КОМПЬЮТЕРНЫЕ НАУКИ

УДК 519.63

© *С. П. Копысов, А. К. Новиков*

МЕТОД ДЕКОМПОЗИЦИИ ДЛЯ ПАРАЛЛЕЛЬНОГО АДАПТИВНОГО КОНЕЧНО-ЭЛЕМЕНТНОГО АЛГОРИТМА[1]

Построен метод декомпозиции области для адаптивного МКЭ с перестроением сетки, который включает параллельные алгоритмы: решения систем линейных уравнений, апостериорной оценки погрешности, локального перестроения сетки и динамической балансировки вычислительной нагрузки. Исследована их эффективность и структура вычислительных затрат при выполнении на мультиядерных вычислительных системах.

Ключевые слова: методы декомпозиции, параллельные вычисления, адаптивное перестроение сетки, метод конечных элементов.

Введение

Применение метода конечных элементов (МКЭ) или других методов с адаптивным перестроением сеточной модели позволяет существенно сократить число неизвестных при решении современных прикладных задач. Сокращение может достигать нескольких порядков. Основные достоинства адаптивных методов реализуются только при наличии достаточно быстрых алгоритмов решения соответствующих систем алгебраических уравнений на неструктурированных сетках. Отметим, что адаптивные методы являются итерационными алгоритмами, использующими полученное приближенное решение, для повышения точности последующих приближений.

Существующие сегодня версии метода конечных элементов связаны с использованием адаптивных алгоритмов, когда при анализе полученного решения определяется стратегия дальнейших вычислений [1]: перестроение сетки без изменения связности конечных элементов (*r-версия*); локальное измельчение/огрубление сетки (*h-версия*); локальное или глобальное повышение порядка аппроксимирующих функций (*p-версия*). Каждый из этих способов адаптации имеет свои достоинства и недостатки, поэтому рассматривается и их комбинированное использование (*hp-версия*, *rp-версия* и т.д.).

Эффективное распараллеливание адаптивных версий МКЭ предполагает распараллеливание каждого из этапов решения задачи. При адаптивном перестроении сетки (*h-версии* МКЭ) на каждом шаге уточнения необходимо частично переформировать решаемую систему линейных алгебраических уравнений. В этом случае применение методов декомпозиции области таких, как метод подструктур [2], которому свойственно выделение крупноблочного параллелизма, становится нецелесообразным. Параллелизму, основанному на методах декомпозиции, присуща зависимость глобальной структуры алгоритма от архитектуры многопроцессорной вычислительной системы (МВС). Процесс проектирования максимально эффективного параллельного алгоритма связан с большими трудозатратами на поиск специфической структуры параллельного алгоритма, оптимальной для конкретной мультиядерной (multi-core) или многоядерной (many-core) архитектуры процессоров МВС. Спроектированная структура обеспечит минимальное время получения результата, но, скорее всего, будет неэффективна для другой МВС.

Вычисления на неструктурированных адаптивных сетках относятся к классу задач, имеющих нерегулярный доступ к данным, то есть мелкую гранулярность обращений, низкую пространственную и временную локализацию обращений к памяти.

[1]Работа выполнена при финансовой поддержке РЦП УрО РАН и РФФИ (гранты 09–01–00061, 10–01–96039-урал)

Для разработки новых параллельных вычислительных алгоритмов необходимо учитывать следующие характиристики мультиядерных МВС: для обмена информацией между приложением, которое работает на разных ядрах, требуется эффективный механизм межпроцессорного взаимодействия, инфраструктура данных в общей памяти, а также примитивы синхронизации, которые обеспечивают защиту общедоступных ресурсов. Как правило, ядра имеют раздельную кэш-память первого уровня, но совместно используют кэш-память второго уровня, подсистемы управления памятью и прерываниями.

В связи с этим алгоритмы должны удовлетворить следующим критериям для достижения эффективного использования мультиядерных процессоров [3]: высокой степенью гранулярности, поскольку с ядрами связаны относительно небольшие объемы локальной памяти, и асинхронностью для сокращения времени ожидания доступа к памяти.

Поэтому рассматриваемый ниже подход представим как неявный метод декомпозиции, для которого распараллеливание отражает мелкозернистость вычислений — уровень отдельных конечных элементов. На уровне отдельных конечных элементов, как будет показано, эффективно распараллеливается решение основной системы уравнений и системы для оценки погрешности, перестроение расчетной сетки выполняется также для отдельных элементов, выбранных для уточнения.

§ 1. Параллельный адаптивный конечно-элементный алгоритм

Для рассматриваемых задач с адаптивным перестроением сетки (*h-версия* МКЭ) общую схему применения методов разделения и отображения по процессорам представим следующим образом (см. **Алгоритм 1**): сеточная модель разделяется каким-либо способом по числу процессоров; формируется и решается система уравнений (СЛАУ) и оценивается погрешность решения; сеточная модель перестраивается с учетом оценки погрешности; находится дисбаланс нагрузки и, если необходимо, выполняется новое разделение сетки; вычислительная нагрузка перераспределяется в соответствии с новым разделением на процессоры.

Алгоритм 1 Параллельный алгоритм *h-версии* МКЭ

1: Строится грубая сетка $\mathcal{T}_0$ и выполняется статическая балансировка нагрузки $\mathcal{P}_0$ (**Алгоритм 5**).
2: $k = 0$
3: **while** $\eta > \eta_{\max}$ **do**
4: Вычисляется решение на сетке $\mathcal{T}_k$ на k-ом шаге перестроения. В этом случае СЛАУ решается на основе поэлементной декомпозиции (**Алгоритм 2**).
5: Оценивается относительная погрешность решения для всей области η и отдельных конечных элементов η_i. Определяется критерий перестроения на каждом конечном элементе ξ_i. Помечаются конечные элементы для перестроения (или огрубления) (**Алгоритм 3**).
6: **if** (Если помеченные элементы существуют) **then**
7: Строится сетка $\mathcal{T}_{k+1}$ локальным перестроением сетки $\mathcal{T}_k$ (**Алгоритм 4**).
8: Вычисляется дисбаланс вычислительной нагрузки $\triangle W$, находится новое разделение $\mathcal{P}_{k+1}$ и выполняется перераспределение нагрузки (**Алгоритм 5**).
9: **end if**
10: **end while**

Рассмотрим некоторые шаги параллельного адаптивного алгоритма более подробно.

1.1. Конечно-поэлементная декомпозиция алгоритма

Обычно в МКЭ считается, что конечно-элементная сетка собрана и каждый узел принадлежит нескольким элементам. Предположим обратное, множество элементов разобрано, так что любой узел принадлежит одному элементу, и нумерация степеней свободы в элементах выполнена от $1 \dots n$.

В результате получим блочно-диагональную матрицу $\mathbf{A}$, которую удобно хранить по процессорам, вводя матрицу связности

$$\mathbf{A} = \sum_{e=1}^{M} \mathbf{C}_e^T \tilde{\mathbf{A}}_e \mathbf{C}_e, \tag{1.1}$$

где $\mathbf{A} \in \mathbb{R}^{N\times N}$ — глобальная матрица жесткости; $\tilde{\mathbf{A}}_e \in \mathbb{R}^{n\times n}$ — локальная (элементная) матрица жесткости; $\mathbf{C}_e \in \mathbb{Z}^{n\times N}$ — булева матрица или матрица связности, выражающая соотношение между независимыми степенями свободы индивидуальных элементов и независимыми перемещениями для всей области. Тогда (1.1) можно записать в виде

$$\mathbf{A} \;=\; \left(\mathbf{C}_1^T \mathbf{C}_2^T \dots \mathbf{C}_M^T\right) \begin{pmatrix} \tilde{\mathbf{A}}_1 & 0 & \dots & 0 \\ 0 & \tilde{\mathbf{A}}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \tilde{\mathbf{A}}_M \end{pmatrix} \begin{pmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \\ \vdots \\ \mathbf{C}_M \end{pmatrix}.$$

Таким образом, матрицы $\tilde{\mathbf{A}}_e$ независимо хранятся на процессорах. В случае перестроения системы уравнений, связанном с адаптацией сетки или локальным изменением степени аппроксимирующих полиномов, корректируются только матрицы перестраиваемых элементов и списки глобальной связанности.

