Послойная декомпозиция в конечно-элементном анализе на гибридных архитектурах *

А.К. Новиков, С.П. Копысов, Н.С. Недожогин

Институт механики УрО РАН

Рассматривается распределение вычислений между процессорами, их ядрами и ускорителями; минимизация числа и размеров сообщений между подзадачами при решении конечно-элементных задач на гибридных архитектурах. Представлены параллельные алгоритмы декомпозиции на основе послойного разделения расчетной сетки для конечно-элементного анализа: итерационный метод дополнения Шура, блочный вариант метода сопряженных градиентов и поэлементная схема. В ходе численных исследований сравнивается разделение расчетных областей без ветвления и с ветвлением топологических связей подобластей.

Ключевые слова: конечно-элементный анализ, методы декомпозиции, метод дополнения Шура, блочный алгоритм метода сопряженных градиентов, поэлементные схемы МКЭ, послойное разделение неструктурированной сетки, разделение сетки без ветвления, GPU, гибридные архитектуры.

1. Введение

Современный уровень исследований и вычислительных систем требует развития параллельных алгоритмов и новых подходов к решению трёхмерных конечно-элементных задач. При этом доступные объёмы памяти на вычислительных системах, будь то оперативная память или память ускорителя, накладывают ограничения на максимальный размер решаемых задач. При создании параллельных алгоритмов необходимо обеспечить: сбалансированное распределение вычислений между ядрами центральных процессоров и ускорителей, минимизировать передачу данных между ними, исключить конфликты при одновременной записи в общую память.

Вычисления на гибридных архитектурах, при которых реализуется массовый параллелизм вычислений в рамках гибридных моделей («общая память — параллелизм данных», «обмен сообщениями — параллелизм данных», «обмен сообщениями — общая память — параллелизм данных») требует новых подходов к разделению сеточных данных. Одним из таких подходов является послойное разделение/упорядочение расчетных сеток, которое позволяет исключить состояние гонки при параллельной записи в общую память. Если разделение на слои структурированных сеток является тривиальной задачей, то разделение неструктурированных сеток, к которым относятся конечно-элементные сетки включает: определение ячеек, которым принадлежит узел сетки; задание начального слоя; формирование слоёв из условия соседства ячеек и анализа связности слоёв; объединения слоёв в подобласти сетки.

В данной работе, рассматривается применение послойного разделения в нескольких известных подходах к распараллеливанию методов декомпозиции: в методе дополнения Шура, в блочном методе сопряжённых градиентов решения сеточной СЛАУ и в поэлементных схемах метода конечных элементов. Метод дополнения Шура [1] обладает естественный параллелизмом, связанным с вычислениями в подобластях. Блочный вариант метода сопряжённых градиентов [2] эффективно задействует multiGPU системы совместно с CPU, что позволяет снизить ограничения на максимальный размер системы. Поэлементные схемы позволяют обходится без хранения глобальной матрицы жёсткости, а так же более эффективно использовать память за счёт локализации данных [3]. Рассмотрим эти методы

*Работа выполнена при частичной финансовой поддержке РФФИ (гранты 16–01–00129-а, 17–01–00402-а).

подробнее.

2. Методы декомпозиции в конечно-элементных вычислениях

Как правило, при распараллеливании метода конечных элементов используется декомпозиция по данным, когда некоторым способом [4–6] разделяется расчетная сетка или система сеточных уравнений. Таким образом, распределение вычислений между процессорными ядрами наследуется от разделения данных (конечно-элементных матриц, строк, неизвестных).

2.1. Метод дополнения Шура

Использование метода дополнения Шура, позволяет уменьшить размер решаемой системы. При этом порождается естественный параллелизм, позволяющий выполнять действия в подобластях абсолютно независимо друг от друга. Рассмотрим использование построение дополнения Шура, как один из вариантов методов декомпозиции области в виде алгоритма метода подструктур [1]. Для обеспечения независимости вычислений в отдельных подобластях и последующее их взаимодействие, все узлы области делятся на два множества: внешних и внутренних узлов. Неизвестные перемещения области рассматриваются в виде суперпозиции двух составляющих. Первая составляющая — перемещения, вызванные внешними силами при закреплении границ в подобластях. Перемещения каждой подобласти определяются из уравнений, включающих неизвестные, связанные только с данной подобластью. Вторая составляющая — перемещения, вызванные смещениями границ подобласти с исключенными внутренними узлами.

