УДК 519.67

ФОРМИРОВАНИЕ И РЕШЕНИЕ КОНЕЧНО-ЭЛЕМЕНТНЫХ СИСТЕМ НА ГИБРИДНОЙ АРХИТЕКТУРЕ[1)]

А.К. НОВИКОВ, С.П. КОПЫСОВ, Н.С. НЕДОЖОГИН

Институт механики УрО РАН, г. Ижевск

E-mail alexander.k.novikov@gmail.com; s.kopysov@gmail.com; nedozhogin@inbox.ru

FINITE ELEMENT SYSTEMS FORMING AND SOLVING ON HYBRID ARCHITECTURE

A.K. NOVIKOV, S.P. KOPYSOV, N.S. NEDOZHOGIN

Institute of Mechanics UB RAS, Izevsk

Аннотация

Предлагаются параллельные алгоритмы численного интегрирования локальных матриц жесткости в МКЭ и решения систем уравнений без явного формирования глобальной матрицы жесткости с распределением вычислений на CPU и GPU. Результаты применения предложенных параллельных алгоритмов на гибридной вычислительной системе сравниваются с известными.

Ключевые слова: Системы алгебраических уравнений МКЭ, гибридные вычислительные архитектуры, параллельные алгоритмы численного интегрирования.

Summary

Parallel algorithms for numerical integration of the local stiffness matrices in the FEM and solving systems of equations without the explicit forming of the global stiffness matrix with computations distributing on the CPU and GPU are offered. The results of the applying proposed parallel algorithms on the hybrid computing systems are compared with known results.

Key words: Systems of FEM algebraic equations, hybrid computing architectures, parallel algorithms of numerical integration.

Введение

При создании параллельных численных алгоритмов, выполняемых на гибридных архитектурах, необходимо не только обеспечить сбалансированное распределение вычислений между графическими процессорами (GPU) и ядрами центрального процессора (CPU), но и минимизировать передачу данных между ними. Рассмотрим особенности параллельных конечно-элементных алгоритмов применительно к гибридной (CPU+GPU) архитектуре.

В методе конечных элементов наиболее эффективно массивный параллелизм [1] вычислений на GPU реализуется при численном интегрировании локальных матриц жесткости [2, 3] и решении полученных систем линейных алгебраических уравнений [4]. Как правило, эти два этапа рассматриваются независимо, а сборка системы уравнений и соответствующие затраты опускаются.

При численном интегрировании на GPU локальных матриц жесткости большее ускорение вычислений достигается для конечных элементов высоких порядков. Так, в [2] на GPU вычисляются локальные матрицы конечных элементов до восьмого порядка. Выделяется два уровня распараллеливания: конечных элементов и блоков степеней свободы в конечном элементе. Переносу на графические процессоры численного интегрирования в известном конечно-элементном проекте FEniCS [5] посвящена работа [3],

[1)] Работа выполнена при поддержке РФФИ (проекты 13-01-00101-а, 14-01-31066-мол_а)

где интегрируются матрицы би- и трилинейных конечных элементов. В указанных работах численное интегрирование рассматривается без учета связи с решением конечно-элементной системы и сравнения с распараллеливанием вычислений на центральном процессоре и гибридным (CPU+GPU).

Формирование конечно-элементной системы в явном виде включает также 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 $K_e \in \mathbb{R}^{N_e \times N_e}$ – локальная матрица жесткости; $C_e \in \mathbb{N}^{N_e \times N_e}$ – локальная матрица связности, которая связывает локальные степени свободы в конечном элементе e и глобальные степени свободы в системе уравнений; N_e – число степеней свободы одного конечного элемента; N – размер системы, m – число конечных элементов. Полученная матрица коэффициентов (глобальная матрица жесткости) K системы уравнений хранится в компактном формате, например, CSR (значения ненулевых элементов, их столбцовые индексы и начальные позиции строк в списках) [6]. При записи собранной матрицы в компактный формат хранения проблематично задействовать большое число параллельных процессов графического процессора. Поэтому 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 и вектор правой части необходимо переслать на графический ускоритель. В некоторых случаях, только пересылки матриц на CPU и GPU составляют существенную часть от времени решения сформированной системы уравнений.

Далее будет рассматриваться подход на основе поэлементных (element-by-element) схем [7] метода сопряженных градиентов, которые не требуют сборки матрицы коэффициентов K и передачи локальных матриц K_e из памяти графического ускорителя в оперативную память и передачи глобальной матрицы жесткости в память графического ускорителя. 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.

1. Численное интегрирование локальных матриц на гибридной архитектуре.

Объектами распараллеливания при численном интегрирования множества из m локальных матриц K_e являются циклы по: конечным элементам ($e = 1, m$) и точкам интегрирования. На этом этапе вычисления распределяются между центральными и графическими процессорами в зависимости от порядков аппроксимирующих функций. Так, в [9] показано, что интегрирование матриц конечных элементов высоких порядков более эффективно выполняется на графических процессорах. При численном интегрировании локальных матриц шестигранных конечных элементов пятого порядка алгоритм с распараллеливанием цикла по точкам интегрирования позволил получить ускорение вычислений примерно в 2.5 раза большее, чем приведено в [2]. На двух GPU – в пять раз, а при распределении вычислений между ядрами CPU и GPU – примерно семикратное ускорение.

