УДК 519.61, 519.174, 519.684.4

© *С. П. Копысов, И. Р. Кадыров, А. К. Новиков*

РЕСУРСНО-ЭФФЕКТИВНЫЕ КОНЕЧНО-ЭЛЕМЕНТНЫЕ ВЫЧИСЛЕНИЯ НА МНОГОЯДЕРНЫХ АРХИТЕКТУРАХ

В данной работе рассматривается построение эффективных конечно-элементных алгоритмов на трехмерных неструктурированных сетках, учитывающих сложные параллельные процессы синхронизации, вопросы распределения памяти и хранения данных. Предлагается послойное разделение расчетной сетки на подобласти без ветвления внутренних границ, упрощающее доступ к независимым данным и параллельным вычислениям на различных этапах конечно-элементного решения задач на неструктурированных сетках в многосвязных областях. Анализируется возможность прогнозирования временной эффективности и ресурсоемкости предложенных алгоритмических решений. Приводится анализ ресурсной эффективности алгоритмов в поэлементной схеме решения системы линейных алгебраических уравнений метода конечных элементов. Показано, что низкая арифметическая интенсивность рассматриваемых алгоритмов приводит к тому, что их производительность ограничивается пропускной способностью подсистемы памяти, а не производительностью процессоров. Графическая память обладает большей пропускной способностью, чем оперативная, что позволяет существенно увеличить производительность алгоритма на GPU.

Ключевые слова: метод конечных элементов, параллельные вычисления, разделение неструктурированной сетки, граф Риба, интенсивность арифметических операций, универсальная модель масштабируемости.

DOI: 10.20537/2226-3594-2019-53-08

Введение

Для решения современных вычислительных задач нельзя надеяться только на возрастающую производительность многоядерных процессоров и увеличения их числа в суперкомпьютерах. Повышение параллельной эффективности вычислений следует достигать за счет оптимизации обменов в системе процессор–память, в алгоритмах, обладающих высокой интенсивностью арифметических операций и локализацией доступа к используемым данным. Для вычислительных алгоритмов на неструктурированных сетках характерны операции над данными, которые превышают объемы кэш-памяти процессоров и требуют нерегулярного доступа к памяти, определяемого нумерацией сеточных данных (узлов, ребер, граней, ячеек). Один из способов повышения производительности вычислений и эффективности доступа к памяти, не требующий модификации структуры данных, заключается в задании новых разделений неструктурированной сетки, определяющих варианты порядка обращения к ним и обработки.

Разработка новых ресурсо-эффективных численных алгоритмов и их программного обеспечения должна основываться на современных методах программирования и схемах представления данных, позволяющих оптимизировать процесс вычислений, исходя из анализа их работоспособности применительно к реальной архитектуре гибридных параллельных вычислительных систем, что также связано с применением декомпозированных сеточных моделей [1]. Возникает также необходимость разработки методов исследования и сравнительного анализа ресурсной эффективности вычислительных алгоритмов.

Важнейшим и одним из ресурсоемких вычислительных этапов численного решения задач методом конечных элементов (МКЭ) на неструктурированных сетках является этап формирования глобальной матрицы системы линейных алгебраических уравнений (СЛАУ). Временные затраты на этапы формирования глобальной матрицы и решения СЛАУ во многом определяют полное время решения задачи. Операция поиска нужного расположения в структуре хранения глобальной матрицы занимает практически половину временных затрат формирования СЛАУ на больших неструктурированных трехмерных сетках.

Исследование разнообразных параллельных алгоритмов в области конечно-элементного анализа позволяет выделить следующее направление их развития: формирование глобальной матрицы

может быть получено за счет принципиально новых алгоритмических решений, повышающих параллельную эффективность за счет оптимизации операций адресации и доступа к данным не только на этапах формирования и решения систем уравнений, но на этапах построения и перестроения неструктурированных сеток.

В работах авторов [2] предложено послойное разделение неструктурированных сеток для масштабируемых параллельных вычислений на современных гибридных архитектурах. Обобщение послойного разделения без ветвления внутренних границ для параллельных вычислительных конечно-элементных алгоритмов на неструктурированных сетках в многосвязных областях выполнено в [3].

В данной работе анализируется алгоритм поэлементной схемы метода конечных элементов на двухуровневом послойном разделении, полученном на основе пространственного графа Риба для многосвязной триангулированной области, и результаты тестирования параллельных алгоритмов с оценками их ресурсной эффективности в универсальной модели масштабируемости и модели производительности.

§ 1. Оптимизации операций адресации и доступа к сеточным данным

Параллельные конечно-элементные вычисления предполагают применение алгоритмов, в которых кроме параллельных и последовательных вычислений существуют операции, суммирующие данные из параллельных ветвей алгоритма на основе некоторого разделения. Ранее такие операции названы композицией или операциями композиции [2].

На этапе сборки в МКЭ выполняется суммирование локальных матриц жесткости конечных элементов в глобальную матрицу жесткости системы конечно-элементных уравнений. Таким образом, каждый элемент k_{IJ} глобальной матрицы жесткости K есть сумма соответствующих элементов k^e_{ij} локальных матриц жесткости K^e. Индексы I и J принадлежат отрезку $[1, N] \subset \mathbb{Z}$, а соответствующие им локальные индексы i и j — отрезку $[1, N_e] \subset \mathbb{Z}$, где N — число степеней свободы всей системы, а N_e — число степеней свободы конечного элемента e; как правило, $N_e \ll N$. Запишем в матричном виде: $K = \sum_{e=1}^{m} C_e^T K^e C_e$, $K \in \mathbb{R}^{N\times N}$, $K^e \in \mathbb{R}^{N_e\times N_e}$; C_e — матрица инцидентности $\mathbb{Z}^{N_e\times N}$, которая отображает локальное пространство степеней свободы (номеров неизвестных и уравнений в сеточной СЛАУ) $[1, 2, \ldots, N_e]$ в глобальное $[1, 2, \ldots, N]$; m — число конечных элементов (ячеек сетки). Такое отображение в конечно-элементном приложении реализуется при косвенной индексации неизвестных с использованием номеров узлов сетки или умножением на матрицу инцидентности, наиболее эффективно выполняемым на графических ускорителях. Хранение матрицы K существенно ограничивает размер используемой сеточной модели и представляет интерес применительно к модели общей памяти. При использовании суперкомпьютеров более эффективно параллельно формировать блоки данной матрицы в памяти вычислительных модулей без хранения всей K на одном модуле с последующим разделением на блоки.

Другим подходом, широко применяемым в МКЭ, являются так называемые поэлементные схемы [4], в которых сборка переносится на этап решения конечно-элементной системы итерационным методом и применяется уже не к матрицам, а к векторам — результатам матрично-векторного произведения в следующем виде:

$$q = Kp = \sum_{e=1}^{m} C_e^T K^e C_e p = \sum_{e=1}^{m} C_e^T \tilde{q}_e = \sum_{e=1}^{m} q_e, \tag{1.1}$$

здесь $q, p, q_e \in \mathbb{R}^N$, $\tilde{q}_e \in \mathbb{R}^{N_e}$.

При распараллеливании выражения (1.1) в рамках модели общей памяти векторы p и q находятся в общей памяти параллельных процессов/нитей, а матрицы K^e, C_e, C_e^T и векторы q_e — в памяти соответствующего процесса. В этом случае конфликты, приводящие к ошибкам вычислений, возникают при суммировании векторов q_e, находящихся в разных процессах и имеющих ненулевые компоненты с одинаковыми номерами.

