объема дисбаланса занимается фоновый процесс, работающий параллельно с приложением.

Стратегия динамической балансировки нагрузки (этап 3) относится к классу централизованных или полностью распределенных (task mapping decision). При централизованной стратегии специальный процессор собирает глобальную информацию о состоянии вычислительной системы и принимает решение о перемещении задач для каждого из процессоров. Количество и объем пересылок между управляющим и рабочими процессорами достаточно велики, что в существенной степени снижает эффективность распараллеливания. При полностью распределенной стратегии на каждом процессоре выполняется алгоритм балансировки нагрузки. Перемещение происходит только между соседними процессорами. Каждый процессор обрабатывает некоторое множество локальных узлов, а затем запрашивает по одному узлу для обработки другими процессорами. Основной недостаток заключается в большом числе сравнительно коротких транзакций, что оказывается неэффективным для систем с высокой латентностью каналов передачи данных.

После принятия решений о балансировке происходит перемещение объектов среди процессоров для достижения баланса нагрузки (этап 4, task migration). При перемещении объекта обеспечивается целостность его состояния.

Для характеристики нагрузки процессоров в целом и системы используется средняя нагрузка, максимальная нагрузка и дисбаланс нагрузки: $\mu = \frac{1}{p}\sum_{i=1}^{p} l_i,\ l_{\max} = \max\{l_1, \ldots, l_p\},\ \sigma = \frac{1}{\mu}\left[\frac{1}{p}\sum_{i=1}^{p}(l_i - \mu)^2\right]^{1/2}$, где l_i — нагрузка процессора i, p — число процессоров.

Если принять, что на каждом из процессоров p_i хранится некоторое число заданий n_i, а время выполнения каждого из заданий равняется t_i^j, то получим оценки минимального и максимального времени обработки заданий на p процессорах. Минимальное время достигается при идеальном (равномерном) распределении вычислительной нагрузки по доступным процессорам: $t_{\min} = \frac{1}{p}\sum_{i=0}^{p-1}\sum_{j=0}^{n_i-1} t_i^j$.

Максимальное время достигается при отсутствии перераспределения заданий между процессорами в ходе вычислений, при этом каждый процессор обрабатывает те узлы, которые на нем хранятся в начале расчета $t_{\max} = \max\limits_i \sum_{j=0}^{n_i-1} t_i^j$.

Качество балансировки зависит как от топологии вычислительной системы, так и от способа программирования. Если принять, что объем вычислительной работы каждого процессора пропорционален числу узлов подобласти, то для балансировки нагрузки процессоров требуется, чтобы число узлов в каждой подобласти было приблизительно одинаковым. Число коммуникаций между процессорами (communication cost) зависит от порядка разностной схемы и числа узлов, вовлеченных в разностный шаблон. На параллельных компьютерах с распределенной памятью отношение времени доступа к удаленной памяти (в соседнем процессоре) ко времени доступа к локальной памяти (в данном процессоре) изменяется от 10 до 1000, что приводит к необходимости минимизации пересылок данных между различными процессорами. Во многих случаях требование максимальной эффективности вычислительного алгоритма приводит к тому, что реализация последовательного и параллельного кода осуществляется по-разному. При этом последовательный и параллельный коды не являются эквивалентными в математическом смысле, что приводит к различным картинам изменения невязки в зависимости от числа итераций (в основном, это относится к алгоритмам, использующим неявные схемы). В результате скорость сходимости (convergence speed) последовательного и параллельного алгоритмов оказывается различной.

Для явных конечно-разностных схем и реализующих их алгоритмов балансировка нагрузки является более важной задачей, чем минимизация коммуникаций, в то время как для неявных разностных схем ситуация оказывается обратной [5].

3. Теория графов. Во многих задачах, связанных с балансировкой нагрузки процессоров, используются методы теории графов (graph theory). Связи контрольных объемов или узлов сетки представляются в виде ненаправленного графа. В качестве вершин графа берутся узлы сетки, ребер графа — грани контрольного объема. Вершины графа определяют вычислительную нагрузку, ребра графа — коммуникационную.

Граф характеризуется множеством вершин V (vertex set), представляющих собой список индексов (list of indices) $V = \{1, 2, 3, 4, \ldots\}$, и множеством ребер (edge set) $E = \{(1, 2), (3, 4), \ldots\}$, каждое из которых состоит из двух узлов $(i, j) \in E$ (иногда для обозначения ребра графа используется обозначение $i \leftrightarrow j$). Число вершин и ребер графа обозначается через $|V|$ и $|E|$. Степень вершины (degree of vertex) представляет собой число ребер, связанных с данной вершиной, а степень графа (degree of graph) — наибольшую степень среди всех вершин графа. В общем случае ребрам графа приписываются некоторые числовые характеристики (взвешенный граф).

Различают ненаправленные (undirected graph) и направленные (directed graph) графы. Каждое ребро направленного графа характеризуется направлением, имея начальную (head) и конечную (tail) вершины. В ненаправленном графе все вершины являются равноправными. Граф называется связным (connected graph), если для любых двух вершин i и j имеется путь, связывающий эти вершины. Путь (path) определяется в виде списка вершин $\{i, i_1, i_2, \ldots, i_k, j\}$, в котором каждые две последовательные вершины формируют ребро. Расстояние (distance) между двумя вершинами графа представляет собой минимальное число ребер среди всех путей, связывающих эти вершины.

Для задания графов используются различные способы. При малом количестве ребер для определения графа используются списки, которые содержат имеющиеся в графах ребра. Представление достаточно

плотных графов, для которых почти все вершины соединяются между собой ребрами, осуществляется при помощи матрицы смежности $A = \{a_{ij}\}$ ($1 \leqslant i, j \leqslant n$, под n понимается число вершин). Ненулевые элементы матрицы смежности соответствуют ребрам графа: $a_{ij} = \begin{cases} w(v_i, v_j) & \text{при} \quad (v_j, v_j) \in R, \\ 0 & \text{при} \quad i = j, \\ -1 & \text{иначе.} \end{cases}$

Для обозначения отсутствия ребра между вершинами в матрице смежности на соответствующей позиции используется любое отрицательное число. Использование матрицы смежности позволяет применить при реализации вычислительной процедуры разделения графа матричные алгоритмы обработки данных.

Представление модели вычислений в виде графа позволяет сравнительно просто решить вопросы хранения обрабатываемых данных и предоставляет возможность применения типовых алгоритмов обработки графов [11] (во многих случаях задача обработки графа является предметом распараллеливания).

Для разбиения сетки и распределения подобластей между процессорами используется коммуникационный граф (communication graph), который характеризует зависимость узлов (node) и граней (edge) сетки. При использовании конечно-разностных, конечно-объемных или конечно-элементных алгоритмов на сетках, представляющих собой множество узлов (node based data structure), расчеты в каждом узле сетки требуют пересылки данных из соседних узлов сетки, связанных гранями с текущим узлом. В этом случае зависимость данных сеточной структуры переносится вдоль граней сетки, вершины коммуникационного графа являются узлами сетки, а ребра коммуникационного графа совпадают с гранями сетки. Для реализации вычислительных алгоритмов на сетках, которые описываются множеством ячеек (cell based data structure), для проведения расчетов в каждой ячейке сетки требуется пересылка данных из соседних ячеек, имеющих с данной ячейкой общие грани. В этом случае зависимость данных переносится вдоль граней сетки. Вершины коммуникационного графа связываются с центрами ячеек сетки, а две вершины графа формируют ребро, если соответствующие ячейки сетки имеют общую грань (дуальный граф, dual graph).

В сеточных задачах вычислительная нагрузка на процессор складывается из суммы нагрузок, связанных с каждой ячейкой сетки. Ребро графа, соединяющее два узла, определяет коммуникационную нагрузку. Процесс балансировки нагрузки представляет собой нахождение разделения дуального графа сетки со множеством вершин, каждая из которых связана с ребром. Вершины и ребра графа задаются с весами, что необходимо при декомпозиции сеточных моделей, в которых нагрузка на ячейку не является равномерной.

Число ребер коммуникационного графа, разрезаемых при разбиении области на подобласти (number of edges cut), определяет стоимость пересылок данных между различными подобластями и используется как критерий качества декомпозиции сетки.

Рассмотрим ненаправленный граф $G = (V, E)$, где V — множество вершин, а E — множество ребер. Для декомпозиции и обеспечения балансировки нагрузки процессоров необходимо найти такие подмножества V_1 и V_2, что $V = V_1 \cup V_2$ и $V_1 \cap V_2 = 0$, причем $|V_1| \simeq |V_2|$, если $|V_2|$ — четное, и $|V_1| \simeq |V_2| - 1$, если $|V_2|$ — нечетное. Требуется минимизировать число ребер $|E_c|$ множества E, разрезаемых при разбиении расчетной области на подобласти $|E_c| = \left|\{h : h \in E,\, h = (v_1, v_2),\, v_1 \in V_1,\, v_2 \in V_2\}\right| \to \min$, где $v_i = \{x_i, y_i, z_i\}$. Условие балансировки нагрузки процессоров приводит к минимизации числа ребер, по которым происходит соприкосновение соседних подобластей.

Оптимальным считается разбиение графа на подграфы, при котором выравнивается суммарный вес вершин в подграфах и минимизируется суммарный вес разрезанных ребер между подграфами. В этой модели суммарный вес вершин в подграфах отвечает за равномерность разбиения сетки на подобласти (равномерность распределения подзадач по процессорам), а суммарный вес разрезанных ребер — за коммуникационную нагрузку между процессорами.

Задача разделения графа относится к классу NP-полных задач. Для решения задачи широкое применение находят эвристические подходы, многие из которых реализованы в различных библиотеках декомпозиции графов (METIS, JOSTLE и др.).

Одно из требований к методам балансировки состоит в обеспечении инкрементности (incremental) алгоритма, когда малое изменение задачи приводит к малым изменениям декомпозиции расчетной области [7]. Свойство инкрементности достигается явным образом (explicitly), накладывая определенные ограничения на перемещение данных, или неявным образом (implicitly).

4. Методы решения задачи. Задача оптимального разделения взвешенного ненаправленного графа состоит в разбиении его вершин на непересекающиеся подмножества с максимально близкими суммарными весами вершин и минимальным суммарным весом ребер, проходящими между полученными

подмножествами вершин. Указанные критерии разбиения графа являются противоречивыми, поскольку равновесность подмножеств вершин не соответствует минимальности весов граничных ребер и наоборот. Во многих случаях необходимым является выбор того или иного компромиссного решения. При невысокой доли коммуникаций оказывается эффективной оптимизация веса ребер только среди решений, обеспечивающих оптимальное разбиение множества вершин по весу.

Геометрические методы (geometric method) выполняют разбиение графа, основываясь на информации о координатах узлов сетки, и не принимают во внимание информацию о связности контрольных объемов, а потому не приводят к минимизации суммарного веса граничных ребер. Для минимизации межпроцессорных коммуникаций геометрические методы оптимизируют некоторые вспомогательные показатели (например, длину границы между разделенными участками сетки). Обычно геометрические методы не требуют большого объема вычислений, но по качеству разбиения уступают методам, учитывающим связность контрольных объемов.

Простейшими алгоритмами решения сформулированной задачи являются методы половинного деления (bisection method) [12, 13], которые при произвольном числе процессоров p и произвольном числе узлов сетки (вершин коммуникационного графа) $n = |V|$ используются рекурсивным образом [14] (при числе процессоров, не являющемся степенью двойки, метод половинного деления приводит к подобластям различного размера).

К недостаткам геометрических подходов относится учет только одной размерности области на каждой итерации. Подходы, учитывающие несколько размерностей, обеспечивают лучшее разбиение области.

