УЧЕНЫЕ ЗАПИСКИ КАЗАНСКОГО УНИВЕРСИТЕТА
Том 154, кн. 3 Физико-математические науки *2012*

УДК 004.272.2+519.612:519.63

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ ФОРМИРОВАНИЯ И РЕШЕНИЯ СИСТЕМЫ ДОПОЛНЕНИЯ ШУРА НА ГРАФИЧЕСКИХ УСКОРИТЕЛЯХ

С.П. Копысов, И.М. Кузьмин, Н.С. Недожогин, А.К. Новиков

Аннотация

В работе рассмотрен параллельный алгоритм вычисления дополнения Шура. Эффективное применение нескольких графических ускорителей для метода дополнения Шура связано с разделением матриц и определением алгоритмов, которые более эффективно выполняются на центральном процессоре (CPU) или графических ускорителях (GPU). Представлен алгоритм обращения матрицы через решение матричной системы множеством параллельных потоков. Показано, что формирование матриц дополнения Шура для нескольких подобластей эффективно выполнять на GPU, а с ростом числа подобластей – на CPU. Для решения интерфейсной системы предложен параллельный алгоритм метода сопряженных градиентов с явным предобуславливателем, позволяющий достигать существенного ускорения вычислений (в 251 раз) на восьми GPU при разделении исходной системы уравнений на 64 подобласти.

Ключевые слова: дополнение Шура, параллельные вычисления, предобусловленный метод сопряженных градиентов, графические ускорители.

Введение

В работе рассматриваются параллельные алгоритмы, основанные на блочной декомпозиции матриц и так называемом дополнении Шура. Впервые, метод дополнения Шура был математически сформулирован и описан в работах Исаи Шура (I. Schur), а название своё получил после выхода работы [1], хотя до этого метод и был известен в вычислительной механике как метод подструктур и впервые встречается в работе [2]. В нашей стране он известен как метод суперэлементов [3]. Метод дополнения Шура используется также для обращения больших матриц при помощи разбиения на клетки [4]. Первоначально [2] он рассматривался как метод декомпозиции области для вычислений на системах с ограниченными вычислительными мощностями. В дальнейшем этот метод использовался для построения методов декомпозиции области без перекрытий при разработке параллельных алгоритмов решения задач вычислительной механики. В настоящее время его всё чаще рассматривают как гибридный метод решения систем линейных алгебраических уравнений (СЛАУ) [5, 6], сочетающий преимущества как прямых, так и итерационных методов и учитывающий различные архитектуры параллельных вычислительных систем. Кроме того, на основе дополнения Шура строятся эффективные предобуславливатели для решения СЛАУ различных видов (см., например, [7]).

Во многих из указанных задач требуется решить блочную (2×2) систему уравнений вида

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}. \tag{1}$$

Дополнение Шура матрицы A_{11} определяется как

$$S = A_{22} - A_{21}A_{11}^{-1}A_{12},$$

если матрица $A_{11} \in \mathbb{R}^{n\times n}$ – квадратная и невырожденная, $A_{12} \in \mathbb{R}^{n\times m}$, $A_{21} \in \in \mathbb{R}^{m\times n}$, $A_{22} \in \mathbb{R}^{m\times m}$. Тогда редуцированная система уравнений для нахождения x_2 записывается в виде

$$Sx_2 = f_2 - A_{21}A_{11}^{-1}f_1.$$

Неизвестные x_1 находятся из соотношений

$$x_1 = A_{11}f_1 - A_{12}A_{11}^{-1}x_2.$$

При большем числе блоков решение системы уравнений через дополнение Шура позволяет ускорить вычисления путем параллельного обращения и умножения подматриц меньшего размера, чем у исходной матрицы.

Современные графические ускорители (GPU) позволяют для ряда задач, в том числе при работе с векторами и матрицами, получать ускорение вычислений в десятки и сотни раз по сравнению с центральным процессором (CPU). Тысячи потоков (нитей) GPU могут эффективно выполнять одновременно большое число простых арифметических операций, что характерно для мультипликативных и аддитивных операций с векторами и матрицами. Вместе с тем последовательные операции и ветвления, характерные, например, при разложении матриц на треугольные множители, выполняются на GPU медленнее, чем ядрами центрального процессора.

Таким образом, эффективное применение GPU, тем более нескольких, для метода дополнения Шура связано с декомпозицией матриц и определением алгоритмов, которые более эффективно выполняются на CPU или на графическом ускорителе.

В основе метода дополнения Шура лежит идея независимого расчета отдельных блоков матриц (подобластей) и последующего учета их взаимодействия. В настоящей работе рассматриваются системы уравнений, получаемые при решении трехмерных задач теории упругости и использующие разделение расчетной конечноэлементной сетки при формировании блочной структуры матриц системы. В данном методе можно реализовать два уровня распараллеливания: первый уровень связан с разделением вычислений между подобластями [8, 10], второй – с параллельной реализацией методов, которые используются внутри отдельной подобласти. Помимо этого, распараллеливанию подлежит и решение системы для дополнения Шура (интерфейсной системы). В представленной работе рассматривается второй уровень распараллеливания, для которого предложены алгоритмы формирования матриц дополнения Шура, а также решение интерфейсной системы уравнений с использованием нескольких графических ускорителей.

1. Ресурсная эффективность метода дополнения Шура

Рассмотрим процесс построения дополнения Шура как один из вариантов методов декомпозиции области в виде алгоритма метода подструктур [2]. Для обеспечения независимости вычислений в отдельных подобластях и последующего их взаимодействия все узлы области делятся на два множества: внешние и внутренние. 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:

$$\Omega = \Omega_1 \bigcup \Omega_2 \bigcup \ldots \bigcup \Omega_{n_\Omega}, \quad \text{где} \quad \Omega_i \bigcap \Omega_j = \emptyset, \quad \Gamma_B = \bigcup_{i=1}^{n_\Omega} \partial\Omega_i \setminus \partial\Omega. \tag{2}$$

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 V вершин графа – это множество конечных элементов расчетной сетки, множество E ребер графа – множество смежных конечных элементов. Множество конечных элементов, образующих подобласть, – это $V_i \subset V$. Далее полагается, что все подграфы $G_i(V_i, E_i)$ связные, в противном случае система уравнений для Ω_i подобласти распадается на несвязанные между собой системы уравнений.

