УДК 519.63

Параллельные алгоритмы метода дополнения Шура в программной модели CUDA+OpenMP

С. П. Копысов[1], И. М. Кузьмин[2], Н. С. Недожогин[3], А. К. Новиков[4]

[1] s.kopysov@gmail.ru, [3] Nedozhogin@inbox.ru

ФГБУН «Институт механики Уральского отделения РАН» (ИМ УрО РАН)

Поступила в редакцию 04.03.2013

Аннотация. Эффективное применение метода дополнения Шура на гибридных (CPU/GPU) архитектурах связано с распределением вычислений между центральным процессором и графическими ускорителями. Показано, что формирование матриц дополнения Шура может эффективно выполняться на графическом ускорителе для матриц, состоящих из нескольких тысяч строк и столбцов. Для решения интерфейсной системы предложен параллельный алгоритм метода сопряженных градиентов с явным предобуславливателем, позволяющий достигать существенного ускорения вычислений на нескольких GPU.

Ключевые слова: метод дополнения Шура, параллельные алгоритмы, метод сопряженных градиентов, графический ускоритель.

ВВЕДЕНИЕ

В настоящее время метод дополнения Шура [1, 2] принято рассматривать как гибридный метод решения систем линейных алгебраических уравнений (СЛАУ) [3, 4], сочетающий преимущества как прямых, так и итерационных методов, и учитывающий различные архитектуры параллельных вычислительных систем. На основе дополнения Шура [5] строятся эффективные предобуславливатели для решения СЛАУ различных видов [6]. Раскрыть преимущества данного метода как гибридного позволяют вычисления на CPU/GPU архитектурах. Эффективное применение графических ускорителей, тем более нескольких, в методе дополнения Шура связано с распределением вычислений между CPU и GPU, и соответствующей декомпозицией матриц.

Появление программного обеспечения для вычислений общего назначения на графических устройствах (GPGPU), такого как CUDA, OpenCL и интерфейса прикладного программирования OpenACC (http://www.openacc.org), позволило на операциях линейной алгебры получать ускорение вычислений в десятки и сотни раз по сравнению с центральным процессором. 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 для прямых методов (разложение матриц на треугольные множители), выполняются на GPU медленнее, чем ядрами CPU. Кроме того, графический ускоритель обладает существенно меньшим объемом собственной оперативной памяти, чем типичный современный вычислительный модуль, что приводит к необходимости использования в расчетах нескольких GPU, в том числе связанных вычислительной сетью.

Переход к параллельным вычислениям на нескольких графических ускорителях предполагает использование разных технологий параллельного программирования: CUDA – для вычислений на одном GPU, OpenMP – для распараллеливания вычислений между несколькими GPU внутри одного вычислительного модуля.

В данной работе метод дополнения Шура рассматривается применительно к системам уравнений, получаемым при решении трехмерных задач теории упругости и наследует разделение расчетной конечно-элементной сетки при формировании блочной структуры матриц системы.

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

1. МЕТОД ДОПОЛНЕНИЯ ШУРА

Рассмотрим построение дополнения Шура как один из вариантов методов декомпозиции области в виде алгоритма метода подструктур [2, 7]. Для обеспечения независимости вычислений в отдельных подобластях и последующего их взаимодействия 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 (1).

$$\Omega = \Omega_1 \bigcup \Omega_2 \bigcup ... \bigcup \Omega_{n_\Omega}, \ \Omega_i \bigcap \Omega_j = 0, \qquad \Gamma_B = \bigcup_{i=1}^{n_\Omega} \partial\Omega_i / \partial\Omega. \tag{1}$$

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 вершина графа V – это множество конечных элементов расчетной сетки, множество ребер графа E – множество смежных конечных элементов, $V_i \subset V$ – множество конечных элементов, образующих подобласть. 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 соответственно.

Система для интерфейсных узлов определяется как

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

$$S_{BB} = \sum_i^{n_\Omega} \left(A^i_{BB} - A^i_{BI} {A^i_{II}}^{-1} A^i_{IB} \right),$$

$$\tilde{f}_B = \sum_i^{n_\Omega} \left(f^i_B - A^i_{BI} {A^i_{II}}^{-1} f^i_I \right),$$

здесь S_{BB} – матрица граничных жесткостей или дополнение Шура для подобласти i, вектор $\tilde{f}_B$ – вектор правых частей.

Как правило, для расчетов задач используется усовершенствованный алгоритм метода подструктур. В его основе лежит использование свойств невырожденности и положительной определенности матриц подобластей. Для этих матриц существует разложение Холесского

$$A_{II} = L_{II} L^T_{II},$$

где L – нижняя треугольная матрица с положительными диагональными элементами. Использование разложения Холесского значительно сокращает вычислительные затраты и используемый объем оперативной памяти.

Рассмотрим реализацию последовательного алгоритма вычисления дополнения Шура ($n_\Omega > n_p$ и число процессоров $n_p = 1$) и вычислительные затраты, связанные с каждым шагом выполнения (после шага алгоритма показано число необходимых операций с учетом симметрии матриц, причем операция сложения и умножения принимается как одна).