Работа комбинаторных (combinatorial) или топологических (topological) подходов основывается на использовании графа, соответствующего конечно-разностной сетке. В отличие от геометрических подходов, комбинаторные методы не принимают во внимание информацию о близости расположения узлов друг относительно друга, руководствуясь только смежностью вершин графа. Комбинаторные методы обеспечивают более сбалансированное разбиение и меньшее информационное взаимодействие полученных подобластей.

Методы разбиения различаются точностью решения (точность оценивается в смысле близости получаемого решения к оптимальному варианту разбиения), временем выполнения и возможностями для распараллеливания. Для решения задачи балансировки находят применение эвристические алгоритмы, не имеющие строгого обоснования, но дающие приемлемое решение задачи в большинстве практически значимых случаев. Выбор подходящего алгоритма является достаточно сложной и не очевидной задачей.

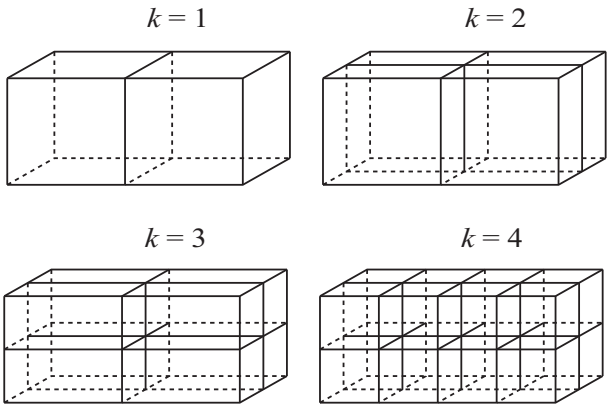


Рис. 4. Декомпозиция расчетной области при помощи метода рекурсивного деления пополам (4 итерации)

5. Геометрические алгоритмы. Поскольку вычислительные алгоритмы имеют дело с обработкой узлов сетки, наиболее простыми методами балансировки являются геометрические методы, в основе которых лежит использование координат узлов сетки в физическом пространстве. Недостаток геометрических методов состоит в том, что они не являются инвариантными к возмущениям узлов сетки. При изменении координат узлов сетки декомпозиция расчетной области проводится заново.

5.1. Методы половинного деления. Среди методов данной группы наиболее простым является метод рекурсивного деления пополам (Recursive Bisection, RB) [5, 6]. Исходная область разделяется на две подобласти в направлении наиболее протяженной стороны. Узлы сортируются в порядке возрастания координат относительно серединного узла. Половина узлов присваивается одной подобласти, половина — другой. Разбиение повторяется для каждой из подобластей. За k итераций получается 2^k подобластей (рис. 4). Метод применим в случае 2^n процессоров (для вычислительной системы с топологией гиперкуба).

Методы статической балансировки, основанные на методе рекурсивного деления пополам, различаются способом выбора метрики $d(v_i, v_j)$.

В более общем случае для декомпозиции области вместо линий и плоскостей используются круги и сферы [15]. Часть области, заключенная внутри сферы, соответствует одной подобласти, а область, являющаяся внешней по отношению к сфере, другой. Качество декомпозиции оказывается сравнимым с тем, которое дают методы разделения графов [16].

5.2. Метод координатного деления пополам. В методе рекурсивного деления по координатным

направлениям (Recursive Coordinate Bisection, RCB) для декомпозиции расчетной области используются координаты узлов сетки в физическом пространстве [17]. Расчетная область сначала делится в самом протяженном координатном направлении, а все узлы сетки сортируются по возрастанию относительно координаты серединного узла [18] (разбиение производится по линии, перпендикулярной одной из координатных осей). Половина узлов сетки присваивается одной подобласти, половина — другой. После этого процедура деления пополам повторяется нужное количество раз. Для разбиения графа на k частей требуется $\log_2 k$ уровней рекурсии и выполнение $k-1$ делений пополам. В случае, когда требуемое количество разбиений не является степенью двойки, каждое деление пополам осуществляется в соответствующем соотношении.

Метод RCB работает достаточно быстро и приводит к хорошим результатам для областей сравнительно простой геометрической формы. В методе RCB не производится оптимизации производительности, в результате чего получаются подобласти, вытянутые в одном из координатных направлений, что приводит к увеличению числа коммуникаций между процессорами.

Для избежания появления областей вытянутой формы или несвязанных подобластей используется метод несбалансированного деления пополам (Unbalanced RB, URB) [19], в котором вместо деления узлов области на равные подмножества и для получения подобластей приблизительно равной протяженности выбирается декомпозиция, минимизирующая отношение сторон подобластей и деление исходного множества узлов на подмножества, содержащие nk/p и $n(p-k)/p$ узлов, где n — число узлов, p — число процессоров, $k \in \{1, 2, \ldots, p-1\}$. При подходящем выборе k метод приводит к подобластям с приблизительно равным числом узлов, но с лучшим отношением сторон каждой подобласти [19].

5.3. Метод инерциального деления пополам. Метод RCB является чувствительным к ориентации сетки. Например, простое вращение сетки приводит к различной декомпозиции области. Метод инерциальной бисекции (Recursive Inertial Bisection, RIB) является инвариантным к вращению области [20], а в его основе лежит нахождение главной оси инерции (principal inertial axis) коммуникационного графа. Каждому узлу графа присваивается условная единичная масса, и ищется главная ось инерции полученной системы [21] (минимизируется угловой момент при вращении графа). Деление пополам производится относительно оси, ортогональной главной оси инерции. Время работы оценивается как $O(n)$.

Пример применения метода инерциального деления пополам приводится на рис. 5 (число процессоров равняется 4). На шаге 1 находится главная ось инерции A1, а деление области производится относительно оси C1, ортогональной оси A1. Процессоры разделяются на два множества — $\{p_0, p_1\}$ и $\{p_2, p_3\}$. Процессоры p_0 и p_1 обмениваются вершинами графа с процессорами p_2 и p_3. На шаге 2 производится нахождение главной оси инерции области, соответствующей процессорам p_0 и p_1 (ось A2a) и процессорам p_2 и p_3 (ось A2b). Разделение графа производится вдоль осей C2a и C2b.

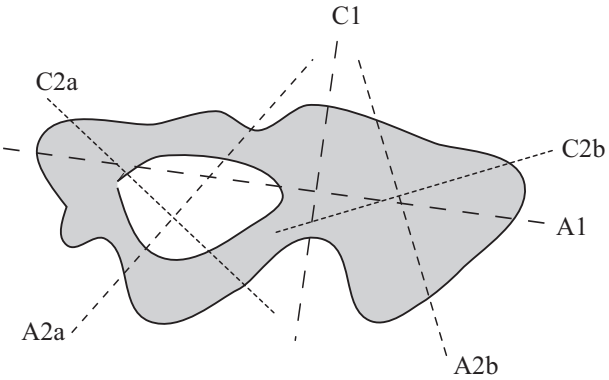


Рис. 5. Применение метода инерциального деления пополам

Рассмотрим граф, связанный с трехмерной сеткой и содержащий n вершин. Главная ось инерции описывается соотношением $\boldsymbol{y}_0 + \alpha\boldsymbol{y}$, где $\boldsymbol{y}_0$ — точка в пространстве, $\boldsymbol{y}$ — единичный вектор, α — вещественное число. Угловой момент узла $\boldsymbol{x}_i \in \mathbb{R}^3$, которому приписывается единичная масса, равняется квадрату расстояния от оси инерции:

$$\|\boldsymbol{x}_i - \boldsymbol{y}_0\|^2 - \frac{(\boldsymbol{x}_i - \boldsymbol{y}_0)'\boldsymbol{y}}{\|\boldsymbol{y}\|^2} = \|\boldsymbol{x}_i - \boldsymbol{y}_0\|^2 - (\boldsymbol{x}_i - \boldsymbol{y}_0)'\boldsymbol{y}, \quad i = 1, \ldots, n.$$

Для нахождения главной оси инерции находится минимум суммы $\sum\limits_{i=1}^{n}\bigl[\|\boldsymbol{x}_i - \boldsymbol{y}_0\|^2 - (\boldsymbol{x}_i - \boldsymbol{y}_0)'\boldsymbol{y}\bigr] \to \min$ при $\|\boldsymbol{y}\|^2 = 1$. Можно показать, что $\boldsymbol{y}_0 = \overline{\boldsymbol{x}} = \dfrac{1}{n}\sum\limits_{i=1}^{n}\boldsymbol{x}_i$, при этом $\boldsymbol{y}$ является собственным вектором, соответствующим наибольшему собственному значению матрицы инерции $I = \sum\limits_{i=1}^{n}\bigl(\boldsymbol{x}_i - \overline{\boldsymbol{x}}\bigr)\bigl(\boldsymbol{x}_i - \overline{\boldsymbol{x}}\bigr)'$. После нахождения собственного вектора каждый узел сортируется на основе координат его проекции на главную ось в соответствии с координатами $(\boldsymbol{x}_i - \overline{\boldsymbol{x}})'\boldsymbol{y}$ или $\boldsymbol{x}_i'\boldsymbol{y}$.

Основные вычислительные затраты алгоритма связаны с формированием матрицы инерции, что требует порядка $O(n)$ операций. Одно из достоинств алгоритма — простая параллелизация [10, 22].

6. Комбинаторные методы. Одна из причин низкого качества декомпозиции расчетной области при использовании геометрических методов состоит в том, что они не используют информацию о связности графа сетки.

6.1. Метод деления пополам графа. В методе рекурсивного деления пополам графа (Recursive Graph Bisection, RGB) в качестве объекта декомпозиции используется граф сетки, который считается связным [17]. Сначала находится диаметр графа, определяемый как наибольшее расстояние между вершинами графа. Расстояние между двумя вершинами графа определяется как наименьшее число граней, которые нужно пройти от узла i к узлу j. Начиная с одной из вершин (root), половина вершин присваивается одной из подобластей, а половина — другой подобласти. Затем рекурсивная процедура применяется к каждой из подобластей. Время работы оценивается как $O(n)$.

Для нахождения диаметра графа, начиная с произвольной вершины R, принятой за потенциальный корень, каждой соседней вершине присваивается метка 1, а соседям соседей присваивается метка 2. Последняя из помеченных вершин имеет метку m, которая имеет смысл расстояния от корневой вершины. Принимая эту вершину за корень, процесс повторяется. Когда m больше не увеличивается, вершина R считается корнем, а m является диаметром графа. На практике указанная процедура сходится за несколько итераций. Реализация алгоритма требует $O(n)$ операций.

6.2. Жадный алгоритм. Жадный алгоритм (greedy algorithm) заключается в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным.

Процедура начинается с вершины, имеющей наименьшую степень [23]. Маркируются все соседние вершины, а затем вершины, соседние с соседями. Первые n/p маркированных вершин присваиваются одной подобласти, и процедура применяется к оставшейся части графа до тех пор, пока все вершины не являются маркированными. Данный алгоритм имеет схожие черты с алгоритмом RGB, хотя и не является алгоритмом деления пополам. Стоимость алгоритма оценивается как $O(n)$.

6.3. Метод спектрального деления пополам. Метод спектрального деления пополам (Recursive Spectral Bisection, RSB) основан на использовании собственных векторов и собственных значений.

