ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ФОРМИРОВАНИЯ И РЕШЕНИЯ ДОПОЛНЕНИЯ ШУРА

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

Институт механики УрО РАН, Ижевск

Введение

Впервые метод дополнения Шура был математически сформулирован и описан в работах Issai Shur, а название своё получил после выхода работы [1], хотя до этого метод был известен в механике как метод подструктур и впервые встречается в работе [2]. В России этот метод известен, как метод суперэлементов [3]. Изначально он рассматривался как метод блочной декомпозиции для вычислений на системах с ограниченными вычислительными мощностями. В дальнейшем использовался для построения методов декомпозиции области без перекрытий на параллельных вычислительных системах. В настоящее время его все чаще рассматривают как гибридный решатель систем линейных алгебраических уравнений [4], сочетающий как прямые, так и итерационные методы для операций над матрицами, и который учитывает различные архитектуры вычислительных систем. В основе метода дополнения Шура лежит идея независимого расчета отдельных подобластей и последующего учета их взаимодействия. В данном методе можно реализовать два уровня распараллеливания: первый уровень связан с независимыми вычислениями между подобластями [5, 6], второй – с параллельной реализацией методов, которые используются для формирования внутри отдельной подобласти. Помимо этого, распараллеливанию поддается и решение интерфейсной системы. В представленной работе рассматривается второй уровень распараллеливания, для которого предложены алгоритмы формирования матрицы дополнения Шура, а также решение интерфейсной системы уравнений с использованием технологии CUDA и нескольких графических ускорителей (GPU). Был рассмотрен метод сопряженных градиентов с декомпозицией матрицы, направленной на минимизацию обмена данных между потоками.

1. Метод дополнения Шура

Пусть область Ω разбита на n_Ω непересекающихся подобластей:

$$\Omega = \Omega_1 \cup \Omega_2 \cup ... \cup \Omega_{n_\Omega}, \text{ где } \Omega_i \cap \Omega_j = \emptyset, \ \Gamma_B = \bigcup_{i=1}^{n_\Omega} \partial\Omega_i / \partial\Omega. \quad (1)$$

Разделение на подобласти (подструктуры) наследуется от процесса разделения дуального графа расчетной сетки $G(V,E) = \bigcup_{k=1}^{n_\Omega} G_k(V,E)$, здесь множество вершин графа V_G – это множество конечных элементов расчетной сетки, множество ребер графа E_G – множество смежных конечных элементов. Множество конечных элементов, образующих подструктуру – $V_{G_k} \subset V_G$.

Узлы расчетной сетки образуют множество $\hat{V}$ и условно разделяются на внешние $\hat{V}_{B_k}$ (принадлежат границе области) и внутренние $\hat{V}_{I_k}$, связанные с узлами подобласти

сетки, соответствующие подграфу $G_k(V,E)$. Из множества внешних узлов выделяются интерфейсные $\hat{V}_{C_k} \subset \hat{V}_{B_k}$, 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:

$$\begin{pmatrix} A_{II}^i & A_{IB}^i \\ A_{BI}^i & A_{BB}^i \end{pmatrix} \begin{pmatrix} u_I^i \\ u_B^i \end{pmatrix} = \begin{pmatrix} f_I^i \\ f_B^i \end{pmatrix}, \quad (2)$$

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

В методе дополнения Шура полагается, что все подграфы $G_k(V,E)$ связные, в противном случае система уравнений (2) для подобласти Ω_k распадается на несвязанные системы уравнений.

Система для общих граничных узлов определяется как

$$S_{BB} u_{\tilde{B}} = f_{\tilde{B}}, \quad S_{BB} = \sum_i^{n_\Omega} A_{BB}^i - A_{BI}^i A_{II}^{i\ -1} A_{IB}^{\ i}, \quad f_{\tilde{B}} = f_B - A_{BI} A_{II}^{-1} f_I, \quad (3)$$

здесь S_{BB} – матрица граничных жесткостей или дополнение Шура для подобласти i, вектор $f_{\tilde{B}}$ – вектор правых частей. Матрицы S_{BB}, A_{II} положительно определены и симметричны. Отметим, что порядок матрицы S_{BB} значительно меньше, чем исходной матрицы, но может быть все-таки большим для ее обращения и хранения, поэтому для системы с дополнением Шура необходимо применять итерационные методы решения и распределённые схемы хранения.