Алгоритм 1 (Последовательный вариант дополнения Шура):

1. Выполним разложение Холесского матрицы A_{II} для соответствующей подобласти (индекс i опущен)
$$A_{II} = L_{II} L^T_{II}, \quad \left(\frac{n_I^3}{6} \right).$$
2. Вычисляем вспомогательные переменные
$$A'_{IB} = A^{-1}_{II} A_{IB}, \quad \left(n_B \cdot n_I^2 \right).$$
3. Сформируем матрицы граничной жесткости подобластей
$$S_{BB} = A_{BB} - A_{BI} A'_{IB}, \quad \left(\frac{n_I \cdot n_B^2}{2} \right).$$
4. Формируем вектор правой части
$$\tilde{f}_B = f_B - A_{BI} A^{-1}_{II} f_I, \quad \left(n_I \cdot n_B \right).$$
5. Собираем и решаем систему уравнений

$$S_{BB}\tilde{u}_B = \tilde{f}_B, \quad \left(k\left(M^{-1}S_{BB}\right) \le C\left(1+\log\left(\frac{H}{h}\right)\right)\right).$$

6. Определяем неизвестные для внутренних узлов

$$u_I = A_{II}^{-1} f_I - A'_{IB}\,\tilde{u}_B, \quad \left(n_I^2\right).$$

Здесь $n_I = m \cdot \left|\hat{V}_{I_k}\right|$, $n_B = m \cdot \left|\hat{V}_{B_K}\right|$ – 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 уравнения. В данной работе в качестве метода решения СЛАУ использовался метод сопряженных градиентов с различными предобуславливателями. Матрицы S_{BB}, A_{II} положительно определены и симметричны.

2. РЕСУРСОЕМКОСТЬ МЕТОДА ДОПОЛНЕНИЯ ШУРА

Для обеспечения эффективной работы с матрицами используются различные схемы хранения. Матрицы S_{BB}, A_{II} являются симметричными, поэтому возможно хранение только ее части (верхний или нижний треугольник с диагональю). Оценка ресурсоемкости схем хранения производится исходя из симметрии матриц. Простейший вариант – хранить матрицу, не исключая нулей. В этом случае для хранения используется массив по числу элементов матрицы N^2. Этот способ является наиболее затратным по использованию памяти, что становится существенным при решении больших задач.

Формат хранения (DCSR), используемый при вычислениях в данной работе, представляет массив, состоящий из списка упакованных строк и реализован в системе конечно-элементного анализа FEStudio [7, 8]. Каждая строка матрицы представляет структуру, состоящую из двух массивов. Первый массив хранит значения ненулевых элементов, второй – 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 в формате CSR выделяются три одномерных массива, в которых хранятся ненулевые значения $\{a_{ij} \mid a_{ij} \neq 0,\ 0 \le i, j \le N\}$, их столбцовые индексы $\{j \mid a_{ij} \neq 0,\ 0 \le i, j \le N\}$ и позиции элементов a_{i0}, $0 \le i \le N$ в двух первых массивах.

Сравним ресурсоемкость предложенных схем хранения матриц по следующим параметрам: занимаемая память, алгоритмическая сложность доступа к элементу и его добавления. Оценка показывает, что алгоритмические сложности доступа к элементу $O(2N_\beta + 1)$ и его добавления для ленточного формата составляет $O((2N_\beta + 1)N)$, для DCSR – O(Nnz*) и $O(Nnz^*)$ соответственно, а для формата CSR – $O(Nnz^*)$ и $O(Nnz)$, здесь

$$Nnz^* = \max_{i=1}^{N}\left\{Nnz_i\right\}.$$

Необходимый объем памяти для ленточного формата – $(8(2N_\beta + 1)N)$, для формата DCSR –

$$\sum_{i=1}^{N}\left(12Nnz_i + 4\right),$$

для формата CSR – $(12Nnz + 4(N + 1))$ байт. В этом случае 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 матрицы 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 $O(Nnz^*)$, чем в формате CSR – $O(Nnz)$. Кроме того, из представленных форматов, необходимых для DCSR, объем памяти является минимальным.

Преимущества ленточного формата по сложности доступа и добавления элемента компенсируются необходимым объемом памяти $(8(2N_\beta + 1)N)$ и частотой доступа к элементам матрицы O$((2N_\beta + 1)N)$, вместо $O(Nnz)$ для форматов DCSR и CSR. Здесь

$$N_\beta = \max_{i=1}^{N}\left\{\max_{j=1}^{N}\left\{j - i \mid a_{ij} \neq 0\right\}\right\}$$

– 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 [9], схема хранения оказывает существенное влияние на время вычислений. Отметим, что для ленточных матриц время формирования дополнения Шура составляет порядка 80 %. Переход на формат DCSR для матриц A_{BB}, A_{IB}, A_{II} в одной и той же задаче дал сокращение затрат в четыре раза. Таким образом, в последовательном варианте наиболее эффективным с точки зрения ресурсоемкости и алгоритмической сложности представляется использовать метод 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 и далее, в ядре CUDA, выполнения вычислений над строкой и возвращения результата в оперативную память или кэш CPU. Таким образом, потребуется $2N$ выделений памяти и копирований массивов размера Nnz_i, вместо выделения памяти и копирования двух массивов размера Nnz, поэтому для вычислений на GPU матрица переводится в CSR формат.