Представим K в следующем виде $K = C^T \tilde{K} C$, где $\tilde{K} \in \mathbb{R}^{\tilde{N}\times\tilde{N}}$ — блочно-диагональная матрица из блоков $\tilde{K}_{ii} = K^e$, $\tilde{N} = m \cdot N_e$, $C^T \in \mathbb{Z}^{N\times\tilde{N}}$ составлена из матриц $C_e^T \in \mathbb{Z}^{N\times N_e}$, а $C \in \mathbb{Z}^{\tilde{N}\times N}$ — из $C_e \in \mathbb{Z}^{N_e\times N}$.

Произведение $q = Kp = C^T\tilde{K}Cp$ запишем в виде последовательности операций: формирования разнесенного вектора $\bar{p} = Cp$, $\bar{p} \in \mathbb{R}^{\tilde{N}}$, произведения $\tilde{q} = \tilde{K}\tilde{p}$ и сборки $q = C^T\tilde{q}$, которые существенно отличаются по вычислительным и коммуникационным затратам, как следствие различной локализации данных (доступ по локальным и глобальным номерам узлов), особенно в случае неструктурированных сеток, и, кроме того, имеют разный потенциал распараллеливания. Критической с точки зрения распараллеливания является операция параллельной сборки, при которой возникают конфликты в общей памяти.

При распараллеливании на n потоков каждый параллельный процесс (поток) выполняет вычисления над $m_i \approx m/n$ конечными элементами, где i — номер процесса. Соответствующий вариант произведения имеет вид

$$q = Kp = \sum_{i=1}^{n} C^{T^{(i)}}\tilde{K}^{(i)}C^{(i)}p = \sum_{i=1}^{n} C^{T^{(i)}}\tilde{q}^{(i)} = \mathcal{A}(\tilde{q}), \tag{1.2}$$

где $C^{(i)}$ — матрица инцидентности, составленная из m_i матриц C_e; $\tilde{K}^{(i)}$ — блочно-диагональная матрица, состоящая из m_i матриц $\tilde{K}^e$, $\tilde{q}^{(i)}$ — вектор, компонентами которого являются m_i векторов $\tilde{q}_e$. Введем оператор сборки $\mathcal{A}$ вектора q, который выполняет суммирование, как внутри потока i в виде $q^{(i)} = C^{T^{(i)}}\tilde{q}^{(i)}$, так и между потоками $q = \sum_{i=1}^{n} q^{(i)}$.

Для того чтобы не допустить указанные выше конфликты, необходимо исключить одновременный доступ к общей памяти из параллельных процессов/потоков при сборке $\mathcal{A}(\tilde{q})$. Существующие методы и подходы к разрешению конфликтов можно разделить на программные, алгоритмические и методы раскраски разделения графов.

§ 1.1. Программные и алгоритмические подходы

Разрешение конфликтов в рамках технологии многопоточного программирования OpenMP осуществляется при помощи директивы `critical`. На рис. 1 представлен фрагмент кода, в котором суммирование компонент вектора `qe` в компоненты вектора `q` выполняется одной нитью OpenMP в критической секции. В представленном программном коде использован целочисленный массив `I[m][Ne]`, задающий соответствие между глобальными `I` и локальными `i` номерами степеней свободы. Преимущество подхода в том, что массив `I[m][Ne]` как правило является частью структуры данных для хранения неструктурированной сетки (список номеров узлов для каждой ячейки сетки). К сожалению, недостаток данного подхода — замедление работы даже в сравнении с последовательным вариантом.

```
#pragma omp parallel for
for(int e=0; e<m; e++) // цикл по конечным элементам
  for(int i=0; i<Ne; i++) // цикл по степеням свободы
                          // в конечном элементе
  {
    #pragma omp critical // критическая секция
         q[I[e][i]] += qe[Ne*e+i];
  }
```

Рис. 1. Поэлементная сборка вектора с синхронизацией

Следующий уровень — это изменение алгоритма сборки. Классические циклы по конечным элементам и степеням свободы в каждом конечном элементе заменяются на цикл по глобальным степеням свободы (рис. 2). В этом случае необходимо сформировать целочисленные массивы `e_[N]` и `i_[N]`, задающие соответствие между глобальными степенями свободы `I` и номерами конечных элементов `e`, а также между `I` и локальными степенями свободы `i`.

Развитием данного подхода является умножение на матрицу инцидентности (рис. 3). Матрично-векторное умножение эффективно реализуется на графических процессорах, а разреженность матрицы и значения ее ненулевых элементов позволяют уменьшить затраты как на ее хранение (хранятся начальные позиции строк `bI_[N+1]` и номера столбцов `CJ_[Nnz]` ненулевых элементов

```
#pragma omp parallel for
      for(int I=0; I < N; I++) // цикл по степеням свободы
            q[I] += qe[Ne*e_[I]+i_[I]];
```

Рис. 2. Изменение направления сборки на «поузловое»

матрицы C), так и при вычислениях (опускается операция умножения). В этом случае необходимо определить Nnz — число ненулевых элементов матрицы C — и сформировать структуру для ее хранения.

```
#pragma omp parallel for
     for(int I=0; I < N; I++) // цикл по степеням свободы
         for(int pos=bI[I]; pos<bI[I+1]; pos++)
            q[I] += qfe[CJ[pos]];
```

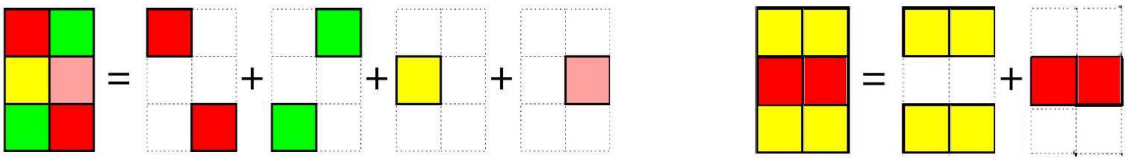
Рис. 3. Умножение на матрицу инцидентности

Поскольку на эффективность приведенных алгоритмов влияют характеристики вычислительных систем, то в качестве одного из подходов необходимо рассматривать совершенствование их требований к ресурсам оперативной памяти и доступу к расчетным данным. Кроме того, помимо временной эффективности алгоритма необходимо построить дополнительные оценки, учитывающие особенности алгоритма и его программной реализации при работе с общими расчетными данными.

§ 1.2. Разрешение конфликтов раскраской сетки

Конфликты при параллельной сборке векторов/матриц могут устраняться специальным упорядочением ячеек расчетной сетки. Для этого расчетная сетка разделяется на упорядоченные подмножества ячеек (конечных элементов) таким образом, что любая пара ячеек из разных подмножеств с одинаковыми номерами в подмножествах не имеет общих вершин. Данное условие можно смягчить, потребовав, чтобы не совпадали локальные номера общих вершин. Тогда при параллельной сборке каждый поток выполняет вычисления над данными, принадлежащими своему подмножеству конечных элементов.

В рамках данного подхода применяется многоцветная раскраска дуального графа или ячеек сетки (рис. 4). Так, в работе [5] ячейки сетки разделяется на конечное число подмножеств, обладающих тем свойством, что любые два элемента из подмножества не имеют общих вершин сетки. Раскраска осуществляется следующим образом: первой ячейке назначается некоторый «цвет», соседним ячейкам назначается «цвет», отличный от текущего, далее не раскрашенным ячейкам назначается новый «цвет». Раскраска завершается, когда все ячейки сетки раскрашены. В результате получается несбалансированное распределение ячеек по «цветам». На втором этапе происходит перераспределение ячеек по цветам с проверкой на наличие общих вершин. Перераспределение завершается, когда большинство подмножеств имеют мощность, близкую к средней, или когда из подмножеств большей мощности невозможно переместить ячейку из-за ограничения на связность ячеек. Результатом такого упорядочения ячеек является некоторое мозаичное разделение сетки, требующее перестроения структуры данных для хранения сетки.