Пусть область Ω разбита на n_Ω непересекающихся подобластей:

$$\Omega = \Omega_1 \bigcup \Omega_2 \bigcup \ldots \bigcup \Omega_{n_\Omega}, \quad \text{где} \quad \Omega_i \bigcap \Omega_j = \emptyset, \quad \Gamma_B = \bigcup_{i=1}^{n_\Omega} \partial\Omega_i \setminus \partial\Omega. \tag{1}$$

Разделение на подобласти наследуется от процесса разделения дуального графа расчетной сетки $G(V,E) = \bigcup_{i=1}^{n_\Omega} G_i(V_i,E_i)$, здесь множество вершин графа V — это множество конечных элементов расчетной сетки, множество ребер графа E — множество смежных конечных элементов. Множество конечных элементов, образующих подобласть — $V_i \subset V$. Далее полагается, что все подграфы $G_i(V_i,E_i)$ связные, в противном случае система уравнений (2) для подобласти Ω_i распадается на несвязанные системы уравнений.

Узлы расчетной сетки образуют множество $\hat{V}$ и условно разделяются на внешние $\hat{V}_{B_i}$ — принадлежат границе области и внутренние $\hat{V}_{I_i}$ — связанные с узлами подобласти сетки, соответствующей подграфу $G_i(V_i,E_i)$. Из множества внешних узлов выделяются интерфейсные $\hat{V}_{C_i} \subset \hat{V}_{B_i}$, связанные с узлами из других подобластей.

Для каждой подобласти Ω_i строятся системы уравнений, причем степени свободы, связанные с внутренними и внешними (граничными) узлами, разделяются:

$$\begin{pmatrix} A^i_{II} & A^i_{IB} \\ A^i_{BI} & A^i_{BB} \end{pmatrix} \begin{pmatrix} u^i_I \\ u^i_B \end{pmatrix} = \begin{pmatrix} f^i_I \\ f^i_B \end{pmatrix}, \tag{2}$$

где индексы I, B относятся к внутренним и граничным степеням свободы.

Система для интерфейсных узлов определяется, как

$$S_{BB}\tilde{u}_B = \tilde{f}_B, \tag{3}$$

здесь $S_{BB} = \sum_i^{n_\Omega} \left(A^i_{BB} - A^i_{BI} A^{i^{-1}}_{II} A^i_{IB} \right)$ — матрица граничных жесткостей или дополнение Шура для подобласти i, вектор $\tilde{f}_B = \sum_i^{n_\Omega} \left(f^i_B - A^i_{BI} A^{i^{-1}}_{II} f^i_I \right)$ — вектор правых частей.

Преимущества данного подхода состоят: в уменьшение размера решаемой интерфейсной системы (3) и ее обусловленности в сравнении с исходной; естественный параллелизм при работе в подобластях.

Недостатки подхода являются: значительные затраты на формирование матрицы дополнения Шура в последовательном варианте; дополнительное выделение памяти для хранения матриц жёсткости в подобластях и матрицы дополнения Шура; ограничение на связность подобластей; с увеличением числа сеточных подобластей (блоков матрицы A), уменьшаются затраты на вычисления в подобластях, но увеличивается размер системы (3).

2.2. Блочный вариант решения сеточных СЛАУ

При решение системы линейных алгебраических уравнений используется блочный алгоритм, реализованный на гибридных вычислительных системах. При использовании этого алгоритма, задействуются несколько GPU и CPU одновременно. Ключевым фактором служит декомпозиция матрицы. Существуют различные подходы к разделению матриц. Самый простой из них — по строкам. Для снижения коммуникаций между вычислительными узлами был реализован следующий алгоритм. При разделении на блоки, матрица системы A представлялась графом, имеющим число вершин равное размерности A, где a_{ij} — элемент матрицы расположенный в i-ой строке и j-м столбце. Вершины графа матрицы A делятся на множества по числу доступных графических ускорителей. Исходя из этого, вершины графа делятся на две группы: внутренние и граничные (связанные хотя бы с одной вершиной из другого множества).

