Формирование конечно-элементных систем в GPGPU*

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

Институт механики УрО РАН

Рассматриваются параллельные алгоритмы формирования конечно-элементных систем, выполняемые на графических процессорах. Обсуждаются особенности реализации предлагаемых алгоритмов в технологии CUDA. Приводятся результаты вычислительных экспериментов при моделировании задач механики сплошной среды на гибридных вычислительных узлах.

Введение

Вычисления общего назначения на графических процессорах (GPGPU) позволяют уменьшить время решения в десятки раз [1], но требуют разработки новых параллельных алгоритмов и программ, учитывающих особенности гибридной аппаратной архитектуры и программного обеспечения GPU. В первую очередь, как наиболее трудоемкие, рассматриваются алгоритмы решения систем линейных алгебраических уравнений [2, 3].

При решении многомерных задач также актуально распараллеливание процесса формирования конечно-элементных систем, который связан с численным интегрированием по объему и поверхности конечных элементов (локальные матрицы жесткости, векторы распределенной нагрузки). Интегрирование глобальных конечно-элементных матриц состоит из двух шагов: интегрирования локальных матриц конечных элементов и их объединения (суммирования) в матрицу коэффициентов системы сеточных уравнений.

Для численного интегрирования локальных матриц и векторов преимущественно применяются квадратурные формулы Гаусса-Лежандра [4], высокая точность которых достигается при небольшом числе точек интегрирования в заданном направлении. В трехмерном случае, число точек интегрирования по направлению возводится в куб и в каждой точке вычисляются локальные матрицы жесткости, содержащие сотни тысяч элементов. Интегрирование осуществляется независимо над каждой локальной матрицей, что делает этот шаг перспективным для распараллеливания, в том числе для GPGPU [5].

Операция объединения локальных матриц не обладает значительной степенью параллелизма, но важна для оптимизации вычислительных затрат.

1. Формирование конечно-элементных систем на GPU

В методе конечных элементов процесс численного интегрирования матриц масс, демпфирования и жесткости состоит из интегрирования матрицы каждого конечного элемента и сложения (сборки) полученных матриц. Покажем поэлементную сборку матриц, на примере матрицы жесткости

$$K = \int_V B^T D B dV \approx \sum_{e=1}^{m} \int_{V^e} B^{eT} D^e B^e dV^e = \sum_{e=1}^{m} C_e^T K^e C_e, \qquad (1)$$

здесь $K^e \in \mathbb{R}^{N_e \times N_e}$ — локальная (элементная) матрица жесткости; $C = [C_e]_{1\times m} \in \mathbb{Z}^{N_e \times N}$ — матрица связности; m — число конечных элементов; N_e — число степеней свободы в одном конечном элементе; N — число неизвестных в системе уравнений.

*Работа выполнена при финансовой поддержке РФФИ (грант 14-01-00055-а, 14-01-31066-мол_а, 13-01-00101-а) и программы Президиума РАН №18 при поддержке УрО РАН (проект 12-П-1-1005).

Для уменьшения конфликтов при сборке матрицы $K = [k_{ij}]_{N\times N}$ в (1) на GPU рассматриваются следующие варианты упорядочения конечных элементов, узлов сетки и степеней свободы (неизвестных). Первый вариант — создание подмножеств конечных элементов топологических не связанных в данной расчетной сетке. Второй вариант — упорядочение (смещение нумерации) узлов внутри элемента таким образом, что при одновременном обращении к конечным элементам, имеющим общий узел, ребро или грань, только один из конечных элементов осуществляет доступ к соответствующей строке или элементу k_{ij}. Третий вариант — смещение нумерации неизвестных в узле сетки, например, в трех конечных элементах параллельно выполняется доступ к элементам k_{ij},k_{i+1j},k_{i+2j}.

Граничные условия первого рода задаются на GPU следующим образом. В матрицах K^e элементы строк и столбцов, соответствующие степеням свободы с заданными граничными условиями первого рода, полагаются равными нулю, кроме диагональных элементов матрицы. Для этого на GPU передаются: номера этих степеней свободы в конечных элементах, значения граничных условий. На GPU формируется вектор поправок к правой части, получаемый при исключении заданных степеней свободы из других уравнений.

Одним из наиболее важных приложений численного интегрирования локальных конечно-элементных матриц является использование конечных элементов высоких порядков. Часто применяются конечные элементы высокого порядка, построенные на основе иерархических базисных функций [4]. В этом случае, функции более высокого порядка получаются добавлением слагаемых соответствующей степени.