С каждой вершиной i графа соотносится условная масса $x_i = -1$ или $x_i = +1$ в зависимости от того, к какой половине исходной области относится вершина. Для обеспечения балансировки нагрузки процессоров минимизируется квадратичная форма [6, 17, 24]

$$|E_c| = \frac{1}{4} \sum_{\substack{i \leftrightarrow j, \\ i,j \in V}} (x_i - x_j)^2 \to \min, \tag{1}$$

где $i \leftrightarrow j$ обозначает ребро, соединяющее вершины i и j. Для сохранения балансировки нагрузки все подобласти имеют приблизительно равное число вершин, поэтому сумма всех весовых множителей, связанных с вершинами, равняется нулю:

$$\sum_{i=1}^{n} x_i = 0. \tag{2}$$

Задача сводится к минимизации квадратичной формы (1) при ограничениях (2), где $x_i = +1$ или $x_i = -1$. Решение сформулированной задачи целочисленного программирования с ограничениями представляется достаточно трудным при большом числе вершин (задача является NP-полной). Игнорируя целочисленные ограничения, минимизация квадратичной формы (1) приводит к задаче непрерывной оптимизации. Решением задачи является нулевой вектор. При $x_i = +1$ или $x_i = -1$ сумма квадратов равняется числу вершин, что дает ограничение

$$\sum_{i=1}^{n} x_i^2 = 0. \tag{3}$$

Для упрощения решения задачи целочисленные ограничения игнорируются, а вместо дискретной рассматривается решение непрерывной задачи. Перепишем квадратичную форму (1) в виде $|E_c| = \frac{1}{4}\boldsymbol{x}'L\boldsymbol{x}$, где $\boldsymbol{x} = \{x_i\}$ представляет собой вектор, составленный из весовых множителей, связанных с вершинами, а L — спектральная матрица графа (матрица Лапласа), имеющая размерность $n \times n$ (под n понимается

число вершин графа). Элементы матрицы Лапласа $L = \{l_{ij}\}_{i,j=1,\dots,n}$ имеют вид

$$L_{ij} = \begin{cases} -1 & \text{при} \quad i \neq j, \quad i \leftrightarrow j, \\ \deg(i) & \text{при} \quad i = j, \\ 0 & \text{иначе}, \end{cases} \tag{4}$$

причем $(v_i, v_j) \in E$, если $i \neq j$ и $i \leftrightarrow j$. Под $\deg(i)$ понимается степень вершины i (число ребер, связанных с узлом). В общем случае матрица Лапласа состоит из весов, приписываемых вершинам графа:

$L_{ij} = w(v_i, v_j)$ при $i \neq j$;

$L_{ij} = -\sum\limits_{k=1}^{n} w(v_i, v_k)$ при $i = j$.

Матрица L удовлетворяет условию $L\boldsymbol{e} = 0$, где $\boldsymbol{e}$ — вектор, составленный из единиц.

Матрица Лапласа совпадает с дискретным представлением оператора Лапласа на пятиточечном шаблоне (при использовании граничных условий типа Неймана). На сетке 8×8 имеем

$$L = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

Матрица Лапласа представляется в виде $L(G) = A - D$, где A — матрица смежности графа, а D — диагональная матрица, состоящая из весов узлов графа.

Минимизация квадратичной формы при учете соответствующих ограничений дает

$$L\boldsymbol{x} = \mu\boldsymbol{e} + \lambda\boldsymbol{x}, \tag{5}$$

где λ и μ представляют собой множители Лагранжа, подлежащие определению. Умножение обеих частей соотношения (5) на вектор $\boldsymbol{e}'$ дает $\mu = 0$, при этом $\boldsymbol{x}$ является собственным вектором матрицы L и $|E_c| = n\lambda/4$. Для минимизации числа ребер, разрезаемых при декомпозиции, необходимо, чтобы вектор $\boldsymbol{x}$ соответствовал наименьшему собственному значению матрицы L, удовлетворяющему ограничениям (2) и (3).

Матрица L является положительно полуопределенной (positive semidefinite) и имеет наименьшее собственное значение $\lambda_1 = 0$ и соответствующий ему собственный вектор $\boldsymbol{e}_1 = \{1, 1, \dots, 1\}'$. Вектор $\boldsymbol{e}_1$ не является решением задачи, поскольку он не удовлетворяет ограничениям (2). Для связного графа следующее наименьшее собственное значение матрицы Лапласа λ_2 является положительным. Собственный вектор $\boldsymbol{e}_2$, связанный с собственным значением λ_2, удовлетворяет ограничениям (2), являясь ортогональным собственному вектору $\boldsymbol{e}_1$. Вектор $\boldsymbol{e}_2$ также удовлетворяет условию (3) при использовании подходящего масштабирующего множителя, являясь тем самым решением задачи минимизации. Этот собственный вектор (вектор Фидлера, Fiedler vector) состоит из весовых множителей, связанных с вершинами графа, что позволяет применить метод деления пополам, разделяя вершины по различным подобластям в соответствии с их весовыми множителями. Собственный вектор матрицы Лапласа, соответствующий ее наименьшему нетривиальному собственному значению, определяет связность графа и его протяженность, а также выступает в качестве метрики.

Для реализации метода RSB находится собственный вектор матрицы Лапласа $\boldsymbol{e}_2$. Вектор делится на две части (находится медиана), элементы из которых присваиваются различным подобластям (вершины, находящиеся выше медианы, перемещаются в один набор, а ниже — в другой). Для каждой подобласти шаги реализуются в такой же последовательности.

Построение матрицы Лапласа и разбиение простого графа, содержащего 6 вершин, методом спектрального деления пополам иллюстрирует рис. 6. На фрагменте 6а показан исходный граф. Матрицы, соответствующие исходному графу, приводятся на фрагменте 6б (пустые клетки соответствуют нулевым элементам). Вектор $\Lambda = \{0, -1, -2, -3, -4, -5\}$ содержит собственные значения матрицы Лапласа. Интерес представляет собственный вектор $\boldsymbol{e}_2$, соответствующий минимальному ненулевому собственному значению матрицы Лапласа (вектор Фидлера). Вектор Фидлера, соответствующий собственному значению

λ_2, имеет вид $\boldsymbol{e}_2 = \{2, -1, 1, -2, 1, -1\}$. Множество индексов вершин графа упорядочивается по неубыванию компонент вектора Фидлера. Вектор $\boldsymbol{s}$, содержащий индексы вершин графа и удовлетворяющий условию $\boldsymbol{e}_2(s_i) \leqslant \boldsymbol{e}_2(s_j)$ при $i < j$, имеет вид $\boldsymbol{s} = \{4, 2, 6, 3, 5, 1\}$. С использованием вектора $\boldsymbol{s}$ вершины графа распределяются в требуемой пропорции. Одна из подобластей содержит вершины, имеющие меньшие значения компонент найденного собственного вектора, а другая — бóльшие значения. Разделение графа на два подграфа показывает фрагмент 6в. В первую подобласть попадают вершины $\{4, 2, 6\}$, а во вторую — вершины $\{3, 5, 1\}$, что соответствует оптимальному разбиению графа на две равные части.

Для нахождения собственного вектора, связанного со вторым наименьшим собственным значением, используется алгоритм Ланцоша (Lanczos algorithm), требующий $O(n)$ операций на каждую итерацию. Считая, что алгоритм Ланцоша сходится за m итераций, время работы оценивается как $O(mn)$. Метод Ланцоша требует хранения m векторов длины n. Другой алгоритм для минимизации квадратичной формы (1) предполагает использование метода сопряженных градиентов (Conjugate Gradient Method), что требует хранения только 4 векторов [25].

В отличие от методов RCB и RGB, метод RSB приводит к связным подобластям при условии, что исходный граф также является связным, при этом число ребер графа, разрезаемых при декомпозиции области, оказывается примерно в два раза меньшим по сравнению с методом RCB. Метод RSB является достаточно дорогостоящим с вычислительной точки зрения, поскольку стоимость нахождения собственных векторов оценивается как $O(n \log n)$. На практике малые изменения сетки (например, в результате адаптивного сгущения узлов) приводят к большúм изменениям вектора Фидлера [17].

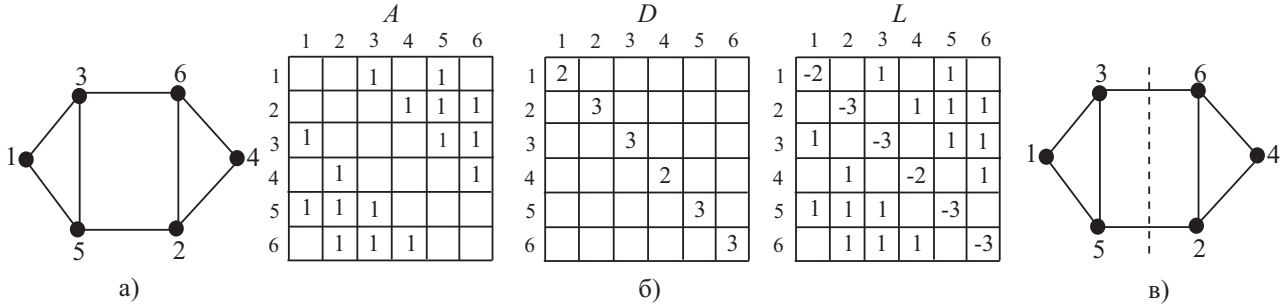


Рис. 6. Применение метода спектрального деления пополам

6.4. KL-алгоритм. Алгоритм Кернигана–Лина (Kernighan–Lin algorithm, KL) имеет итерационную природу [13, 26] и основан на понятии веса — величины, которая определяет выигрыш от перемещения вершины из одного подмножества в другое. Вес рассчитывается для каждой вершины как количество соединений вершины с другим подмножеством минус количество соединений с подмножеством, в котором вершина находится. Пока есть вершины с положительным весом, алгоритм меняет местами вершины с максимальным весом с вершинами из другого подмножества.

Начальное разбиение графа (например, случайное) улучшается в течение некоторого количества итераций при помощи обменов вершинами между подмножествами имеющегося разбиения графа. Данный алгоритм использует обмен парами и действует проходами.

В начале итерации все вершины графа являются разблокированными. На основе некоторого алгоритма половинного деления в качестве базового для каждой вершины графа оценивается коэффициент усиления (gain coefficient), представляющий собой уменьшение числа ребер, разрезаемых при разбиении, при условии, что вершина перемещается из одной подобласти графа в другую. На каждой внутренней итерации незаблокированная вершина, дающая наибольший коэффициент усиления, перемещается из подобласти, содержащей большее число вершин, в подобласть с меньшим числом вершин. Затем перемещенная вершина блокируется и обновляется общее число ребер, разрезаемых при разбиении. Процедура повторяется до тех пор, пока все вершины не окажутся заблокированными. Последние несколько перемещений обычно дают отрицательный вклад в коэффициент усиления, внутренние итерации прекращаются, а метод половинного деления на внешней итерации применяется к тому разбиению, которое дает наименьшее число ребер, разрезаемых при разбиении. Внешние итерации прекращаются, когда число ребер, разрезаемых при разбиении, прекращает уменьшаться. Применение KL-алгоритма иллюстрирует рис. 7. На шаге число разрезаемых ребер уменьшается с 7 (а) до 3 (б).

Начальное разбиение обычно формируется случайным образом. Конечный результат во многих случаях зависит от выбора начального приближения. Стоимость внешних итераций оценивается как $O\big(|E|\big)$. Несмотря на то что в большинстве случаев количество проходов сравнительно невелико, алгоритм Кер-

нигана–Лина требует $O(n^2)$ замен перед каждым шагом, что приводит к сложности $O(n^2 \log n)$.