3. ПАРАЛЛЕЛЬНЫЙ МЕТОД ДОПОЛНЕНИЯ ШУРА И ЕГО ЭФФЕКТИВНОСТЬ

В методе дополнения Шура можно реализовать два уровня распараллеливания: первый уровень связан с разделением вычислений между подобластями [8, 9], второй – с параллельной реализацией методов, которые используются для формирования 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.

Рассмотрим Алгоритм 1, исходя из обозначенных вариантов распараллеливания. Как видно, шаги 1–4, 6 выполняются для каждой подобласти независимо, что говорит о естественном параллелизме, который обеспечивает первый уровень распараллеливания – на уровне области.

Наиболее трудоемкими шагами алгоритма являются: процессы формирования матрицы дополнения Шура – шаги 1–3, вектора правых частей – шаг 4, а так же решение системы – шаг 5. Вычисления внутри каждой подобласти выполняются независимо от других подобластей, значит, становится возможной их параллельная реализация.

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 (3)), т. е. 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.

Сбалансированное распределение вычислительной нагрузки при выполнении шагов 1–4 зависит не от общего числа узлов $\left|\hat{V}_i\right|$ в подобластях Ω_i, а от числа внутренних $\left|\hat{V}_{I_i}\right|$ и внешних $\left|\hat{V}_{B_I}\right|$ узлов в подобластях. Вместе с тем сетки для трехмерных областей с развитой поверхностью (пружины, тонкие пластины и оболочки) содержат большое число конечных элементов, в которых все узлы принадлежат границе расчетной области. Разделение дуальных графов таких сеток и выполнение условия

$$\left|\hat{V}_i\right| \approx \left|\hat{V}_j\right|, \quad \forall i \neq j$$

приводит к подобластям $\Omega_i : \left|\hat{V}_{I_i}\right| = 0.$

Наглядным примером является задача моделирования напряженно-деформированного со-

стояния пружины, рассматриваемая в данной работе.

Геометрия пружины аппроксимируется неструктурированной расчетной сеткой (рис. 1) с ячейками в виде тетраэдров, число ячеек $|V|$ = =174264, число узлов $\left|\hat{V}\right|$ = 40743. Для получения подобластей $\Omega_i : \left|\hat{V}_{I_k}\right| > 0$, дуальный граф полагался взвешенным *G(V,E,W)*, с множеством весов $W = \{\omega_k\}$. Если хотя бы один из узлов конечного элемента не лежит на $\partial\Omega$, то вес соответствующей вершины графа $\omega_k = 3$, в другом случае $\omega_k = 1$.

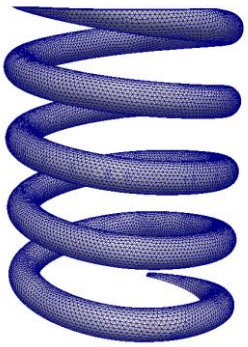
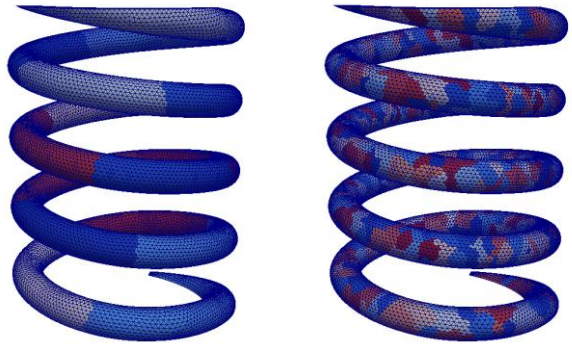


Рис. 1. Исходная сетка



а *б*

Рис. 2. Разделенная сетка: *а* – на 16 подобластей; *б* – на 1024 подобласти

Проведенные вычислительные эксперименты с различным числом подобластей (n_Ω = 16, 32, …, 1024) показали, что увеличение числа подструктур приводит к увеличению размера матрицы дополнения Шура. Отметим, что число подобластей увеличилось в 64 раза, а размер матрицы S_{BB} только в 1.44 раза (N = 66030 в случае n_Ω = 16, и N = 95523 – для n_Ω = 1024). Вместе с тем число ненулевых элементов матрицы дополнения Шура уменьшилось в 18 раз (с $Nnz \approx 2{,}7\cdot10^8$ в случае 16 до $Nnz \approx 1{,}5\cdot10^7$ – для 1024 подобластей), как следствие, уменьшилось ее заполненность с 6.11 % до 0.16 %. В среднем примерно каждый шестнадцатый элемент в строке S_{BB} является ненулевым (4041 из 66030), при n_Ω = 16 подобластей (рис. 2, *а*), и примерно каждый шестисотый (154 из 95523) – при разделении на 1024 подобласти (рис. 2, *б*).