Узлы расчетной сетки образуют множество $\widehat{V}$ и условно разделяются на внешние $\widehat{V}_{B_i}$ (принадлежат границе области) и внутренние $\widehat{V}_{I_i}$ (связанные с узлами подобласти) узлы сетки, соответствующей подграфу $G_i(V_i, E_i)$. Из множества внешних узлов выделяются интерфейсные $\widehat{V}_{C_i} \subset \widehat{V}_{B_i}$, связанные с узлами из других подобластей. В соответствии с этими обозначениями и введённым упорядочиванием система уравнений примет вид

$$\begin{pmatrix} A_{II}^1 & & & & A_{IB}^1 \\ & A_{II}^2 & & & A_{IB}^2 \\ & & \ddots & & \vdots \\ & & & A_{II}^{n_\Omega} & A_{IB}^{n_\Omega} \\ A_{BI}^1 & A_{BI}^2 & \ldots & A_{BI}^{n_\Omega} & A_{BB} \end{pmatrix} \begin{pmatrix} u_I^1 \\ u_I^2 \\ \vdots \\ u_I^{n_\Omega} \\ u_B \end{pmatrix} = \begin{pmatrix} f_I^1 \\ f_I^2 \\ \vdots \\ f_I^{n_\Omega} \\ f_B \end{pmatrix}, \tag{3}$$

где верхний индекс i определяет принадлежность подобласти Ω_i, а нижние индексы I, B относятся к внутренним и граничным степеням свободы. Здесь невырожденная матрица $A_{II} = \sum_{i=1}^{n_\Omega} A_{II}^i$ соответствует A_{11} из системы уравнений (1), матрица A_{BB} – блоку A_{22}, блок матрицы $A_{IB} = \sum_{i=1}^{n_\Omega} A_{IB}^i$ – блоку A_{12}. Отметим, что блок матрицы A_{II} положительно определен и симметричен, так же как и полная матрица в (3) для рассматриваемого класса задач.

Система для интерфейсных узлов есть

$$S_{BB} u_B = \widetilde{f}_B. \tag{4}$$

Здесь $S_{BB} = \sum_i^{n_\Omega} \left(A_{BB}^i - A_{BI}^i A_{II}^{i^{-1}} A_{IB}^i \right)$ – матрица дополнения Шура для подобласти i, вектор $\widetilde{f}_B = \sum_i^{n_\Omega} \left(f_B^i - A_{BI}^i A_{II}^{i^{-1}} f_I^i \right)$ – вектор правых частей.

Для решения задач используется усовершенствованный алгоритм метода подструктур. В его основе лежит использование свойств невырожденности и положительной определенности матриц подобластей. Для таких матриц существует разложение Холесского $A_{II} = L_{II} L_{II}^T$, где L – нижняя треугольная матрица с положительными диагональными элементами. Использование разложения Холесского значительно сокращает вычислительные затраты и объем используемой оперативной памяти вычислительных машин.

Ниже рассмотрена реализация последовательного алгоритма вычисления дополнения Шура ($n_\Omega > n_p$ и число процессоров $n_p = 1$) и приведены вычислительные затраты, связанные с каждым шагом алгоритма (для каждого шага

алгоритма показано число необходимых операций с учетом симметрии матриц, причем операции сложения и умножения принимаются за одну).

Алгоритм 1. Последовательный вариант дополнения Шура

1: Выполним разложение Холесского матрицы A_{II} для соответствующей подобласти (индекс i опущен)
$$A_{II} = L_{II} L_{II}^T. \qquad (n_I^3/6)$$
2: Вычисляем вспомогательные переменные
$$A'_{IB} = A_{II}^{-1} A_{IB}. \qquad (n_B \cdot n_I^2)$$
3: Сформируем матрицы граничной жесткости подобластей
$$S_{BB} = A_{BB} - A_{BI} A'_{IB}. \qquad ((n_I \cdot n_B^2)/2)$$
4: Формируем вектор правой части
$$\widetilde{f}_B = f_B - A_{BI} A_{II}^{-1} f_I. \qquad (n_I \cdot n_B)$$
5: Собираем и решаем систему уравнений
$$S_{BB} u_B = \widetilde{f}_B. \qquad (k(M^{-1} S_{BB}) \leq C(1 + \log(H/h)))$$
6: Определяем неизвестные для внутренних узлов
$$u_I = A_{II}^{-1} f_I - A'_{IB} u_B. \qquad (n_I^2).$$

Приняты следующие обозначения: $n_I = m \cdot |\widehat{V}_{I_k}|$ и $n_B = m \cdot |\widehat{V}_{B_k}|$ – число внутренних и граничных степеней свободы соответственно, m – число степеней свободы в узле сетки, h – шаг сетки, H – размер подобласти, M – предобуславливатель для дополнения Шура. На пятом шаге **Алгоритма 1** приведена оценка обусловленности матрицы S_{BB}, а не вычислительная сложность, которая зависит от выбора метода решения системы уравнений. В настоящей работе в качестве метода решения СЛАУ использовался метод сопряженных градиентов с различными предобуславливателями. Матрицы S_{BB}, A_{II} положительно определены и симметричны. Отметим, что порядок матрицы S_{BB} значительно меньше, чем исходной матрицы, но достаточно большой. Поэтому для решения системы $S_{BB} u_B = \widetilde{f}_B$ применяются итерационные методы решения и компактные схемы хранения матриц.

Для обеспечения эффективной работы с матрицами используются различные схемы хранения. Матрицы S_{BB}, A_{II} являются симметричными, поэтому возможно хранение только её части (верхней или нижней треугольной и главной диагоналей), оценка ресурсоёмкости схем хранения производится в зависимости от способа хранения симметричных матриц. Простейший вариант – хранить матрицу, не исключая нули. В этом случае для хранения используется массив по числу элементов матрицы N^2. Этот способ является наиболее затратным с точки зрения требуемой памяти, что особенно существенно для больших задач.

Формат хранения DCSR, используемый при вычислениях в данной работе, представляет собой массив, состоящий из списка упакованных строк, реализован в системе конечноэлементного анализа FEStudio [8, 9]. Каждая строка матрицы представляет из себя структуру, состоящую из двух массивов. В первом массиве хранятся значения ненулевых элементов, во втором – столбцовые индексы этих элементов. Размер каждого из массивов для строки i равен числу ненулевых элементов Nnz_i и меняется динамически по мере необходимости.

