Конечно-элементные технологии для кластеров гибридной архитектуры*

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

Институт механики УрО РАН

Предлагаются новые конечно-элементные технологии для кластера CPU+GPU архитектуры, основанные на распределении вычислительной нагрузки между ядрами центральных и графических процессоров. Обсуждается реализация предлагаемых алгоритмов в технологии CUDA+OpenMP+MPI. Процесс сборки матрицы коэффициентов системы уравнений переносится на этап ее решения, где применяется поэлементная схема. Сборка матрицы заменяется сборкой вектора — результата матрично-векторного произведения. Вычисления между CPU и GPU распределяются с учетом затрат на пересылку данных.

1. Введение

Применение метода конечных элементов (МКЭ), как правило, связано с использованием адаптивных алгоритмов, когда при анализе полученного решения определяется стратегия дальнейших вычислений [1]: перестроение сетки без изменения связности конечных элементов (r-версия); локальное измельчение/огрубление сетки (h-версия); локальное или глобальное повышение порядка аппроксимирующих функций (p-версия) и их комбинированное использование (hp-версия, rp-версия и т.д.). На каждом шаге уточнения аппроксимации необходимо переформировать решаемую систему линейных алгебраических уравнений. В том числе, по этой причине в адаптивном МКЭ применяются поэлементные схемы решения, не требующие сборки глобальной матрицы жесткости.

Эффективное распараллеливание адаптивных версий МКЭ предполагает параллельную реализацию каждого из этапов решения задачи [2]. Общую схему применения методов разделения и отображения по процессорам представим следующим образом: сеточная модель разделяется каким-либо способом по числу вычислительных узлов; формируется и решается система уравнений (СЛАУ) и оценивается погрешность решения; сеточная аппроксимация перестраивается с учетом оценки погрешности; находится дисбаланс нагрузки и, если необходимо, выполняется новое разделение сетки; вычислительная нагрузка перераспределяется в соответствии с новым разделением.

При вычислениях на кластерах гибридной архитектуры, на GPU эффективно реализуется численное интегрирование локальных матриц жесткости [3, 4], особенно в случае высоких порядков аппроксимации [5] и решение полученных систем линейных алгебраических уравнений [6]. В данном исследовании рассматриваются параллельные алгоритмы: численного интегрирования локальных матриц, задания граничных условий первого рода, поэлементных схем решения с различными вариантами сборки векторов.

2. Формирование конечно-элементной системы уравнений на гибридной архитектуре

Формирование конечно-элементной системы в явном виде включает сборку проинтегрированных локальных матриц $K = \sum_{e=1}^{m} C_e^T K_e C_e$ и векторов правой части $f = \sum_{e=1}^{m} C_e^T f_e$ в систему уравнений $Ku = f$. Здесь $K_e = \int_{V_e} B_e^T D_e B_e dV$ — локальная матрица жесткости $K_e \in \mathbb{R}^{N_e \times N_e}$; $C_e \in \mathbb{N}^{N_e \times N}$ — локальная матрица связности, которая связывает локальные неизвестные в конечном элементе e и глобальные неизвестные в системе уравнений; N_e —

*Работа выполнена при поддержке РФФИ (проекты 13-01-00101-а, 14-01-00055-а, 14-01-31066-мол_а)

число неизвестных одного конечного элемента; N — размер системы, m — число конечных элементов. Полученная матрица коэффициентов (глобальная матрица жесткости) K системы уравнений обычно хранится в компактном формате, например, CSR (списки значений ненулевых элементов, их столбцовых индексов и начальные позиции строк в этих списках) [7]. При сборке матрицы K в компактный формат хранения проблематично задействовать большое число нитей графического процессора. В этом случае для формирования матрицы коэффициентов системы проинтегрированные локальные матрицы и векторы правой части передаются на CPU. Если полученная система уравнений будет решаться с применением GPU, то матрицу коэффициентов K и вектор правой части f необходимо переслать на графический ускоритель. В некоторых случаях, только пересылки матриц на CPU и GPU составляют существенную часть от времени решения сформированной системы уравнений.