На основе полученного разделения в каждом блоке формируется несколько матриц A_k, где k — номер блока. В каждом блоке выделяется несколько типов матриц:

- $A_k^{[i_k,i_k]}$ матрица, элементы которой связывают внутренние вершины;
- $A_k^{[i_k,b_k]}$, $A_k^{[b_k,i_k]}$ матрицы, связывающие внутренние вершины с граничными;
- $A_k^{[b_k,b_m]}$ матрица, связывающая граничные вершины k-го блока с граничными вершинами m-го блока.

Здесь $k \neq m$, $k, m \in [1, n_p]$, где n_p — число блоков (равно числу доступных GPU).

Используя эти обозначения, матрица A примет вид

$$A = \begin{pmatrix} A_1^{[i_1,i_1]} & A_1^{[i_1,b_1]} & \cdots & 0 & 0 & \cdots & 0 & 0 \\ A_1^{[b_1,i_1]} & A_1^{[b_1,b_1]} & \cdots & 0 & A_1^{[b_1,b_k]} & \cdots & 0 & A_1^{[b_1,b_{n_p}]} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & A_k^{[i_k,i_k]} & A_k^{[i_k,b_k]} & \cdots & 0 & 0 \\ 0 & A_k^{[b_k,b_1]} & \cdots & A_k^{[b_k,i_k]} & A_k^{[b_k,b_k]} & \cdots & 0 & A_k^{[b_k,b_{n_p}]} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & A_{n_p}^{[i_{n_p},i_{n_p}]} & A_{n_p}^{[i_{n_p},b_{n_p}]} \\ 0 & A_{n_p}^{[b_{n_p},b_1]} & \cdots & 0 & A_{n_p}^{[b_{n_p},b_k]} & \cdots & A_{n_p}^{[b_{n_p},i_{n_p}]} & A_{n_p}^{[b_{n_p},b_{n_p}]} \end{pmatrix}. \quad (4)$$

Тогда при решении методом сопряжённых градиентов каждая векторная операция делится на две: соответствующим внутренним и граничным вершинам. Эти операции независимы и могут выполняться параллельно. Для нахождения матрично-векторного произведения $q = Ap$ на каждом GPU (номер графического ускорителя обозначим k) необходимо вычислить: $q_k^b(GPU) = A_k^{[b_k,i_k]} p_k^i$ и $q_k^i = A_k^{[i_k,i_k]} p_k^i + A_k^{[i_k,b_k]} p_k^b$.

Параллельно с этим, на CPU вычисляется $q_k^b(CPU) = \sum_{m=1}^{m \leqslant n_p} A_k^{[b_k, b_m]} p_m^b$.

Далее векторы $q_k^b(GPU)$ и $q_k^b(CPU)$ суммируются в каждой параллельной нити.

Такой подход уменьшает затраты на обмены между блоками на каждой итерации метода сопряженных градиентов, при условии минимизации границы при разделения графа матрицы A. Распределение блоков матрицы между вычислительными модулями при помощи обмена сообщениями MPI позволило снять ограничения (объем памяти ускорителей в пределах одного вычислительного модуля) на размер решаемой интерфейсной системы.

Преимуществами подхода являются: использование ресурсов multiGPU и центральных процессоров; входными данными в этом подходе может быть любая матрица, включая матрицу дополнения Шура и поэлементную матрицу.

К недостаткам подхода можно отнести: необходимость сформированной матрицы системы; дублирование данных при разделении матрицы системы; затраты на разделение матрицы и поиска оптимального разделения.

2.3. Поэлементная схема метода конечных элементов

В этом случае глобальная матрица жёсткости не формируется, а при решении системы методом подпространств Крылова, например, сопряжённых градиентов матрично-векторное произведение $q = Ap$ разбивается на две операции: по(конечно-)элементное произведение $\tilde{q} = \tilde{A}Cp$ (здесь $\tilde{q}$ и $\tilde{A}$ — вектор и матрица составленные соответственно из локальных векторов и матриц, а C — матрица инцидентности) и сборку вектора $q = C^T\tilde{q}$. Матрица инцидентности C отображает локальное пространство номеров неизвестных (степеней свободы) $[1, 2, \ldots, N_e]$ в глобальное $[1, 2, \ldots, N]$, где N_e — число неизвестных в одном конечном элементе, N — число неизвестных на всей сетке. Такое отображение в конечно-элементном приложении реализуется в виде косвенной индексации в программном коде с использованием номеров узлов сетки или при умножении на матрицу инцидентности, которое эффективно выполняется на графических ускорителях. Стоит отметить, что поэлементные произведения выполняются независимо друг от друга, поэтому могут распределены между параллельными процессами, и нитями, как CPU, так и ускорителей.