Пусть область Ω разбита на n_p-подобластей $\Omega = \Omega^{(1)} \cup \Omega^{(2)} \cup \ldots \cup \Omega^{(n_p)}$, где $\Omega^{(i)} \cap \Omega^{(j)} = \varnothing$, $(i \neq j; \quad i,j = 1, n_p)$. Тогда подобласть $\Omega^{(i)}$ содержит по N_i узлов и M_i элементов, причем $M = \sum_{i=1}^{n_p} M_i$, $N < \sum_i^{n_p} N_i$. Введем следующее упорядочивание матрицы жесткости системы

$$\mathbf{A} = \sum_{i=1}^{n_p} \mathbf{C}^{T(i)} \mathbf{A}^{(i)} \mathbf{C}^{(i)}, \quad \mathbf{A}^{(i)} = \sum_{e=1}^{M_i} \mathbf{C}_e^{T(i)} \tilde{\mathbf{A}}_e^{(i)} \mathbf{C}_e^{(i)}.$$

На каждом процессоре будем хранить M_i элементных матриц жесткости $\tilde{\mathbf{A}}_e$ и векторов $\tilde{\mathbf{b}}_e$.

Обозначим через $V = \{v_i | i = 1, N\}$ множество вершин в конечно-элементной сетке и $\mathcal{T} = \{t_i | i = 1, M\}$ множество конечных элементов. Также введем $G = (V, E)$ граф, связанный с сеткой, где множество ребер $E = \{e_{i,j} = (v_i, v_j) | v_i, v_j \in t_k\}$, и дуальный граф $D = (\mathcal{T}, F)$, связанный с сеткой, в котором $F = \{(t_i, t_j) | f_{i,j} \in t_i, t_j\}$.

Определим n_p-разделение для дуального графа $D = (\mathcal{T}, F)$ как отображение $\mathcal{P}\colon \mathcal{T} \to [1 \dots n_p]$ множества вершин, т.е. конечных элементов на n_p подмножеств $\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \dots, \mathcal{T}^{(n_p)}$, где $\cup_{i=1}^{n_p} \mathcal{T}^{(i)} = \mathcal{T}$ и $\mathcal{T}^{(i)} \cap \mathcal{T}^{(j)} = \varnothing$, если $i \neq j$ (более детально см. раздел 1.4).

Таким образом, в основе поэлементной декомпозиции лежит преобразование связности, базирующееся на топологических операциях обьединения и разъединения.

1.1.1. Структура данных для матрично-векторных операций

В методе конечных элементов генерируется множество плотных матриц жесткости элемента. Матрицы жесткости формируются и хранятся различными способами: поэлементно, поребёрно и в глобальной матрице жесткости без нулевых элементов. Введение той или иной схемы вычисления матрично-векторного и скалярного произведения в итерационных методах решения систем линейных уравнений позволяет выбрать уровень распараллеливания при одном и том же разделении области $\mathcal{P}$. Рассмотрим вариант поэлементной декомпозиции алгоритма [4].

Определение 1. Глобальные переменные вектора $\mathbf{u}$ или матрицы $\mathbf{A}$ в локально распределенном формате обозначим как $\tilde{\mathbf{u}}^{(i)}$ и $\tilde{\mathbf{A}}^{(i)}$, если они содержат только локальные данные.

Определение 2. Глобальные переменные вектора $\mathbf{u}$ или матрицы $\mathbf{A}$ в глобально распределенном формате обозначим как $\bar{\mathbf{u}}^{(i)}$ и $\bar{\mathbf{A}}^{(i)}$, если они содержат глобальные данные, в том числе вклады из соседних подобластей.

Таким образом, глобальный вектор $\mathbf{u}$ может быть представлен через глобально распределенные вектора $\bar{\mathbf{u}}^{(i)}$ или локально распределенные вектора $\tilde{\mathbf{u}}^{(i)}$.

Обозначим $\mathbf{C}^{(i)} \in \mathbb{Z}^{N_i \times N}$ булеву матрицу или матрицу связности, выражающую отображение глобального вектора из подобласти i. Тогда

$$\bar{\mathbf{u}}^{(i)} \equiv \mathbf{C}^{(i)}\mathbf{u}, \qquad \mathbf{u} \equiv \sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\tilde{\mathbf{u}}^{(i)}.$$

Локально распределенные вектора $\tilde{\mathbf{u}}^{(i)}$ могут быть преобразованы в глобально распределенные вектора $\bar{\mathbf{u}}^{(i)}$ как

$$\bar{\mathbf{u}}^{(i)} \equiv \mathbf{C}^{(i)} \sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\tilde{\mathbf{u}}^{(i)}.$$

Данная операция может быть эффективно выполнена с ограниченным объемом по коммуникациям с соседними подобластями и представлена как $\bar{\mathbf{u}}^{(i)} \equiv \sum_{\leftrightarrows}^{\partial\Omega^{(i)}} \tilde{\mathbf{u}}^{(i)}$. Аналогично векторам, глобальная матрица жесткости $\mathbf{A}$ может быть представлена через множество глобально распределенных подматриц $\bar{\mathbf{A}}^{(i)}$ или локально распределенных подматриц $\tilde{\mathbf{A}}^{(i)}$

$$\mathbf{A} = \sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\tilde{\mathbf{A}}^{(i)}\mathbf{C}^{(i)}, \qquad \bar{\mathbf{A}}^{(i)} = \sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\mathbf{A}\mathbf{C}^{(i)}. \tag{1.2}$$

1.1.2. Операции произведения векторов, матриц и предобуславливания

Рассмотрим основные операции, необходимые для параллельной реализации предобусловленного метода сопряженных градиентов. Скалярное произведение векторов $(\mathbf{u}, \mathbf{v})$ для $\mathbf{u}, \mathbf{v} \in \mathbb{R}^N$ можно вычислить как

$$(\mathbf{u}, \mathbf{v}) = \left(\sum_{i=1}^{n_p} \mathbf{C}^{(i)}\tilde{\mathbf{u}}^{(i)}, \sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\tilde{\mathbf{v}}^{(i)} \right) = \sum_{i=1}^{n_p} \tilde{\mathbf{u}}^{(i)^T}\mathbf{C}^{T^{(i)}} \sum_{i=1}^{n_p} \mathbf{C}^{(i)}\tilde{\mathbf{v}}^{(i)} = \sum_{i=1}^{n_p} \left(\tilde{\mathbf{u}}^{(i)}, \bar{\mathbf{v}}^{(i)} \right). \tag{1.3}$$

Если оба вектора хранятся в локально распределенном формате, то скалярное произведение (1.3) может быть вычислено в следующей последовательности операций

$$\bar{\mathbf{v}}^{(i)} = \sum_{\leftrightarrows}^{\partial\Omega^{(i)}} \tilde{\mathbf{v}}^{(i)}; \quad \gamma^{(i)} = (\tilde{\mathbf{u}}^{(i)}, \bar{\mathbf{v}}^{(i)}), \quad \gamma = \sum_{i=1}^{n_p} \gamma^{(i)}.$$

Матрично-векторное произведение $\mathbf{v} \leftarrow \mathbf{A}\mathbf{u}$, где $\mathbf{A} \in \mathbb{R}^{N \times N}$, $\mathbf{u}, \mathbf{v} \in \mathbb{R}^N$ будем вычислять как

$$\left(\sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\tilde{\mathbf{A}}^{(i)}\mathbf{C}^{(i)} \right) \left(\sum_{i=1}^{n_p} \bar{\mathbf{u}}^{(i)} \right) = \sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\tilde{\mathbf{A}}^{(i)}\tilde{\mathbf{u}}^{(i)}. \tag{1.4}$$

Будем вычислять произведения матрицы на вектор как $\bar{\mathbf{v}}^{(i)} = \sum_{i=1}^{n_p} \tilde{\mathbf{A}}^{(i)}\bar{\mathbf{u}}^{(i)}$ с последующим выполнением операции $\mathbf{v} \equiv \sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\bar{\mathbf{v}}^{(i)}$.

