РОССИЙСКАЯ АКАДЕМИЯ НАУК
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
им. М.В.КЕЛДЫША

На правах рукописи

Головченко Евдокия Николаевна

ДЕКОМПОЗИЦИЯ РАСЧЕТНЫХ СЕТОК ДЛЯ РЕШЕНИЯ ЗАДАЧ МЕХАНИКИ СПЛОШНЫХ СРЕД НА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

Специальность 05.13.18
Математическое моделирование, численные методы
и комплексы программ

ДИССЕРТАЦИЯ
на соискание ученой степени кандидата физико-математических наук

Научный руководитель

доктор физико-математических наук,
профессор
Якобовский М.В.

Москва - 2014

Оглавление

Введение

Актуальность.

Задача рациональной декомпозиции расчетных сеток возникает при численном моделировании на высокопроизводительных вычислительных системах проблем механики сплошных сред, импульсной энергетики, электродинамики и многих других. При распараллеливании подобных вычислительных приложений [1, 2, 3] используется метод геометрического параллелизма, при котором сетка, аппроксимирующая расчетную область, распределяется между процессорами по геометрическому признаку. В ходе расчета каждый процессор обрабатывает свою часть сетки. Эффективность работы многопроцессорной вычислительной системы определяется тем, насколько равномерно распределена сетка по процессорам и насколько минимизированы затраты на передачу данных между процессорами. Объем передаваемых между процессорами данных зависит от числа связей между распределенными по процессорам доменами (частями сеток). Задача декомпозиции возникает в различных параллельных приложениях, в том числе и в тех, в которых сеток нет, например, в задачах молекулярной динамики, где на домены разбивается моделируемая область с взаимодействующими между собой частицами.

Декомпозиция регулярных сеток намного проще декомпозиции нерегулярных сеток, однако, нерегулярные сетки, в частности треугольные и тетраэдральные, лучше аппроксимируют области сложной геометрической формы. Под областями сложной геометрической формы подразумеваются, например, области с внутренними полостями, декомпозиция которых приводит к возникновению несвязных доменов.

В данной работе сделан акцент на статической декомпозиции сеток. Статическая декомпозиция сетки проводится один раз перед началом расчета задачи. В отличие от нее, динамическая декомпозиция выполняется периоди-

чески в ходе расчета для балансировки загрузки и используется в задачах, вычислительная структура которых меняется в процессе счета.

Задача сбалансированного разбиения сетки на домены сводится к более общей задаче разбиения графа на домены. В этом случае выполняется разбиение графа, аппроксимирующего вычислительные и коммуникационные нагрузки сетки. Существует несколько моделей декомпозиции графов [4], отличающиеся видом графа и критериями сбалансированного разбиения: стандартная модель графа, модель двудольного графа, модель гиперграфа, модель декомпозиции с несколькими ограничениями, модель декомпозиции с несколькими целевыми функциями, модель асимметричного разбиения. В случае разбиения сеток хорошо себя зарекомендовал наиболее распространенный подход, использующий стандартную модель графа. В нем сетка аппроксимируется неориентированным графом $G = (V, E)$, где V – множество вершин, E – множество ребер. И вершины, и ребра, имеют вес. Оптимальным считается разбиение на домены, при котором выровнен суммарный вес вершин в доменах и минимизирован суммарный вес разрезанных ребер (разрезанное ребро – ребро, соединяющее вершины из разных доменов). В данной модели суммарный вес вершин в доменах отвечает за равномерность распределения вычислительной нагрузки по процессорам, которые будут обрабатывать эти домены, а суммарный вес разрезанных ребер – за коммуникационную нагрузку между процессорами. Как известно, поставленная задача декомпозиции графа является NP-полной, поэтому для ее решения используются различные эвристические методы. К геометрическим методам относятся алгоритмы рекурсивных координатной и инерциальной бисекций [5, 6] и декомпозиция с использованием кривой Гильберта [6, 9]. К методам разбиения графов относятся алгоритм спектральной бисекции [7, 8], алгоритм Kernighan-Lin (KL) и Fiduccia-Mattheyses (FM) [10], иерархические алгоритмы [10, 11, 12, 13, 6], диффузионные [14, 13] и генетические алгоритмы [15, 16, 17], используемые в рамках иерархического подхода, алгоритмы, оптими-

зирующие характеристические отношения доменов [18, 19], жадные алгоритмы (greedy methods), или алгоритмы наращивания доменов [20, 7, 21], и инкрементный алгоритм декомпозиции графов [22]. Эти алгоритмы, за исключением инкрементного, реализованы в следующих последовательных пакетах декомпозиции графов: METIS, JOSTLE, SCOTCH, CHACO и PARTY. К параллельным пакетам относятся PARMETIS (параллельная версия пакета METIS), JOSTLE, PT-SCOTCH (параллельная версия пакета SCOTCH) и ZOLTAN.

Число процессоров, на котором будет считаться вычислительная задача, как правило, заранее неизвестно. Поэтому имеет смысл предварительно однократно разбить сетку на большое число микродоменов, а потом формировать из них домены. Количество микродоменов на несколько порядков меньше числа вершин, поэтому многократное разбиение микродоменов на домены быстрее многократного разбиения всей сетки.

Широко известны методы декомпозиции областей, используемые для решения линейных и нелинейных систем уравнений, возникающих при дискретизации дифференциальных уравнений с частными производными, например, метод Шварца [23]. В нем геометрическая область разбивается на множество микродоменов с перекрытием. Микродомены раскрашиваются так, чтобы никакие два соседних микродомена не оказались раскрашенными в один цвет. Нахождение решения в микродоменах одного цвета выполняется параллельно. Сначала параллельно считаются микродомены первого цвета, затем второго и т.д. В алгоритме композиции подобластей [24] на первом этапе решения в соседних подобластях вычисляются параллельно с приближением значений на внутренних границах значениями с предыдущего шага. На втором этапе значения в приграничных полосах пересчитываются со значениями на внутренних границах, полученными на предыдущем этапе, которые считаются верными.

Еще одной областью использования разбиения сеток на микродомены является хранение больших сеток. В микродоменах функции достаточно гладкие, что позволяет отрезать биты в рамках одного микродомена. В результате на хранение информации затрачивается меньше памяти. Экономит память также локальная нумерация вершин от нуля в рамках одного микродомена, что позволяет на хранение одного номера вершины тратить меньше байтов. Существуют и другие способы компрессии сеточных данных в рамках одного микродомена.

Областью данного исследования являются нерегулярные сетки, содержащие 10^9 и более вершин. В настоящее время такие сетки невозможно разместить в памяти одного процессора (на гексаэдральную сетку, состоящую из $1.2 \cdot 10^8$ ячеек, требуется порядка 200 ГБ), поэтому для декомпозиции нужен параллельный алгоритм. Методы разбиения графов параллельных пакетов PARMETIS, JOSTLE, PT-SCOTCH и ZOLTAN основываются на иерархических алгоритмах [25, 26, 13, 27, 6], состоящих из следующих частей: поэтапное огрубление графа, декомпозиция самого маленького из полученных графов и отображение разбиения на предыдущие графы с периодическим локальным уточнением границ доменов. Недостатком таких алгоритмов является образование доменов, границы которых состоят из неоптимальных наборов сегментов [18, 28]. В частности, домены могут оказаться несвязными. Такое ухудшение качества доменов для некоторых задач является критичным. На доменах с длинными границами, или сложной конфигурацией, алгоритмы решения систем линейных уравнений сходятся за большее число итераций. Связность микродоменов важна при хранении больших сеток, поскольку на связных микродоменах коэффициент сжатия информации о сеточных данных, как правило, будет больше. В алгоритме композиции подобластей [24] у несвязных подобластей длиннее приграничные полосы, в которых требуется повторное вычисление значений, а на узких приграничных полосах возникают проблемы с применимостью метода. Несвязные домены с

оторванными ячейками являются неприемлемыми, например, для распараллеливания методики ТИМ-2D решения задач механики сплошной среды [29] на нерегулярных многоугольных сетках произвольной структуры.

Другим недостатком указанных пакетов является получение сильно несбалансированных разбиений. В частности, в разбиениях, получаемых пакетом PARMETIS, числа вершин в доменах могут отличаться в два раза. А на вычислительных системах петафлопсного и ожидаемого экзафлопсного уровня дисбаланс даже в несколько процентов приведет к существенным потерям вычислительных мощностей и дополнительным энергозатратам.

К тому же разбиения больших сеток на большое число микродоменов не всегда удается получить методами существующих пакетов разбиения графов. Например, методами пакета PARMETIS не удалось разбить несколько тетраэдральных сеток с числом вершин порядка $2.6 \cdot 10^8$ на 51200 микродоменов. Что, в совокупности с указанным ранее, обуславливает актуальность разработки новых математических моделей, алгоритмов и программ декомпозиции больших нерегулярных сеток.

Цели работы.

- Определение критериев и разработка модели декомпозиции расчетных сеток, учитывающих особенности вычислительных алгоритмов моделирования физических процессов в пространственных областях сложной геометрической формы.
- Разработка алгоритмов, позволяющих разбивать нерегулярные сетки, содержащие 10^9 и более вершин, на большое количество микродоменов при выполнении критериев сбалансированности получаемых разбиений, связности формируемых микродоменов и минимизации суммарного веса разрезанных ребер.
- Разработка программного комплекса декомпозиции больших сеток.

- Проведение вычислительных экспериментов по сравнению эффективности параллельного счета задач газовой динамики при распределении сеток по ядрам в соответствии с разбиениями, полученными разработанными алгоритмами и алгоритмами из известных пакетов.

Научная новизна.

- Определена модель декомпозиции, учитывающая особенности вычислительных алгоритмов моделирования физических процессов в пространственных областях сложной геометрической формы.
- Разработаны параллельные алгоритмы декомпозиции больших сеток, получающие разбиения, удовлетворяющие критериям сбалансированности, связности формируемых микродоменов и минимизации суммарного веса разрезанных ребер.

Теоретическая и практическая значимость.

В диссертационной работе были разработаны два алгоритма: параллельный алгоритм геометрической декомпозиции сеточных данных и параллельный инкрементный алгоритм декомпозиции графов, на основе которых был создан комплекс программ GRIDSPIDERPAR декомпозиции больших сеток. Разработанные алгоритмы поддерживают два основных этапа декомпозиции больших сеток: предварительную декомпозицию сетки по процессорам и параллельную декомпозицию сетки высокого качества. [103 – 107].

Параллельный алгоритм геометрической декомпозиции сеточных данных основан на методе рекурсивной координатной бисекции. На каждом этапе рекурсивной координатной бисекции окаймляющий сетку параллелепипед разбивается на две части. Выбирается координатная ось, вдоль которой параллелепипед имеет наибольшую протяженность. Параллелепипед разрезается перпендикулярно выбранной оси. Достоинством данного метода является

то, что при разбиении на равные домены числа вершин в получаемых доменах отличаются не больше, чем на единицу. Другими достоинствами являются экономичное использование памяти и относительная быстрота работы. Подобный алгоритм реализован в пакете ZOLTAN. Отличие рекурсивной координатной бисекции созданного алгоритма от аналогичного алгоритма в пакете ZOLTAN состоит в том, что в нем секущая плоскость (медиана) при необходимости разрезается по нескольким координатам, что позволяет обрабатывать ситуации наличия на одной плоскости множества узлов с одинаковым значением координаты. В пакете ZOLTAN вершины из медианы распределяются по областям произвольным образом, что увеличивает число разрезанных ребер.

Параллельный инкрементный алгоритм декомпозиции графов комплекса программ GRIDSPIDERPAR основан на последовательном инкрементном алгоритме декомпозиции графов (ИПМ им. М.В.Келдыша РАН) [22]. Достоинством инкрементного алгоритма является формирование преимущественно связных доменов. Инкрементный алгоритм не основывается на иерархическом подходе. Наиболее близкими к нему являются алгоритмы пузырькового роста [21] и диффузионные. Однако алгоритм пузырькового роста, в отличие от инкрементного алгоритма, не гарантирует получение сбалансированного разбиения. А существенным отличием инкрементного алгоритма от диффузионных алгоритмов является то, что в инкрементном алгоритме в случае получения разбиения низкого качества происходит освобождение части вершин из плохих доменов, после чего повторяется этап роста доменов. Другим отличием инкрементного алгоритма от известных алгоритмов является новый критерий оценки качества доменов, в соответствии с которым проверяется связность оболочек доменов. Параллельный инкрементный алгоритм комплекса программ GRIDSPIDERPAR является расширенной параллельной версией последовательного инкрементного алгоритма декомпозиции графов. Следует отметить следующие отличия. В параллельный алгоритм декомпо-

зиции графов добавлена возможность выделения групп плохих доменов и отдельная работа с ними. Ещё одним отличием является то, что в параллельном инкрементном алгоритме рост доменов происходит не просто поиском в ширину, но с учетом минимизации суммарного веса разрезанных ребер. Изменен критерий оценки качества доменов. Учитывается не только связность оболочек, но также количество плохих доменов и суммарный вес разрезанных ребер. Предложенные решения позволили ускорить нахождение разбиений высокого качества.

Комплекс программ GRIDSPIDERPAR декомпозиции больших сеток представляет интерес для применения в использующих расчетные сетки параллельных приложениях вычислительных механики сплошных сред, импульсной энергетики, электродинамики и др.

Структура и объем диссертации.

Диссертация состоит из введения, трех глав, заключения и списка литературы.

В первой главе рассмотрены различные критерии декомпозиции графов, из которых отобраны важные для диссертационной работы, учитывающие особенности вычислительных алгоритмов моделирования физических процессов в пространственных областях сложной геометрической формы. Определена поставка задачи: разработать алгоритмы, обеспечивающие разбиение нерегулярных сеток, содержащих 10^9 и более вершин, на большое количество микродоменов при выполнении критериев сбалансированности получаемых разбиений, связности формируемых микродоменов и минимизации суммарного веса разрезанных ребер.

Представлены следующие физические задачи и результаты их моделирования:

Моделирование газоплазменных потоков в диверторе токамака ITER. Токамак - тороидальная установка для магнитного удержания плазмы с целью достижения условий, необходимых для протекания управляемого термоядерного синтеза. Дивертор является одним из ключевых компонентов токамака ITER [32]. Он расположен вдоль нижней части вакуумной камеры и служит для приема потоков примесей и излучений из плазмы. Расчетная область аппроксимировалась тетраэдральной сеткой, содержащей порядка $2.8 \cdot 10^6$ ячеек. Описана математическая модель. Представлены результаты моделирования газоплазменного течения в диверторе токамака ITER.

Моделирование распространения ударной волны от приземного источника энергии взрывного типа. Для моделирования приземного взрыва была выбрана кубическая область, которая аппроксимировалась гексаэдральными сетками, содержащими порядка $6.1 \cdot 10^7$ и $1.2 \cdot 10^8$ ячеек со сгущением в области взрыва. Описана математическая модель. Представлены результаты моделирования.

Моделирование распространения ударной волны от взрыва химического взрывчатого вещества в протяженном сооружении с нетривиальной геометрией. Для рассмотрения процесса обтекания воздушной ударной волной препятствий в трубу был помещен некоторый объект испытаний, который рассматривался как жесткое тело. Расчетная область аппроксимировалась тетраэдральной сеткой, содержащей порядка $2.6 \cdot 10^7$ ячеек со сгущением вблизи области взрыва, в районе инженерной секции и вокруг объекта. Описана математическая модель. Представлены результаты моделирования.

Приведены краткие сведения о пакете MARPLE3D, с помощью которого проводилось тестирование разбиений на физических задачах.

Во второй главе рассмотрены различные модели декомпозиции графов, определена модель, описывающая проблему декомпозиции, учитывающая

особенности вычислительных алгоритмов моделирования физических процессов в пространственных областях сложной геометрической формы. Подробно рассмотрены различные алгоритмы геометрической декомпозиции сеток и декомпозиции графов. Описаны разработанные параллельный алгоритм геометрической декомпозиции сеточных данных и параллельный инкрементный алгоритм декомпозиции графов.

В третьей главе представлены следующие результаты. На основе разработанных алгоритмов был создан комплекс программ GRIDSPIDERPAR декомпозиции больших сеток (до 10^9 вершин) на большое число микродоменов.

Выполнено сравнение разбиений небольшой треугольной сетки, полученных параллельным и последовательным инкрементными алгоритмами декомпозиции графов на одном процессоре, показавшее, что параллельному инкрементному алгоритму требуется меньшее число итераций и намного меньшее время для нахождения разбиения.

Проведены тесты по оценке масштабируемости (сильное масштабирование) параллельного инкрементного алгоритма декомпозиции графов и параллельного алгоритма геометрической декомпозиции сеточных данных созданного комплекса программ GRIDSPIDERPAR. Были получены практически линейные графики увеличения ускорений обоих алгоритмов.

Проведено сравнение разбиений на микродомены четырех тетраэдральных сеток, полученных методами созданного комплекса программ GRIDSPIDERPAR, методами пакета PARMETIS, пакета ZOLTAN, и пакетом PT-SCOTCH, выявившее преимущества разработанных алгоритмов. Также проведено сравнение различных разбиений графов микродоменов на домены и разбиений сразу на домены, показавшее, что дисбаланс числа вершин в доменах, сформированных из микродоменов, не зависит от дисбаланса числа вершин в микродоменах.

Проведено сравнение эффективности параллельного счета физических задач при распределении сеток по ядрам в соответствии с различными разбиениями, полученными методами созданного комплекса программ GRIDSPIDERPAR, пакета PARMETIS, пакета ZOLTAN и пакета PT-SCOTCH, показавшее, что на разбиениях, полученных алгоритмами созданного комплекса программ GRIDSPIDERPAR, параллельный счет рассмотренных задач идет быстрее.

Проведено тестирование различных разбиений графов микродоменов на домены и разбиения сразу на домены, полученных параллельным инкрементным алгоритмом, на задаче моделирования приземного взрыва, показавшее, что при достаточном количестве микродоменов в доменах разбиения графов микродоменов не уступают по качеству разбиению сразу на домены, что подтверждается малым уменьшением скорости счета рассматриваемой физической задачи. К тому же на декомпозицию графа микродоменов при массовых расчетах требуется меньше процессоро-часов.

В заключении сформулированы **результаты** диссертационной работы, **выносимые на защиту**:

1). Разработана модель декомпозиции графов, учитывающая особенности вычислительных алгоритмов моделирования физических процессов в пространственных областях сложной геометрической формы. Разработаны параллельные алгоритмы, поддерживающие два основных этапа декомпозиции больших сеток: предварительную декомпозицию сетки по процессорам и параллельную декомпозицию высокого качества.

2). На основе разработанных алгоритмов создан комплекс программ GRIDSPIDERPAR параллельной декомпозиции больших сеток на большое число микродоменов. Проведены вычислительные эксперименты по сравнению разбиений на микродомены и домены четырех тетраэдральных сеток

(порядка 10^8 вершин, 10^9 тетраэдров), полученных методами созданного комплекса программ GRIDSPIDERPAR, пакета PARMETIS, пакета ZOLTAN и пакетом PT-SCOTCH, показавшие преимущества алгоритмов декомпозиции созданного комплекса программ GRIDSPIDERPAR в качестве получаемых разбиений.

3). Поставлен ряд вычислительных экспериментов со следующими задачами газовой динамики: моделирование газоплазменных потоков в диверторе токамака, моделирование ударной волны в протяженном сооружении и моделирование приземного взрыва. На их основе проведено тестирование различных разбиений расчетных сеток ($10^6 \div 10^8$ ячеек), подтвердившее сокращение времени счета на разбиениях, полученных алгоритмами созданного комплекса программ GRIDSPIDERPAR.

Первые два результата получены автором диссертации лично, третий результат получен совместно с Дорофеевой Е.Ю. (ИПМ им. М.В.Келдыша РАН). Автор диссертации занималась разработкой алгоритмов и программ выполнения рационального разбиения графов расчетных сеток, а также оценкой качества разбиений, Дорофеева Е.Ю. – постановкой физических задач и проведением вычислительных экспериментов. Эффективность параллельного счета физических задач оценивалась совместно.

Апробация работы.

Основные результаты диссертационной работы докладывались на конференциях:

- 8-ая Международная Конференция «Высокопроизводительные параллельные вычисления на кластерных системах (HPC-2008)», 17 – 21 ноября 2008, Казань

- Международная конференция «Современные проблемы вычислительной математики и математической физики», 16 – 18 июня 2009, Москва, ВМК МГУ имени М.В. Ломоносова
- Международная суперкомпьютерная конференция "Научный сервис в сети ИНТЕРНЕТ: суперкомпьютерные центры и задачи", 20 – 25 сентября 2010, Абрау-Дюрсо
- XI Всероссийская конференция «Высокопроизводительные параллельные вычисления на кластерных системах», 1 – 3 ноября 2011, Нижний Новгород, ННГУ
- Международная суперкомпьютерная конференция "Научный сервис в сети ИНТЕРНЕТ: поиск новых решений", 17 – 22 сентября 2012, Абрау-Дюрсо
- International Conference on Parallel Computing - ParCo2013, 10 – 13 сентября 2013, Мюнхен, Германия
- Международная суперкомпьютерная конференция "Научный сервис в сети Интернет: все грани параллелизма", 23 – 28 сентября 2013, Новороссийск

Публикации.

Основные результаты диссертации опубликованы в двенадцати работах, четыре из которых в изданиях, рекомендованных ВАК:

1). Головченко Е.Н. Комплекс программ параллельной декомпозиции сеток // Вычислительные методы и программирование. 2010. Т. 11. 360-365.

2). Е.Н. Головченко. Параллельный пакет декомпозиции больших сеток // Математическое моделирование. 2011. Т. 23. № 10. 3-18.

3). Бухановский А. В., Марьин С. В., Князьков К. В., Сиднев А. А., Жабин С. Н., Баглий А. П., Штейнберг Р. Б., Шамакина А. В., Воеводин В. В., Го-

ловченко Е. Н., Фалалеев Р. Т., Духанов А. В., Тарасов А. А., Шамардин Л. В., Моисеенко А. И. Результаты реализации проекта «Мобильность молодых ученых» в 2010 году: развитие функциональных элементов технологии iPSE и расширение состава прикладных сервисов // Известия высших учебных заведений. Приборостроение. 2011. Т. 54. №10. 80 - 86.

4). Е.Н. Головченко. Разбиение больших сеток // Вестник Нижегородского университета им. Н.И. Лобачевского. Серия «Информационные технологии». Нижний Новгород: Издательство ННГУ. 2012. № 5(2). С. 309-315.

Автором получено свидетельство о государственной регистрации программы для ЭВМ:

Якобовский М.В., Головченко Е.Н. Библиотека параллельной декомпозиции больших сеток // Свидетельство о государственной регистрации программы для ЭВМ №2013617356 от 21.06.2013. Российская Федерация.

Благодарности.

Автор выражает искреннюю признательность своему научному руководителю доктору физико-математических наук, профессору Якобовскому М.В. за постановку задачи, поддержку в подготовке диссертации, терпение, мудрое и тонкое руководство.

Автор благодарна доктору физико-математических наук Гасилову В.А. за постановку физических задач и поддержку в вопросах, связанных с диссертацией.

Автор благодарна Дорофеевой Е.Ю. за постановку физических задач, проведение вычислительных экспериментов с физическими задачами, а также за активное участие, помощь и поддержку в нашей совместной деятельности.

Глава 1. Постановка задачи

1.1 Критерии декомпозиции

В статьях упоминаются следующие критерии декомпозиции графов:

- сбалансированность разбиений
- минимизация суммарного веса разрезанных ребер [4, 30]
- минимизация суммы чисел соседей доменов, что приведет к минимизации суммарного числа сообщений между процессорами [4]
- минимизация максимального суммарного веса исходящих из доменов ребер [4, 30], что предполагает минимизацию максимального объема коммуникаций, выполняемых каждым процессором
- минимизация максимального числа вершин в доменах, инцидентных другим доменам (имеющих соседей в других доменах) [4], что также предполагает минимизацию максимального объема коммуникаций, выполняемых каждым процессором
- минимизация максимального числа соседей доменов, что приведет к минимизации максимального числа сообщений, обрабатываемых каждым процессором [4]
- минимизация расстояний между посылающими и принимающими процессорами [4]
- оптимизация характеристических отношений доменов (aspect ratio) [18, 28]. Выполнение этого критерия стремится сделать домены шарообразной формы, в результате чего они получаются более компактными. Варианты вычисления характеристических отношений

доменов могут быть разными. Например, отношение самого длинного ребра на границе домена к самому короткому, или отношение площади наименьшего описанного круга к площади наибольшего вписанного и т. д. Подробнее про оптимизацию характеристических отношений доменов написано в Главе 2

- связность доменов [31]. Связность критична для некоторых задач. На доменах с длинными границами, или сложной конфигурацией, алгоритмы решения систем линейных уравнений сходятся за большее число итераций. Связность микродоменов важна при хранении больших сеток, поскольку на связных микродоменах коэффициент сжатия информации о сеточных данных, как правило, будет больше. В алгоритме композиции подобластей [24] у несвязных подобластей длиннее приграничные полосы, в которых требуется повторное вычисление значений, а на узких приграничных полосах возникают проблемы с применимостью метода. Несвязные домены с оторванными ячейками являются неприемлемыми, например, для распараллеливания методики ТИМ-2D решения задач механики сплошной среды [29] на нерегулярных многоугольных сетках произвольной структуры.

Несмотря на ограничения метрики разрезанных ребер стандартный подход доказал свою успешность при параллельном решении дифференциальных уравнений и задач, основанных на сетках, по нескольким причинам [4]. Первое, вершины в сетках обычно имеют сравнительно небольшое число соседей, поэтому число разрезанных ребер отличается на небольшой множитель от реального коммуникационного объема. Второе, вычислительные сетки обычно имеют высокую степень геометрической локальности, что ограничивает число сообщений, которое посылает каждый из процессоров.

Вычисление характеристических отношений доменов требует знания геометрии графа, которое не всегда доступно.

Исходя из написанного выше, определена следующая

Постановка задачи:

Разработать алгоритмы, обеспечивающие разбиение нерегулярных сеток, содержащих 10^9 и более вершин, на большое количество микродоменов при выполнении критериев сбалансированности получаемых разбиений, связности формируемых микродоменов и минимизации суммарного веса разрезанных ребер.

Следует уточнить, что одновременное удовлетворение указанным критериям является невыполнимой задачей, поэтому во всех эвристических алгоритмах выбирается некоторый баланс, или предпочтение, между выигрышами по одним критериям и проигрышами по другим.

При выборе критериев декомпозиции учтены особенности алгоритмов моделирования физических задач, рассмотренных в диссертационной работе. Кроме критериев сбалансированности получаемых разбиений и минимизации суммарного веса разрезанных ребер, которые влияют на равномерность распределения вычислительной нагрузки по процессорам и на коммуникационную нагрузку между процессорами, учитывается также критерий связности формируемых микродоменов. Несвязные домены обладают большим числом соседей и длинными границами, что ведет к большему количеству обменов между процессорами, обрабатывающими данные домены, и к большим объемам передаваемых данных. Другие особенности рассматриваемых физических задач приведены ниже.

1.2 Математические модели

В параграфах 1.3 – 1.4 представлен ряд физических задач и результаты их моделирования.

Как представляющие наибольший интерес для области исследований, рассматривались только неструктурированные разномасштабные сетки или прямоугольные сетки с различным характеристическим отношением. Были выбраны сетки различных размеров: небольшие (от 10^6 до 10^7 ячеек), средние (от 10^7 до 10^8 ячеек) и большие (свыше 10^8 ячеек).

При решении физических задач использовались различные комбинации следующих моделей:

Идеальная РМГД-модель (в предельном случае отсутствия излучения сводится к идеальной ГД модели)

В общем случае решается полная система уравнений радиационной магнитной газовой динамики с табличными уравнениями состояния:

$$\frac{\partial}{\partial t}\rho + \nabla(\rho\vec{w}) = 0$$

(уравнение неразрывности)

$$\frac{\partial}{\partial t}\rho w_i + \sum_k \frac{\partial}{\partial x_k}\Pi_{ik} = 0$$

(закон сохранения импульса)

$$\Pi_{ik} = \rho w_i w_k + P\delta_{ik} - \frac{1}{4\pi}\left(H_i H_k - \frac{1}{2}H^2\delta_{ik}\right)$$

(выражение для тензора Π_{ik})

$$\frac{\partial}{\partial t}\vec{H} - \nabla\times\left(\vec{w}\times\vec{H}\right) = 0$$

(уравнение эволюции магнитного поля)

$$\frac{\partial}{\partial t}\left(\rho\varepsilon+\frac{1}{2}\rho w^2+\frac{H^2}{8\pi}\right)+\nabla\vec{q}=0$$

(уравнение для полной энергии / ЗСЭ)

$$\vec{q}=\left(\rho\varepsilon+\frac{1}{2}\rho w^2+P\right)\vec{w}+\frac{1}{4\pi}\vec{H}\times\left(\vec{w}\times\vec{H}\right)$$

(выражение для вектора плотности потока энергии)

Здесь и далее: ρ - плотность среды, $\vec{w}=\left(w_x,w_y,w_z\right)^T$ - вектор скорости среды, ε - удельная внутренняя энергия, P - давление, $\vec{q}$ - вектор плотности потока энергии, $\vec{H}$ - вектор напряженности магнитного поля.

Интегрирование по времени осуществляется в рамках схемы предиктор-корректор, обеспечивающей второй порядок аппроксимации. На каждом этапе применялась явная схема (центрально-разностная схема Лакса-Фридрихса, модифицированная и обобщенная на случай использования трехмерных неструктурированных сеток) с ограничением на шаг по времени согласно условию Куранта.

Данная схема носит локальный характер, т.е. значение в каждой конкретной ячейке зависит от значений в нескольких соседних ячейках в соответствии с выбранным шаблоном, но не зависит от значений в остальной расчетной области, что в случае параллельных вычислений существенно снижает объем межпроцессорных обменов.

Модель теплопереноса (в предельном случае отсутствия излучения учитывается только кондуктивный теплоперенос)

В общем случае учитывается радиационный и кондуктивный теплоперенос:

$$\frac{\partial}{\partial t}(\rho\varepsilon)=-\nabla\left(\hat{\kappa}\ gradT\right)+Q_{Rad}$$

Тензор теплопроводности $\hat{\kappa}$, учитывающий анизотропию среды, зависит от термодинамического состояния вещества (ρ, T), а также от величины и направления магнитного поля, что обусловлено эффектом замагниченности. Q_{Rad} - изменение энергии вследствие лучистого теплообмена.

$$Q_{Rad} = -\nabla\vec{W}$$

Квазистационарное уравнение переноса лучистой энергии в диффузионном приближении заменяется двумя уравнениями: точным уравнением неразрывности для лучистой энергии и приближенным уравнением, связывающим поток и плотность лучистой энергии. Второе уравнение получается в предположении угловой изотропии поля лучистой энергии. Система уравнений имеет вид

$$\nabla\vec{W} + \kappa^P cU = \kappa^P cJ$$

(баланс энергии)

$$\vec{W} = -\frac{c}{3\kappa^R}\mathrm{grad}U$$

(поток энергии)

или
$$-\nabla\frac{c}{3\kappa^R}\mathrm{grad}U + \kappa^P cU = \kappa^P cJ$$

Здесь κ^P означает осреднение коэффициента поглощения по Планку, а κ^R – по Росселанду. Граничное условие на выходе для нормального потока: $W_n = -cU/2$.

Уравнения радиационного и кондуктивного теплопереноса решаются по неявной схеме на основе оригинального варианта метода Галеркина с разрывными базисными функциями. Нелокальность данной схемы (зависимость значения в каждой конкретной ячейке от значений во всей расчетной облас-

ти) при параллельных вычислениях приводит к значительному увеличению межпроцессорных обменов.

Комбинированная модель турбулентности

Модель учитывает влияние турбулентной вязкости на скорость потока:

$$\frac{\partial}{\partial t} w_k = \frac{\partial}{\partial x_k}\left[(\mu + \mu_t)\left(\frac{\partial w_i}{\partial x_k} + \frac{\partial w_k}{\partial x_i} - \frac{2}{3}\delta_{i,k}\frac{\partial w_l}{\partial x_l} \right) \right]$$

Здесь μ — коэффициент динамической вязкости (сдвиговая вязкость), μ_t — дополнительная “турбулентная вязкость”.

Турбулентная вязкость рассчитывается по комбинированной модели Смагоринского и Спаларта-Альмареса. Для реализации модели Спаларта-Альмареса применялась полунеявная схема. Источниковые члены, отвечающие за производство и диссипацию, рассчитывались со старого временного слоя. Оператор Лапласа, отвечающий за диффузию турбулентного параметра, рассчитывался с нового временного слоя по неявной схеме. Турбулентная вязкость в модели Смагоринского вычислялась по скорости, взятой с текущего временного слоя гидродинамического солвера, но без учета вязкости. Величина турбулентной вязкости в комбинированной модели находится как минимальное из значений, полученных в моделях Смагоринского и Спаларта-Альмареса: $\mu_t = \min\{\mu_{SA}; \mu_{LES}\}$.

Для решения модифицированного уравнения Навье-Стокса с дополнительной турбулентной вязкостью использовалась консервативная неявная схема. Пространственная дискретизация вязких членов выполнялась методом Галеркина с разрывными базисными функциями.

Следует отметить, что турбулентная вязкость значительна только в пристеночных областях. Таким образом, при параллельном решении задач с учетом турбулентной вязкости объем межпроцессорных обменов тем больше,

чем больше элементов, принадлежащих геометрической границе сетки (например, в узких протяженных областях или областях с большим количеством неровностей). Данная особенность не была учтена в работе, но ее можно учесть введением весов вершин в разбиваемые графы (весов, усредненных по различным солверам).

1.3 Моделирование газоплазменных потоков в диверторе токамака

Токамак (тороидальная камера с магнитными катушками) - тороидальная установка для магнитного удержания плазмы с целью достижения условий, необходимых для протекания управляемого термоядерного синтеза. Плазма в токамаке удерживается специально создаваемым комбинированным магнитным полем – тороидальным внешним, и полоидальным полем тока, протекающего по плазменному шнуру. Ток в плазме обеспечивает разогрев плазмы и удержание равновесия плазменного шнура в вакуумной камере. Этим токамак, в частности, отличается от стелларатора, являющегося одной из альтернативных схем удержания, в котором и тороидальное, и полоидальное поля создаются с помощью внешних магнитных катушек (Рис. 1 - 2).

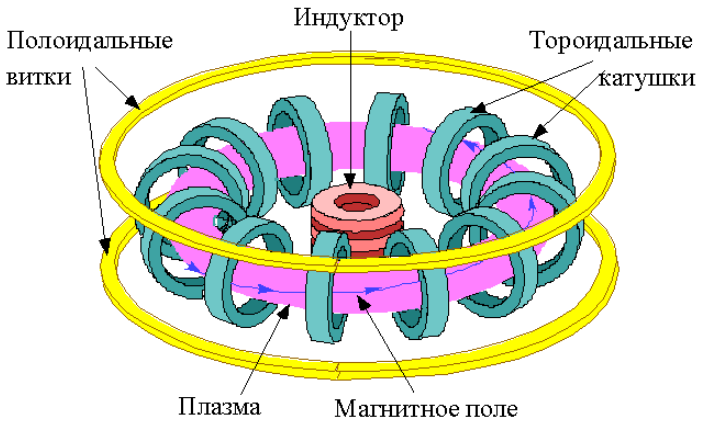


Рис. 1. Схема принципиальных узлов токамака

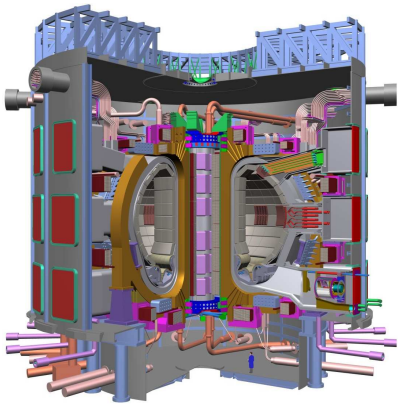


Рис. 2. Проектируемый токамак ITER - общий вид

Дивертор является одним из ключевых компонентов токамака ITER [32] (Рис. 4). Он расположен вдоль нижней части вакуумной камеры и служит для приема потоков примесей и излучений из плазмы. В диверторе можно выделить два принципиальных блока: несущая конструкция, изготавливаемая из нержавеющей стали, и компоненты, непосредственно контактирующие с плазмой. В машине ITER дивертор – это 54 дистанционно заменяемые кассеты, каждая из которых содержит три основных компоненты, контактирующие с плазмой: диверторные пластины (внутренняя и внешняя) и купол (Рис. 3). Диверторные пластины расположены на пересечении силовых линий магнитного поля. Здесь потоки высокоэнергетичной плазмы врезаются в пластины. Их кинетическая энергия преобразуется в тепло, и тепловой поток, приходящийся на пластины, чрезвычайно интенсивен. В диверторных камерах удаётся смягчить нагрузку от плазмы на диверторные пластины за счёт дополнительного охлаждения.

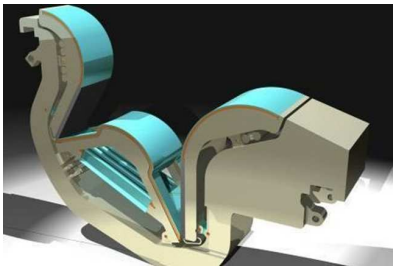


Рис. 3. Кассета дивертора ITER

Рис. 4. Дивертор токамака ITER - общий вид

На Рис. 5 представлена принципиальная схема дивертора, максимально соответствующая доступному описанию дивертора ITER. Желтым цветом на рисунке выделена часть расчетной области. Лиловым обозначен купол дивертора.

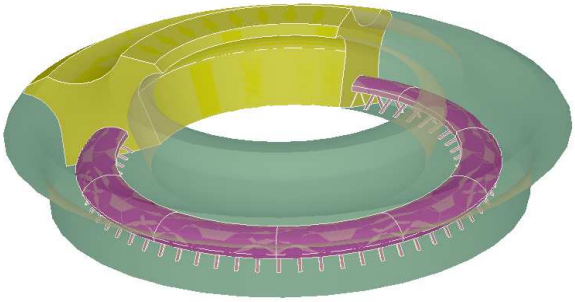


Рис. 5. Геометрия дивертора

Расчетная область аппроксимировалась тетраэдральной сеткой, содержащей порядка $2.8 \cdot 10^6$ ячеек (Рис. 6). Для расчета задачи использовались разбиения сетки на 256 доменов.

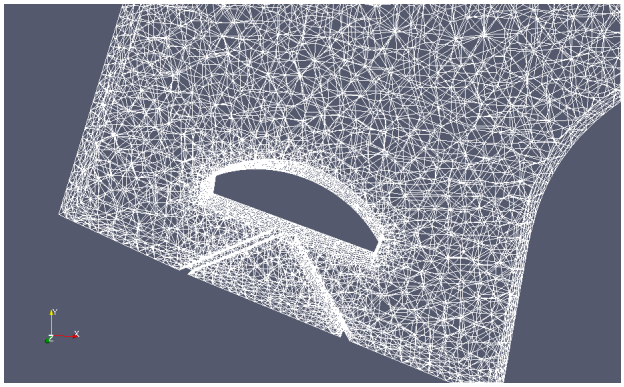


Рис. 6. Фрагмент сетки в районе купола дивертора

Считалась полная система уравнений радиационной магнитной газовой динамики с учетом радиационного и кондуктивного теплопереноса и турбулентной вязкости. Справедливость такого приближения для расчета течений в диверторе ранее обсуждалась в ряде работ (например, [33]).