К преимуществам подхода можно отнести то, что не требуется формировать и хранить глобальную матрицу жёсткости. Кроме того, естественный параллелизм при формировании локальных (элементных) матриц жёсткости и умножении их на вектор, а также возможность одновременного использования ресурсов multiGPU и CPU.

Недостатками подхода являются: большее число операций при решении системы, когда сборка глобальной матрицы заменяется на сборку вектора, которая выполняется на каждой итерации решения сеточной СЛАУ; множество элементных матриц занимает больше памяти, чем собранная из них глобальная матрица жёсткости, возможно дополнительное хранение матрицы инцидентности.

2.4. Затраты по памяти

Сравним предложенные подходы с точки зрения затрачиваемой памяти на этапах формирования и решения системы для шестигранных конечных элементов с тремя степенями свободы в каждом узле. Далее N — число неизвестных на всей сетке, N_{fe} — число конечных элементов, $N_{nz}(X)$ — число ненулевых элементов некоторой матрицы X, N_B — число неизвестных на границе, включая границы между подобластями.

Поэлементный метод:

- Формирование: N_{fe} матриц 24×6, 6×24, 24×24.
- Решение: 5 вспомогательных векторов при решении методом сопряженных градиентов $8 \cdot N$, N_{fe} матриц 24×24.

Блочный метод сопряжённых градиентов:

- Формирование. Элементные матрицы 24×24 байт, глобальная матрица в формате CSR $12 \cdot N_{nz}(A) + 4 \cdot (N+1)$ байт вектора 24 байт в элементах и глобальный $4 \cdot N$ байт.
- Решение:
  - В общей памяти CPU: глобальная матрица жесткости в формате CSR $12 \cdot N_{nz}(A) + 4 \cdot (N+1)$ байт.
  - На каждом GPU: пять вспомогательных векторов при решении методом сопряженных градиентов $8 \cdot N$.

Метод дополнения Шура

- Формирование. В каждой подобласти: матрицы A_{II}, A_{IB}, A_{BB} и вектора f_I, f_B, промежуточная матрица $A_{II}^{(-1)}$, но хранятся матрица $A'_{IB} = A_{II}^{(-1)} A_{IB}$ и вектор $f'_B = A_{II}^{(-1)} f_I$.
- Решение:
  - Для граничных узлов: пять вспомогательных векторов при решении методом сопряженных градиентов $8 \cdot N_B$, матрица S_{BB} в CSR формате: $12 \cdot N_{nz}(S_{BB}) + 4 \cdot (N_B + 1)$, вектор решения $8 \cdot N_B$.
  - Для внутренних узлов: матрица A'_{IB} и вектор f'_B, вектор результата на границе и вектор результата на внутренних узлах.

3. Послойное разделение для параллельных алгоритмов методов декомпозиции

Появление многоядерных и гибридных вычислительных систем с одной стороны, и увеличение размеров сеточных моделей с другой, привело к развитию многоуровневых подходов при разделении сеток. Если ранее многоуровневые алгоритмы [6] применялись с целью ускорения алгоритмов разделения, то сейчас акцент делается на построение иерархии подобластей для работы со сверхбольшими моделями [7] и эффективным использованием коммуникаций в случае многоядерных [8] и гибридных систем [2]. Можно выделить два направления в построении таких иерархий: последовательное деление существующих областей сетки и деление сетки на множество из большого числа подобластей с последующим их объединением.

Одним из вариантов второго подхода является разделение сетки на слои ячеек и их последующее объединение в подобласти (блоки). В рассматриваемых случаях послойное разделение направлено на исключение ветвления топологических связей подобластей, как следствие, упрощения коммуникационной части алгоритмов, а также на исключение состояния гонки при параллельных вычислениях в общей памяти.

3.1. Метод дополнения Шура

Распараллеливание метод дополнения Шура для гибридных архитектур связано с распределением вычислений между CPU и GPU на этапах формирования интерфейсной системы (5)–(9), ее решения (10) и последующих вычислений в подобластях (11).