Применение предобуславливателя рассмотрим на примере, когда матрица предобуславливателя $\mathcal{M}$ хранится в локально распределенном формате $\tilde{\mathcal{M}}^{(i)}$, т.е. $\mathcal{M} = \sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\tilde{\mathcal{M}}^{(i)}\mathbf{C}^{(i)}$. Операция предобуславливания может быть записана следующим образом:

$$\mathbf{s} = \mathcal{M}\mathbf{u} = \left(\sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\tilde{\mathcal{M}}^{(i)}\mathbf{C}^{(i)} \right) \left(\sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\tilde{\mathbf{u}}^{(i)} \right).$$

Данное выражение может упрощено:

$$\mathbf{s}^{(i)} = \tilde{\mathcal{M}}^{(i)}\mathbf{C}^{(i)} \sum_{i=1}^{n_p} \mathbf{C}^{T^{(i)}}\tilde{\mathbf{u}}^{(i)} = \tilde{\mathcal{M}}^{(i)}\bar{\mathbf{u}}^{(i)}. \tag{1.5}$$

В рассматриваемом случае полагалось, что предобуславливатель доступен на каждом вычислительном процессоре и операций обмена между ними не требуется.

Конечно, эффективные предобуславливатели могут быть построены, используя методы типа последовательной верхней релаксации, неполной факторизации, полиномиального разложения или многосеточных методов [5]. Однако многие из этих подходов требуют конструирования глобальных предобуславливателей, которые предполагают большие запросы оперативной памяти. Также они не полностью используют преимущества поэлементной декомпозиции.

Более сложные элементные предобуславливатели были построены на основе неполной факторизации элементных матриц, разложения Краута и др. [6]. Вычислительные эксперименты показали, что время выполнения одной итерации в предобусловленном МСГ с этими предобуславливателями значительно возрастает, и эффективность параллельного предобусловленного МСГ меньше, чем не предобусловленного.

1.1.3. Распараллеливание решения системы уравнений

Реализация проекционных методов на подпространствах Крылова [7, 8], например алгоритма сопряженных градиентов без непосредственной сборки глобальной матрицы системы линейных алгебраических уравнений и соотношений (1.2)–(1.5), позволяет эффективно реализовать метод решения на параллельной вычислительной системе как с распределенной, так и с общей памятью (см. **Алгоритм 2**).

Алгоритм 2 Параллельный предобусловленный метод сопряженных градиентов

Задано: $\tilde{\mathbf{r}}_0^{(i)} = \tilde{\mathbf{b}}^{(i)} - \tilde{\mathbf{A}}^{(i)}\bar{\mathbf{u}}_0^{(i)}$; $\bar{\mathbf{r}}_0^{(i)} = \sum_{\leftrightarrows}^{\partial\Omega^{(i)}} \tilde{\mathbf{r}}_0^{(i)}$; $\tilde{\mathbf{s}}_0^{(i)} = \tilde{\mathcal{M}}^{(i)}\bar{\mathbf{r}}_0^{(i)}$; $\bar{\mathbf{s}}_0^{(i)} = \sum_{\leftrightarrows}^{\partial\Omega^{(i)}} \tilde{\mathbf{s}}_0^{(i)}$;
$\bar{\mathbf{p}}_0^{(i)} = \bar{\mathbf{s}}_0^{(i)}$; $\rho_0 = \sum_{i=1}^{n_p}(\bar{\mathbf{s}}_0^{(i)}, \tilde{\mathbf{r}}_0^{(i)})$;

1: **for** $l = 0, 1, \ldots$ **do**
2: $\tilde{\mathbf{q}}_l^{(i)} := \tilde{\mathbf{A}}^{(i)}\bar{\mathbf{p}}_l^{(i)}$; {Матрично-векторное произведение}
3: $\gamma_l := \sum_{i=1}^{n_p}(\bar{\mathbf{p}}_l^{(i)}, \tilde{\mathbf{q}}_l^{(i)})$;
4: $\alpha_l := -\rho_l/\gamma_l$;
5: $\bar{\mathbf{u}}_{l+1}^{(i)} := \bar{\mathbf{u}}_l^{(i)} - \alpha_k\bar{\mathbf{p}}_l^{(i)}$; $\tilde{\mathbf{r}}_{l+1}^{(i)} := \tilde{\mathbf{r}}_l^{(i)} + \alpha_l\tilde{\mathbf{q}}_l^{(i)}$;
6: $\bar{\mathbf{r}}_{l+1}^{(i)} = \sum_{\leftrightarrows}^{\partial\Omega^{(i)}} \tilde{\mathbf{r}}_{l+1}^{(i)}$
7: **if** $(||\mathbf{r}_{l+1}||_2/||\mathbf{r}_0||_2 < \varepsilon)$ **then**
8: **return**
9: **end if**
10: $\tilde{\mathbf{s}}_{l+1}^{(i)} := \tilde{\mathcal{M}}^{(i)}\bar{\mathbf{r}}^{(i)}$;
11: $\bar{\mathbf{s}}_{l+1}^{(i)} := \sum_{\leftrightarrows}^{\partial\Omega^{(i)}} \tilde{\mathbf{s}}_{l+1}^{(i)}$;
12: $\rho_{l+1} := \sum_{i=1}^{n_p}(\bar{\mathbf{s}}_{l+1}^{(i)}, \tilde{\mathbf{r}}_{l+1}^{(i)})$;
13: $\beta_l := \rho_{l+1}/\rho_l$; $\bar{\mathbf{p}}_{l+1}^{(i)} := \bar{\mathbf{s}}_{l+1}^{(i)} + \beta_k\bar{\mathbf{p}}_l^{(i)}$;
14: $l = l + 1$;
15: **end for**

Главные преимущества поэлементной схемы заключаются в следующем: может быть использована с любыми типами элементов; простота представления и отображения на МВС. Элементные матрицы могут быть вычислены заново без изменения всей матрицы. Основной недостаток — требуется использование эффективных поэлементных предобуславливателей.

Согласно принятому разделению, подобласть, находящаяся на процессоре, содержит блоки матриц и векторов для внутренних и граничных узлов. Пусть подобласть $\Omega^{(i)}$ связана с конечно-элементной подсеткой $\mathcal{T}^{(i)}$, причем элементы внутри подобласти пронумерованы $e = 1, M_I^{(i)}$, а элементы лежащие на границах подобластей имеют номера $e = M_I^{(i)} + 1, M_B^{(i)}$. Сгруппировав таким образом неизвестные в подобласти, можно выполнять операции матрично-векторного и скалярного произведения в двух процессах для внутренних и граничных узлов.