Данная задача имеет следующие особенности:

- сравнительно небольшая сетка;

- сложная геометрия области (множество небольших отверстий в расчетной области, соответствующих элементам конструкции, удерживающим купол дивертора), что усложняет получение разбиений на связные домены;
- большое число элементов, принадлежащих геометрической границе сетки (большая площадь поверхности относительно объема области);
- расчет МГД по явной схеме кондуктивного и радиационного теплопереноса и турбулентной вязкости по неявной схеме.

Расчет газоплазменных течений в диверторе токамака проводился на суперкомпьютере «Helios» на 256 ядрах. На Рис. 7 представлены результаты моделирования газоплазменного течения в диверторе токамака ITER.

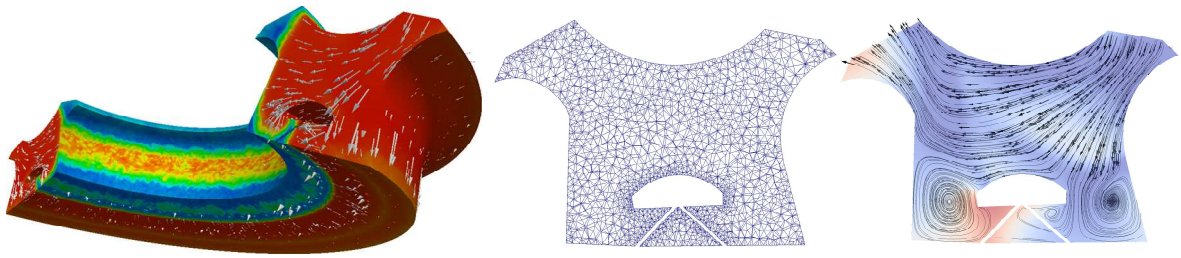


Рис. 7. Результаты моделирования газоплазменного течения в диверторе токамака ITER: фрагмент расчетной области (слева), поперечное сечение тетраэдральной сетки со сгущением в районе купола дивертора (посередине) и полностью установившееся течение газа и поле температур в диверторе (справа)

1.4 Моделирование воздушных ударных волн

1.4.1 Моделирование распространения ударной волны от приземного источника энергии взрывного типа.

Собственно взрыв, если под этим термином подразумевать первичное энерговыделение, развивается за время от нескольких микросекунд до миллисекунд. Однако чаще всего, говоря о взрыве, имеют в виду внешнюю кар-

тину – появление светящейся области, приход в точку наблюдения ударной волны, подъем облака взрыва и т.п.

Количественные значения параметров во фронте ударной волны (давление, плотность, температура воздуха, массовая скорость и др.) на стадии взрыва, когда можно пренебречь формой и размерами источника, удовлетворительно описываются автомодельным решением (модель точечного взрыва [34]), если учесть, что в ударную волну может быть передано до 50% энергии взрыва. При взрыве вблизи поверхности земли важную роль в распространении воздушной ударной волны играет взаимодействие с землей и образование отраженной волны. При взаимодействии со светящейся областью отраженной от земли ударной волны происходит нарушение сферической симметрии (нижняя кромка сферы деформируется).

Внешняя картина развития взрыва во многом зависит от высоты, на которой осуществляется взрыв. При более низких взрывах ударная волна может сместить светящуюся область как целое и увеличить размер по горизонтали. В момент отражения давление во фронте скачкообразно увеличивается в несколько раз относительно давления в падающей волне.

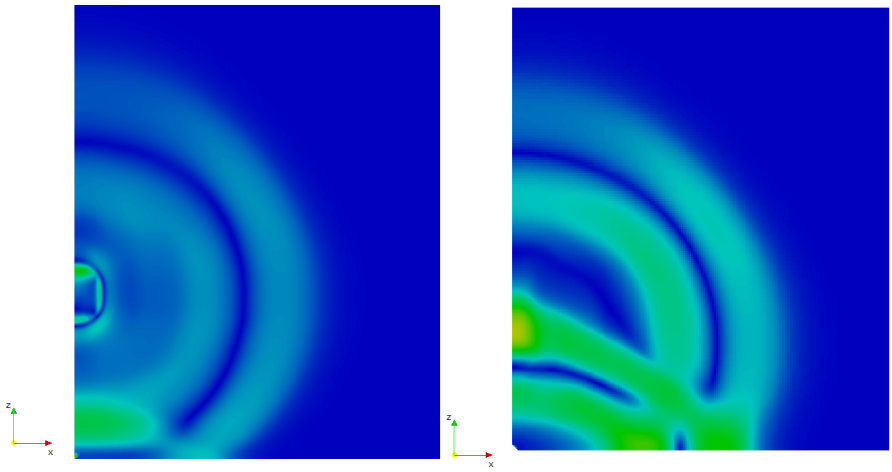


Рис. 8. Приземный взрыв, результаты моделирования

Распространение воздушной ударной волны, трансформация светящейся области в облако взрыва и дальнейший подъем облака сопровождаются

крупномасштабными газодинамическими течениями и загрязнением среды в районе взрыва [35].

Для моделирования приземного взрыва была выбрана кубическая область, которая аппроксимировалась гексаэдральными сетками, содержащими порядка $6.1 \cdot 10^7$ и $1.2 \cdot 10^8$ ячеек со сгущением в области взрыва. Общий вид аналогичной, но более крупной сетки представлен на Рис. 9. Для проведения расчетов сетки были разбиты на 3072, 4096 и 10080 доменов, соответственно.

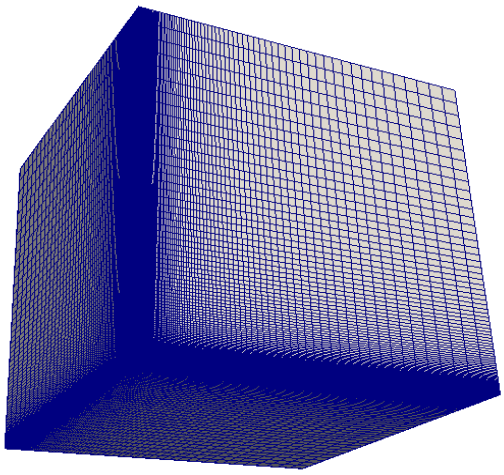


Рис. 9. Гексаэдральная сетка для моделирования приземного взрыва

Считалась полная система уравнений газовой динамики с учетом кондуктивного теплопереноса.

Данная задача имеет следующие особенности:

- большие сетки;
- простая геометрия области;
- значительная разномасштабность сеточных элементов;
- сильная неоднородность физических процессов;
- расчет ГД по явной схеме, кондуктивного теплопереноса по неявной схеме.

Расчет процесса формирования и распространения ударной волны от приземного взрыва проводился на суперкомпьютере «Ломоносов» на 3072, 4096 и 10080 ядрах. На Рис. 10 в качестве результатов моделирования приземного взрыва представлены пространственные изменения давления в момент времени t=1000 мс. Отображены проекции изоповерхностей, построенных с различным шагом, на сетку.

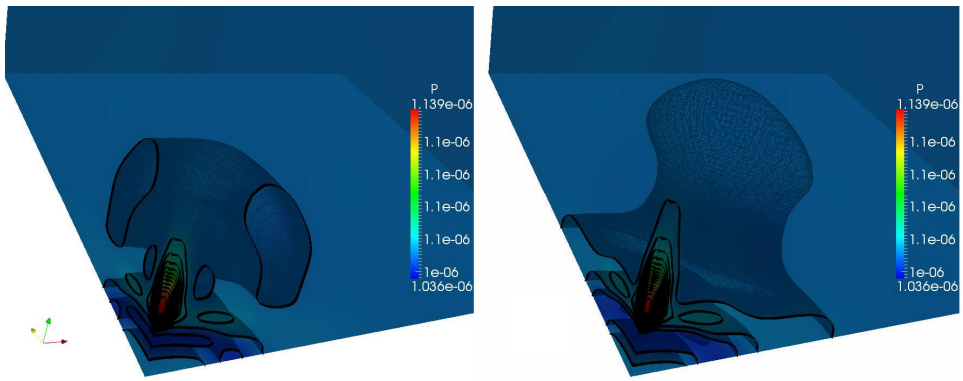


Рис. 10. Апроксимация изоповерхностей давления на расчетную сетку P[10^{11} Па] в момент времени t=1000 мс с шагом dP=691 Па (слева) и dP=518 Па (справа)

1.4.2 Моделирование распространения ударной волны от взрыва химического взрывчатого вещества в протяженном сооружении с нетривиальной геометрией.

Воздушная ударная волна химического взрыва вблизи границы раздела воздух-земля является фактором, к изучению которого имеется повышенный интерес в фундаментальных и прикладных исследованиях. Механическое действие воздушной ударной волны на объекты представляет собой многоплановую проблему из-за разнообразия самих объектов, различия условий воздействия и степени разрушения объекта в процессе воздействия. Действие ударной волны на преграду (объект) формирует динамическую нагрузку, которая определяется параметрами ударной волны, формой и размерами преграды, а также ориентацией преграды относительно вектора скорости движения фронта волны. Процесс взаимодействия ударной волны с преградой обычно разделяют на две характерные фазы:

- фаза дифракции (начальный период) – от момента соприкосновения фронта волны с преградой до установления сравнительно стабильного процесса обтекания преграды потоком сжатого воздуха;

- фаза квазистационарного обтекания – после окончания фазы дифракции до момента окончания действия положительной фазы волны на преграду.

При взрывах химического взрывчатого вещества возникают воздушные волны умеренной интенсивности, в которых нет интенсивного излучения, но может оказаться существенным турбулентный тепло-массоперенос [36].

На Рис. 11 представлена принципиальная схема ударной трубы. Для рассмотрения процесса обтекания воздушной ударной волной препятствий в трубу был помещен объект сложной формы, который рассматривался как твердое тело. Расчетная область аппроксимировалась тетраэдральной сеткой, содержащей порядка $2.6 \cdot 10^7$ ячеек со сгущением вблизи области взрыва, в районе инженерной секции и вокруг объекта. Для проведения расчетов сетка была разбита на 4096 доменов.

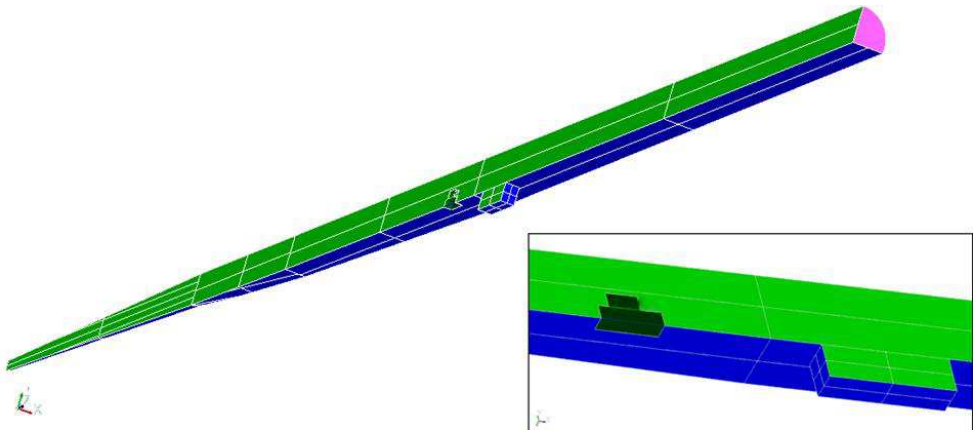


Рис. 11. Геометрия расчетной области

Считалась полная система уравнений газовой динамики с учетом турбулентной вязкости.

Данная задача имеет следующие особенности:

- сетка средней величины;
- протяженная область, сложная геометрия области;
- значительная разномасштабность сеточных элементов;
- сравнительно большое число элементов, принадлежащих геометрической границе сетки;
- сильная неоднородность физических процессов;
- расчет МГД по явной схеме, турбулентной вязкости по неявной схеме.

Расчет процесса формирования и распространения ударной волны в протяженном сооружении проводился на суперкомпьютере «Ломоносов» на 4096 ядрах. На Рис. 12 представлены изменение температуры в коллекторе по мере распространения ударной волны (вверху) и пространственное изменение температуры (внизу). Стрелками обозначена скорость движения частиц.

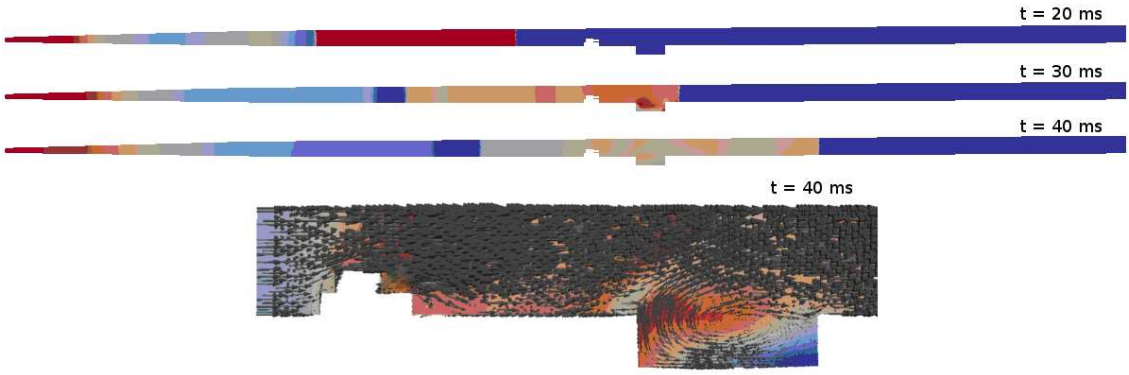


Рис. 12. Распространение ударной волны (вверху) и потоки газа в ударной трубе на поле температуры (внизу)

1.5 Программный комплекс MARPLE3D

В качестве инструмента исследования был выбран параллельный программный комплекс MARPLE3D [37]. Пакет прикладных программ MARPLE3D создан в ИПМ им. М.В.Келдыша РАН на основе численных методик с применением неструктурированных расчетных сеток. Предметной областью па-

кета являются задачи двухтемпературной радиационной магнитной гидродинамики. Для таких задач на каждом временном шаге необходимо вычислять несколько десятков физических величин в каждой ячейке расчетной сетки. Для моделирования существенно трехмерных процессов с учетом реальной геометрии устройств необходимы расчетные сетки, содержащие от нескольких миллионов до десятков и сотен миллионов ячеек. Решение задач такой размерности возможно только с использованием распределенных вычислений на базе современных высокопроизводительных ЭВМ. Поэтому пакет MARPLE3D был сконструирован как программное обеспечение для высокопроизводительных параллельных вычислительных систем. Для выполнения расчетов на массивно-параллельных системах с распределенной памятью применяется известный подход геометрического параллелизма, поддержка которого встроена в дискретную модель MARPLE3D на уровне структур сеточных данных. В пакет MARPLE3D встроена внутренняя система журналирования, с помощью которой и производились измерения (фиксировалось астрономическое время выполнения операций на каждом вычислительном узле). Тестирование выполнялось совместно с Дорофеевой Е.Ю. (ИПМ им. М.В.Келдыша РАН). Автор диссертации занималась разработкой алгоритмов и программ выполнения рационального разбиения графов расчетных сеток, а также оценкой качества разбиений, Дорофеева Е.Ю. – постановкой физических задач и проведением вычислительных экспериментов. Эффективность параллельного счета физических задач оценивалась совместно.

Глава 2. Алгоритмы декомпозиции

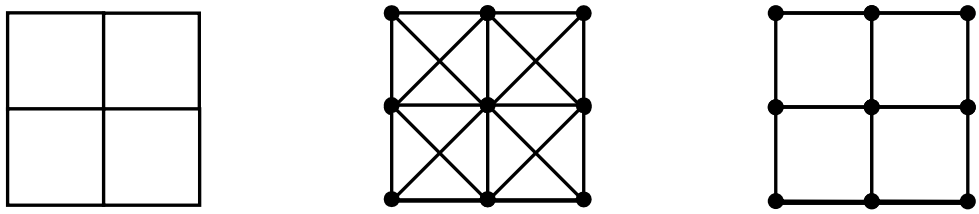
2.1 Модель декомпозиции

2.1.1 Построение графа по сетке

Для учета коммуникационных нагрузок сетки задача разбиения сетки на домены обобщается на задачу разбиения графа на домены. Для каждой сетки можно построить несколько графов.

Нодальный граф

В нодальном графе (nodal graph) вершины графа соответствуют узлам сетки, а ребра – связям между узлами [38]. В полном нодальном графе (complete nodal graph) между вершинами существует ребро, если соответствующие узлы связаны элементом (Рис. 13б). В сокращенном нодальном графе (reduced nodal graph) между вершинами есть ребро, только если соответствующие узлы связаны линией на границе элемента (Рис. 13в).



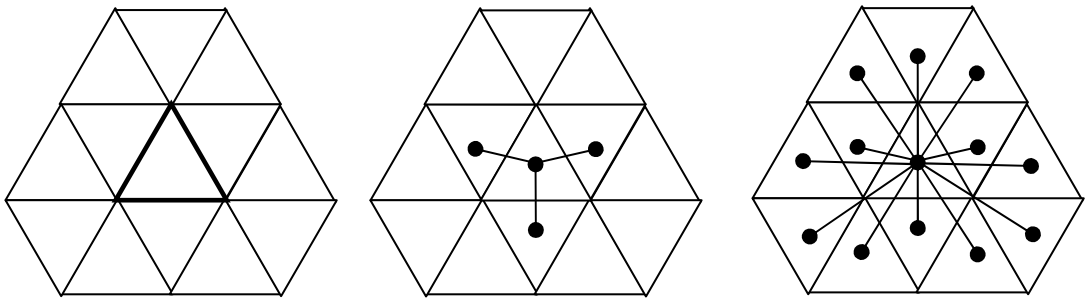
а б в

Рис. 13. Построение нодального графа, а) – исходная сетка, б) – полный нодальный граф, в) – сокращенный нодальный граф

Дуальный граф

В дуальном графе (dual graph) вершины представляют собой элементы сетки. В сокращенном дуальном графе (reduced dual graph) на Рис. 14б вер-

шины соединены ребром, если соответствующие элементы сетки имеют общую грань (для двумерной сетки – ребро). В полном дуальном графе (complete dual graph) на Рис. 14в между вершинами существует ребро, если соответствующие элементы имеют общие грань, ребро, или узел (для двумерной сетки – ребро, или узел).



а б в

Рис. 14. Построение дуального графа, а) – исходная сетка, б) – часть сокращенного дуального графа для выделенного элемента, в) – часть полного дуального графа для выделенного элемента

2.1.2 Модели декомпозиции графов

Существует несколько моделей декомпозиции графов [4, 31], отличающиеся видом графа и критериями сбалансированного разбиения: стандартная модель графа (standard graph model), модель двудольного графа (bipartite graph model), модель гиперграфа (hypergraph model), модель декомпозиции с несколькими ограничениями (multi-constraint partitioning), модель декомпозиции с несколькими целевыми функциями (multi-objective partitioning) и модель асимметричного разбиения (skewed partitioning).

В стандартной модели графа сетка аппроксимируется неориентированным графом $G = (V, E)$, где V – множество вершин, E – множество ребер. Веса вершин аппроксимируют вычислительную нагрузку в вершинах, веса ребер – коммуникационную нагрузку. Задача декомпозиции графа заключается

в таком разбиении вершин на домены, в котором сбалансирован вес вершин в доменах и минимизирован суммарный вес разрезанных ребер.

У метрики разрезанных ребер выделяют несколько недостатков. Первое, поскольку вершина с одного процессора может быть связана с несколькими вершинами с другого процессора, суммарный вес разрезанных ребер не пропорционален суммарному объему коммуникаций. Второе, эффективность параллельных приложений определяется работой самого медленного процессора. Поэтому имеет смысл минимизировать объем коммуникаций, или число сообщений, каждого процессора.

Стандартная модель графа имеет следующие ограничения. Во-первых, данная модель отражает только симметричные зависимости между данными, что является проблемой для итерационных методов решения дифференциальных уравнений с несимметричными матрицами. В статье [4] описывается несколько способов устранения этого ограничения: использование ориентированного графа и варьирование весов ребер. Во-вторых, в задаче параллельного умножения матрицы на вектор стандартная модель использует одну вершину для ряда и столбца с одинаковыми индексами. В результате невозможно представить неквадратные матрицы. В-третьих, стандартная модель графа непригодна для многофазных вычислений, возникающих, например, при постановке граничных условий, моделирующих различные явления в физических вычислениях.

Одной из альтернативных моделей декомпозиции графа является модель двудольного графа (bipartite graph model) [31, 39]. В ней вершины в графе $G = (V_1, V_2, E)$ разделены на два множества V_1 и V_2 так, что ни одно ребро не соединяет вершины из одного множества, все ребра связывают вершины из разных множеств. В задаче параллельного умножения матрицы на вектор данная модель использует одно множество вершин для представления строк, другое для столбцов, что позволяет описывать несимметричные неквадрат-

ные матрицы. Модель двудольного графа также хорошо описывает задачи с двумя вычислительными операциями, например, переход между фазами в двухфазных вычислениях.

Другая альтернатива – модель гиперграфа (hypergraph model), предложенная Catalyurek, Aykanat и Pinar в [40, 41, 42]. Гиперграф $G = (V, H)$ состоит из множества вершин V и множества гиперребер H. Каждое гиперребро включает в себя подмножество вершин из V. Таким образом, гиперребро является обобщением ребра в графе, когда может быть связано больше двух вершин. Задача декомпозиции гиперграфа заключается в таком разбиении множества вершин на домены одинакового веса, в котором минимизировано число разрезанных гиперребер. Чтобы построить гиперграф из графа, нужно в гиперребро включить вершину и всех ее соседей. Поскольку объем вычислений зависит от того, с каким числом соседних процессоров связана вершина, минимизация числа процессоров, разрезающих гиперребро, приводит к уменьшению объема коммуникаций, в отличие от метрики разрезанных ребер. Также гиперграф является привлекательной альтернативой модели двудольного графа в несимметричных задачах с одной операцией.

Многофазные вычисления хорошо описываются моделью разбиения с несколькими ограничениями (multi-constraint partitioning), предложенной Karypis и Kumar [43, 44]. Данная модель является расширением стандартной модели графа, когда каждая вершина в графе обладает вектором весов, k-ый вес соответствует нагрузке вершины на k-ой фазе вычислений. Ребра отображают зависимости между данными на всех фазах вычислений. Задача декомпозиции состоит в том, чтобы разбить вершины на домены так, чтобы минимизировать вес разрезанных ребер и сбалансировать каждый из весов вершин, тогда будет сбалансирована каждая фаза вычислений.

Похожей моделью является модель разбиения с несколькими целевыми функциями (multi-objective partitioning), предложенная Schloegel, Karypis и

Kumar [45]. В ней одновременно минимизируются несколько целевых функций. Каждая вершина обладает вектором весов, *j*-ый вес соответствует *j*-ой целевой функции. Цель разбиения состоит в такой балансировке весов вершин, при которой минимизирована каждая из целевых функций. Данная модель хорошо описывает многофазные вычисления, в которых на каждой фазе возникает свое распределение коммуникационной нагрузки.

Хотя модели разбиения с несколькими ограничениями и несколькими целевыми функциями очень привлекательны в плане общности, реализация декомпозиции в данных моделях сложна, поэтому по возможности используются более простые модели.

Еще одна альтернатива стандартной модели графа – модель асимметричного разбиения (skewed partitioning), которая также является расширением стандартной модели [46]. В ней каждой вершине присваивается набор из *k* значений, каждое из которых определяет относительное предпочтение попадания данной вершины в *k*-ый домен. Предпочтения учитываются вместе с коммуникационной метрикой при построении разбиения. Модель асимметричного разбиения используется, например, при динамической балансировке загрузки, когда наряду с минимизацией коммуникаций желательно ограничить количество перемещаемых вершин.

Несмотря на ограничения метрики разрезанных ребер стандартный подход доказал свою успешность при параллельном решении дифференциальных уравнений и задач, основанных на сетках, по нескольким причинам [4]. Первое, вершины в сетках обычно имеют сравнительно небольшое число соседей, поэтому число разрезанных ребер отличается на небольшой множитель от реального коммуникационного объема. Второе, вычислительные сетки обычно имеют высокую степень геометрической локальности, что гарантирует существование хорошего разбиения [47]. Если размер сетки *n* увеличивается, а число процессоров остается фиксированным, коммуникационный

объем растет как $n^{2/3}$ в трехмерном пространстве и $n^{1/2}$ в двумерном. Аналогично, геометрическая локальность ограничивает число сообщений, которое посылает каждый из процессоров. Из всего вышесказанного следует, что при решении задач на больших сетках важна вычислительная эффективность, а критерии определения коммуникационного объема не настолько критичны.

В данной работе сделан акцент на статической декомпозиции сетки, то есть декомпозиции, которая проводится один раз перед расчетом задачи. В отличие от нее, динамическая декомпозиция производится периодически в течение счета для балансировки загрузки, и на нее налагаются дополнительные ограничения, такие как быстрота выполнения и небольшие объемы перемещаемых данных. В статической декомпозиции сетки хорошо работает стандартная модель графа, поэтому в данной работе была выбрана именно она.

2.1.3 Модель декомпозиции

Пусть $G = (V, E)$ – неориентированный граф, где $V = \{v_i\}$ – множество вершин, $E = \{e_{ij}\}$ – множество ребер. И вершины, и ребра, имеют целочисленные веса $|v_i|$ и $|e_{ij}|$, соответственно. Декомпозиция является отображением множества вершин V на заданное число p доменов S_j такое, что каждая вершина v_i принадлежит некоторому домену S_j, и $S_k \cap S_m = \varnothing, \quad \bigcup_j S_j = V$. Суммарный вес вершин в домене S_j равен сумме весов вершин, принадлежащих этому домену: $|S_j| = \sum_i |v_i|, \; v_i \in S_j$. Суммарный вес разрезанных ребер равен сумме весов ребер, соединяющих вершины из разных доменов: $|E_c| = \sum_{ij} |e_{ij}|, \; v_i \in S_k, \; v_j \in S_m, \; k < m$. Каждый домен

S_j является некоторым подграфом $G^j = \left(V^j, E^j\right)$ графа $G = (V, E)$, где $V^j \subseteq V, E^j \subseteq E$. Домен является связным, если его граф состоит из одной компоненты связности, то есть между любой парой вершин этого графа существует как минимум один путь. Требуется найти такое разбиение множества вершин V на заданное число p связных доменов S_j, при котором выровнен суммарный вес вершин в доменах ($\left|S_j\right| \approx \left|V\right| / p$) и минимизирован суммарный вес разрезанных ребер ($\min\left|E_c\right|$).

В данной модели суммарный вес вершин в доменах отвечает за равномерность разбиения сетки на домены (а в дальнейшем равномерность распределения по процессорам, которые будут обрабатывать эти домены), а суммарный вес разрезанных ребер – за коммуникационную нагрузку между процессорами. Связность критична для задач, упомянутых ранее.

2.2 Обзор и анализ существующих алгоритмов

2.2.1 Простые алгоритмы разбиения сеток

Наиболее простыми алгоритмами разбиения сеток являются алгоритмы, учитывающие только номера вершин. К таким алгоритмам относятся линейное распределение вершин (linear method), рассеивание (scattered method) и случайное распределение (random method) [21]. При линейном распределении непрерывный диапазон номеров вершин разбивается на интервалы, вершины из одного интервала попадают в один домен. При рассеивании номер вершины, взятый по модулю числа доменов, определяет, в какой домен попадет вершина. То есть сначала в каждый домен по очереди распределяется по одной вершине, потом по две и т.д. При случайном распределении домен вершины выбирается случайным образом среди доменов, в которых вершин меньше, чем должно быть при равномерном распределении. Описанные ме-

тоды включены в пакеты CHACO [5] и PARTY [21, 48]. Достоинством данных алгоритмов является быстрота выполнения. Поэтому они используются для получения предварительного разбиения сетки, когда декомпозиция сетки выполняется на распределенной вычислительной системе, и перед вычислением самой декомпозиции сетку нужно неким образом распределить между процессорами. Однако разбиения, получаемые данными методами, не учитывают геометрию сетки, поэтому в качестве предварительного разбиения они подходят только для тех алгоритмов, результат работы которых не зависит от качества начального распределения вершин по процессорам.

2.2.2 Алгоритмы геометрической декомпозиции сеток

Более сложными, но довольно быстрыми, алгоритмами разбиения сетки являются алгоритмы, учитывающие геометрическую информацию о сетке. К ним относятся методы рекурсивных координатной бисекции (recursive coordinate bisection), инерциальной бисекции (recursive inertial bisection) и разбиение с использованием кривой Гильберта (Hilbert space-filling curve partitioning) [5, 6, 49, 50, 51, 52].

На каждом этапе рекурсивной бисекции область разбивается на две части. Соотношение размеров частей зависит от количества доменов, которые должны быть образованы в каждой из частей. Полученные подобласти разбиваются дальше аналогичным образом до тех пор, пока в подобластях не останется по одному домену. При рекурсивной координатной бисекции на этапе разбиения выбирается координатная ось, вдоль которой область имеет наибольшую протяженность. Область разбивается перпендикулярно полученной оси. Рекурсивная инерциальная бисекция отличается от координатной тем, что выбираемая ось не обязана быть параллельной одной из осей координат. Вычисляется ось инерции, которая обычно является направлением, вдоль которого область имеет наибольшую протяженность.

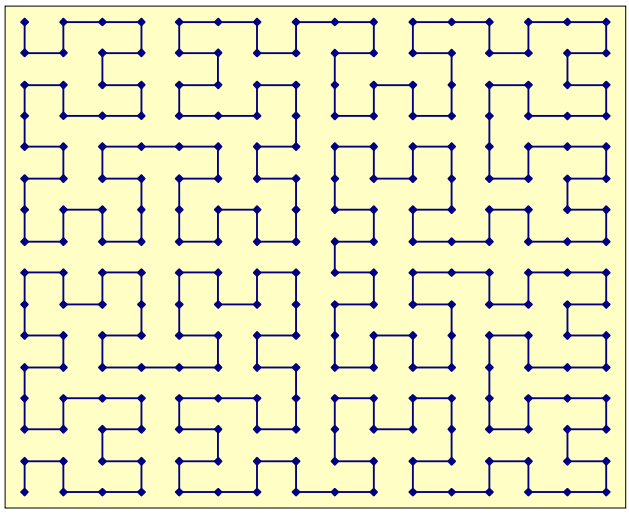


Рис. 15. Кривая Гильберта 4-го порядка на 2-мерной области

Кривая Гильберта – непрерывная кривая в смысле Жордана (кривая является непрерывным образом отрезка), целиком заполняющая некоторый гиперпараллелепипед (Рис. 15). Иначе говоря, кривая Гильберта всюду плотна в области, в которой построена, то есть проходит через любую сколь угодно малую окрестность каждой точки этой области [53, 54, 55, 56]. Кривой Гильберта отрезок [0; 1] отображается на n-мерный гиперпараллелепипед. Точки, расположенные близко на отрезке, находятся на близком расстоянии в гиперпараллелепипеде. Алгоритм получения разбиения с использованием кривой Гильберта состоит из следующих частей [6, 9]:

- построение кривой Гильберта, отображающей отрезок [0; 1] на наименьший гиперпараллелепипед, содержащий все вершины;
- отрезок [0; 1] разбивается на малые отрезки [left, right), число которых превышает число формируемых доменов (в последний отрезок включается единица);
- слева направо производится суммирование весов вершин в малых отрезках, в результате чего из малых отрезков набираются большие отрезки, соответствующие доменам;

Геометрические методы входят в пакеты CHACO, PARTY, PARMETIS [57, 58, 59] и ZOLTAN [60]. В отличие от простых алгоритмов, геометриче-

ские алгоритмы позволяют получать более компактные домены, которые связаны с меньшим числом соседей. Хотя эти алгоритмы не учитывают коммуникационные нагрузки сетки, они используются как для получения предварительных разбиений при распределении вершин по процессорам в других параллельных алгоритмах, так и тогда, когда важны быстрота получения разбиения и экономность использования памяти. Разбиения, полученные геометрическими методами, отличаются меньшим дисбалансом числа вершин в доменах, чем полученные алгоритмами декомпозиции графов. Поэтому геометрические алгоритмы используются еще и тогда, когда сбалансированность разбиения важнее коммуникационных нагрузок.

2.2.3 Алгоритмы декомпозиции графов

2.2.3.1 Спектральная бисекция

В алгоритме спектральной бисекции (spectral bisection) для разбиения графа используется спектральная информация. Строится матрица Лапласа $Q = A - D$, где D – диагональная матрица, A – матрица смежности графа [7, 8]:

$$a_{i,j} = \begin{cases} ew(v_i, v_j), \text{ если } (v_i, v_j) \in E, \\ 0 \qquad\quad , \text{ иначе}, \end{cases}$$

$$d_{i,i} = \sum_j ew(v_i, v_j) \text{ для всех } (v_i, v_j) \in E,$$

где $ew(v_i, v_j)$ – вес ребра (v_i, v_j), E – множество ребер. Вычисляется собственный вектор, соответствующий второму наибольшему собственному значению. Этот вектор называется вектором Фидлера (Fiedler vector), и его величина является мерой связей графа. Далее множество вершин графа V упорядочивается по не убыванию значений компонент вектора Фидлера y:

$y_{v_i} \le y_{v_j}, V_1 = \{v_i, v_j, ..\}$. Затем определяется взвешенная медиана множества V_1, и первая часть вершин распределяется в первый домен, вторая – во второй. То есть в первый домен попадают вершины с меньшими соответствующими значениями компонент вектора Фидлера. Вектор Фидлера вычисляется алгоритмом Lanczos [61]. Это итеративный алгоритм, число итераций которого зависит от желаемой точности [7]. Аналогичным образом можно разбить множество вершин на четыре домена, используя второй и третий собственные вектора, и на восемь частей, используя второй, третий и четвертый собственные вектора [5, 62].

Рекурсивная спектральная бисекция включена в пакеты METIS, CHACO, PARTY (через интерфейс к библиотеке CHACO), а также в TOP/DOMDEC [63], HARP [64] и S-HARP [65, 66] (параллельная версия HARP). Алгоритм производит разбиения неструктурированных сеток, лучшие по качеству, чем получаемые геометрическими методами, но его вычислительная стоимость зависит нелинейно от размера сетки, поэтому алгоритм рекурсивной спектральной бисекции в основном используется в рамках иерархического подхода.

2.2.3.2 Алгоритм локального уточнения

Изначально алгоритм локального уточнения был разработан Kernighan и Lin (KL), как алгоритм бисекции графа [67]. Затем Fiduccia и Mattheyses (FM) предложили линейную реализацию данного алгоритма [68]. Поэтому обычно алгоритм локального уточнения называется KL/FM алгоритмом. Hendrickson и Leland усовершенствовали данный алгоритм, распространив его на произвольное число доменов, и реализовали в пакете CHACO [10]. Они также ввели случайность в алгоритм, что сделало его более устойчивым, и отказались от пересчета всех структур на каждой итерации внешнего цикла при переме-

щении малой доли вершин, что ускорило работу алгоритма. Именно алгоритм локального уточнения Hendrickson и Leland описан ниже.

В основе алгоритма лежит понятие выигрыша от перемещения вершины v_i из домена l в домен k:

$$g^k(v_i) = \sum_{(v_i, v_j) \in E} \begin{cases} ew(v_i, v_j), \textit{если } P(v_j) = k, \\ -ew(v_i, v_j), \textit{если } P(v_j) = l, \\ 0, \textit{в остальных случаях}, \end{cases}$$

где $g^k(v_i)$ – выигрыш от перемещения вершины v_i из домена l в домен k, $ew(v_i, v_j)$ – вес ребра (v_i, v_j), E – множество ребер, $P(v_j)$ – домен, которому принадлежит вершина v_j. Таким образом, выигрыш от перемещения вершины – это уменьшение общего веса разрезанных ребер, полученное от перемещения этой вершины из одного домена в другой.

Сам алгоритм состоит из двух циклов, внешнего и внутреннего:

Внешний цикл

приравнивание лучшего разбиения к текущему разбиению

вычисление начальных выигрышей

Внутренний цикл

выбор перемещения с наибольшим выигрышем

перемещение

обновление выигрышей у соседей перемещенной вершины

запоминание лучшего разбиения

Конец внутреннего цикла

приравнивание текущего разбиения к лучшему разбиению, полученному во внутреннем цикле

Конец внешнего цикла.

На первой итерации внешнего цикла вычисляются выигрыши от перемещений всех вершин в другие домены. На каждой итерации внутреннего цикла выбирается перемещение с наибольшим выигрышем. При этом учитывается критерий балансировки: вершины перемещаются только из доменов, размер которых больше среднего, в домены, размер которых, меньше, или равен, среднему. Каждую вершину можно перемещать только один раз в течение внутреннего цикла, что предотвращает циклическое перемещение одной вершины между несколькими доменами. Наибольший выигрыш может оказаться отрицательным. Важным качеством KL алгоритма является способность выбираться из локальных минимумов, то есть несколько перемещений с отрицательными выигрышами могут, в конечном счете, привести к разбиению с меньшим общим весом разрезанных ребер. После каждого перемещения обновляются выигрыши соседей перемещенной вершины. В течение всего внутреннего цикла запоминается лучшее разбиение, полученное во внутреннем цикле. Критерии завершения внутреннего цикла могут быть разными [10, 69, 1, 7]. Например, когда больше не существует перемещений из больших доменов в меньшие. Или когда получена слишком большая разница между качеством текущего разбиения и наилучшего разбиения, полученного к данному моменту во внутреннем цикле. Этот критерий используется в иерархическом подходе, когда считается, что локальное уточнение не должно производить слишком большие изменения полученного разбиения. Перед началом следующей итерации внешнего цикла восстанавливается наи-

лучшее разбиение, полученное во внутреннем цикле. Для ускорения процесса пересчитываются не все выигрыши, а только выигрыши вершин, перемещенных во внутреннем цикле, и их соседей. Внешний цикл завершается, когда его несколько итераций не привели к получению лучшего разбиения.

Поскольку обычно существует несколько перемещений с одинаковыми выигрышами, начальные выигрыши на первой итерации внешнего цикла подсчитываются в случайном порядке. В дальнейшем при пересчете выигрыша от перемещения вершины, этот выигрыш попадает на первое место среди таких же выигрышей. Поскольку обновляются выигрыши у всех соседей перемещенной вершины, их выигрыши рассматриваются вместе, что соответствует духу всего алгоритма.

Время выполнения алгоритма локального уточнения $O((s-1) \cdot m)$ [10], где s – число доменов, m – число ребер в графе. И хотя алгоритм рассчитан на произвольное число доменов, он плохо масштабируется по числу доменов и числу вершин [5].

Качество разбиений получаемых алгоритмом локального уточнения сильно зависит от выбираемого начального разбиения, поэтому его ценность не в использовании алгоритма самого по себе, а в улучшении разбиений, получаемых другими алгоритмами, например, в иерархических методах.

Алгоритм локального уточнения используется в таких пакетах, как METIS (PARMETIS), CHACO, PARTY и JOSTLE. В них реализовано несколько вариаций данного алгоритма. В частности, в алгоритме локального уточнения пакета JOSTLE рассматриваются только вершины на границах между доменами [1], и добавлена балансировка загрузки, использующая диффузионный алгоритм, разработанный Hu и Blake [70].