В свою очередь, локальные матрицы би- и трилинейных конечных элементов предпочтительнее интегрировать на GPU, когда каждый блок нитей CUDA вычисляет несколько локальных матриц [6], а гибридная архитектура строится на одно- или двухъядерных центральных процессорах. На большинстве современных вычислительных систем, существенно большее ускорение обеспечивается при интегрировании матриц на ядрах центрального процессора с применением OpenMP.

Далее в проинтегрированные локальные матрицы вносятся граничные условия первого рода с сохранением симметрии матриц. Если локальные матрицы вычислялись CPU и GPU, то вектор – результат поэлементного матрично-векторного произведения получится тоже распределенным. Это необходимо учитывать при сборке вектора в поэлементной схеме.

2. Распараллеливание поэлементного матрично-векторного произведения.

Преимущества поэлементной схемы заключаются в следующем: локальные матрицы могут быть вычислены заново без изменения всей матрицы; простота представления и отображения на многопроцес-

сорную вычислительную систему. Основным отличием элементной схемы является способ вычисления матрично-векторного произведения, которое переносится на уровень локальных матриц, объединение локальных матриц в матрицу системы заменяется объединением вектора – результата произведения. Наиболее известны два варианта схемы: с извлечением данных в поэлементный вектор, размер которого – число степеней свободы одного элемента и схема с разнесением данных в вектор, являющийся объединением поэлементных векторов [8].

В схеме с извлечением необходимые компоненты вектора (направления или решения) $u \in \mathbb{R}^N$ извлекаются в вектор $u_e \in \mathbb{R}^{N_e}$ (в рассматриваемых случаях $N_e \ll N$), который умножается на локальную матрицу жесткости.

$$u_e = C_e u, \;\; v_e = K_e u_e, \;\; v \leftarrow v + C_e^T v_e, \;\; e = 1, m. \tag{1}$$

Как правило, операции $C_e u$ и $C_e^T v_e$ реализуются не в виде произведений, а как переход от одного индексного пространства к другому. При распараллеливании данной схемы происходят конфликты, как при чтении данных из вектора u в u_e большим числом нитей CUDA, так и при записи в вектор v слагаемых одновременно из нескольких векторов v_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. Кроме того, в случае нескольких GPU необходимые компоненты векторов u и v должны быть доступны соответствующей нити CUDA.

В схеме с разнесением глобальная матрица жесткости K выражается через блочно-диагональной матрицу $\tilde{K} = [K_e]_{m \times m}$, блоками которой являются локальные матрицы K_e в виде $K = C^T \tilde{K} C$. В отличии от (1), матрично-векторное произведение осуществляется над векторами размерности $\tilde{N} = m \cdot N_e$, которые являются объединением элементных векторов $u_e \in \mathbb{R}^{N_e}$ и $v_e \in \mathbb{R}^{N_e}$. Вводятся операции Cv – разнесение (расширение в пространстве индексов) вектора $v \in \mathbb{R}^N$ и $C^T \tilde{v}$ – суммирования компонент вектора $\tilde{v}$, здесь $C \in \mathbb{N}^{\tilde{N} \times N}$ – матрица связности, составленная из локальных матриц связности C_e. Произведение $Ku = C^T \tilde{K} C u$ записывается в виде:

$$\bar{u} = Cu, \;\; \tilde{v} = \tilde{K}\bar{u}, \;\; v = C^T \tilde{v}. \tag{2}$$

здесь $\bar{u}$ – вектор, составленный из m векторов u_e.

При выполнении операции $v = C^T \tilde{v}$ в (2) возникает «состояние гонки», особенно заметное на GPU. Чтобы исключить его рассматривалось несколько вариантов реализации, включая умножение на матрицу C^T, хранящуюся в формате CSR [6]. Предлагается новый вариант данной операции. Каждый параллельный процесс суммирует компоненты назначенной ему степени свободы из всех связанных с ней конечных элементов. Эта операция обеспечивается структурой данных, в которой номеру узла в расчетной сетке ставятся в соответствие номера конечных элементов, содержащих данный узел и номера узла в каждом из этих конечных элементов. Кроме того, операцией вида $v = C^T \tilde{v}$ выполняется сборка диагонального предобуславливателя $diag(K) = C^T diag(\tilde{K})$.

3. Численные исследования.

При решении задач статики и динамики деформирования сравнивались несколько вариантов отображения параллельных вычислений на гибридную архитектуру:

– 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 предлагаемым новым способом сборки вектора результата в матрично-векторном произведении.