Предложенный формат DCSR является разновидностью распространенного формата хранения разреженных матриц – формат CSR (Compressed Sparse Row Storage Format). Для матрицы A при использовании формата CSR выделяются три одномерных массива, в которых хранятся ненулевые значения $\{a_{ij} \,|\, a_{ij} \neq 0\}$, $1 \leqslant i, j \leqslant Nnz$, их столбцовые индексы $\{j \,|\, a_{ij} \neq 0\}$, $1 \leqslant i, j \leqslant Nnz$, и номера элементов a_{i1}, $1 \leqslant i \leqslant N + 1$, в двух первых массивах.

Оценим ресурсоёмкость предложенной схемы хранения матриц, для чего сравним три формата хранения матриц (ленточный, DCSR и CSR) по следующим параметрам: требуемая память, алгоритмическая сложность доступа к элементу и его внесения. Сравнение показывает, что алгоритмические сложности доступа к элементу $\mathcal{O}(2N_\beta+1)$ и его внесения для ленточного формата – $\mathcal{O}((2N_\beta+1)N)$, для DCSR – $\mathcal{O}(Nnz^*)$ и $\mathcal{O}(Nnz^*)$ соответственно, а для формата CSR – $\mathcal{O}(Nnz^*)$ и $\mathcal{O}(Nnz)$, здесь $Nnz^* = \max\limits_{i=1,\ldots,N}\{Nnz_i\}$. Необходимый объем памяти для ленточного формата – $8(2N_\beta+1)N$, для формата DCSR – $\sum\limits_{i=1}^{N}(12Nnz_i+4)$, для формата CSR – $12Nnz+4(N+1)$ байт. Для для формата DCSR полагается, что при хранении вещественной величины используется 8 байт памяти и 4 байта для целого числа. В симметричном случае величины $2N_\beta+1$ в оценках сложности алгоритма (и приведенной далее частоты доступа) заменяются на $N_\beta+1$, а Nnz и Nnz_i относятся к элементам треугольной матрицы.

Отметим, что формат DCSR более удобен, чем CSR, при выполнении треугольного разложения матриц A_{II}, когда в строках появляются новые ненулевые элементы. В этом случае изменение в одной из строк не требует смещения всех последующих элементов в массивах, как в формате CSR. Поэтому процедура добавления нового элемента имеет меньшую алгоритмическую сложность $\mathcal{O}(Nnz^*)$, чем в формате CSR – $\mathcal{O}(Nnz)$. Кроме того, среди представленных форматов объем памяти, необходимый в случае формата DCSR, равный $\sum\limits_{i=1}^{N}(12Nnz_i+4)$ байт, является минимальным.

Преимущества ленточного формата по сложности доступа и добавления элемента нивелируются увеличением необходимого объема памяти $8(2N_\beta+1)N$ и частоты доступа к элементам матрицы $\mathcal{O}((2N_\beta+1)N)$ вместо $\mathcal{O}(Nnz)$ для форматов DCSR и CSR. Здесь $N_\beta = \max_{i=1}^{N}\max_{j=1}^{N}\{j-i \,|\, a_{ij}\neq 0\}$ – так называемая полуширина ленты, зависящая от нумерации неизвестных и уравнений, N – размерность матрицы, Nnz – число ненулевых элементов в матрице, Nnz_i – число ненулевых элементов в i-й строке матрицы.

Как показали эксперименты [10], схема хранения оказывает существенное влияние на время вычислений. Отметим, что для ленточных матриц время формирования дополнения Шура составляет порядка 80% от общего времени, затраченного на решение. Переход на формат DCSR для матриц A_{BB}, A_{IB}, A_{II} для одной и той же задачи дал сокращение затрат в четыре раза. Таким образом, в последовательном варианте наиболее эффективным, с точки зрения ресурсоемкости и алгоритмической сложности, представляется использование метода дополнения Шура с разложением Холесского матрицы A_{II} и хранением матриц в сжатом формате.

Обработка строк матрицы, хранящейся в формате DCSR на графическом ускорителе (GPU), потребует для каждой строки: выделения памяти на GPU, копирования строки, далее в ядре выполнения вычислений над строкой и возвращения результата в оперативную память или кэш CPU. Таким образом, вместо выделения памяти и копирования двух массивов размера Nnz потребуется $2N$ выделений памяти и копирований массивов размера Nnz_i, поэтому для вычислений на GPU матрица из формата DCSR переводится в формат CSR.

2. Параллельная эффективность метода дополнения Шура

В методе декомпозиции области можно выделить два направления распараллеливания процесса решения задачи. Первое связано с методами явного деления на подобласти, когда параллельные вычисления осуществляются на высоком уровне

(уровне области), то есть число ветвей параллельного алгоритма равно числу подобластей. Как показывает практика, такой подход дает значительное ускорение, если каждый процессор решает в своей подобласти свою подзадачу. Второе направление (уровень подобласти) – неявные методы подобластей. Внутри каждой подобласти различные задачи могут быть решены наиболее эффективным способом.

Рассмотрим, исходя из обозначенных вариантов распараллеливания, **Алгоритм 1**. Как видно, *Шаги 1–4, 6* выполняются для каждой подобласти независимо, что говорит о естественном параллелизме, который обеспечивает первый уровень распараллеливания – на уровне области. Распараллеливание *Шага 5* зависит от выбранного метода решения системы (4).

Наиболее трудоёмкими являются шаги формирования (*Шаги 1–4*) и решения системы дополнения Шура (*Шаг 5*). Поэтому распределение вычислительных затрат на этих шагах по нескольким процессорам (центральный процессор, графический ускоритель) является решающим фактором для параллельного ускорения вычислений и определяет эффективность распараллеливания **Алгоритма 1**.

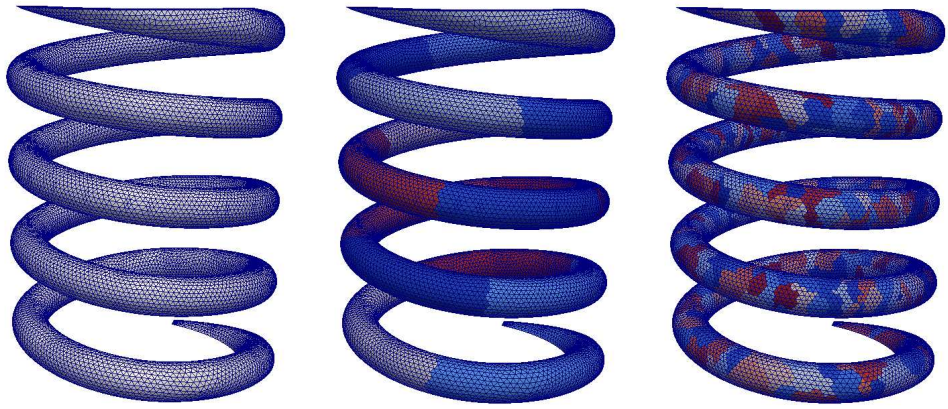
Этапы формирования внутри каждой подобласти выполняются независимо от других подобластей, а значит, становится возможной их параллельная реализация внутри каждой подобласти.