Один из алгоритмов локального уточнения из пакетов METIS и PARMETIS, – жадное уточнение (greedy refinement) [71, 72]. Этот алгоритм,

как и остальные, используется в рамках иерархического подхода, который подробно описан ниже. В жадном уточнении предполагается, что перемещение одной вершины огрубленного графа аналогично перемещению группы вершин исходного графа, поэтому можно не ожидать улучшения разбиения после нескольких перемещений, а перемещать только вершины с положительным выигрышем. В результате, во внутреннем цикле все вершины навещаются в случайном порядке, и если вершина лежит на границе между доменами, она перемещается:

- в домен с наибольшим выигрышем, если выигрыш положительный и не нарушается сбалансированность разбиения;

- в домен с нулевым выигрышем, если баланс улучшается;

- если таких доменов нет, вершина не перемещается.

Алгоритм имеет меньшую зависимость от числа доменов, но теряет способность выбираться из локальных минимумов.

В последовательном пакете PARTY реализован алгоритм полезных множеств (helpful-sets), который очень похож на алгоритм локального уточнения, только работает не с выигрышами отдельных вершин, а с целыми наборами [21, 73].

2.2.3.3 Иерархические алгоритмы

Иерархические алгоритмы состоят из следующих основных частей: поэтапное огрубление графа, декомпозиция самого маленького из полученных графов и отображение разбиения на предыдущие графы с периодическим локальным уточнением границ доменов.

Огрубление графа

На стадии огрубления граф аппроксимируется цепочкой графов с меньшим числом вершин. В основе огрубления лежит понятие стягивания ребра [10]. Когда ребро стягивается, две вершины лежащие на его концах превращаются в одну с весом, равным сумме весов стягиваемых вершин. На Рис. 16 первоначальный граф состоит из трех вершин, веса всех вершин и ребер равны единице. Стягивается ребро между вершинами с номерами 1 и 2, на рисунке оно покрашено в синий цвет. В результате получается граф, состоящий из двух вершин с весами 1 и 2. Веса ребер остаются неизменными, за исключением случая, когда обе стягиваемые вершины связаны с одним соседом. В этом случае новое ребро замещает собой два, и его вес становится равным сумме весов соответствующих ребер. На Рис. 16 обе вершины 1 и 2 связаны с вершиной с номером 0. Вес итогового ребра между вершиной 0 и новой вершиной становится равным двойке.

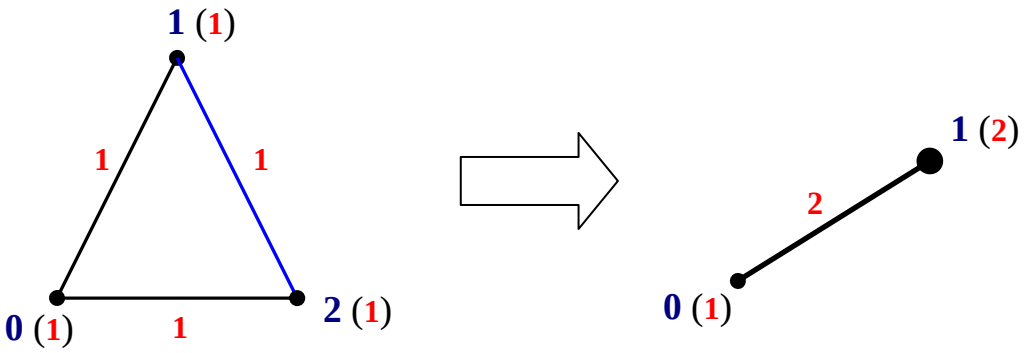


Рис. 16. Стягивание ребра

Поскольку обычно граф состоит из множества ребер, вначале находится так называемое максимальное покрытие (maximal matching), то есть набор ребер, никакие два из которых не связаны с одной вершиной, и в набор нельзя добавить ни одного ребра без нарушения этого свойства. Наиболее простой способ нахождения такого покрытия (он используется в пакете CHACO [10]) - проход по всем вершинам в случайном порядке. Если вершина не была

отмечена ранее, случайным образом выбирается ее неотмеченный сосед, и ребро помещается в максимальное покрытие стягиваемыми ребрами. Более эффективным способом выбора соседа является выбор ребра с наибольшим весом [7, 12], такой способ используется в пакетах METIS (PARMETIS) и JOSTLE. В таком случае разрезанными ребрами окажутся менее тяжелые ребра. В пакете JOSTLE при выборе среди одинаково тяжелых ребер предпочтение отдается соседу с наименьшей степенью (имеет наименьшее число соседей), во избежание получения сильно связанных вершин. В пакете WGPP [74, 75] используется комбинация огрублений по самому тяжелому ребру и самому тяжелому треугольнику (имеет наибольшую сумму весов трех ребер).

Далее производится непосредственное стягивание ребер и определение весов вершин и ребер полученного графа. На Рис. 17 и 18 представлены исходный граф и результаты двух его последовательных огрублений. Размер вершины на рисунках увеличивается с увеличением ее веса, чем больше вес ребра, тем более красного оттенка его цвет. У исходного графа веса всех вершин и ребер равны единице.

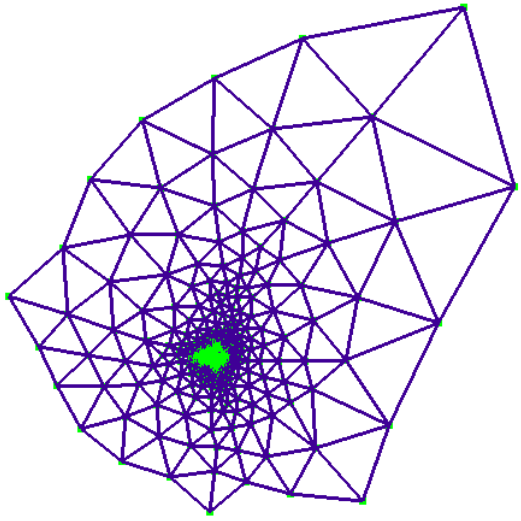


Рис. 17. Исходный граф

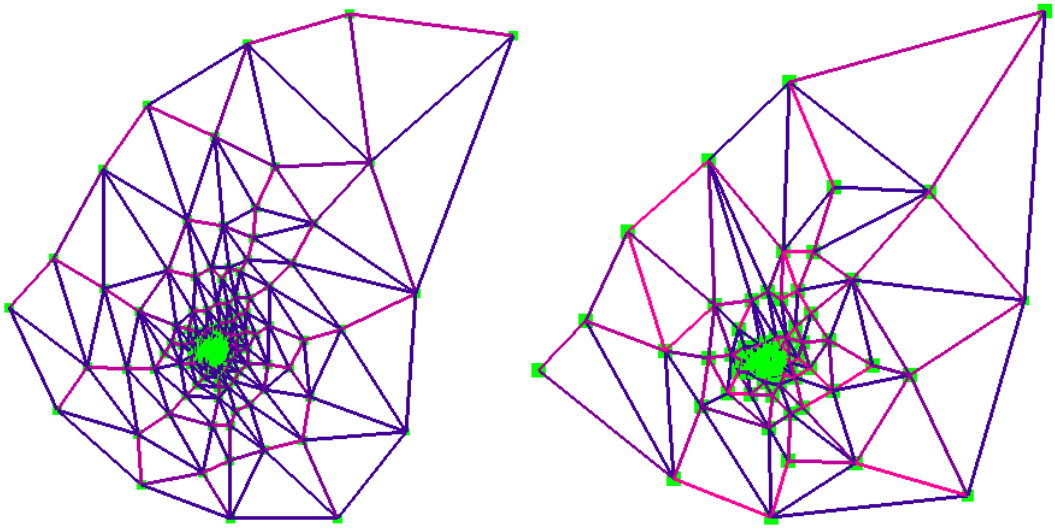


Рис. 18. Результаты первого и второго огрублений

На каждой стадии огрубления число вершин уменьшается в лучшем случае вдвое. Огрубление графа имеет несколько важных свойств. Первое, каждая вершина при огрублении переходит в одну вершину, поэтому легко спроектировать разбиение огрубленного графа в разбиение исходного графа: вершины исходного графа распределяются в те домены, в которые попали вершины, порожденные ими. Второе, поскольку веса вершин суммируются, сохраняются ограничения на размеры доменов, которые зависят только от суммарного веса вершин, распределенных в каждый из доменов. Третье, так как суммируются веса ребер, сохраняется функция стоимости, зависящая от весов разрезанных ребер. Таким образом, для большинства метрик качества разбиений хорошее разбиение огрубленного графа, скорее всего, окажется хорошим и для исходного графа.

Итак, на стадии огрубления проводятся несколько огрублений графа до тех пор, пока размер графа не станет меньше порогового, или когда уменьшение числа вершин окажется меньше заданного.

Разбиение огрубленного графа

Алгоритмы получения самого разбиения могут быть разными, однако они должны поддерживать веса вершин и ребер для возможности работы с ог-

рубленным графом. Часто на данной стадии используется алгоритм спектральной бисекции (CHACO, METIS) [10, 7], поскольку он дает качественные разбиения, а иерархический подход позволяет разбивать не большой исходный граф, а маленький огрубленный, что, в силу нелинейной зависимости времени работы этого алгоритма от размера сетки, критично. Еще одним способом получения разбиения является исключение данной стадии путем огрубления графа до числа вершин, равного числу доменов. Однако на заключительных стадиях огрубление дает все меньший выигрыш в уменьшении числа вершин, и такой вариант может оказаться невозможным. Если же огрубление удастся, веса полученных доменов окажутся сильно несбалансированными. Однако именно такой способ используется в пакете JOSTLE [2, 12], и проблема дисбаланса решается на этапе локального уточнения.

Восстановление разбиения

Каждая из вершин огрубленного графа является объединением одной, или двух вершин, исходного графа. Поэтому домен вершины огрубленного графа становится доменом ее прообразов в исходном графе. И так по всей цепочке графов. Поскольку вес вершины огрубленного графа является суммой весов составляющих ее вершин исходного графа, в процессе восстановления графа вес вершин в доменах сохраняется. Аналогичным образом сохраняется суммарный вес разрезанных ребер.

Локальное уточнение разбиения

Восстановленный граф имеет больше степеней свободы, чем его огрубленный вариант, поэтому лучшее разбиение огрубленного графа не обязательно окажется оптимальным для его предшественника. Для нахождения оптимального разбиения через каждые несколько этапов восстановления графа производится локальное уточнение доменов либо алгоритмом KL/FM, описанным ранее, либо диффузионным, либо генетическим алгоритмами, ко-

торые будут рассмотрены далее. Локальное уточнение KL/FM используется в пакетах CHACO, METIS, PARMETIS, JOSTLE, SCOTCH [76], диффузионный алгоритм – в пакете PT-SCOTCH (параллельная версия пакета SCOTCH) [13, 28, 16].

Параллельные реализации иерархических алгоритмов

Во всех существующих параллельных пакетах декомпозиции сеток используются именно иерархические алгоритмы, поэтому хочется остановиться подробнее на вопросах параллельной реализации данных алгоритмов.

Одной из первых реализаций иерархического подхода был алгоритм PMRSB, алгоритм параллельной рекурсивной спектральной бисекции [78, 79]. Данный вариант алгоритма отличался от той структуры иерархических методов, которая описана в данном параграфе. В нем вычислялся вектор Фидлера огрубленного графа, после чего проводилась интерполяция для формирования приближений к векторам Фидлера предыдущих графов с уточнением обратной итерацией. В результате формировался вектор Фидлера исходного графа, и дальше действия проводились в соответствии с обычным алгоритмом спектральной бисекции. Алгоритм PMRSB был реализован на Cray платформах.

В остальных пакетах параллельной декомпозиции сеток реализован традиционный вариант иерархического алгоритма, и используется библиотека MPI.

Одним из наиболее используемых параллельных пакетов декомпозиции сеток является пакет PARMETIS. В нем после огрубления все части графа собираются на всех процессорах [11, 80, 81]. При огрублении графа меньше некоторого порога, дальнейшее огрубление производится на половине процессоров, чтобы избавиться от задержек при отправке сообщений на маленьких графах, когда эти задержки начинают доминировать [72, 78, 82, 83]. Раз-

биение вычисляется алгоритмом рекурсивной бисекции, основанным на вложенном рассечении и жадном локальном уточнении [84]. Недостатком разбиений, получаемых пакетом PARMETIS, является получение сильно несбалансированных разбиений при разбиении на большое число доменов.

В пакете JOSTLE граф огрубляется параллельно до того момента, как его можно будет поместить на один процессор [12]. После сбора на одном процессоре граф огрубляется до числа вершин, равного числу доменов, стадия разбиения исключается. После огрубления веса полученных вершин могут сильно отличаться, но проблема дисбаланса решается на этапе локального уточнения. Считается, что больший дисбаланс вершин после огрубления позволяет найти разбиение лучшего качества в процессе локального уточнения [1]. В пакете JOSTLE, в отличие от пакета PARMETIS, на каждом процессоре образовывается только один домен, поскольку пакет рассчитан больше на динамическую декомпозицию [85, 86].

В пакете PT-SCOTCH декомпозиция графа является частным случаем отображения графа процессов программы на граф архитектуры [13]. Огрубление графа позволяет собрать его на одном процессоре и там разбить диффузионным алгоритмом. Поскольку обычный вариант локального уточнения KL/FM плохо масштабируем по числу процессоров, что ухудшает его свойства вылезания из локальных минимумов, на этапах локального уточнения используется граничный вариант диффузионного алгоритма. Вообще граничный метод в пакете PT-SCOTCH используется в качестве надстройки над другими методами. В нем рассматриваются только вершины на расстоянии от границы, не превышающем заданное, что позволяет использовать дорогостоящие методы путем рассмотрения только части вершин. Обычно берут расстояние в три вершины от границы. Считается, что граничный вариант при использовании с локальным уточнением, стартуя от хорошего разбиения грубого графа, не позволяет алгоритму зациклиться на локальных миниму-

мах восстановленного графа [13]. В статье [16] предполагается, что результат граничного локального уточнения не будет сильно отличаться от результата обычного локального уточнения, потому как локальный минимум можно найти на следующих уровнях восстановления графа. То есть на нескольких уровнях можно скомпенсировать то, что не удалось сделать на одном. В процессе восстановления графа наступает момент, когда граф снова распределен по процессорам, однако подграфы границы собираются на каждом из процессоров, и локальное уточнение происходит на процессорах независимо. Выбирается наилучший вариант, и он проектируется дальше.

В пакете ZOLTAN при разбиении графов либо используются интерфейсы к алгоритмам разбиений графов пакетов PARMETIS, JOSTLE и PT-SCOTCH, либо разбивается соответствующий гиперграф алгоритмом пакета ZOLTAN [6, 87]. При формировании гиперграфа вершины остаются такими же, как и в графе. Для каждой вершины формируется гиперребро, которое состоит из всех соседей вершины и ее самой.

2.2.3.4 Диффузионные и генетические алгоритмы

Диффузионные алгоритмы

Для большого класса задач на нерегулярных сетках, решаемых параллельно, их вычислительная структура постепенно меняется в процессе счета. Например, при использовании адаптивных сеток [88] некоторые области огрубляются, другие сгущаются для получения более точных результатов. В итоге меняются веса вершин и ребер в графе, что вызывает дисбаланс существующего разбиения, и возникает необходимость подкорректировать разбиение и перераспределить нагрузку между процессорами. В динамических алгоритмах разбиения важны быстрота получения разбиения и небольшая миграция вершин между процессорами в соответствии с полученным разбиением. Именно в динамических разбиениях в основном используются диффу-

зионные алгоритмы. Это объясняется тем, что по своей сути диффузионные алгоритмы являются инкрементными, что обеспечивает небольшую миграцию вершин между процессорами при перемещении их в соответствии с полученным разбиением. Поскольку после первоначального разбиения сетка уже была распределена по процессорам в соответствии с полученным разбиением, и идет счет задачи, при динамических балансировках загрузки она перебивается на том же числе процессоров, равном числу доменов. Для ускорения сходимости диффузионные алгоритмы обычно применяют в иерархических методах. Далее описывается параллельный иерархический диффузионный алгоритм динамического разбиения адаптивных сеток пакета PARMETIS, который похож на аналогичный алгоритм в пакете JOSTLE.

В соответствии с обсуждаемым алгоритмом, сначала производится огрубление графа, в результате которого граф аппроксимируется цепочкой графов с меньшим числом вершин [89, 14]. Затем происходит диффузия вершин на самом маленьком из графов, после чего разбиение уточняется на цепочке графов. В алгоритме используется два вида диффузии: неориентированная и ориентированная. При неориентированной диффузии вершины на границах доменов навещаются в случайном порядке. Вершина переполненного домена перемещается в домен с меньшим весом, а если таких несколько, то в тот домен, перемещение в который обеспечит меньший итоговый вес разрезанных ребер. Вершина из домена среднего веса перемещается в домен, который не переполнится при уменьшении веса разрезанных ребер. Если таких доменов несколько, то выбирается перемещение, обеспечивающее наиболее сильное уменьшение веса разрезанных ребер. Вершины из незаполненных доменов не перемещаются. Процесс продолжается несколько итераций до наступления баланса, или прекращения улучшения баланса.

При ориентированной диффузии в соответствии с алгоритмом Hu и Blake [90] вычисляется матрица, определяющая, какой вес вершин должен быть

перемещен между соседними доменами. Вершины на границах доменов навещаются в случайном порядке. Если вершина соседствует с доменами, в которые, в соответствии с матрицей, нужно перенести нагрузку, она переносится в тот из них, который даст больший выигрыш в весе разрезанных ребер. Матрица обновляется. Когда проверены все вершины на границах доменов, процесс повторяется до тех пор, пока не будет достигнут баланс, или прекратится его улучшение. И при ориентированной, и неориентированной диффузии может оказаться невозможным сбалансировать самый грубый граф, потому как там может быть недостаточно подходящих вершин. Тогда граф восстанавливается на один уровень, и процесс диффузии начинается на этом уровне.

Локальное уточнение напоминает неориентированную диффузию и его целью является уменьшение суммарного веса разрезанных ребер и миграции вершин между процессорами.

В параллельной версии ориентированная диффузия производится только на самом маленьком из графов последовательно. Граф копируется на все процессоры. Поскольку в диффузию включена случайность, каждый процессор получает свой результат. Выбирается лучший из них. На остальных графах, если не удалось сбалансировать более грубые графы, производится параллельно неориентированная диффузия.

В пакете PT-SCOTCH диффузионный алгоритм внедрен в иерархический подход несколько иным способом, упомянутым ранее [13, 76, 77, 28]. Подразумевается статическое разбиение, то есть разбиение, производимое один раз перед началом расчета. Диффузионные методы позволяют построить разбиения с небольшими и гладкими границами. Однако диффузионные алгоритмы дорогостоящие по времени, и использование их в иерархических методах позволяет избежать этого недостатка.

Диффузионный алгоритм пакета PT-SCOTCH выглядит следующим образом. Вершины в графе представляются бочками, а ребра трубами. В систему заливаются из инициирующих бочек две жидкости, scotch и anti-scotch (описывается бисекция). Граница между доменами определяется границей между вершинами, содержащими различные жидкости. Из каждой бочки в единицу времени вытекает в качестве потерь количество жидкости, равное ее весу, что не позволяет каждой из жидкостей заполнить больше половины бочек. Пропускная способность труб равна весам соответствующих ребер. Из каждой бочки оставшаяся от потерь жидкость передается соседям. Когда встречаются одинаковые количества обеих жидкостей, они исчезают. Полного схождения алгоритма не ждут, а ждут только стабильного определения того, в какой бочке, какая жидкость доминирует. Алгоритм сходится, так как $|V|$ (суммарный вес вершин) обеих жидкостей заливается в целом в единицу времени в систему, и все бочки могут терять такое же суммарное количество жидкости за то же время.

В граничном варианте описываемого диффузионного алгоритма рассматриваются только вершины, находящиеся на расстоянии в три вершины от границы между доменами. Две дополнительные вершины-якоря заменяют остальные части графа. Вес каждой вершины-якоря равен суммарному весу удаленных вершин с соответствующей стороны от границы между доменами. Вершины-якоря соединяются с граничным графом. Эти вершины и являются инициирующими бочками. Можно представить, что граница между доменами является периметром кругов вершин, завуалированных в каждой инициирующей бочке. Тогда жидкости заливается как бы из центров кругов. Таким образом, просто решен вопрос определения источников жидкостей, который является дорогостоящей операцией при работе со всеми вершинами в графе.

Диффузионные алгоритмы хороши в рамках иерархического подхода при выполнении динамической декомпозиции, поскольку вызывают малую ми-

грацию вершин между процессорами. В статической декомпозиции при использовании на этапах локального уточнения в иерархических алгоритмах они обеспечивают формирование доменов с гладкими границами.

Генетические алгоритмы

Генетические алгоритмы позволяют решать многокритериальные оптимизационные задачи, используя эволюционный подход. Это итерационные методы, модулирующие эволюцию популяций индивидуумов, которые представляют собой решения задачи, выбирая наиболее подходящих индивидуумов для рождения следующих поколений. Генетические алгоритмы сходятся медленно [15, 91], поэтому в задаче декомпозиции графа они обычно используются в иерархических методах. Разработчики пакета PT-SCOTCH реализовали на общей памяти иерархический метод, в котором граничный вариант генетического алгоритма вызывается на этапе локального уточнения [16, 94]. В обсуждаемом генетическом алгоритме в задаче декомпозиции графа каждая вершина может находиться в одном из трех состояний: лежать на границе между доменами, или в одном из двух доменов. Таким образом, каждый индивидуум в популяции представляется линейным массивом, модулирующим хромосому, который сопоставляет каждой вершине число от нуля до двух. Оператор скрещивания применяется к случайно выбранной позиции в массивах двух индивидуумов. Он меняет местами значения в данной позиции для порождения двух потомков. Оператор мутации меняет местами состояния случайно выбранных вершин у некоторых индивидуумов. Поскольку операторы не гарантируют, что после скрещивания, или мутации, индивидуумы будут отображать возможные решения, затем индивидуумы проверяются на корректность, в результате чего некоторые вершины помещаются на границу между доменами, а другие наоборот оттуда удаляются. Индивидуумы оцениваются функцией пригодности, которая комбинирует такие безразмерные величины, как доля вершин на границе между доменами, дисбаланс числа вер-

шин в доменах и доля ребер, соединяющих вершины на границе между доменами. Первое поколение составляется из индивидуумов, мутированных от спроектированного разбиения, и случайных индивидуумов, обеспечивающих генетическое многообразие. Для выбора и мутации индивидуумов используются различные алгоритмы [92, 93]. Затем индивидуумы подбираются в пары по убыванию пригодности, и рождают потомство, сохраняя популяцию постоянной. В статье [16] утверждается, что хотя данный алгоритм медленнее традиционных иерархических алгоритмов с локальным уточнением KL/FM, они лучше масштабируются по числу процессоров и дают более качественные результаты. К тому же, их качество может легко варьироваться (в терминах размера популяций и числа поколений) для учета ограничений на время выполнения и качество разбиений. Однако алгоритмы были реализованы на общей памяти и максимальное число потоков, которое использовалось – 64, что совсем немного.

Разработчики пакета JOSTLE реализовали последовательную комбинацию метода эволюционного поиска и иерархического подхода [17]. В данной комбинации конструируется популяция вариантов исходного графа, отличающихся только весами ребер. Иерархический метод используется в качестве оператора, как «черный ящик», для определения пригодности вариантов путем вычисления разбиения каждого из вариантов. Каждое из разбиений может оказаться также хорошим разбиением исходного графа. Популяция эволюционирует либо мутацией индивидуумов, либо их скрещиванием для рождения отличающегося, и предположительно, более пригодного ребенка. Полученные результаты оказались лучше результатов разбиения пакетами JOSTLE, METIS и CHACO. Однако небольшие сетки (10^4 вершин) считались часами, и даже днями, поэтому разработчики не стали реализовывать параллельную версию алгоритма.

Таким образом, использование генетических алгоритмов в рамках иерархического подхода дает лучшие разбиения, однако является чересчур затратным по времени.

2.2.3.5 Оптимизация характеристических отношений доменов

Как уже отмечалось, недостатком иерархических алгоритмов является образование доменов, границы которых состоят из неоптимальных наборов сегментов [18, 28]. В частности, домены могут оказаться несвязными. Такое ухудшение качества доменов для некоторых задач является критичным. Например, на доменах с длинными границами, или сложной конфигурацией, алгоритмы решения систем линейных уравнений сходятся за большее число итераций. Диффузионные алгоритмы, встроенные в иерархический метод, позволяют получать домены с небольшими и гладкими границами. Еще одной попыткой решения данной проблемы стал критерий оптимизации характеристических отношений доменов (aspect ratio). Выполнение этого критерия стремится сделать домены шарообразной формы, в результате чего они получаются более компактными. Первые шаги в оптимизации характеристических отношений доменов были в выставлении весов вершин в зависимости от расстояния до центра домена и включении этих весов в функции стоимости итерационных алгоритмов, таких как KL и других.

Варианты вычисления характеристического отношения домена могут быть разными:

$L_{\max} / L_{\min}$ - отношение самого длинного ребра на границе домена к самому короткому [18],

R_o^2 / R_i^2 - отношение площади наименьшего описанного круга к площади наибольшего вписанного,

R_o^2 / S, где S - площадь домена,

$C^2/16S$, C - периметр домена, это отношение площади квадрата с периметром C к площади домена,

$C/4\sqrt{S}$ - отношение периметра домена к периметру квадрата с такой же площадью [19, 95],

$C/2\sqrt{\pi S}$ - отношение периметра домена к периметру круга с такой же площадью.

$L_{\max}/L_{\min}$ плохо отражает форму у нерегулярных сеток [18]. Круги дают лучшее отношение периметра к площади, но соседние домены становятся вогнутыми. Также основанная на кругах метрика не учитывает зигзаги и вписанные углы.

Разработчики последовательного пакета PARTY добавили оптимизацию характеристических отношений доменов в пузырьковый метод с балансировкой диффузионным алгоритмом, который будет рассмотрен ниже [18].

Разработчики пакета JOSTLE вставили оптимизацию характеристических отношений доменов в последовательный иерархический алгоритм [19, 95]. В нем используется отношение $C/2\sqrt{\pi S}$. Вершины в графе представляют собой элементы сетки, ребра есть между теми вершинами, элементы которых имеют общую грань (дуальный граф). Веса ребер равны размеру поверхности, к которой они относятся. С таким взвешиванием ребер обычный поиск разбиения с минимальным весом разрезанных ребер оптимизирует характеристические отношения доменов. При огрублении стягивание самого тяжелого ребра аналогично стягиванию через самую большую поверхность. В алгоритме локального уточнения выигрыши зависят от характеристических отношений доменов. Описываемый иерархический алгоритм, оптимизирующий характеристические отношения доменов, реализован только в последовательном варианте, и работа оптимизации в параллельной реализации пока не

проверена. К тому же вычисление характеристических отношений требует знания геометрии графа, которое не всегда доступно.

2.2.3.6 Алгоритмы наращивания доменов

Еще одной попыткой устранения несвязности получаемых доменов стали алгоритмы наращивания доменов.

Алгоритм Фархата

Жадный алгоритм Фархата (Farhat) основан на поиске в ширину. Вершина наименьшей ненулевой степени (имеет наименьшее число соседей) становится инициализирующей вершиной нулевого домена [20]. Остальные вершины для этого домена находятся поиском в ширину. Когда число вершин в домене станет равным $\left\lceil |V| / p \right\rceil$, где p - число доменов, вершина, соседняя с нулевым доменом, становится инициализирующей вершиной первого домена. Остальные домены заполняются подобным образом. Алгоритм производит достаточно быстрые разбиения на связные домены, но границы доменов могут быть длинными, поскольку при поиске в ширину выбор между соседями вершины может оказаться неоптимальным. Также результаты алгоритма чувствительны к нумерации вершин в графе, то есть при смене нумерации могут быть получены отличающиеся разбиения. Описанный алгоритм реализован в последовательном пакете PARTY [21]. Похожий алгоритм роста доменов (graph growing partition algorithm) вызывается в иерархическом методе на этапе бисекции в последовательном пакете METIS [7, 96, 97, 98].

Жадный алгоритм с выигрышами

В пакете PARTY реализован жадный алгоритм, аналогичный алгоритму Фархата, но учитывающий выигрыши от перемещений вершин [21]. Выигрыши высчитываются так же, как и в алгоритме локального уточнения KL. При выборе следующей вершины из всех невыбранных соседей текущего

домена выбирается тот сосед, который минимизирует общий вес разрезанных ребер [99].

Похожий алгоритм (greedy graph growing partition algorithm) реализован в пакете METIS [7]. Алгоритм менее чувствителен к выбору инициализирующей вершины, чем ранее рассмотренный алгоритм пакета METIS graph growing partition algorithm, не учитывающий выигрыши от перемещения вершин.

Алгоритм пузырькового роста

Рассмотренные жадные алгоритмы плохо масштабируются на большое число доменов. В алгоритме пузырькового роста (bubble growing algorithm), реализованном в пакете PARTY [21], рост всех доменов происходит одновременно. Случайные вершины становятся инициирующими для доменов. Домены наращиваются поиском в ширину. Затем для каждого домена вычисляется его центр. Это вершина с наименьшей суммой длин кратчайших путей до всех вершин домена. Центры становятся новыми инициирующими вершинами, и домены заново наращиваются. Процесс прекращается, когда инициирующие вершины перестают меняться, или когда возникает конфигурация, полученная ранее. Метод не гарантирует получения сбалансированного разбиения, поэтому после этого алгоритма для балансировки загрузки приходится вызывать другие эвристические методы, что делает его дорогостоящим по времени [28]. Как упоминалось ранее, разработчики пакета PARTY адаптировали алгоритм пузырькового роста к оптимизации характеристических отношений доменов.

В работе [100] алгоритм пузырькового роста получил дальнейшее развитие. Вместо вычисления центров доменов решается система линейных уравнений. Алгоритм, хотя и параллельный, но очень медленный и также выдает несбалансированные разбиения, для балансировки которых приходится вызывать жадные алгоритмы.

2.2.3.7 Инкрементный алгоритм декомпозиции графов

Инкрементный алгоритм декомпозиции графов разработан Якобовским М.В. из ИПМ им. М.В.Келдыша РАН. Алгоритм является последовательным. Достоинством инкрементного алгоритма является формирование связных доменов. Инкрементный алгоритм не основывается на иерархическом подходе. Наиболее близкими к нему являются алгоритм пузырькового роста и диффузионные алгоритмы. Однако алгоритм пузырькового роста, в отличие от инкрементного алгоритма, не гарантирует получение сбалансированного разбиения. А одним из отличий инкрементного алгоритма от диффузионных алгоритмов является то, что в инкрементном алгоритме происходит освобождение части вершин из плохих доменов и последующий рост доменов.

Вначале выполнения инкрементного алгоритма декомпозиции графов [22] все вершины считаются свободными, то есть нераспределенными по доменам. Затем для каждого домена случайным образом выбирается вершина, которая помещается в этот домен и становится его инициализирующей вершиной. Далее разбиение производится посредством итерационного процесса, на каждом шаге которого выполняются следующие действия:

1. Инкрементный рост доменов (захват вершин). На данном этапе происходит постепенное присоединение к доменам прилегающих к ним свободных вершин и диффузное перераспределение вершин на границах между доменами с целью выравнивания суммарного веса вершин в доменах. В конце данного этапа все вершины должны быть распределены по доменам (Рис. 19 и 20). Расширение доменов производится поиском в ширину, при диффузном перераспределении вершин вычисляется матрица переносов, определяющая, какой вес вершин можно перенести из домена в домен.

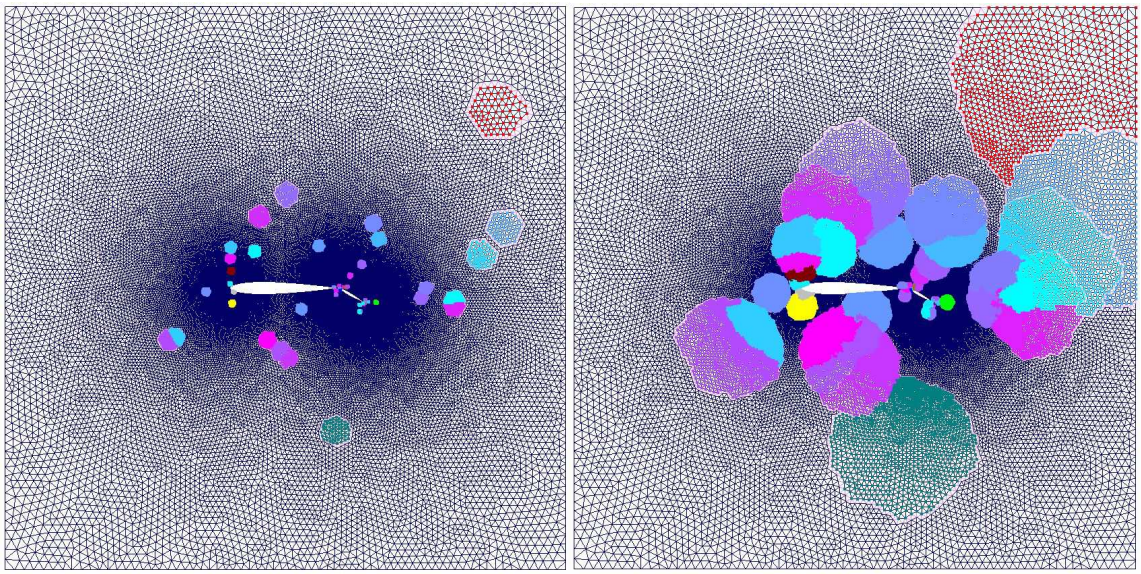


Рис. 19. Инкрементный рост доменов

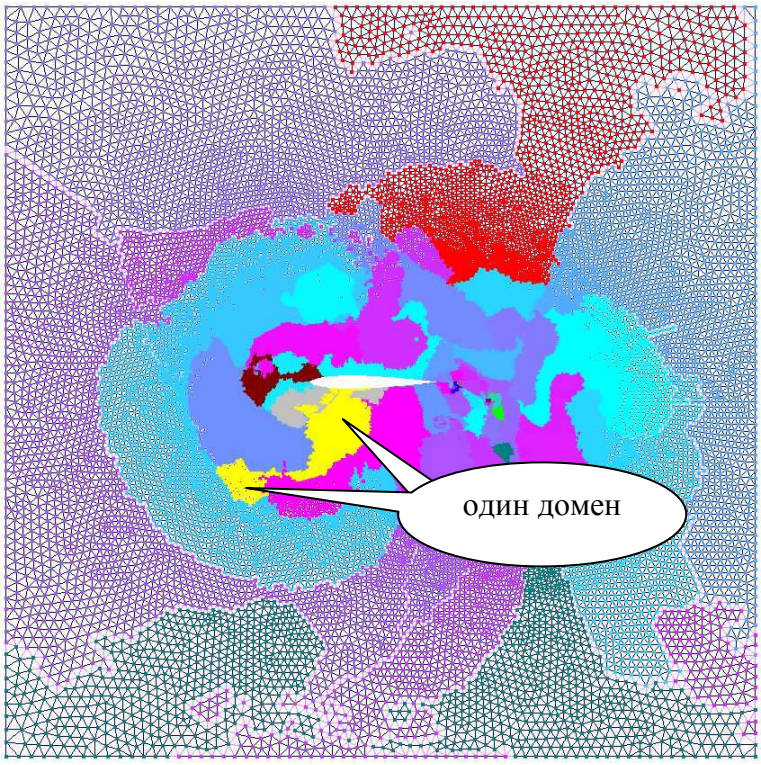


Рис. 20. Диффузное перераспределение вершин между доменами

2. Локальное уточнение доменов (Рис. 21). Выполняется KL/FM алгоритм локального уточнения [10].

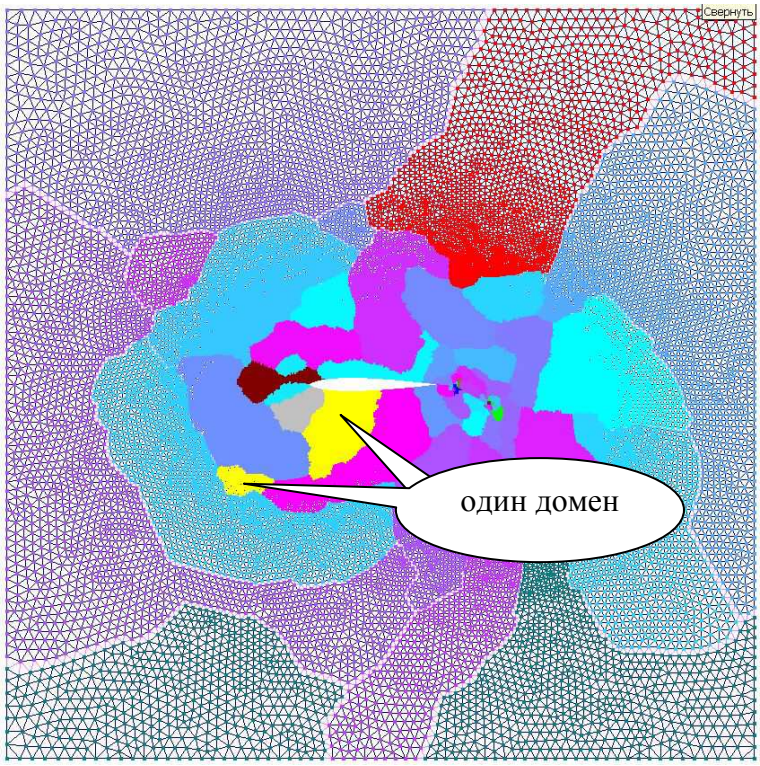


Рис. 21. Локальное уточнение доменов

3. Проверка качества доменов. Если качество доменов соответствует заданному, разбиение считается найденным, и происходит выход из цикла, иначе - переход к следующему этапу.

4. Освобождение части вершин плохих доменов и переход к первому этапу. Плохими доменами считаются домены, качество которых не соответствует заданному, и соседи таких доменов. В плохих доменах часть вершин освобождается, то есть вновь считается нераспределенной (Рис. 22). Освобождается часть внешних оболочек, а затем все компоненты связности, кроме той, которая содержит наибольшее число вершин.

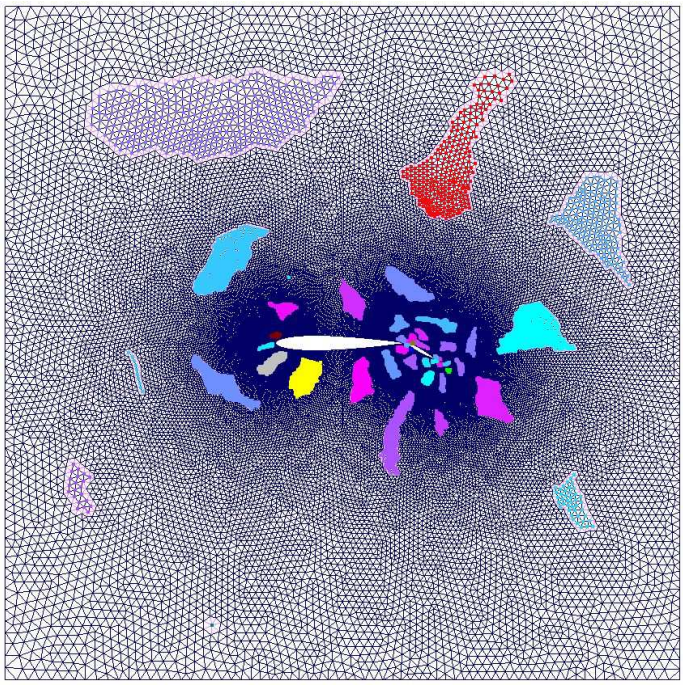


Рис. 22. Освобождение части вершин

Результирующее разбиение представлено на Рис. 23.