- Параллельное формирование дополнение Шура:
  1. Параллельно вычисляем матрицы A'_{IB} на GPU в каждой i-ой подобласти (далее индекс i опущен), решая систему

$$A_{II} A'_{IB} = A_{IB} \tag{5}$$

2. Параллельно вычисляем матрицы S_{BB} — слагаемые матрицы дополнения Шура, соответствующие подобласти i.

$$S_{BB} = A_{BB} - A_{BI}A'_{IB}. \tag{6}$$

3. Для каждой подобласти i решаем на GPU систему:

$$A_{II}f'_I = f_I. \tag{7}$$

4. В каждой подобласти i вычисляем слагаемые вектора правых частей для границы

$$\tilde{f}_B = f_B - A_{BI}f'_I. \tag{8}$$

5. Формируем на CPU матрицу интерфейсной системы в формате DCSR [1] и вектор правой части

$$\tilde{S}_{BB} = \sum_i S^i_{BB}, \quad \tilde{f}_B = \sum_i \tilde{f}^i_B. \tag{9}$$

- Параллельное решение:

1. Решаем систему уравнений для границы на multiGPU

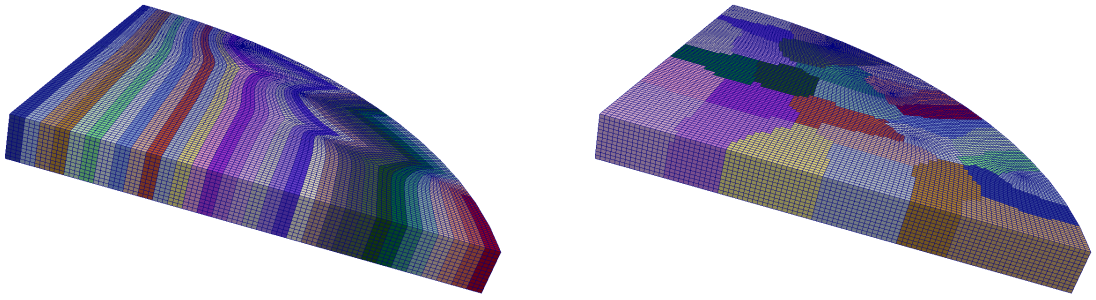
$$S_{BB}\tilde{u}_B = \tilde{f}_B. \tag{10}$$

2. Нахождение неизвестных для внутренних узлов

$$u_I = f'_I - A'_{IB}\tilde{u}_B. \tag{11}$$

В подобластях матрицы S_{BB} и A_{BI}, A'_{IB} хранятся, как множества строк, в формате CSR, решение (5) хранится в столбцах. Для числа обусловленности κ системы (10) справедлива оценка $\kappa(M^{-1}S_{BB}) \leq C(1 + \log(H/h))$, где C — некоторая константа, H — размер расчетной области, h — шаг сетки, M — предобуславливатель.

Важным моментом, при распараллеливании метода дополнения Шура является отсутствие ветвления топологических связей подобластей. Это существенно упрощает определение «граничных» неизвестных. Таким свойством обладает послойное разделение расчетной сетки, при котором подобласти, сформированные из последовательности слоёв, в общем случае, связаны только с предыдущей и последующей подобластью (рисунок 1а), в отличие от разделения, полученного при применении библиотеки METIS (рисунок 1б).



а) б)

Рис. 1. Неструктурированная сетка, разделенная на 32 подобласти: а) без ветвления; б) с ветвлением (METIS).

Отметим, также, что послойное разделение упрощает процесс формирование связных подобластей из разрывных, что существенно для метода дополнения Шура. В таком случае, возможны два подхода: на этапе формирования слоёв и при объединении слоёв в подобласти. В первом подходе части разрывного слоя присоединяются к предыдущему или последующему (не вносятся в текущий слой, а оставляются для формирования следующего слоя) слою. Во втором подходе, части разрывных слоёв присоединяются к предыдущей или последующей подобласти.

3.2. Блочный метод сопряжённых градиентов

В этом случае, каждая операция над векторами разделяется на две: соответствующую «внутренним» и «граничным» строкам матрицы. Эти операции независимы и могут выполняться параллельно, но требуют предварительного разделения строк матрицы A на «внутренние» и «граничные» [2]. Для этого используются следующий подход: строится граф матрицы (см. п. 2.2), который разделяется алгоритмом из библиотеки METIS, исходя из полученного разделения определяются «внутренние» и «граничные» строки матрицы. Следующий псевдокод иллюстрирует реализацию блочного метода для multiGPU с использованием библиотеки cublas для параллельного выполнения операций над векторами на графических ускорителях.