При реализации параллельных алгоритмов неизбежно возникает проблема балансировки вычислительной нагрузки. В качестве примера рассмотрим *Шаг 4* алгоритма метода дополнения Шура. На этом шаге происходит формирование локальных матриц дополнения Шура, выполняемое независимо на каждой из подобластей. На следующем шаге решается система уравнений с глобальной матрицей дополнения Шура, состоящей из локальных (см. соотношение (4)), то есть *Шаг 5* не может быть выполнен, пока не завершится формирование локальных матриц жесткости всеми параллельными процессами (потоками).

Таким образом, источников неравномерности нагрузки может быть несколько. Одним из них является неравномерное распределение количества подобластей на вычислительные узлы – этот недостаток легко исключить, разделив область на количество подобластей, кратное количеству используемых вычислительных узлов. Другой источник – неравномерное распределение узлов сетки и конечных элементов по подобластям. В этом случае разбалансировка и неоднородность разделения исключается заданием дополнительных условий при разделении сетки.

Так, сбалансированное распределение вычислительной нагрузки при выполнении *Шагов 1–4* зависит не от общего числа узлов $|\hat{V}_i|$ в подобластях Ω_i, а от числа внутренних $|\hat{V}_{I_i}|$ и внешних $|\hat{V}_{B_i}|$ узлов в подобластях.

Вместе с тем сетки для трёхмерных областей с развитой поверхностью (пружины, тонкие пластины и оболочки) содержат большое число конечных элементов, у которых все узлы принадлежат границе расчётной области. Разделение дуальных графов таких сеток и выполнение условия $|V_i| \approx |V_j|$, $i \neq j$, приводит к подобластям Ω_i, на которых $|\widehat{V}_{I_i}| = 0$. Наглядным примером является задача моделирования напряженно-деформированного состояния пружины, рассматриваемая в настоящей работе. Геометрия пружины аппроксимируется неструктурированной расчетной сеткой (см. рис. 1, *а*) с ячейками в виде тетраэдров, число ячеек $|V| = 174264$, число узлов $|\widehat{V}| = 40743$. Для получения подобластей Ω_i, на которых $|\widehat{V}_{I_i}| > 0$, дуальный граф $G(V, E, W)$ полагался взвешенным с множеством весов $W = \{w_k\}$. Если хотя бы один из узлов конечного элемента не лежит на $\partial\Omega$, то вес соответствующей вершины графа $w_k = 3$, в противном случае $w_k = 1$. Проведены вычислительные эксперименты с различным числом подобластей ($n_\Omega = 16, 32, \ldots, 1024$). Отметим, что число подобластей увеличилось в 64 раза,



а) *б)* *в)*

Рис. 1. Сетка: *а)* исходная; *б)* разделенная на 16 подобластей; *в)* разделенная на 1024 подобласти с помощью двухуровневого разделения

а размер матрицы S_{BB} – только в 1.44 раза ($N = 66030$ в случае $n_\Omega = 16$ и $N = 95523$ для $n_\Omega = 1024$). При этом число ненулевых элементов матрицы дополнения Шура уменьшилось в 18 раз (с $Nnz \approx 2.7 \cdot 10^8$ в случае 16 подобластей, до $Nnz \approx 1.5 \cdot 10^7$ – для 1024 подобластей), как следствие, уменьшилась её заполненность с 6.11% до 0.16%. В среднем примерно каждый шестнадцатый элемент в строке S_{BB} является ненулевым (4041 из 66030) при $n_\Omega = 16$ (см. рис. 1, *б)* и примерно каждый шестисотый (154 из 95523) – при разделении на 1024 подобласти (см. рис. 1, *в*).

Важно отметить, что размер системы для дополнения Шура меньше размера исходной конечноэлементной системы (в рассмотренных случаях в 1.4–2.0 раза). При этом заполненность матрицы S_{BB} на один-два порядка больше, чем заполненность глобальной матрицы жесткости (0.03%).

3. Формирование матрицы дополнения Шура

Одной из самых затратных операций формирования дополнения Шура является обращение матрицы A_{II}. Обычно на практике для нахождения обратной матрицы используются плохо распараллеливаемые прямые методы, например метод обращения на основе LL^T-разложения.

Рассматриваемый в работе алгоритм вычисления обратной матрицы состоит из решений матричной системы вида $A_{II}X = E$, где E – единичная матрица $n_I \times n_I$. Система эффективно решается на GPU предобусловленным алгоритмом сопряженных градиентов (см. следующий параграф). Если в правой части системы вместо матрицы E брать A_{IB}, то её решением будет матрица $A'_{IB} = A_{II}^{-1}A_{IB}$. Такое представление позволяет заменить операции обращения матрицы и матричного произведения на решение n_B систем, каждая из которых решается независимо, что позволяет использовать одновременно несколько GPU. Дополнение Шура, или матрица граничных жесткостей, вычисляется по соотношениям (4), где $A_{BB} \in$ $\in \mathbb{R}^{n_B \times n_B}$, $A_{II} \in \mathbb{R}^{n_I \times n_I}$, $A_{BI} \in \mathbb{R}^{n_B \times n_I}$, $A_{IB} \in \mathbb{R}^{n_I \times n_B}$.