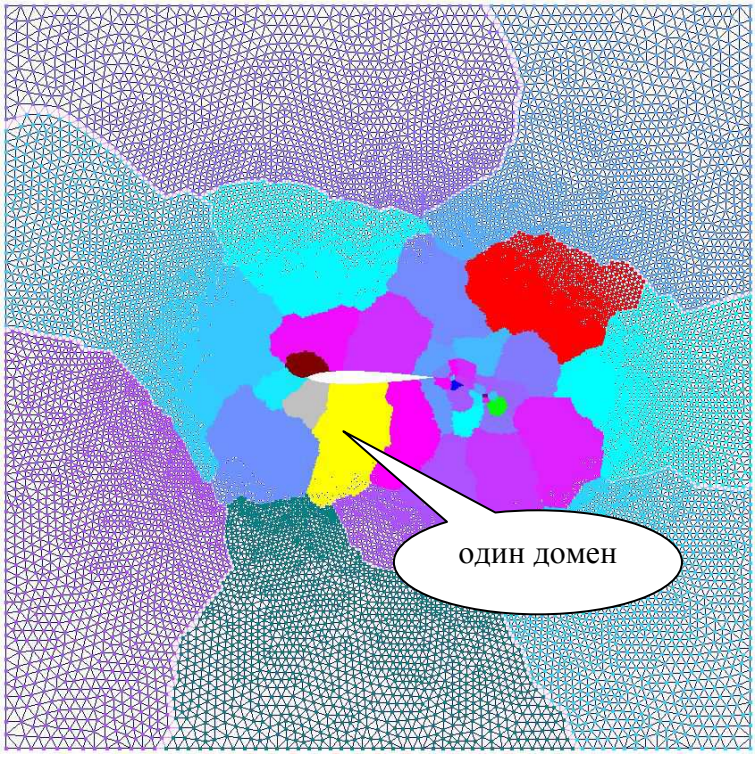


Рис. 23. Результирующее разбиение

Качество доменов проверяется следующим образом. Все вершины, расположенные на геометрической границе графа и на границах между домена-

ми, считаются принадлежащими первой *оболочке* своего домена. В каждом домене вершинами второй оболочки считаются соседи вершин из первой оболочки данного домена, которые так же принадлежат этому домену и не попали в первую оболочку. Остальные оболочки вычисляются аналогично. На Рис. 24 представлены оболочки домена при условии, что вся сетка принадлежит одному домену. Проверяется связность оболочек каждого домена и вычисляется номер несвязной оболочки с наименьшим номером (номер первой несвязной оболочки) в каждом домене. Хорошими считаются те домены, номер первой несвязной оболочки в которых не меньше заданного порогового значения.

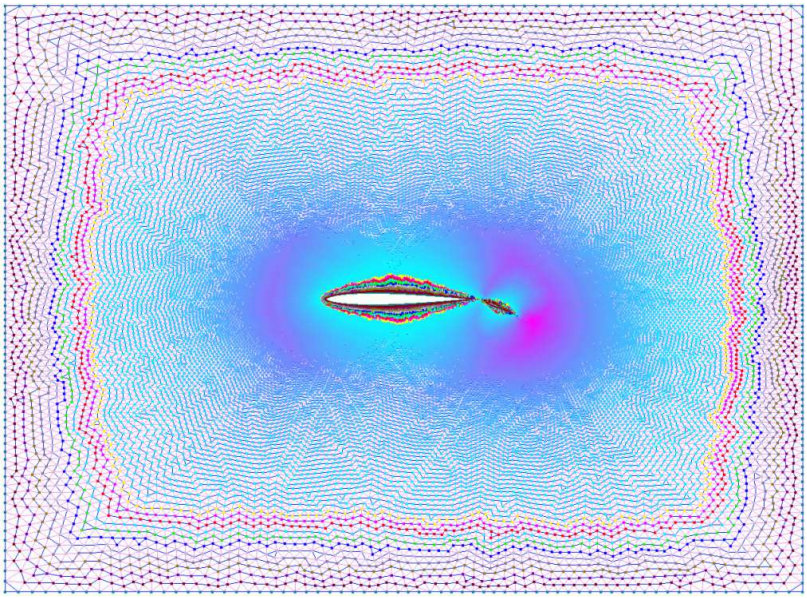


Рис. 24. Оболочки домена

В отличие от всех рассмотренных алгоритмов, инкрементный алгоритм декомпозиции графов, выполняет сбалансированные разбиения на связные домены. Поэтому именно на основе него в защищаемой работе реализован параллельный инкрементный алгоритм декомпозиции графов.

2.3 Параллельный алгоритм геометрической декомпозиции сеточных данных

Алгоритм геометрической декомпозиции, разработанный в данной диссертации, основывается на методе рекурсивной координатной бисекции [6, 21]. На каждом этапе рекурсивной бисекции область разбивается на две части. Соотношение размеров частей зависит от количества доменов, которые должны быть образованы в каждой из частей. Полученные подобласти разбиваются дальше аналогичным образом до тех пор, пока в подобластях не останется по одному домену. При рекурсивной координатной бисекции на этапе разбиения выбирается координатная ось, вдоль которой область имеет наибольшую протяженность. Область разбивается перпендикулярно выбранной оси. Алгоритм работает только с координатами вершин и не учитывает связи между ними, что делает его экономичным по памяти.

Основные этапы алгоритма следующие [103 - 107]:

1. Начальное распределение вершин по процессорам. Начальное распределение может быть любым, например, в соответствии с порядковыми номерами вершин.

2. Определение числа доменов, которые должны быть сформированы на каждом из процессоров, и количества вершин в них.

3. Рекурсивная координатная бисекция вершин по процессорам. На каждом этапе сначала вычисляется координатная ось, вдоль которой область имеет наибольшую протяженность. Затем блок вершин бьется на две части перпендикулярно данной оси (Рис. 25). Группа процессоров делится на две, далее каждая из групп делит свой блок вершин аналогичным образом. Для разделения блока вершин используется параллельная сортировка. После рекурсивной координатной би-

секции вершин по процессорам на процессорах оказывается столько вершин, сколько должно быть в образуемых на них доменах.

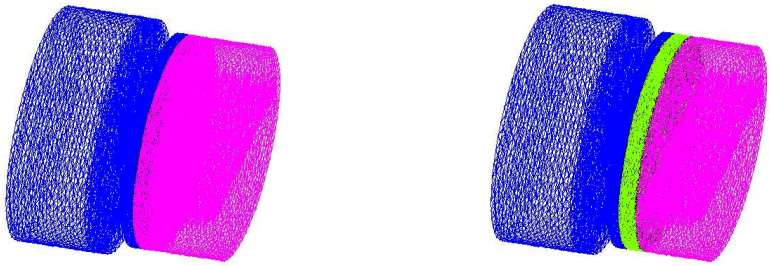


Рис. 25. Рекурсивная координатная бисекция вершин по трем процессорам

4. Локальная рекурсивная координатная бисекция вершин по доменам. Дальнейшее разбиение на домены проводится локально на каждом процессоре. На Рис. 26 показана локальная рекурсивная координатная бисекция на три домена на одном из процессоров. На Рис. 27 представлен результат разбиения сетки на семь доменов на трех процессорах.

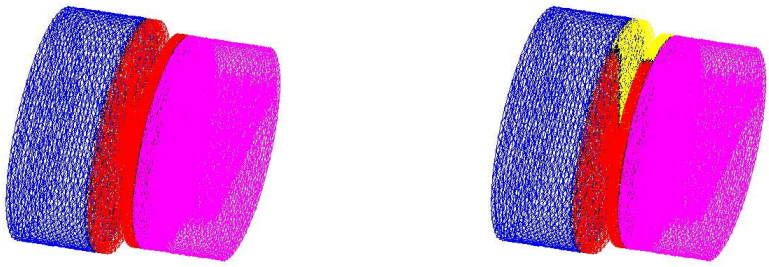


Рис. 26. Локальная рекурсивная координатная бисекция вершин на одном из процессоров

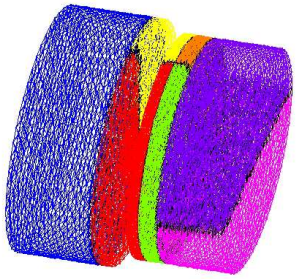


Рис. 27. Результат разбиения сетки на семь доменов на трех процессорах

Отличие разработанной рекурсивной координатной бисекции от обычного метода рекурсивной координатной бисекции состоит в следующем. Разбиение проводится одновременно по нескольким координатам, вершины сортируются сначала по одной координате, потом внутри нее по следующей координате в циклическом порядке и т.д., что позволяет обрабатывать ситуации наличия нескольких узлов с одним значением координаты (Рис. 28). В результате медиана проводится точно, и в разбиении на равные домены числа вершин в доменах отличаются не больше, чем на единицу.

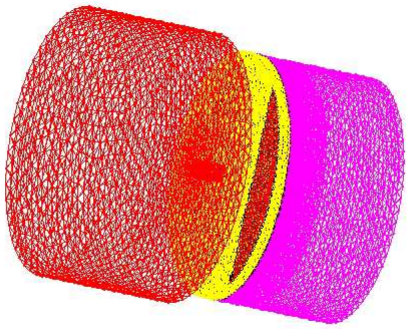


Рис. 28. Разбиение по нескольким координатам в локальной рекурсивной координатной бисекции

Ниже части параллельного алгоритма геометрической декомпозиции сеточных данных описаны более детально.

2.3.1 Параллельная сортировка

В процессе выполнения рекурсивной координатной бисекции вершин по процессорам локально на каждом процессоре данные сортируются либо сортировкой слиянием, либо пирамидальной, либо оптимизированной сортировкой (гибрид сортировки слиянием и пирамидальной сортировки) [101]. В качестве параллельной сортировки была взята битонная сортировка Бэтчера [102]. Для сортировки данных использовалась библиотека параллельной сор-

тировки (Якобовский М.В., ИПМ им. М.В.Келдыша РАН). Автором диссертации в нее были внесены следующие изменения:

1. Добавлена возможность сортировки массивов наборов данных, состоящих из нескольких элементов одинакового произвольного типа. Теперь функции сортировки принимают на вход такие параметры, как число элементов в наборе данных, индекс ключа в наборе данных и указатель на функцию сравнения наборов данных.

2. В битонную сортировку Бэтчера добавлено выравнивание длин сортируемых массивов, поскольку с массивами разной длины алгоритм может работать неверно. Теперь алгоритм устроен следующим образом:

- На каждом процессоре выполняется локальная сортировка данных.

- Запоминается минимальный набор данных. Затем с помощью поданной в сортировку функции сравнения и сбора по бинарному дереву определяется минимальный набор данных между всеми процессорами. Если предположить, что процессоры пронумерованы от единицы, то процессор с номером k получает минимальные наборы данных от процессоров с номерами $(2 \cdot k)$ и $(2 \cdot k + 1)$. Сравнивает их со своим минимальным набором и минимальный набор из всех посылает на процессор с номером $(k / 2)$. В итоге минимальный набор данных по процессорам попадает на процессор с номером один, а потом рассылается на все остальные. Данный алгоритм был реализован для нумерации процессоров от нуля, поскольку именно она используется в MPI. Далее на всех процессорах в массивы с левого края добавляется недостающее количество минимальных наборов, и запоминается суммарное количество добавленных минимальных наборов по всем процессорам.

- Выполняется битонная сортировка Бэтчера [102].

- Побитовым сравнением на каждом из процессоров определяется количество наборов, равных минимальному. На процессорах, начиная с нулевого, удаляются минимальные наборы в суммарном количестве, которое было добавлено. Подсчитывается оставшееся число наборов на процессорах.

- Выполняется *сдвиг данных между процессорами*, обеспечивающий получение на процессорах заданного числа наборов. Пусть нужно вычислить, сколько наборов и кому, должен отдать процессор с номером *p*. Подсчитывается, сколько сейчас наборов расположено на процессорах с номерами, меньшими *p*. Далее вычисляется, на сколько процессоров хватит этих наборов, с учетом количества наборов, которое должно оказаться на процессорах. Находится процессор, на котором закончатся эти наборы. Ему и должен будет отдать начальные свои наборы процессор с номером *p*. В зависимости от того, сколько наборов данных на процессоре *p*, вычисляется, сколько он должен отдать последующим процессорам. Для выяснения количества присылаемых наборов и того, какие процессоры их пришлют, подсчитывается, сколько наборов должно оказаться после сдвига на процессорах с номерами, меньшими *p*. Как и раньше, подсчитывается, количество процессоров, которые пошлют эти наборы, если учесть, сколько данных сейчас на процессорах, и что они начинают посылать данные по порядку. Находится процессор, у которого останется часть наборов данных для процессора *p*. Далее, в зависимости от того, сколько наборов данных должно оказаться на процессоре *p*, подсчитывается, сколько пришлют следующие процессоры. Выполняется обмен данными. Сдвиг данных между процессорами используется не только в параллельной сортировке, но и отдельно в параллельном алгоритме геометрической декомпозиции и в параллельном инкрементном алгоритме.

2.3.2 Деревья разбиений

В процессе параллельной геометрической декомпозиции сетки производится построение глобального дерева распределения вершин по процессорам и локальных деревьев разбиения вершин на домены в рамках каждого из процессоров (данная опция может быть отключена). В двоичных деревьях разбиения каждая вершина содержит информацию о медиане разбиения на две части на данном этапе рекурсии. Информация включает в себя координаты медианы, номер вершины медианы и номер координаты, по которой производится деление в данный момент. В глобальном дереве каждый лист содержит в поле номера координаты «-1», а в поле номера вершины – номер процессора, который дальше продолжит деление. Лист соответствует моменту, когда группа процессоров содержит только один процессор, и дальше соответствующая ему группа вершин будет разбиваться на домены локально. В локальном дереве разбиения каждый лист содержит в поле номера координаты «-1», а в поле номера вершины – номер домена, в который попали данные вершины. В процессе выполнения параллельной геометрической декомпозиции сетки на каждом процессоре создается глобальное дерево разбиения вершин по процессорам и массив локальных деревьев разбиений вершин по доменам на каждом из процессоров.

Для того чтобы определить, в какой домен попадет вершина при полученном разбиении, сначала осуществляется поиск в глобальном дереве разбиения вершин по процессорам. В каждой вершине дерева производится сравнение искомой вершины с медианой и, в зависимости от результата, дальнейший поиск производится либо в левом поддереве, либо в правом. Найденный лист позволяет определить, в локальном дереве какого процессора следует продолжить поиск. В локальном дереве поиск производится аналогично, а найденный лист дает информацию о номере домена искомой вершины.

Построение деревьев было включено для случая разбиения различных сеток на одинаковые параллелепипеды, что потребовалось при параллельной интерполяции значений с одной сетки на другую. Интерполяция используется в задачах аэроакустики, когда часть расчета производится на грубой сетке, а продолжение – на измельченной сетке, что позволяет сократить время расчета.

2.3.3 Определение количества доменов, формируемых на процессорах

В параллельном алгоритме геометрической декомпозиции реализованы два варианта определения количества доменов, формируемых на процессорах, и числа вершин, принадлежащих каждому из доменов. Первый вариант вызывается, когда геометрическая декомпозиция используется в качестве самостоятельного метода декомпозиции вершин на домены равного размера. Второй вариант вызывается, когда геометрическая декомпозиция используется в качестве предекомпозиции вершин по процессорам в другом алгоритме, например, в параллельном инкрементном алгоритме декомпозиции графов, описанном ниже. В этом случае принимается решение, сколько и каких доменов будет сформировано позже на процессорах, и вершины разбиваются по процессорам в соответствии с принятым решением.

Определение количества доменов на процессорах в первом варианте происходит так. Пусть нужно распределить *n* вершин по *nproc* процессорам, чтобы в дальнейшем сформировать на них *npart* доменов. Из *npart* доменов *(n % npart)* доменов (остаток от деления) будут содержать *(n / npart + 1)* вершину, а остальные – *(n / npart)* вершин (целая часть от деления). Обозначим *(n / npart)* как *nc*, число доменов с *(nc + 1)* вершинами за *kpart1*, а число доменов с *nc* вершинами за *kpart*. И *kpart1*, и *kpart* доменов разбиваются равномерно по процессорам, аналогично разбиению вершин по доменам. Домены с *nc* вершинами распределяются по процессорам, начиная с нулевого: процес-

соры с меньшими номерами будут формировать *(kpart / nproc + 1)* домен с *nc* вершинами, а с большими номерами – *(kpart / nproc)* доменов. Домены с *(nc + 1)* вершинами распределяются в обратном порядке: процессоры с большими номерами будут формировать *(kpart1 / nproc + 1)* домен с *(nc + 1)* вершинами, а с меньшими номерами – *(kpart1 / nproc)* доменов. В соответствии с количеством доменов каждого вида, подсчитывается, сколько вершин должно быть на каждом из процессоров.

При определении количества доменов во втором варианте *n* вершин разбивается на *npart* доменов (будущие процессоры) для будущего формирования на них *ng* микродоменов. Разбиение производится на *nproc* процессорах. Количество микродоменов, равное *(n % ng)* будет содержать *(n / ng + 1)* вершину, остальные микродомены – *(n / ng)* вершин. Обозначим *(n / ng)* как *nc*. Число доменов, равное *(ng % npart)* будет содержать *(ng / npart + 1)* микродомен, а остальные – *(ng / npart)* микродоменов. Обозначим *(ng / npart)* как *np*. Микродомены по доменам распределяются следующим образом: в доменах с меньшими номерами будет содержаться *(np + 1)* микродомен, а с большими – *np* микродоменов. При этом домены с меньшими номерами сначала заполняются микродоменами, содержащими *nc* вершин, а после распределения всех микродоменов с *nc* вершинами, домены заполняются микродоменами, содержащими *(nc + 1)* вершину. В результате получатся домены четырех типов. Домены нулевого типа будут содержать *(np + 1)* микродомен с *nc* вершинами. Домены третьего типа будут содержать *np* микродоменов с *(nc + 1)* вершиной. С доменами первого и второго типа возможны варианты в зависимости от количества микродоменов с *nc* вершинами. Если микродоменов с *nc* вершинами меньше, чем нужно для доменов, содержащих *(np + 1)* микродомен, то один домен первого типа будут содержать часть микродоменов с *nc* вершинами и часть с *(nc + 1)* вершиной, всего *(np + 1)* микродомен. Домены второго типа будут содержать *(np + 1)* микродомен с *(nc + 1)* вершиной. Если микродоменов с *nc* вершинами ровно столько, сколько нужно для доменов,

содержащих *(np + 1)* микродомен, то доменов первого и второго типа не будет. А если микродоменов с *nc* вершинами больше, чем нужно для доменов, содержащих *(np + 1)* микродомен, то домены первого типа будут содержать *np* микродоменов с *nc* вершинами, а один домен второго типа будет содержать *np* микродоменов, часть из которых с *nc* вершинами, а часть с *(nc + 1)* вершиной. Предполагается, что алгоритмами, использующими геометрическую предекомпозицию, поддерживается аналогичное распределение микродоменов по доменам (на тот момент доменам по процессорам), при котором на процессорах с меньшими номерами формируется больше доменов, чем на процессорах с большими номерами. Домены распределяются по *nproc* процессорам равномерно, *(npart % nproc)* процессоров с меньшими номерами будет содержать *(npart / nproc + 1)* домен, а остальные процессоры – *(npart / nproc)* доменов. Заполнение процессоров доменами происходит, начиная с нулевого процессора. Сначала распределяются домены нулевого типа, потом первого, второго, и только в конце третьего типов. Подсчитывается, сколько вершин должно быть на каждом из процессоров.

Для каждого процессора вычисляется начальный номер *gr0* номеров доменов, формируемых на нем.

2.3.4 Рекурсивная координатная бисекция вершин по процессорам

Массив данных, используемый в параллельном алгоритме геометрической декомпозиции, содержит в себе наборы данных о каждой вершине, включающие координаты и номер вершины.

На каждом этапе рекурсивной координатной бисекции вершин по процессорам происходит следующее:

1. Определяются границы наименьшего параллелепипеда, охватывающего все вершины, которые делятся на данном этапе. Вычисляются ми-

нимумы и максимумы по каждой из координат. Определяется координата *j*, вдоль которой параллелепипед имеет наибольшую протяженность.

2. Если на предыдущем шаге рекурсии вершины сортировались по другой координате, то выполняется параллельная сортировка вершин по координате *j*. В противном случае вершины только сдвигаются по процессорам для получения нужного количества вершин на каждом из процессоров. При делении группы из *nproc* процессоров в левой группе процессоров (процессоры с меньшими номерами, которые получат вершины меньшей координатой *j*) окажется (*nproc1 = nproc / 2)* процессоров, остальные *(nproc - nproc1)* процессоров попадут в правую группу. И при сортировке, и при сдвиге подсчитывается, сколько всего вершин должно быть в доменах на *nproc1* процессорах. Данное количество вершин распределяется равномерно среди *nproc1* процессоров. Когда *nproc1* станет равным единице, на процессор попадает итоговое количество вершин, определенное заранее. Распределение вершин среди процессоров в правой группе происходит аналогично.

3. Вблизи медианы выполняется сортировка вершин по всем координатам. Медиана проходит между процессорами с номерами *(nproc1 - 1)* и *nproc1*. На процессоре с номером *(nproc1 - 1)* выделяются вершины с наибольшей координатой *j*, которые расположены у правого края массива данных. На процессоре с номером *nproc1* выделяются вершины с наименьшей координатой *j*, которые расположены у левого края массива данных. Процессоры обмениваются информацией о вершинах, и суммарные массивы отобранных вершин сортируются по всем координатам локально. Затем процессор с номером *(nproc1 - 1)* отбирает себе нужное количество вершин с левого края отсортированного массива, а процессор с номером *nproc1* – с правого края массива. При *сортировке*

по всем координатам при сравнении данных о двух вершинах сравниваются их координаты в циклическом порядке, начиная с координаты *j*, до нахождения первого расхождения. Если все координаты оказались равными, сравниваются глобальные номера вершин, и вершина с меньшим номером считается меньше.

4. Процессором с номером *(nproc1 - 1)* запоминаются данные о медиане этого этапа для дальнейшей сборки глобального дерева разбиения (если в параллельный алгоритм геометрической декомпозиции поданы параметры, отвечающие за построение деревьев). Медиана является самой последней вершиной в массиве данных этого процессора, значение ее координаты *j* является наибольшим на процессоре. Запоминаются координаты медианы, номер вершины медианы и координата *j*, по которой происходило деление на данном этапе.

5. Создаются две новые группы процессоров с *nproc1* и *(nproc – nproc1)* процессорами. Процессоры с номерами от *0* до *(nproc1 – 1)*, включительно, помещаются в левую группу, остальные процессоры – в правую группу. Создаются новые коммуникаторы, и процессоры получают новые идентификаторы в них. На следующем этапе каждая группа процессоров будет делить свой блок данных.

После завершения рекурсивной координатной бисекции вершин по процессорам, если предполагается построение деревьев, на нулевом процессоре строится глобальное дерево разбиения по процессорам. Со всех процессоров собирается информация о сохраненных на них медианах. Полученная информация сортируется в порядке следования вершин двоичного дерева с помощью рекурсии. Медиана, соответствующая вершине дерева с номером *i*, была получена от процессора с номером *(ipf + nproc1 – 1)*, где *nproc1 = nproc / 2*. Она соответствует моменту, когда делилась группа из *nproc* процессоров с номерами, большими, либо равными, *ipf*. Для *i*, равного единице, *ipf = 0*.

Медиана вершины с номером *(2 · i)* находится аналогично для группы из *nproc1* процессоров с номерами, начинающимися с *ipf*. Медиана вершины с номером *(2 · i + 1)* вычисляется для группы из *(nproc – nproc1)* процессоров с номерами, начинающимися с *(ipf + nproc1)*. И далее по рекурсии. В момент, когда *nproc1* станет равным единице, в лист дерева будет записана информация о то, что дальнейшее деление продолжит процессор с номером *ipf*. Для *(nproc – nproc1)*, равного единице, в лист глобального дерева будет записан процессор с номером *(ipf + nproc1)*. Запоминается максимальный номер вершины в глобальном дереве. Глобальное дерево разбиения по процессорам и максимальный номер вершины в нем, рассылаются на все процессоры.

2.3.5 Локальная рекурсивная координатная бисекция вершин по доменам

На каждом процессоре вычисляются минимумы и максимумы по каждой из координат, определяющие параллелепипед, охватывающий вершины данного процессора. Далее запускается процесс рекурсии, на каждом шаге которого выполняется следующее:

1. Определяется координата *j*, вдоль которой параллелепипед, разрезающийся на данном этапе, имеет наибольшую протяженность.

2. Определяется количество доменов, которое делится на данном этапе. В левую часть (вершины с меньшей координатой *j*) попадет половина доменов. Сначала набираются домены нулевого типа, потом, если они закончились, первого, и т.д. до получения нужного числа доменов в левой части. Подсчитывается общее число вершин *n1*, которое окажется в этих доменах. В правой части будут сформированы оставшиеся домены.

3. Пусть все вершины в параллелепипеде имеют координаты *j*, принадлежащие интервалу *[minj, maxj]*. Интервал делится на *K* равных малых

интервалов. *K* – некоторая константа, равная, например, 10^5. Подсчитываются количества вершин, принадлежащие каждому малому интервалу. Исходя из того, что в левую часть должна попасть *n1* вершина с меньшей координатой *j*, определяется, в какой малый интервал *[fromj, toj]* попадает медиана. Затем вершины из интервала *[fomj, toj]* записываются в отдельный массив, отсортированный по всем координатам по аналогии с сортировкой, использованной при делении вершин по процессорам. Определяется медиана – последняя вершина, которая попадет в левую часть.

4. Если предполагается построение локальных деревьев, для вершины *i* двоичного дерева запоминаются данные о медиане: ее координаты, номер вершины медианы и номер координаты *j*, по которой проводилось деление.

5. Проводится разделение данных по аналогии с обменной сортировкой с разделением Хоару (быстрая сортировка) [101], чтобы на следующих этапах просматривались только вершины из разбиваемой области. Два указателя, *beg* и *end*, выставляются на начало и на конец массива данных, соответственно. Вершина, на которую указывает указатель *beg*, сравнивается с медианой по всем координатам, и, если она попадает в левую часть параллелепипеда, указатель *beg* сдвигается к следующей вершине. Так до тех пор, пока указатель *beg* не остановится на вершине, которая должна попасть в правую часть. Указатель *end* конца массива сдвигается влево до тех пор, пока не остановится на вершине, которая должна попасть в левую часть. Если указатель *beg* находится левее указателя *end*, вершины, на которые указывают указатели, меняются местами, после чего указатель *beg* сдвигается вправо, а *end* – влево. Если указатель *beg* все еще левее указателя *end*, весь процесс повторяется снова. Если после выхода из цикла указатели указывают на одну и

ту же вершину, она сравнивается с медианой по всем координатам, и если вершина попадает в левую часть, указатель *beg* сдвигается к следующей вершине. Указатель *beg* определяет начало массива данных правой части.

6. По рекурсии левая и правая части отправляются на деление. Левой части передается указатель на начало исходного массива данных и число вершин *n1*. Максимальные координаты параллелепипеда остаются теми же, только максимальная координата *j* приравнивается к координате *j* медианы. Передается число доменов каждого вида, которое будет в левой части. Номер вершины двоичного дерева, отвечающей за деление левой части, будет равным $(2 \cdot i)$. Правой части передается указатель *beg* в качестве нового указателя на массив данных и оставшееся число вершин. Минимальная координата *j* приравнивается к координате медианы. Передаются числа доменов каждого вида. Номер вершины двоичного дерева, отвечающей за деление правой части, будет равным $(2 \cdot i + 1)$.

7. Перед делением каждой из частей проверяется, сколько доменов осталось в данной части. Если остался только один домен, то все вершины помещаются в данный домен. Домены считаются по порядку от номера *gr0* первого домена на процессоре. Если предполагается построение локальных деревьев, в лист дерева записывается номер домена. Запоминается максимальный номер вершины в локальном дереве данного процессора. Если в части осталось больше одного домена, она отправляется на деление.

После завершения процесса локальной рекурсивной бисекции, если предполагается построение локальных деревьев разбиений, массив всех локальных деревьев разбиений вершин по доменам собирается на каждом из процессоров.

Информация о доменах вершин перераспределяется между процессорами в соответствии с распределением вершин по процессорам вначале параллельного геометрического алгоритма декомпозиции сеточных данных.

2.3.6 Принятые решения

Решения, принятые в алгоритме рекурсивной координатной бисекции вершин по процессорам:

- Рекурсивная параллельная сортировка была добавлена для улучшения качества распределения блоков вершин между процессорами, в сравнении с однократным разделением общего блока вершин между процессорами сортировкой по одной из координат, хотя она и увеличивает время работы алгоритма. Поскольку количество памяти, необходимое для работы данного алгоритма, невелико, и процессоров требуется немного (сетка, состоящая из $2 \cdot 10^8$ вершин, разбивалась на 40 процессорах, 512 Мб на каждом), такой вариант приемлем. Однократное разделение блока вершин сортировкой по одной из координат на упомянутой сетке на 40 процессорах выполняется за 18 сек., рекурсивная параллельная сортировка – за 67 сек, высота двоичного дерева при рекурсивной параллельной сортировке равна 6.

- Создание новых групп процессоров и новых коммуникаторов обеспечивает автономное взаимодействие процессоров в рамках групп.

- Равномерное распределение вершин между процессорами в группах обосновано тем, что в этом случае при параллельной сортировке придется добавлять меньше минимальных наборов для выравнивания длин массивов.

Решения, принятые в алгоритме локальной координатной бисекции:

- Разбиение на малые интервалы позволило заменить сортировку всего массива вершин по всем координатам на сортировку малого интервала, в который попадает медиана.

- Разделение данных было добавлено для того, чтобы на каждом этапе просматривались вершины не из всего массива данных процессора, а только из разбиваемой области.

2.4 Параллельный инкрементный алгоритм декомпозиции графов

Параллельный инкрементный алгоритм декомпозиции графов является расширенной параллельной версией последовательного инкрементного алгоритма декомпозиции графов, описанного ранее.

Прямое распараллеливание всех этапов инкрементного алгоритма декомпозиции графов невозможно, поскольку этапы инкрементного роста и локального уточнения доменов последовательны по своей природе. В них требуется периодическое обновление информации, которое в параллельном варианте приведет к большим коммуникационным издержкам. Поэтому выбрана следующая стратегия распараллеливания [103 -107]. Каждый процессор бьет свой блок вершин локально инкрементным алгоритмом декомпозиции графов, что позволяет избежать передач данных между процессорами. Недостатком локального разбиения является то, что группы не могут расширяться за пределы процессора, что ограничивает число просматриваемых хороших вариантов разбиения. В результате на границах между процессорами могут образоваться плохие группы доменов. Для устранения этого недостатка в последствии плохие группы доменов перераспределяются между процессорами так, чтобы каждая группа плохих доменов оказалась собранной на одном процессоре. Затем происходит локальное повторное разбиение плохих групп доменов.

В данной модели предполагается, что после локального разбиения инкрементным алгоритмом декомпозиции графов плохими окажутся не все домены на процессорах, а только небольшая их часть на границах между процессорами. Это утверждение выполняется, когда на процессорах образовывается достаточно большое количество доменов. На практике число процессоров, на которых будет считаться задача, заранее неизвестно, и выгоднее сразу разбить граф на большое количество микродоменов, чтобы в дальнейшем распределять эти домены по нужному числу процессоров. Количество микродоменов на несколько порядков меньше числа вершин, поэтому многократное распределение микродоменов по нужному числу процессоров быстрее многократного разбиения всего графа. Также при хранении сетки, разбитой на микродомены, можно сэкономить память. Таким образом, предположение о том, что количество доменов будет достаточно велико, является оправданным.

Как и в любом параллельном алгоритме декомпозиции графов, предполагается, что вершины изначально неким образом распределены по процессорам. Здесь возникает задача в задаче, когда для нахождения разбиения, нужно изначально разбить граф. В качестве начального разбиения обычно берут более простое разбиение, например, разбиение по номерам вершин, или геометрическое разбиение. Ограничение на расширение доменов за пределы процессоров позволило исключить синхронизацию между процессорами. Однако, в результате наложения данного ограничения, возникла зависимость локального алгоритма инкрементного роста доменов от начального распределения вершин по процессорам. Если при начальном распределении на процессор попадет множество мелких групп вершин, то после выполнения алгоритма инкрементного роста некоторые группы вершин окажутся не распределенными по доменам. Поэтому для получения относительно компактного начального распределения вершин по процессорам целесообразно использо-

вать геометрическое разбиение. Подробнее о геометрическом разбиении написано ранее.

После геометрического разбиения на процессор должен попасть такой общий вес вершин, какой будет в образуемых на нем доменах, что позволит сформировать домены одинакового веса.

Однако геометрическое разбиение не гарантирует, что на процессор вообще не попадут мелкие группы вершин. Например, если граф описывает сетку, содержащую в себе отверстия, при геометрическом разбиении на процессор может попасть основной блок вершин с одной стороны отверстия и небольшая группа вершин с другой стороны (Рис. 30). Таким образом, даже после геометрической декомпозиции необходимо предварительное перераспределение малых блоков вершин между процессорами.

Учитывая все сказанное, параллельный инкрементный алгоритм декомпозиции графов выглядит следующим образом:

1. Разбиение вершин на число доменов, равное числу процессоров, на которых будет выполняться дальнейшее разбиение, параллельным алгоритмом геометрической декомпозиции сеток, описанным выше.

2. Распределение вершин по процессорам в соответствии с результатами геометрической декомпозиции (Рис. 29 и 30). Общий вес вершин на процессорах определяется количеством будущих доменов.

3. Присоединение малых блоков вершин к большим и восстановление исходного веса вершин на процессорах (Рис. 29 и 30).

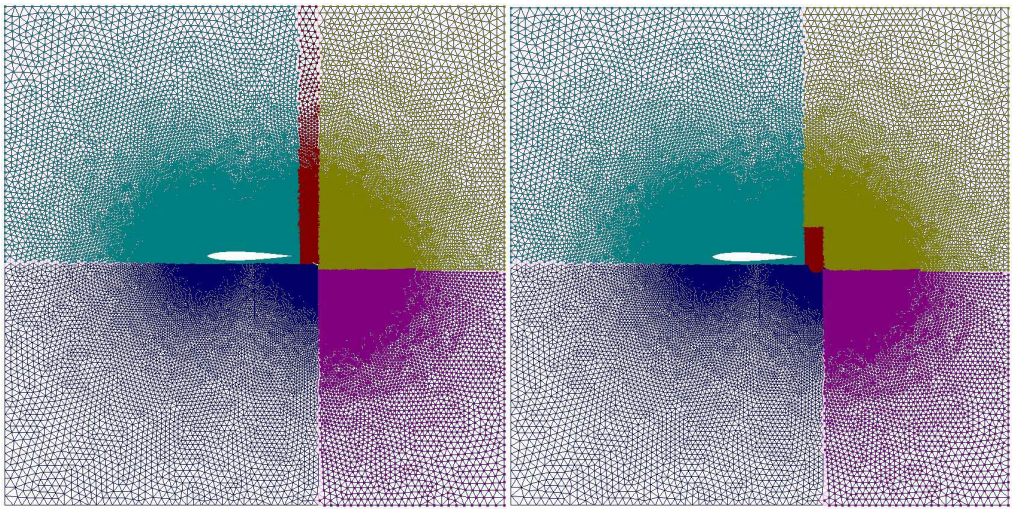


Рис. 29. Геометрическое разбиение сетки по процессорам (слева) и перераспределение малых блоков вершин (справа)

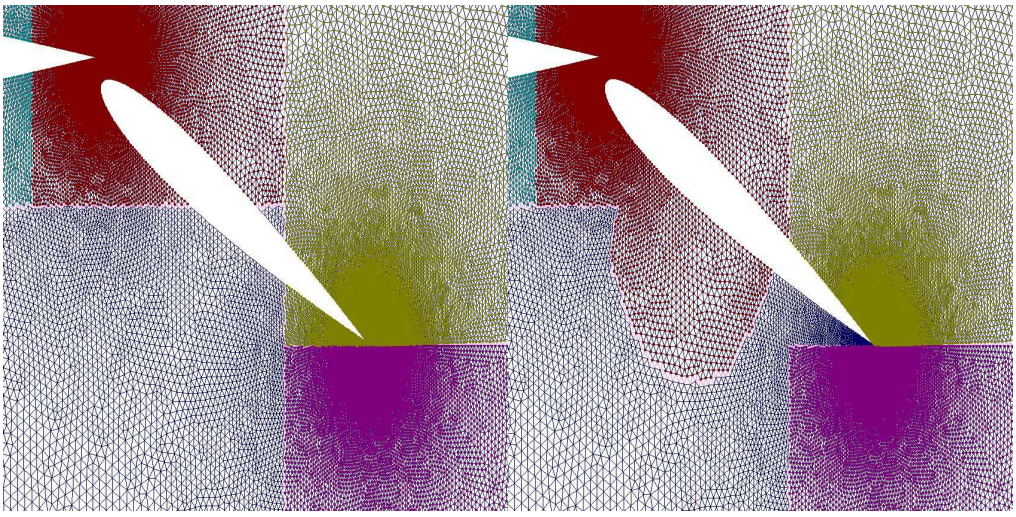


Рис. 30. Фрагменты геометрического разбиения сетки по процессорам (слева) и перераспределения малых блоков вершин (справа)

4. Инициализация доменов. Поскольку на процессоре может оказаться несколько больших блоков вершин, сначала оценивается вес вершин в блоках, в соответствии с ним определяется, сколько доменов будет образовано в каждом из блоков, а затем случайным образом выбирается нужное количество инициализирующих вершин в каждом из блоков.

5. Локальное разбиение вершин на домены инкрементным алгоритмом декомпозиции графов (Рис. 31 и 32).

6. Перераспределение плохих групп доменов. Сбор каждой группы плохих доменов на одном процессоре.

7. Локальное повторное разбиение плохих групп доменов инкрементным алгоритмом декомпозиции графов (Рис. 31 и 32).