а) б)

Рис. 4. Раскраска/упорядочение/разделение дуального графа или ячеек сетки: а) многоцветная раскраска ячеек сетки; б) послойное разделение

Развитием алгоритмов раскраски сетки стало послойное разделение, когда в качестве исходной берется не одна ячейка, а заданное множество ячеек на границе области, на межсеточных интерфейсах, на границах перестроения и других особенностях расчетных сеточных областей.

Реализация алгоритма послойного разделения и многочисленные эксперименты на сеточных областях различной геометрии показали, что использование только структуры соседства ячеек расчетной сетки для формирования несвязных слоев и формирование подобластей без разрывов оказывается недостаточно во многих случаях многосвязных областей и триангуляций при наличии локальных зон с резким изменением шага сетки.

В серии работ [3, 6, 7] развивается подход формирования послойного разделения на основе построения пространственного графа Риба для областей, имеющих произвольную топологию и использующих алгоритм послойного разделения на основе соседства (смежности) ячеек сетки [8].

§ 2. Послойное разделение сетки на основе пространственного графа Риба

Конечно-элементные сетки часто представляют графами и, в частности, для получения подобластей при разделении в параллельных методах декомпозиции области [9].

Используя теорию Морса, определим способ описания формы поверхности, основанный на эволюции поверхности изолиний уровней $f(z)$ (см. рис. 5, *а*). Для описания триангулированных расчетных областей будем применять дискретные функции Морса, которые имеют минимальный набор типов критических точек [10]. На их основе построим произвольные поверхности уровня и компоненты связности которых назовем слоями. Триангулированная поверхность области разбивается, таким образом, на объединение слоев и в результате получается слоение с особенностями.

Сопоставляя каждый слой одной точке и вводя естественную фактор-топологию в пространстве слоев, получаем некоторое фактор-пространство, которое является графом Риба функции Морса [11]. Вершинами графа Риба являются точки, отвечающие слоям исходной функции, содержащие критические функции. Вершины графа являются концевыми, если они являются концами ровно одного ребра графа и отвечают локальным минимумам и максимумам функции. Все остальные вершины внутренние, которые отвечают особым слоям функции, содержащим седловые критические точки (см. рис. 5, *б*). Тогда граф представляет собой структуру, описывающую критические точки заданной поверхности, определяет эту поверхность однозначно и позволяет восстановить топологию поверхности (см. рис. 5, *в*).

Отметим, что граф Риба хорошо отражает основную топологию двумерных областей, при анализе многосвязных трехмерных областей пропускаются важные особенности, такие как внутренние отверстия и др. (см. рис. 5).

Для точного описания ветвления топологии в случае многосвязных трехмерных областей использование только поверхностной триангуляции и одной дискретной функции Морса оказывается недостаточно, и для построения графа Риба необходимо использовать полное представление объемной сетки в расчетной области и вводить пространственный граф Риба (см. рис. 6).

Определим дерево связей между критическими точками v^c и построим граф Риба $R(V, E)$ с ребрами $e(v_i, v_j) \in E$ для триангуляции, состоящей из 3-симплексов. В отличии от реализованного ранее подхода формирования графа Риба [4], в данном алгоритме используется объемная триангуляция области, а 2-симплексы триангуляции, принадлежащее поверхности области используется только для формирования плоского графа.

Будем использовать функции «уровня» Морса f в разных координатных плоскостях. Для определения критических точек v^c введем параметр $c(v^c)$, соответствующий числу исходящих из нее ребер и принимающий значение: 1 — одно ребро; 2 — два ребра; 3 — ветвь; 0 — вспомогательная точка v^a, которая с ближайшими критическими точками v^c образуют криволинейные дуги между критическими точками при ветвлении (см. рис. 6) .

Насколько нам известно, в литературе не был предложен алгоритм разделения неструктурированных сеток на основе пространственного графа Риба, что является новым направлением в построении многоуровневых разделений. Более подробное описание алгоритма построения пространственного графа Риба представлено в [6, 7].

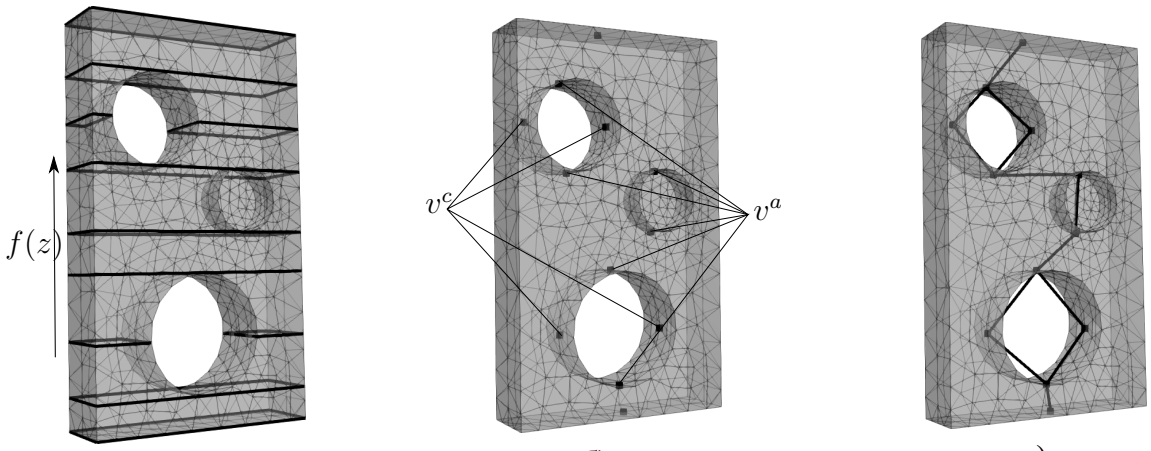


Рис. 5. Построение плоского граф Риба: а) функция Морса — линии уровня по $f(z)$; б) критические v^c и вспомогательные v^a вершины; в) плоский граф Риба $R(V, E)$.

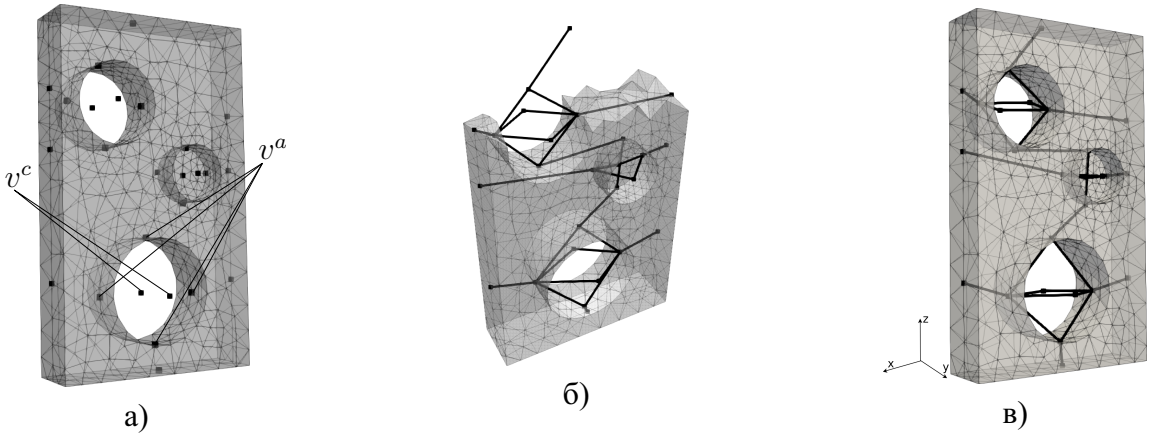


Рис. 6. Построение пространственного графа Риба $R_V(V, E)$: а) критические вершины функций Морса по z и x; б) граф слоя по $f(x)$; в) граф Риба двух функций Морса $f(z)$ и $f(x)$.