1. Инициализация (1a)–(1g):
   (a) $A, \bar{M} \in \mathbb{R}^{N\times N}$ $\{\bar{M}$ *Формируется на GPU и хранится в CSR формате*$\}$
   (b) $u, r, p, q, z \in \mathbb{R}^N$ $\{$*Вектора хранятся на GPU, каждый вектор состоит из двух компонент* $(\cdot)_I$ *и* $(\cdot)_B\}$
   (c) $r_0 \leftarrow f$ $\{cublasDcopy\}$
   (d) $u_0 \leftarrow 0$ $\{$*Инициализация на GPU*$\}$
   (e) $z_0 \leftarrow \bar{M} r_0$ $\{$*Вычисление GPU*$\}$
   (f) $p_0 \leftarrow z_0$ $\{cublasDcopy\}$
   (g) $\rho_0 \leftarrow (r_0, z_0)$ $\{$*здесь и далее* $(\cdot,\cdot) = \sum\limits_P (\cdot,\cdot)^{(P)}\}$
2. Итеративный процесс.
   While $||r_i||_2/||b||_2 > \varepsilon$ do (2a)–(2j):
   (a) $q_k^b = \sum\limits_{m=1}^{m\leqslant n_p} A_k^{[b_k,b_m]} p_m^b$ $\{$*Вычисление на CPU и копирование результата в память GPU*$\}$
   (b) $q_k^b += A_k^{[b_k,i_k]} p_k^i$ $\{$*Вычисление на GPU*$\}$
   (c) $q_k^i = A_k^{[i_k,i_k]} p_k^i + A_k^{[i_k,b_k]} p_k^b$ $\{$*Вычисление на GPU*$\}$
   (d) $\alpha_i \leftarrow (r_i, z_i)/(q_i, p_i)$ $\{cublasDdot\}$
   (e) $u_{i+1} \leftarrow u_i + \alpha_i p_i$ $\{cublasDasxpy\}$
   (f) $r_{i+1} \leftarrow r_i - \alpha_i q_i$ $\{cublasDasxpy\}$
   (g) $z_{i+1} \leftarrow \bar{M} r_{i+1}$ $\{$*выполняется на GPU*$\}$
   (h) $\rho_{i+1} \leftarrow (r_{i+1}, z_{i+1})$ $\{cublasDdot\}$
   (i) $\beta_{i+1} \leftarrow \rho_{i+1}/\rho_i$
   (j) $p_{i+1} \leftarrow z_{i+1} + \beta_{i+1} p_i$ $\{$*сочетание функций cublasDscal и cublasDasxpy*$\}$

В комментариях к псевокоду указано устройство, где выполняются вычисления (CPU или GPU) и формат в котором хранятся матричные данные. Дальнейшим развитием подхода, в рамках конечно-элементного анализа, является использование послойного разделения сетки для разделения матрицы на граничные, внутренние и общие элементы. В этом случае разделение матрицы будет наследоваться от разделения сетки, тогда строки (4) будут содержать не более двух подматриц $A_k^{[b_k,b_m]}$. Таким образом, будет обеспечена большая, в сравнении с разделением без ветвления (METIS), локализация доступа к данным.

3.3. Поэлементная схема

При параллельной реализации поэлементной схемы необходимо выделить множества конечных элементов, не имеющих общих вершин (с одинаковыми локальными номерами) для исключения одновременного доступа к памяти и каждое такое множество связать с параллельным процессом/нитью, а сборку вектора результата одновременно выполнять над не имеющими общих узлов конечными элементами.

Рассматривалось несколько вариантов формирования множеств конечных элементов из слоёв неструктурированной сетки [9], приводящих к упорядочению ячеек сетки и исключающих одновременный доступ из параллельных процессов (нитей) к конечным элементам, имеющим общие вершины. Например, в блочном варианте слои объединялись последовательно, полученное объединение слоёв разделялось на подобласти, содержащие примерно равное число конечных элементов. При объединении по чётности использовались номера слоёв, исходя из которых формировались множества четных и нечетных слоёв.