Недостаток подхода состоит в том, что перемещения вершин могут привести к локальному минимуму. Для преодоления локальных минимумов алгоритм использует поиск экстремума — вершины с отрицательным весом перемещаются для нахождения глобального минимума. Перемещенные вершины не участвуют в дальнейшем поиске. Во время этих перемещений лучшее полученное разбиение запоминается и восстанавливается после окончаний всех перемещений.

С методом Кернигана–Лина тесно связан эвристический алгоритм Фидуччиа–Мэтьюза (Fiduccia–Mattheyses algorithm, FM) [27]. За один шаг алгоритма перемещается только одна вершина, после чего веса пересчитываются для каждой вершины. Для каждого подмножества строится очередь вершин, в которую вершины помещаются по мере убывания их веса. Перебор вершин производится в том порядке, в котором они располагаются в очереди. При нарушении условия балансировки вершина не допускает переноса. Перемещение вершины в другой набор приводит к ее удалению из очереди. При пересчете весов всех вершин очереди строятся заново. Пример использования FM-алгоритма приводится на рис. 8.

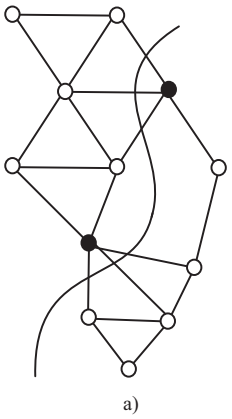


Рис. 7. Применение KL-метода

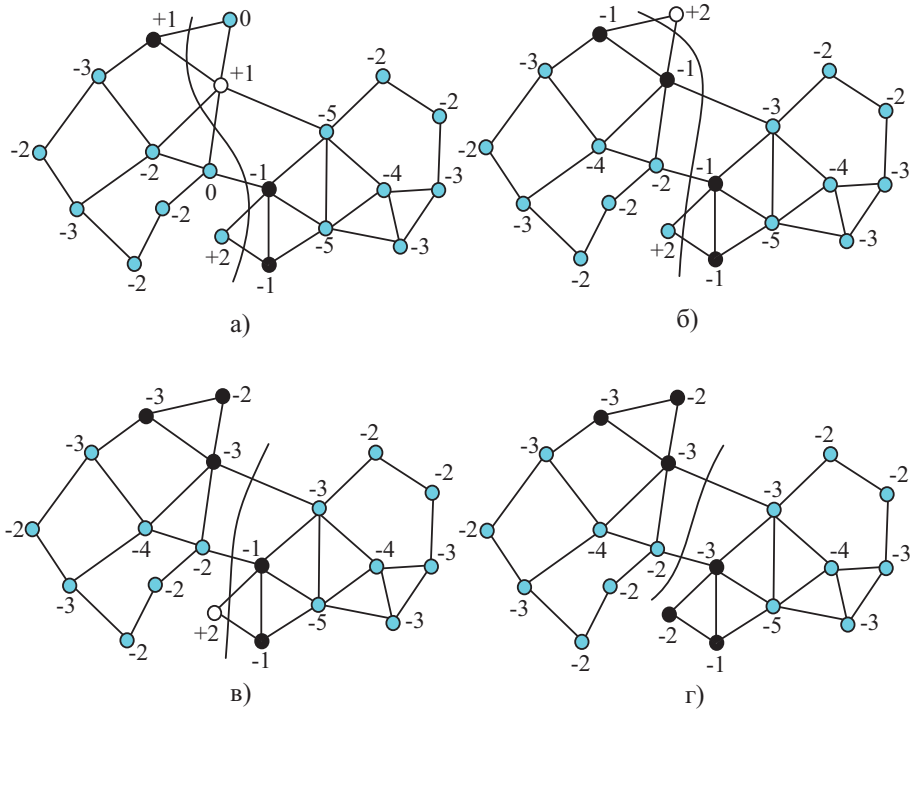


Рис. 8. Шаги FM-метода (∘ — перемещаемая вершина, • — вершины в очереди)

7. Другие методы. Методы данной группы основаны на различных идеях декомпозиции расчетной области или представляют собой комбинации известных подходов.

7.1. Методы оптимизации. Поскольку задача декомпозиции графа представляет собой задачу глобальной оптимизации, то для ее решения используются различные подходы, среди которых следует отметить генетические алгоритмы (Genetic Algorithm, GA) [28]. Алгоритмы указанного класса в силу их последовательной природы требуют существенно больше вычислительного времени и памяти по сравнению с другими подходами [29, 30]. С другой стороны, методы оптимизации достаточно просто параллелизуются и при одинаковых условиях приводят к декомпозиции области лучшего качества. В частности, методы оптимизации находят применение для улучшения качества декомпозиции, полученной при помощи более простых и быстрых подходов.

7.2. Методы, уменьшающие ширину диагонали. Из уравнения (4) следует, что любой ненаправленный граф допускает представление в виде матрицы порядка n, элемент (i, j) которой имеет ненулевое значение, если вершины i и j связаны ребром, и нулевое значение, если вершины i и j не связаны ребром.

Метод половинного деления графа оказывается эквивалентным упорядочиванию элементов матрицы

(матрица смежности, connectivity matrix) с использованием симметричных перестановок таким образом, что конечная матрица имеет малое число ненулевых элементов, стоящих вне двух основных блоков порядка $n/2$. Несмотря на то что алгоритмов упорядочивания матрицы такого вида не существует, имеются алгоритмы, которые минимизируют ширину ленты матрицы (bandwidth) [31].

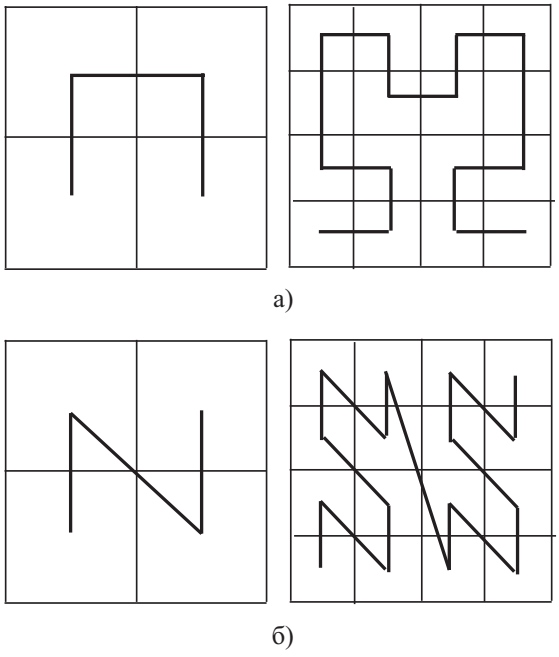


Рис. 9. Кривые Гильберта (а) и Мортона (б), используемые для упорядочивания сетки в двумерном пространстве

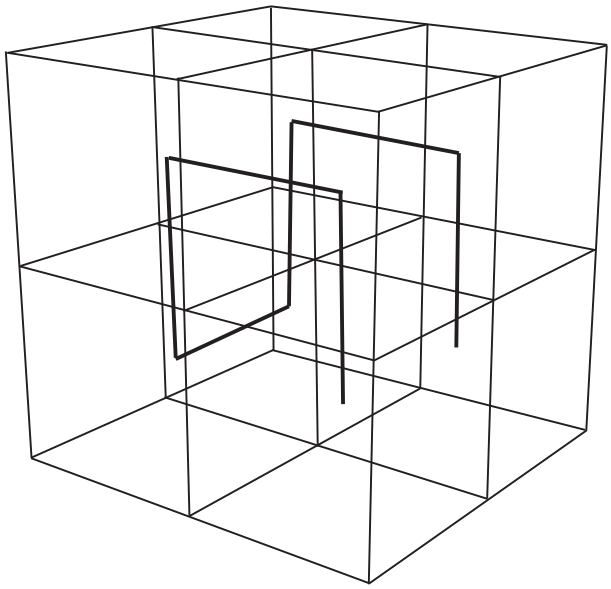


Рис. 10. Кривая Гильберта, используемая для упорядочивания сетки в трехмерном пространстве

7.3. Индексные методы. Работа многих методов балансировки основана на построении упорядоченного списка узлов и разбиении этого списка на равные сегменты. Нерегулярная сетка допускает индексирование и встраивание в d-мерную регулярную сетку.

Имеется достаточно много подходов (index based method), преобразующих d-мерную сетку в одномерный список (mapping), например метод заполнения вдоль фрактальной кривой [32] (Space-Filling Curve, SFC), проходящей через центры ячеек сетки (иногда используется термин octree или quadtree partitioning). На практике используются кривые Пеано (Peano curve) и их частные случаи — кривые Гильберта (Hilbert curve) и Мортона (Morton curve). Свойства кривых Гильберта и Мортона в d-мерном пространстве обсуждаются в работах [33, 34], а их примеры в двумерном (показаны 2 уровня сетки) и трехмерном (показан 1 уровень сетки) пространстве приводятся на рис. 9 и 10. При отображении n-мерной физической области на одномерное пространство соседние ячейки сетки, соединенные участками фрактальной кривой, представляются соседними элементами списка (свойство локальности, locality). Воспроизведение кривой Гильберта или Мортона на физической плоскости требует только локальной информации (свойство компактности, compactness). Свойство локальности используется для разработки экономичных методов хранения сеточных данных (например, древовидные структуры данных). На практике локальность приводит к решению систем линейных уравнений с разреженной матрицей, которая легко факторизуется.

В двумерном пространстве линия, образующая участок кривой Гильберта, заполняет 4 ячейки сетки (блок 2×2) и имеет форму буквы U, а линия, образующая кривую Мортона — форму буквы N. Каждый последующий уровень кривой Гильберта или Мортона делит каждую из ячеек сетки на 4, а линии, формирующие участки кривых, имеют форму повернутых букв U и N (свойство самоподобия, self-similarity). В трехмерном пространстве правила построения кривых сохраняются, но добавляется третья размерность и поворот линии, формирующей кривую (линии в форме букв U или N), в соответствующем направлении (деление происходит на блоки размерности $2 \times 2 \times 2$).

Пример работы одного из таких подходов поясняет рис. 11. Качество декомпозиции, полученное при помощи метода SFC, обычно хуже, чем при использовании геометрических подходов [7]. Привлекательная сторона состоит в высокой скорости работы, которая выше, чем у методов RCB и RIB, при этом свойство локальности улучшает использование кэш-памяти [35].

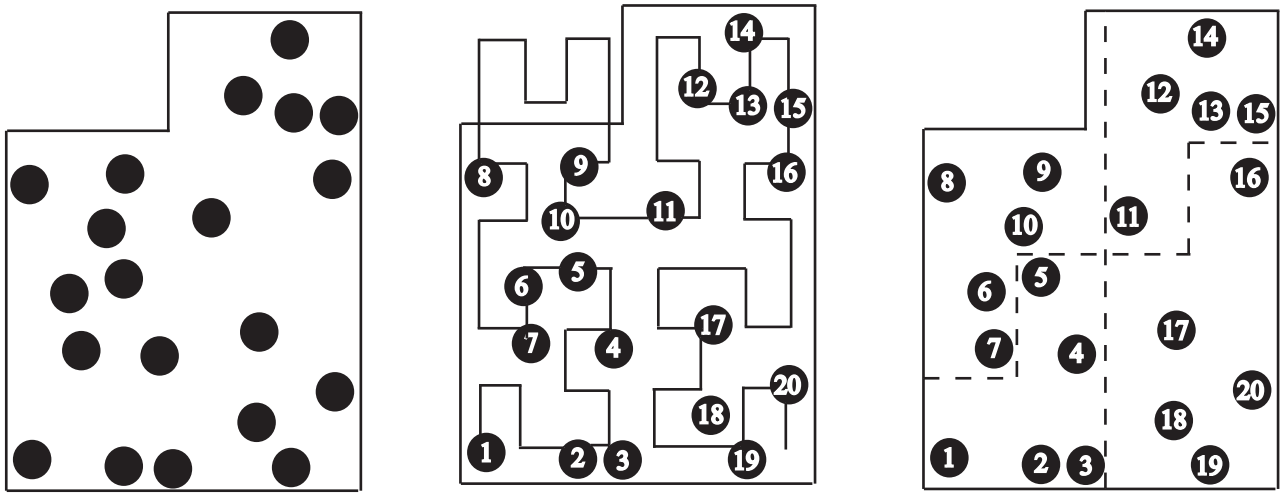


Рис. 11. Метод заполнения плоскости вдоль фрактальной кривой: а) узлы, б) заполнение области фрактальной кривой, в) декомпозиция

В одном из подходов (метод разбиения с использованием кривых Пеано) производится упорядочивание элементов в соответствии с позициями центров их масс вдоль кривых Пеано, полностью заполняющих область большой размерности. После получения списка элементов, упорядоченного в зависимости от расположения точек на кривой, достаточно разделить список на необходимое число частей в соответствии с установленным порядком [32]. Производительность таких подходов обычно ограничивается производительностью алгоритмов сортировки, которые производят упорядочивание узлов сетки вдоль кривой, и оценивается как $O(n \log n)$ [36].