Разделение трехмерных моделей на более простые подобласти перспективно во многих направлениях исследований: в задачах реконструкции геометрических моделей из триангулированных областей и воксельных моделей с последующей объемной декомпозицией на конструкторско-технологические элементы; построения в задачах сжатия и восстановления сеток в области вычислительной геометрии и графики; в методах пространственно-временной декомпозиции при численном решении уравнений; при поиске топологии в оцифрованных моделях в задачах компьютерного зрения и многих других.

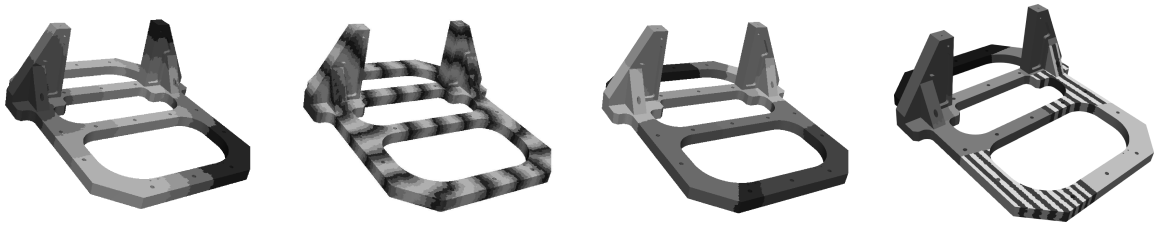
Получение разделения на заданное число подобластей может быть обеспечено рекурсивным делением полученных подобластей неструктурированной сетки и выделением слоев ячеек, обеспечивающих независимый доступ к данным.

Перспективные гибридные вычислительные архитектуры предполагают локализацию используемых расчетных сеточных данных в блоках малой размерности, обеспечивающих многократные обращения и копирование. Рассмотренные алгоритмы разделения позволяют получать подобласти, имеющие только две общие границы и состоящие из неперекрывающихся слоев ячеек [2], которые не содержат общих вершин и допускают параллельную обработку, а также их возможную дальнейшую одномерную декомпозицию. Построенные разделения сеточных областей благодаря исключению ветвлений внутренних границ уменьшают количество и упрощают структуру обменов между вычислительными процессами в параллельных алгоритмах сеточных методов. Послойное упоря-

дочение ячеек в подобластях неструктурированной сетки исключает конфликты при вычислениях в общей памяти вычислительных систем с гибридной архитектурой.

§ 3. Численные исследования

Рассмотрим декомпозицию неструктурированной тетраэдральной расчетной сетки для области типа «рама», содержащей 485843 ячеек и 120949 узлов. Сложность трехмерной геометрии области можно охарактеризовать наличием 8 глухих и 24 сквозных отверстий с фасками, различными выточками.



а) б) в) г)

Рис. 7. Разделение области «рама»: а) блочное; б) нечетное+четное; в) METIS; г) послойное разделение графа Риба в подобластях, полученных METIS

На рис. 7 представлены разделения области «рама», полученные объединением слоев ячеек сетки [9] многоуровневым графовым алгоритмом k-way из METIS [12], а также с послойным разделением пространственного графа Риба.

В блочном-варианте подобласти сетки образуются последовательным объединением соседних слоев, в варианте нечетное+четное каждая подобласть состоит из последовательности нечетных слоев, за которой следует последовательность четных слоев. Представленные алгоритмы, используют слои ячеек сетки и приводят к разделению без ветвления внутренних границ. На рис. 7, *г* показано послойное разделение с помощью функции Морса для подобластей, полученных алгоритмом METIS k-way. Послойное разделение в подобластях содержало от 20 до 35 слоев и для всей области соответственно 146 слоям. Сбалансированность вычислений в слоях достигается различными вариантами их объединения, обеспечивающих также независимый доступ к данным.

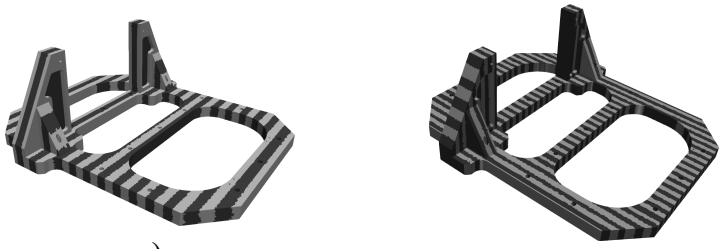
а) б) в)

Рис. 8. Разделение многосвязной сеточной области на подобласти: а) функцией Морса $f(y)$; б) функцией Морса $f(z)$; в) разделение METIS k-way.

На рис. 8, *а*, *б* показаны разделения на подобласти без ветвления различными функциями Морса по разным координатным осям.

На рис. 9 изображено послойное разделение с помощью функции Морса по разным координатным осям. Послойное разделение на рис. 9, *а* по функции Морса $f(y)$ имеет 23 слоя, где в каждой подобласти находится от 10234 до 18231 ячеек. Разделение на рис. 9, *б* по функции Морса $f(z)$ имеет 48 слоев в каждом из которых находится от 8526 до 15351 ячеек.

Важным фактором является возможность исключения конфликтов при параллельной операции сборки вектора q за счет послойного разделения сетки. В таблице 1 приведена оценка возможных



а) б)

Рис. 9. Послойное разделение функцией Морса многосвязной сеточной области по разным координатным осям: а) по оси y; б) по оси z

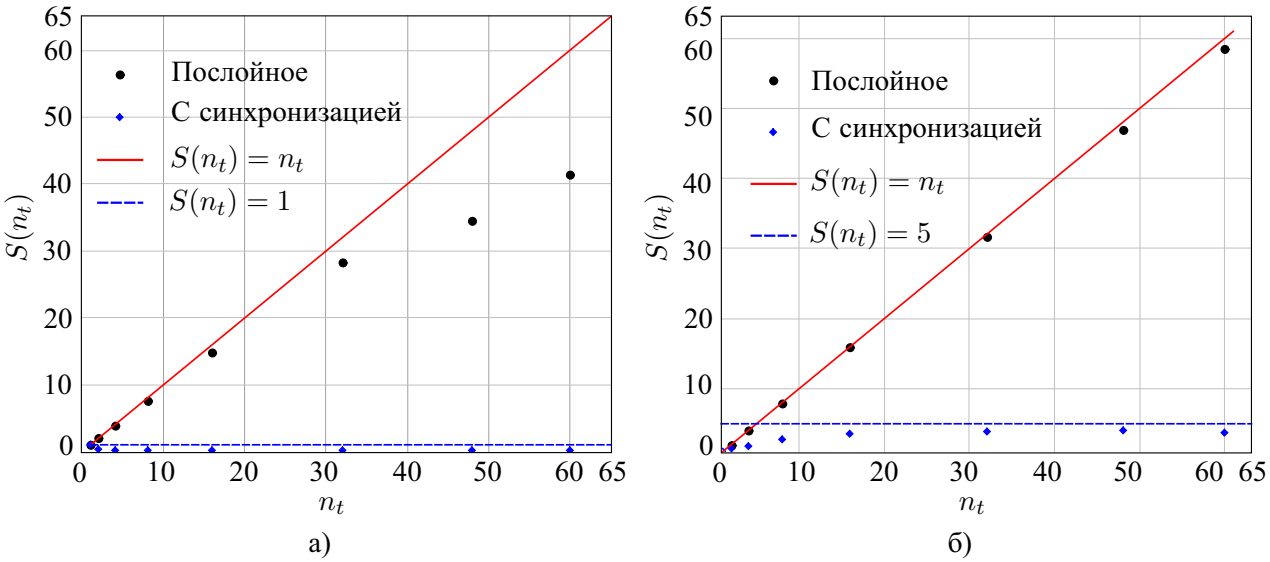
конфликтов, построенная на совпадении номеров ячеек (числитель), содержащих общие узлы и позиций узлов (знаменатель) в ячейках в подобластях сетки, принадлежащих разным параллельным потокам.