Предложенные варианты упорядочения ячеек сравнивались по влиянию на сбалансированность вычислительной нагрузки. Для этого рассматривались результаты поэлементного матрично-векторного произведение без сборки векторов (данная операция полностью параллельна) при четырех вариантах разделения сетки: блочном, по четности, он же сбалансированный и многоуровневого графового разделения из пакета METIS [6]. Полученные результаты показали возможности рассматриваемых алгоритмов разделения по получению сбалансированных по числу расчетных ячеек подобластей/слоёв [10]. Следует отметить, что достижение линейного ускорения на данной операции ограничивается только дисбалансом вычислительной нагрузки (по числу ячеек), который меньше уровня разбалансированности, получаемой многоуровневым разделением (METIS) дуального графа расчетной сетки.

4. Заключение

Рассмотрено применение послойного разделения неструктурированной сетки в рамках нескольких известных подходов к распараллеливанию методов декомпозиции в конечно-элементном анализе: в методе дополнения Шура, в блочном методе сопряжённых градиентов решения сеточной СЛАУ и в поэлементной схеме метода конечных элементов.

Применение послойного разделения сетки в методе дополнения Шура позволило: в полтора раза сократить время формирования локальных матриц S_{BB}^i за счет локализации доступа к данным (более компактного портрета матриц); исключить конфликты при параллельной сборке глобальной матрицы S_{BB}, как и в случае поэлементной схемы. В дальнейшем этот подход будет использоваться для ограничения размера интерфейсной системы на этапе формирования иерархии подобластей.

При использовании блочного метода сопряженных градиентов представляется перспективным наследование послойного разделения сетки для разделения матрицы на граничные, внутренние и общие элементы. В этом случае будет обеспечена большая, в сравнении с разделением без ветвления, локализация доступа к данным.

В поэлементной схеме благодаря послойному разделению (разделение без ветвления) были исключены конфликты в общей памяти при параллельной сборке вектора результата матрично-векторного произведения, что позволило достичь ускорения близкого к линейно-

му при вычислениях в модели «общая память — параллелизм данных».

Литература

1. Kopysov S.P., Kuzmin I. M., Nedozhogin N.S., Novikov A.K., Sagdeeva Y.A. Scalable hybrid implementation of the Schur complement method for multi-GPU systems // Journal of Supercomputing. 2014. Vol. 69, No 1. P. 81–88.

2. Копысов С.П., Кузьмин И.М., Недожогин Н.С., Новиков А.К. Параллельные алгоритмы метода дополнения Шура в программной модели CUDA+OpenMP // Вестник УГАТУ. 2013. Т. 17. № 5(58). С. 219–229.

3. Kopysov S.P., Novikov A.K., Nedozhogin N.S., Rychkov V.N. Accelerating assembly operation in element-by-element FEM on multi-core platforms // Communications in Computer and Information Science. 2016. Vol. 687. P. 3–14.

4. Markall G.R., Slemmer A., Ham D.A., Kelly P.H.J., Cantwell C.D., Sherwin S.J. Finite element assembly strategies on multi-core and many-core architectures // Int. J. Numer. Meth. Fluids. 2013. Vol. 71, No. 1. P. 80–97.

5. Löhner R. Cache-efficient renumbering for vectorization // International Journal for Numerical Methods in Biomedical Engineering. 2010. Vol. 26, No 5. P. 628–636.

6. Karypis G., Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs // SIAM Journal on Scientific Computing. 1999. Vol. 20, No 1. P. 359–392.

7. Golovchenko E.N., Yakobovskiy M.V. Parallel partitioning tool GridSpiderPar for large mesh decomposition // Russian Supercomputing Days: Proceedings of the International Conference, Moscow, Russia, 28–29 September, 2015. Moscow State University. Moscow. 2015. P. 303–315.

8. Копысов С.П. Оптимальное разделение области для параллельного метода подструктур // Сб. трудов Пятого Всероссийского семинара «Сеточные методы для решения краевых задач и приложения». – Казань: Изд-во КГУ, 2004. – С. 121–124.

9. Копысов С.П., Новиков А.К., Пиминова Н.К. Параллельная композиция в поэлементной схеме метода конечных элементов // Параллельные вычислительные технологии (ПаВТ'2016): Труды межд. науч. конф. (Архангельск, 28 марта – 1 апреля 2016 г.). Челябинск: Изд. центр ЮУрГУ, 2016. С. 555–560.

10. Новиков А.К., Пиминова Н.К., Копысов С.П. Послойное разделение конечно-элементных сеток для мультиядерных архитектур // Суперкомпьютерные дни в России. Москва. Изд-во МГУ. 2016. С. 493–504.