Пример декомпозиции прямоугольной расчетной области на три подобласти при помощи заполнения плоскости кривой Гильберта (U-упорядочивание) приводится на рис. 12 (толстые линии указывают границы между подобластями). Для декомпозиции области используются свойства отображения двумерной области на одномерный список ячеек и свойство локальности.

Подходы, основанные на методе SFC, применяются не только для декомпозиции расчетной области и балансировки нагрузки процессоров, но и в качестве одного из компонентов многосеточных технологий для построения последовательности вложенных сеток [37].

Преимущество индексных методов состоит в том, что упорядоченный список строится достаточно быстро. Индексные подходы допускают добавление и удаление узлов, возникающих, например, в результате адаптации сетки. Качество балансировки сравнимо с тем, которое дает метод RCB. По сравнению с методами половинного деления методы данного класса требуют только локальной информации (свойство компактности).

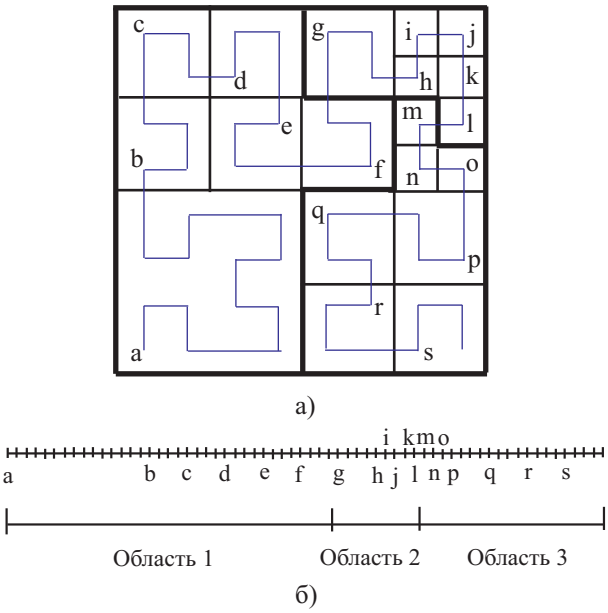


Рис. 12. Декомпозиция области в физическом пространстве (а) и одномерный список ячеек (б)

7.4. Комбинированные подходы. Развитие методов балансировки идет в направлении комбинации различных подходов, построения многоуровневых подходов (multilevel approach) и параллелизации алгоритмов.

Комбинации различных алгоритмов (hybrid approach) позволяют получить более качественную декомпозицию области [14, 38].

Идея многоуровневых, или иерархических подходов состоит в том, чтобы построить несколько графов

более грубого разрешения [39]. Обычно алгоритмы такого класса включают в себя три стадии (поэтапное огрубление графа, декомпозиция самого маленького из полученных графов, отображение разбиения на предыдущие графы с периодическим локальным уточнением границ подобластей, рис. 13).

1. Фаза сжатия графа (coarsening). Исходный граф $G = (V_0, E_0)$ представляется в виде последовательности более грубых графов $G_1, G_2, \ldots, G_k$, имеющих меньшую размерность:

$$|V_0| > |V_1| > \ldots > |V_m|.$$

Графы меньших размерностей формируются построением наибольших паросочетаний (maximal matching) и стягиванием согласованных вершин в мультивершины. Пример стягивания вершин графа G_{i-1} и формирования графа G_i приводится на рис. 14.

Для построения последовательности графов используется метод схлопывающихся граней (edge collapsing), при этом каждая вершина и каждое ребро грубого графа имеют весовые множители, указывающие на количество вершин и ребер исходного графа. Поскольку число ребер грубого графа, разрезаемых при декомпозиции, равняется числу ребер подробного графа, разрезаемых при декомпозиции, модифицированная задача имеет меньшую размерность. Имеются различные подходы к выбору ребер, подверженных схлопыванию (matching).

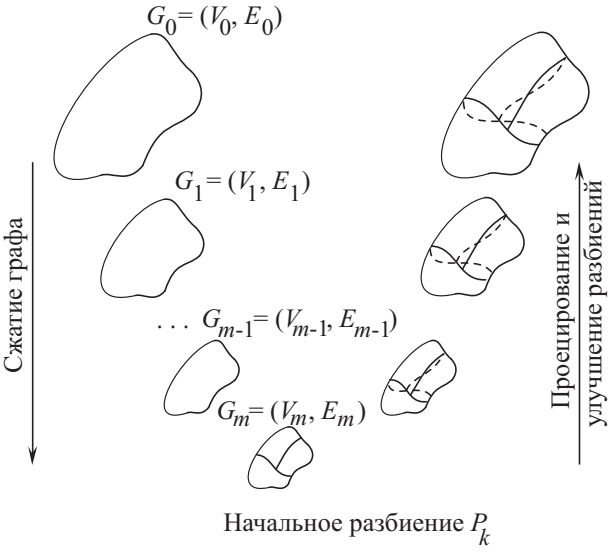


Рис. 13. Схема работы многоуровневого алгоритма разбиения графа. Сплошная линия соответствует исходному разбиению P_i, пунктирная линия — улучшенному разбиению P_{i-1}

2. Фаза начальной декомпозиции (partitioning). Проводится декомпозиция графа $G_k = (V_k, E_k)$ малой размерности на p непересекающихся подмножеств. Результатом является разбиение P_k. Поскольку размерность грубого графа сравнительно невелика, его декомпозиция проводится быстро и эффективно.

На данном этапе применяются алгоритмы рекурсивной бисекции графа (метод RGB), а для формирования разделения графа на две части используются комбинаторные и спектральные методы. При этом выбор метода декомпозиции на самом грубом уровне не играет существенной роли и не оказывает решающего влияния на качество декомпозиции [5].

3. Фаза улучшения разбиения (refinement). Разбиение P_k проецируется на граф $G_{k-1} = (V_{k-1}, E_{k-1})$, в результате чего получается разбиение P_{k-1}, которое улучшается известными алгоритмами локальной оптимизации. Процедура проецирования разбиения и его улучшения продолжается на всех уровнях и прекращается на уровне G_0. В основе многих алгоритмов улучшения разбиения лежит эвристический алгоритм Кернигана–Лина [13]. Перемещение одной вершины в укрупненном графе является эквивалентным перемещению большого числа вершин исходного графа. Перемещение группы вершин позволяет избежать некоторых видов локальных минимумов.

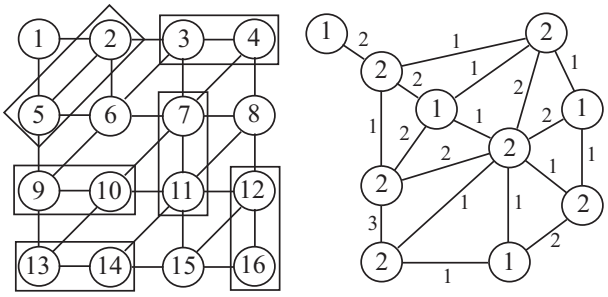


Рис. 14. Пример сжатия графа. Исходный граф с выделенными ребрами (а) и сжатый граф с мультивершинами (б)

В многоуровневом методе спектрального деления пополам (Multilevel Recursive Spectral Bisection, MRSB) собственный вектор грубого графа G_{k+1} интерполируется на граф G_k и принимается в качестве начального приближения собственного вектора графа G_k. Для улучшения приближения вместо алгоритма Ланцоша используются итерации Рэлея (Rayleigh iteration). При этом число матрично-векторных (в качестве матрицы используется матрица Лапласа соответствующего графа) произведений обычно не превосходит 10 (в алгоритме Ланцоша число таких произведений достигает сотни и более). Метод MRSB оказывается на порядок быстрее, чем алгоритм Ланцоша при реализации метода RSB. К недостаткам подхода относится его рекурсивная природа (метод является методом половинного деления). Стоимость каждой итерации оценивается как $O(n)$, где n — число вершин графа.

Основное достоинство многоуровневых алгоритмов состоит в возможности обрабатывать графы бóльших размерностей за приемлемое время. Имеется аналогия между многоуровневыми подходами к деком-

позиции области и многосеточными методами. Многоуровневый подход используется в работе [40] для ускорения метода RSB и находит применение в сочетании с KL-методом.

7.5. Параллельные методы. Для декомпозиции сеток большой размерности обычно используются параллельные версии различных подходов, например параллельный многоуровневый метод рекурсивного деления пополам (Parallel Multilevel RSB, PMRSB) и инкрементный алгоритм декомпозиции графов (incremental algorithm). Различные параллельные алгоритмы и соответствующее программное обеспечение рассматриваются в работах [14, 41].

8. Методы динамической балансировки. Методы динамической балансировки используются, например, в расчетах на адаптивных сетках, когда число узлов сетки изменяется (обычно в результате оценки ошибки дискретизации). Несбалансированность нагрузки процессоров вызывается добавлением узлов сетки, а также в результате использования локального шага по времени (local time stepping) или схем различного порядка точности [42]. Другой причиной несбалансированности являются нелинейные свойства материала в конечно-элементных расчетах.

8.1. Имеющиеся подходы. Методы динамической балансировки обеспечивают балансировку нагрузки каждого процессора, а также минимизируют число ребер, разрезаемых при декомпозиции (число коммуникаций между процессорами).

Один из методов динамической балансировки состоит в том, чтобы декомпозицию модифицированной сетки провести заново (repartition). Для этой цели используются методы статической балансировки, имеющие параллельные версии, при этом важно гарантировать близость нового разбиения области к исходному. Обычно декомпозиция всей области используется в случае существенно локализованного измельчения расчетной сетки [43, 44]. Для представления одномерного списка ячеек или узлов сетки используются древовидные структуры данных. Для сглаживания различных сегментов применяются подходы, основанные на миграции граничных элементов.

Поскольку стандартные методы балансировки минимизируют лишь число ребер, разрезаемых при декомпозиции области, но не принимают во внимание число коммуникаций между процессорами, находят применение специальные методики, такие как метод виртуальной вершины (virtual vertex) [45]. Виртуальная вершина присваивается каждой подобласти и связывается с каждой вершиной графа при помощи виртуальной грани. Вес этого ребра отражает стоимость коммуникации миграции вершины.

После новой декомпозиции расчетной области обычно применяются алгоритмы, позволяющие обрабатывать подобласти наиболее подходящими процессорами (remapping) для того, чтобы сократить число перемещений данных.