Тогда, совместив глобальные коммуникации, связанные с вычислением с теми же операциями для внутренних узлов, не требующих обменов, можно записать

$$\mathbf{Ap} = \sum_{i=1}^{n_p} \sum_{e=1}^{M^i} C_e^{T^{(i)}} \mathbf{A}_e^{(i)} C_e^{(i)} \mathbf{p} = \sum_{i=1}^{n_p} \sum_{e=1}^{M_I^i} \mathbf{C}_e^{T^{(i)}} \mathbf{A}_e^{(i)} \mathbf{C}_e^{(i)} \mathbf{p} + \sum_{i=1}^{n_p} \sum_{e=M_I^i+1}^{M_B^i} \mathbf{C}_e^{T^{(i)}} \mathbf{A}_e^{(i)} \mathbf{C}_e^{(i)} \mathbf{p}.$$

Совмещение вычислений и обменов в совокупности с уменьшением числа обменов позволило уменьшить время выполнения **Алгоритма 2** на 15–20 % .

Приведенный алгоритм сравнивался со схемами, использующими другие типы представления элементных векторов и матриц, и показал более высокие характеристики ускорения и эффективности. Кроме того, адаптированный вариант поэлементной декомпозиции был использован и для разрывного метода Галёркина [9].

1.2. Оценка погрешности и критерий адаптации

Априорные теоретические оценки погрешности МКЭ практически малопригодны для контроля достоверности полученных решений. Поэтому особый интерес представляют удобные для численной реализации достаточно надежные апостериорные оценки, не требующие к тому же слишком больших вычислительных затрат (см. **Алгоритм 3**). С вычислительной точки зрения получение оценки погрешности $||\mathbf{E}||$ приводит к решению дополнительной задачи (во многих случаях к системе алгебраических уравнений) на всей области решения или только ее части. Важно, чтобы процесс получения оценки погрешности можно было распараллелить и сократить общие вычислительные затраты адаптивного решения задачи. Существует много

Алгоритм 3 Оценка погрешности и критерий перестроения

1: Оценивается погрешность решения на каждом конечном элементе $||\mathbf{E}||_e$ и на всей сетке $||\mathbf{E}||$.
2: Вычисляется критерий перестроения $\xi_e = ||\mathbf{E}||_e/(\eta_{max}||\mathbf{u}||/\sqrt{M})$ на каждом конечном элементе t_e в $\mathcal{T}$.
3: **if** $\xi_e > \xi$ **then**
4: Добавляется элемент t_e в Q — множество элементов помеченных для перестроения в $\mathcal{T}$.
5: **end if**

подходов к апостериорной оценке погрешности решения для тех или иных задач, например в работах [10, 11] (см. также приведенную в них библиографию). Традиционно используемые адаптивные стратегии для многих задач основаны на использовании энергетической нормы ошибки и критерия оптимальности сетки, исходя из равного распределения ошибки между всеми элементами.

Условие адаптации формулируется следующим образом. Конечный элемент перестраивается, когда критерий перестроения превышает допустимую величину ξ. Часто используемый в расчетах критерий основан на идее равнораспределения погрешности по всем конечным элементам. Считается, что сетка будет оптимальной в том случае, когда погрешность распределяется равномерно по всем элементам.

Пусть относительная погрешность определяется как отношение нормы погрешности к норме точного решения

$$\eta = \frac{\|\mathbf{E}\|}{\|\mathbf{u}\|} \leqslant \eta_{\max},$$

где норма точного решения $\|\mathbf{u}\| = \sqrt{\|\mathbf{u}^h\|^2 + \|\mathbf{E}\|^2}$, где $\mathbf{u}^h$ — конечно-элементное решение. Тогда критерий перестроения для всей области можно записать как $\xi = \|\mathbf{E}\|/\eta_{\max}\|\mathbf{u}\|$, где $\xi \leqslant 1$ — однородное перестроение всей рассматриваемой области. Для перестроения в локальных областях или отдельных конечных элементах построим элементный критерий адаптации. Пусть задана величина допустимой элементной оценки погрешности, например $\|\mathbf{E}\|_e^* =$

$\|\mathbf{E}\|/\sqrt{M}$, где $\|\mathbf{E}\|$ — оценка погрешности в области; M — число конечных элементов в сеточной области. Введем критерий перестроения $\bar{\xi}_e = \|\mathbf{E}\|_e/\|\mathbf{E}\|_e^*$, а для всей области запишем как

$$\xi_e = \bar{\xi}_e \xi = \frac{\|\mathbf{E}\|\|\mathbf{E}\|_e}{\eta_{\max}\|\mathbf{u}\|\|\mathbf{E}\|_e^*} = \frac{\|\mathbf{E}\|_e}{\eta_{\max}\|\mathbf{u}\|/\sqrt{M}},$$

или с учетом введенных соотношений будем иметь

$$\xi_e = \frac{\|\mathbf{E}\|_e}{\eta_{\max}(\sqrt{\|\mathbf{u}^h\|^2 + \|\mathbf{E}\|^2})/\sqrt{M}},$$

здесь $\|\mathbf{u}^h\|$ — энергетическая норма конечно-элементного решения; $\|\mathbf{E}\|$ — норма погрешности решения.

Как показали численные эксперименты при любых других критериях перестроения получаются сетки с большим числом узлов для той же самой глобальной энергетической нормы погрешности.

1.3. Параллельное построение и перестроение сетки

Построение сетки является первым этапом адаптивного решения задачи (см. **Алгоритм 1**), во многом определяющим дальнейшие вычисления. При параллельной реализации метода декомпозиции этот этап необходимо рассматривать, с одной стороны, как первоначальное распределение данных и вычислительной нагрузки процессоров, а с другой, — как основу для алгоритмов, построенных на распределенных сетках, где появляются шаги, связанные с модификацией сетки (перестроение, динамическая балансировка нагрузки и пр.). Таким образом, актуальность параллельной реализации построения начальной грубой сетки обусловлена: возможностью распределения затрат оперативной памяти, особенно для больших 3D неструктурированных сеток; сокращением затрат на передачу данных между последовательным генератором сетки и параллельным приложением (с появлением эффективных параллельных алгоритмов решения СЛАУ с предобуславливанием максимальные затраты алгоритма МКЭ смещаются к этапам построения, распределения, разделения и перестроения сетки) [12].

Локальные методы перестроения позволяют получить сетки с разной плотностью узлов и построить адаптивный процесс для решения задачи. Деление отдельных элементов сетки изменяет характерный размер ячеек расчетной сетки в соответствии с локальными критериями перестроения, основанными на апостериорной оценке погрешности решения (см. **Алгоритм 3**).

Далее будем полагать, что задан список элементов для перестроения и все операции перестроения будут выполняться на уровне отдельных конечных элементов. Отметим только некоторые свойства локального перестроения при делении элементов: построение конформной сетки выполняется практически автоматически с небольшим разрастанием области перестроения. Некоторые теоретические оценки показывают, что построение конформной триангуляции получается за конечное число шагов согласования; в перестроении используется минимальный набор шаблонов; история перестроения может быть сохранена или быстро восстановлена и использована для обратного перестроения. Перестроение грубой сетки ограничено начальной триангуляцией; шаг сетки явно не задается, но может быть получен из оценки погрешности и в дальнейшем использован при локальном делении элементов. Кроме того, при делении конечных элементов углы в триангуляции ограничены.