Таблица 1. Число возможных конфликтов при параллельной сборке вектора q.

n	**по построению**	**METIS**	**блочное**	**нечетное+четное**	**3D граф Риба**
Расчетная область «рама», $m = 485843$					
8	284/88	7/2	0	0	0
16	623/198	13/4	0	0	0
32	1144/352	60/25	0	0	0
60	2425/815	169/71	167/40	4/3	0

Появление конфликтов при разделении на 60 подобластей связано с отсутствием упорядочения ячеек в слоях. Эти конфликты исключаются при введении упорядочения ячеек по координатным направлениям или вдоль поверхности, проходящей через центры ячеек слоя, а также в случае подобластей состоящих из двух и более слоев ячеек сетки.

Как видно на рис. 10, *а*, применение критических секций для разрешения конфликтов существенно снижает ускорение параллельной сборки, что приводит к уменьшению ускорения при вычислении $q = Kp$ на 61-ядерном процессоре архитектуры MIC до уровня многоядерного CPU (рис. 10, *б*). В свою очередь, применение послойного разделения позволило получить ускорение близкое к ли-



а) б)

Рис. 10. Ускорение на сопроцессоре архитектуры MIC: а) сборки $q = \mathcal{A}(\tilde{q})$; б) произведения $q = Kp$ в виде (1.1)

нейному $S(n_t) = n_t$, здесь n_t — число потоков OpenMP.

Выполнены исследования ускорения параллельной сборки на многоядерных процессорах различной архитектуры, в ходе которых строилась модель ускорения на основе экспериментальных данных и универсальной модели масштабируемости USL (Universal Scaling Law) [13], также прогнозировалось максимально достижимое ускорение.

Универсальная модель масштабируемости Гюнтера (USL) с тремя параметрами имеет вид

$$S_U(n_t) = \frac{\mu n_t}{1 + \sigma(n_t - 1) + \lambda n_t(n_t - 1)}, \tag{3.1}$$

где μ — коэффициет, определяющий линейную масштабируемость; σ — доля сериализованной (непараллелизируемой) части вычислений; λ — коэффициент взаимного влияния, которое происходит при взаимодействии потоков.

Наиболее характерные варианты параметров, определяющие поведение модели, представим в следующем виде:

- $0 < \mu \leqslant 1$, $\sigma, \lambda = 0$ — линейная (неограниченная) масштабируемость (отдельные составляющие системы работают без взаимодействий);
- $0 < \mu \leqslant 1$, $\sigma > 0$, $\lambda = 0$ — масштабируемость ограничена конкуренцией (конфликты между частями системы, вызванные сериализацией и очередями) в пределе равна $1/\sigma$, что соответствует закону Амдаля;
- $0 < \mu \leqslant 1$, $\sigma \gg 0$, $\lambda = 0$ — число конфликтов увеличивается и рост ускорения сильно замедляется;
- $0 < \mu \leqslant 1$, $\sigma \gg 0$, $\lambda > 0$ — масштабируемость ограничена когерентностью (поддержание системы в согласованном состоянии кэш-памяти) и достигает максимального значения при $n_{\max} = \sqrt{(1-\sigma)/\lambda}$, а при большем числе потоков параллельное ускорение замедляется;
- $\mu > 1$ — достижение сверхлинейного ускорения в модели (3.1) возможно в среднем (при некотором числе потоков), если предположить $\sigma < 0$, а размеры параллельных частей эффективно помещаются в кэши-памяти процессора.

В модели (3.1) наличие сериализации и очередей приводит к тому что будет наблюдаться асимптотическое приближение к пределу параллельного ускорения. При максимальной оценке затрат на согласование двух взаимодействующих потоков на полном графе модель описывает минимально достижимые ускорения. Тем самым в модели представлены оценки двух механизмов достижения и возможности прогнозирования высокой масштабируемости, связанные с уменьшением затрат на сериализацию, создание очередей последовательной части алгоритма и затрат на согласование работы параллельных частей алгоритма (см. табл. 2).

Слагаемые в модели (3.1) с параметрами σ и λ можно представить как часть времени последовательного выполнения алгоритма. При $\lambda, \mu = 0$ и $\sigma = (1 - f)$, где $(1 - f)$ – последовательная часть алгоритма, получаем закон Амдаля:

$$S_A(n_t) = \frac{1}{(1-f) + \frac{f}{n_t}}.$$

Тогда можно говорить о том, что вычисленная величина, например $\lambda = 0.01$, составляет один процент затрачиваемого времени на синхронизацию и согласования процессов от времени последовательного выполнения алгоритма.

Таким образом, при наличии данных измерений времени выполнения и ускорения вычислений параллельного алгоритма при различном числе процессов на том или процессоре или ускорителе вычислений, параметры модели ускорения (3.1) определяются при помощи регрессионного анализа.

Здесь необходимо отметить, что при регрессионном анализе времени параллельного выполнения небольшого числа потоков возможно получение параметра σ меньше нуля, тем самым допускается возможность появления значения локального сверхлинейного ускорения. Подобное поведение также

Таблица 2. Соответствие характеристик процессоров и параметров USL модели при операции сборки

	Xeon E5-2609	Opteron 8435	Xeon E5-2690	Xeon Phi 7110X	GeForce GTX 980
n_t	4	6	8	61	2048
	Пропускная способность, ГБ/с				
		pmbw		mic-meter	gpumemb
L1	27/**69**	8/57	30/199	45/2700	**1155**
L2	25/62	7/57	25/179	3.5/–	480
L3	22/53	7/20	20/150	–	–
RAM	7/**16**	3/5	7/18	5/160	**180**
		$\mathcal{A}^{ind}$			$\mathcal{A}^{inc}$
σ	0.1	0.021	0.08	0.006	0.0018
λ	0.0001	0.02	0.0001	0.00006	0.00000044
$(S/n)_{\max}$	3.1/4	3.5/5	5.2/8	39.4/60	219/384
$(S_{USL}/n_{USL})_{\max}$	8.4/ 94	3.6/7	10/96	46.8/129	318.9/1511

характерно в случае неточной оценки времени выполнения алгоритма одним потоком. В дальнейшем рассматривались варианты, когда в регрессионном анализе накладывались ограничения на положительное значения коэффициентов модели, так и без ограничений.

В качестве процедуры минимизации использовалась эффективная реализация процедуры Левенберга–Марквардта решения задач нелинейных наименьших квадратов, усиленная пересчетом матрицы Гессе целевой функции по схеме переменной метрики — алгоритм NL2SOL [14].