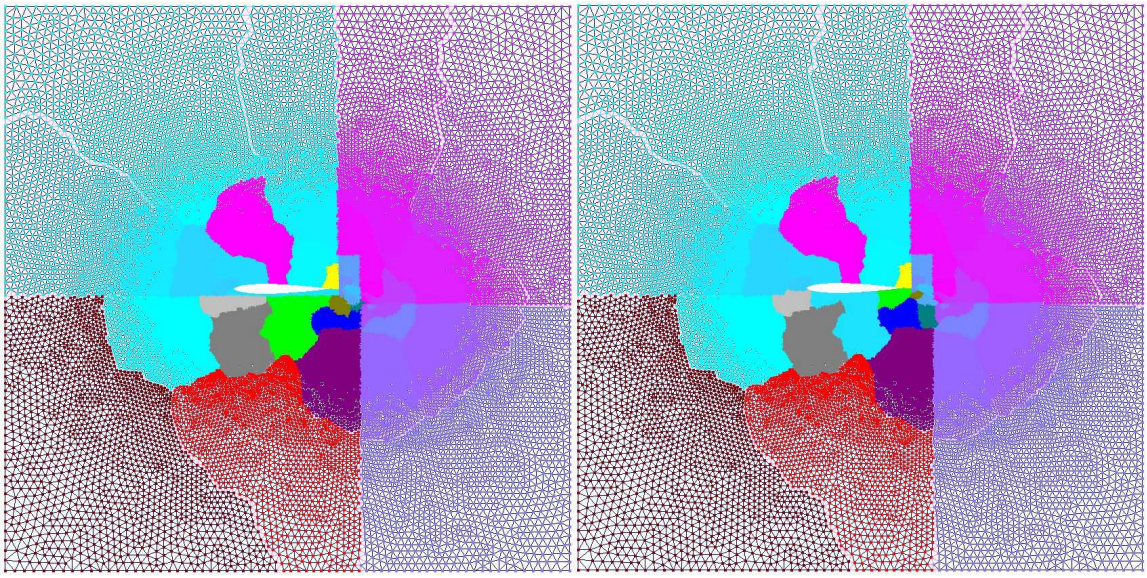


Рис. 31. Результаты локального разбиения (слева) и сбора плохих групп доменов и их повторного разбиения (справа)

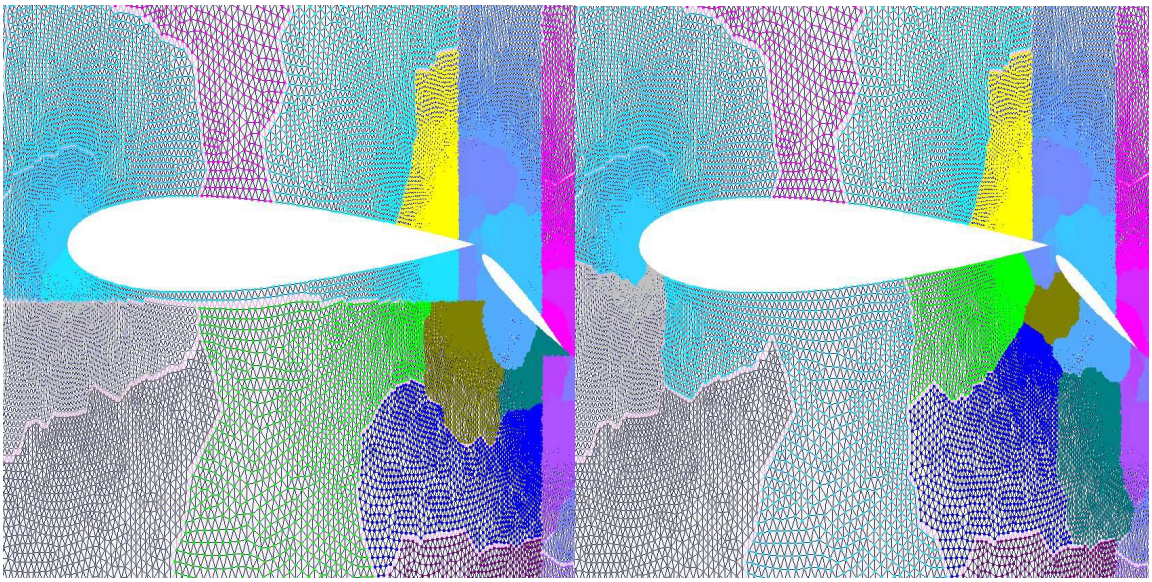


Рис. 32. Фрагменты результатов локального разбиения (слева) и сбора плохих групп доменов и их повторного разбиения (справа)

На Рис. 32 слева видно, что границы между доменами совпадают с границами между процессорами, полученными геометрической декомпозицией. На

Рис. 32 справа заметно, что часть доменов теперь расширяется за эти границы.

При считывании данных использовалась библиотека распределенного хранения и обработки тетраэдральных сеток (Суков С.А., ИПМ им. М.В.Келдыша РАН).

Параллельный инкрементный алгоритм декомпозиции графов реализован на MPI.

Рассмотрим алгоритм подробнее.

Изначально вершины распределяются по процессорам по номерам вершин: вершины с наименьшими номерами размещаются на нулевом процессоре, вершины из следующего интервала номеров на первом и т.д.

2.4.1 Геометрическая декомпозиция

Геометрическая декомпозиция работает только с координатами вершин, и для нее требуется меньше памяти, чем для всего инкрементного алгоритма. Поэтому создаются подгруппа процессоров и коммуникатор для вычисления геометрической декомпозиции, и производится копирование данных о координатах вершин на подгруппу процессоров. Вершины разбиваются параллельным алгоритмом геометрической декомпозиции сеток на число доменов, равное числу процессоров, на которых будет выполняться дальнейшее разбиение. Размеры доменов не равны, а определяются тем, сколько итоговых доменов будет сформировано на каждом из процессоров. Данные о результирующем разбиении сдвигаются на все процессоры. На каждом процессоре окажется массив с номерами новых процессоров вершин, расположенных в данный момент на этом процессоре. Подгруппа процессоров и коммуникатор освобождаются.

Хранение графа

В последовательном инкрементном алгоритме граф хранится в виде структуры, содержащей массив структур информации о каждой вершине и количество вершин. Информация о вершине включает в себя вес вершины, домен, число соседей, номера соседей и веса ребер. В параллельном инкрементном алгоритме способ хранения графа подобный с некоторыми дополнениями. На каждом процессоре хранится информация только о вершинах, принадлежащих этому процессору и их граничных вершинах. Назовем *граничными вершинами* процессора образа соседей вершин этого процессора, принадлежащих другим процессорам. У граничных вершин в соседях хранятся только вершины с данного процессора, поскольку только они нужны при локальных вычислениях на процессоре. Такой способ хранения позволяет получать от процессоров-владельцев граничных вершин только информацию об их весе и домене, принадлежности геометрической границе всего графа, а число соседей, номера соседей и веса ребер формировать локально. В рамках каждого процессора вершины нумеруются локально для поддержания непрерывности нумерации вершин и быстроты доступа к информации о соседях. Граф содержит также:

- массив глобальных номеров вершин
- массив номеров процессоров, которым принадлежат данные вершины
- массив принадлежности вершин геометрической границе всего графа
- число вершин в локальном графе
- общее число вершин в графе, распределенном между процессорами
- и другую информацию.

Принадлежность вершин геометрической границе всего графа определяется входными данными алгоритма. Поскольку на процессоре зачастую приходится просматривать информацию либо только о своих вершинах, либо только о граничных вершинах, в массиве вершин хранится сначала информация о своих вершинах, а в конце о граничных.

Перераспределение вершин и преобразование глобальных номеров соседей вершин в локальные

После геометрической декомпозиции происходит перераспределение вершин между процессорами в соответствии с результатами геометрической декомпозиции. Формируются локальные графы. Самым важным на данном этапе является алгоритм преобразования глобальных номеров соседей вершин в локальные. Замена двоичного (бинарного) поиска в отсортированном массиве на алгоритм, описанных ниже, позволила снизить время преобразования глобальных номеров соседей вершин со 170 сек. до 6 сек. на тетраэдральной сетке с $2 \cdot 10^8$ вершинами, а время добавления вершин в граф при перераспределении групп доменов, где используется подобный алгоритм, — с 620 сек. до 8 сек.

Алгоритм преобразования глобальных номеров соседей вершин выглядит следующим образом. В первый массив записываются наборы из глобальных номеров и *индексов* (локальных номеров) для всех вершин в графе. Во второй массив записываются наборы из трех чисел для каждого неизвестного соседа рассматриваемых вершин: { глобальный номер неизвестного соседа; индекс вершины; индекс соседа }. Индекс вершины, имеющей неизвестного соседа, и индекс соседа позволяют однозначно определить место ссылки в графе на данного соседа. Первый массив сортируется по глобальным номерам вершин, второй — по глобальным номерам неизвестных соседей. Используется последовательная сортировка из библиотеки параллельной сортировки, описан-

ной ранее. Далее осуществляется проход, начиная с нулевых наборов данных, одновременно по двум массивам:

- если в первом массиве все вершины пройдены, или текущий глобальный номер вершины больше текущего глобального номера неизвестного соседа из второго массива, то, если новая граничная вершина еще не была добавлена в процессе преобразования, она добавляется в граф. Ссылка на ее глобальный номер в графе меняется на ее индекс. А ей в соседи добавляется вершина, ссылающаяся на нее. Вес ребра выставляется таким же, как и у ссылающейся вершины, поскольку граф неориентированный. Если же новая граничная вершина уже была добавлена в граф (граничные вершины добавляются в конец, а второй массив отсортирован, поэтому это самая последняя вершина), то ей добавляется сосед, ссылающийся на нее. А ссылка у соседа меняется на ее индекс. В конце происходит переход к следующему неизвестному соседу во втором массиве.

- если текущий глобальный номер вершины из первого массива меньше текущего глобального номера неизвестного соседа из второго массива, происходит переход к следующей вершине в первом массиве.

- иначе (в этом случае текущий глобальный номер вершины из первого массива и текущий глобальный номер неизвестного соседа из второго массива совпадают) вместо ссылки на глобальный номер неизвестного соседа в граф записывается индекс вершины из первого массива. Если вершина из первого массива является граничной, то ей в соседи добавляется ссылающаяся на нее вершина. Затем происходит переход к следующему неизвестному соседу во втором массиве.

После преобразования глобальных номеров соседей вершин в локальные номера производится получение недостающей информации о созданных граничных вершинах. Поскольку изначально вершины были распределены по

процессорам по номерам, а результаты геометрической декомпозиции сохранены там же, где находились вершины изначально, то сначала происходит получение номеров новых процессоров-владельцев граничных вершин от их начальных владельцев, а потом получение недостающей информации от новых владельцев.

2.4.2 Присоединение малых блоков вершин

Алгоритм присоединения малых блоков вершин выглядит следующим образом:

1. На каждом процессоре локально происходит разбиение вершин на классы (блоки) и вычисление их мощностей. Инициализирующей вершиной класса становится первая *своя вершина* (не являющаяся граничной), не отмеченная другим классом. Класс формируется поиском в ширину от инициализирующей вершины, глобальный номер которой становится идентификатором данного класса. Классы не расширяются за границы процессоров. *Мощность класса* вычисляется как сумма весов входящих в него вершин.

2. Определение классов на передачу другим процессорам. Передаются классы, мощность которых меньше определенной доли (например, 0.1) от мощности максимального класса на данном процессоре. Направления передачи пока не определены. Направлением передачи остальных классов считается данный процессор, то есть они останутся на данном процессоре.

3. Получение идентификаторов классов граничных вершин от процессоров-владельцев.

4. Отправка информации о направлениях передачи своих классов на соседние процессоры. *Соседними процессорами* считаются процессоры, чьи вершины связаны с вершинами на данном процессоре.

5. Пока не определены направления передачи своих классов или классов с соседних процессоров Цикл:

   а). Получение информации о направлениях передачи классов с соседних процессоров. Если направления передачи всех своих классов известны, или не определились направления передачи каких-либо классов граничных вершин, происходит переход к следующей итерации цикла.

   б). Проверяются свои классы, для которых еще не известны направления передачи. Просматриваются граничные вершины, связанные с данным классом, до обнаружения первой, направление передачи класса которой известно. Направление передачи данного класса выставляется таким же, как и у класса найденной граничной вершины.

   в). Если были определены направления передачи каких-то из своих классов, информация о них посылается на соседние процессоры.

6. Определение частей классов на возврат для получения исходного числа вершин на процессорах.

7. Обмен классами вершин между процессорами. Осуществляется обмен вершинами, удаление лишних граничных вершин, создание новых и получение информации о них.

Определение частей классов на возврат происходит по следующему алгоритму:

1. Сбор информации обо всех классах на процессорах на каждом процессоре.

2. Определение сумм весов вершин, которые процессоры должны вернуть друг другу. Если класс мощности m должен быть передан процессором i на процессор j, то процессор j должен вернуть процессору i сумму весов вершин, равную m.

3. Определение связей между классами. На каждом процессоре просматриваются соседи своих вершин, и выставляется наличие связей между своими классами и другими. Общая информация о связях между классами собирается на всех процессорах.

4. Минимизация передач между процессорами. Если процессор i должен вернуть процессору j вес вершин, равный m_1, а процессор j должен процессору i вес вершин m_2, то при $m_1 > m_2$ считается, что процессор i должен вернуть процессору j вес вершин, равный $m_1 - m_2$, а процессор j ничего возвращать не должен. В случае $m_2 > m_1$ процессор j должен вернуть процессору i вес вершин, равный $m_2 - m_1$, а процессор i ничего возвращать не должен.

5. Отделение подклассов из классов на передачу. Два раза (две итерации) просматриваются по порядку все процессоры, и для каждого процессора – все те, кому он должен. Для каждой пары (i, j) (i-ый процессор должен вернуть j-ому некоторый вес вершин) производится поиск подклассов на отделение:

   - минимизируются передачи между процессорами i и j, поскольку могли добавиться новые передачи.

   - проверяется, были ли уже отделены подклассы, передающиеся в обратном направлении. Если были, то либо их мощности уменьшаются,

либо они удаляются. Соответственно уменьшается вес, который должен быть передан от *i*-го процессора *j*-ому.

- просматриваются классы, остающиеся на *i*-м процессоре с мощностями, большими некоторого минимального значения (например, предполагаемого суммарного веса вершин в двух доменах). Если класс связан с каким-то классом, остающимся, или передаваемым, на *j*-ый процессор, записывается, что из него должен быть выделен подкласс мощностью, не превышающей долг *i*-го процессора *j*-ому и не уменьшающей мощность данного класса ниже минимальной. Далее просматривается следующий класс, пока не будет набрана нужная сумма весов, или просмотрены все классы с *i*-го процессора.

- на первой итерации, если остался вес, который *i*-ый процессор должен передать *j*-ому, производится поиск процессора, через который можно будет передать подкласс. Просматриваются классы, остающиеся на *i*-м процессоре, из которых можно выделить нужный вес без уменьшения их мощности ниже минимальной, и проверяется, с классами каких процессоров они связаны. Затем просматриваются классы отмеченных процессоров до нахождения первого класса, не передаваемого ранее, мощность которого можно уменьшить на данный вес, и который связан с каким-нибудь классом на *j*-ом процессоре. Если процессор *q* был найден, для обработки на второй итерации записывается, что процессор *i* должен передать определенный вес процессору *q*, а процессору *q* добавляется такой же вес к передаче процессору *j*. На обеих итерациях вес, который процессор *i* должен был передать процессору *j*, обнуляется.

На данном этапе не производится выделение вершин из классов и предполагается, что не передаваемые ранее классы, достаточно большие, и нужные подклассы из них потом можно будет выделить.

6. Выделение вершин в подклассы. В свою очередь на процессоре *i* выделяются вершины во все подклассы, которые должны быть выделены для передачи другим процессорам. Для выделения подкласса из класса *a* процессора *i* для передачи процессору *j*, на процессоре *i* среди соседей граничных вершин с процессора *j* ищется вершина из класса *a*. Если она находится, подкласс набирается поиском в ширину от этой вершины по вершинам класса *a*. Если нет, записывается, что процессор *i* должен процессору *j* данный вес вершин для следующего отделения подклассов в пункте 5. На соседних процессорах обновляется информация о подклассах граничных вершин с процессора *i*, и наступает очередь следующего процессора на выделение вершин в подклассы.

Пункты 3 – 6 выполняются в цикле, пока есть ненулевые веса, которые процессоры должны вернуть друг другу.

В данном алгоритме все операции, за исключением сбора информации о классах, о связях между классами и выделения передаваемых вершин (в пунктах 1, 3 и 6), производятся локально на каждом процессоре без обменов с остальными процессорами, поскольку все необходимые данные есть на всех процессорах, и получаемые результаты идентичны.

Весь алгоритм присоединения малых блоков вершин запускается в цикле, пока находятся малые блоки (классы вершин) для пересылки на другие процессоры.

2.4.3 Инициализация доменов и локальное разбиение

После присоединения малых блоков вершин происходит переход к циклу, включающему в себя инициализацию доменов и локальное разбиение на домены на каждом из процессоров. Цикл выполняется, пока не найдено хорошее локальное разбиение, но не более заданного числа раз (например, 2). Критерии оценки качества разбиения см. ниже. Повторная инициализация

доменов позволяет исследовать разбиения, не найденные ранее при многократном освобождении части вершин в доменах и росте доменов.

Инициализация доменов

На процессоре может оказаться несколько больших блоков вершин, не связных между собой, поэтому в параллельную версию были внесены следующие изменения. Вершины разбиваются на связные классы (блоки) поиском в ширину. Определяется суммарный вес вершин в каждом классе, и запоминаются вершины, относящиеся к каждому классу. *Средний вес домена* для данного процессора оценивается, как отношение суммарного веса вершин на процессоре к числу доменов, которые должны быть сформированы на данном процессоре. Количество доменов, которое должно быть сформировано в каждом классе, определяется как отношение суммарного веса вершин в данном классе к среднему весу домена. Если остаток от деления больше 0.5, количество доменов увеличивается на единицу. В последнем классе будет сформировано количество доменов, оставшееся от числа доменов, которые должны быть сформированы на процессоре, после вычитания количеств доменов в предыдущих классах.

Для каждого класса из вершин, принадлежащих данному классу, случайным образом выбирается необходимое количество инициализирующих вершин. Инициализирующие вершины принадлежат соответствующим доменам, считающимся от единицы, а остальные нераспределенные вершины относятся к нулевому домену (домен нераспределенных вершин).

Для каждого класса вычисляется отношение суммарного веса вершин в классе к количеству доменов в классе. Вычисляется модуль максимального отклонения полученных отношений от среднего веса домена, который должен быть сформирован на данном процессоре. Результат используется в дальнейшем в качестве допустимого порогового отклонения *rhoPorog* в сум-

марных весах доменов на данном процессоре, поскольку средние веса доменов из разных классов могут отличаться.

Локальное разбиение

Нахождение локального разбиения выполняется в цикле с использованием рекурсии. Цикл прекращается, когда лучшее разбиение, найденное в цикле, не изменилось за заданное число итераций, например, 10. Каждая итерация цикла происходит по следующему алгоритму (подробное описание отдельных частей см. ниже):

Заполняются первые оболочки доменов. Запоминаются вершины, принадлежащие первым оболочкам доменов, поскольку именно они будут захватываться на первом этапе инкрементного роста доменов.

Выполняется инкрементный рост доменов.

Происходит локальное уточнение полученного разбиения.

Заполняются оболочки доменов. Определяются оболочки, которым принадлежат вершины, вычисляется наименьший номер первой несвязной оболочки в доменах, и подсчитывается число доменов с наименьшим номером первой несвязной оболочки.

Наименьший номер первой несвязной оболочки сравнивается с пороговым значением. Пороговое значение задается константой в параллельный инкрементный алгоритм декомпозиции графов, например, 4. Если номер первой несвязной оболочки в доменах больше, либо равен, пороговому значению, информация о разбиении запоминается, и происходит выход из цикла.

Иначе определяются плохие домены. Плохими доменами считаются домены, номер первой несвязной оболочки в которых меньше порогового значения, и их соседи.

Проверяется улучшение качества доменов. Качество доменов улучшилось, если либо номер первой несвязной оболочки увеличился, и число плохих доменов не увеличилось, либо число плохих доменов уменьшилось, и номер первой несвязной оболочки не уменьшился больше, чем на заданную величину (например, на 2). При этом не образовались несвязные домены, если предыдущее лучшее разбиение содержало только связные домены. Либо номер первой несвязной оболочки и число плохих доменов не изменились, а суммарный вес разрезанных ребер уменьшился, или уменьшилось число доменов с наименьшим номером первой несвязной оболочки, а суммарный вес разрезанных ребер не увеличился. Если качество разбиения улучшилось, оно запоминается, и номер итерации цикла приравнивается к единице, иначе номер итерации цикла при том же лучшем разбиении увеличивается на единицу.

Проверяется, нужно ли выделять плохие домены в подгруппу доменов и работать с ней отдельно. Группа доменов делится в том случае, если в исходной группе доменов число доменов больше, либо равно заданному (например, 10), число плохих доменов больше, либо равно другому заданному значению (например, 5), найденное разбиение было признанно лучшим, и число плохих доменов меньше некоторого коэффициента, умноженного на число доменов в исходной группе, не считая нулевой домен (например, 0.8).

Вначале разбиение группы доменов всегда ищется с параметром, что ее можно делить. В случае не нахождения хорошего разбиения, подгруппа плохих доменов увеличивается, и производится поиск разбиения без деления. В этом случае всегда будут освобождаться все домены из исходной подгруппы плохих доменов, а не только домены, являющиеся плохими для найденного разбиения.

Далее если число итераций цикла при одном и том же наилучшем разбиении достигло максимума, происходит возврат к наилучшему найденному

разбиению. Если цикл запускался с параметром, отвечающим за то, что группу доменов нельзя делить, то наилучшее разбиение считается хорошим, происходит выход из цикла, а улучшение разбиение оставляется на перераспределение групп доменов между процессорами. Если группу доменов делить можно, производятся заполнение оболочек для наилучшего разбиения, заполнение макро-графа доменов и определение плохих доменов. Если сумма числа плохих доменов и минимального числа доменов, которое можно прибавить к подгруппе (например, 2), больше числа доменов в исходной группе доменов, разбиение считается плохим, и происходит выход из цикла. В этом случае улучшение разбиения переносится на внешний уровень рекурсии, если исходная группа доменов ранее была выделена в подгруппу. Иначе к подгруппе плохих доменов добавляются соседние домены (домены, вершины в которых имеют соседей в подгруппе доменов). Если подгруппа плохих доменов увеличилась, освобождается часть вершин плохих доменов и выполняется рекурсивный вызов поиска локального разбиения в исходной группе доменов с указанием подгруппы плохих доменов, и того, что ее нельзя делить. Вне зависимости от того, будет ли найдено хорошее разбиение, производится выход из цикла. Если подгруппа плохих доменов не увеличилась, улучшение разбиения оставляется на перераспределение групп доменов, разбиение отмечается, как хорошее и происходит выход из цикла.

Если изначально предполагалось, что сетка такова, что в доменах недостаточно оболочек для получения нужного номера первой несвязной оболочки, то в этом случае выполняется заданное число итераций для нахождения наилучшего возможного разбиения. Учитываются итерации, выполненные на всех уровнях рекурсии. И если выполненное число итераций стало больше, либо равно, заданному числу, производится возврат к наилучшему разбиению, и происходит выход из цикла. При этом разбиение отмечается, как плохое, чтобы потом не происходило перераспределение групп доменов.

Если было определено, что группу доменов делить не надо, освобождается часть вершин плохих доменов (или изначально отмеченных, как плохие, если группу делить нельзя), и выполняется переход к следующей итерации цикла.

Если группа была определена, как подлежащая делению, то запускается внутренний цикл. В этом цикле, если группа идет на деление (этот параметр может измениться на следующей итерации), строятся граф и макро-граф подгруппы плохих доменов, заполняются первые оболочки доменов и новый макро-граф доменов. Освобождается часть вершин плохих доменов. Выполняется рекурсивный вызов поиска локального разбиения на новых графе и макро-графе подгруппы доменов (с указанием плохой подгруппы, если подгруппу делить нельзя). Если найденное разбиение оказалось плохим, отмечается, что далее группу доменов делить не нужно. Производятся восстановления графа и макро-графа подгруппы доменов (возврат к исходной группе доменов) и определение плохих доменов. Если группа не идет на деление, происходит освобождение части вершин плохих доменов, и выполняется рекурсивный вызов поиска локального разбиения на исходных графе и макро-графе доменов. Для разбиений, найденных и при делении, и без деления, подсчитывается число плохих доменов. Если плохих доменов нет, происходит выход из внутреннего и внешнего циклов. В противном случае заполняются первые оболочки доменов и макро-граф доменов, и заново определяется, нужно ли делить группу доменов. Группа доменов делится, если исходное число доменов в группе больше заданного, и число плохих доменов меньше коэффициента, умноженного на число доменов в исходной подгруппе, не считая нулевой домен.

Учет числа плохих доменов, числа доменов с минимальным номером первой несвязной оболочки и суммарного веса разрезанных ребер были введены в параллельный инкрементный алгоритм для учета других характери-

стик улучшения качества разбиения. В последовательном алгоритме учитывался только номер первой несвязной оболочки, и выход из цикла производился, когда этот номер становился больше, либо равен, пороговому значению. Также работа с подгруппами доменов была введена в параллельный инкрементный алгоритм для улучшения сходимости всего алгоритма. В результате в параллельном алгоритме, пока группа доменов не пошла на деление, освобождаются плохие домены, а в инкрементном росте происходит изменение всех доменов в группе. Затем, когда плохих доменов стало меньше, выделяется подгруппа плохих доменов, и меняется только она, остальные домены не затрагиваются. На начальном этапе изменение всех доменов в группе доменов позволяет исследовать больше вариантов, а затем выделение подгруппы доменов помогает локализовать поиск хорошего разбиения.

Заполнение первых оболочек доменов

Изначально все вершины в графе обозначаются, как неотмеченные. Далее осуществляется проход по своим вершинам. У каждой неотмеченной вершины просматриваются все соседи с данного процессора до нахождения первого, домен которого не совпадает с доменом данной вершины. Если важны связи между всеми доменами на процессорах, рассматриваются все соседи. Если такой сосед найден, и вершина, и ее сосед, отмечаются. После завершения просмотра всех своих вершин осуществляется еще один проход по своим вершинам, локальные номера всех отмеченных вершин помещаются в массив первых оболочек доменов *firstEnv*. Запоминание первых оболочек было введено в параллельный инкрементный алгоритм для ускорения алгоритма инкрементного роста и других частей, где теперь вместо проходов по всем своим вершинам в графе используются проходы по первым оболочкам доменов.

Макро-граф доменов

Для хранения информации о доменах была создана структура макро-граф доменов, включающая в себя:

- общее число доменов на процессорах *ng*
- число доменов, формируемых на данном процессоре *ngHereF*
- число доменов *ngHere* на данном процессоре, рассматриваемое в настоящий момент (число доменов в макро-графе подгруппы доменов) (о формировании графа и макро-графа подгруппы доменов см. ниже)
- массив глобальных номеров доменов на данном процессоре, включая нулевой домен нераспределенных вершин
- массив локальных номеров доменов на данном процессоре, рассматриваемых в данных момент (входящих в макро-граф подгруппы доменов), включая нулевой домен нераспределенных вершин
- допустимое пороговое отклонение *rhoPorog* в суммарных весах доменов на данном процессоре
- массив *rho*, содержащий информацию о суммарном весе вершин в каждом домене
- суммарный вес разрезанных ребер
- двумерный массив *edg*, где в каждой ячейке *edg[i][j]* записан суммарный вес вершин домена *i*, инцидентных домену *j* (имеющих соседей в домене *j*).

Заполнение макро-графа доменов

Если в соответствующем параметре указано, что нужно пересчитать массив *rho*, то массив *rho* обнуляется, и осуществляется проход по всем своим вершинам в графе. Веса вершин прибавляются к весам их доменов в *rho*.

Обнуляются суммарный вес разрезанных ребер и массив *edg*. Далее осуществляется проход по всем вершинам из первых оболочек доменов (в массив *fristEnv* ранее были записаны только свои вершины). Для каждой вершины *v* просматриваются ее соседи с данного процессора. Если заполняется информация о связях между всеми доменами на процессорах, рассматриваются все соседи. Если домен *j* соседа не совпадает с доменом *i* вершины *v*, вес ребра между ними добавляется к суммарному весу разрезанных ребер. Также вес вершины *v* добавляется в ячейку *edg[i][j]*, если ее вес еще не был добавлен в данную ячейку. Либо в ячейку *edg[i][j]* добавляется вес рассматриваемого ребра, если собирается информация о разрезанных ребрах.

Если собирается информация о связях между всеми доменами на процессорах, общий вес разрезанных ребер суммируется по процессорам, а в ячейки *edg[i][j]*, где *i* – домен граничных вершин, *j* – *свой домен* (домен своих вершин), копируется информация из ячеек *edg[j][i]*.

Вес разрезанных ребер делится на два, поскольку каждое ребро учитывалось и у вершины, и у соседа.

Проход по первой оболочке был введен в параллельный инкрементный алгоритм декомпозиции, в последовательной версии рассматривались все вершины в графе. Также в параллельный инкрементный алгоритм был введен учет весов вершин и ребер, в последовательном инкрементном алгоритме веса вершин и ребер не учитывались при заполнении макро-графа доменов и в других функциях.

Инкрементный рост доменов

Инкрементный рост доменов (захват вершин) происходит следующим образом. Заполняется макро-граф доменов (массив *rho* пересчитывается, рассматриваются только соседи с данного процессора, в *edg[i][j]* заносится информация о суммарном весе вершин домена *i*, инцидентных домену *j*). По всем доменам в макро-графе подгруппы доменов дисбаланс вычисляется, как максимальное отклонение весов доменов в *rho* от среднего веса домена (отношение суммарного веса вершин на процессоре к числу доменов на данном процессоре). Далее запускается цикл:

- заполняется массив, содержащий информацию о том, какой вес вершин должен быть перенесем между доменами. Если никаких возможных переносов не обнаружено, происходит выход из цикла.
- выполняется один этап захвата вершин
- обновляется макро-граф доменов (*rho* не пересчитывается)
- вычисляется дисбаланс
- если в нулевом домене больше вершин не осталось, и дисбаланс меньше заданного, происходит выход из цикла.

Пороговое значение на дисбаланс весов доменов определяется суммой *rhoPorog* на данном процессоре и порогового значения на дисбаланс числа вершин (например, 10), умноженного на максимальный вес вершины в графе. Пороговое значение на дисбаланс числа вершин задается константой в инкрементный алгоритм.

Двумерный массив *c* весов вершин, которые нужно перенести между доменами, заполняется по следующему алгоритму. Изначально массив обнуляется. В некоторый массив *r* копируется информация из *rho*. Выполняется

цикл, на каждой итерации которого для всех своих доменов *i* в макро-графе подгруппы доменов рассматриваются соседние свои домены с номерами *j* > *i*. Домен *j* является соседним, если *edg[i][j] > 0*. Значение в ячейке *r[i]* уменьшается на *d = (r[i] – r[j]) / 2*, а *r[j]* увеличивается на *d*. Если *r[i] < r[j]*, величина *d* будет отрицательной. Происходит попытка сравнять веса доменов *i* и *j*. Значение в ячейке *c[i][j]* увеличивается на *d*, а в *c[j][i]* уменьшается на *d*. Подразумевается, что если *c[i][j] > 0*, из домена *i* нужно перенести вес вершин, равный *c[i][j]*, в домен *j*, а если *c[i][j] < 0*, то вес вершин, равный *|c[i][j]|*, нужно перенести из домена *j* в домен *i*. Значения *|d|* суммируются в некоторой переменной, отвечающей за общий вес вершин, перенесенных между доменами. Цикл прекращается, когда после рассмотрения всех своих доменов *i* и *j*, с номерами *j > i* , общий вес перенесенных вершин оказывается меньше числа своих доменов на процессоре, умноженного на максимальный вес вершины в графе. Далее для всех своих доменов *i* и *j*, с номерами *j > i*, если *c[i][j] > 0* и *c[i][j] > edg[i][j]*, *c[i][j]* выставляется равным *edg[i][j]*, поскольку нельзя перенести больший вес, чем суммарный вес вершин в домене *i*, инцидентных домену *j*. Аналогичная проверка проводится и для *c[j][i]*. Затем проверяется равенство *|c[i][j]|* и *|c[j][i]|*, и, в случае неравенства, модули значений приравниваются к минимальному модулю, знаки значений остаются противоположными. Значения *c[0][i]* приравниваются к *edg[0][i]* для всех своих доменов *i*.

Один этап захвата вершин происходит следующим образом. Производится проход по первым оболочкам доменов в массиве *firstEnv*. Для каждой вершины *v* просматриваются все соседи с данного процессора. Если в массиве *c* разрешены переносы из домена *i* вершины *v* в домен *j* соседа *b* (переносы в тот же самый домен не разрешены), то номер домена *j* записывается в отсортированный массив номеров доменов соседей с помощью двоичного поиска. Вес ребра между вершинами *v* и *b* добавляется в выигрыш от переноса вершины *v* в домен *j*. Выигрыш в весе разрезанных ребер от переноса верши-

ны *v* в домен *j*, равен сумме весов ребер, соединяющих вершину *v* с вершинами из домена *j*, с вычетом суммы весов ребер, соединяющих вершину *v* с вершинами из домена *i*. Поскольку исходный домен у вершины один, для сравнения выигрышей от переноса вершины *v* в другие домены вычитание весов ребер, соединяющих вершину *v* с соседями из своего домена, можно не производить. После просмотра всех соседей, просматривается массив номеров доменов соседей, и перенос вершины *v* осуществляется в тот домен, выигрыш от переноса в который является наибольшим, или при равных выигрышах в домен, вес которого оказался меньшим. Обновляется информация о весах исходного и нового доменов вершины *v* в *rho*, и о соответствующих переносах в массиве *c*. После окончания просмотра всех вершин из первых оболочек, у вершин, которые были перемещены, и их соседей, просматриваются все соседи до нахождения первого из другого домена. Если такой сосед найден, вершина отмечается. Затем все отмеченные вершины помещаются в массив первых оболочек доменов для следующего этапа захвата. Введение учета выигрышей в весе разрезанных ребер на этапе захвата вершин в параллельный инкрементный алгоритм декомпозиции графов улучшило сходимость самого алгоритма.

Локальное уточнение доменов

Алгоритм локального уточнения, реализованный в параллельном инкрементном алгоритме, во многом совпадает с алгоритмом локального уточнения, описанным ранее в обзоре существующих алгоритмов, с некоторыми отличиями, как от описанного алгоритма, так и от реализованного в последовательной версии. Во внутреннем цикле выбираются перемещения количества вершин, меньшего минимума из числа итераций внутреннего цикла, потребовавшегося на предыдущей итерации внешнего цикла для получения наилучшего разбиения, умноженного на некоторый коэффициент (например, 2), и некоторой доли (например, 0.3) от числа своих вершин в графе подгруп-

пы доменов. В последовательной версии рассматривается количество перемещений, равное числу вершин в графе. Также в параллельную версию было введено прекращение внутреннего цикла, когда разница между максимальным выигрышем, полученным во внутреннем цикле, и текущим выигрышем, становится больше максимального возможного выигрыша от перемещения одной вершины, умноженного на некоторый коэффициент (например, 2). Максимальный и текущий выигрыши вычисляются как сумма выигрышей, полученных на цепочке перемещений (максимальный – до получения наилучшего разбиения). Максимальный возможный выигрыш от перемещения одной вершины вычисляется, как максимальная сумма весов ребер, исходящих от одной вершины, и отражает ситуацию, когда вершина переносится в домен, в котором находятся все ее соседи. Внешний цикл завершается, когда максимальный выигрыш, полученный во внутреннем цикле, становится меньше некоторого порогового значения (например, 10).

В последовательной версии вершины переносятся из доменов с большим весом, в домены с меньшим весом. В параллельном инкрементном алгоритме вес отдающего домена может быть меньше веса принимающего домена на *rhoPorog* для учета возможного дисбаланса вершин в доменах в несвязанных между собой частях локального графа.

Информация о выигрышах соседей перемещенной вершины обновляется только у своих соседей (соседей с данного процессора).

В последовательной версии в конце итерации внешнего цикла обновляется информация о выигрышах всех вершин. В параллельном инкрементном алгоритме информация о выигрышах всех вершин обновляется только, если было перемещено больше половины своих вершин. В противном случае выигрыши пересчитываются только у перемещенных вершин и их соседей. Это улучшило сходимость локального уточнения и уменьшило результирующий суммарный вес разрезанных ребер (при разбиении тетраэдральной сетки, со-

стоящей из 75790 вершин, на 50 доменов на 5 процессорах суммарный вес разрезанных ребер после первого локального уточнения уменьшился с 1369 до 1242).

Для уменьшения объема выделяемой памяти в параллельной версии введена локальная нумерация доменов из подгруппы доменов.

Алгоритм локального уточнения плохо масштабируется по числу доменов и является наиболее затратной по времени частью всего инкрементного алгоритма. Поэтому в параллельный инкрементный алгоритм декомпозиции графов были добавлены еще три варианта локального уточнения в попытке ускорить весь алгоритм. Первым вариантом является жадное уточнение (greedy refinement), описанное выше при обзоре реализаций алгоритма локального уточнения в различных пакетах. Во втором и третьем вариантах подгруппа доменов делится на небольшие подгруппы, и они уточняются по отдельности. Во втором варианте определяются граничные домены (домены, в которых есть вершины, принадлежащие геометрической границе всего графа, или домены, которые связаны с доменами на других процессорах). Вначале инициализирующими доменами в группах становятся неотмеченные домены, не являющиеся граничными. Отмечаются домены, попавшие в группы ранее. В группы добавляются все соседи инициализирующих доменов. Если после прохода по всем доменам из исходной подгруппы доменов, остались неотмеченные граничные домены, они становятся инициализирующими для следующих групп. В третьем варианте группы доменов формируются вокруг пар доменов, имеющих наибольший вес разрезанных ребер между ними. Группы доменов включают в себя определенное количество доменов (например, 16). Группы образовываются число раз, равное отношению количества доменов в исходной подгруппе к количеству доменов в малой группе, умноженному на некоторый коэффициент (например, 2). Группы формируются одним из двух способов. В первом, сначала к двум доменам добавляются со-

седи, которые связаны с обоими доменами. Если таких соседей не хватает, добавляются соседи, имеющие связь с одним из двух доменов. Во втором, в группу доменов вокруг пары доменов добавляются домены, связанные с наибольшим числом доменов из группы. В обоих способах, если в группе доменов все еще не хватает доменов, формируется еще одна группа доменов вокруг другой пары доменов с наибольшим весом разрезанных ребер. Использование жадного уточнения не дало уменьшения суммарного веса разрезанных ребер после уточнения, поскольку выигрыши от перемещений вершин в различные домены подобным образом уже учитывались при инкрементном росте доменов. При уточнении небольших подгрупп доменов весь алгоритм локального разбиения находил худшие разбиения и хуже сходился. Выигрыш по времени при вызове измененного локального уточнения давал выигрыш по времени на одной итерации нахождения локального разбиения, но проигрыш в количестве итераций, а, следовательно, и во времени нахождения локального разбиения. Запуск обычного локального уточнения после уточнения малых групп доменов, или жадного уточнения, тоже не дал результатов. Поэтому в настоящий момент в параллельном инкрементном алгоритме декомпозиции графов используется обычное локальное уточнение, описанное выше.

Заполнение оболочек

Для каждого домена из подгруппы доменов оболочки, которым принадлежат его вершины, определяются следующим образом. Производится проход по всем своим вершинам в подграфе, и заполняется список вершин (подразумевается массив), принадлежащих рассматриваемому домену *i*. Далее проходом по списку определяются вершины, принадлежащие первой оболочке домена. В первую оболочку домена попадают вершины из геометрической границы всего графа, или вершины, у которых есть соседи в других доменах на своем, или чужих процессорах. Вершины из первой оболочки кла-

дутся в очередь. Далее запускается цикл, пока очередь не пуста, на каждой итерации которого из очереди достается вершина *v* с номером оболочки *env(v)*. Просматриваются все соседи вершины *v*, принадлежащие домену *i*. Если у соседа *b* номер оболочки еще не заполнен, он приравнивается к значению *(env(v) + 1)*, и сосед *b* кладется в очередь. Если номер оболочки вершины *b env(b) > (env(v) + 1)*, то *env(b)* приравнивается к *(env(v) + 1)*, и вершина *b* помещается в очередь, если в данный момент ее там нет. Если *env(v) > (env(b) + 1)*, *env(v)* приравнивается к *(env(b) + 1)*. И после просмотра всех соседей вершины *v*, если номер оболочки вершины *v* менялся, она снова кладется в очередь, поскольку номера оболочек ее соседей нужно обновить.