Применение поэлементных (element-by-element) схем решение [2], не требует сборки матрицы коэффициентов K и указанных выше передач матриц. В этих схемах сборка глобальной матрицы жесткости заменяется сборкой вектора — результата матрично-векторного произведения. Таким образом, проинтегрированные локальные матрицы остаются в памяти графического ускорителя и при решении системы уравнений умножаются на соответствующие компоненты векторов.

При вычислении матриц конечных элементов K_e, $e = \overline{1, m}$ распараллеливаются:
— цикл по конечным элементам e;
— цикл по точкам интегрирования Гаусса i, j, k

$$K_e = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} \sum_{k=1}^{n_k} {B_e}^T D_e B_e w_i w_j w_k \det J;$$

— вычисления элементов или блоков матрицы жесткости K_e:

$$K^e_{\alpha\beta} = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} \sum_{k=1}^{n_k} {B^e_\alpha}^T D_e B^e_\beta w_i w_j w_k \det J, \quad \alpha = \overline{1, N_e}, \beta = \overline{1, N_e}.$$

При высоких порядках аппроксимирующих функций возможно распараллеливание произведения матриц производных B_e и упругих постоянных D_e. В случае иерархических функций формы, полиномы более высоких порядков строятся на основе уже существующих: при формировании матриц B_e и K_e вычисляются только изменяемые и новые элементы матриц.

2.1. Отображение на гибридную архитектуру с несколькими GPU

Отображение вычислений по процессорам наследует разделение расчетной сетки, поэлементный граф которой разделяется одним из алгоритмов разделения графов [2]. Результатом разделения являются подмножества конечных элементов, пересекающиеся по общим узлам сетки. Число подмножеств конечных элементов равно числу узлов вычислительной системы. Каждому такому подмножеству ставится в соответствие процесс MPI. В каждом процессе MPI создается несколько нитей OpenMP равное числу ядер CPU данного вычислительного узла. Мастер-нить OpenMP обеспечивает MPI коммуникации. Часть оставшихся нитей обеспечивает передачу данных между CPU и GPU внутри узла, и запуск kernel-функций CUDA. Число таких нитей OpenMP полагается равным числу GPU в узле, другие нити выполняют только вычисления.

На этапе численного интегрирования вычисления распределяются между центральными и графическими процессорами в зависимости от порядков аппроксимирующих функций. Интегрирование матриц конечных элементов высоких порядков более эффективно выполняется на графических процессорах [3, 5]. Матрицы конечных элементов первого порядка

формируются на ядрах GPU таким образом, что каждый блок нитей CUDA вычисляет несколько локальных матриц [5]. В результате интегрирования на CPU и GPU полученные матрицы K^e размещаются в оперативной памяти и памяти ускорителей. Это необходимо учитывать при матрично-векторном произведении и сборке векторов в поэлементной схеме.

Далее в проинтегрированные локальные матрицы вносятся граничные условия первого рода с сохранением симметрии матриц.

2.2. Задание граничных условий первого рода

Задание граничных условий первого рода (Дирихле) на графических ускорителях позволяет исключить пересылку матриц на этом этапе, но необходимо учитывать особенности массивно-параллельных вычислений, по возможности избегая ветвлений алгоритма. При задании граничных условий первого рода в конечно-элементных вычислениях стараются сохранить симметрию матрицы коэффициентов, для этого [8]: 1) умножают на большое число ($k_{ii} \leftarrow k_{ii} * \beta$, $f_i = \mathring{u}_i * \beta$, полагая $\beta = 10^{10}$); 2) сохраняют в системе тривиальные уравнения, обнуляя в них внедиагональные элементы K; 3) исключают тривиальные уравнения $u_i = \mathring{u}_i$.

Умножение на большое число — прием, наиболее простой в реализации, но изменяет спектр собственных значений матрицы коэффициентов K. Исключение тривиальных уравнений связано с перенумерацией степеней свободы, т.к. требуется выделить и хранить два подмножества степеней свободы: значения которых заданы $\{u_i : u_i = \mathring{u}_i\}$ и подмножество неизвестных степеней свободы, а также их отображения в множество всех степеней свободы.