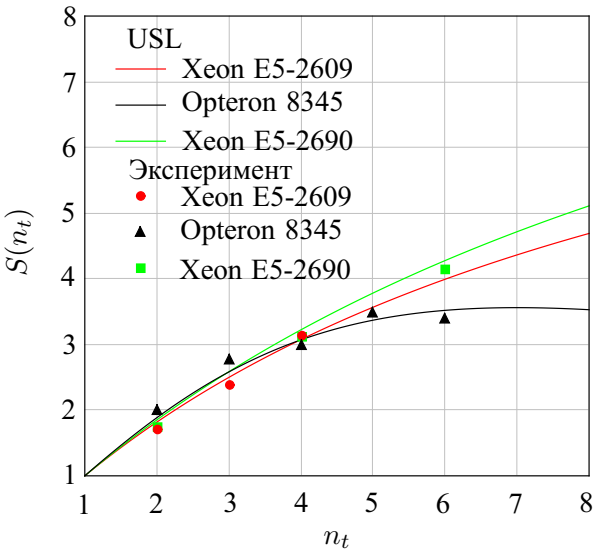
В рамках построенной модели ускорения оценивались как операция параллельной сборки на различных аппаратных архитектурах (см. рис. 11), так и матрично-векторное произведение $q = Kp$ в целом в виде (1.2).

На рисунке 11, *б*) показана масштабируемость ускорения операции $q = \mathcal{A}(\tilde{K}\tilde{p})$ на вычислительном узле с процессором Intel Xeon E5-2697A v.4 (16 ядер, базовая тактовая частота 2.60 ГГц, кэш L1 (512 кБ инструкций, 4096 кБ данных), L2 — 4096 кБ, L3 — 40 МБ SmartCache, каналов памяти — 4) при использовании двух компиляторов (`g++` (GCC) 4.8.5 и `icpc` (ICC) 17.0.4), оптимизации третьего уровня (`-O3`)).

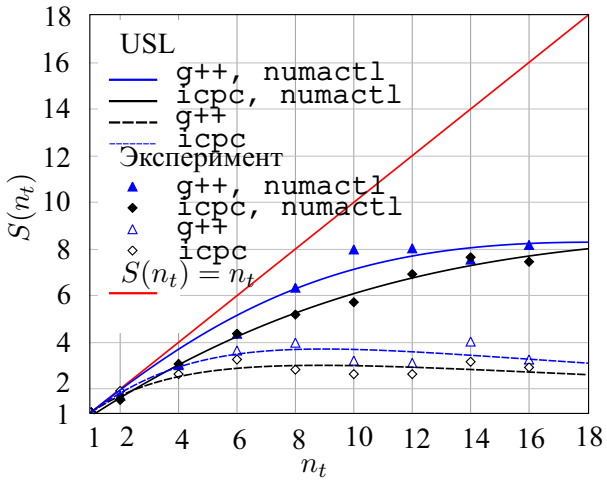
В ходе вычислительных экспериментов запуск приложения, в котором реализованы параллельные алгоритмы матрично-векторного произведения $q = \mathcal{A}(\tilde{K}\bar{p})$, осуществлялся с помощью утилиты `numactl` с опцией `--interleave=all`, уменьшающей частоту доступа в дальнюю память. Применение данной утилиты привело к увеличению значений ускорения примерно вдвое.

Анализ данных, полученных по двухпараметрической модели USL показывает, что компилятор `icpc` генерирует исполняемый код с вдвое меньшим значением параметра σ, т. е. уменьшением последовательной части, связанной с очередями. Применение утилиты `numactl` при запуске приложения приводит к уменьшению параметра σ в пять раз для `g++` и только в два — для `icpc`. Значение параметра λ в таком случае для `g++` уменьшается в три раза, а для `icpc` стремится к нулю или становится отрицательным, что говорит о сокращении задержек за счет сохранения локальности доступа памяти.

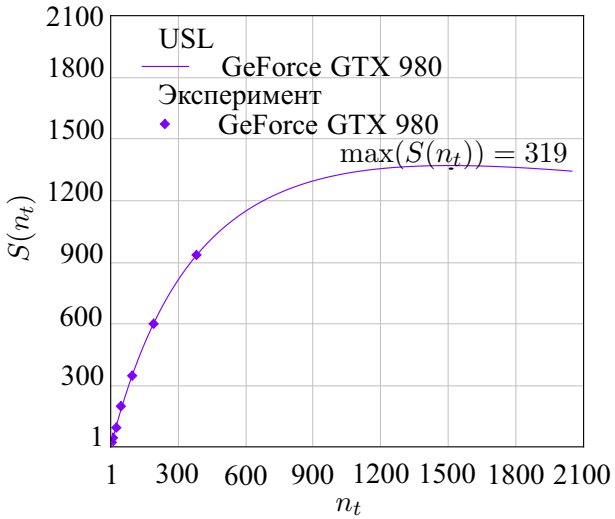
При анализе масштабируемости с помощью трехпараметрической модели USL прогнозируется увеличение производительности приложения (увеличивается значение множителя в числителе модели USL) при использовании утилиты `numactl` в сочетании с компилятором `g++` и ее уменьшение при использовании `icpc`. Параметр σ изменяется от 10^{-8} до соразмерного параметру λ в случае `g++`, и соответственно проблема задержек, связанных очередями, представляется более значимой. Оценка параметров σ и λ в случае применения компилятора `icpc` и утилиты `numactl` изменяются несущественно, при этом $\sigma/\lambda \approx 0.1$. Предварительные оценки в модели USL с тремя параметрами



а)



б)



в)

Рис. 11. Ускорение сборки $q = \mathcal{A}(\tilde{q})$: а) на многоядерных процессорах с различным числом ядер; б) при использовании компиляторов GCC, ICC; в) на ускорителе GPU

показывают, что требуется проведение дополнительных исследований на большем числе исходных данных и вариантов ускорения по отдельным базовым операциям и алгоритмам.

Для того, чтобы определить аппаратные ограничения, накладываемые на распараллеливание вычислений, оценим представленные алгоритмы в рамках модели производительности Roofline [15]

$$P, (\text{Гфлоп/с}) = \begin{cases} P_{\text{пик.}}, (\text{Гфлоп/с}), \\ B, (\text{ГБ/с}) \cdot I, (\text{флоп/Б}). \end{cases} \tag{3.2}$$

Здесь $I = W/Q$ — арифметическая интенсивность операций алгоритма; W — число арифметических операций с плавающей точкой; Q — число байт, переданных процессору; B — пропускная способность.

В табл. 3 представлены арифметические интенсивности операций поэлементного произведения (1.2). Число байт, переданных процессору при вычислении $\tilde{q} = \tilde{K}\bar{p}$, состоит из $N_e^2 \cdot m$ чтений значений двойной точности (8 байт) из $\tilde{K}$ и $N_e \cdot m$ из $\bar{p}$, и $N_e \cdot m$ записей в $\tilde{q}$, $Q(\tilde{K}\bar{p}) = 8(N_e^2 + 2N_e)m$ байт.

При сборке $q = \mathcal{A}(\tilde{q})$ происходит чтение и запись $N_e \cdot m$ значений двойной точности и также используются номера ячейки сетки и локальные номера степеней свободы — $Q(\mathcal{A}(\tilde{q})) = N_e \cdot m \cdot (2 \cdot 8 + +2 \cdot 4) = 24N_e \cdot m$ байт. Отметим, что арифметическая интенсивность представленных операций не

зависит от числа используемых потоков и числа конечных элементов.