Достаточно много последовательных алгоритмов перестроения треугольных и тетраэдральных сеток было предложено в [13]. Одним из часто используемых является алгоритм разделения по наибольшему ребру в элементе, предложенный М. С. Ривarой [14]. Подобные алгоритмы являются рекурсивными. В двумерном случае каждый треугольник t_i из множества Q разделяется на два треугольника ребром, соединяющим новую вершину v_{N+1} с вершиной противолежащей наибольшей стороне. Новая вершина помещается в середину наибольшего ребра t_i. После деления t_i смежный с ним по наибольшему ребру конечный элемент t_j становится несогласованным и должен быть перестроен в дальнейшем. В работах М. С. Ривары

предлагается сразу перестраивать t_j или отслеживать последовательность таких перестроений. Такие алгоритмы не так эффективны в параллельной реализации, когда $t_i \in \mathcal{T}^{(i)}$, а $t_j \in \mathcal{T}^{(j)}$ требуется обмен данными между подобластями. Поэтому для несогласованных элементов было введено множество $R = \cup_{i=1}^{n_p} R^{(i)}$, которое позволило объединить передаваемые данные и минимизировать число обменов. Кроме того, в рассматриваемом далее алгоритме новая вершина помещалась как на середину ребра, так при необходимости и на $\partial\Omega^{(i)} \in \partial\Omega$, чем обеспечивалась адаптация сетки $\mathcal{T}_{k+1}$ к геометрии области Ω.

Рассмотрим **Алгоритм 4** параллельного перестроения, основанный на дуальном графе конечно-элементной сетки и его разделении $\mathcal{T} = \cup_{i=1}^{n_p} \mathcal{T}^{(i)}$ на непересекающиеся подмножества. Каждый процессор i хранит множество конечных элементов $\mathcal{T}^{(i)}$ и множество помеченных элементов для перестроения $Q = \cup_{i=1}^{n_p} Q^{(i)}$ и выполняет следующий алгоритм.

Алгоритм 4 Параллельное перестроение сетки

1: $R = 0$
2: **while** $(R \cup Q) \neq \varnothing$ **do**
3: **while** $(R^{(i)} \cup Q^{(i)}) \neq \varnothing$ **do**
4: Разбиваются все элементы t_k из $R^{(i)} \cup Q^{(i)}$ по наибольшей стороне и затем исключаются из рассмотрения.
5: Пополняется $R^{(i)}$ элементами из $\mathcal{T}^{(i)}$, смежными с t_k по разделенному ребру.
6: **end while**
7: Пополняется $R^{(i)}$ элементами из $\mathcal{T}^{(i)}$, содержащими ребра, разделенные из соседней подобласти.
8: **end while**

Алгоритмы последовательного и параллельного адаптивного перестроения отличаются этапом согласования частей сетки $\mathcal{T}^{(i)}$, который требует межпроцессорных обменов. Параллельное перестроение выполняется над частями сетки, что приводит к появлению множества несогласованных ребер в смежных частях сетки $R^{(i)}$. Для обеспечения согласованности всех подобластей сетки в согласующее перестроение вводится дополнительный шаг — перестроение несогласованных треугольников, образовавшихся в результате деления конечных элементов в смежных подобластях.

Отметим, что в **Алгоритме 4** могут применяться различные методы перестроения: регулярное деление с согласующим соединением с противолежащей вершиной; деление по наибольшему ребру и согласующее перестроение — соединение с противолежащей вершиной [15]. Это связано с тем, что алгоритм параллельного перестроения выполняется над сеткой, распределенной без перекрытия подобластей, т.е. подобласти не имеют общих конечных элементов и не происходит нарушения согласованности сетки, связанного с применением комбинированных алгоритмов деления.

Программная реализация параллельного перестроения предполагает глобальную нумерацию сеточных объектов.

1.4. Динамическая балансировка вычислительной нагрузки

Специфические особенности адаптивных алгоритмов определяют алгоритмический дисбаланс вычислительной нагрузки, приходящийся на один процессор/ядро. Локальное перестроение сеточной модели с увеличением числа конечных элементов или повышениями порядка аппроксимирующих функций приводит к разбалансировке вычислительной нагрузки, для устранения которой требуется динамически перераспределять сеточные данные между процессорами.

В *h-версии* МКЭ после каждого шага или нескольких шагов перестроения сетки — итерационного уточнения необходимо производить балансировку загрузки. В связи с этим условия и обеспечение сбалансированности загрузки процессоров является наиболее важным моментом в параллельных методах декомпозиции для задач с адаптивным уточнением.

Для большинства задач вычислительной механики естественными объектами для балансировки нагрузки являются компоненты вычислительной сетки: конечные элементы или объемы; узлы; грани и ребра. Геометрические и топологические свойства сеток гарантируют, что хорошее разделение для них существуют. Расчетным сеткам можно поставить в соответствие несколько видов графов [16, 17]. Выбор графа для разделения расчетной сетки определяется методами декомпозиции алгоритма, рассмотренными выше. Наиболее употребительными являются графы узлов и элементов.

Разделение в сеточных задачах можно определить в терминах нескольких критериев: равнораспределения вычислительной нагрузки (подобласти должны иметь почти равное число элементов или узлов); минимальных длин границ (минимальное число общих граней или узлов между подобластями); минимальной связности подобластей (минимальное число соседних подобластей); минимальной ширины ленты матрицы жесткости, принадлежащей каждому процессору и оптимальной обусловленности матриц подобластей (локальные матрицы должны быть хорошо обусловлены); в ряде случаев топология и размеры подобластей должны соответствовать возможностям сеточных генераторов.

В сеточных задачах вычислительная нагрузка на процессор складывается из суммы нагрузок, связанных с каждым объектом сетки, так, в МКЭ это конечный элемент. Каждому конечному элементу — вершине графа — соответствуют своя вычислительная нагрузка. Ребро графа, соединяющее два элемента, соответствует связи и определяет коммуникационную нагрузку. Тогда процесс балансировки нагрузки можно представить как нахождение разделения дуального графа сетки $D(\mathcal{T}, F, W)$ со множеством вершин $\mathcal{T} = \{t_i | i = 1, 2, \ldots, M\}$, каждая из которых связана ребром $F = \{f_{i,j} | \{t_i, t_j\}\}$. Вершины и ребра графа могут быть заданы с весами, что необходимо при рассмотрении разделения сеточных моделей, в которых нагрузка на элементах может быть не равномерной, например для *p-версии* МКЭ: $W_{\mathcal{T}} = \{w_t(t_i) \in \mathbb{R} | t_i \in \mathcal{T}\}$ — вес для вершин(конечных элементов) и $W_F = \{w_f(f_{i,j}) \in \mathbb{R} | f_{i,j} \in F)$ — соответственно вес для ребер.

Задача нахождения бикритериального разделения $\mathcal{P}$ графа $D(\mathcal{T}, F, W)$ с весами $W_{\mathcal{T}}$ и W_F на n_p подобластей $(\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \ldots \mathcal{T}^{(n_p)})$ заключается в следующем:

$$\mathcal{T} = \cup_{i=1}^{n_p} \mathcal{T}^{(i)}, \quad \mathcal{T}^{(i)} \cap \mathcal{T}^{(j)} = \varnothing \quad \text{при} \quad \forall \quad i \neq j, \tag{1.6}$$

или с учетом того, что $\sum\limits_{t_i \in \mathcal{T}^{(j)}} w_t(t_i) = \sum\limits_{t_k \in \mathcal{T}^{(l)}} w_t(t_k) \quad \forall j \neq l \quad j, l \in 1, 2, \ldots, n_p$ и число получаемых общих ребер или сумма весов всех ребер, принадлежащих границе между подобластями, должна быть минимальна

$$\min |E_c| = \{f_{i,j} \in f | t_i \in \mathcal{T}^{(p)}, t_j \in \mathcal{T}^{(q)} \quad \text{при} \quad p \neq q\}. \tag{1.7}$$

Обзор существующих алгоритмов создания разделения $\mathcal{P}$ и перераспределения $\mathcal{P}_k \to \mathcal{P}_{k+1}$ можно найти в [17].