При сохранении в системе $Ku = f$ тривиальных уравнений $u_i = \mathring{u}_i$, полагают $f_i = \mathring{u}_i$, $f_j \leftarrow f_j - k_{ji} * \mathring{u}_i$, $k_{ii} = 1$, $k_{ij} = 0$, $k_{ji} = 0$, $i, j = \overline{1, N}$. В случае больших значений k_{ii} также существенно меняется обусловленность K, поэтому в данном исследовании применялся вариант с сохранением значения k_{ii} и $f_i = k_{ii} * \mathring{u}_i$. В поэлементных схемах сборка глобальной матрицы жесткости не выполняется, поэтому задание граничных условий Дирихле переносится на матрицы K_e и элементные векторы f_e в виде: $f_i^e = k_{ii}^e * \mathring{u}_i^e$, $f_j^e \leftarrow f_j^e - k_{ji}^e * \mathring{u}_i^e$, $k_{ij}^e = 0$, $k_{ji}^e = 0$, $i, j = \overline{1, N_e}$ и $e = \overline{1, m}$. Суммирование слагаемых отдельных конечных элементов происходит при сборке вектора правой части f. На GPU эти операции выполняются двумя kernel-функциями: первая — формирует вектор $\tilde{f} = \cup_{e=1}^{m} f^e$, вторая — отображает $\tilde{f}$ из $\mathbb{R}^{m \cdot N_e}$ в вектор f из $\mathbb{R}^N$.

Рассматривается также вариант задания условий Дирихле в виде $\hat{C}^T K \hat{C} u = \hat{C}^T (f - K\mathring{u})$ [9], где элементы матрицы $\hat{C}$ принимают значения: $\hat{c}_{ij} = 1$, если $u_i = \mathring{u}_i$ и $\hat{c}_{ij} = 0$ — в другом случае. Правая часть преобразуется до решения системы, а выражение $\hat{C}^T K \hat{C} u$ реализуется в ходе решения системы уравнений в виде последовательности матрично-векторных произведений: $\hat{C}u$, $K(\hat{C}u)$, $\hat{C}^T(K\hat{C}u)$, которые эффективно выполняются на GPU.

3. Поэлементное решение СЛАУ на гибридной архитектуре

Поэлементные схемы используют информацию о способе построения матрицы коэффициентов системы уравнений в виде суммы матриц конечных элементов: $K = \sum_{e=1}^{m} C_e^T K_e C_e$. Так в итерационных методах подпространств А.Н. Крылова произведение Kp записывается, как $\sum_{e=1}^{m} C_e^T K_e C_e p$, а вектор p вносится под знак суммы. В этом случае выражение $C_e p$ представляет компоненты вектора, соответствующие конечному элементу e.

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

Наиболее известны следующие варианты поэлементной схемы: с извлечением данных в поэлементный вектор, размер которого — число степеней свободы одного элемента N_e и схе-

ма с разнесением данных в вектор размера $N_e \cdot m$, являющийся объединением поэлементных векторов [2].

В схеме с извлечением необходимые компоненты вектора $p \in \mathbb{R}^N$ извлекаются в вектор $p_e \in \mathbb{R}^{N_e}$ (в рассматриваемых случаях $N_e \ll N$), который умножается на локальную матрицу жесткости K_e

$$p_e = C_e p, \;\; q_e = K_e p_e, \;\; q \leftarrow q + C_e^T q_e, \;\; e = \overline{1, m}. \tag{1}$$

Как правило, операции $C_e p$ и $C_e^T q_e$ реализуются не в виде произведений, а как переход от одного индексного пространства к другому. При выполнении данной схемы большим числом нитей CUDA происходят конфликты, как при чтении данных из вектора p в p_e (снижающие производительность), так и при записи в вектор q слагаемых одновременно из нескольких векторов q_e. Возникает так называемое «состояние гонки» (race condition), когда параллельные процессы пытаются записать результат суммирования компонент из нескольких конечных элементов в одну область памяти. Результат такой операции не определен. В случае нескольких GPU необходимые компоненты векторов p и q должны быть доступны соответствующей нити CUDA.