2. Формирование матрицы дополнения Шура на нескольких GPU

Одной из самых затратных операций формирования дополнения Шура является обращение матрицы A_{II}. Обычно на практике методами нахождения обратной матрицы являются плохо распараллеливаемые прямые методы, например метод обращения на основе LL^T-разложения. Для последовательного варианта использовалась объектно-ориентированная реализация метода декомпозиции [6] в рамках пакета программ FEStudio [7], в котором операции над матрицами реализованы в формате DCSR (Dynamic Compressed Storage Row – динамический сжатый построчный формат хранения). В этом случае матрица хранится в виде двух списков AV и AVC. Элементами каждого списка являются массивы, которые хранят значения и номера столбцов ненулевых элементов исходной матрицы (AV – значения, AVC – номера столбцов). При этом i-й элемент списка хранит информацию, взятую из i-й строки матрицы. Однако ограничения технологии CUDA не позволяют в полной мере использовать объектно-ориентированный подход, в параллельном варианте данные удобнее хранить небольшим числом массивов, и в этом случае применим формат CSR (Compressed Storage Row) – сжатый построчный формат хранения. При использовании такого формата матрица хранится в виде трех массивов: AV – массив значения ненулевых элементов, AVC – массив номеров столбцов ненулевых элементов и AVN – массив номеров из AV, с которых начинается новая строка исходной матрицы. Отметим, что затраты на перевод матрицы из DCSR в CSR несущественны, что позволяет матричные операции на CPU выполнять в DCSR формате, а CSR формат использовать для вычислений на GPU.

Рассматриваемый в работе алгоритм вычисления обратной матрицы для параллельной реализации заключается в решении матричной системы вида $A_{II} X = E$, где E – единичная матрица $n_I \times n_I$. Система решается на GPU предобусловленным алгоритмом сопря-

женных градиентов (см. раздел 3). Если в правой части системы вместо матрицы E подставить 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 (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 $\tilde{n}_B^i$ – число столбцов матрицы A_{BB}, пересылаемых на i-й GPU, $\tilde{A}_{BB}^i \in R^{n_B \times \tilde{n}_B^i}$ – матрица, состоящая из столбцов матрицы A_{BB} с номерами от $\sum_{j=0}^{i} \tilde{n}_B^j$ до $\sum_{j=0}^{i+1} \tilde{n}_B^j$, где i – номер GPU. Каждый графический ускоритель решает $\tilde{n}_B^i$ систем $A_{II} a^k = A_{IB}^k$, здесь $k \in \left[\sum_{j=0}^{i} \tilde{n}_B^j, \sum_{j=0}^{i+1} \tilde{n}_B^j\right]$, а $A'_{IB} = \left\{a^1, a^2, \ldots, a^{n_B}\right\}$. В реализации подхода независимого решения систем уравнений на нескольких графических ускорителях используется технология OpenMP. Для этого создаются несколько нитей (число которых равно числу доступных устройств), определяется номер нити и каждой назначается графический ускоритель с тем же номером.

Ниже представлен параллельный алгоритм формирования дополнения Шура на каждой подобласти Ω_i (индекс i опущен).