Затем проходом по списку вершин домена *i* определяется число оболочек в рассматриваемом домене. Проводится проверка всех оболочек на связность. Просматриваются все свои вершины в подграфе, считается количество вершин, принадлежащих оболочке *j* домена *i*, и произвольная вершина *v* из данной оболочки кладется в очередь. Далее поиском в ширину просматриваются остальные вершины из оболочки *j* домена *i*, и считается их количество. Если количество вершин, просмотренных поиском в ширину от произвольной вершины *v* из оболочки *j* домена *i*, оказалось равным общему количеству вершин в оболочке *j* домена *i*, то оболочка считается связной, в противном случае – нет. Запоминается номер первой несвязной оболочки в домене *i*. Если все оболочки связны, запоминается (число оболочек в домене + 1).

Вычисляется наименьший номер первой несвязной оболочки для доменов из подгруппы доменов, и подсчитывается число доменов с таким номером первой несвязной оболочки.

Определение плохих доменов

Осуществляется проход по всем доменам из подгруппы доменов. Если номер первой несвязной оболочки домена меньше заданного порогового зна-

чения (например, 4), домен отмечается, как плохой. Все неотмеченные соседи рассматриваемого домена из подгруппы доменов также отмечаются как плохие. Домен *j* является соседним для домена *i*, если *edg[j][i] > 0*, то есть если суммарный вес вершин в нем, инцидентных домену *i*, больше нуля. Подсчитывается количество плохих доменов в подгруппе доменов.

Освобождение части вершин плохих доменов

Обновляется информация в *rho* о суммарном весе вершин в доменах из подгруппы доменов.

Просматриваются все свои вершины из подграфа. Если домен *i* вершины *v* отмечен, как плохой, и номер оболочки вершины меньше заданного порогового значения, проверяется, является ли вес домена, большим веса вершины, то есть не осталась ли эта вершина последней в домене. Если вес домена *i* больше веса вершины, вершина *v* переносится в нулевой домен нераспределенных вершин. Вес вершины *v* отнимается от веса домена *i* и прибавляется к весу нулевого домена. Пороговое значение на номера освобождаемых оболочек задается константой в инкрементный алгоритм, например, 7, и обычно оно больше порогового значения на номера первых несвязных оболочек в доменах.

Все плохие домены разбиваются на классы вершин, не связные между собой. Изначально все вершины считаются неотмеченными. Осуществляется проход по своим вершинам в подграфе. Каждая неотмеченная вершина из домена *i* становится инициализирующей вершиной следующего класса. Она помещается в очередь, и далее происходит поиск в ширину остальных вершин из данного класса. Пока очередь не пуста, из нее достается вершина *v*, и просматриваются все соседи данной вершины из домена *i*. Неотмеченные соседи отмечаются и кладутся в очередь. В течение поиска в ширину, суммируется общий вес вершин в классе. Вычисляется класс с наибольшим весом, и

остальные классы освобождаются поиском в ширину от инициализирующих вершин. Суммарный вес освобожденных вершин отнимается от веса домена *i* и прибавляется к весу нулевого домена.

Построение графа и макро-графа подгруппы доменов

В макро-граф подгруппы доменов копируются значения и ссылки на массивы исходного макро-графа. Из массива локальных номеров доменов на данном процессоре удаляются локальные номера доменов, отсутствующих в подгруппе (их будет несложно восстановить при переходе к предыдущему уровню рекурсии), и вычисляется новое число доменов *ngHere* в подгруппе.

В исходном графе все свои вершины, домены которых принадлежат подгруппе доменов, сдвигаются к границе между своими и граничными вершинами. Каждый сдвиг производится как взаимная смена мест двух вершин, соответственно меняются ссылки на эти вершины у соседей. Запоминаются индексы *first*, начала вершин из подгруппы доменов, и last – последней вершины, принадлежащей подгруппе доменов (ею становится последняя своя вершина). Граничные вершины в новом графе отсутствуют, подгруппа доменов рассматривается как отдельный граф. Массивы графа подгруппы доменов являются ссылками на ячейки с индексами *first* в массивах исходного графа. Число вершин в новом графе равно *(last - first + 1)*. У вершин графа подгруппы доменов удаляются ссылки на соседей не из подгруппы доменов, а ссылки у соседей на номера данных вершин приравниваются к значениям *(-vNum - 1)*, где *vNum* – глобальный номер вершины из подгруппы доменов. Вершины из подгруппы доменов, у которых удалялись ссылки на соседей не из подгруппы доменов, отмечаются, как принадлежащие геометрической границе нового графа, а их глобальные номера запоминаются.

Восстановление графа и макро-графа

Создается массив *vNs*, в который для каждой вершины из графа подгруппы доменов заносится набор { глобальный номер вершины; ее локальный номер в исходном графе }. Массив сортируется по глобальным номерам вершин. Также по глобальным номерам вершин сортируется массив, содержащий информацию о вершинах, у которых менялся признак принадлежности геометрической границе при создании графа подгруппы доменов. Осуществляется проход одновременно по двум массивам для поиска локальных номеров вершин с измененным признаком принадлежности геометрической границе. Найденные вершины отмечаются, как не принадлежащие геометрической границе. В графе подгруппы доменов номера соседей вершин увеличиваются на *first* для приведения в соответствие их локальных номеров положению в исходном графе. Просматриваются вершины из исходного графа, не входящие в подгруппу доменов. Домены своих вершин добавляются в массив локальных номеров доменов на данном процессоре в макро-графе доменов. Проверяются соседи данных вершин, и при обнаружении соседа с отрицательным номером *vNum*, в массив неизвестных соседей заносится набор { *(- vNum – 1)*; индекс вершины; индекс соседа }, где *(- vNum – 1)* – глобальный номер соседа. Также в этом случае домен своей вершины добавляется к подгруппе доменов, если разбиение подгруппы доменов оказалось плохим. Затем массив неизвестных соседей сортируется по глобальным номерам вершин. Далее осуществляется проход одновременно по массиву неизвестных соседей и по массиву *vNs* номеров вершин из графа подгруппы доменов для нахождения локальных номеров неизвестных соседей. Номера неизвестных соседей в исходном графе меняются на их локальные номера, а к ним в соседи добавляются вершины из исходного графа. Вес ребра приравнивается к весу ребра между вершиной в исходном графе и неизвестным соседом. Если найденное разбиение оказалось хорошим, все домены удаляются из отмеченных доменов на последующее разбиение.

2.4.4 Перераспределение плохих групп доменов и повторное локальное разбиение

После выполнения инициализаций доменов и нахождения локальных разбиений со всех процессоров собирается информация о том, являются ли полученные разбиения хорошими. Если на каком-то процессоре полученное разбиение является плохим, то общее разбиение считается плохим.

С соседних процессоров собирается информация о доменах граничных вершин. Домены нумеруются локально и добавляются в макро-граф доменов. Первые оболочки доменов и макро-граф доменов заполняется локально с учетом связей между своими доменами и граничными. Связи между граничными доменами не заполняются. Общий вес разрезанных ребер суммируется по процессорам. Локальные номера доменов вершин переводятся в глобальные.

Выполняется перераспределение плохих групп доменов, описанное ниже.

На каждом процессоре, на который были перераспределены плохие домены, перебивается только подгруппа плохих доменов. Строятся граф и макро-граф подгруппы плохих доменов, и выполняется локальное разбиение. Затем производятся восстановления графа и макро-графа подгруппы доменов. Выполняется обмен информацией о доменах граничных вершин между процессорами. Номера доменов вершин переводятся в глобальные.

Информация о доменах вершин перераспределяется между процессорами в соответствии с распределением вершин по процессорам вначале параллельного инкрементного алгоритма декомпозиции графов.

Перераспределение плохих групп доменов

Перераспределение плохих групп доменов происходит по следующему алгоритму:

1. Глобальное определение плохих доменов. Домен отмечается, как плохой, либо если номер первой несвязной оболочки домена меньше заданного порогового значения (например, 4), либо если он связан с доменами на других процессорах (*edg[i][j] > 0*, где *i* - свой домен, *j* – граничный домен). Соседи отмеченного домена также отмечаются, как плохие. Информация о плохих доменах собирается со всех процессоров и подсчитывается суммарное число плохих доменов. Если локальное разбиение оказалось хорошим, то сначала отмечаются, как плохие, и домены с маленькими номерами первой несвязной оболочки, и домены на границах между процессорами, и их соседи. Если же локальное разбиение плохое, то отмечаются только домены на границах между процессорами и их соседи. В обоих случаях, если отмеченных доменов оказалось больше некоторой доли от всех доменов (например, 0.4), производится повторное отмечание плохих доменов. В этот раз отмечаются только домены с маленькими номерами первой несвязной оболочки и их соседи. Если плохих доменов оказалось много и в этот раз, перераспределение плохих групп доменов и повторное локальное разбиение не производятся.

2. Для всех доменов определяются процессоры, которым принадлежат домены.

3. На каждом процессоре поиском в ширину по соседним плохим доменам от инициализирующих плохих доменов формируются связные классы плохих доменов. Глобальные номера инициализирующих доменов становятся идентификаторами классов. Граничные домены также включаются в классы.

4. Каждым процессором на соседние процессоры посылаются идентификаторы классов плохих граничных доменов. При получении других классов своих плохих доменов на процессоре заполняется информация

о совпадении классов. Информация заносится в двумерный массив, каждая строка которого содержит количество и идентификаторы совпадающих классов. Со всех процессоров информация о совпадении классов посылается на нулевой процессор, где формируется итоговый массив совпадающих классов, который затем рассылается на все процессоры. Идентификаторы совпадающих классов приравниваются к идентификатору первого класса из соответствующей строки массива совпадающих классов. Собирается информация о классах всех доменов, и на каждом процессоре локально для каждого класса подсчитывается его мощность (сколько доменов принадлежит данному классу).

5. Определяются процессоры, на которые будут передаваться классы плохих доменов. Если все домены из класса расположены на одном процессоре, то класс доменов там и останется. Если домены расположены на разных процессорах, класс будет собираться на процессоре, на который потребуется передавать меньше доменов, с учетом передач доменов для сбора классов, направления передач которых уже определены. При равных передачах класс будет собираться на процессоре, на котором итоговое количество плохих доменов окажется меньше. После определения направлений передач классов, подсчитывается, сколько дополнительных доменов процессоры должны вернуть друг другу для восстановления исходного числа доменов на процессорах.

6. Минимизация дополнительных передач между процессорами. Если процессор i должен вернуть процессору j количество доменов, равное m_1, а процессор j должен процессору i количество доменов m_2, то при $m_1 > m_2$ считается, что процессор i должен вернуть процессору j количество доменов, равное $m_1 - m_2$, а процессор j ничего возвращать не должен. В случае $m_2 > m_1$ процессор j должен вернуть процессору i ко-

личество доменов, равное $m_2 - m_1$, а процессор *i* ничего возвращать не должен.

7. Минимизация передач дополнительных доменов, исходя из исходного неравенства числа доменов на процессорах. Подсчитывается исходное количество доменов на процессорах. Минимальное возможное число доменов на процессоре приравнивается к *ng / nproc* (деление без остатка), где *ng* – общее число доменов, *nproc* – число процессоров. Если *ng / nproc* нацело не делится, то максимальное возможное число доменов на процессоре считается равным *(ng / nproc + 1)*, иначе *ng / nproc*. Номера всех процессоров помещаются в очередь. Для каждого процессора *i*, извлеченного из очереди, проверяются его передачи всем процессорам *j*. Положительные передачи уменьшаются настолько, насколько можно увеличить число доменов на процессоре *i* и уменьшить число доменов на процессоре *j*. Если на процессоре *j* можно увеличить число доменов, и он уже извлечен из очереди, процессор *j* помещается туда снова. Действия повторяются, пока очередь не окажется пустой.

8. Два раза (две итерации) рассматриваются по порядку все процессоры:

   - Для каждого процессора *i* просматриваются все процессоры, кому он должен. Для каждой пары *(i, j)* (*i*-ый процессор должен вернуть *j*-ому некоторое количество дополнительных доменов) *i*-ым процессором производится поиск доменов для передачи. Домены процессора *i* считаются *связанными с процессором j*, если они связаны с нераспределенными доменами на процессоре *j*, или с доменами, которые будут переданы на процессор *j*. Домены процессора *i* считаются *хорошими дополнительными доменами*, если они не связаны с плохими доменами, которые будут собраны не на процессоре *j*. Сначала процессором *i* рассматриваются только нераспределенные хорошие дополнительные домены, связанные с процессором *j*. Они помещаются в очередь. Если в

очередь не помещено ни одного домена, туда помещается нераспределенный плохой дополнительный домен, связанный с процессором *j*. Затем для каждого домена, извлеченного из очереди, просматриваются соседние с ним хорошие дополнительные домены. Если для передачи нужно больше доменов, процесс повторяется, но в очередь кладутся все нераспределенные домены, соседние с процессором *j*, а у доменов, извлеченных из очереди, просматриваются все нераспределенные соседи. Дополнительные домены, не являющиеся хорошими, не берутся сразу на случай, если после перераспределения доменов к плохим доменам понадобится добавить их соседей для рассмотрения. При написании алгоритма такая ситуация считалась возможной, хотя позже она не возникла.

- После рассмотрения процессором *i* доменов для передач другим процессорам на другие процессоры посылается информация о новых направлениях передачи доменов с процессора *i*, и об оставшихся долгах процессора *i* другим процессорам, для которых не было найдено дополнительных доменов.

- На второй итерации, если остались долги процессора *i* другим процессорам, они обнуляются. На первой итерации каждым процессором подсчитывается количество своих нераспределенных доменов, связанных с каждым процессором. Если количество доменов мало (например, меньше 2), оно обнуляется. Процессором *i* на другие процессы передается информация о связях его доменов. Все процессоры, с которыми связаны нераспределенные домены процессора *i*, посылают информацию о связях своих доменов на процессор *i*. Для передачи долга процессора *i* процессору *j* выбирается такой процессор *k*, на котором наибольшее число нераспределенных доменов связано с процессором *j*. Так для каждого процессора *j*. Информация о выбранных процессорах

для передачи распространяется на все процессоры. Если процессор *k* для передачи был найден, отмечается, что теперь процессором *i* количество дополнительных доменов, необходимое для передачи на процессор *j*, должно быть отдано процессору *k*, а процессором *k* такое же число дополнительных доменов должно быть отдано процессору *j*.

9. На каждом процессоре составляется новый макро-граф доменов, подсчитываются число плохих доменов, наименьший номер первой несвязной оболочки и число доменов с наименьшим номером первой несвязной оболочки.

10. Выполняется передача вершин на другие процессоры и встраивание их в графы.

11. Номера доменов вершин преобразовываются в локальные, и вычисляется новый *rhoPorog*. Определение *rhoPorog* происходит так же, как и при инициализации доменов, только теперь вершины уже распределены по доменам, и число доменов в классах просто подсчитывается.

2.4.5 Принятые решения

Большинство решений, принятых при разработке параллельного инкрементного алгоритма декомпозиции графов, были получены опытным путем. Проверялись различные варианты, некоторые сработали, некоторые – нет:

- В первом варианте распараллеливания инкрементного алгоритма домены расширялись за границы процессоров. Но в процессе инкрементного роста приходилось синхронизировать между процессорами информацию о весе вершин, перенесенных в другие домены. Частая синхронизация приводила к большим временным потерям, редкая ухудшала сходимость алгоритма инкрементного роста. Частые синхронизации потребовались бы и в процессе локального уточнения. Проблема была решена запрещением расширения доменов за границы процессоров,

что повлекло за собой введение геометрической предекомпозиции для распределения вершин между процессорами. Поскольку инкрементный рост доменов имеет общие черты с ориентированной диффузией, другим способом решения данной проблемы могла бы стать замена ориентированной диффузии на неориентированную с сохранением возможности расширения доменов за границы процессоров. А при локальном уточнении доменов можно было бы использовать жадное уточнение, которое легче распараллеливается. Но введение жадного уточнения в инкрементный алгоритм было проверено, и давало худшие результаты локального уточнения, что увеличивало число итераций всего инкрементного алгоритма (как описано выше в параграфе про локальное уточнение). Неориентированная диффузия также не так хороша, как ориентированная диффузия. Поэтому был выбран вариант с ограничением расширения доменов рамками одного процессора.

- Геометрическая предекомпозиция вершин по процессорам делает необходимым наличие геометрической информации о сетке. Для устранения этого ограничения прераспределение вершин по процессорам можно получить огрублением графа. Но на данный момент используется геометрическая предекомпозиция, поскольку рассматриваемые сетки из физических расчетов уже обладают геометрическими данными.

- Критерии улучшения качества доменов и моменты формирования графа и макро-графа подгруппы доменов были установлены путем запуска параллельного инкрементного алгоритма на различных сетках, оценкой получаемых разбиений и их сравнением с результатами других алгоритмов декомпозиции.

- Повторная инициализация доменов была добавлена для исследования разбиений, не найденных ранее при многократном освобождении части вершин в доменах и росте доменов. Однако она не дала существенно лучших разбиений, поэтом в настоящем варианте параллельного ин-

крементного алгоритма инициализация доменов проводится только два раза.

- Допустимые пороговые отклонения в суммарных весах доменов на процессорах были введены из-за невозможности уменьшения алгоритмом инкрементного роста разницы в весах доменов меньше некоторого порога, что было обнаружено при разбиении некоторых графов.
- Как описывалось ранее, были реализованы различные варианты локального уточнения в попытках уменьшить время выполнения алгоритма. Но все варианты давали разбиения худшего качества, что влияло на сходимость всего инкрементного алгоритма, и в итоге, был оставлен первоначальный вариант KL/FM локального уточнения.
- Во многих частях параллельного инкрементного алгоритма используется сортировка. После обнаружения радикального уменьшения времени перевода глобальных номеров вершин в локальные при ее использовании, оказалось, что одновременные проходы по отсортированным массивам сокращают время выполнения многих алгоритмов.
- Для уменьшения времени выполнения параллельного инкрементного алгоритма декомпозиции графов в него были добавлены параллельное и последовательное огрубления графа. Проверялись различные варианты: огрубление графа перед инкрементным ростом и восстановление после локального уточнения, огрубление графа перед всем локальным разбиением и восстановление после него и др. Но на огрубленных графах домены не удовлетворяли критерию связности требуемого количества оболочек, а огрубления и восстановления графа увеличивали число итераций всего алгоритма. Поэтому в дальнейшем огрубление графа было убрано из алгоритма.

Глава 3. Результаты

3.1 Комплекс программ параллельной декомпозиции сеток GRIDSPIDERPAR

На основе описанных методов в рамках данной работы был создан комплекс программ GRIDSPIDERPAR декомпозиции больших сеток (до 10^9 вершин) на большое число микродоменов. В него вошли два алгоритма: параллельный алгоритм геометрической декомпозиции сеточных данных и параллельный инкрементный алгоритм декомпозиции графов [103 - 107]. Оба алгоритма были реализованы на распределенной памяти с использованием библиотеки MPI. Подробное описание комплекса программ можно найти по ссылке URL: http://lira.imamod.ru/FondProgramm/Decomposition/

Проведено сравнение на небольшой сетке параллельного и последовательного инкрементных алгоритмов декомпозиции графов.

Проведены тесты по оценке масштабируемости разработанных алгоритмов декомпозиции.

Проведено сравнение на четырех тетраэдральных сетках разбиений на микродомены, полученных методами созданного комплекса программ GRIDSPIDERPAR, методами пакета PARMETIS, пакета ZOLTAN и пакетом PT-SCOTCH. Также проведено сравнение различных разбиений графов микродоменов на домены и разбиений сразу на домены.

Проведено сравнение эффективности параллельного счета физических задач при распределении сеток по ядрам в соответствии с различными разбиениями, полученными методами созданного комплекса программ GRIDSPIDERPAR, пакета PARMETIS, пакета ZOLTAN и пакета PT-SCOTCH.

Проведено тестирование различных разбиений графов микродоменов на домены и разбиения сразу на домены, полученных параллельным инкрементным алгоритмом, на задаче моделирования приземного взрыва, показавшее, что при достаточном количестве микродоменов в доменах разбиения графов микродоменов не уступают по качеству разбиению сразу на домены, что подтверждается малым уменьшением скорости счета рассматриваемой физической задачи.

3.2 Сравнение параллельного инкрементного алгоритма с последовательным инкрементным алгоритмом

Были получены разбиения треугольной сетки, состоящей из $7.6 \cdot 10^4$ вершин, $2.3 \cdot 10^5$ ребер, на 20 доменов на одном процессоре последовательным и параллельным инкрементными алгоритмами декомпозиции графов. Пороговое значение на наименьший номер первой несвязной оболочки равнялось 15, пороговое значение на номера освобождаемых оболочек – 17. В обоих разбиениях все домены оказались связными с номерами несвязных оболочек, большими, или равными 15. Результаты сравнения разбиений отображены в таблице 1. Под дисбалансом подразумевается максимальное отклонение от среднего арифметического числа вершин в доменах.

Таблица 1. Сравнение разбиений сетки, состоящей из $7.6 \cdot 10^4$ вершин, полученных последовательным и параллельным инкрементными алгоритмами

Алгоритмы	**Число итераций**	**Время, сек.**	**Дисбаланс числа вершин в доменах**	**Число разрезанных ребер**
Последовательный	22	407	4	3762
Параллельный	14	7	3	3873

Как видно из таблицы 1, параллельному инкрементному алгоритму при выполнении на одном процессоре потребовалось меньше итераций и намного меньшее время для нахождения разбиения. Но число разрезанных ребер в разбиении параллельным инкрементным алгоритмом больше, чем в разбиении, полученном последовательным методом. Это объясняется тем, что в параллельный инкрементный алгоритм для уменьшения времени его работы при разбиении больших сеток (порядка 10^8 вершин) на большое число доменов (порядка 10^4) были внесены изменения в условия завершения внутреннего и внешнего циклов алгоритма локального уточнения. На рассматриваемой сетке в параллельном инкрементном алгоритме в локальном уточнении проводилось около 10 итераций внешнего цикла, а в последовательной версии – 20 итераций. Локальное уточнение сокращает число разрезанных ребер, но является одним из самых затратных по времени мест инкрементного алгоритма.

3.3 Масштабируемость алгоритмов

Вычисления проводились на кластере Helios (1524.1 TFlop/s).

Проведены тесты по оценке масштабируемости (сильное масштабирование) параллельного инкрементного алгоритма декомпозиции графов и параллельного алгоритма геометрической декомпозиции сеточных данных созданного комплекса программ GRIDSPIDERPAR. Оценивался рост ускорения получения разбиений гексаэдральной сетки, содержащей $14.7 \cdot 10^6$ ячеек, на 1024 доменов с увеличением числа процессоров (до 1024 для инкрементного алгоритма и до 512 для геометрического алгоритма). Результаты представлены в таблицах 2 и 3 и на диаграмме 33.

Таблица 2. Времена счета параллельного инкрементного алгоритма на различном числе процессоров

Proc (total)/ Proc (geom)	**32**	**64**	**128**	**256**	**512**	**1024**

4	452,035	204,471	110,98	66,35	42,814	41,909
8	446,188	198,573	105,381	63,25	32,695	32,774
16	443,735	192,445	99,18	55,88	27,136	26,155
32	441,942	188,475	96,802	48,381	20,132	16,69
64		186,001	93,72	45,43	21,58	14,75
128			89,98	47,053	16,43	9,863
256				39,754	13,928	9,655
512					13,874	9,373

В первой колонке таблицы 2 отображены числа процессоров, на которых запускалась геометрическая предекомпозиция вершин по процессорам в параллельном инкрементном алгоритме. Времена в таблицах 2 и 3 в секундах.

Таблица 3. Времена счета параллельного алгоритма геометрической декомпозиции на различном числе процессоров

Processors	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**
Time (sec.)	28,4568	18,7739	11,3254	8,1366	3,917	1,1874	0,9627	0,7209

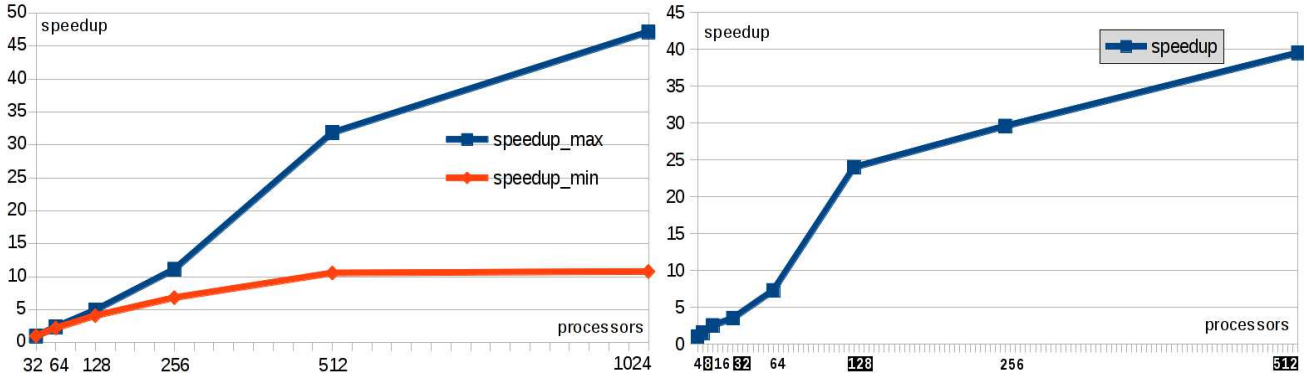


Рис. 33. Сильное масштабирование параллельного инкрементного алгоритма и параллельного алгоритма геометрической декомпозиции

В параллельном инкрементном алгоритме для каждого общего числа процессоров *N* число процессоров *Ng*, на которых запускалась геометрическая предекомпозиция вершин по процессорам, варьировалось в пределах от 4 до *N*. Для каждого *N* были найдены минимальное и максимальное времена работы алгоритма при изменении *Ng*. Ускорение speedup_max на левой диа-

грамме является отношением максимального времени, полученного на наименьшем N, к минимальному времени, полученному на текущем N. Ускорение speedup_min является отношением минимального времени, полученного на наименьшем N, к максимальному времени, полученному на текущем N.

Были получены практически линейные графики увеличения ускорений обоих алгоритмов. Большая разница между speedup_max и speedup_min на левой диаграмме объясняется изменением времени геометрической предекомпозиции при увеличении числа процессоров Ng с четырех до 512. Поэтому при больших N геометрическую предекомпозицию выгоднее выполнять на числе процессоров $Ng = N / 2$.

3.4 Разбиения на микродомены

Вычисления проводились на кластере МВС-100К (1460 модулей по два четырехядерных процессора Intel Xeon, 140.16 TFlop/s).

Четыре сгущающиеся тетраэдральные сетки, содержащие $10^8 \div 2.7 \cdot 10^8$ вершин, $7 \cdot 10^8 \div 1.6 \cdot 10^9$ тетраэдров, $8 \cdot 10^8 \div 1.9 \cdot 10^9$ ребер, были разбиты на 25600 микродоменов методами PartKway (на графиках обозначен PK) и PartGeomKway (на графиках PGK) пакета PARMETIS, методами GeomDecomp (параллельный алгоритм геометрической декомпозиции, на графиках G) и IncrDecomp (параллельный инкрементный алгоритм, на графиках I) разработанного комплекса программ параллельной декомпозиции сеток GRIDSPIDERPAR, иерархическим диффузионным алгоритмом пакета PT-SCOTCH и методами RCB, RIB и HSFC пакета ZOLTAN [106]. Методы PartKway и PartGeomKway пакета PARMETIS основаны на иерархическом алгоритме разбиения графов. В PartGeomKway выполняется предварительное геометрическое разбиение. Метод RCB пакета ZOLTAN основан на алгоритме рекурсивной координатной бисекции, метод RIB - на алгоритме рекурсивной инерциальной бисекции, HSFC выполняет геометрическое разбиение с

использованием кривой Гильберта. Метод PartGeom (аналог HSFC) пакета PARMETIS не подходит для разбиения на микродомены, так как в нем предполагается равенство числа формируемых доменов числу процессоров, на которых производится разбиение. Метод PHG (разбиение гиперграфа) пакета ZOLTAN не может бить такие большие сетки, а небольшую сетку, содержащую $2 \cdot 10^5$ вершин, разбил на 200 доменов с образование двух доменов с нулевым числом вершин. Метод REFTREE пакета ZOLTAN разбивает кривую Гильберта с использованием дерева, отображающего процесс адаптивного уточнения, которым была построена сетка. Для рассматриваемых сеток этот метод неприменим.

Первая сгущающаяся тетраэдральная сетка состоит из 209028730 вершин, 1244316672 тетраэдров и 1456626424 ребер. Данная сетка описывает пространство вокруг автомобиля и является типичной расчетной сеткой (Рис. 34).

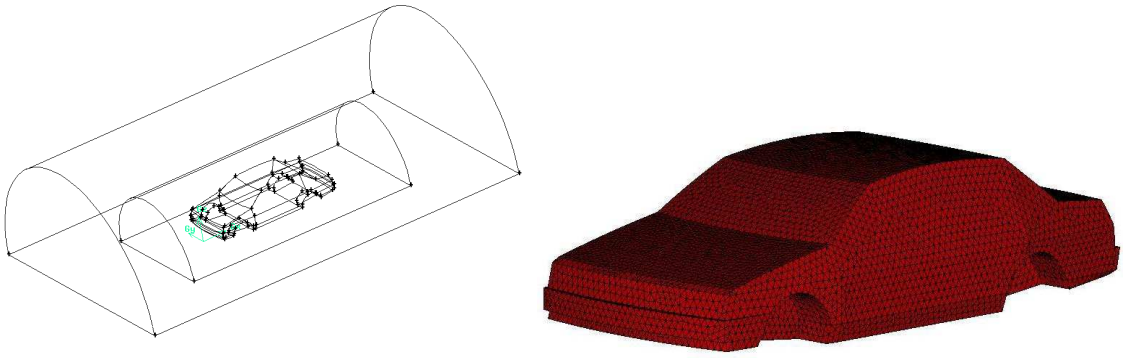


Рис. 34. Модель и часть тетраэдральной сетки, содержащей $2 \cdot 10^8$ вершин, $1.46 \cdot 10^9$ ребер

Вторая сгущающаяся тетраэдральная сетка состоит из 260517739 вершин, 1555767296 тетраэдров, 1818729840 ребер (Рис. 35, 36). Методы пакета PARMETIS не смогли побить данную сетку на 51200 микродоменов, в отличие от остальных методов.

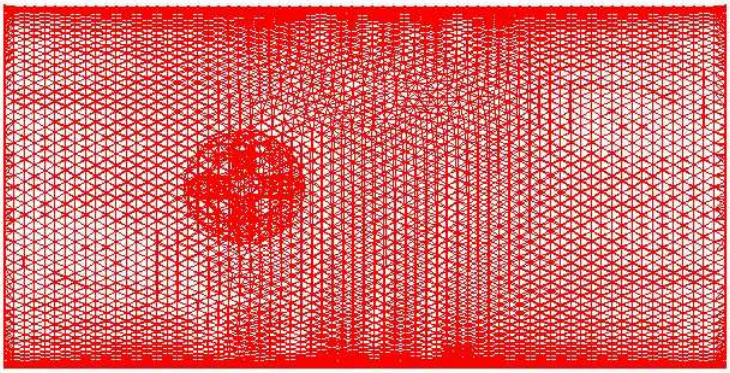


Рис. 35. Вид сбоку тетраэдральной сетки, содержащей $2.6 \cdot 10^8$ вершин, $1.8 \cdot 10^9$ ребер

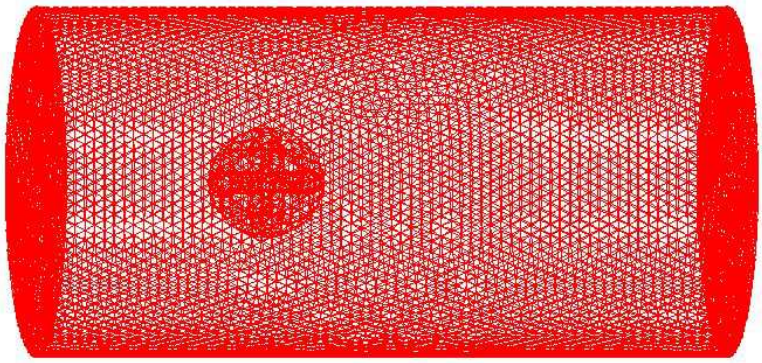


Рис. 36. Тетраэдральная сетка, содержащая $2.6 \cdot 10^8$ вершин, $1.8 \cdot 10^9$ ребер

Третья тетраэдральная сетка состоит из 110533834 вершин, 659316736 тетраэдров, 771145160 ребер (Рис. 37, 38).

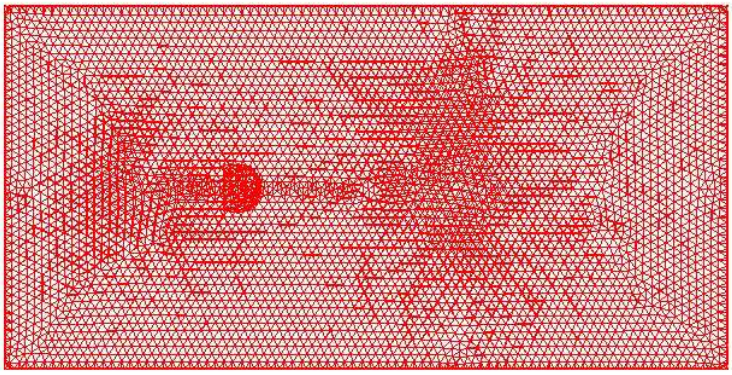


Рис. 37. Вид сбоку тетраэдральной сетки, содержащей 10^8 вершин, $7.7 \cdot 10^8$ ребер

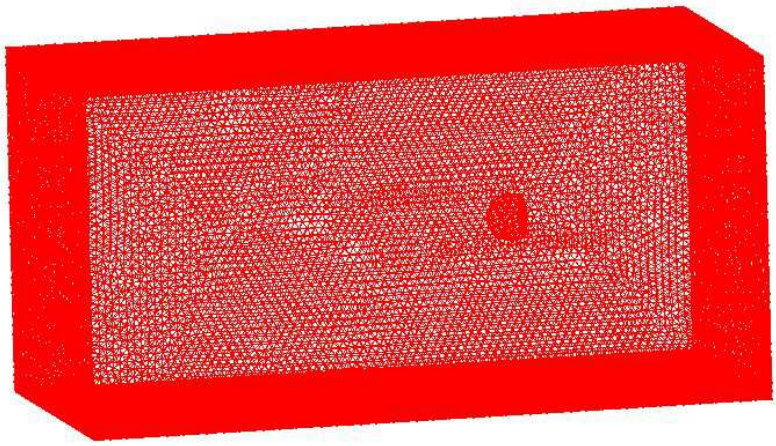


Рис. 38. Тетраэдральная сетка, содержащая 10^8 вершин, $7.7 \cdot 10^8$ ребер

Четвертая сгущающаяся тетраэдральная сетка состоит из 274823570 вершин, 1645850624 тетраэдров, 1921659536 ребер (Рис. 39). Методы пакета PARMETIS не смогли разбить данную сетку на 51200 микродоменов, в отличие от остальных методов.

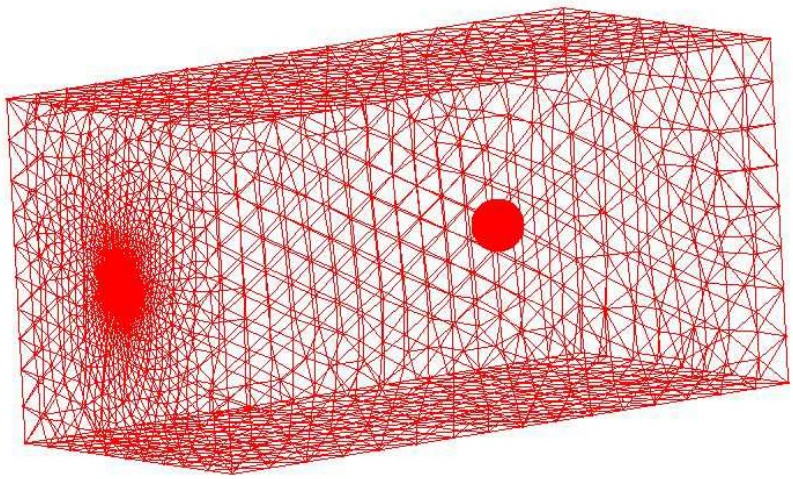


Рис. 39. Тетраэдральная сетка, содержащая $2.8 \cdot 10^8$ вершин, $1.9 \cdot 10^9$ ребер

На Рис. 40 представлено процентное отношение максимального модуля отклонения от среднего арифметического числа вершин в микродомене в разбиениях четырех тетраэдральных сеток различными методами на 25600 микродоменов. Как видно из Рис. 40, наибольшее отклонение у методов PartKway и PartGeomKway пакета PARMETIS (50 - 60 %), у PT-SCOTCH отклонение порядка 10%, у всех остальных методов - в среднем меньше 1 %.

Наименьшие отклонения с разницей в одну вершину у методов геометрической декомпозиции GeomDecomp, RCB, RIB и HSFC.