Другой подход состоит в том, чтобы новые узлы сетки перенести и обрабатывать соседними процессорами (node migration), сдвигая границы между разбиениями для достижения эффективной балансировки нагрузки. Число коммуникаций между процессорами обычно уменьшается по сравнению с декомпозицией всей расчетной области, хотя число ребер, разрезаемых при декомпозиции, возрастает. Данный подход представляется эффективным, когда несбалансированность нагрузки, связанная с локальным измельчением сетки, является сравнительно малой [46] (оптимальное разбиение области оказывается довольно близким к начальному разбиению).

Реализация алгоритма динамической балансировки зависит от конкретного приложения и обычно проводится в два этапа [4], заключающихся в расчете потока и выборе узла для миграции.

8.2. Расчет потока. Рассмотрим граф $G = (V, E)$, с каждой вершиной которого $i \in V$ связана некоторая нагрузка l_i. Задача расчета потока (flow calculation) состоит в том, чтобы найти такую мигрирующую нагрузку δ_e вдоль каждого ребра $e \in E$, что после миграции нагрузка на каждую вершину оставалась бы той же самой:

$$l_i + \sum_{i \leftrightarrow j} \delta_{ji} = \overline{l}, \quad i \in V. \tag{6}$$

Здесь l_i — нагрузка, соответствующая вершине i (например, число узлов в подобласти i) и $\overline{l}$ — средняя нагрузка на все вершины. Под δ_{ji} понимается нагрузка, мигрирующая от вершины j к вершине i. Средняя нагрузка рассчитывается по формуле $\overline{l} = \dfrac{1}{|V|} \sum\limits_{i \in V} l_i$.

Граф задачи называется процессорным графом (processor graph), соответствующим данному разбиению. Каждая вершина процессорного графа соответствует некоторой подобласти или процессору, а две вершины связаны ребром, если две подобласти имеют общую границу (в этом контексте вершина графа и процессор являются эквивалентными понятиями).

В системе линейных уравнений (6) величины δ_{ji} являются неизвестными. Имеется $|V|$ уравнений, из

которых $|V|-1$ уравнений являются независимыми, а $|E|$ остается неизвестной величиной.

В обычной ситуации граф имеет больше ребер, чем вершин ($|E| > |V|$), поэтому число неизвестных превышает число уравнений, а решение задачи не является единственным.

Подходы к решению задачи обсуждаются в работах [47–51]. Помимо задачи динамической балансировки процессоров, похожие задачи возникают при параллельной реализации метода молекулярной динамики (parallel molecular dynamic simulation) [52, 53].

8.2.1. Диффузный алгоритм. Одним из наиболее популярных алгоритмов решения задачи является диффузный алгоритм (diffusive algorithm) [47]. При решении задачи тепловой диффузии начальное неоднородное пространственное распределение температуры приводит к возникновению потока тепла, а через некоторое время система приходит в состояние теплового равновесия.

В качестве критерия балансировки нагрузки используется время счета процессора. Для определения количества передаваемых процессоров и направления их передачи решается уравнение теплопроводности для времени счета:

$$\frac{\partial l}{\partial t} = c\nabla^2 l, \tag{7}$$

где l — нагрузка процессора, c — коэффициент диффузии. Решение ищется на сетке, являющейся физической сетью процессоров. Для дискретизации уравнения диффузии используются конечные разности первого порядка. Применение более сложных подходов (например, неявной схемы второго порядка) обеспечивает более высокую скорость сходимости (сходимость достигается за несколько итераций), но требует большего числа коммуникаций на каждой итерации [7].

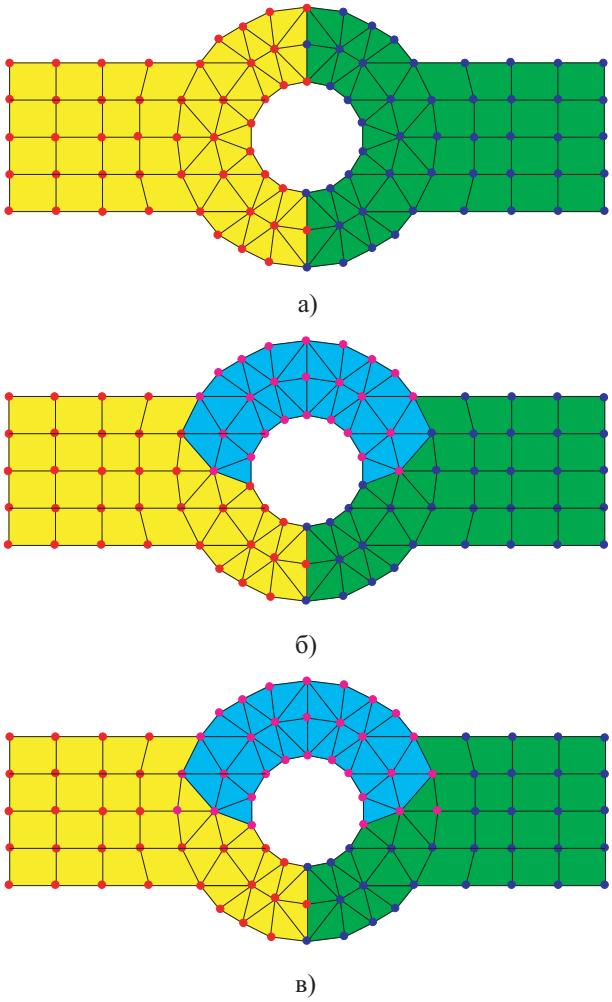


Рис. 15. Разбиение расчетной области на две подобласти (а), несбалансированное (б) и сбалансированное разбиение области на три подобласти (в)

На итерации $k+1$ процессор i делает число пересылок по отношению к процессору j, пропорциональное разности между его нагрузкой и нагрузкой соседнего процессора: $c_{ij}\left(l_i^k - l_j^k\right)$. Пересылка нагрузки c_{ij} производится в направлении градиента $l_i - l_j$ вдоль грани (i,j). Пусть $c_{ij} = c_{ji}$; тогда новая нагрузка l_i^{k+1} процессора i находится как линейная комбинация его собственной нагрузки на предыдущей итерации l_i^k и вклада соседних вершин:

$$l_i^{k+1} = l_i^k - \sum_{i \leftrightarrow j} c_{ij}\left(l_i^k - l_j^k\right), \tag{8}$$

где $i, j \in V$ и $k = 1, 2, \ldots$. Верхний индекс k относится к номеру итерации. Нагрузка вершины $i \in V$ на итерации 1 представляется в виде $l_i^1 = l_i$. Выражение (8) представляет собой итерационную схему решения уравнения диффузии, записанного в виде (7), при этом $c_{ij} = 0$, если процессоры i и j не связаны ребром процессорного графа и $1 - \sum\limits_j c_{ij} \geqslant 0$ для всех i. Выбор c_{ij} оказывает существенное влияние на скорость сходимости [7]. Для топологии вычислительной системы в виде кольца с p процессорами скорость сходимости диффузного алгоритма оценивается как $O\left(p^2\right)$. При увеличении p скорость сходимости ухудшается [48].

В матричной форме уравнение (8) приобретает вид

$$l^{k+1} = \left(I - AWA'\right) l^k, \tag{9}$$

где W — диагональная матрица порядка $|E|$, состоящая из коэффициентов c_{ij}, A — матрица размерности $|E| \times |V|$, коэффициенты которой подлежат определению.

Матрица $L = AWA'$ называется взвешенной матрицей Лапласа (weighted Laplacian matrix). Коэффициенты c_{ij} выбираются в виде $c_{ij} = \dfrac{1}{\max\{\deg(i), \deg(j)\} + 1}$, где $i \leftrightarrow j$, $i, j \in V$ [52].

Сходимость диффузного алгоритма, описываемого соотношением (9), является достаточно медленной (особенно на графах малой связности). В случае линейного графа (худший случай) достижение заданной точности приводит к затратам порядка $O(p^2)$, где p — число вершин [52]. Для улучшения сходимости вместо (9) решается уравнение

$$(I + AWA')\, l^{k+1} = l^k. \qquad (10)$$

Решение уравнения (10) рассматривается в работе [54]. Для ускорения сходимости диффузного алгоритма применяются метод сопряженных градиентов [14] и полиномы Чебышева [55]. Различные версии диффузного алгоритма (tiling algorithm, iterative tree balancing algorithm) обсуждаются в работах [42, 56].

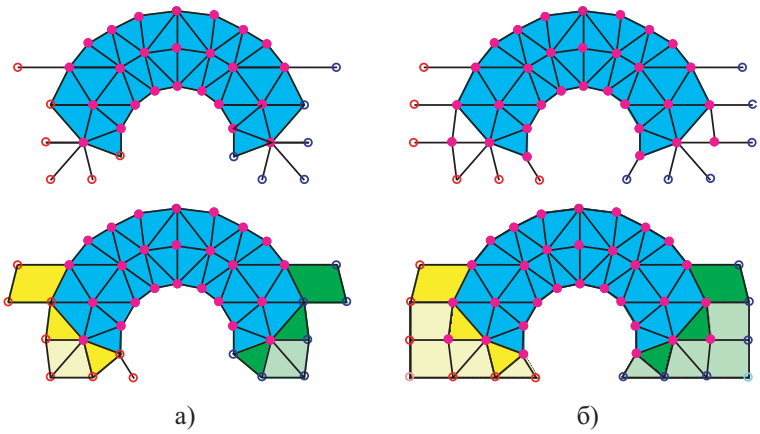


Рис. 16. Узлы и ячейки сетки, используемые для коммуникаций между процессорами в случае несбалансированного (а) и сбалансированного (б) разбиения сетки

Применение диффузного алгоритма иллюстрирует рис. 15. Исходная сетка, содержащая 102 ячейки и 94 узла, разбивается на 2 подобласти, имеющие 51 ячейку и 47 узлов каждая (фрагмент 15а). Разбиение той же самой сетки на 3 подобласти, содержащие по 34 ячейки и по 34, 35 и 25 узлов, не является сбалансированным (фрагмент 15б). Диффузный алгоритм позволяет получить сбалансированное разделение сетки на подобласти, имеющие 34 ячейки и 31, 32 и 31 узел (фрагмент 15в). В этом случае объем коммуникаций между процессорами снижается. Узлы и ячейки расчетной сетки, используемые для обменов данными между процессорами, показаны на рис. 16.

В работе [46] отмечается, что применение диффузного алгоритма является оправданным при малом уровне дисбаланса нагрузки, требуя сравнительно мало перемещений данных.

8.2.2. Алгоритм обмена размерностью. В алгоритме обмена размерностью (dimension exchange algorithm) ребра графа расцвечиваются таким образом, чтобы никакие два ребра, имеющие общую вершину, не имели одинакового цвета [47]. Пары процессоров, имеющих одинаковый цвет, группируются, и пара процессоров (i, j) с нагрузками l_i и l_j обменивается нагрузками, после чего каждый процессор из пары приобретает нагрузку $(l_i + l_j)/2$. Алгоритм сходится за d шагов, если граф рассматривается как гиперкуб размерности d.

Расширение алгоритма проводится в работе [49], в которой считается, что после обмена нагрузками процессор i приобретает нагрузку $al_i + (1 - a)l_j$. При $a = 1/2$ алгоритмы [47] и [49] оказываются эквивалентными. Анализ собственных значений показывает, что для некоторых графов подходящий выбор коэффициента a позволяет улучшить сходимость. Для графов малой связности модифицированный алгоритм имеет приблизительно такие же характеристики сходимости, как и диффузный алгоритм.