Важно отметить, что размер системы для дополнения Шура меньше размера исходной конечно-элементной системы (в рассмотренных случаях в 1.4–2 раза). Вместе с тем заполненность матрицы S_{BB} на один-два порядка больше, чем заполненность глобальной матрицы жесткости (0.03 %).

3.1. Формирование матрицы дополнения Шура

Одной из самых затратных операций формирования дополнения Шура является обращение матрицы A_{II}. Обычно методами нахождения обратной матрицы являются плохо распараллеливаемые прямые методы, например метод обращения матрицы 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 ее 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, и использовать одновременно несколько GPU. Дополнение Шура S_{BB} или матрица граничных жесткостей вычисляется по соотношениям (3), где

$$A_{BB} \in R^{n_B \times n_B}, A_{II} \in R^{n_I \times n_I}, A_{BI} \in R^{n_B \times n_I}, A_{IB} \in R^{n_I \times n_B}.$$

Пусть $\tilde{n}_B^j$ – количество столбцов матрицы A_{BB}, пересылаемых на i-й GPU. Каждый графический ускоритель решает $\tilde{n}_B^j$ систем вида:

$$A_{II} a^k = a_{IB}^k,$$

здесь a_{IB}^k – k-й столбец матрицы A_{IB},

$$k \in \left[\sum_{j=0}^{i} \tilde{n}_B^j , \sum_{j=0}^{i+1} \tilde{n}_B^j , \right], \quad A'_{IB} = \left\{ a^1, a^2, \ldots, a^{n_B} \right\}.$$

При реализации 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. На каждом GPU после решения систем остается матрица

$$\left(A'_{IB}\right)^i \in R^{n_I \times n_B^i}, \left(A'_{IB}\right)^i = \left\{ a^k \mid k \in \left[\sum_{j=0}^{i} \tilde{n}_B^j, \sum_{j=0}^{i+1} \tilde{n}_B^j, \right] \right\}.$$

Алгоритм 2 (параллельный алгоритм формирования дополнения Шура на каждой подобласти Ω_I (индекс i опущен):

1. Решаем систему
$$A_{II} A'_{IB} = A_{IB};$$
{матрицы хранятся на GPU в CSR формате, решение – по столбцам}
2. Сформируем матрицы граничной жесткости подобластей
$$S_{BB} = A_{BB} - A_{BI} A'_{IB};$$
{S_{BB} и результат произведения матриц – построчно, A_{BB} – CSR формате}
3. Формируем вектор правой части
$$\tilde{f}_B = f_B - A_{BI} x;$$
{x – решение системы $A_{II} x = f_I$}
4. Собираем и решаем систему уравнений
$$S_{BB} \tilde{u}_B = \tilde{f}_B;$$
{S_{BB} формируется в DCSR на CPU, а затем копируется в CSR на GPU}
5. Определяем неизвестные для внутренних узлов
$$u_I = x - A'_{IB} \tilde{u}_B.$$
{x и A'_{IB} вычислены ранее, и хранятся на CPU}

Это позволяет без дополнительных коммуникаций выполнить оставшиеся операции (произведение и разность матриц, см. соотношение (3)) для каждой матрицы $(A'_{IB})^i$ независимо на нескольких GPU, которые используются при вычислении матриц S_{BB}^i. Далее формируется глобальная матрица дополнения Шура S_{BB} (шаг 4 в Алгоритме 2), состоящая из локальных S_{BB}^i, принадлежащих i-й подобласти.

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

В табл. 1 приведено время параллельного формирования матрицы дополнения Шура в зависимости от числа подобластей и GPU. Во второй колонке представлены данные для последовательного алгоритма, выполняемого на центральном процессоре (Алгоритм 1).

Ускорение параллельного алгоритма (Алгоритм 2), реализованного на GPU, по отношению к CPU, определим как $s(n_p)_{CPU} = t_{CPU}/t(n_p)_{GPU}$, где t_{CPU} – время выполнения последовательного алгоритма, реализованного на CPU, а $t(n_p)_{GPU}$ – время выполнения параллельной реализации на n_p GPU. Аналогичным образом введем ускорение $s(n_p)_{GPU} = t(1)_{GPU}/t(n_p)_{GPU}$. В рассмотренных вариантах максимальное ускорение составляет $s(8)_{GPU} = 2.7$ и достигается при числе подобластей $n_\Omega = 16$, при этом в каждой подобласти находится в среднем по 11000 ячеек сетки.

Формирование матрицы дополнения Шура на двух GPU приводит к ускорению $s(2)_{GPU} > 1.5$, в зависимости от числа подобластей. Использование восьми графических ускорителей дает 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 требуются время на копирование данных между GPU и CPU, которое становится больше, чем время решения систем. В рамках одного GPU (при использовании нескольких) затраты на решение сокращаются, но не покрываются затрат на инициализацию и копирование.