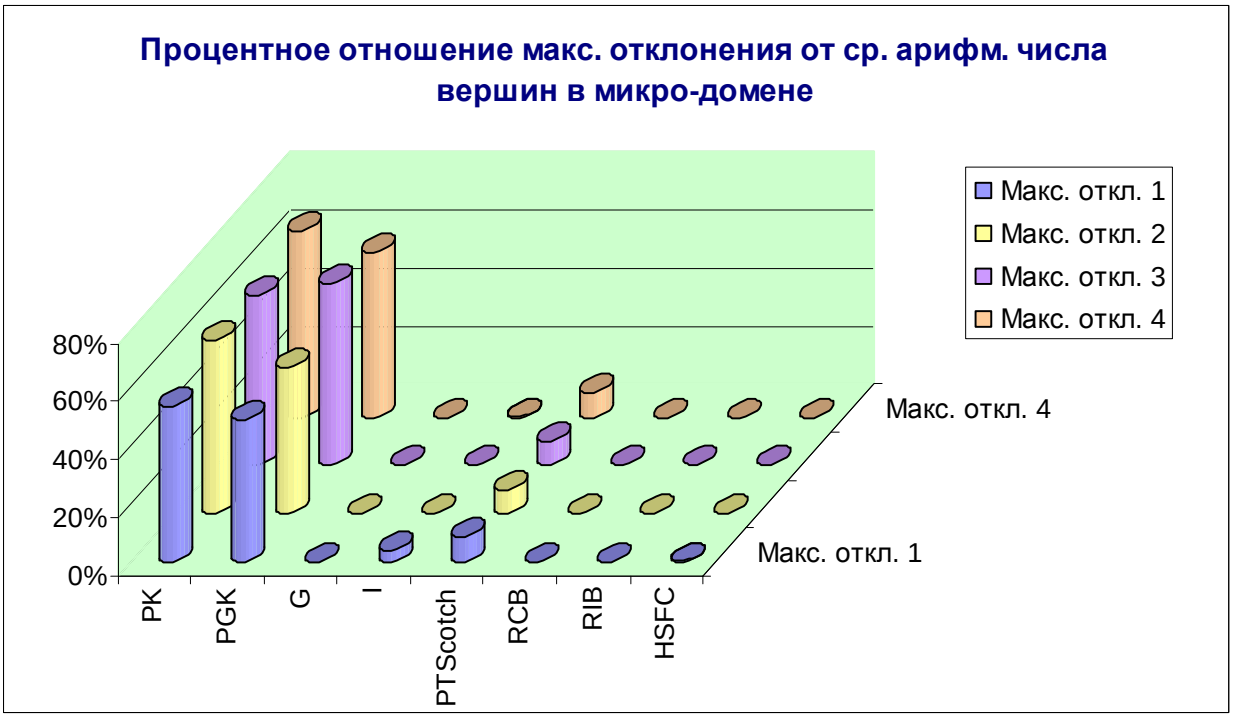


Рис. 40. Процентное отношение макс. модуля отклонения от ср. арифметического числа вершин в микродомене в разбиениях тетраэдральных сеток на 25600 микродоменов

На Рис. 41 представлено число несвязных микродоменов из 25600 в разбиениях четырех тетраэдральных сеток различными методами. На Рис. 42 представлен фрагмент данной гистограммы, не учитывающий разбиения методами RIB и HSFC пакета ZOLTAN. Для наглядности Рис. 42 порядок сеток изменен, при этом цвета, относящиеся к сеткам, сохранены. Как видно из Рис. 41 и 42, наибольшее число несвязных микродоменов в разбиениях, полученных методом HSFC пакета ZOLTAN (4 %). Методом RIB формируется 1 - 2 % несвязных микродоменов, остальными методами - меньше 1 % (порядка 30 - 40 микродоменов из 25600). Наименьшее число несвязных микродоменов в разбиениях, полученных методом IncrDecomp (не больше 1 несвязного микродомена) и PT-SCOTCH (7 несвязных микродоменов).

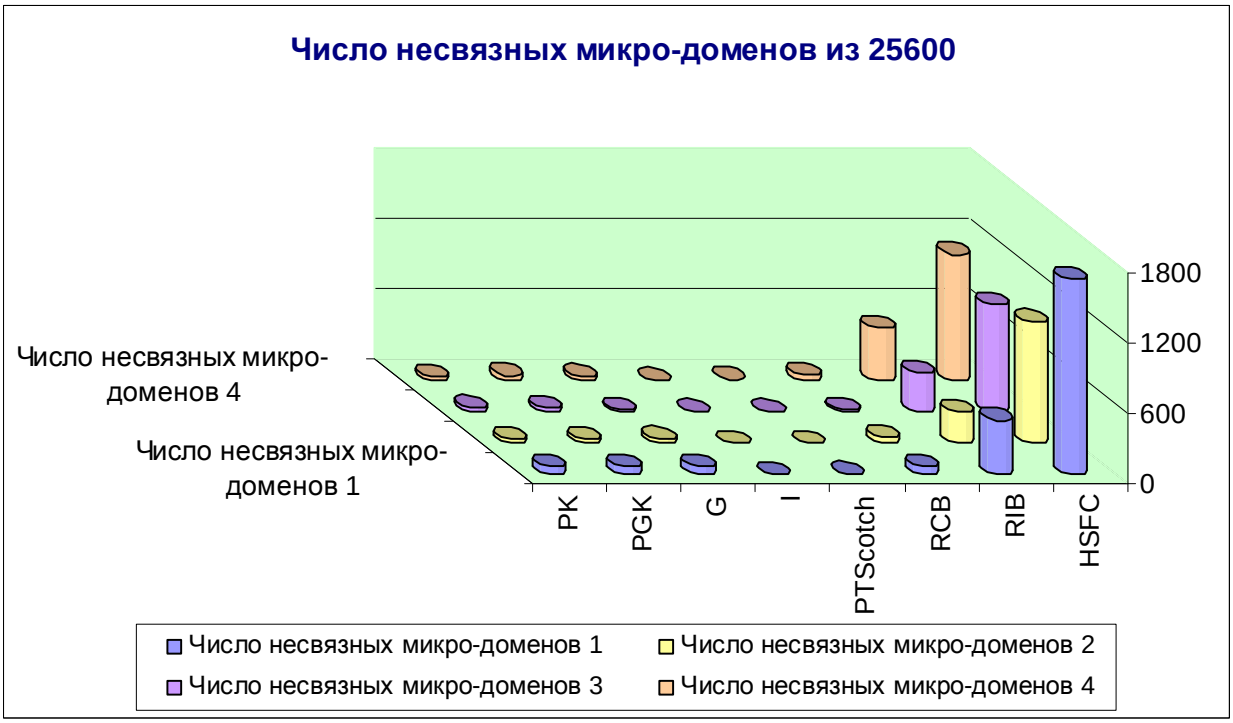


Рис. 41. Число несвязных микродоменов в разбиениях тетраэдральных сеток на 25600 микродоменов

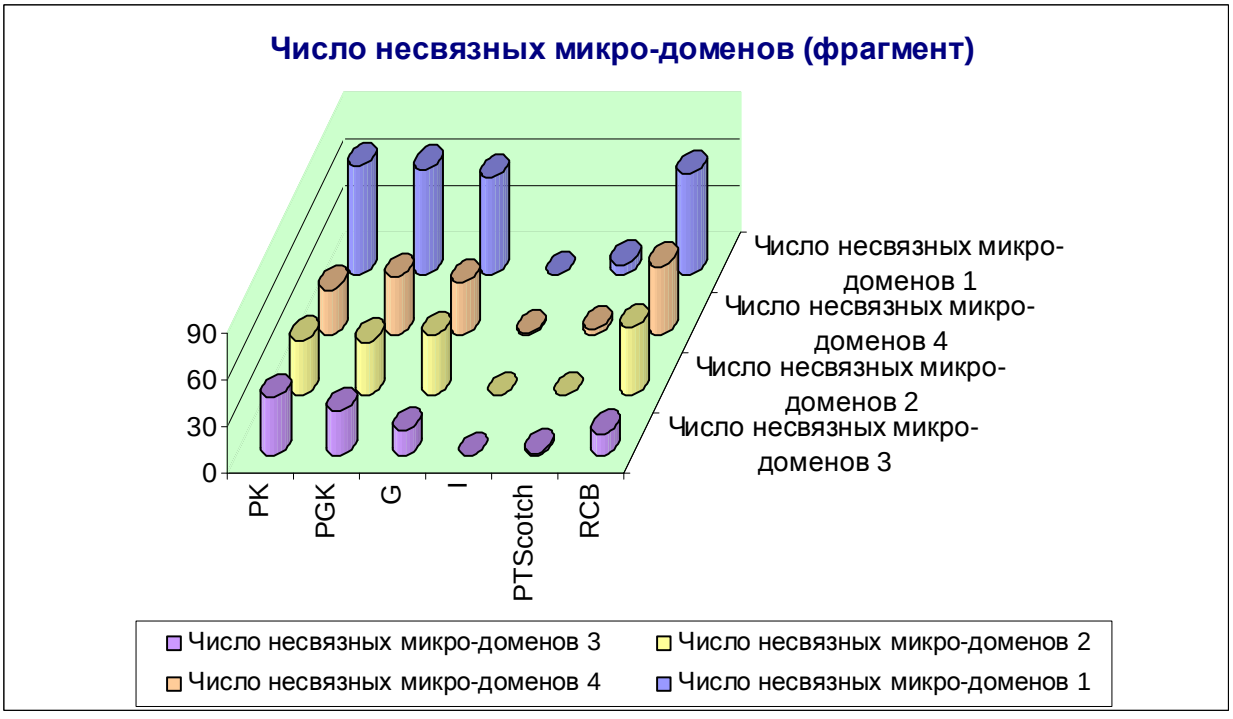


Рис. 42. Число несвязных микродоменов в разбиениях тетраэдральных сеток на 25600 микродоменов (фрагмент)

На Рис. 43 представлено число разрезанных ребер в разбиениях четырех тетраэдральных сеток различными методами. Для наглядности гистограммы

порядок сеток изменен, при этом цвета, относящиеся к сеткам, сохранены. Как видно из Рис. 43, наибольшее число разрезанных ребер у метода HSFC пакета ZOLTAN, наименьшее – у методов PartKway и PartGeomKway пакета PARMETIS. Методы геометрической декомпозиции GeomDecomp, RCB и RIB, не учитывающие связи между вершинами, дают разбиения с большим числом разрезанных ребер, чем методы разбиения графов. В разбиениях, полученных IncrDecomp и PT-SCOTCH, число разрезанных ребер немного больше, чем у методов PartKway и PartGeomKway пакета PARMETIS, PT-SCOTCH несколько выигрывает у IncrDecomp. Такое число разрезанных ребер в разбиениях, получаемых методом IncrDecomp, объясняется достаточно большой степенью локальности формирования микродоменов на процессорах.

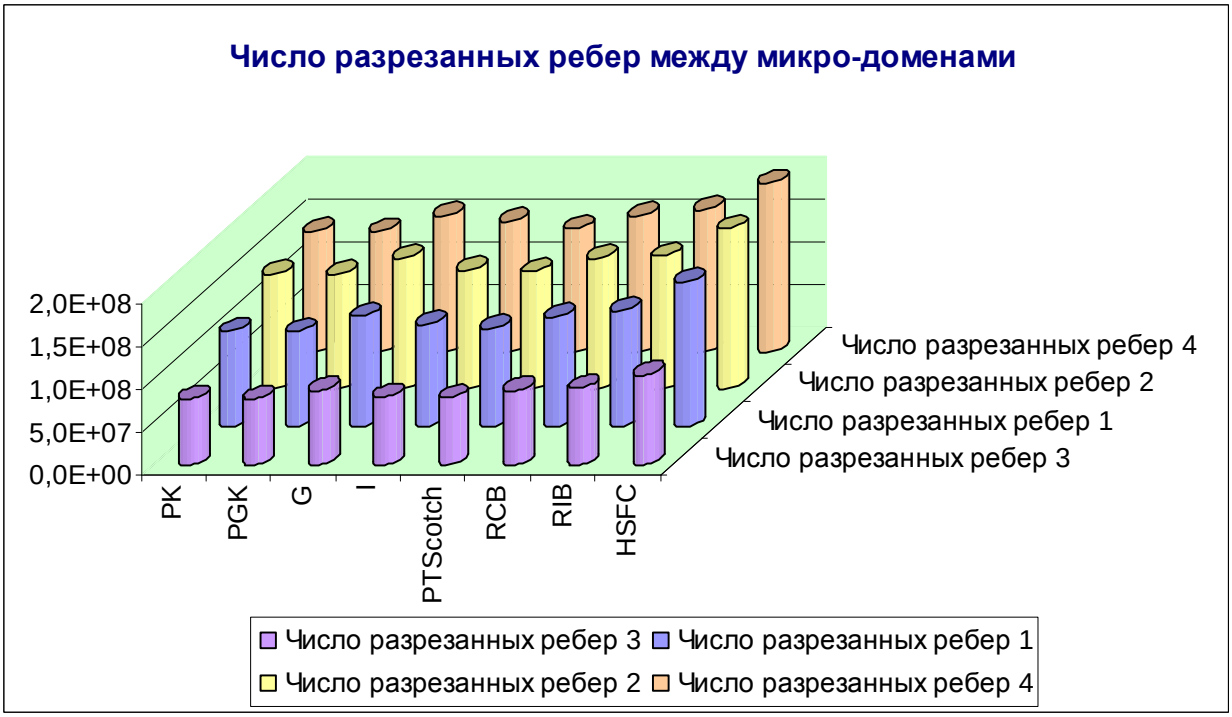


Рис. 43. Число разрезанных ребер в разбиениях тетраэдральных сеток на 25600 микродоменов

На Рис. 44 представлены числа процессоров, на которых были получены разбиения тетраэдральных сеток на 25600 микродоменов. Для наглядности гистограммы порядок сеток изменен, при этом цвета, относящиеся к сеткам,

сохранены. Числа процессоров выбирались исходя из того, чтобы хватило памяти на размещение сетки. Как видно из Рис. 44, методам геометрической декомпозиции GeomDecomp, RCB, RIB и HSFC требуется гораздо меньшее число процессоров, чем остальным методам. Это объясняется тем, что методы геометрической декомпозиции не хранят информацию о связях между вершинами.

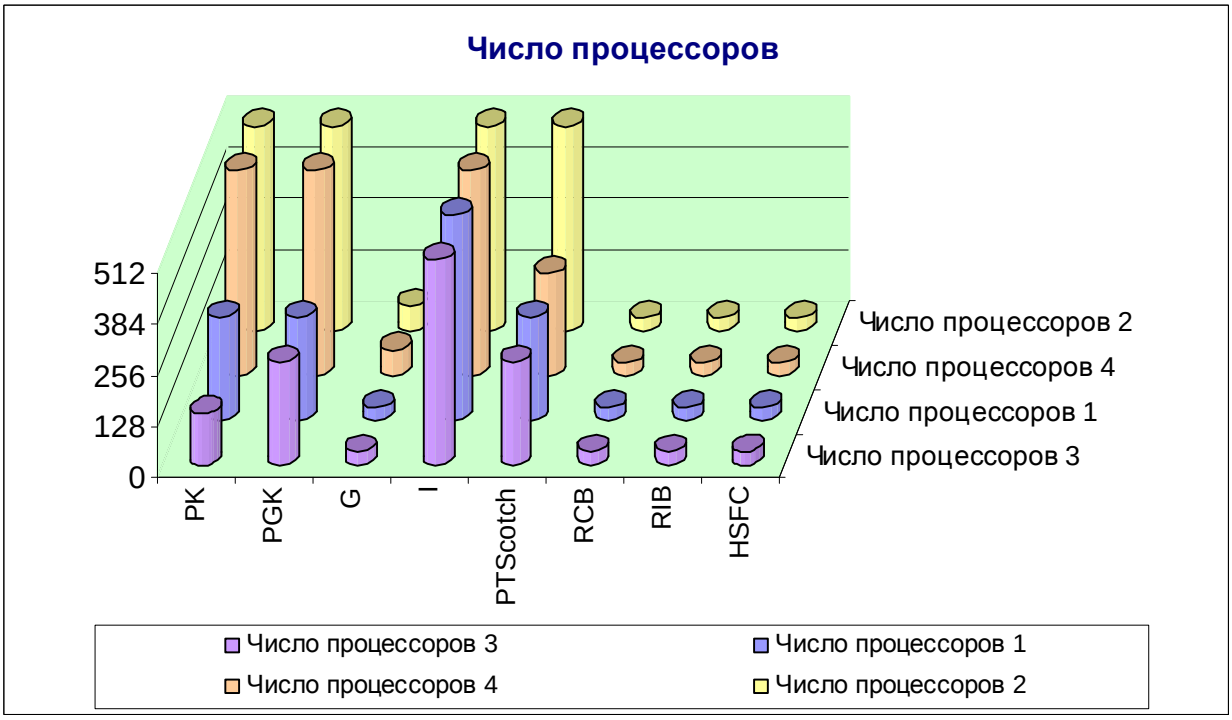


Рис. 44. Числа процессоров, на которых были получены разбиения тетраэдральных сеток на 25600 микродоменов

На Рис. 45 отображены времена получения разбиений четырех тетраэдральных сеток различными методами на 25600 микродоменов. Для наглядности гистограммы порядок сеток изменен, при этом цвета, относящиеся к сеткам, сохранены. Как видно из Рис. 45, наименьшее время получения разбиений у методов геометрической декомпозиции GeomDecomp, RCB, RIB и HSFC, наибольшее – у IncrDecomp и диффузионного алгоритма PT-SCOTCH. Время метода PartKway пакета PARMETIS сильно варьируется, что связано с зависимостью от качества начального распределения вершин по процессорам. Метод IncrDecomp в среднем сильно проигрывает по времени. Основ-

ные затраты идут на алгоритм локального уточнения. Время его выполнения *O((s-1) * m)* [10], где *s* – число доменов, *m* – число ребер в графе. Алгоритм локального уточнения плохо масштабируется по числу доменов и числу вершин [5], и обычно им бьют максимум на 8 доменов. В рассматриваемых случаях на каждом процессоре располагалось порядка 10^5 вершин, 50 микродоменов. Попытки ускорить алгоритм локального уточнения путем разбиения микродоменов на группы и поочередного уточнения микродоменов в рамках каждой группы не дали успеха, поскольку снижение качества локального уточнения приводило к увеличению числа итераций всего параллельного инкрементного алгоритма. Время одной итерации уменьшалось, но число итераций увеличивалось. Однако параллельный инкрементный алгоритм разрабатывался в качестве инструмента статической декомпозиции, которая проводится один раз перед расчетом, а времена расчета физических задач в разы больше, чем времена получения разбиений. Поэтому увеличение времени получения разбиений не играет роли в сравнении со временами расчета физических задач.

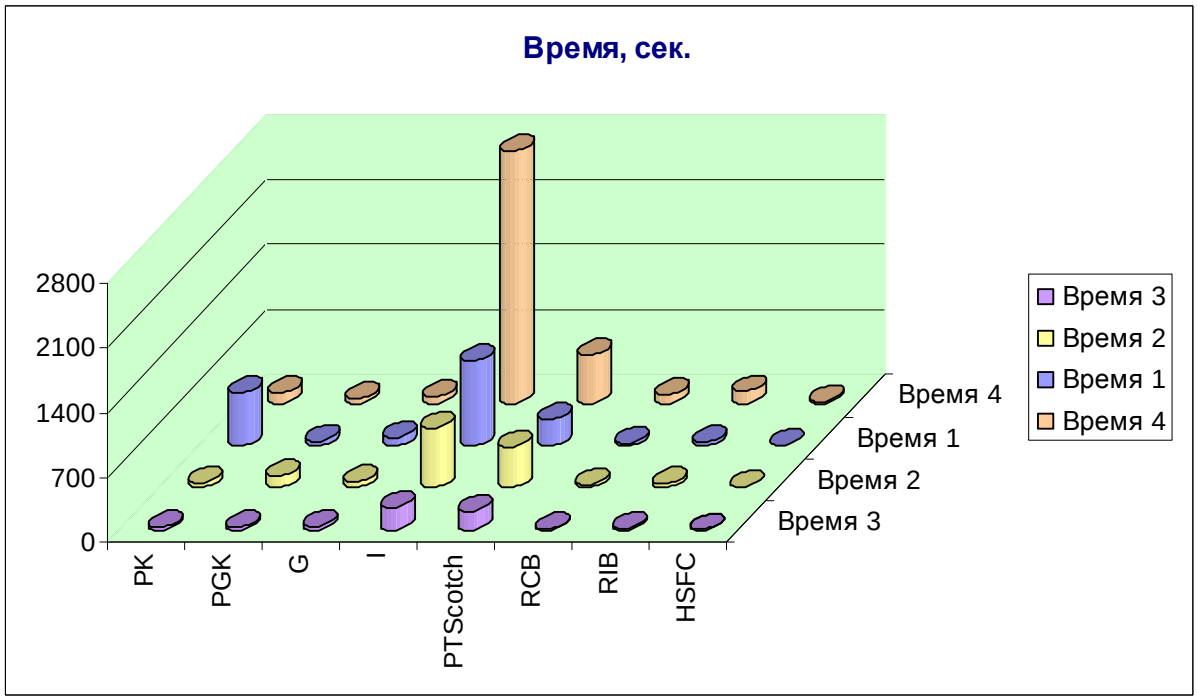


Рис. 45. Времена получения разбиений тетраэдральных сеток на 25600 микродоменов

По сумме критериев наилучшими оказались методы GeomDecomp, RCB, IncrDecomp и диффузионный алгоритм пакета PT-SCOTCH. Методы геометрического разбиения GeomDecomp и RCB за наименьшее время на небольшом числе процессоров получают разбиения с разницей числа вершин в микродоменах, не превышающей единицу. Метод GeomDecomp несколько выигрывает у RCB по числу разрезанных ребер и числу несвязных микродоменов, но проигрывает по времени. Метод IncrDecomp и PT-SCOTCH получают разбиения с меньшими числами несвязных микродоменов и разрезанных ребер. При сравнении двух методов, в разбиениях IncrDecomp меньший дисбаланс числа вершин в микродоменах и меньшее число несвязных микродоменов, чем в разбиениях PT-SCOTCH, но несколько большее число разрезанных ребер, и время получения разбиения.

3.5 Разбиения на домены

По разбиениям всех сеток на микродомены методами пакета PARMETIS и созданного комплекса программ GRIDSPIDERPAR были составлены графы связей между микродоменами с весами вершин, соответствующими количеству вершин в микродоменах. Графы связей были разбиты на 512 доменов на одном процессоре методами PartGraphRecursive (PGR) и PartGraphKway (PGrK) пакета METIS, методом PartKway (PK) пакета PARMETIS и методом IncrDecomp (I) созданного комплекса программ GRIDSPIDERPAR, запущенными на одном процессоре [106]. Проведено сравнение различных вариантов разбиений микродоменов на домены между собой и с разбиениями сразу на домены методами PartKway (PK), PartGeomKway (PGK) и PartGeom (PG) пакета PARMETIS, GeomDecomp (G) созданного комплекса программ GRIDSPIDERPAR, диффузионным алгоритмом пакета PT-SCOTCH и методами RCB, RIB и HSFC пакета ZOLTAN. Метод IncrDecomp не позволяет разбивать на такое маленькое число доменов, так как в нем предполагается,

что при выполнении разбиения на процессоре формируется порядка 10 - 50 микродоменов.

На Рис. 46 представлено процентное отношение максимального модуля отклонения от среднего арифметического числа вершин в домене для различных методов на четырех сетках. Слева направо сначала отображены результаты разбиений сразу на 512 доменов, а затем для каждого разбиения на микродомены представлены различные разбиения графов микродоменов. Разбиения графов микродоменов обозначены, как 'метод разбиения на микродомены + метод разбиения графа микродоменов'. Как видно из Рис. 46, разбиения методами PartKway и PartGeomKway пакета PARMETIS сразу на домены заметно проигрывают всем остальным разбиениям по дисбалансу числа вершин в доменах. Дисбаланс числа вершин в доменах, сформированных из микродоменов, не зависит от дисбаланса числа вершин в микродоменах. Наименьший дисбаланс числа вершин в доменах при разбиениях графов микродоменов получен методами PartGraphRecursive и PartGraphKway пакета METIS. Среди всех разбиений наименьший дисбаланс числа вершин в доменах, равный единице, получен методами PartGeom пакета PARMETIS, GeomDecomp созданного комплекса программ и методами RCB, RIB и HSFC пакета ZOLTAN.

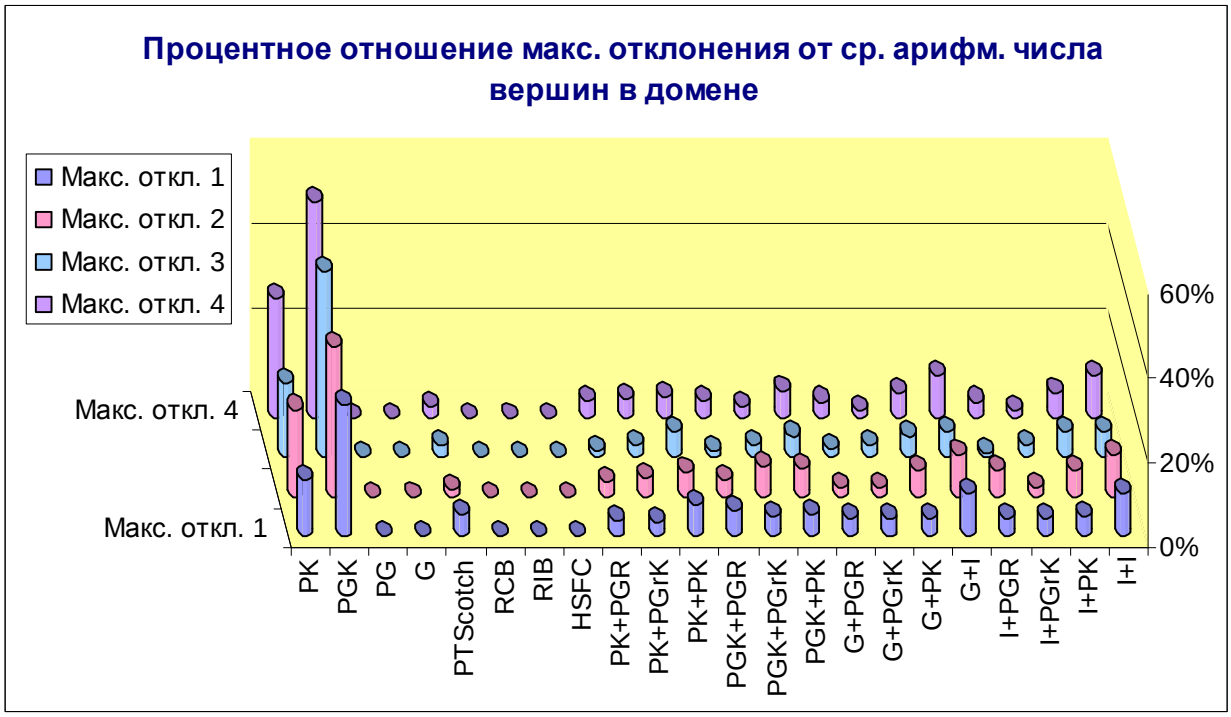


Рис. 46. Процентное отношение максимального модуля отклонения от среднего арифметического числа вершин в домене в разбиениях тетраэдральных сеток на 512 доменов

На Рис. 47 представлено число несвязных доменов в различных разбиениях тетраэдральных сеток на 512 доменов. Как видно из гистограммы, метод PartGeom пакета PARMETIS формирует практически все домены несвязными. На Рис. 48 представлен фрагмент данной гистограммы. Для наглядности порядок сеток изменен, при этом цвета, относящиеся к сеткам, сохранены. Как видно из Рис. 47 и 48, наибольшее число несвязных доменов (не считая, метод PartGeom пакета PARMETIS) сформировано методами RIB и HSFC пакета ZOLTAN, также при разбиениях графа микродоменов, полученного методом геометрической декомпозиции GeomDecomp. Остальные разбиения содержат примерно одинаковое число несвязных доменов. Среди разбиений сразу на домены наименьшее число несвязных доменов в разбиениях, полученных пакетом PT-SCOTCH.

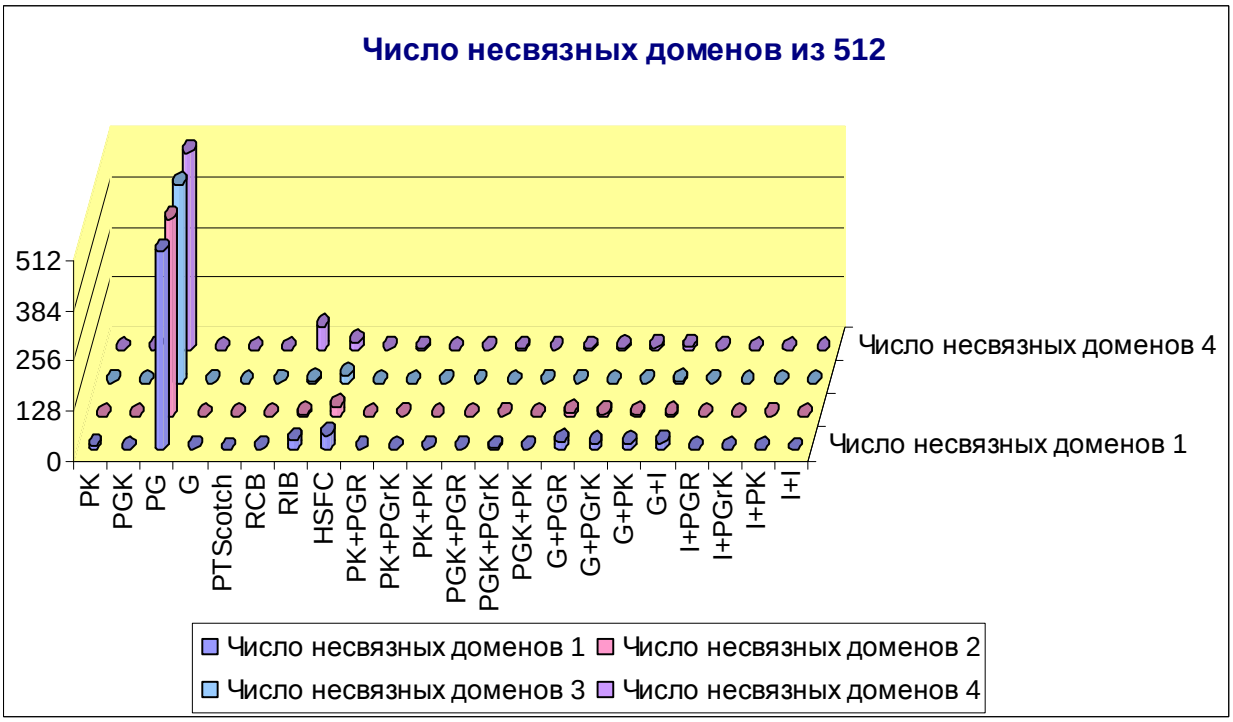


Рис. 47. Число несвязных доменов в разбиениях тетраэдральных сеток на 512 доменов

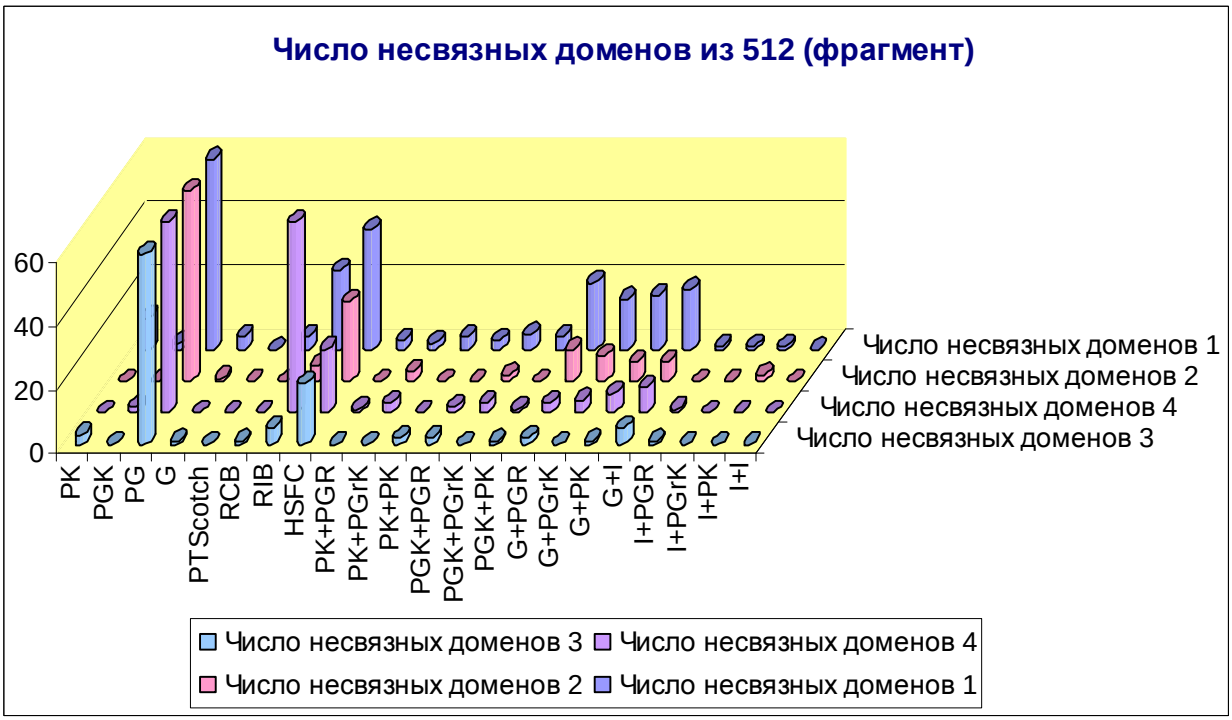


Рис. 48. Число несвязных доменов в разбиениях тетраэдральных сеток на 512 доменов (фрагмент)

На Рис. 49 отображены числа разрезанных ребер в разбиениях тетраэдральных сеток на 512 доменов. Как видно из гистограммы, методы PartGeom

пакета PARMETIS и HSFC пакета ZOLTAN также сильно проигрывают по числу разрезанных ребер, что неудивительно при таком большом числе несвязных доменов. Остальные разбиения сразу на домены имеют меньшее число разрезанных ребер, чем разбиения графов микродоменов. Среди разбиений графов микродоменов наименьшее число разрезанных ребер получено методами PartGraphKway пакета METIS и PartKway пакета PARMETIS. Среди всех разбиений наименьшее число разрезанных ребер в разбиениях, полученных методами PartKway и PartGeomKway пакета PARMETIS и пакетом PT-SCOTCH.

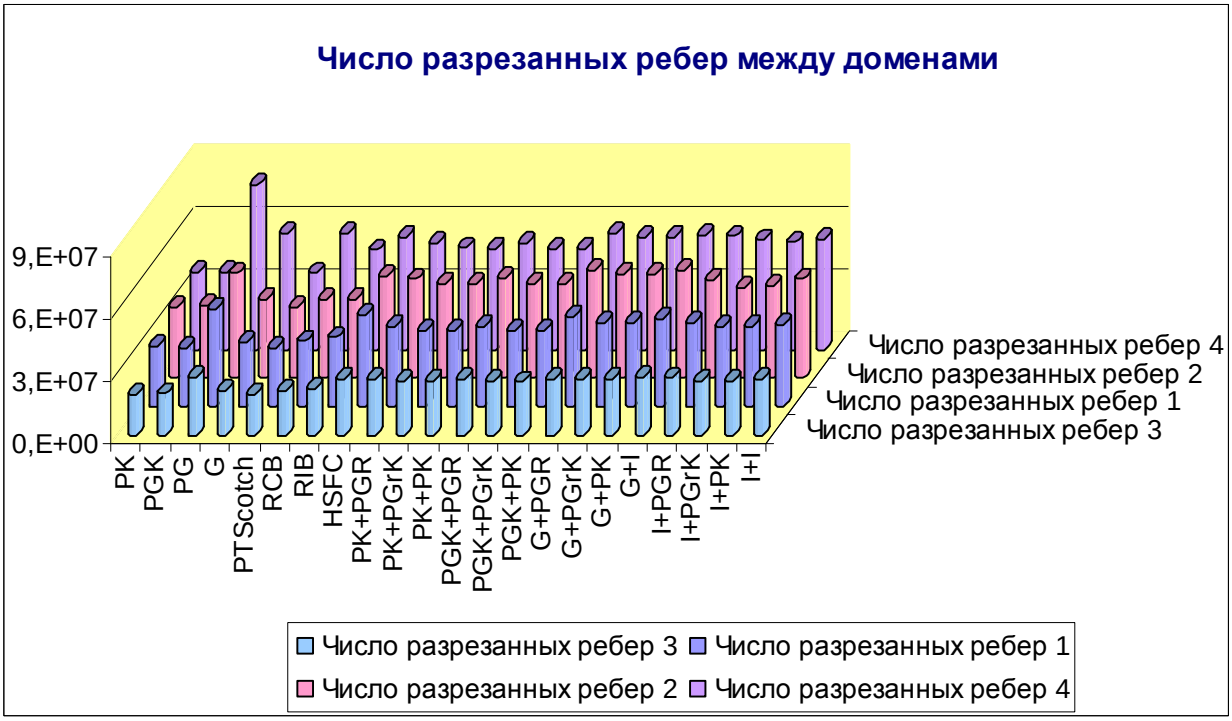


Рис. 49. Число разрезанных ребер в разбиениях тетраэдральных сеток на 512 доменов

Основные результаты по разбиениям на домены следующие. Вне зависимости от того, насколько различным был дисбаланс числа вершин в микродоменах, дисбаланс числа вершин в образуемых из микродоменов доменах оказался на одном уровне. Видимо, это связано с недостаточной чувствительностью алгоритмов разбиения графов к весам вершин. Наилучшие разбиения графов микродоменов получены методом PartGraphKway пакета

METIS. Среди всех разбиений наиболее качественными оказались разбиения, полученные методами GeomDecomp созданного комплекса программ GRIDSPIDERPAR, RCB пакета ZOLTAN и пакетом PT-SCOTCH. Методы GeomDecomp и RCB формируют домены с меньшим дисбалансом числа вершин в доменах. В разбиениях, полученных пакетом PT-SCOTCH, дисбаланс числа вершин в доменах сравним с разбиениями графов микродоменов. Однако пакетом PT-SCOTCH получаются разбиения с меньшим числом разрезанных ребер и несколько меньшим числом несвязных доменов.

3.6 Результаты тестирования разбиений на физических задачах

Вычисления проводились на кластерах МВС-100К (227,94 TFlop/s), "Ломоносов" (1700 TFlops) и Helios (1524.1 TFlop/s).

На описанных выше физических задачах проведено тестирование разбиений, полученных методами созданного комплекса программ GRIDSPIDERPAR, пакета PARMETIS, пакета ZOLTAN и пакета PT-SCOTCH [107]. Сравнивалась эффективность параллельного счета физических задач пакетом MARPLE3D при распределении сеток по ядрам в соответствии с различными разбиениями.

Для всех расчетных сеток были построены дуальные графы, учитывающие связи между ячейками через ребро. Число вершин в графах $2.8 \cdot 10^6 \div 1.2 \cdot 10^8$, число ребер $2.3 \cdot 10^7 \div 1.0 \cdot 10^9$.

Разбиения графов на домены были получены следующими методами: PartKway (на графиках обозначен PK) и PartGeomKway (на графиках PGK) пакета PARMETIS, методами GeomDecomp (параллельный алгоритм геометрической декомпозиции, на графиках G) и IncrDecomp (параллельный инкрементный алгоритм, на графиках I) разработанного комплекса программ параллельной декомпозиции сеток GRIDSPIDERPAR, иерархическим диффу-

зионным алгоритмом пакета PT-SCOTCH и методами RCB, RIB, HSFC и PHG пакета ZOLTAN. Методы PartKway и PartGeomKway пакета PARMETIS основаны на иерархическом алгоритме разбиения графов. В PartGeomKway выполняется предварительное геометрическое разбиение. Метод RCB пакета ZOLTAN основан на алгоритме рекурсивной координатной бисекции, метод RIB - на алгоритме рекурсивной инерциальной бисекции. Метод HSFC выполняет геометрическое разбиение с использованием кривой Гильберта. Метод PHG разбивает гиперграф.

Далее в тексте приводятся краткие обозначения для физических задач и соответствующих им графов: divertor – моделирование газоплазменных потоков в диверторе токамака, tube – моделирование распространения ударной волны от взрыва химического взрывчатого вещества в протяженном сооружении (ударная труба) с нетривиальной геометрией, boom и boomL – моделирование распространения ударной волны от приземного взрыва (boomL – сетка с $1.2 \cdot 10^8$ гексаэдрами).

В таблицах 4 – 7 представлены результаты разбиений упомянутых графов различными методами. Под дисбалансом подразумевается процентное отношение максимального модуля отклонения от среднего арифметического числа вершин в домене.