1.1. Иерархические конечно-элементные аппроксимации

Иерархические конечные элементы высокого порядка, используемые в работе, строятся на основе полиномов Лежандра. Аппроксимацию искомой функции u (перемещений) запишем следующим образом $u \approx \sum\limits_{i=0}^{k} \psi_i u_i + \sum\limits_{j=2}^{p} \psi_j a_j$, где ψ_i — линейные базисные функции; ψ_j — иерархические базисные функции; u_i — узловые перемещения; a_j — обобщенные параметры; k — число узлов в элементе; p — порядок аппроксимации.

Запишем трехмерные базисные функции для конечного элемента кубической формы с локальными координатами $-1 \leqslant r, s, t \leqslant 1$. *Узловые базисные функции* конечного элемента имеют вид $\psi_v = \frac{1}{8}(1 + r_v r)(1 + s_v s)(1 + t_v t)$, $v = 1, 8$, где v — номер узла в элементе.

С увеличением порядка аппроксимации p на элементе, добавляются *базисные функции, связанные с обобщенными параметрами на ребрах элемента* $\psi_e^p = \frac{1}{4}(1 + r_e r)(1 + s_e s)P_p(t)$, $p \geqslant 2$, где $P_p(t)$ — проинтегрированный полином Лежандра степени p; e — ребро в элементе. Это соотношение записано для ребра e, параллельного оси t. Для ребер, параллельных осям r или s, функции ψ_e^p получаются перестановкой соответствующих координат.

Базисные функции для граней элемента имеют вид $\psi_f^{mn} = \frac{1}{2}(1 + r_f r)P_m(s)P_n(t)$, $p \geqslant 4$, $m, n \geqslant 2$, $m + n \leqslant p$, здесь f — грань элемента, перпендикулярная оси r. Базисные функции для остальных граней получаются также перестановкой координат.

При $p \geqslant 6$ *в базисных функциях используются внутренние обобщенные параметры.* В данной работе использовались аппроксимации до шестого порядка.

Выбором порядка, во многом, определяются вычислительные затраты. Число точек интегрирования для квадратур Гаусса-Лежандра определяется как

$$n_G \geqslant \big((3p - 2)/2\big)^d, \tag{2}$$

где p — максимальная степень полинома в конечном элементе, d — размерность пространства. С точки зрения вычислительных затрат, интегрирование по трёхмерной области по схеме Гаусса-Лежандра требует $O(n_G)$ вычислений значений функций и для элементов высокого порядка становится существенным.

1.2. Декомпозиция вычислений для гибридных архитектур

В случае вычислений на гибридных вычислительных систем, декомпозиция алгоритмов начинается с распределения вычислений между GPU и центральным процессором (CPU). При интегрировании матриц масс, демпфирования и жесткости рассматриваются следующие варианты распределения вычислений для гибридной вычислительной системы:

1. На GPU численно интегрируется множества матриц, например K^e, $e = 1, 2, \ldots, m$. Полученные матрицы передаются в оперативную память для сборки матрицы K на CPU.

2. Вычисление матриц K^e, задание граничных условий Дирихле на GPU и сборка системы уравнений выполняются на графическом ускорителе. В этом случае передача матриц исключается, а центральный процессор выполняет функцию управляющего процессора, обеспечивает ввод/вывод и пост-процессную обработку результатов.

3. Интеграция вычисления локальных матриц жесткости и задания граничных условий первого рода в вычисление матрично-векторного произведения для методов подпространств Крылова, выполняемых на GPU. Локальная матрица K^e, после учета граничных условий первого рода, умножается на вектор $p^e \in R^{N_e}$ — соответствующую часть вектора направления p, например, в методе сопряженных градиентов.

Остановимся на распараллеливании процесса формирования локальной матрицы жесткости K^e. Последовательный алгоритм формирования K^e включает следующие шаги:
– построение функций формы элемента ψ_e;
– переход от глобальных системы координат к локальной, вычисление матрицы Якоби J, обратного преобразования J^{-1} и якобиана $detJ$;
– вычисление $B^e = [B_1^e B_2^e \ldots B_{N_e}^e]$;
– численное интегрирование по объёму для вычисления матрицы жесткости

$$K^e = [K^e_{\alpha\beta}]_{N_e \times N_e}, \quad K^e_{\alpha\beta} \approx \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} \sum_{k=1}^{n_k} B_\alpha^{eT} D^e B_\beta^e w_i w_j w_k \det J, \tag{3}$$

где B^e_α — матрица производных ψ_e в точке (x_{1i}, x_{2j}, x_{3k}); D^e— матрица характеристик материала (среды); $n_l = \sqrt[3]{n_G}$, $l = i, j, k$, а n_G берется из условия (2); w_i, w_j, w_k — весовые коэффициенты квадратур Гаусса.