Пусть $\widetilde{n}_B^i$ – количество столбцов матрицы A_{BB} пересылаемых на i-й GPU, $\widetilde{A}_{BB}^i \in \mathbb{R}^{n_B \times \widetilde{n}_B^i}$ – матрица, состоящая из столбцов матрицы A_{BB} с номерами от $\sum_{j=0}^{i} \widetilde{n}_B^j$ до $\sum_{j=0}^{i+1} \widetilde{n}_B^j$, где i – номер GPU. Каждый графический ускоритель решает $\widetilde{n}_B^i$

Табл. 1

Время формирования дополнения Шура, мин

n_Ω	CPU	1 GPU	2 GPU	4 GPU	6 GPU	8 GPU
16	26:23.6	31:52.6	19:25.5	13:09.9	10:57.6	09:49.4
32	08:00.6	19:33.9	11:01.8	06:41.7	05:14.0	04:26.5
64	02:25.1	11:18.8	06:18.2	03:44.8	02:54.7	02:29.0
128	–	–	03:57.2	02:25.8	01:58.5	01:42.5
256	–	–	03:14.0	02:01.0	01:37.7	01:26.0
512	–	–	02:48.3	01:45.7	01:26.0	01:15.4
1024	00:18.7	03:53.4	02:24.0	01:32.2	01:17.5	01:08.0

систем $A_{II}a^k = a^k_{IB}$, здесь a^k_{IB} – k-й столбец матрицы A_{IB}, $k \in \left[\sum\limits_{j=0}^{i} \widetilde{n}^j_B, \sum\limits_{j=0}^{i+1} \widetilde{n}^j_B\right]$, $A^{'}_{IB} = \{a^1, a^2 \dots a^{n_B}\}$. В реализации подхода независимого решения систем уравнений на нескольких графических ускорителях используется технология OpenMP. Для этого создаются несколько нитей (число которых равно количеству доступных устройств GPU), определяется номер нити и каждой назначается графический ускоритель с тем же номером.

Ниже представлен **Алгоритм 2**, реализующий этот подход.

Алгоритм 2. Параллельный алгоритм формирования дополнения Шура на каждой подобласти Ω_i (индекс i у матриц опущен)

1: Решаем систему $A_{II}A'_{IB} = A_{IB}$; {матрицы хранятся на GPU в CSR-формате, решение находится по столбцам}.
2: $S_{BB} = A_{BB} - A_{BI}A'_{IB}$; {$S_{BB}$ и результат произведения матриц – построчно, A_{BB} – CSR-формате}
3: $\widetilde{f}_B = f_B - A_{BI}x$; {$x$ – решение системы $A_{II}x = f_I$}.
4: Формируем и решаем: $S_{BB}u_B = \widetilde{f}_B$; {$S_{BB}$ формируется в DCSR на CPU, а затем копируется в CSR на GPU }.
5: Определяем $u_I = x - A'_{IB}u_B$; {x и A'_{IB} вычислены ранее и хранятся на CPU}.

На каждом GPU после решения систем остаётся матрица $(A^{'}_{IB})^i \in \mathbb{R}^{n_I \times n^i_B}$, состоящая из столбцов матрицы $A^{'}_{IB}$ с номерами от $\sum\limits_{j=0}^{i} \widetilde{n}^j_B$ до $\sum\limits_{j=0}^{i+1} \widetilde{n}^j_B$. Это позволяет без дополнительных коммуникаций выполнить оставшиеся операции (произведение и разность матриц, см. соотношения (4)) для каждой матрицы $(A^{'}_{IB})^i$ независимо на нескольких GPU, которые используются при вычислении матриц S^i_{BB}. Далее формируется глобальная матрица дополнения Шура S_{BB} (*Шаг* 4 в **Алгоритме 2**), состоящая из локальных S^i_{BB}, принадлежащих i-й подобласти.

Локальные матрицы дополнения Шура S^i_{BB} хранятся в несжатом формате. Матрица S_{BB} после формирования из локальных S^i_{BB} преобразуется из несжатого к тому формату, в котором будет наиболее удобно с ней работать на GPU, например к CSR-формату.

В табл. 1 приведены временные затраты по параллельному формированию матриц дополнения Шура в зависимости от числа подобластей и GPU. Во второй колонке приведены данные для последовательного алгоритма, выполняемого на центральном процессоре (**Алгоритм 1**). Ускорение параллельного алгоритма (**Алгоритм 2**), реализованного на GPU, по отношению к CPU, определим как $s(n_p)_{\mathrm{CPU}} = t_{\mathrm{CPU}}/t(n_p)_{\mathrm{GPU}}$, где t_{CPU} – время выполнения последовательного алгоритма, реа-

лизованного на CPU, а $t(n_p)_{\mathrm{GPU}}$ – время выполнения параллельной реализации на n_p GPU. Аналогичным образом введём ускорение $s(n_p)_{\mathrm{GPU}} = t(1)_{\mathrm{GPU}}/t(n_p)_{\mathrm{GPU}}$. В рассмотренных вариантах максимальное ускорение составляет $s(8)_{\mathrm{GPU}} = 2.7$ и достигается при числе подобластей $n_\Omega = 16$, при этом в каждой подобласти находится в среднем примерно по 11000 ячеек сетки. Формирование матрицы дополнения Шура на двух GPU приводит к ускорению $s(2)_{\mathrm{GPU}} > 1.5$ в зависимости от числа подобластей. Использование восьми графических ускорителей даёт наименьшее время выполнения для реализации на GPU, но для задач с небольшим числом ячеек в каждой подобласти (< 2500), CPU-реализация оказывается эффективнее. В этом случае при решении большого числа систем уравнений малой размерности для каждой подобласти требуется время на копирование данных между GPU и CPU, которое становится большим, чем время решения систем. Для одного GPU (при использовании нескольких) затраты на решение сокращаются, но не покрывают затрат на инициализацию и копирование.

Полученные данные позволяют применять рассматриваемый алгоритм формирования для обеспечения более сбалансированного использования GPU и CPU в гибридных вычислительных системах.

4. Решение интерфейсной системы уравнений на GPU

Матрица дополнения Шура $S_{BB} \in \mathbb{R}^{N\times N}$ (далее $S = [s_{ij}]$) имеет порядок и обусловленность меньше, чем у исходной матрицы, и является симметричной, положительно определённой и разреженной. Решение интерфейсной системы линейных алгебраических уравнений вида (4) и систем уравнений из предыдущего раздела выполняется предобусловленным методом сопряженных градиентов.

В большинстве работ, посвящённых реализации итерационных методов на GPU, рассматривается предобуславливатель Якоби или его блочный аналог. Оптимальным выбором для вычислений на GPU нам представляются предобуславливатели, для которых предполагается, что известна аппроксимация обратной матрицы системы [11]. В этом случае дополнительные операции, вызванные переходом к предобусловленной системе, сводятся к матрично-векторному произведению $z_{k+1} = \overline{M} r_{k+1}$.

В настоящей работе рассматриваются методы сопряжённых градиентов с диагональным предобуславливателем (DIAG), предобуславливанием по методу симметричной последовательной верхней релаксации (SSOR) и предобуславливателем, построенным масштабированием системы (DIP). К сожалению, в наших экспериментах предобуславливатели, основанные на аппроксимации обратной матрицы с использованием дополнения Шура, не показали сравнимую эффективность вычислений на рассматриваемых матрицах.