3.2. Решение интерфейсной системы уравнений на GPU

Матрица дополнения Шура

$$S_{BB} \in R^{N \times N}, \quad S = S_{BB} = \{s_{ij}\}$$

имеет порядок и обусловленность меньше, чем у исходной матрицы и является симметричной, положительно определенной и разреженной. Решение интерфейсной системы линейных алгебраических уравнений вида (3) и систем уравнений из предыдущего параграфа выполняются предобусловленным методом сопряженных градиентов.

В большинстве работ, посвященных реализации итерационных методов на GPU, рассматривается предобуславливатель Якоби или его блочный аналог. Оптимальным выбором для вычислений на GPU представляются предобуславливатели, в которых считается, что известна аппроксимация обратной матрицы системы [10].

Таблица 1

Время формирования дополнения Шура, мин.:сек

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

В этом случае дополнительные операции, вызванные переходом к предобусловленной системе, сводятся к матрично-векторному произведению $z_{k+1} = Mr_{k+1}$.

Решение интерфейсной системы методом сопряженных градиентов распараллелено с помощью технологии CUDA для вычисления на GPU. Все вспомогательные массивы, в частности r, p, q, z, а также матрица системы, предобуславливатель, вектора правых частей и вектор решения хранятся в памяти графического ускорителя (см. Алгоритм 3). После завершения работы метода сопряженных 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 библиотеки CUBLAS.

При выполнении матрично-векторного произведения вектор хранится в текстурной памяти, которая кэшируется, что дает более быстрый доступ и уменьшает временные затраты.

Алгоритм 3 (Алгоритм метода сопряженных градиентов с предобуславливателем):

1. $S, M \in R^{N \times N}$
   {M формируется на GPU, матрицы хранятся в CSR формате}
2. $u, r, p, q, z \in R^N$
   {вектора хранятся в памяти GPU, копии на CPU нет}
3. $r_0 \leftarrow f$
   {копирование векторов осуществляется с помощью cublasDcopy}
4. $u_0 \leftarrow 0$
   {инициализация выполняется на GPU}
5. $z_0 \leftarrow Mr_0$
   {выполняется на GPU}
6. $p_0 \leftarrow z_0$
   {копирование векторов осуществляется с помощью cublasDcopy}
7. $\rho_0 \leftarrow (r_0, z_0)$
   {здесь и далее $(\cdot,\cdot)=\sum(\cdot,\cdot)^{(P)}$, выполняется с помощью функции cublasDdot}
8. **while** $||r_i||_2/||b||_2 > \varepsilon$ **do**
9. $q_i \leftarrow S\, p_i$
   {выполняется на GPU или по формуле (4)}
10. $\alpha_i \leftarrow (r_i, z_i)/(q_i, p_i)$
    {вычисляется с помощью cublasDdot}
11. $u_{i+1} \leftarrow u_i + \alpha_i\, p_i$
    {операция выполняется с помощью функции cublasDasxpy}
12. $r_{i+1} \leftarrow r_i - \alpha_i\, q_i$
    {операция выполняется с помощью функции cublasDasxpy }
13. $z_{i+1} \leftarrow M\, r_{i+1}$
    {выполняется на GPU}
14. $\rho_{i+1} \leftarrow (r_{i+1}, z_{i+1})$
    {вычисляется с помощью cublasDdot }
15. $\beta_{i+1} \leftarrow \rho_{i+1}/\rho_i$
    {вычисляется на CPU}
16. $p_{i+1} \leftarrow z_{i+1} + \beta_i\, p_i$
    {последовательное использование функций cublasDscal и cublasDasxpy}
17. **end while**

В вычислениях каждой координаты вектора результата используется от 2 до 32 потоков в зависимости от разреженности матрицы. Предобуславливатель вычисляется на GPU или на CPU в зависимости от его типа.

3.3. Особенности решения интерфейсной системы на нескольких GPU

Для решения интерфейсной системы (3) реализован блочный алгоритм метода сопряженных градиентов с распределением вычислений между n_p графическими ускорителями

средствами OpenMP. В этом случае Алгоритм 3 выполнялся в параллельной области, созданной директивой parallel. Результаты скалярных произведений в отдельных нитях OpenMP помещаются в общие переменные и суммируются при помощи директивы atomic, с последующей барьерной синхронизацией.

При разделении на блоки, матрица $S = \{s_{ij}\}$ системы (3) представлялась графом $G_S(V, E)$, где $V = \{i\}$ – множество вершин графа, образованное строчными индексами; $E = \{(i, j)\}$ – множество ребер графа, образованное парами строчных и столбцовых индексов ненулевых элементов матрицы S; число вершин графа равно размерности S. Граф G_S разделяется на n_p частей многоуровневым алгоритмом [11]. На основе вычисленного разделения, каждой вершине графа ставится в соответствие номер графического ускорителя k. Исходя из назначенного номера GPU, вершины графа подразделяются на внутренние и граничные (связанные хотя бы с одной вершиной, которой сопоставлен другой номер графического ускорителя).