Выделим уровни распараллеливания приведенного выше последовательного алгоритма: **I** уровень — отдельных конечных элементов одного порядка; **II** уровень — точек интегрирования Гаусса в случае конечных элементов высоких и/или разных порядков. Параллельные алгоритмы, соответствующие уровням распараллеливания обозначим **Алгоритмы I** и **II**.

Кроме того, возможны параллельные вычисления блоков матрицы жесткости K^e и произведения матриц производных B^e и упругих постоянных D^e. Особенность построения иерархических аппроксимаций на основе существующих полиномов меньших степеней, также можно использовать, вычисляя в параллельных процессах только соответствующие слагаемые при формировании матриц B^e и K^e.

2. Особенности программной реализации.

Рассматриваемые параллельные алгоритмы для численного интегрирования локальных матриц жесткости реализованы в виде библиотеки CUNInt (CUda Numerical Integration), которая интегрирована в пакет конечно-элементного анализа FEStudio [6]. **Алгоритмы I,II** программно реализованы в виде `kernel`-функций CUDA.

Учитывая ограничения на число нитей в блоке N_{tib} и число блоков в сетке нитей CUDA, при программной реализации **Алгоритма I** использовалась двумерная сетка блоков `dim3 grid(NBlksX,NBlksY,1)`, где `NBlksX`, `NBlksY` — соответствующие числа блоков, для **Алгоритма II** и элементов высоких порядков достаточно одномерной сетки (`NBlksY=1`).

В **Алгоритме I** выполняется m CUDA-нитей, каждая из который вычисляет одну мат-

рицу жесткости K^e и не имеет общих данных с другими нитями. Каждой нити передаются массивы координат узлов конечных элементов, характеристик материала, порядки аппроксимирующих полиномов.

С повышением порядка полиномов увеличиваются размеры матриц производных функций формы, как в локальных, так и в глобальных координатах, которые для шестигранных конечных элементов с тремя неизвестными в узлах, при $p = 5$ требуют 21312 байт для хранения матриц B_α^{eT} и DB_α^{eT} и 1776 байт — для матрицы локальных производных. Матрица K^e, хранящаяся с учетом ее симметрии занимает в этом случае 198024 байт памяти и не может быть помещена полностью в регистровую, разделяемую, или локальную память графического ускорителя и поэтому множество матриц K^e хранится в глобальной памяти.

Алгоритм II рассматривался в рамках статической модели, с фиксированным числом нитей $m \times n_G$ и суммированием матриц K^e, получаемых нитями в разных точках интегрирования. Как показали эксперименты, использование атомарной операции суммирования медленнее, чем реализация **Алгоритма I**, обладающего меньшей степенью параллелизма. В программной реализации **Алгоритма II** используется разделяемая память ускорителя, а вычисления организуются следующим образом. Блоком нитей CUDA ($N_{tib} = n_G$) параллельно вычисляются локальные матрицы K^e во всех точках интегрирования одного конечного элемента, каждая нить выполняет вычисления в соответствующей точке интегрирования. Полученное в (3) значение $K^e_{\alpha\beta}$ помещается в разделяемую память ускорителя, в которой создается массив `Kij`, размерности $\min\{2^k | 2^k \geqslant n_G\}$, $k \in \mathbb{N}$. Элементы массива `Kij` суммируются при помощи алгоритма сдваивания, результат операции передается в глобальную память, где хранится $K^e_{\alpha\beta}$.

3. Результаты вычислительных экспериментов.

Предложенные варианты распараллеливания применялись при решении динамической задачи теории упругости в плоской и трехмерной постановке (на сетке шестигранных конечных элементах, далее обозначенной, как 3D сетка) [3]. Вычислительные эксперименты выполнены на узлах кластера X4 (Институт механики УрО РАН) следующих конфигураций: два CPU Intel Xeon E5430 Quad Core 2.66GHz, 8 ГБ оперативной памяти и одна видеокарта GeForce GTX 580 (1536 CUDA ядер, 3 ГБ графической памяти); два CPU Intel Xeon E5-2609 Quad Core 2.4GHz, 32 ГБ оперативной памяти и две видеокарты GeForce GTX 680 (1536 CUDA ядер, 4 ГБ графической памяти в каждой).