Require: $\tilde{r}_0^{(i)} = \tilde{f}^{(i)}; \quad M^{(i)} = ({C^{(i)}}^T diag(\tilde{K}^{(i)}) C^{(i)})^{-1}; \quad \bar{u}_0^{(i)} = 0;$

$s_0^{(i)} = M^{(i)}(\sum_{\leftrightarrows}^{\partial\Omega^{(i)}} {C^{(i)}}^T \tilde{r}_0^{(i)})$;{ $\sum_{\leftrightarrows}^{\partial\Omega^{(i)}}$ — *обмен между смежными областями сетки* }

$\bar{s}_0^{(i)} = C^{(i)} s_0^{(i)}$;

$\bar{p}_0^{(i)} = \bar{s}_0^{(i)}$;

$\rho_0 = \sum_{i=1}^{n_p} (\bar{s}_0^{(i)}, \tilde{r}_0^{(i)})$;{ $\sum_{i=1}^{n_p}$ — *сложение по всем процессам MPI* }

1: **for** $j = 0, 1, \cdots$ **do**

2: $\tilde{q}_j^{(i)} := \tilde{K}^{(i)} \bar{p}_j^{(i)}$; { *матрично-векторное произведение* }

3: $\gamma_j := \sum_{i=1}^{n_p} (\bar{p}_j^{(i)}, \tilde{q}_j^{(i)})$;

4: $\alpha_j := -\rho_j / \gamma_j$;

5: $\bar{u}_{j+1}^{(i)} := \bar{u}_j^{(i)} - \alpha_k \bar{p}_j^{(i)}$;

6: $\tilde{r}_{j+1}^{(i)} := \tilde{r}_j^{(i)} + \alpha_j \tilde{q}_j^{(i)}$;

7: $\bar{r}_{j+1}^{(i)} = C^{(i)} \sum_{\leftrightarrows}^{\partial\Omega^{(i)}} {C^{(i)}}^T \tilde{r}_{j+1}^{(i)}$

8: **if** $(||r_{j+1}||_2 / ||r_0||_2 < \varepsilon)$ **then**

9: **return** { *норма вычисляется в виде* $|| \cdot ||_2 = \sqrt{\sum_{i=1}^{n_p} (\bar{\cdot}, \tilde{\cdot})^{(i)}}$ }

10: **end if**

11: $s_{j+1}^{(i)} := M^{(i)}(\sum_{\leftrightarrows}^{\partial\Omega^{(i)}} {C^{(i)}}^T \tilde{r}_{j+1}^{(i)})$;

12: $\bar{s}_{j+1}^{(i)} := C^{(i)} s_{j+1}^{(i)}$;

13: $\rho_{j+1} := \sum_{i=1}^{n_p} (\bar{s}_{j+1}^{(i)}, \tilde{r}_{j+1}^{(i)})$;

14: $\beta_j := \rho_{j+1} / \rho_j$;

15: $\bar{p}_{j+1}^{(i)} := \bar{s}_{j+1}^{(i)} + \beta_j \bar{p}_j^{(i)}$;

16: **end for**

Рис. 1. Параллельный предобусловленный метод сопряженных градиентов

В схеме с разнесением матрица K представляется блочно-диагональной матрицей $\tilde{K}$, блоками которой являются локальные матрицы K_e, следующим образом: $K = C^T \tilde{K} C$. В отличии от (1), матрично-векторное произведение осуществляется над векторами размерности $\tilde{N} = m \cdot N_e$, которые являются объединением элементных векторов $p_e \in \mathbb{R}^{N_e}$ и $q_e \in \mathbb{R}^{N_e}$. Вводятся операции Cq — разнесения (расширение в пространстве индексов) вектора $q \in \mathbb{R}^N$ и $C^T \tilde{q}$ — суммирования компонент вектора $\tilde{q}$, здесь $C \in \mathbb{N}^{\tilde{N} \times N}$ — матрица связности, со-

ставленная из локальных матриц связности C_e. Произведение Kp записывается в виде:

$$\bar{p} = Cp, \;\; \tilde{q} = \tilde{K}\bar{p}, \;\; q = C^T\tilde{q}. \tag{2}$$

здесь $\bar{p}$ — вектор, составленный из m векторов p_e.