Решаем систему $A_{II} A'_{IB} = A_{IB}$ *(матрицы хранятся на GPU в CSR формате, решение – по столбцам)*;

$S_{BB} = A_{BB} - A_{BI} A'_{IB}$ *(S_{BB} и результат произведения матриц – построчно, A_{BB} – в CSR формате)*;

$\hat{f}_B = f_B - A_{BI} x$ *(x – решение системы $A_{II} x = f_I$)*;

Формируем и решаем: $S_{BB} u_B = \hat{f}_B$ *(S_{BB} формируется в DCSR на CPU, а затем копируется в CSR на GPU)*;

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 $A'^i_{IB} \in R^{n_I \times \tilde{n}_B^i}$, состоящая из столбцов матрицы A'_{IB} с номерами от $\sum_{j=0}^{i} \tilde{n}_B^j$ до $\sum_{j=0}^{i+1} \tilde{n}_B^j$. Это позволяет без дополнительных коммуникаций выполнить оставшиеся операции (произведение и разность матриц, см. соотношения (3)) для каждой матрицы A'^i_{IB} независимо на нескольких GPU, которые используются при вычислении матриц S_{BB}^i. Далее формируется глобальная матрица дополнения Шура S_{BB} (четвертый шаг алгоритма), состоящая из локальных S_{BB}^i, принадлежащих i-й подобласти.

Локальные матрицы дополнения Шура S_{BB}^i хранятся в ленточном виде. Матрица S_{BB} после формирования из локальных S_{BB}^i преобразуется из ленточного к тому формату, в котором будет наиболее удобно с ней работать на GPU, например CSR.

В рассмотренных примерах максимальное ускорение при параллельном формировании по отношению к времени CPU составляет 1.87 и достигается при небольшом числе подобластей $n_\Omega = 16$ с использованием восьми GPU. С ростом числа подобластей размерности матриц уменьшаются, а вычисления на CPU становятся эффективней.

3. Решение интерфейсной системы уравнений на GPU

Отметим, что система для дополнения Шура является разреженной, но её заполненность на один-два порядка больше заполненности исходной конечно-элементной

системы. Для эффективного решения подобных задач на параллельных системах необходимо построение предобуславливателей, обладающих достаточным ресурсом параллелизма. Отметим, что многие предобуславливатели плохо распараллеливаются как на стадии построения, так и на стадии применения в итерационном алгоритме. В работе использовались методы сопряжённых градиентов с диагональным предобуславливателем (DIAG), предобуславливанием по методу симметричной последовательной верхней релаксации (SSOR) и предобуславливателем, построенным после масштабирования системы (DIP). Оптимальным выбором для вычислений на GPU будут предобуславливатели, в которых считается, что известна аппроксимация обратной матрицы системы. В этом случае дополнительные операции, вызванные переходом к предобусловленной системе, сводятся к матрично-векторному произведению. При его выполнении вектор хранится в текстурной памяти, которая кэшируется, что даёт более быстрый доступ и уменьшает временные затраты. Для вычисления каждой компоненты 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 его типа.

Для решения интерфейсной системы (3) реализован блочный алгоритм сопряженных градиентов, использующий несколько GPU. Граф матрицы S_{BB} системы (3) разделяется на блоки по числу GPU с использованием методов из библиотеки [8]. Вершины графа делятся на две группы: внутренние и граничные (связанные хотя бы с одной вершиной другого блока). На основе такого разделения в каждом блоке формируется несколько матриц: S_I – матрица внутренних неизвестных; S_{IB} – матрица связи внутренних и граничных неизвестных; $S_B^{i,j}$ – матрица связи граничных неизвестных с j-м блоком, здесь i – номер блока. Тогда одна из самых затратных операции сопряженных градиентов, матрично-векторное произведение $q = S_{BB} p$, примет вид:

$$\begin{pmatrix} q_1 \\ q_B^1 \\ q_2 \\ q_B^2 \\ \vdots \\ q_N \\ q_B^N \end{pmatrix} = \begin{pmatrix} S_1 & S_{1B} & 0 & 0 & \cdots & 0 & 0 \\ S_{B1} & S_B^{1,1} & 0 & S_B^{1,2} & \cdots & 0 & S_B^{1,N} \\ 0 & 0 & S_2 & S_{2B} & \cdots & 0 & 0 \\ 0 & S_B^{2,1} & S_{B2} & S_B^{2,2} & \cdots & 0 & S_B^{2,N} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & S_N & S_{NB} \\ 0 & S_B^{N,1} & 0 & S_B^{N,2} & \cdots & S_{BN} & S_B^{N,N} \end{pmatrix} \begin{pmatrix} p_1 \\ p_B^1 \\ p_2 \\ p_B^2 \\ \vdots \\ p_N \\ p_B^N \end{pmatrix}. \quad (4)$$