Диагональный предобуславливатель имеет вид:

$$\overline{M} = M^{-1} = \{\overline{m}_{ij}\}_{ij=1}^{N}, \quad \overline{m}_{ij} = \begin{cases} s_{ij}^{-1}, & \text{если } i = j; \\ 0 & \text{в остальных случаях.} \end{cases}$$

Определим предобуславливатель SSOR следующим образом. Пусть матрица системы представима в виде $S = L + D + L^T$, тогда

$$M = KK^T, \quad K = \frac{1}{\sqrt{2-\omega}}\,\widetilde{D}(I + \widetilde{D}^{-1}L)\widetilde{D}^{-1/2},$$

или

$$K^{-1} = \sqrt{2-\omega}\,\widetilde{D}^{1/2}(I + \widetilde{D}^{-1}L)^{-1}\widetilde{D}^{-1},$$

Алгоритм 3. Алгоритм метода сопряжённых градиентов с предобуславливателем

$S, \overline{M} \in \mathbb{R}^{N\times N}$ {$\overline{M}$ формируется на GPU, матрицы хранятся в CSR-формате}
$u, r, p, q, z \in \mathbb{R}^N$ {Вектора хранятся в памяти GPU, копии на CPU нет}
  $r_0 \leftarrow f$ {копирование векторов осуществляется с помощью cublasDcopy}
$u_0 \leftarrow 0$ {инициализация выполняется на GPU}
$z_0 \leftarrow \overline{M} r_0$ {выполняется на GPU}
$p_0 \leftarrow z_0$ {копирование векторов осуществляется с помощью cublasDcopy}
$\rho_0 \leftarrow (r_0, z_0)$ {здесь и далее $(\cdot,\cdot) = \sum\limits_P (\cdot,\cdot)^{(P)}$}
while $||r_i||_2/||b||_2 > \varepsilon$ **do**
  $q_i \leftarrow S p_i$ {выполняется на GPU}
  $\alpha_i \leftarrow (r_i, z_i)/(q_i, p_i)$ {вычисляется с помощью функции cublasDdot}
  $u_{i+1} \leftarrow u_i + \alpha_i p_i$ {операция выполняется с помощью функции cublasDasxpy}
  $r_{i+1} \leftarrow r_i - \alpha_i q_i$ {операция выполняется с помощью функции cublasDasxpy}
  $z_{i+1} \leftarrow \overline{M} r_{i+1}$ {выполняется на GPU}
  $\rho_{i+1} \leftarrow (r_{i+1}, z_{i+1})$ {вычисляется с помощью функции cublasDdot}
  $\beta_{i+1} \leftarrow \rho_{i+1}/\rho_i$
  $p_{i+1} \leftarrow z_{i+1} + \beta_{i+1} p_i$ {последовательное использование функций cublasDscal и cublasDasxpy}
end while

где $0 < \omega < 2$, $\widetilde{D} = (1/\omega)D$. Приближенно обратную матрицу вычислим, ограничиваясь первым членом в разложении

$$\overline{K} = K^{-1} \approx \sqrt{2-\omega}\,\widetilde{D}^{-1/2}(I - L\widetilde{D}^{-1}). \tag{5}$$

Тогда предобуславливатель SSOR примет вид

$$\overline{M} = \overline{K}^T \overline{K}.$$

Если положить $\omega = 1$ в (5), получим предобуславливатель вида

$$\overline{K} = D^{-1/2}(I - LD^{-1})$$

и, соответственно,

$$\overline{M} = D^{1/2}\overline{K}\overline{K}^T D^{1/2}.$$

При построении предобуславливателя DIP масштабируем исходную систему:

$$\widetilde{S} = D^{-1/2} S D^{-1/2}; \quad \widetilde{f} = D^{-1/2} f; \quad \widetilde{u} = D^{1/2} u;$$

тогда предобуславливатель примет вид

$$\overline{M} = (I - \widetilde{L}^T)(I - \widetilde{L}).$$

Решение интерфейсной системы методом сопряжённых градиентов распараллелено с помощью технологии CUDA для вычисления на GPU. Все вспомогательные массивы, в частности r, p, q, z, а также матрица системы, предобуславливатель, вектор правых частей и вектор решения хранятся в памяти графического ускорителя (см. **Алгоритм 3**). После завершения работы метода сопряжённых градиентов, массив u, в котором хранится приближение вектора решения, копируется в память CPU.

Для реализации операций суммы, скалярного произведения, копирования векторов и умножения вектора на скаляр использовались функции, предоставляемые

библиотекой CUBLAS. При выполнении матрично-векторного произведения, вектор хранится в текстурной памяти, которая кэшируется, что даёт более быстрый доступ и уменьшает временные затраты. Для вычисления каждой координаты вектора результата используется от 2 до 32 потоков в зависимости от разреженности матрицы. Вычисление предобуславливателя выполняется либо на GPU, либо на CPU в зависимости от типа предобуславливателя.

Предобуславливатель SSOR вычисляется на центральном процессоре. Разреженность матрицы K равна разреженности матрицы S, и поэтому выгоднее её также хранить в компактном формате. Теоретически метод сопряжённых градиентов решает систему не более чем за N итераций. Пусть система решается за $\widetilde{N}$ итераций, значит, при вычислении $\overline{K}^T\,\overline{K}$ требуется N матрично-векторных произведений плюс $\widetilde{N}$ матрично-векторных произведений $z_{k+1} = \overline{M} r_{k+1}$ для решения системы методом сопряжённых градиентов. Если не вычислять $\overline{M}$ явно, а произведение $z_{k+1} = \overline{M} r_{k+1}$ заменить на два следующих друг за другом матрично-векторных произведения $\widetilde{z}_{k+1} = \overline{K} r_{k+1}$ и $z_{k+1} = \overline{K}^T \widetilde{z}_{k+1}$, то нам потребуется всего $2\widetilde{N}$ матрично-векторных произведений, что меньше, чем $N + \widetilde{N}$ – при нахождении матрицы $\overline{M}$ в явном виде. Поэтому при реализации метода сопряжённых градиентов с SSOR-предобуславливателем произведение $z_{k+1} = \overline{M} r_{k+1}$ заменено на $z_{k+1} = \overline{K}^T\,\overline{K} r_{k+1}$.