8.2.3. Многоуровневый алгоритм. Для ускорения сходимости диффузного алгоритма используется многоуровневый алгоритм (multilevel algorithm) [50]. К процессорному графу применяется алгоритм деления пополам, после чего оценивается и устраняется несбалансированность нагрузки между двумя подграфами. Процедура повторяется рекурсивным образом до тех пор, пока подграфы допускают деление пополам. Преимущество алгоритма состоит в том, что он гарантирует сходимость за $\log p$ делений пополам, а конечный результат оказывается почти всегда сбалансированным (даже если нагрузки являются целыми). Недостаток подхода состоит в том, что во многих случаях достаточно трудно разделить связный граф на два связных подграфа. Для восстановления связности добавляются новые ребра к несвязному подграфу, что является эквивалентным перемещению данных между несоседними процессорами.

8.2.4. Метод потенциала. Решение системы линейных уравнений (6) не является единственным. В работе [48] реализуется метод потенциала (method of potential), который среди всех решений выбирает то, которое минимизирует число перемещений данных между процессорами. Процессорный граф предпола-

гается направленным, а направление каждого ребра соответствует движению от вершины с наивысшим индексом к вершине, имеющей наименьший индекс.

Рассмотрим матрицу A, связанную с системой линейных уравнений (6), а также вектор $\boldsymbol{x}$, составленный из нагрузок вдоль ребер δ_{ij}, и вектор $\boldsymbol{b}$, имеющий смысл правой части системы уравнений (6). Система уравнений (6) записывается в виде $A\boldsymbol{x} = \boldsymbol{b}$. Предполагается, что перемещение данных характеризуется евклидовой нормой, а стоимость единицы коммуникаций между любыми двумя процессорами оценивается как $1/c_{ij}^2$.

Решение задачи сводится к минимизации квадратичной формы $\dfrac{1}{2}\,\boldsymbol{x}'W\boldsymbol{x} \to \min$. Матрица W является диагональной матрицей порядка $|E|$, составленной из коэффициентов c_{ij}. Матрица A имеет размерность $|V| \times |E|$ (vertex-edge incident matrix), а ее элементы для направленного процессорного графа имеют вид [24]:

$$A_{ik} = \begin{cases} +1, & \text{если узел } i \text{ является начальной вершиной ребра } k, \\ -1, & \text{если узел } i \text{ является конечной вершиной ребра } k, \\ 0 & \text{иначе.} \end{cases}$$

Решение задачи оптимизации с ограничениями представляется в виде

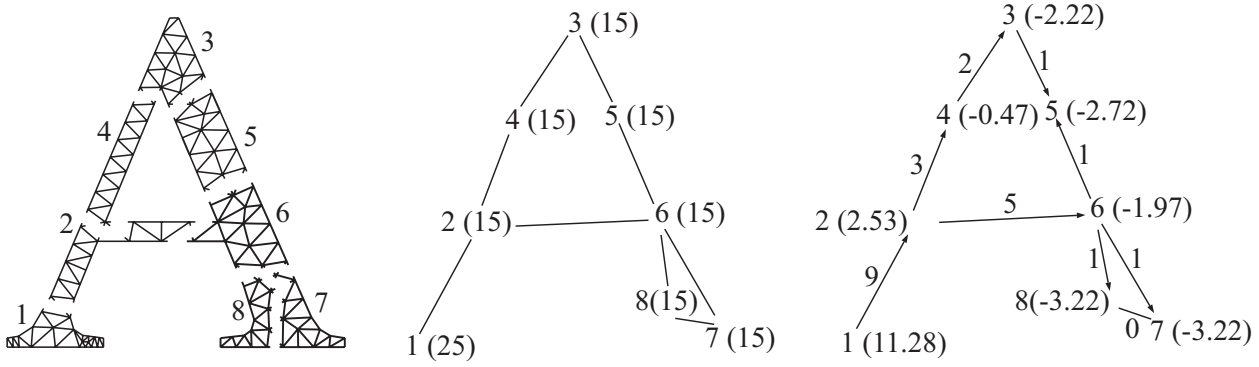
$$\boldsymbol{x} = WA'\boldsymbol{d}. \tag{11}$$

Вектор $\boldsymbol{d}$ составлен из множителей Лагранжа и находится из системы уравнений

$$L\boldsymbol{d} = \boldsymbol{b}, \tag{12}$$

где $L = AWA'$.

Задача нахождения оптимальной нагрузки процессоров сводится к задаче решения системы линейных уравнений (12). После нахождения множителей Лагранжа вектор переноса нагрузки (load transfer vector) находится из соотношения (11).



а) б) в)

Рис. 17. Сетка (а), процессорный граф (б) и нагрузки процессоров (в)

Для любого графа каждая строка k матрицы A' имеет два ненулевых элемента ($+1$ и -1), соответствующих начальной и конечной вершинам ребра k. Число нагрузок, подлежащих переносу от процессора i к процессору j (предполагается, что i является начальной вершиной, а j — конечной вершиной) вдоль ребра $e = (i, j)$, описывается соотношением $\delta_{ij} = c_{ij}(d_i - d_j)$, где d_i и d_j представляют собой множители Лагранжа, связанные с вершинами i и j. Параметры d_i и d_j имеют смысл потенциалов, связанных с вершинами i и j, а их взвешенная разность дает поток между двумя вершинами.

Матрица $L = AWA'$ представляет собой обобщенную форму матрицы Лапласа, записанной в виде (4). Для многих параллельных компьютеров стоимость единицы коммуникаций между любыми двумя процессорами в первом приближении остается такой же, поэтому $W = I$ и матрица Лапласа имеет структуру, описываемую соотношением (4).

Система линейных уравнений (12) решается любыми стандартными численными методами, например при помощи метода сопряженных градиентов.

В качестве примера рассмотрим расчетную область, имеющую вид буквы А и приведенную на рис. 17 (область взята из работы [48]), которая разбивается на 8 подобластей (фрагмент 17а). Подобласть 1 имеет

25 узлов, в то время как другие подобласти содержат по 15 узлов каждая. Процессорный граф приведен на фрагменте 17б. Нагрузка каждого процессора дается на рисунке в круглых скобках. Средняя нагрузка равняется 16.25, а наибольший коэффициент несбалансированности составляет $(25-16.25)/16.25 = 53.8\%$.

Если положить $W = I$, то система уравнений записывается в виде

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 2 & -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 2 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 4 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \\ d_8 \end{pmatrix} = \begin{pmatrix} 25-16.25 \\ 15-16.25 \\ 15-16.25 \\ 15-16.25 \\ 15-16.25 \\ 15-16.25 \\ 15-16.25 \\ 15-16.25 \end{pmatrix} = \begin{pmatrix} 8.75 \\ -1.25 \\ -1.25 \\ -1.25 \\ -1.25 \\ -1.25 \\ -1.25 \\ -1.25 \end{pmatrix}.$$

Решение системы линейных уравнений имеет следующий вид:

$$(d_1, \ldots, d_8) = (+11.28, +2.53, -2.22, -0.47, -2.72, -1.97, -3.22, -3.22).$$

Нагрузка, требующая переноса между двумя соседними процессорами, представляет собой разность между их потенциалами (множители Лагранжа) и указывается вдоль ребер на фрагменте 7в. Например, процессор 1 посылает процессору 2 нагрузку, равную $11.28 - 2.53 \approx 9$.

Результаты сравнения метода потенциала и диффузного алгоритма приводятся в таблице (число итераций, необходимых для полной сходимости) для случайного графа различной связности. Метод потенциала оказывается быстрее, причем при уменьшении связности графа ускорение метода потенциала увеличивается по сравнению с диффузным алгоритмом. В среднем метод потенциала в $2.5 \div 3$ раза быстрее диффузного алгоритма.

Сравнение сходимости метода потенциала и диффузного алгоритма

p	Диаметр графа	Степень	Метод потенциала	Диффузный алгоритм
64	9	3	25	177
64	5	5	15	57
64	4	7	11	38
64	3	9	8	23
128	87	2	116	11507
128	11	3	27	168
128	6	5	15	65
128	5	7	13	43
128	4	9	10	25
256	155	2	223	32243
256	14	3	34	155
256	7	5	19	65
256	6	7	15	48
256	4	9	12	45

8.2.5. Методы линейного программирования. В модели [48] стоимость коммуникаций в процессе миграции нагрузки оценивается как максимальная стоимость миграции нагрузки по всем процессорам: $\text{cost} = \max\limits_{i \in E}\bigl(t_0 + \alpha|x_i|\bigr)$, где t_0 — время задержки посылки сообщения, α — стоимость коммуникаций в расчете на одно слово. Задача сводится к расчету потока и его минимизации $\Bigl\{\max\limits_{i \in E}\bigl(t_0 + \alpha|x_i|\bigr)\Bigr\} \to \min$, что оказывается эквивалентным задаче минимизации $c \to \min$ при условии, что $A\boldsymbol{x} = \boldsymbol{b}$ и $c \geqslant t_0 + \alpha|x_i|$ при $i \in E$. Однако для данной задачи линейного программирования (linear programming) не существует эффективного параллельного алгоритма.

Похожая задача формулируется в работе [32], требуя минимизации соотношения $\sum\limits_{i \leftrightarrow j} \delta_{ij} \to \min$ при условии, что $\sum\limits_{i \leftrightarrow j}(\delta_{ij} - \delta_{ji}) = l_j - \overline{l}$, где $0 \leqslant \delta_{ij} \leqslant \alpha_{ij}$, $i, j \in V$, $i \leftrightarrow j$. Под α_{ij} понимается число вершин подобласти i, которые перемещаются в подобласть j, используя метод выбора вершины (node selection). Для решения задачи линейного программирования используется симплекс-метод (simplex method). Задача имеет $2|E|$ переменных и $|V| + |E|$ ограничений.

8.3. Выбор узла для миграции. После расчета потока находятся узлы, которые мигрируют от одного процессора к другому (node selection for migration). В дополнение к сохранению балансировки минимизируется число ребер, разрезаемых при разбиении, а также общее число миграций.

Для выяснения числа перемещаемых нагрузок с учетом необходимых ограничений в работе [4] используется многоуровневый алгоритм. Для выбора узлов применяется относительный коэффициент усиления

(relative gain coefficient). Работоспособность подхода демонстрируется на основе конечно-элементных расчетов на адаптивной сетке, содержащей 31172 узлов, которая за 8 шагов преобразуется в сетку с 224843 узлами. Число мигрирующих узлов составляет несколько процентов от их общего количества и достигает 100% при полном перестроении разбиения.

Послойный процесс (layer process) используется в работе [32] для выбора узла. Каждому граничному узлу присваивается метка, равная разбиению, которое дает максимальное число узлов, оказывающихся по соседству с данным узлом. Узлы, являющиеся соседними по отношению к граничным узлам, маркируются на основе меток граничных узлов. Процесс повторяется рекурсивным образом до тех пор, пока все узлы не окажутся маркированными.

9. Заключение. Проведено сравнение подходов к обеспечению статической и динамической балансировки нагрузки процессоров при решении задач механики жидкости и газа на многопроцессорных вычислительных системах. При разработке методов балансировки следует учитывать достаточно большое число факторов, оказывающих влияние на качество декомпозиции. Помимо числа ребер, разрезаемых при декомпозиции, к таким факторам относится отношение сторон различных подобластей (aspect ratio), что оказывает влияние на сходимость итерационного процесса. Для решения многофизичных задач (multiphysics problem) требуется разработка алгоритмов, удовлетворяющих целому ряду дополнительных ограничений.