Таблица 1. Время вычисления (сек.) и ускорение на GPU

3D сетка, $m = 10000$, $(N_e \times N_e)$	t_{CPU} E5430/E5-2609		$\frac{t_{CPU}}{t_{GPU}}$ GTX580/GTX680	
	`-O0`	`-O3`	I	II
$p = 1, n_G = 8, (24 \times 24)$	1.23/1.12	0.17/0.14	0.41/0.74	0.39/0.67
$p = 2, n_G = 8, (60 \times 60)$	5.73/5.20	0.74/0.58	1.09/1.08	0.90/1.05
$p = 3, n_G = 64, (96 \times 96)$	108.30/97.45	13.30/10.52	2.68/1.48	7.07/6.41
$p = 4, n_G = 125, (150 \times 150)$	491.10/441.04	59.02/47.52	2.69/1.41	8.41/6.75
$p = 5, n_G = 216, (222 \times 222)$	1806.84/1619.41	213.17/172.07	2.66/1.39	7.34/5.48

В приводимых результатах, под ускорением понимается отношение времени вычисления m матриц жесткости оптимизированным (`-O3`) исполняемым кодом на одном ядре централь-

ного процессора (t_{CPU}) к времени вычисления на некоторой конфигурации вычислительной системы: графическом ускорителе t_{GPU}, двух ускорителях, восьми ядрах CPU ($t_{8\times CPU}$) и их комбинациях. Время вычислений на GPU включает: вычисления, копирование данных в память ускорителя и возвращение результата в оперативную память. Представленные результаты, получены при вычислениях с двойной точностью.

При интегрировании матриц жесткости для четырехугольных конечных элементов первого порядка ($p = 1$), предложенные параллельные алгоритмы обеспечивают ускорение вычислений начиная с $\lg m \geqslant 4$, для шестигранных элементов — при $\lg m \geqslant 5$. В этих случаях, **Алгоритм I** эффективнее **Алгоритма II** в полтора раза в двумерном и 1.2 раза в трехмерном случае. Модификация **Алгоритма II**, в которой блок нитей интегрирует несколько матриц K^e из (1), а N_{tib} равно числу нитей в `warp`, обеспечила ускорение уже относительно **Алгоритма I** в полтора раза в двумерном и двукратное — в трехмерном случае.

Проведены эксперименты по интегрированию матриц K^e для сетки с фиксированным числом $m = 10000$ при различных порядках аппроксимации $p = 1, 2, ..., 5$. При $p \geqslant 3$ (см. таблицу 1), **Алгоритм II** обеспечил ускорение относительно **Алгоритма I** до трех раз при вычислениях на видеокарте GeForce GTX 580 и четыре – пять раз на GTX 680.

На видеокартах GeForce GTX 680 выполнено сравнение с ускорениями, полученными при распараллеливании средствами технологии OpenMP (см. рисунки 1,2). В ходе экспериментов получено, что восемь нитей OpenMP обеспечили ускорение $t_{CPU}/t_{8\times CPU}$ в 7–7.7 раза, а применение одного GPU — только пять с половиной раз (рис. 2). Для двух GPU, kernel-функции численного интегрирования (**Алгоритм II**) вызывались из двух нитей OpenMP, при $p = 5$ получено ускорение примерно в полтора раза относительно восьми нитей OpenMP, запущенных на восьми ядрах CPU.

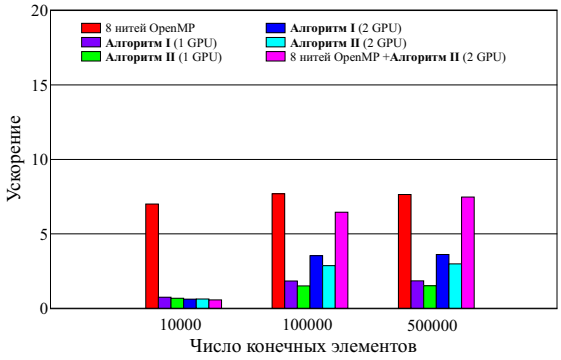

Рис. 1. Ускорение при интегрировании матриц жесткости конечных элементов первого порядка ($p = 1$).