Таблица 3. Арифметическая интенсивность поэлементного произведения

Операция	W, флоп	Q, Б	I, флоп/Б
$\tilde{q} = \tilde{K}\bar{p}$	$2N_e^2 \cdot m$	$(8N_e^2 + 16N_e)m$	$1/(4 + 8/N_e)$
$q = \mathcal{A}(\tilde{q})$	$N_e \cdot m$	$24N_e \cdot m$	1/24
$q = \mathcal{A}(\tilde{K}\bar{p})$	$(1 + 2N_e)N_e \cdot m$	$(8N_e^2 + 40N_e)m$	$(1 + 2N_e)/(8N_e + 40))$

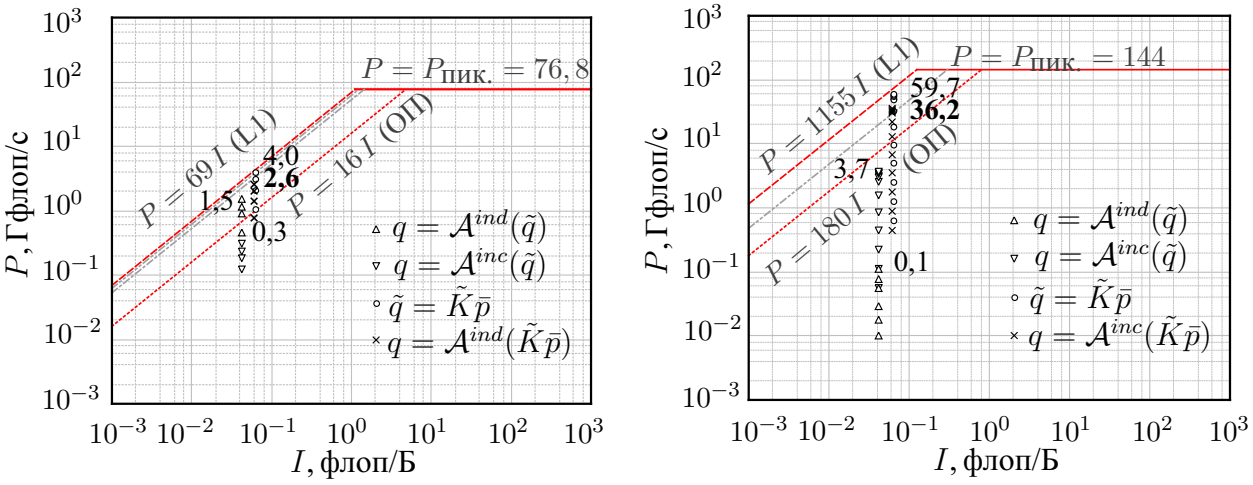


Рис. 12. Производительность процессоров при выполнении сборки в виде $\mathcal{A}^{ind}$ и $\mathcal{A}^{inc}$, произведения (1.2) и всего поэлементного произведения $q = \mathcal{A}(\tilde{K}\bar{p})$: а) на четырехядерном процессоре Intel Xeon E5-2609; б) на графической карте NVIDIA GeForce GTX 980

Пропускная способность B подсистемы памяти, использованная при построении зависимостей Roofline [15] была получена в результате тестирования данных архитектур тестами (при полной загрузке процессоров):

- `pmbw` (`https://panthema.net/2013/pmbw/`);
- `mic-meter` (`https://github.com/haibo031031/mic-meter`);
- `gpumemb` (`https://github.com/ekondis/gpumembench`).

В табл. 2 приводятся данные для кэшей трех уровней (L1, L2, L3) и оперативной памяти (ОП), выделены значения, примененные при построение зависимостей, представленных на рис. 12.

Результаты моделирования производительности для мульти- и многоядерного процессоров представлены на рис. 12.

Небольшая арифметическая интенсивность рассматриваемых алгоритмов приводит к тому, что их производительность ограничивается пропускной способностью подсистемы памяти, а не производительностью процессоров. Большая пропускная способность графической памяти (в сравнении с оперативной) позволяет существенно увеличить производительность алгоритма на GPU: произведения $\tilde{K}\bar{p}$ — в 15 раз, сборки — в 2.5, всего поэлементного произведения $q = \mathcal{A}(\tilde{K}\bar{p})$ — примерно в 14 раз.

Вычислительные эксперименты проводились на вычислительных системах ИММ УрО РАН и МСЦ РАН — филиал ФГУ ФНЦ НИИСИ РАН.

Финансирование. Работа выполнена при частичной финансовой поддержке Российского Фонда Фундаментальных Исследований (грант 17–01–00402-а).

Список литературы

1. Копысов С.П., Новиков А.К., Пономарев А.Б., Рычков В.Н., Сагдеева Ю.А. Программная среда построения расчетных моделей метода конечных элементов для параллельных распределенных вычислений // Информационные технологии. 2008. № 3. С. 75–82.
2. Копысов С.П., Новиков А.К., Недожогин Н.С., Караваев А.С. Послойное упорядочение ячеек для задач разделения, отображения и параллельных вычислений без конфликтов на неструктурированных сетках // Параллельные вычислительные технологии — XI международная конференция, ПаВТ'2017, Казань, 2017. Челябинск: Издательский центр ЮУрГУ, 2017. С. 386–398.
3. Кадыров И.Р., Копысов С.П., Новиков А.К. Разделение триангулированной многосвязной области на подобласти без ветвления внутренних границ // Учен. зап. Казан. ун-та. Сер. Физ.-матем. науки. 2018. Т. 160. № 3. С. 544–560. http://mi.mathnet.ru/uzku1477
4. Копысов С.П., Новиков А.К. Метод декомпозиции для параллельного адаптивного конечно-элементного алгоритма // Вестник Удмуртского университета. Математика. Механика. Компьютерные науки. 2010. Вып. 3. С. 141–154. https://doi.org/10.20537/vm100315
5. Komatitsch D., Michea D., Erlebacher G. Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA // Journal of Parallel and Distributed Computing. 2009. Vol. 69. Issue 5. P. 451–460. https://doi.org/10.1016/j.jpdc.2009.01.006
6. Kadyrov I.R., Kopysov S.P., Novikov A.K. Partitioning of an arbitrary domain into subdomains without branching of inner boundaries // Journal of Physics: Conference Series. 2019. Vol. 1158. Issue 3. 032001. https://doi.org/10.1088/1742-6596/1158/3/032001
7. Kadyrov I.R., Kopysov S.P., Novikov A.K. Parallel partitioning without branching of inner boundaries for arbitrary domain // Proceedings of the 4th Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists (Ural-PDC 2018). Yekaterinburg, Russia. CEUR Workshop Proceedings. 2018. Vol-2281. P. 60–66. http://ceur-ws.org/Vol-2281/
8. Novikov A., Piminova N., Kopysov S., Sagdeeva Yu. Layer-by-layer partitioning of finite-element meshes for multi-core architectures // Communications in Computer and Information Science. Cham: Springer, 2016. P. 106–117. https://doi.org/10.1007/978-3-319-55669-7_9
9. Копысов С.П., Новиков А.К. Методы декомпозиции: разделение расчетных сеток. Ижевск: Издательский центр «Удмуртский университет», 2018. 102 с.
10. Постников М.М. Введение в теорию Морса. М.: Наука, 1971. 568 с.
11. Иванов А.О., Тужилин А.А., Фоменко А.Т. Компьютерное моделирование кривых и поверхностей // Фундамент. и прикл. матем. 2009. Т. 15. Вып. 5. С. 63–94.
12. Karypis G., Kumar V. Multilevel k-way partitioning scheme for irregular graphs // Journal of Parallel and Distributed Computing. 1998. Vol. 48. No. 1. P. 96–129. https://doi.org/10.1006/jpdc.1997.1404
13. Gunther N.J., Puglia P., Tomasette K. Hadoop superlinear scalability // Communications of the ACM. 2015. Vol. 58. No. 4. P. 46–55. https://doi.org/10.1145/2719919
14. Dennis J.E. (Jr.), Gay D.M., Welsch R.E. Algorithm 573: NL2SOL — An adaptive nonlinear least-squares algorithm // ACM Transactions on Mathematical Software. 1981. Vol. 7. No. 3. P. 369–383. https://doi.org/10.1145/355958.355966
15. Williams S., Waterman A., Patterson D. Roofline: an insightful visual performance model for multicore architectures // Communications of the ACM. 2009. Vol. 52. No. 4. P. 65–76. https://doi.org/10.1145/1498765.1498785