Алгоритм 5 Разделение и перераспределение нагрузки

1: Определяются вычислительные затраты $W^{(i)}$ на каждой $\mathcal{T}^{(i)}$.
2: Находится величина разбалансированности $\triangle W$.
3: **if** $\triangle W > W^*$ **then**
4: Строится новое разделение $\mathcal{P}_{k+1}$.
5: Перераспределяется вычислительная нагрузка $W^{(i)}$ в соответствии с новым разделением $\mathcal{P}_{k+1}$.
6: **else**
7: Имеющееся разделение $\mathcal{P}_k$ сохраняется.
8: **end if**

Алгоритм разделения и распределения вычислительных затрат W адаптивного алгоритма решения на основе элементной декомпозиции с использованием дуального графа конечно-элементной сетке запишем виде **Алгоритма 5**.

Рис. 1. Грубая сетка $N = 3127$, $M = 5970$ с разделением на $n_p = 96$ подобластей. Исходная погрешность решения $\eta = 12.7\%$

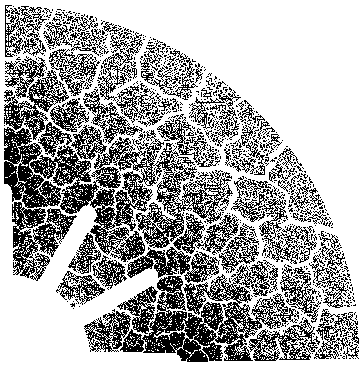

Рис. 2. Адаптивная сетка после $k = 3$ шагов перестроения ($N = 34900$, $M = 69009$, $\eta = 2.7\%$)

§ 2. Численный пример

Рассмотрим применение адаптивного конечного-элементного метода (**Алгоритм 1**) с линейной аппроксимацией перемещений $\mathbf{u}^h = \sum_{i=1}^{N} \psi_i u_i$ на треугольных элементах для решения двумерной задачи теории упругости, результирующая система уравнений которой имеет вид $\mathbf{Au} = \mathbf{b}$, где $\mathbf{u}$ — вектор неизвестных перемещений; $\mathbf{b}$ — вектор приложенных узловых сил и $\mathbf{A}$ — матрица жесткости, имеющая следующий вид $\mathbf{A} = \sum_{e=1}^{M} \int_{\Omega_e} \mathbf{B}_e^T \mathbf{D} \mathbf{B}_e \, d\Omega_e$, здесь $\mathbf{B}_e$ — матрица производных функций формы $\psi_i \cdots \psi_N$; $\mathbf{D}$ — матрица упругих характеристик.

Исходная грубая сетка $\mathcal{T}_0$ содержала $N = 3127$ узлов и $M = 5970$ элементов (рис. 1). Погрешность решения на данной сетке составила $\eta = 12.7\%$. Число шагов перестроения сетки равнялось $k = 6$. Погрешность решения на перестроенной сетке $\mathcal{T}_6$ ($N = 322\,130$, $M = 641\,726$) достигла $\eta = 2.7\%$. Пример перестроенной сетки после $k = 3$ шагов перестроения и разделенной на $n_p = 96$ подобластей приведен на рис. 2. Начальная декомпозиция на неперекрывающиеся подобласти выполнялась алгоритмом, основанным на геометрической информации. После каждого шага перестроения вычислялось новое разделение вычислительной нагрузки W по диффузионному алгоритму для каждой i-подобласти

$$W_{k+1}^{(i)} = W_k^{(i)} + \sum_j \alpha_{ij}(W_k^{(i)} - W_k^{(j)}) + \theta_{k+1}^{(i)} - \vartheta_{k+1}^{(i)},$$

где $W_{k+1}^{(i)}$ — вычислительная нагрузка на $k+1$ шаге перестроения; α_{ij} — диффузионный параметр определяющий обмен между процессорами; $\theta_{k+1}^{(i)}$ — появившееся нагрузка на шаге перестроения $k+1$; $\vartheta_{k+1}^{(i)}$ — нагрузка исчезающая к шагу $k+1$.

Для оценки погрешности и вычисления критерия перестроения при решении уравнений теории упругости применялся подход, основанный на теории сопряженной аппроксимации напряжений [18]. Применение теории сопряженной аппроксимации свелось к трем системам алгебраических уравнений для компонент напряжений. Порядок систем совпадает с порядком, используемым для нахождения значений искомых перемещений [19, 20]. Полученные системы уравнений формируются и решаются на основе поэлементной декомпозиции. Отметим, что в этом случае последовательность операций **Алгоритма 2** выполнялась одновременно над тремя системами. Таким образом, были совмещены коммуникационные операции всех систем, необходимые при вычислении матрично-векторного произведения, скалярных произведений и предобуславливания.

Расчёты по **Алгоритму 1** проводились на вычислительных системах, имеющих разную

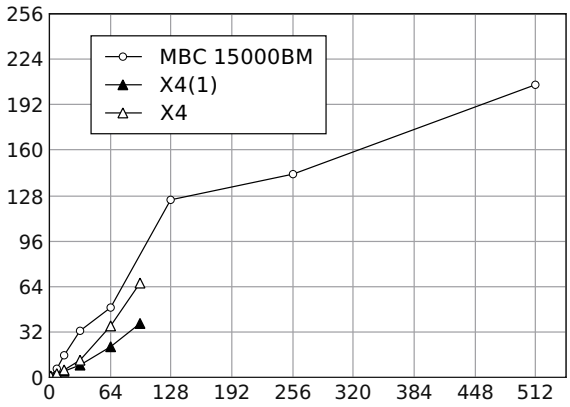

Рис. 3. Сравнение ускорение ($S = T_1(\cdot)/T_{n_p}(\cdot)$) параллельного адаптивного алгоритма на одноядерных и мультиядерных процессорах

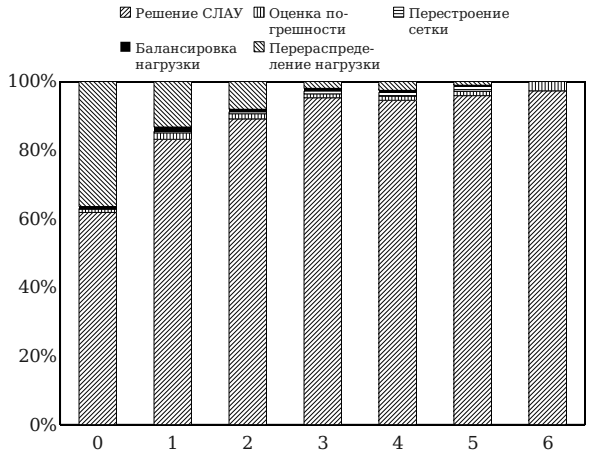

Рис. 4. Вклады в вычислительные затраты по шагам перестроения

архитектуру: МВС-15000ВМ (МСЦ РАН) — система, состоящая из 574-х вычислительных узлов с одноядерными процессорами IBM PowerPC (процессоров 2 x IBM PowerPC 970FX 2.2 ГГц/L2=1MB; 4 Гб PC-2700), объединённых сетью Myrinet; X4 (ИПМ УрО РАН) — система, состоящая из 12-ти вычислительных узлов (2 x Intel Xeon E5430 «Harpertown» 2.66 ГГц/L2=12Мб; 8 Гб оперативной памяти PC 5300), объединённых сетью Gigabit; X6 (ИПМ УрО РАН) — система, состоящая из двух вычислительных узлов с процессорами AMD Opteron, объединённых сетью Gigabit: AMD1 — вычислительный узел из 4 четырехядерных процессоров (4 x CPU AMD Opteron 8380 «Barcelona» 2.5 ГГц/L2+L3=8Мб; 64 Гб PC2-6400) и AMD2 — вычислительный узел из 4 шестиядерных процессоров (4 x CPU AMD Opteron 8435 «Istanbul» 2.6 ГГц/L2+L3=9Мб; 64 Гб PC2-6400).