Масштабирование системы и вычисление предобуславливателя DIP выполняется на GPU. Как и в случае с SSOR, разреженности матриц совпадают, и они также хранятся в разреженном формате. Произведение $z_{k+1} = \overline{M} r_{k+1}$ также заменяем на два последовательных матрично-векторных произведения $y = (I - \widetilde{L}) r_{k+1}$ и $z_{k+1} = (I - \widetilde{L}^T) y$, выполняемых на GPU.

Применение предобуславливателя SSOR сокращает число итераций при решении системы в полтора раза (оптимальный параметр релаксации для данной задачи $\omega = 0.5$), что позволяет покрыть затраты на его создание и выполнение дополнительного матрично-векторного произведения. В результате этого нет выигрыша по времени вычислений в сравнении с диагональным предобуславливателем. Аналогичные результаты получены для систем разрывного метода Галёркина [11].

Однако отличие в свойствах матрицы S и матриц систем разрывного метода Галёркина позволило значительно уменьшить число итераций при использовании предобуславливателя DIP, которое стало примерно равным их числу в случае использования предобуславливателя SSOR.

Для решения интерфейсной системы (4) реализован блочный алгоритм сопряженных градиентов, использующий n_p графических ускорителей. При разделении на блоки S_k, $k = 1, \ldots, n_p$, матрице S системы (4) соответствовал граф, имеющий число вершин, равное размерности S. Каждой вершине графа матрицы S, на основе вычисленного разделения многоуровневым алгоритмом [12], ставится в соответствие номер графического ускорителя (далее в терминах раскрашивания графов – «цвет»). Вершины раскрашенного графа подразделяются на внутренние и граничные (связанные хотя бы с одной вершиной, имеющей другой «цвет»).

На основе полученного разделения в каждом блоке S_k выделялось несколько матриц: $S_k^{[i_k,i_k]}$ – матрица, элементы которой связывают внутренние вершины в блоке; $S_k^{[i_k,b_k]}$, $S_k^{[b_k,i_k]}$ – матрицы, связывающие внутренние вершины с граничными; $S_k^{[b_k,b_m]}$ – матрица, связывающая граничные вершины k-го блока с граничными вершинами m-го блока. Здесь $k \neq m$ и $k, m \in [1, n_p]$, где n_p – число блоков.

Табл. 2

Время решения интерфейсной системы, ч:мин:с

n_Ω	CPU	1 GPU	2 GPU	4 GPU	6 GPU	8 GPU
16	$>$ 8:00:00	0:15:12	0:07:01	0:03:25	0:04:41	0:04:24
32	$>$ 8:00:00	0:16:23	0:10:30	0:07:10	0:05:17	0:04:34
64	5:01:07	0:04:47	0:02:54	0:01:38	0:01:22	0:01:12
128	–	–	0:02:07	0:01:18	0:01:06	0:01:00
256	–	–	0:01:46	0:01:10	0:01:01	0:00:56
512	–	–	0:01:25	0:01:03	0:00:56	0:00:53
1024	1:48:22	0:01:09	0:01:16	0:00:59	0:00:54	0:00:52

Запишем матрицу S в виде

$$S = \begin{pmatrix} S_1^{[i_1,i_1]} & S_1^{[i_1,b_1]} & \cdots & 0 & 0 & \cdots & 0 & 0 \\ S_1^{[b_1,i_1]} & S_1^{[b_1,b_1]} & \cdots & 0 & S_1^{[b_1,b_k]} & \cdots & 0 & S_1^{[b_1,b_{n_p}]} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & S_k^{[i_k,i_k]} & S_k^{[i_k,b_k]} & \cdots & 0 & 0 \\ 0 & S_k^{[b_k,b_1]} & \cdots & S_k^{[b_k,i_k]} & S_k^{[b_k,b_k]} & \cdots & 0 & S_k^{[b_k,b_{n_p}]} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & S_{n_p}^{[i_{n_p},i_{n_p}]} & S_{n_p}^{[i_{n_p},b_{n_p}]} \\ 0 & S_{n_p}^{[b_{n_p},b_1]} & \cdots & 0 & S_{n_p}^{[b_{n_p},b_k]} & \cdots & S_{n_p}^{[b_{n_p},i_{n_p}]} & S_{n_p}^{[b_{n_p},b_{n_p}]} \end{pmatrix}.$$

При выполнении матрично-векторного произведения матрицы S на вектор $p^T = \left(p_1^i, p_1^b, \ldots, p_k^i, p_k^b, \ldots, p_{n_p}^i, p_{n_p}^b\right)$ на каждом GPU вычисляются два подвектора

$$q_k^b = S_k^{[b_k,i_k]} p_k^i + \sum_{m=1}^{m \leqslant n_p} S_k^{[b_k,b_m]} p_m^b; \quad q_k^i = S_k^{[i_k,i_k]} p_k^i + S_k^{[i_k,b_k]} p_k^b, \tag{6}$$

где k – номер GPU. Такая реализация матрично-векторного произведения позволяет снизить затраты, связанные с обменом между блоками на каждой итерации метода сопряженных градиентов, так как для выполнения последующих операций с векторами требуется обмен q_k^b, которые при минимизации границ имеют размер гораздо меньший, чем размер q.

Решение СЛАУ методом сопряженных градиентов требует меньше временных затрат в случае алгоритма, использующего GPU (см. табл. 2), при этом ускорение составляет $s(1)_{\text{CPU}} = 72$ при разделении на 16 подобластей, и $s(1)_{\text{CPU}} = 94$ при разделении на 1024 подобласти. Использование нескольких GPU даёт максимальные ускорения $s(8)_{\text{CPU}} = 251$ для $n_\Omega = 64$ и $s(8)_{\text{GPU}} = 3.5$ для $n_\Omega = 16$. С увеличением числа подобластей количество ненулевых элементов в полученной матрице дополнения Шура уменьшается, это приводит к тому, что эффективность использования нескольких GPU для решения интерфейсной системы снижается. Так, например, при разделении на 1024 подобласти $s(8)_{\text{GPU}} = 1.3$. *Шаг 6* в **Алгоритме 1**, отвечающий за нахождения решений на внутренних узлах, также выполняется на GPU. Произведение $A_{II}^{-1} f_I$ уже вычисленно на *Шаге 4*, матрица A'_{IB} получена на *Шаге 2*. Поэтому необходимо сделать лишь две операции: матрично-векторное произведение и разность векторов, которые выполняются на GPU.