Таблица 4. Результаты разбиений графа divertor на 256 доменов

Информация о сетке	**Алгоритм**	**Дисбаланс, %**	**Разрезанные ребра**	**Несвязные домены**	**Время, сек.**	**Процессоры**
имя: Divertor 2 835 592 тетраэдров,	IncrDecomp	0,07	1 228 744	0	910	4
	PartKway	24,03	1 125 075	2	15,6	8
	PartGeomKway	32,31	1 119 266	2	7,9	8
	PTScotch	5,00	1 117 984	0	40	8
	GeomDecomp	0,009	1 553 256	33	1,9	8
	RCB	0,009	1 558 580	34	1,1	8

256	RIB	0,009	1 504 862	75	1,5	8
доменов	HSFC	0,009	2 504 987	101	0,7	8

Таблица 5. Результаты разбиений графа tube на 4096 доменов

Информация о сетке	Алгоритм	Дисбаланс, %	Разрезанные ребра	Несвязные домены	Время, сек.	Процессоры
имя:	IncrDecomp	1,58	17 851 910	3	582	128
Tube	PartKway	44,82	17 054 497	3	33	128
	PartGeomKway	39,24	17 136 247	10	13	128
26 495 921	PTScotch	5,01	17 364 573	1	101	128
тетраэдров,	GeomDecomp	0,02	19 642 371	11	7	16
	RCB	0,02	22 269 536	17	9	16
4096	RIB	0,02	19 707 202	28	12	16
доменов	HSFC	0,02	65 351 208	1443	5	16

Таблица 6. Результаты разбиений графа boom на 4096 доменов

Информация о сетке	Алгоритм	Дисбаланс, %	Разрезанные ребра	Несвязные домены	Время, сек.	Процессоры
имя:	IncrDecomp	0,06	36 556 880	0	622	128
Boom	PartKway	76,08	39 062 321	8	9	128
	PartGeomKway	42,51	38 431 682	10	4,9	128
61 162 984	PTScotch	5,00	36 356 679	0	115,6	128
гексаэдров	GeomDecomp	0,007	44 283 280	0	30,3	32
	RCB	0,007	44 755 295	0	11,7	32
4096	RIB	0,007	47 490 143	0	29,9	32
доменов	HSFC	0,007	52 924 900	130	6,1	32

Таблица 7. Результаты разбиений графа boomL на 10080 доменов

Информация о сетке	Алгоритм	Дисбаланс, %	Разрезанные ребра	Несвязные домены	Время, сек.	Процессоры

имя:	IncrDecomp	0,08	78 279 853	0	779	256
BoomL	PartKway	65,51	82 096 962	6	17	256
	PartGeomKway	43,23	81 373 389	11	18	256
116 214 272	PTScotch	8,28	79 739 465	0	190	256
гексаэдров	GeomDecomp	0,009	88 852 852	0	35	64
	RCB	0,009	92 740 019	1	29	64
10 080	RIB	0,009	97 931 486	6	18	64
доменов	HSFC	0,03	106 207 852	226	7	64

Как видно из таблиц 4 – 7, наибольший дисбаланс числа вершин в доменах обнаружен в разбиениях, полученных методами PartKway и PartGeomKway пакета PARMETIS (до 80 %), наименьший (с разницей в одну вершину) – в разбиениях геометрическими методами GeomDecomp, RCB, RIB и HSFC. Пакетом PT-SCOTCH получены разбиения с отклонением от среднего арифметического порядка 5 %, методом IncrDecomp созданного комплекса программ – до 0.1 %. В разбиениях, полученных методом PHG пакета ZOLTAN, присутствуют домены с нулевым числом вершин, поэтому вычисление физических задач на данных разбиениях не проводилось, и информация по разбиениям не представлена в таблицах 4 – 7.

Во всех разбиениях, за исключением полученных методом HSFC пакета ZOLTAN, практически все домены оказались связными. Разбиения HSFC содержат до 40% несвязных доменов.

Наименьшее число разрезанных ребер получено пакетом PT-SCOTCH, методом PartKway пакета PARMETIS и методом IncrDecomp комплекса программ GRIDSPIDERPAR, с выигрышем то одного, то другого метода. Методы геометрической декомпозиции GeomDecomp, RCB, RIB и HSFC, не учитывающие связи между вершинами, дают разбиения с большим числом разрезанных ребер, чем методы разбиения графов. При этом методом GeomDe-

comp созданного комплекса программ получены разбиения с меньшим числом разрезанных ребер, чем аналогичным методом RCB пакета ZOLTAN. Наибольшее и сильно выделяющееся число разрезанных ребер получено методом HSFC пакета ZOLTAN.

В соответствии с разбиениями дуальных графов были получены разбиения вершин сеток. Для расчета каждой физической задачи на всех разбиениях выделялось одинаковое машинное время. Были получены числа шагов по времени и модельные времена, до которых досчитали задачи. Распределения разбиений по числу шагов по времени и модельным временам оказались аналогичными, поэтому ниже приводятся только распределения по числу шагов по времени. Распределения представлены на Рис. 50 – 53.

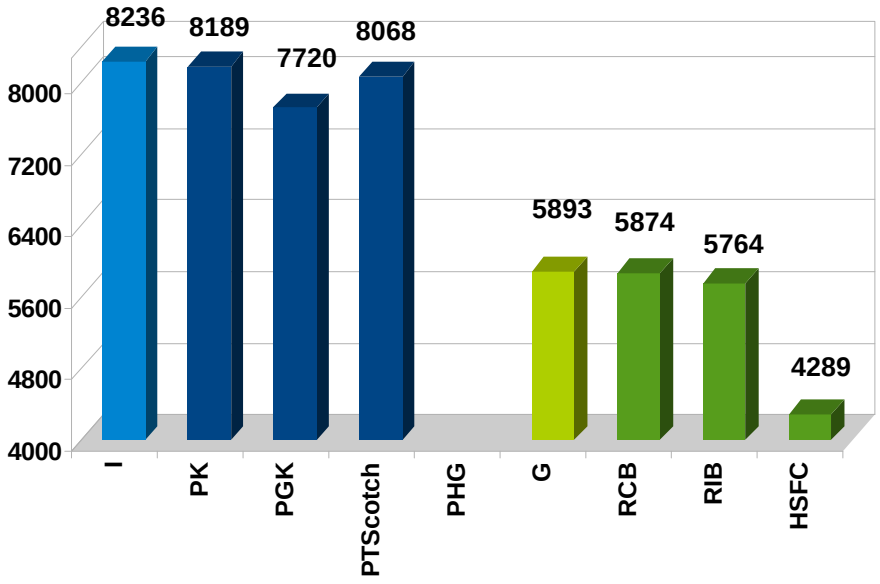


Рис. 50. Число шагов по времени для divertor

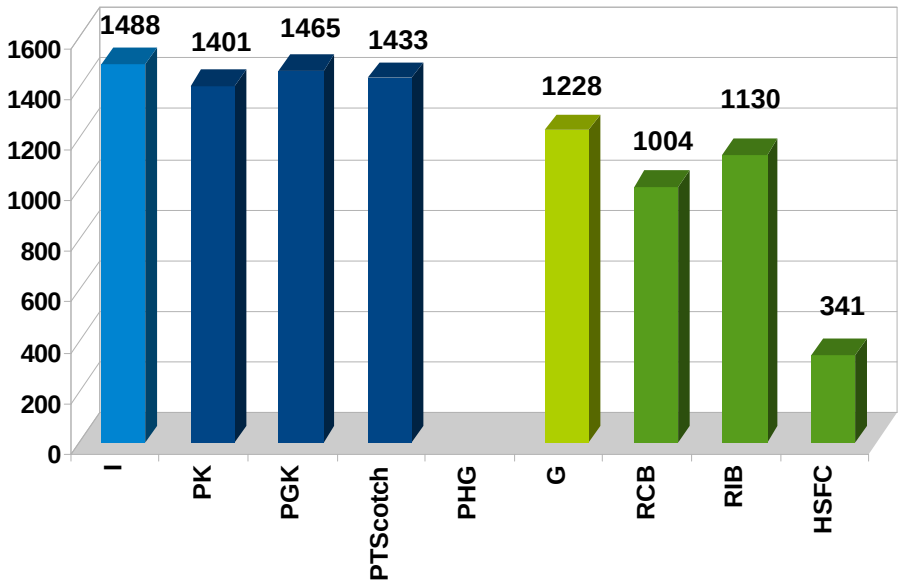


Рис. 51. Число шагов по времени для tube

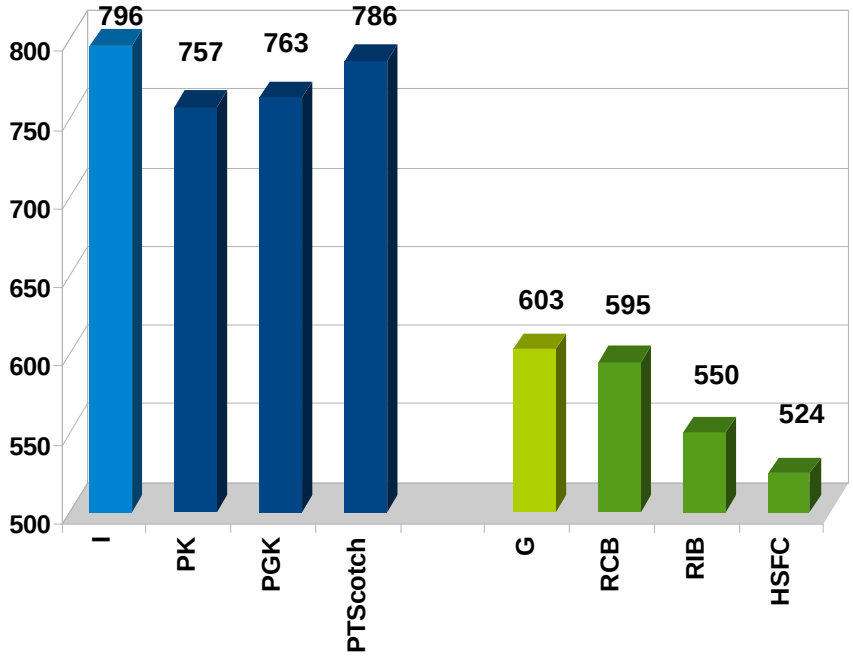


Рис. 52. Число шагов по времени для boom

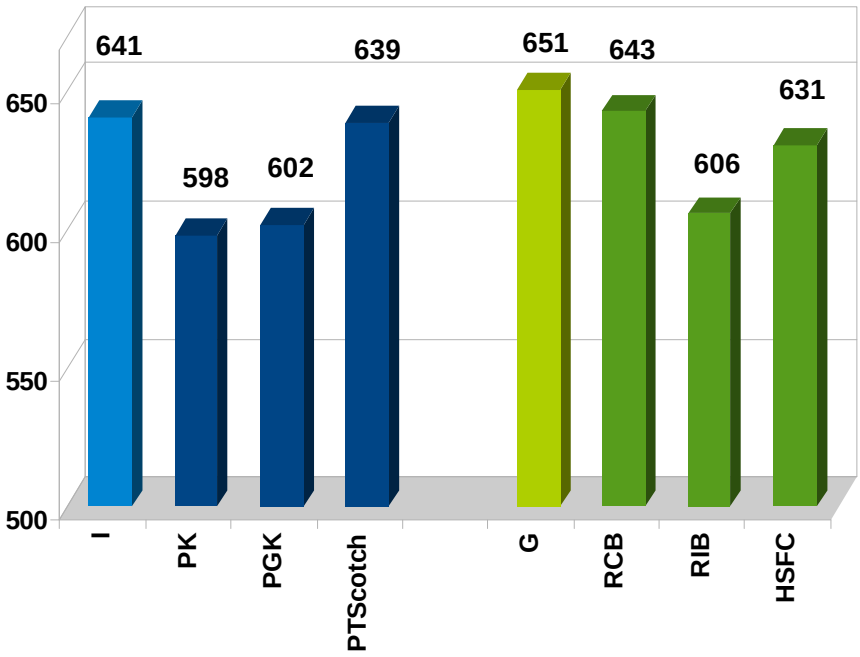


Рис. 53. Число шагов по времени для boomL

При сравнении получены следующие результаты. По числу шагов по времени метод IncrDecomp пакета GRIDSPIDERPAR опережает остальные методы декомпозиции графов, а метод GeomDecomp пакета GRIDSPIDERPAR опережает похожий метод RCB пакета ZOLTAN, а также остальные геометрические методы. На разбиениях, полученных методами декомпозиции графов, числа шагов по времени больше, чем на разбиениях, полученных геометрическими методами, не учитывающими связи между вершинами. Исключение составляют разбиения сетки boomL, для которых числа шагов по времени, полученные на геометрических разбиениях, по величине не уступают числам шагов по времени, полученным на разбиениях методами декомпозиции графов. Расчет физической задачи моделирования распространения ударной волны от приземного взрыва на сетке boomL велся с другим параметром кластера, отвечающим за коммуникационный обмен. С этим же параметром кластера проводились дополнительные расчеты задачи моделирования газоплазменных потоков в диверторе токамака на сетке divertor, и были получены похожие результаты. Что позволяет сделать вывод, что на результаты повлиял шаблон передачи сообщений на кластере. Провести

расчет физической задачи на сетке boomL с другим параметром кластера, с которыми проводились расчеты остальных задач, к сожалению, не удалось, так как расчет физической задачи на такой большой сетке с тем параметром кластера не идет.

3.7 Результаты тестирования разбиений на микродомены на физических задачах

Вычисления проводились на кластерах МВС-100К (227,94 TFlop/s) и "Ломоносов" (1700 TFlops).

На задаче моделирования распространения ударной волны от приземного взрыва проведено тестирование различных разбиений графов микродоменов и разбиения сразу на домены, полученных параллельным инкрементным алгоритмом (IncrDecomp) созданного комплекса программ GRIDSPIDERPAR. Сравнивалась эффективность параллельного счета рассматриваемой физической задачи пакетом MARPLE3D при распределении сетки по ядрам в соответствии с различными разбиениями.

Дуальный граф boomL, содержащий $1.2 \cdot 10^8$ вершин и $1.0 \cdot 10^9$ ребер, был разбит на различное число микродоменов (от 24576 до 196608) и сразу на 3072 домена алгоритмом IncrDecomp. Составлены графы связей микродоменов с весами вершин, соответствующими количеству вершин в микродоменах. Графы микродоменов были разбиты алгоритмом IncrDecomp на 3072 домена.

В соответствии с разбиениями дуального графа boomL были получены разбиения вершин сеток. Для расчета физической задачи на всех разбиениях выделялось одинаковое машинное время (5 часов). Были получены числа шагов по времени, до которых досчитала задача.

В таблице 8 представлены результаты тестирования разбиений. Под дисбалансом подразумевается процентное отношение максимального модуля отклонения от среднего арифметического числа вершин в домене.

Таблица 8. Результаты тестирования разбиений графа boomL на 3072 домена, полученных алгоритмом IncrDecomp

Информация о сетке	Микродомены	Микродомены в домене	Дисбаланс, %	Разрезанные ребра	Соседние домены (макс.)	Несвязные домены	Шаги по времени
имя:	3072	1	9,1	53 140 207	28	0	1107
BoomL	24576	8	62,5	64 611 859	25	0	833
	49152	16	37,5	66 566 874	25	0	880
116 214 272	98304	32	18,7	68 841 339	23	0	949
гексаэдров	196608	64	7,9	68 207 798	21	0	999

Как видно из таблицы 8, чем больше микродоменов, тем меньше дисбаланс получаемых разбиений, тем меньше максимальное число соседних доменов, но тем больше общее число разрезанных ребер. Увеличение общего числа разрезанных ребер объясняется тем, что при составлении графов микродоменов не учитывались веса ребер между доменами. Максимальное число соседних доменов влияет на количество обменов между процессорами, обрабатывающими данные домены. С увеличением числа микродоменов, увеличивается также число шагов по времени, полученных на разбиениях. Что говорит о том, что для задачи моделирования распространения ударной волны от приземного взрыва равномерность распределения вычислительной нагрузки по процессорам и количество обменов между процессорами критичнее, чем объем передаваемых данных. При сравнении разбиения сразу на 3072 домена и разбиений графов микродоменов заметно, что в разбиении, составленном из 196608 микродоменов, дисбаланс числа вершин в доменах меньше, чем в разбиении сразу на домены, и меньше максимальное число соседних доменов. Результат, объясняется тем, что при разбиении на опреде-

ленное количество доменов не всегда удается получить требуемый дисбаланс. Например, при разбиении данного графа на 4096 доменов получаемый дисбаланс составлял 0.03%, что значительно меньше 9.1%, полученных при разбиении на 3072 домена. Число шагов по времени, полученных на разбиении, составленном из 196608 микродоменов, не намного меньше, чем полученных на разбиении сразу на домены. Таким образом, можно сделать вывод, что при достаточном количестве микродоменов в доменах разбиения графов микродоменов не уступают по качеству разбиению сразу на домены, что подтверждается малым уменьшением скорости счета рассматриваемой физической задачи. К тому же на декомпозицию графа микродоменов при массовых расчетах требуется меньше процессоро-часов.

Заключение

Результаты, выносимые на защиту:

1. Разработана модель декомпозиции графов, учитывающая особенности вычислительных алгоритмов моделирования физических процессов в пространственных областях сложной геометрической формы. Разработаны параллельные алгоритмы, поддерживающие два основных этапа декомпозиции больших сеток: предварительную декомпозицию сетки по процессорам и параллельную декомпозицию высокого качества.

2. На основе разработанных алгоритмов создан комплекс программ GRIDSPIDERPAR параллельной декомпозиции больших сеток на большое число микродоменов. Проведены вычислительные эксперименты по сравнению разбиений на микродомены и домены четырех тетраэдральных сеток (порядка 10^8 вершин, 10^9 тетраэдров), полученных методами созданного комплекса программ GRIDSPIDERPAR, пакета PARMETIS, пакета ZOLTAN и пакетом PT-SCOTCH, показавшие преимущества алгоритмов декомпозиции созданного комплекса программ GRIDSPIDERPAR в качестве получаемых разбиений.

3. Поставлен ряд вычислительных экспериментов со следующими задачами газовой динамики: моделирование газоплазменных потоков в диверторе токамака, моделирование ударной волны в протяженном сооружении и моделирование приземного взрыва. На их основе проведено тестирование различных разбиений расчетных сеток ($10^6 \div 10^8$ ячеек), подтвердившее сокращение времени счета на разбиениях, полученных алгоритмами созданного комплекса программ GRIDSPIDERPAR.

Первые два результата получены автором диссертации лично, третий результат получен совместно с Дорофеевой Е.Ю. (ИПМ им. М.В.Келдыша

РАН). Автор диссертации занималась разработкой алгоритмов и программ выполнения рационального разбиения графов расчетных сеток, а также оценкой качества разбиений, Дорофеева Е.Ю. – постановкой физических задач и проведением вычислительных экспериментов. Эффективность параллельного счета физических задач оценивалась совместно.

Список литературы

1. C. Walshaw, M. Cross. Mesh Partitioning: A Multilevel Balancing and Refinement Algorithm. SIAM J. Sci. Comput., Vol. 22, № 1, 2000, pp. 63 – 80.
2. C. Walshaw, M. Cross. Parallel Mesh Partitioning on Distributed Memory Systems. Invited lecture. In: Proc. Parallel & Distributed Computing for Computational Mechanics, Weimar, Cermany, 1999.
3. Kirk Schloegel, George Karypis and Vipin Kumar. Graph Partitioning for High Performance Scientific Simulations. CRPC Parallel Computing Handbook, Army HPC Research Center, Dept. of Computer Science and Engineering, University of Minnesota, Minneapolis, 2000.
4. Bruce Hendrickson, Tamara G. Kolda. Graph partitioning models for parallel computing. Parallel Computing, 26, 2000, 1519-1534.
5. Bruce Hendrickson and Robert Leland. The Chaco User's Guide, Version 2.0. SAND95-2344, Unlimited Release, 1995.
6. Erik Boman, Karen Devine, Umit Catalyurek, Doruk Bozdag, Bruce Hendrickson, William F. Mitchell, James Teresco. Zoltan: Parallel Partitioning, Load Balancing and Data-Management Services. Developer's Guide, Version 3.3. Sandia National Laboratories, Copyright©2000-2010, http://www.cs.sandia.gov/Zoltan/dev_html/dev.html .
7. G. Karypis, V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. 1998. Vol 20, No 1, pp 359-392.
8. Якобовский М.В. Обработка сеточных данных на распределенных вычислительных системах. // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2004. Вып.2. с. 40-53.
9. Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson and Courtenay Vaughan. Zoltan Data Management Services for Parallel Dynamic Applications. Computing in Science & Engineering, 2002, 90 – 97.
10. Hendrickson B., Leland R. A Multilevel Algorithm for Partitioning Graphs // SAND93-1301, Unlimited Release, 1993.
11. G. Karypis, V. Kumar. Parallel Multilevel k-Way Partitioning Scheme for Irregular Graphs // SIAM REVIEW. 1999. Vol. 41. No 2. 278-300.

12. C. Walshaw, M. Cross. JOSTLE: parallel multilevel graph-partitioning software –an overview // School of Engineering, University of Swansea.
13. Francois Pellegrini. PT-Scotch and libScotch 5.1 User's Guide. Bacchus team, INRIA Bordeaux Sud-Ouest, ENSEIRB & LaBRI, UMR CNRS 5800, Universite Bordeaux I, Talence, France, 2010, 1- 78.
14. K. Schloegel, G. Karypis, V. Kumar. Parallel Multilevel Diffusion Algorithms for Repartitioning of Adaptive Meshes // Technical Report TR 97-014, University of Minnesota, Department of Computer Science. 1997.
15. S. Areibi, Y. Zeng. Effective memetic algorithms for VLSI design automation = genetic algorithms + local search + multi-level clustering // Evolutionary Computation. 2004. 3. 12. 327-353.
16. Cedric Chevalier, Francois Pellegrini. Improvement of the Efficiency of Genetic Algorithms for Scalable Parallel Graph Partitioning in a Multi-Level Framework // LaBRI and INRIA Futurs, Universite Bordearux I.
17. A. J. Soper, C. Walshaw, M. Cross. A Combined Evolutionary Search and Multilevel Optimization Approach to Graph-Partitioning // Journal of Global Optimization. 2004. 29. 225-241.
18. R. Diekmann, R. Preis, F. Schlimbach, C. Walshaw. Aspect Ratio for Mesh Partitioning // Euro-Par'98, LNCS 1470. 1998. 347-351.
19. C. Walshaw, M. Cross, R. Diekmann, F. Schlimbach. Multilevel Mesh Partitioning for Optimizing Domain Shape // The International Journal of High Performance Computing Applications. 1999. Vol 13. No 4. pp. 334-353.
20. C. Farhat. A simple and efficient automatic fem domain decomposer // Computers & Structures. 1988. 5. 28. 579-602.
21. Robert Preis and Ralf Diekmann. PARTY – A Software Library for Graph Partitioning. Advances in Computational Mechanics with Parallel and Distributed Processing, CIVIL-COMP PRESS, 1997, 63 – 71.
22. М. В. Якобовский. Инкрементный алгоритм декомпозиции графов. Вестник Нижегородского университета им. Н.И.Лобачевского. Серия «Математическое моделирование и оптимальное управление», Вып. 1(28). Нижний Новгород: Издательство ННГУ, 2005, с. 243-250.
23. Barry F. Smith, Petter E. Bjørstad, William Gropp. Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations // Cambridge University Press, 1996, 225 p.

24. А. И. Илюшин, А. А. Колмаков, И. С. Меньшов. Построение параллельной вычислительной модели путем композиции вычислительных объектов // Математическое моделирование. 2011. Т. 23. № 7. 97–113.
25. George Karypis, Vipin Kumar. A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering. Journal of Parallel and Distributed Computing, 48, 1998, 71-95.
26. C. Walshaw, M. Cross. Parallel optimization algorithms for multilevel mesh partitioning. Parallel Computing, 26, 2000, 1635-1660.
27. Erik Boman, Karen Devine, Umit Catalyurek, Doruk Bozdag, Bruce Hendrickson, William F. Mitchell, James Teresco. Zoltan: Parallel Partitioning, Load Balancing and Data-Management Services. User's Guide, Version 3.3. Sandia National Laboratories, Copyright © 2000-2010, http://www.cs.sandia.gov/Zoltan/ug_html/ug.html .
28. Francois Pelegrini. A parallelizable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries // ENSEIRB, LaBRI and INRIA Futurs, Universite Bordeaux I.
29. А. А. Воропинов. Декомпозиция данных для распараллеливания методики ТИМ-2D и критерии оценки ее качества // Вестник ЮУрГУ. Серия «Математическое моделирование и программирование:», вып. 4. 2009. №37(170). 40-50.
30. Navaratnasothie Selvakkumaran and George Karypis. Multi-objective Hypergraph Partitioning Algorithms for Cut and Maximum Subdomain Degree Minimization // IEEE Transactions on computer aided design, Vol xx, No. xx, 2005.
31. Bruce Hendrickson. Graph Partitioning and Parallel Solvers: Has the Emperor No Clothes? Sandia National Labs, Albuquerque.
32. Vahala G., Vahala L., Morrison J., Krasheninnikov S., Sigmar D. - *K*-ε compressible 3D neutral fluid turbulence modeling of the effect of toroidal cavities on flame-front propagation in the gas-blanket regime for tokamak divertors // J. Plasma Physics, 1997, vol. 57, part 1, pp. 155-173.
33. ITER - the way to new energy. - URL: http://www.iter.org/.
34. Зельдович Я.Б., Райзер Ю.П. Физика ударных волн и высокотемпературных гидродинамических явлений, М.: Наука, 1966.

35. Сэффмен Ф. Динамика завихренности // В кн.: Современная гидродинамика: успехи и проблемы. Под. ред. Дж. Бэтчелора, Г. Моффата. – М.: Мир, 1984. С. 77-90.
36. М.Н.Оцисик. Сложный теплообмен // М. Мир, 1976, стр. 347-348, 352-355.
37. Гасилов В.А. и др. Пакет прикладных программ MARPLE3D для моделирования на высокопроизводительных ЭВМ импульсной магнитоускоренной плазмы // Матем. моделирование, 2012, Т.24, №1, С.55–87.
38. B. H. V. Topping. Mesh Partitioning for Parallel and Distributed Computations // Lecture in the course High Performance Computations for Engineering, University of Pecs, Hungary. 2009.
39. Bruce Hendrickson and Tamara G. Kolda. Partitioning Sparse Rectangular Matrices for Parallel Computations of Ax and A^T v. Applied Parallel Computing in Large Scale Scientific and Industrial Problems, PARA'98, number 1541 in Lecture Notes in Computer Science, Springer, Berlin, 1998, pp. 239 - 247.
40. U. V. Catalyurek, C. Aykanat. Decomposing irregularly sparse matrices for parallel matrix-vector multiplication. Parallel Algorithms for Irregularly Structured Problems, Irregular'96, number 1117 in Lecture Notes in Computer Science, Springer, Berlin, 1996, pp. 75-86.
41. U. V. Catalyurek, C. Aykanat. Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans. Parallel Distrib. Syst. 10 (1999) 673-693.
42. A. Pinar, U. V. Catalyurek, C. Aykanat, M. Pinar. Decomposing linear programs for parallel solution. Applied Parallel Computing in Computations in Physics, Chemistry and Engineering Science, PARA'95, number 1041 in Lecture Notes in Computer Science, Springer, Berlin, 1996, pp. 473-482.
43. George Karypis and Vipin Kumar. Multilevel Algorithms for Multi-Constraint Graph Partitioning. Technical Report # 98-019, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1998.
44. Kirk Schloegel, George Karypis and Vipin Kumar. Parallel static and dynamic multi-constraint graph partitioning. Concurrency and Computation: Practice and Experience, 2002, 14, 219-240.

45. K. Schloegel, G. Karypis, V. Kumar. A new algorithm for multi-objective graph partitioning. Proc. EuroPar'99, Lecture Notes in Computer Science, Springer, Berlin, 1999.
46. B. Hendrickson, R. Leland, R. V. Driessche. Skewed graph partitioning. Proceedings of the Eighth SIAM Conference Parallel Processing for Scientific Computing, SIAM, 1997.
47. S.-H. Teng. Points, spheres and separators: a unified geometric approach to graph partitioning. Ph.D. Thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.
48. Robert Preis. The PARTY Graphpartitioning – Library. User Manual – Version 1.99 // Universitat Paderborn, Germany, October 13, 1998, 1 – 8.
49. Horst D. Simon. Partitioning of Unstructured Problems for Parallel Processing. Numerical Aerodynamic Simulation (NAS) Systems Division, NASA Ames Research Center, 1994.
50. Roy D. Williams. Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations. Concurrent Supercomputing Facility, California Institute of Technology, 1990.
51. Erik Boman, Karen Devine. Tutorial: Partitioning, Load Balancing and the Zoltan Toolkit // SciDAC Tutorial, MIT, CSCAPES Institute, Sandia National Laboratories, 2007.
52. Cedric Chevalier, Erik Boman, Karen Devine, Vitus Leung, Lee Ann Riesen. Tutorial: The Zoltan Toolkit // ACTS Workshop presentation, August 2009.
53. Ho-Kwok Dai and Hung-Chi Su. Approximation and Analytical Studies of Inter-clustering Performances of Space-Filling Curves. Discrete Mathematics and Theoretical Computer Science AC, 2003, 53-68.
54. William Gilbert. A Cube-filling Hilbert Curve // Mathematical Intelligencer. 1984. 6(3). 78.
55. Островский С.Л., Гольдшлаг О.Я. Фрактальные кривые // Информатика, 1995, № 23.
56. B.B. Mandelbrot. The fractal Geometry of Nature // Freeman, San Francisco. 1982.
57. George Karypis. METIS – A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of

Sparse Matrices. Version 5.0 // University of Minnesota, Minneapolis, August 4, 2011.

58. George Karypis, Kirk Schloegel and Vipin Kumar. ParMETIS – Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 3.1 // University of Minnesota, Minneapolis, 2003.
59. George Karypis, Kirk Schloegel. ParMETIS – Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 4.0 // University of Minnesota, Minneapolis, August 4, 2011.
60. Karen D. Devine, Erik G. Boman, Lee Ann Riesen, Umit V. Catalyurek, Cedrik Chevalier. Getting Started with Zoltan: a Short Tutorial // CSCAPES SciDAC Institure, 1– 10.
61. B. N. Parlett. The Symmetric Eigenvalue Problem // Prentice-Hall, Englewood Cliffs, NJ, 1980.
62. B. Hendrickson, R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations // SIAM J. Sci. Comput. 1995. 16(2). 452-469.
63. C. Farhat, H. D. Simon. TOP/DOMDEC – a Software Tool for Mesh Partitioning and Parallel Processing // Report RNR-93-011. NASA Ames Research Center. 1993.
64. H. D. Simon, A. Sohn, R. Biswas. HARP: A dynamic spectral partitioner // Journal of Parallel and Distributed Computing. 1998. 50. pp. 88-103.
65. A. Sohn, H. Simon. S-HARP: A Parallel Dynamic Spectral Partitioner // NASA JOVE Program, NASA Ames Research Center.
66. A. Sohn, H. Simon. JOVE: A Dynamic Load Balancing Framework for Adaptive Computations on an SP-2 Distributed-Memory Multiprocessor // NJIT CIS Technical Report 94-60. 1994.
67. B. Kernighan, S. Lin. An efficient heuristic procedure for partitioning graphs // Bell System Technical Journal. 1970. 29. 291-307.
68. C. M. Fiduccia, R. M. Mattheyses. A linear time heuristic for improving network partitions // Proc. 19th IEEE Design Automation Conference, IEEE. 1982. 175-181.
69. J. R. Gilbert, E. Zmijewski. A Parallel Graph Partitioning Algorithm for a Message-Passing Multiprocessor // International Journal of Parallel Programming. 1987. Vol 16. No 6. 427-449.

70. Y. F. Hu, R. J. Blake, D. R. Emerson. An optimal migration algorithm for dynamic load balancing // Concurrency: Practice and Experience. 1998. 10. 467-483.
71. G. Karypis, V. Kumar. METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System // University of Minnesota, Department of Computer Science, Minneapolis. 1995.
72. G. Karypis, V. Kumar. Multilevel k-way Partitioning Scheme for Irregular Graphs // Journal of Parallel and Distributed Computing. 1998. 48. 96-129.
73. J. Hromkovic, B. Monien. The Bisection Problem for Graphs of Degree 4 (Configuring Transputer Systems) // Extended abstract. University of Paderborn, Paderborn, West Germany.
74. A. Gupta. Fast and Effective Algorithms for Graph Partitioning and Sparse Matrix Ordering // IBM Research Report. RC 20496 (90799). Computer Science / Mathematics. 1996.
75. A. Gupta. WGPP: Watson Graph Partitioning (and sparse matrix ordering) Package. Users Manual: Version 3.2 // IBM Research Report. RC 20453 (90427). Computer Science / Mathematics. 1996.
76. Francois Pellegrini. Scotch and libScotch 5.1 User's Guide // Bacchus team, INRIA Bordeaux Sud-Ouest, ENSEIRB & LaBRI, UMR CNRS 5800, Universite Bordeaux I. 2010. 1-133.
77. Francois Pellegrini. PT-Scotch and libPTScotch 6.0 User's Guide // Bacchus team, INRIA Bordeaux Sud-Ouest, Universite Bordeaux 1 & LaBRI, UMR CNRS 5800, Talence, France, 2012, 1- 90.
78. S. Barnard. PMRSB – Parallel Multilevel Recursive Spectral Bisection // Cray Research, Inc.
79. S. T. Barnard, H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems // Report RNR-92-033. NAS Systems Division. Applied Research Branch. 1992.
80. G. Karypis, V. Kumar. A Coarse-Grain Parallel Formulation of Multilevel k-way Graph Partitioning Algorithm // Proceeding of the 8th SIAM Conference on Parallel Processing for Scientific Computing.
81. K. Schloegel, G. Karypis, V. Kumar. Parallel Multilevel Algorithms for Multi-constraint Graph Partitioning // Euro-Par 2000, LNCS 1900. 2000. 296-310.

82. M. Luby. A simple parallel algorithm for the maximal independent set problem // SIAM J. Comput. 1986. 15. 1036-1053.
83. P. Raghavan. Parallel Ordering Using Edge Contraction // Tech. Rep. CS-95-293, Department of Computer Science, University of Tennessee, Knoxville. 1995.
84. A. George. Nested dissection of a regular finite element mesh // SIAM J. Numer. Anal. 1973. 10. 345-363.
85. C. Walshaw, M. Cross, M. G. Everett. Parallel dynamic graph-partitioning for unstructured meshes // Mathematics Research Report 97/IM/20, Centre for Numerical Modelling and Process Analysis, University of Greenwich. 1997.
86. C. Walshaw, M. Cross, M. G. Everett. Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes // Journal of Parallel and Distributed Computing. 1997. 47. 102-108.
87. K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, U. V. Catalyurek. Parallel Hypergraph Partitioning for Scientific Computing // Tech. Report, Computer Science Research Institute, Sandia.
88. R. Biswas, R. C. Strawn. A new procedure for dynamic adaptation of three-dimensional unstructured grids // Applied Numerical Mathematics. 1994. 13. 437-452.
89. K. Schloegel, G. Karypis, V. Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes // Technical Report TR 97-013, University of Minnesota, Department of Computer Science. 1997. http://www.cs.umn.edu/karypis.
90. Y. F. Hu, R. J. Blake. An optimal dynamic load balancing algorithm // Technical Report DL-P-95-011, Daresbury Laboratory, Warrington, UK. 1995.
91. T. N. Bui, B. R. Moon. Genetic algorithm and graph partitioning // IEEE Trans. Comput. 1996. 7. 45. 841-855.
92. J. Horn, N. Nafpliotis, D. E. Goldberg. A niched Pareto genetic algorithm for multi-objective optimization // IEEE World Congress on Computational Intelligence. 1994. 1. 82-87.
93. P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts, towards memetic algorithms // Technical Report 826, California Institute of Technology, Pasadena, CA 91125, U.S.A. 1989.

94. D. Whitley, S. Rana, R. B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence // Journal of Computing and Information Technology. 1999. 7. 33-47.
95. C. Walshaw, M. Cross, R. Diekmann, F. Schlimbach. Multilevel Mesh Partitioning for Optimizing Subdomain Aspect Ratio // Developments in Computational Mechanics with High Performance Computing, Civil-Comp Press, Edinburgh. 1999. 9-19.
96. A. George, J. W.-H. Liu. Computer Solution of Large Sparse Positive Definite Systems // Prentice-Hall, Englewood Cliffs, NJ. 1981.
97. T. Goehring, Y. Saad. Heuristic Algorithms for Automatic Graph Partitioning // Tech. rep., Department of Computer Science, University of Minnesota, Minneapolis. 1994.
98. J. P. Ciarlet, F. Lamour. On the Validity of a Front-Oriented Approach to Partitioning Large Sparse Graphs with a Connectivity Constraint // Tech. rep. 94-37, Computer Science Department, UCLA, Los Angeles, CA. 1994.
99. R. Preis. Efficient partitioning of very large graphs with the new and powerful helpful-set heuristic // Diplomarbeit, Universitat-GH Paderborn, Germany. 1994.
100. H. Meyerhenke, S. Schamberger. Balancing parallel adaptive FEM computations by solving systems of linear equations // Proc. Europar. 2005. 209-219.
101. Кнут Д.Э. Искусство программирования, т.3. Сортировка и поиск 2-е изд.: Пер. с английского – М.: Издательский дом «Вильямс», 2001.
102. Якобовский М.В. Параллельные алгоритмы сортировки больших объемов данных. В кн.: Фундаментальные физико-математические проблемы и моделирование технико-технологических систем: Сб. науч. тр., Выпуск 7 / Под ред. Л.А. Уваровой. – М.: Издательство "Янус-К", 2004. – с. 235-249.

Публикации автора по теме диссертации

103. Головченко Е.Н. Комплекс программ параллельной декомпозиции сеток // Вычислительные методы и программирование. 2010. Т. 11. 360-365.

104. Головченко Е.Н. Параллельный пакет декомпозиции больших сеток // Математическое моделирование. 2011. Т. 23. № 10. 3-18.

105. Бухановский А. В., Марьин С. В., Князьков К. В., Сиднев А. А., Жабин С. Н., Баглий А. П., Штейнберг Р. Б., Шамакина А. В., Воеводин В. В., Головченко Е. Н., Фалалеев Р. Т., Духанов А. В., Тарасов А. А., Шамардин Л. В., Моисеенко А. И. Результаты реализации проекта «Мобильность молодых ученых» в 2010 году: развитие функциональных элементов технологии iPSE и расширение состава прикладных сервисов // Известия высших учебных заведений. Приборостроение. 2011. Т. 54. №10. 80 - 86.

106. Е.Н. Головченко. Разбиение больших сеток // Вестник Нижегородского университета им. Н.И. Лобачевского. Серия «Информационные технологии». Нижний Новгород: Издательство ННГУ. 2012. № 5(2). С. 309-315.

107. Evdokia GOLOVCHENKO, Elizaveta DOROFEEVA, Irina GASILOVA and Alexey BOLDAREV. Numerical Experiments with New Algorithms for Parallel Decomposition of Large Computational Meshes // Parallel Computing: Accelerating Computational Science and Engineering (CSE). Advances in Parallel Computing. IOS Press. 2014. Vol. 25. 441 - 450.