В системах X4 и X6 используется сеть Gigabit Ethernet, пропускная способность которой в два раза меньше чем Myrinet, а латентность — существенно больше. Тактовая частота рассматриваемых многоядерных процессоров только на 20% больше, чем у одноядерных PowerPC, что позволило сравнить результаты на системах X4 и X6 с полученными на МВС-15000ВМ [21]. Однако надо отметить, что в мультиядерных процессорах существует дисбаланс между производительностью ядер и пропускной способностью шины. Например, для процессора Intel «Harpertown», имеющего четыре ядра, каждое из которых имеет производительность 10.64 Гфлопс/сек., пропускная способность шины — 10.64 ГБ/сек. Как следствие, для загрузки шины достаточно только одного ядра процессора, а использование остальных ядер не обеспечивает вычисления в четыре раза быстрее. Как показали эксперименты, **Алгоритм 1** выполнялся на всех ядрах системы X4 только в 1.75 раза быстрее, чем при использовании одного ядра на каждом процессоре.

Интересно сравнение результатов, полученных на МВС-15000ВМ (двухсокетовые платы с одноядерными процессорами) и на мультиядерных МВС. На МВС-15000ВМ было достигнуто высокое параллельное ускорение $S = T_1(\cdot)/T_{n_p}(\cdot)$ поэлементной декомпозиции (см. рис. 3). Однако при числе процессоров $n_p > 128$ эффективность начинала снижаться вследствие сильной недогруженности вычислительных процессоров. Зависимости ускорения от числа процессоров/ядер приведены для запуска одной и той же программной реализации адаптивного алгоритма на одноядерной (МВС-15000ВМ) и мультиядерной системе X4. Наблюдается существенное снижение ускорения $S = T_1(X4(1))/T_{n_p}(X4(1))$ и эффективности 2–4 раза при запуске приложения на X4 (на графике обозначено X4(1)). Тем не менее, за счет сокращения числа межузловых обменов время выполнения на мультиядерной системе X4 уменьшилось, и в случае $n_p = 64$ значение $T_{64}(MBC)/T_{64}(X4(1)) = 1.26$, а при $n_p = 96$ составило

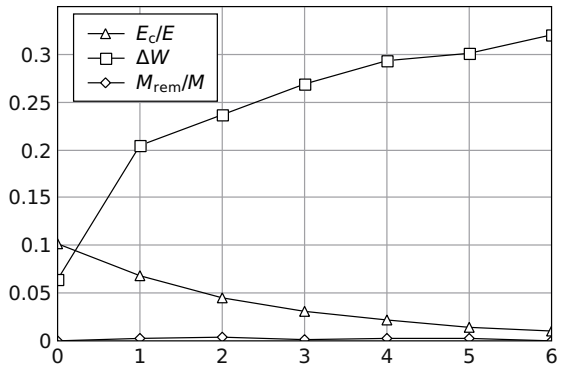

Рис. 5. Зависимости числа пересылаемых элементов, числа разрезанных ребер, разбалансировки по шагам перестроения

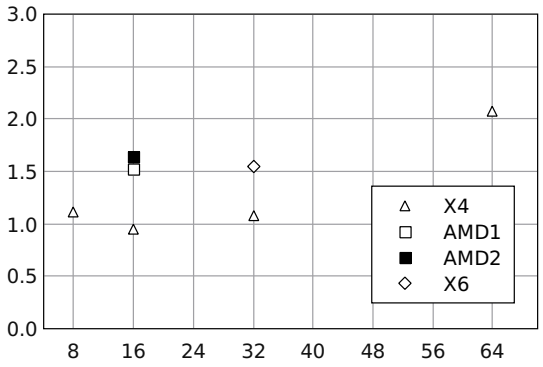

Рис. 6. Сравнение временных затрат ($T_{n_p}(MBC)/T_{n_p}(\cdot)$) параллельного адаптивного алгоритма для многосокетовых мультиядерных узлов при различном числе параллельных процессов

$T_{64}(MBC)/T_{96}(X4(1)) = 2.2$, где MBC — МВС-15000ВМ.

Представленные результаты получены на системе X4 при задействовании всех ядер системы.

Анализ вычислительных затрат показал, что в рассматриваемом численном примере во времени выполнения **Алгоритма 1** существенно преобладает доля решения конечно-элементной СЛАУ (рис. 4). Эта доля возрастает по шагам перестроения сеточной модели с увеличением размерности системы уравнений. Дисбаланс нагрузки (ΔW, рис. 5) также возрастает, что, конечно же, уменьшает ускорение и эффективность алгоритма. Малая доля затрат на балансировку нагрузки связана с перераспределением небольшого объема расчетных данных (M_{rem}/M). Перераспределение конечных элементов сетки выполнялось на основе стратегии перемещения сеточного объекта с созданием некоторых или всех его свойств заново. Примерно такое же распределение вычислительных затрат наблюдалось и для МВС- 15000ВМ. Были также проведены тестовые запуски с разделением сетки при идеальной сбалансированности вычислений, которые показали, что разбалансировка не является основной причиной невысокого ускорения $S = T_1(X4(1))/T_{n_p}(X4(1))$ для **Алгоритма 2**.

Профилирование программы поэлементной декомпозиции по **Алгоритму 1** показало, что на мультиядерных системах плохо масштабируется операция поэлементного матричного векторного произведения и значительно возрастает время на коллективные коммуникации при вычислении скалярных произведений.

Модификация алгоритма поэлементного матрично-векторного произведения, улучшающая пространственную и временную локализацию обращений к памяти и сокращающая время ожидания доступа к памяти, позволила существенно увеличить эффективность параллельного алгоритма решения СЛАУ. Как показано на рис. 3, ускорение за счет данной модификации (вариант X4) в сравнении с исходным вариантом (X4(1)), составило почти два раза для случая 96 процессов и приблизилось к значениям для одноядерных процессоров (МВС 15000ВМ). Эффективность применения мультиядерных вычислительных узлов показана на рис. 6.

Модификация алгоритма и программной реализации позволила сократить время выполнения более чем в 1.5 раза для четырехсокетовой системы X6 ($T_{32}(MBC)/T_{32}(X6)$) и в два раза для системы X4 ($T_{64}(MBC)/T_{64}(X4)$). В случае $n_p = 96$ процессов, вычисления на 96 процессорных ядрах были выполнены в полтора раза быстрее, чем на 128 одноядерных процессорах и почти в четыре раза быстрее, чем на 64 ($T_{64}(MBC)/T_{96}(X4) = 3.8$). Данный прирост ускорения и уменьшение времени вычислений связан с более эффективным использованием кэш-

памяти многоядерных процессоров. Ранее в варианте (X4(1)) на хранение массива локальных матриц жесткости в каждом процессе динамически выделялось примерно 2 МБ оперативной памяти (например, на шестом шаге адаптивного уточнения), из которой в операцию $\tilde{\mathbf{A}}^{(i)}\bar{\mathbf{p}}_l^{(i)}$ передавались элементы локальных матриц жесткости. При переходе к пересчету локальных матриц жесткости на каждой итерации **Алгоритма 2** исключается обращение к оперативной памяти за элементами локальных матриц. Таким образом, уменьшается конкуренция вычислительных процессов за ресурсы, чем и объясняется увеличение эффективности алгоритма.