В ходе проведения вычислительных экспериментов минимальные значения суммарного времени формирования и решения системы для дополнения Шура (4) получены при $n_\Omega = 1024$ (см. табл. 1 и 2). При выполнении этих шагов только

центральным процессором потребовался 1 ч 48 мин. В случае использования только одного GPU для формирования и решения СЛАУ затраты сократились в 22 раза. Минимальное время вычислений на графических ускорителях получено на восьми GPU: $t(8)_{\mathrm{GPU}} = 2$ мин. Формирование системы (4) на центральном процессоре и решение на одном графическом ускорителе выполнено за полторы минуты. Наименьшие суммарные затраты потребовались при формировании системы на CPU и её решении на восьми GPU.

Полученные результаты показывают, что при решении задач с помощью метода, использующего дополнения Шура, оптимальный выбор алгоритма зависит от числа и размера подобластей, на которые разбивается расчётная сетка. Если в одной подобласти находится относительно небольшое число ячеек сетки (< 5000), то для формирования матрицы дополнения Шура эффективнее использовать прямые методы нахождения обратных матриц и, как следствие, задействовать только CPU. При больших размерах подобластей наиболее эффективны итерационные алгоритмы, использующие для вычислений несколько GPU. Решение системы уравнений дополнений Шура эффективнее производить на GPU. В этом случае время решения сокращается в десятки и сотни раз.

Метод дополнения Шура следует применять для обеспечения более сбалансированного использования GPU и CPU на гибридных вычислительных системах при решении СЛАУ и в других приложениях.

Отметим некоторые из возможных обобщений предложенных алгоритмов. Все представленные в работе алгоритмы обобщаются без изменений на случай, когда подматрица A_{22} в (1) является нулевой. Системы такого типа возникают, например, при использовании смешанных аппроксимаций для численного решения эллиптических уравнений второго порядка. В случае несимметричной матрицы дополнения Шура при решения системы (4) может применяться метод бисопряженных градиентов с матрично-векторными произведениями в виде (6). Для обращения несимметричных подматриц A_{11} в рамках **Алгоритмов 1**, **2** разложение Холесского необходимо заменить на LU-факторизацию. Для систем уравнений, в которых блок A_{11} вырожден, предложенный алгоритм обращения может быть заменён на псевдообращение Мура–Пенроуза, остальные реализации матричных операций на CPU или GPU остаются без изменений.

Вычислительные эксперименты выполнены на узлах гибридного кластера «Уран» ИММ УрО РАН, каждый из которых содержит два процессора Intel Xeon E5675, восемь графических ускорителей NVIDIA Tesla M2090.

Работа выполнена в рамках программы Президиума РАН № 18 при поддержке УрО РАН (проект 12-П-1-1005) и РФФИ (проект № 11-01-00275-а).

Summary

S.P. Kopysov, I.M. Kuzmin, N.S. Nedozhogin, A.K. Novikov. Parallel Algorithms for Constructing and Solving the Schur Complement on Graphics Accelerators.

The paper deals with a parallel algorithm for computing the Schur complement on multiple GPU. The implementation of a parallel subdomain is shown at the stage of constructing the Schur complement matrices. An algorithm for matrix inversion is presented by the solution of the matrix system for multiple parallel streams. The realization of the matrix-vector product by means of the matrix decomposition algorithm is described for a parallel conjugate gradient method proposed for the interface system solution.

Key words: Schur complement, parallel computing, preconditioned conjugate gradient method, graphics accelerators.

Литература

1. *Haynsworth E.V.* On the Schur Complement // Basel Math. Notes. – 1968. – No 20. – 17 p.
2. *Przemieniecki J.S.* Theory of Matrix Structural Analysis. – N. Y.: McGaw-Hill, 1968. – 480 p.
3. *Постнов В.А.* Метод суперэлементов в расчетах инженерных сооружений. – Л.: Судостроение, 1979. – 288 с.
4. *Фаддеев Д.К., Фаддеева В.Н.* Вычислительные методы линейной алгебры. – М.: Физматгиз, 1960. – 656 с.
5. *Giraud L., Haidar A., Saad Y.* Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D // Numer. Math. – 2010. – V. 3. – P. 276–294.
6. *Rajamanickam S., Boman E.G., Heroux M.A.* ShyLU: A Hybrid-Hybrid Solver for Multicore Platforms // IEEE 26th Int. Parallel and Distributed Processing Symposium (IPDPS), 21–25 May 2012. – P. 631–643.
7. *Корнеев В.Г., Енсен С.*, Эффективное предобуславливание методом декомпозиции области для p-версии с иерархическим базисом // Изв. вузов. Матем. – 1999. – № 5. – С. 37–56.
8. *Копысов С.П., Красноперов И.В., Рычков В.Н.* Объектно-ориентированный метод декомпозиции области // Вычислительные методы и программирование. – 2003. – Т. 4, № 1. – С. 176–193.
9. *Kopysov S.P., Krasnopyorov I.V., Novikov A.K., Rychkov V.N.* Parallel Distributed Object-Oriented Framework for Domain Decomposition // Domain Decomposition Methods in Science and Engineering. – Springer, 2005. – V. 40. – P. 605–614.
10. *Копысов С.П.* Оптимальное разделение области для параллельного метода подструктур // Сеточные методы для решения краевых задач и приложения. Материалы Пятого Всерос. семинара. – Казань: Изд-во Казан. ун-та, 2004. – С. 121–124.
11. *Копысов С.П., Новиков А.К., Сагдеева Ю.А.* Решение систем уравнений метода Галёркина с разрывными базисными функциями на графическом ускорителе // Вестн. Удмурт. ун-та. Математика. Механика. Компьютерные науки. – 2011. – № 3. – С. 137–147
12. *Karypis G., Kumar V.* Parallel multilevel k-way partitioning scheme for irregular graphs // SIAM Rev. – 1999. – V. 41, No 2. – P. 278–300.

Поступила в редакцию
18.06.12

Копысов Сергей Петрович – доктор физико-математических наук, профессор, заведующий лабораторией Института механики УрО РАН, г. Екатеринбург.
E-mail: *s.kopysov@gmail.com*

Кузьмин Игорь Михайлович – младший научный сотрудник Института механики УрО РАН, г. Екатеринбург.
E-mail: *imkuzmin@gmail.com*

Недожогин Никита Сергеевич – аспирант Института механики Уро РАН, г. Екатеринбург.
E-mail: *Nedozhogin@inbox.ru*

Новиков Александр Константинович – кандидат физико-математических наук, старший научный сотрудник Института механики УрО РАН, г. Екатеринбург.
E-mail: *sc_work@mail.ru*