Layer-by-Layer Decomposition in Finite Element Analysis on Hybrid Architectures

A.K. Novikov, S.P. Kopysov, N.S. Nedozhogin

Institute of Mechanics UB RAS

The distribution of computations between processors, their cores and accelerators and minimization of the number and size of messages between subtasks when solving finite-element problems on hybrid architectures are considered. Parallel algorithms of decomposition based on layer-by-layer mesh partioning for finite element analysis are presented: an iterative Schur complement method, a block variant of the conjugate gradient method, and element-by-element scheme. In the numerical studies, the partitioning without branching and with branching of the topological links of the subdomains are compared.

Keywords: finite element analysis, layer-by-layer mesh partitioning, domain decomposition, Schur complement method, block conjugate gradients, element-by-element, mesh partitioning without branching, GPU, hybrid architecture.

References

1. Kopysov S.P., Kuzmin I. M., Nedozhogin N.S., Novikov A.K., Sagdeeva Y.A. Scalable hybrid implementation of the Schur complement method for multi-GPU systems // Journal of Supercomputing. 2014. Vol. 69, No 1. P. 81–88.

2. Kopysov S.P., Kuzmin I. M., Nedozhogin N.S., Novikov A.K. Parallel'nye algoritmy metoda dopolneniya Shura v programmnoj modeli CUDA+OpenMP [Parallel algorithms of the Shur complement method in software model CUDA+OpenMP] // Vestnik UGATU. [Bulletin of UGATU]. 2013. Vol. 17, No 5(58). P. 219–229. (in Russian)

3. Kopysov S.P., Novikov A.K., Nedozhogin N.S., Rychkov V.N. Accelerating assembly operation in element-by-element FEM on multi-core platforms // Communications in Computer and Information Science. 2016. Vol. 687. P. 3–14.

4. Markall G.R., Slemmer A., Ham D.A., Kelly P.H.J., Cantwell C.D., Sherwin S.J. Finite element assembly strategies on multi-core and many-core architectures // Int. J. Numer. Meth. Fluids. 2013. Vol. 71, No. 1. P. 80–97.

5. Löohner R. Cache-efficient renumbering for vectorization // International Journal for Numerical Methods in Biomedical Engineering. 2010. Vol. 26, No 5. P. 628–636.

6. Karypis G., Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs // SIAM Journal on Scientific Computing. 1999. Vol. 20, No 1. P. 359–392.

7. Golovchenko E.N., Yakobovskiy M.V. Parallel partitioning tool GridSpiderPar for large mesh decomposition // Russian Supercomputing Days: Proceedings of the International Conference, Moscow, Russia, 28–29 September, 2015. Moscow State University. Moscow. 2015. P. 303–315.

8. Kopysov S.P. Optimal'noe razdelenie oblasti dlya parallel'nogo metoda podstruktur [Optimal separation of the parallel substructure method] // Materialy Vserossijskogo seminara, posvyashhennogo 200-letiyu Kazanskogo gosudarstvennogo universiteta (Kazan', 17 - 21 sentyabrya 2004) [Proceedings of the 5-th Russia seminar "Mesh methods for boundary-value problems and applications" (Kazan, Russia, 17–21 sept. 2004)]. Kazan: KGU, 2004. P. 121–124. (in Russian)

9. Kopysov S.P., Novikov A.K., Piminova N.K. Parallel'naya kompozitsiya v poelementnoy skheme metoda konechnykh elementov [Parallel composition in element-by-element scheme of finite element method] // Parallel'nye vychislitel'nye tekhnologii (PaVT'2016): Trudy mezhd. nauch. konf. (Arkhangel'sk, 28 Marta – 1 Aprelya 2016) [Parallel Computational Technologies (PCT'2016): Proceedings of the International Scientific Conference (Arhangelsk, Russia, March, 28 – April, 1, 2016)]. Chelyabinsk: Publishing of the South Ural State University. 2016. P. 555–560. (in Russian)

10. Novikov A.K., Piminova N.K., Kopysov S.P. Layer-by-Layer Partitioning of Finite-Element Meshes for Multi-core Architectures // Russian Supercomputing Days: Proceedings of the International Conference, Moscow, Russia, 26–27 September, 2016. Moscow State University. Moscow. 2016. P. 493–504.