На основе полученного разделения в каждом блоке S_k выделялось несколько матриц: $S_k^{[i_k,i_k]}$ – матрица, элементы которой связывают 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.

Запишем матрицу S в виде

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

При вычислении произведения матрицы S на вектор p, здесь

$$p^T = (p_1^i, p_1^b, \ldots, p_k^i, p_k^b, \ldots p_{n_p}^i, p_{n_p}^b),$$

на каждом GPU вычисляются два вектора:

$$\begin{aligned} q_k^b &= S_k^{[b_k,i_k]} p_k^i + \sum_{m=1}^{m \le n_p} S_k^{[b_k,b_m]} p_m^b; \\ q_k^i &= S_k^{[i_k,i_k]} p_k^i + S_k^{[i_k,b_k]} p_k^b, \end{aligned} \qquad (4)$$

где k – номер GPU. Данная реализация матрично-векторного произведения позволяет уменьшить затраты, связанные с обменом между блоками на каждой итерации метода сопряженных градиентов, так как для выполнения последующих операций с векторами требуется обмен векторами q_k^b, размерность которых существенно меньше, чем у вектора q.

Разработанное программное обеспечение для матрично-векторного произведения представлено в Листингах 1–2. Этот программный код является частью параллельной области программы метода сопряженных градиентов и предназначен для вычисления q_k^i и q_k^b графическим ускорителем k. Барьерная синхронизация обеспечивает получение векторов к моменту начала вычислений на CPU (листинг 1, строка 11) и GPU (там же, строка 25).

При выполнении матрично-векторного произведения, текстурная память привязывается к полю V структур *d_p_b* и *d_p_i* в функции *MatrVectVul* (Листинг 2, строка 24 и 44). Собственно произведения вычисляются kernel-функцией *dev_matrVectMul*, в которой используется текстурная память – для координат вектора p и разделяемая – для доступа к элементам матрицы и сбора промежуточных результатов.

Из приведенных в табл. 2 результатов видно, что решение интерфейсной СЛАУ методом сопряженных градиентов требует меньше временных затрат в алгоритме, использующем GPU. Получено ускорение $s(1)_{CPU} = 251$ для $n_\Omega = 64$ и $s(8)_{GPU} = 3.5$ 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 уменьшается. Например, при разделении на 1024 подобласти s(8)GPU = 1.3.

В ходе выполнения вычислительных экспериментов минимальные значения суммарного времени формирования и решения системы для дополнения Шура (3) получены при $n_\Omega = 1024$ (см. табл. 1 и 2).

Листинг 1. Фрагмент параллельной области программы метода сопряженных градиентов

```
cudaMemcpyAsync(h_p_b[cpu_thread_id].V, d_p_b.V, sizeof(double)*d_p_b.dim,
                cudaMemcpyDeviceToHost,stream1);
// вычисление q^i_k
    MatrVectMul(maxDimGrid, d_Ap_i, d_p_i, d_p_b, d_Ai, d_Aib, stream2);
// вычисление S_k^{[b_k,i_k]}*p^i_k + S_k^{[b_k,b_k]}*p^b_k
    MatrVectMul(maxDimGrid, d_Ap_b, d_p_b, d_p_i, d_Ab, d_Abi, stream2);

// вычисление q^b_k += \sum_{m=1}^{m<n_p} S_k^{[b_k,b_k]}*p^b_k
  for(i=0;i<h_Ap_b.dim;i++) h_Ap_b.V[i]=0;
#pragma omp barrier
     for(i=0;i<subdomain;i++)
      {
       if(i!=cpu_thread_id)
        {
           for(j=0; j<Abb[i].dim; j++)
             for(int l=Abb[i].NL[j]; l<Abb[i].NL[j+1]; l++)
              {
                h_Ap_b.V[j] += Abb[i].V[l]*h_p_b[i].V[Abb[i].NC[l]];
              }
        }
      }
cudaMemcpyAsync(d_Ap_b_loc.V, h_Ap_b.V, sizeof(double)*h_Ap_b.dim,
                cudaMemcpyHostToDevice, stream1);
#pragma omp barrier
    cublasDaxpy(d_Ap_b.dim, 1.0, d_Ap_b_loc.V, 1, d_Ap_b.V, 1);
```