Поступила в редакцию 01.05.2019

Копысов Сергей Петрович, д. ф.-м. н., профессор, заведующий кафедрой вычислительной механики, Удмуртский государственный университет, 426034, Россия, г. Ижевск, ул. Университетская, 1.
E-mail: s.kopysov@gmail.com

Кадыров Ильяс Ринатович, аспирант, кафедра вычислительной механики, Удмуртский государственный университет, 426034, Россия, г. Ижевск, ул. Университетская, 1.
E-mail: slasheeck@gmail.com

Новиков Александр Константинович, к. ф.-м. н., доцент, кафедра вычислительной механики, Удмуртский государственный университет, 426034, Россия, г. Ижевск, ул. Университетская, 1.
E-mail: sc_work@mail.ru

S. P. Kopysov, I. R. Kadyrov, A. K. Novikov
Resource efficient finite element computing on multicore architectures

Citation: *Izvestiya Instituta Matematiki i Informatiki Udmurtskogo Gosudarstvennogo Univiversiteta*, 2019, vol. 53, pp. 83–97 (in Russian).

Keywords: finite element methods, parallel computing, partitioning mesh, Reeb graph, arithmetic intensity, universal scalability model.

MSC2010: 65Y20, 65F10, 05C15

DOI: 10.20537/2226-3594-2019-53-08

In this paper, we consider the construction of efficient finite element algorithms on three-dimensional unstructured meshes that take into account the complex parallel synchronization processes, the memory distribution problems and data storage. A layer-by-layer partitioning of the meshes into subdomains without branching internal boundaries is proposed to simplify the access to independent data and parallel computing at different stages of the finite element problem solving on unstructured meshes in multiply connected domains. The predictive capacity of the time efficiency and resource intensity for the proposed algorithmic solutions is analyzed. The analysis of the resource efficiency of the algorithms is given for the element-by-element scheme for forming and solving the system of linear algebraic equations of the finite element method. It is shown that the low arithmetic intensity of the algorithms considered results in the fact that their performance is limited by the bandwidth of the memory subsystem rather than by the processors'performance. The graphic memory has a larger bandwidth than the random-access memory. This allows a significant increase in the performance of the algorithm on GPU.

Funding. The research was partially supported by the Russian Foundation for Basic Research (project no. 17–01–00402-a).

REFERENCES

1. Kopysov S.P., Novikov A.K., Ponomarev A.B., Rychkov V.N., Sagdeeva Yu.A. A program environment for construction of computational models for parallel distributed computing, *Informatsionnye Tekhnologii*, 2008, no. 3, pp. 75–82 (in Russian).
2. Kopysov S.P., Novikov A.K., Nedozhogin N.S., Karavaev A.S. Layer-by-layer ordering of cells for problems of partitioning, mapping and parallel computing with conflict-free access on unstructured grids, *Parallel computational technologies (PCT)'2017: Proceedings of Int. Conf.*, Kazan Federal University, Kazan, 2017. Chelyabinsk: South Ural State University, 2017, pp. 386–398 (in Russian). http://omega.sp.susu.ru/pavt2017/short/044.pdf
3. Kadyrov I.R., Kopysov S.P., Novikov A.K. Partitioning of triangulated multiply connected domain into subdomains without branching of inner boundaries, *Uchenye Zapiski Kazanskogo Universiteta. Ser. Fiziko-Matematicheskie Nauki*, 2018, vol. 160, no. 3, pp. 544–560 (in Russian). http://mi.mathnet.ru/eng/uzku1477
4. Kopysov S.P., Novikov A.K. Domain decomposition for parallel adaptive finite element algorithm, *Vestnik Udmurtskogo Universiteta. Matematika. Mekhanika. Komp'yuternye Nauki*, 2010, issue 3, pp. 141–154 (in Russian). https://doi.org/10.20537/vm100315
5. Komatitsch D., Michea D., Erlebacher G. Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA, *Journal of Parallel and Distributed Computing*, 2009, vol. 69, issue 5, pp. 451–460. https://doi.org/10.1016/j.jpdc.2009.01.006
6. Kadyrov I.R., Kopysov S.P., Novikov A.K. Partitioning of an arbitrary domain into subdomains without branching of inner boundaries, *Journal of Physics: Conference Series*, 2019, vol. 1158, issue 3, 032001. https://doi.org/10.1088/1742-6596/1158/3/032001

7. Kadyrov I.R., Kopysov S.P., Novikov A.K. Parallel partitioning without branching of inner boundaries for arbitrary domain, *Proceedings of the 4th Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists (Ural-PDC 2018)*, Yekaterinburg, Russia, CEUR Workshop Proceedings, 2018, pp. 60–66. http://ceur-ws.org/Vol-2281/
8. Novikov A., Piminova N., Kopysov S., Sagdeeva Yu. Layer-by-layer partitioning of finite-element meshes for multi-core architectures, *Communications in Computer and Information Science*, Cham: Springer, 2016, pp. 106–117. https://doi.org/10.1007/978-3-319-55669-7_9
9. Kopysov S.P., Novikov A.K. *Metody dekompozitsii: razdelenie raschetnyh setok* (Decomposition methods: meshes partitioning), Izhevsk: Udmurt State University, 2018.
10. Postnikov M.M. *Vvedenie v teoriyu Morsa* (Introduction to Morse theory), Moscow: Nauka, 1971.
11. Ivanov A.O., Tuzhilin A.A., Fomenko A.T. Computer modeling of curves and surfaces, *Journal of Mathematical Sciences*, 2011, vol. 172, issue 5, pp. 663–689. https://doi.org/10.1007/s10958-011-0212-2
12. Karypis G., Kumar V. Multilevel k-way partitioning scheme for irregular graphs, *Journal of Parallel and Distributed Computing*, 1998, vol. 48, no. 1, pp. 96–129. https://doi.org/10.1006/jpdc.1997.1404
13. Gunther N.J., Puglia P., Tomasette K. Hadoop superlinear scalability, *Communications of the ACM*, 2015, vol. 58, no. 4, pp. 46–55. https://doi.org/10.1145/2719919
14. Dennis J.E. (Jr.), Gay D.M., Welsch R.E. Algorithm 573: NL2SOL — An adaptive nonlinear least-squares algorithm, *ACM Transactions on Mathematical Software*, 1981, vol. 7, no. 3, pp. 369–383. https://doi.org/10.1145/355958.355966
15. Williams S., Waterman A., Patterson D. Roofline: an insightful visual performance model for multicore architectures, *Communications of the ACM*, 2009, vol. 52, no. 4, pp. 65–76. https://doi.org/10.1145/1498765.1498785

Received 01.05.2019

Kopysov Sergei Petrovich, Doctor of Physics and Mathematics, Professor, Head of the Department of Computational Mechanics, Udmurt State University, ul. Universitetskaya, 1, Izhevsk, 426034, Russia.
E-mail: s.kopysov@gmail.com
Kadyrov Il'yas Rinatovich, Postgraduate Student, Department of Computational Mechanics, Udmurt State University, ul. Universitetskaya, 1, Izhevsk, 426034, Russia.
E-mail: slasheeck@gmail.com
Novikov Aleksandr Konstantinovich, Candidate of Physics and Mathematics, Associate Professor, Department of Computational Mechanics, Udmurt State University, ul. Universitetskaya, 1, Izhevsk, 426034, Russia.
E-mail: sc_work@mail.ru