Следует отметить, что эффективность **Алгоритма 2** может быть увеличена за счет оптимизации скалярных произведений, при реализации которых в обменах участвуют все процессы [22].

В дальнейшем интерес представляет такая многоуровневая декомпозиция алгоритма, при которой первому уровню соответствуют процессоры или вычислительные узлы, а второму — ядра процессоров. Первый уровень отвечает за согласованность сеточной модели и вычислений в подобластях, а также за коммунникации между подобластями. Второй уровень обеспечивает мелкозернистый параллелизм при операциях с конечно-элементными данными (матрицы, векторы, сеточные объекты) в подобластях модели. Такая декомпозиция наиболее полно учитывает современную архитектуру многопроцессорных вычислительных систем и применима как для мультиядерных, так и для многоядерных систем. В этом случае программная реализация базируется на динамических процессах/потоках и может осуществляться средствами MPI+OpenMP, MPI2, для многоядерных систем — MPI+CUDA. Отметим, что важнейшую роль будет играть привязка операций второго уровня к ядрам процессоров.

СПИСОК ЛИТЕРАТУРЫ

1. Error-controlled Adaptive Finite Elements in Solid Mechanics / Ed. E. Stein. — John Wiley & Sons, 2002. — 422 p.
2. Le Tallec P. Domain decomposition methods in computational mechanics // Computational Mechanics Advances, 1993. — Vol. 1, № 2. — P. 121–220.
3. Dongarra J., Peterson G., Tomov S., Allred J., Natoli V., Richie D. Exploring New Architectures in Accelerating CFD for Air Force Applications // Proceedings of the 2008 DoD HPCMP Users Group Conference: HPCMP-UGC '08. July 14–17 2008: — IEEE Computer Society: Washington, USA, 2008. — P. 472–478.
4. Novikov A. N., Kopysov S. P. Parallel element-by-element conjugate gradient method with decreased communications costs // The International Summer School Iterative Methods and Matrix Computations (2–9 June 2002) / Ed. Gene H. Golub, Lev A. Krukier. — Rostov-on-Don, 2002. — P. 450–454.
5. Chen K. Matrix Preconditioning Techniques and Applications. — Cambridge: Cambridge University Press, 2005. — 568 p.
6. Dayde M. J., L'Excellent J. Y., Gould N. I. M. Element-by-element preconditioners for large partially separable optimisation problems // SIAM Journal of Scientific Computing, 1997. — Vol. 18, № 6. — P. 1767–1787.
7. Баландин М. Ю., Шурина Э. П. Методы решения СЛАУ большой размерности. — Новосибирск: Изд-во НГТУ, 2000. — 70 с.
8. Van der Vorst H. A. Iterative Krylov Methods for Large Linear Systems. — Cambridge: Cambridge University Press, 2003. — 221 p.
9. Копысов С. П., Новиков А. К., Сагдеева Ю. А. Параллельные схемы разрывного метода Галеркина // Трехмерная визуализация научной, технической и социальной реальности. Кластерные технологии моделирования: Сб. трудов конф. — Ижевск: УдГУ, 2009. — Т. 1. — С. 144–147.
10. Репин С. И., Фролов М. Е. Об апостериорных оценках точности приближенных решений краевых задач для уравнений эллиптического типа // Журнал выч. мат. и мат. физики. — 2002. — Т. 42, №12. — С. 1774–1787.
11. Verfürth R. A review of a posteriori error estimation and adaptive mesh refinement techniques. — Wiley-Teubner, 1996. — 127 p.
12. Копысов С. П., Новиков А. К, Пономарёв А. Б. О параллельном построении неструктурированных сеток // Параллельные вычислительные технологии (ПаВТ'2007): Сб. трудов Межд. конф. 29 янв.–2 фев. 2007, г. Челябинск: Изд. ЮУрГУ, 2007. — Т. 2. — С. 65–76.

13. Круглякова Л. В., Неледова А. В., Тишкин В. Ф., Филатов А. Ю. Неструктурированные адаптивные сетки для задач математической физики (обзор). — Матем. моделирование, 1998. — Т. 10, № 3. — С. 93–116.
14. Rivara M. C. Mesh refinement processes based on the generalized bisection of simplices // SIAM Journal on Numerical Analysis, 1984. — Vol. 21. — P. 604–613.
15. Копысов С. П., Новиков А. К. Анализ способов перестроения треугольных конечно-элементных сеток // Численные методы решения линейных и нелинейных краевых задач: Труды Матем. центра им. Н. И. Лобачевского. — Казань: Изд-во Казан. мат. об-ва, 2003. — Т. 20. — С. 170–180.
16. Копысов С. П., Новиков А. К. Параллельные алгоритмы адаптивного перестроения и разделения неструктурированных сеток // Матем. моделирование. 2002. — Т. 14, № 9. — С. 91–96.
17. Копысов С. П., Новиков А. К., Рычков В. Н. Уровни и алгоритмы динамической балансировки вычислительной нагрузки процессоров // Современные проблемы математического моделирования: Сб. трудов XII Всерос. школы-семинара, Абрау-Дюрсо: Изд-во Южного федерал. ун-та, 2007. — С. 136–165.
18. Оден Д. Конечные элементы в нелинейной механике сплошных сред. — М.: Мир, 1976. — 465 с.
19. Kopysov S., Novikov A. Parallel Adaptive Mesh Refinement with Load Balancing for Finite Element Method // Lecture Notes in Computer Science. — 2001. — Vol. 2127. — P. 266–267.
20. Kopyssov S. P., Novikov A. K. Parallel Adaptive Mesh Refinement with Load Balancing on Heterogeneous Cluster // Parallel and Distributed Computing Practices. — 2002. — Vol. 5, № 4.
21. Копысов С.П. Методы декомпозиции и параллельные распределенные технологии для адаптивных версий метода конечных элементов: автореф. дис. . . . д-ра физ.-мат. наук: 05.13.18 / Институт матем. модел-ния РАН. — М., 2007. — 39 с.
22. Копысов С. П., Новиков А. К. Оптимизация выполнения MPI-приложений на многоядерных многосокетовых вычислительных системах // Актуальные проблемы математики, механики, информатики: Труды конф. Екатеринбург: ИММ УрО РАН, 2009. — С. 58–64.

Поступила в редакцию 01.03.10

S. P. Kopysov, A. K. Novikov
Domain decomposition for parallel adaptive finite element algorithm

The decomposition method for adaptive FEM with refinement of a mesh which includes parallel algorithms is constructed: solutions of systems of the linear equations, a posteriori estimation of an error, local refinement of a mesh and dynamic balancing of computing loading. Their efficiency and structure of computing load is researched at performance on multicore computing systems.

Keywords: domain decomposition, parallel calculations, adaptive mesh refinement, finite element method.

Mathematical Subject Classifications: 65M55, 65M60, 65M50.

Копысов Сергей Петрович, д. ф.-м. н., ведущий научный сотрудник, Институт прикладной механики УрО РАН, 426067, Россия, г. Ижевск, ул. Т. Барамзиной, 34,
E-mail: s.kopysov@gmail.com

Новиков Александр Константинович, к. ф.-м. н., старший научный сотрудник, Институт прикладной механики УрО РАН, 426067, Россия, г. Ижевск, ул. Т. Барамзиной, 34,
E-mail: an@udman.ru