В выражении (4) в каждом потоке вычисляются векторы:

$$q_B^{loc} = \sum_{j=1, j\neq i}^{j\leq N} S_B^{I,j} p_B^j, \quad (5)$$

$$q_B^I = S_{BI} p_I + S_B^{II} p_B^I, \quad q_I = S_I p_I + S_{IB} p_B^I, \quad (6)$$

$$q_B^I = q_B^I + q_B^{loc}, \quad (7)$$

где I – номер потока. Вектор q_B^{loc} (5) вычисляется на CPU, остальные операции (6), (7) – на GPU. Перед выполнением операции с каждого GPU отправляются векторы p_B^j на CPU, которые при минимизации границ имеют размер гораздо меньший по сравнению с размерностью вектора p, где $j = 1,2,\ldots,N$. Обмен данными между GPU осуществляется асинхронно через память CPU. После выполнения (5) вектор q_B^{loc} пересылается на GPU и вычисляется выражение (7). Стоит отметить, что обмен векторами q_I и q_B^I между потоками не требуется. Векторные операции сложения и умножения на скаляр вы-

полняются независимо на каждом потоке отдельно для компонент, принадлежащих внутренним узлам и граничным. Скалярное произведение векторов вычисляется суммированием локальных произведений в глобальной переменной.

Решение интерфейсной СЛАУ предобусловленным методом сопряженных градиентов позволяет достичь ускорения на одном GPU по сравнению с CPU, равное 67, при разделении на 64 блока, а при решении системы с использованием восьми GPU ускорение увеличивается более чем в 4 раза. С увеличением числа блоков разделения до 1024 растет и ускорение, равное 93, однако использование нескольких GPU практически нецелесообразно при этом размере блоков. При этом использование различных предобуславливателей не дает различий по общему времени выполнения. Решение интерфейсной СЛАУ на тестовых задачах с одинаковой вычислительной нагрузкой на каждый GPU показало, что с ростом числа ускорителей алгоритм хорошо масштабируется.

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

Численные эксперименты проводились на гибридных кластерах ИМ УрО РАН и «Уран» ИММ УрО РАН.

Работа выполняется в рамках программы Президиума РАН №18 при поддержке УрО РАН (проект 12-П-1-1034) и гранта РФФИ №11-01-00275-а.

Литература

1. Haynsworth E.V. On the Shur complement // Basel Mathematical Notes. 1968. № 20. 17 p.
2. Przemieniecki J.S. Theory of Matrix Structural Analysis. New York: McGaw-Hill, 1968. 480 p.
3. Постнов В.А. Метод суперэлементов в расчетах инженерных сооружений. Л.: Судостроение, 1979. 288 с.
4. Gaidamour J., Henon P. A parallel direct/iterative solver based on a Shur complement approach // IEEE 11th International Conference on Computational Science and Engineering, San Paulo, Brazil, 2008. – P. 98–105.
5. Копысов С.П., Кузьмин И.М., Новиков А.К., Сагдеева Ю.А. Двухуровневое разделение области для параллельного метода подструктур // Актуальные проблемы математики, механики, информатики: Сб. статей ИМСС УрО РАН. Пермь, 2011. С. 69–76.
6. 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.
7. Копысов С.П., Новиков А.К., Пономарёв А.Б., Рычков В.Н, Сагдеева Ю.А. Программная среда расчётных сеточных моделей для параллельных вычислений // Программные продукты и системы. 2008. № 2. С. 87–89.
8. Karypis G., Kumar V. 500 METIS – Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0. Tech. Report. 1995.