При параллельной реализации (см. рис. 1), блоки $\tilde{K}^{(i)}$, соответствующие подобласти Ω_i расчетной сетки формируются (см. п. 2) ядрами CUDA в процессе MPI с номером i и хранятся в памяти графических ускорителей. Операции с векторами осуществляются функциями cuBLAS, результаты скалярных произведений, полученные на графических ускорителях суммируются (см. рис. 1, операции $\sum_{i=1}^{n_p}$) в рамках вычислительного узла — при помощи клаузы OpenMP reduction и функции MPI_Allreduce — в рамках кластера. Сложение компонент вектора невязки ($\sum_{\leftrightarrows}^{\partial\Omega^{(i)}} {C^{(i)}}^T \tilde{r}_{j+1}^{(i)}$) происходит только по смежным процессам (MPI) и потокам (OpenMP, через массивы в общей памяти), для компонент, соответствующих узлам расчетной сетки, принадлежащим $\partial\Omega^{(i)}$.

При выполнении операции $q = C^T\tilde{q}$ в (2) возникает «состояние гонки», особенно заметное на GPU. Для того, чтобы уменьшить число конфликтов рассматривалось несколько вариантов реализации: умножение на матрицу C^T, хранящуюся в формате CSR; поузловой вариант данной операции. При поузловой сборке каждая нить CUDA суммирует компоненты степени свободы из всех содержащих данный узел конечных элементов. Для этого каждому узлу расчетной сетки ставятся в соответствие номера конечных элементов, содержащих данный узел и локальные номера этого узла в этих конечных элементах.

4. Заключение.

При решении статических задач деформирования сравнивались несколько вариантов отображения параллельных вычислений на гибридные узлы кластера X4 Института механики УрО РАН:

– численное интегрирование матриц K_e на CPU и GPU, сборка K в формате CSR на центральном процессоре, передача ее на графический ускоритель, где решалась сформированная система уравнений;

– вычисление матриц K_e, внесение граничных условий первого рода на CPU, передача матриц на графический ускоритель и решение с использованием схемы (2);

– интегрирование локальных матриц, внесение граничных условий первого рода на GPU и применение поэлементной схема с разнесением;

– предыдущий вариант с поузловым способом сборки вектора результата в матрично-векторном произведении.

Кратко остановимся на предварительных результатах вычислительных экспериментов. Без учета формирования матрицы коэффициентов и передачи данных между CPU и GPU, поэлементная схема (2) на GPU выполняется примерно в три раза медленнее, чем вариант, в котором матрица K хранится в формате CSR. Вместе с тем, ускорение относительно последовательного варианта с собранной матрицей составило около двадцати раз. Схема с поузловой сборкой вектора результата незначительно уступает варианту с произведением на матрицу связности. Сравнение по общему времени вычислений показало, что варианты с поэлементной схемой выполняются в десятки раз быстрее, чем вариант со сборкой на центральном процессоре матрицы K, хранящейся в компактном формате (CSR).

Литература

1. Error-controlled Adaptive Finite Elements in Solid Mechanics / Ed. E. Stein. John Wiley & Sons. 2002. 422 p.

2. Копысов С.П., Новиков А.К. Метод декомпозиции для параллельного адаптивного конечно-элементного алгоритма // Вестник Удмуртского университета. Математика.

Механика. Компьютерные науки. 2010. №3. С. 141–154.

3. Banas K., Plaszewski P., Maciol P. Numerical integration on GPUs for higher order finite elements URL: http://arxiv.org/abs/1310.1191 (дата обращения: 05.12.2013).

4. Knepley M.G., Terrel A.R. Finite Element Integration on GPUs // ACM Transactions on Mathematical Software. 2013. V. 39. No. 2. P. 10:1–10:13.

5. Новиков А.К., Копысов С.П., Кузьмин И.М., Недожогин Н.С. Формирование матриц конечно-элементных систем в GPGPU // Вестник Нижегородского университета им. Н.И. Лобачевского. 2014. № 3(1). С. 120–125.

6. Кузьмин И.М., Недожогин Н.С., Новиков А.К., Рычков В.Н., Сагдеева Ю.А., Тонков Л.Е. Параллельная реализация конечно-элементных алгоритмов на графических ускорителях в программном комплексе FEStudio // Компьютерные исследования и моделирование. 2014. Т. 6. № 1. С. 79–97.

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

8. Норри Д., де Фриз Ж. Введение в метод конечных элементов. М. Мир. 1981. 304 с.

9. Даутов Р.З. Программирование МКЭ в MATLAB. Казань. 2010. 71 с.