Листинг 2. Функция, вычисляющая вектор q_k^i

```
void MatrVectMul(const int maxDimGrid,
                 Vector d_Ap, Vector d_p_i, Vector d_p_b,
                 CSRmatrix d_Ai, CSRmatrix d_Aib,
                 cudaStream_t stream)
{
    const size_t dimBlock = 128;
//      заполнение вектора q нулями
    vec_def <<< maxDimGrid, dimBlock >>> (d_Ap.V, d_Ap.dim);

    int nnz_per_row = d_Ai.NumEl / d_Ai.dim;
    unsigned int thr_per_vec;

         if(nnz_per_row <= 2)   thr_per_vec=2;
    else if(nnz_per_row <= 4)   thr_per_vec=4;
    else if(nnz_per_row <= 8)   thr_per_vec=8;
    else if(nnz_per_row <= 16)  thr_per_vec=16;
    else                        thr_per_vec=32;

    size_t vecs_per_bl = dimBlock / thr_per_vec;

    size_t DimGrid = std::min<int>(maxDimGrid,
            (d_Ai.dim + vecs_per_bl-1)/vecs_per_bl);

    cudaBindTexture(0, tex_cgb, d_p_i.V, sizeof(double)*d_p_i.dim);
//      вычисление первых слагаемых из соотношений (4)
    dev_MatrVectMul <<<DimGrid,dimBlock,0,stream>>> (d_Ap.V, d_Ai.V,
                d_Ai.NC, d_Ai.NL, d_Ai.dim,vecs_per_bl, thr_per_vec);
    cudaUnbindTexture(tex_cgb);

    nnz_per_row = d_Aib.NumEl / d_Aib.dim;

         if(nnz_per_row <= 2)   thr_per_vec=2;
    else if(nnz_per_row <= 4)   thr_per_vec=4;
    else if(nnz_per_row <= 8)   thr_per_vec=8;
    else if(nnz_per_row <= 16)  thr_per_vec=16;
    else                        thr_per_vec=32;

    vecs_per_bl = dimBlock / thr_per_vec;

    DimGrid=std::min<int>(maxDimGrid,
                (d_Aib.dim+vecs_per_bl-1)/vecs_per_bl);

    cudaBindTexture(0, tex_cgb, d_p_b.V, sizeof(double)*d_p_b.dim);
//      вычисление вторых слагаемых из соотношений (4)
    dev_MatrVectMul <<<DimGrid,dimBlock,0,stream>>> (d_Ap.V,d_Aib.V,
                d_Aib.NC,d_Aib.NL,d_Aib.dim,vecs_per_bl,thr_per_vec);
    cudaUnbindTexture(tex_cgb);
}
```

Таблица 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	0:01:10	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

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 и составляет 2 мин. Формирование системы (3) на центральном процессоре и решения на одном графическом ускорителе выполнено за полторы минуты. Наименьшие суммарные затраты потребовались при формировании системы на CPU и ее решении на восьми GPU.

Полученные результаты показывают, что при решении задач с помощью метода дополнения Шура оптимальный выбор алгоритма зависит от числа и размера подобластей, на которые делится расчетная сетка. Если в одной подобласти находится относительно небольшое число ячеек сетки (< 5000) или неизвестных (< 1500 – для внутренних и < 2500 – для граничных), 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 интерфейсной системы уравнений эффективнее производить на графических ускорителях. В этом случае время решения сокращается в десятки и сотни раз.

Следует отметить, что метод дополнения Шура позволяет сбалансированно распределить вычисления между центральным процессором и графическими ускорителями.

Представленные результаты получены при проведении вычислительных экспериментов на узлах гибридного кластера «Уран» ИММ УрО РАН, каждый из которых содержит два процессора Intel Xeon E5675, восемь графических ускорителей NVIDIA Tesla M2090.

СПИСОК ЛИТЕРАТУРЫ

1. **Haynsworth E. V.** On the Shur complement // Basel Mathematical Notes. 1968. № 20. 17 p.

2. **Przemieniecki J. S.** Theory of Matrix Structural Analusis. New York: McGaw-Hill, 1968. 480 p.

3. **Giraud L., HaidarA., Saad Y.** Sparse approximations of the Shur complement for parallel algebraic hybrid solvers in 3D // Numerical Mathematics. 2010. Vol. 3. P. 276–294.

4. **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.

5. **Фадеев Д. К., Фадеева В. Н.** Вычислительные методы линейной алгебры. М. Физматгиз, 1960. 656 с.

6. **Корнеев В. Г., Енсен С.** Эффективное предобуславливание методом декомпозиции области для *p*-версии с иерархически базисом // Известия вузов. Математика. 1999. Т. 444, № 5. С. 37–56.

7. **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.

8. **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. Vol. 40. P. 605–614.

9. **Копысов С. П.** Оптимальное разделение области для параллельного метода подструктур // Сеточные методы для решения краевых задач и приложения: сб. тр. 5-го Всерос. сем. Казань: КГУ, 2004. С. 121–124.

10. **Копысов С. П., Новиков А. К., Сагдеева Ю. А.** Решение систем уравнений метода Галеркина с разрывными базисными функциями на графическоем ускорителе // Вестник Удмуртского университета. Математика. Механика. Компьютерные науки. 2011. № 3. С. 137–147.

11. **Karypis G., Kumar V.** Parallel multilevel k-way partitioning scheme for irregular graphs // SIAM Rev. 1999. Vol. 41, no. 2. P. 278–300.

ОБ АВТОРАХ

Копысов Сергей Петрович, проф., зав. лаб. выч. и инф. технологий. Дипл. инж.-мех. (Ижевск. мех. ин-т, 1988). Д-р физ.-мат. наук (ИММ РАН, 2007). Иссл. в обл. параллельных вычислений.