Для решения задачи декомпозиции области обсуждаются геометрические методы, использующие информацию о координатах узлов сетки, и комбинаторные методы, учитывающие смежность вершин графа. К геометрическим методам относятся покоординатный метод деления пополам, инерционный метод деления пополам, деление графа с использованием кривых Пеано. К комбинаторным методам относятся метод деления графа с учетом связности вершин и алгоритм Кернигана–Лина. Приведена общая сравнительная характеристика алгоритмов по времени выполнения, точности получаемого решения и возможностям для распараллеливания.

Несмотря на достаточно большое число методов балансировки нагрузки процессоров, требуется их дальнейшее развитие в направлении масштабируемости при решении задач, в которых нагрузка на каждый процессор изменяется достаточно часто.

СПИСОК ЛИТЕРАТУРЫ

1. *Беликов Д.А., Говязов И.В., Данилкин Е.А., Лаева В.И., Проханов С.А., Старченко А.В.* Высокопроизводительные вычисления на кластерах. Томск: Изд-во ТГУ, 2008.
2. *Le Tallec P.* Domain decomposition methods in computational mechanics // Advances in Computational Mechanics. 1993. **1**, N 2. 121–220.
3. *Волков К.Н.* Применение средств параллельного программирования для решения задач механики жидкости и газа на многопроцессорных вычислительных системах // Вычислительные методы и программирование. 2006. **7**, № 1. 73–88.
4. *Walshaw C., Cross M., Everett M.* Parallel dynamic load balancing for adaptive unstructured meshes // J. of Parallel and Distributed Computing. 1997. **47**, N 2. 102–108.
5. *Karypis G., Kumar V.* A fast and high quality multilevel scheme for partitioning irregular graphs. University of Minnesota. Department of Computer Science. Technical Report TR 95-035. Minneapolis, 1995.
6. *Simon H.D.* Partitioning of unstructured problems for parallel processing // Computing Systems in Engineering. 1991. **2**, N 2, 3. 135–148.
7. *Hendrickson B., Devine K.* Dynamic load balancing in computational mechanics // Computer Methods in Applied Mechanics and Engineering. 2000. **184**, N 2–4. 485–500.
8. *Копысов С.П., Новиков А.К.* Параллельные алгоритмы адаптивного перестроения и разделения неструктурированных сеток // Математическое моделирование. 2002. **14**, № 9. 91–96.
9. *Круглякова Л.В., Неледова А.В., Тишкин В.Ф., Филатов А.Ю.* Неструктурированные адаптивные сетки для задач математической физики (обзор) // Математическое моделирование. 1998. **10**, № 3. 93–116.
10. *Diniz P., Plimpton S.J., Hendrickson B., Leland R.* Parallel algorithms for dynamically partitioning unstructured grids // Proc. of the 7th SIAM Conference on Parallel Processing for Scientific Computing. 15–17 February 1995. San Francisco, 1995. 615–621.
11. *Schloegel K., Karypis G., Kumar V.* Graph partitioning for high performance scientific simulations // CRPC Parallel Computing Handbook. San Francisco: Morgan Kauffman, 2000. 491–541.
12. *Bui T.N., Chaudhuri S., Leighton F.T., Sipser M.* Graph bisection algorithms with good average case behavior // Combinatorica. 1987. **7**, N 2. 171–191.
13. *Kernighan B.W., Lin S.* An efficient heuristic procedure for partitioning graph // Bell System Technical J. 1970. **49**. 291–308.
14. *Hu Y.F., Blake R.J.* Numerical experiences with partitioning of unstructured meshes // Parallel Computing. 1994.

20, N 5. 815–829.
15. *Miller G.L., Teng S.-H., Vavasis S.A.* A unified geometric approach to graph separators // Proc. of the 32nd Symposium on Foundations of Computer Science. 1–4 October 1991. San Juan, Puerto Rico. 1991. 538–547.
16. *Gilbert G.L., Teng S.* Geometric mesh partitioning: implementation and experiments // Proc. of the 9th International Parallel Processing Symposium. 25–28 April 1995. Santa Barbara: Computer Society Press, 1995. 418–427.
17. *Williams R.D.* Performance of dynamic load balancing algorithms for unstructured mesh applications // Concurrency — Practice and Experience. 1991. **3**, N 5. 457–481.
18. *Berger M.J., Bokhari S.H.* A partitioning strategy for nonuniform problems on multiprocessors // IEEE Trans. on Computers. 1987. **36**, N 5. 570–580.
19. *Jones M.T., Plassmann P.E.* Scalable iterative solution of sparse linear systems // Parallel Computing. 1994. **20**, N 5. 753–773.
20. *Keyser J.D., Roose D.* Grid partitioning by inertial recursive bisection. University of Leuven, Department of Computer Science. Technical Report TW-174. Leuven, 1992.
21. *Nour-Omid B., Raefsky A., Lyzenga G.* Solving finite element equations on concurrent computers // Proc. of the Symposium on Parallel Computations and Their Impact on Mechanics. 13–18 December 1986. Boston, 1986. 209–227.
22. *Shephard M.S., Flaherty J.E., de Cougny H.L., Özturan C., Bottaso C.L., Beall M.W.* Parallel automated adaptive procedures for unstructured meshes // Special Course on Parallel Computing in CFD (AGARD-R-807). 1995. 6.1–6.49.
23. *Farhat A.* A simple and efficient automatic FEM domain decomposer // Computer and Structures. 1988. **28**, N 5. 579–602.
24. *Pothen A., Simon D.H., Liou K.* Partitioning sparse matrices with eigenvectors of graphs // SIAM J. of Matrix Analysis and Applications. 1990. **11**, N 3. 430–452.
25. *Kruyt N.* A conjugate gradient method for the spectral partitioning of graphs // Parallel Computing. 1997. **22**, N 11. 1493–1502.
26. *Sadayappan P., Ercal F., Ramanujam J.* Cluster partitioning approach to mapping parallel programs onto a hypercube // Parallel Computing. 1990. **13**, N 1. 1–16.
27. *Fiduccia C.M., Mattheyses R.M.* A linear-time heuristic for improving network partitions // Proc. of the 19th IEEE Design Automation Conference. 14–16 June 1982. Las Vegas, 1982. 175–181.
28. *Bui T.N., Moon B.R.* Genetic algorithm and graph partitioning // IEEE Trans. on Computers. 1996. **45**, N 7. 841–855.
29. *Mansour N.* Allocating data the multicomputer nodes by physical optimization algorithms for loosely synchronous computations // Concurrency — Practice and Experience. 1992. **4**, N 7. 557–574.
30. *Martin O.C., Otto S.W.* Partitioning of unstructured meshes for load balancing // Concurrency — Practice and Experience. 1995. **7**, N 4. 303–314.
31. *Malone J.G.* Automated mesh decomposition and concurrent finite element analysis for hypercube multiprocessor computers // Computer Methods in Applied Mechanics and Engineering. 1988. **70**, N 1. 27–58.
32. *Ou C., Ranka S., Fox G.* Fast and parallel mapping algorithms for irregular problems // J. of Supercomputing. 1996. **10**, N 1/2. 119–140.
33. *Sagan H.* Space filling curves. New York: Springer, 1994.
34. *Liu X., Schrack G.* Encoding and decoding the Hilbert order // Software — Practice and Experience. 1996. **26**, N 12. 1335–1346.
35. *Flaherty J., Loy R., Shephard M., Szymanski B., Teresco J., Ziantz L.* Adaptive local refinement with octree load-balancing for the parallel solution of three-dimensional conservation laws // J. of Parallel and Distributed Computers. 1997. **47**, N 2. 139–152.
36. *Aftosmis M.J., Berger M.J., Murman S.M.* Applications of space-filling curves to Cartesian methods for CFD // AIAA Paper. N 2004-1232.
37. *Griebel M., Tilman N., Regler H.* Algebraic multigrid methods for the solution of the Navier–Stokes equations in complicated geometries // Int. J. for Numerical Methods in Fluids. 1998. **26**, N 3. 281–301.
38. *Vanderstraeten D., Keunings R.* Optimized partitioning on unstructured finite-element meshes // Int. J. for Numerical Methods in Engineering. 1995. **38**, N 3. 433–450.
39. *Karypis G., Kumar V.* A parallel algorithm for multilevel graph partitioning and sparse matrix ordering // J. of Parallel and Distributed Computing. 1998. **48**, N 1. 71–95.
40. *Barnard S.T., Simon H.D.* Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems // Concurrency — Practice and Experience. 1994. **6**, N 2. 101–117.
41. *Головченко Е.Н.* Комплекс программ параллельной декомпозиции сеток // Вычислительные методы и программирование. 2010. **11**. 366–372.
42. *Flaherty J.E., Loy R.M., Özturan C., Shepard M.S., Szymanski B.K., Teresco J.D., Ziantz L.H.* Parallel structures and dynamic load balancing for adaptive finite element computation // Applied Numerical Mathematics. 1998. **26**, N 1, 2. 241–263.
43. *Biswas R., Oliker L.* Experiments with repartitioning and load balancing adaptive meshes // NASA Ames Research Center. Technical Report NAS-97-021. Moffett Field, 1997.

44. *Biswas R., Das S.K., Harvey D.J., Oliker L.* Parallel dynamic load balancing strategies for adaptive irregular applications // Applied Mathematical Modelling. 2000. **25**, N 2. 109–122.
45. *Walshaw C., Berzins M.* Dynamic load-balancing for PDE solvers on adaptive unstructured meshes // Concurrency — Practice and Experience. 1995. **7**, N 1. 17–28.
46. *Schloegel K., Karypis, Kumar V., Biswas R., Oliker L.* A performance study of diffusive vs re-mapped load-balancing schemes. University of Minnesota, Department of Computer Science. Technical Report TR 98-018. Minneapolis, 1998.
47. *Cybenko G.* Dynamic load balancing for distributed memory multi-processors // J. of Parallel and Distributed Computing. 1989. **7**, N 2. 279–301.
48. *Hu Y.F., Blake R.J., Emerson D.R.* An optimal dynamic load balancing algorithm // Concurrency — Practice and Experience. 1998. **10**, N 6. 467–483.
49. *Xu C.Z., Lau F.C.M.* Analysis of the generalized dimension exchange method for dynamic load balancing // J. of Parallel and Distributed Computing. 1992. **16**, N 4. 385–393.
50. *Horton G.* A multi-level diffusion method for dynamic load balancing // Parallel Computing. 1993. **19**, N 2. 209–218.
51. *Song J.* A partially asynchronous and iterative algorithm for distributed load balancing // Parallel Computing. 1994. **20**, N 6. 853–868.
52. *Boillat J.E., Bruge F.* A dynamic load-balancing algorithm for molecular dynamics simulation on multi-processor systems // J. of Computational Physics. 1991. **96**, N 1. 1–14.
53. *Kohring G.A.* Dynamic load balancing for parallel particular simulation on MIMD computers // Parallel Computing. 1995. **21**, N 4. 683–693.
54. *Heririch A., Taylor S.* A parabolic load balancing method // Proc. of the 24th Int. Conference on Parallel Processing. 14–18 August 1995. Oconomowoc, Wisconsin. **3**. New York: CRC Press, 1995. 192–202.
55. *Hu Y.F.* An improved diffusion algorithm for dynamic load balancing // Parallel Computing. 1999. **23**, N 4. 417–444.
56. *de Cougny H.L., Devine K.D., Flaherty J.E., Loy R.M., Özturan C., Shephard M.S.* Load balancing for the parallel adaptive solution of partial differential equations // Applied Numerical Mathematics. 1994. **16**, N 1, 2. 157–182.

Поступила в редакцию
5.01.2012