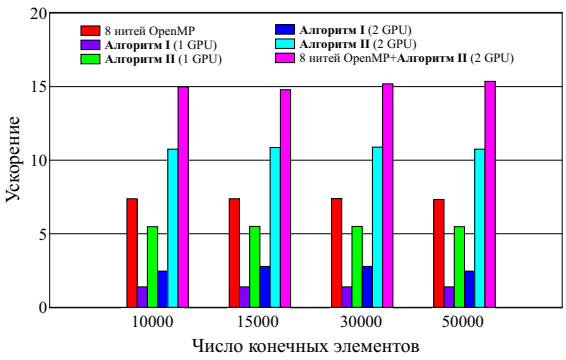

Рис. 2. Ускорение при интегрировании матриц жесткости конечных элементов пятого порядка ($p = 5$).

Рассмотрены варианты распараллеливания с распределением вычислений между центральными процессорами и GPU (на рисунках обозначен «8 нитей OpenMP + **Алгоритм II** (2 GPU)», в этом случае всего выполнялось десять нитей OpenMP). При однородном ($p = 5$) порядке аппроксимации достигнуто ускорение более пятнадцать раз, что более чем в два раза больше, чем в варианте восьми нитей OpenMP. В этом случае, 20000 матриц K^e интегрировались на CPU и 30000 — на двух GPU. Разделение вычислительной нагрузки между ядрами CPU и видеокартами позволило рассматривать варианты, превышающие доступный объем графической памяти.

В случае неоднородной аппроксимации ($p \in \{1, 3\}$) на сетке из $m = 100000$ шестигранных конечных элементов получено тринадцатикратное ускорение при интегрировании $m|_{p=1} = 50000$ матриц K^e на восьми ядрах CPU и $m|_{p=3} = 50000$ матриц на двух GPU.

Неоднородное распределение порядков, характерное для адаптивных версий МКЭ с по-

вышением порядка аппроксимирующих функций требует балансировки нагрузки на нескольких уровнях: между центральными процессорами и графическими ускорителями, между ядрами CPU, между ядрами графических ускорителей.

Заключение

Проведенные исследования показали, что в ряде случаев возможности современных компиляторов по оптимизации программного кода и параллельные технологии, применяемые для центральных процессоров успешно конкурируют с GPGPU, а параллельные алгоритмы должны использовать ядра центральных и графических процессоров.

Эффективное применение графических процессоров при вычислении конечно-элементных матриц в больших математических моделях связано с разделением вычислительной нагрузки между всеми ресурсами гибридной архитектуры с учетом их производительности и требуемой памяти. При интегрировании матриц конечных элементов высоких порядков, существенное ускорение достигается, когда на один ускоритель приходится в несколько раз больше конечных элементов, чем на одно ядро CPU. В случае неоднородного распределения порядков p-/hp-версии МКЭ конечные элементы меньших степеней предпочтительнее интегрировать на ядрах CPU, а больших степеней — на ядрах графических процессоров.

Отметим, что массивный параллелизм GPU наиболее эффективно использовался при распараллеливании на уровне точек интегрирования, а предложенные алгоритмы хорошо масштабируются при увеличении числа используемых графических процессоров.

Литература

1. Morozov D.N., Chetverushkin B.N., Churbanova N.G., Trapeznikova M.A. An Explicit Algorithm for Porous Media Flow Simulation using GPUs // Proc. of the Second Int. Conf. on Parallel, Distributed, Grid and Cloud Comp. for Eng. Stirlingshire. UK. 2011. Paper 19.

2. Губайдуллин Д.А., Никифоров А.И., Садовников Р.В. Об особенностях использования архитектуры гетерогенного кластера для решения задач механики сплошных сред // Вычислительные методы и программирование. 2011. Т. 12. С. 450–460.

3. 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. Vol. 7979. P. 65–79.

4. Szabo B., Duster A., Rank E. The p-version of the Finite Element Method // Encyclopedia of Computational Mechanics. Volume 1: Fundaementals. 2004. P. 119-139.

5. Plaszewski P., Maciol P., Banas K. Finite Element Numerical Integration on GPUs // Lecture Notes in Computer Science. 2010. Vol. 6067. P. 411–420.

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., Кузьмин И.М., Тонков Л.Е. Алгоритмическое и программное обеспечение решения задач взаимодействия конструкции с жидкостью/газом на гибридных вычислительных системах // Компьютерные исследования и моделирование. 2013. Т. 5. №2. С. 153–164.