Кузьмин Игорь Михайлович, мл. науч. сотр. той же лаб. Дипл. магистр математики (УдГУ, 2009). Иссл. в обл. параллельных вычислений.

Недожогин Никита Сергеевич, асп. той же лаб. Дипл. математик, сист. программист (УдГУ, 2011). Иссл. в обл. параллельных вычислений.

Новиков Александр Константинович, ст. науч. сотр. той же лаб. Дипл. инж.-мех. (ИжГТУ, 1994). Канд. физ.-мат. наук (ИММ РАН, 2005). Иссл. в обл. параллельных вычислений.

METADATA

Title: Parallel algorithms of the Shur complement method in software model CUDA+OpenMP.

Authors: S. P. Kopysov, I. M. Kuzmin, N. S. Nedozhogin, and A. K. Novikov.

Affiliation: Institute of Mechanics Ural Branch of Russian Academy of Sciences , Russia

Email: nedozhogin@inbox.ru.

Language: Russian.

Source: Vestnik UGATU (scientific journal of Ufa State Aviation Technical University), vol. 17, no. 5 (58), pp. 219-229, 2013. ISSN 2225-2789 (Online), ISSN 1992-6502 (Print).

Abstract: Implementation of the Shur complement method on the hybrid (CPU/GPU) architecture is effective because of quality of computing distribution between CPU and GPUs. We have show that the formation of the Shur complement matrix can be performed efficiently on the GPU for matrix consisting of several thousand rows and columns. To solve the interface system, we proposed parallel algorithm of the conjugate gradients method with explicit preconditioning. This helps to achieve significant acceleration of the computing on several GPUs.

Key words: Shur complement method; parallel algorithms; conjugate gradient method; GPU.

References (English Transliteration):

1. E. V. Haynsworth**, "**On the Shur complement," *Basel Mathematical Notes*, no. 20, 1968.
2. J. S. Przemieniecki**,** *Theory of Matrix Structural Analusis*. New York: McGaw-Hill, 1968.
3. L. Giraud, A. Haidar, and Y. Saad, "Sparse approximations of the Shur complement for parallel algebraic hybrid solvers in 3D," *Numerical Mathematics*. vol. 3, pp. 276-294, 2010.
4. S. Rajamanickam, E. G. Boman, and M. A. Heroux, "ShyLU: A hybrid-hybrid solver for multicore platforms," *IEEE 26th Int. Parallel and Distributed Processing Symp. (IPDPS)*, 2012, pp. 631-643.
5. D. K. Fadeev and V. N. Fadeeva, *Computational Methods of Linear Algebra,* (in Russian). Moscow: Fizmatgiz, 1960.
6. V. G. Korneev, S. Ensen, "Efficient pre-domain decomposition method for the p-version with a hierarchical basis", (in Russian), *Izvestiya Vuzov. Matematika* (Proc. Higher Education. Mathematics), vol. 444, no. 5, pp. 37-56, 1999.
7. S. P. Kopysov, I. V. Krasnopyorov, and V. N. Rychkov, "Object-oriented domain decomposition method", (in Russian), *Vychislitel'nye Metody i Programmirovanie* (Numerical Methods and Programming), vol. 4, no. 1, pp. 176-193, 2003.
8. S. P. Kopysov, I. V. Krasnopyorov, A. K. Novikov, and V. N. Rychkov, "Parallel distributed object-oriented framework for domain decomposition," *Domain Decomposition Methods in Science and Engineering*, vol. 40, pp. 605-614*,* Springer, 2005.
9. S. P. Kopysov, "Optimal separation of the parallel substructure method," (in Russian), in *Proc. 5th All-Russian seminar on "Grid methods for solving value problems and applications"*, Kazan: KGU, 2004, pp. 121-124.
10. S. P. Kopysov, A. K. Novikov, and J. A. Sagdeeva, "Solving simultaneous equations Galerkin method on a discontinuous basis functions on GPU," (in Russian), *Vestnik Udmurtskogo Universiteta. Matematika. Mechanika. Kompyuternye Nauki (*Bulletin of Udmurt State University. Mathematics. Mechanics. Computer Science), no. 3, pp. 137-147, 2011.
11. G. Karypis, V. Kumar, "Parallel multilevel k-way partitioning scheme for irregular graphs," *SIAM Rev*., vol. 41, no. 2, pp. 278-300, 1999.

About authors:

Kopysov, Sergey Petrovich, Doctor of Physics and Mathematics, Professor, Institute of Mechanics, Ural Branch of Russian Academy of Sciences.

Kuzmin, Igor Michailovich, Junior Researcher, Institute of Mechanics, Ural Branch of Russian Academy of Sciences.

Nedozhogin, Nikita Sergeyevich, Post-graduate student, Institute of Mechanics, Ural Branch of Russian Academy of Sciences.

Novikov, Alexander Konstantinovich, Candidate of Physics and Mathematics, Senior Researcher, Institute of Mechanics, Ural Branch of Russian Academy of Sciences.