Исследования проводились на гибридном узле кластера X4 Института механики УрО РАН, который оснащен двумя центральными процессорами Intel Xeon E5-2609, двумя видеокартами GeForce GTX 680 (4 ГБ графической памяти на каждой карте) и содержит 64 ГБ оперативной памяти.

Кратко остановимся на результатах вычислительных экспериментов. Сравнение поэлементной схемы (2) на GPU с первым вариантом без учета сборки и пересылок матриц показало, что схема с разнесением в два раза медленнее, чем вариант, в котором матрица K хранится в формате CSR. Тем не менее, ускорение относительно последовательного варианта с собранной матрицей составило более двадцати раз. Следует также отметить, что пересылки матриц K_e на CPU и матрицы K на графический ускоритель в случае динамической задачи составили третью часть от времени решения сформированной системы уравнений на GPU.

Суммарные вычислительные затраты, включающие пересылки матриц, сборку матрицы K с записью в формат CSR в первом варианте больше, чем в вариантах с поэлементной схемой. Вариант поэлементной схемы с разнесением и предложенным способом сборки вектора результата в матрично-векторном произведении показал высокую эффективность и экономичность.

ЛИТЕРАТУРА

1. **Воеводин В.В.** Массивный параллелизм и декомпозиция алгоритмов // Журнал вычислительной математики и математической физики — 1995. — Т. 35, № 6. — С. 988–996.
2. **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).
3. **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.
4. **Kopysov S.P., Kuzmin I.M., Nedozhogin N.S., Novikov A.K., Sagdeeva Y.A.** Hybrid Multi-GPU solver based on Schur complement method // Lecture Notes in Computer Science. — 2013. — V. 7979. — P. 65–79.
5. **Logg A., Mardal K.A., Wells G.N.** Automated Solution of Differential Equations by the Finite Element Method URL: http://fenicsproject.org/pub/book/book/fenics-book-2011-06-14.pdf (дата обращения: 20.08.2014).
6. **Копысов С.П., 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. **Фрид И.** Еще о градиентных итерационных методах в конечно-элементных исследованиях // Ракетная техника и космонавтика. — 1969. — № 3. — С. 63–69.
8. **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.
9. **Новиков А.К., Кузьмин И.М., Недожогин Н.С., Копысов С.П.** Формирование конечно-элементных систем в GPGPU // Параллельные вычислительные технооги (ПаВТ 2014): труды международной научной конференции (1–3 апреля 2014 г., г. Ростов-на-Дону). — Челябинск: Издательский центр ЮУрГУ, 2014. — С. 288–293.

REFERENCES

1. **Voevodin V.V.** Massive parallelism and algorithms decomposition [Massivnyi parallelizm i dekompoziciya algoritmov] // Zhurnal vychislitel'noy matematiki i matematicheskoy fiziki. — 1995. — V. 35, № 6. — P. 988–996. (in Russian)

2. **Banas K., Plaszewski P., Maciol P.** Numerical integration on GPUs for higher order finite elements. http://arxiv.org/abs/1310.1191
3. **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.
4. **Kopysov S.P., Kuzmin I.M., Nedozhogin N.S., Novikov A.K., Sagdeeva Y.A.** Hybrid Multi-GPU solver based on Schur complement method // Lecture Notes in Computer Science. – 2013. – V. 7979. – P. 65–79.
5. **Logg A., Mardal K.A., Wells G.N.** Automated Solution of Differential Equations by the Finite Element Method. http://fenicsproject.org/pub/book/book/fenics-book-2011-06-14.pdf
6. **Kopysov S.P., Kuzmin I.M., Nedozhogin N.S., Novikov A.K., Rychkov V.N., Sagdeeva Y.A., Tonkov L.E.** Parallel implementation of a finite-element algorithms on a graphics accelerator in the software package FEStudio [Parallel'naya realizaciya konechno-elementnyh algoritmov na graficheskih uskoritelyah v programmnom komplekse FEStudio] // Komp'uternye issledovaniya i modelirovanie. – 2014. – V. 6, № 1. – P. 79–97. (in Russian)
7. **Fried I.** More on gradient iterative methods in finite-element analysis // AIAA Journal. – 1969. – V. 7. – № 3. – P. 565–567.
8. **Kopysov S.P., Novikov A.K.** Domain decomposition for parallel adaptive finite element algorithm [Metod dekompozicii dlya parallel'nogo konechno-elementnogo algoritma] // Vestnik Udmurtskogo universiteta. Matematika. Mehanika. Komp'uternye nauki. – 2010. – № 3. – P. 141–154. (in Russian)
9. **Novikov A.K., Kuzmin I.M., Nedozhogin N.S., Kopysov S.P.** Forming of finite element system in GPGPU [Formirovanie konecho-elementnyh sistem v GPGPU] // Parallel'nye vychislitel'nye tehnologii (PaVT 2014): Trudy mezhdunarodnoy nauchnoy konferencii (1–3 aprelya 2014. Rostov-on-Don). – Chelyabinsk: Izdatel'skiy centr YuUrGU. – 2014. – P. 288–293. (in Russian)