УДК 004.272.26

Д.Г. Прибыток[1], Э.Н. Середин[2]

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ МОДЕЛИРОВАНИЯ ТРЕХМЕРНОГО ТЕЧЕНИЯ СТОКСА МЕТОДОМ ГРАНИЧНЫХ ЭЛЕМЕНТОВ

Рассматривается применение техники параллельных вычислений в моделировании трехмерного течения вязкой жидкости (течения Стокса) прямым методом граничных элементов. Задача решается в три этапа: дискретизация и построение системы линейных алгебраических уравнений (СЛАУ), ее решение, нахождение вектора скорости жидкости в заданных точках. Для построения СЛАУ и нахождения вектора скорости разрабатываются и реализовываются параллельные алгоритмы с помощью технологии программирования видеокарт CUDA. Для решения СЛАУ используются готовые программные библиотеки. Проводится сравнение временных затрат для трех основных алгоритмов на примере расчета движения вязкой жидкости в трехмерной каверне.

Введение

С началом появления многопроцессорных устройств возник вопрос, каким образом использовать их вычислительную мощность наиболее эффективно. Так появилось целое направление в области численных методов – параллельные вычисления. Однако при разработке параллельных версий многих алгоритмов оказалось, что далеко не все из них могут быть эффективно распараллелены или по разным причинам не поддаются распараллеливанию вообще. Тем не менее для многих научных, инженерных и прикладных задач имеется достаточно большое количество алгоритмов, для которых довольно успешно разрабатываются их параллельные аналоги [1, 2]. Рассмотрим создание параллельного алгоритма на примере моделирования трехмерных течений Стокса и проведем сравнительный анализ быстродействия отдельных частей алгоритма с их последовательными версиями.

Течение Стокса [3, 4] – это модель движения жидкости, которая выводится из уравнений Навье – Стокса при допущении малости числа Рейнольдса, Re << 1. Данная модель может использоваться для описания течения вязких полимеров, течения лавы, в теории смазки, для моделирования движения крови в организме человека и т. д. Математическая модель для трехмерного случая может быть представлена как в виде системы дифференциальных уравнений

$$\nabla p - \mu \Delta \vec{u} = 0; \tag{1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2}$$

с граничными условиями скорости или напряжения, так и в интегральной форме

$$\xi(x_0) \cdot u_j(x_0) = -\frac{1}{\mu} \oint_{\Omega} \varphi_i(x) \cdot G_{ij}(x_0 - x) d\Omega + \oint_{\Omega} u_i \cdot T_{ijk}(x_0 - x) \cdot n_k(x) d\Omega, \tag{3}$$

где i, j, k могут принимать значения 1, 2, 3;

$\vec{u}$ – трехмерный вектор скорости жидкости;

p – значение нормального давления на стенках элементарного объема жидкости;

$\mu = \text{const}$ – коэффициент динамической вязкости жидкости;

Ω – граница области решения V, $\Omega \subset V$;

x – точка, принадлежащая Ω;

$n(x)$ – вектор внешней нормали к Ω в точке x;

x_0 – произвольная точка пространства;

$\xi(x_0)$ – весовая функция, $\xi(x_0)=\begin{cases}8, \text{если } x_0 \in V,\\ 4, \text{если } x_0 \in \Omega,\\ 0, \text{если } x_0 \notin \Omega \vee x \notin V;\end{cases}$

$G_{ij}(r)$ – тензорный потенциал Стокса [5], $G_{ij}=\dfrac{\delta_{ij}}{|r|}+\dfrac{r_i r_j}{|r|^3}$;

$T_{ijk}(r)$ – тензор, ассоциированный с G_{ij}, $T_{ijk}=-6\dfrac{r_i r_j r_k}{|r|^5}$;

$\varphi(x)$ – вектор напряжений на границе Ω в точке x.

Далее будем рассматривать интегральное представление уравнений Стокса. Их решение строится методом граничных элементов [4, 6], главной особенностью которого является необходимость в дискретизации лишь границы области решения, что уменьшает количество дискретных элементов (граничных элементов (ГЭ)) в задаче, а также позволяет решать задачи в областях со сложной геометрией, как, например, в случае многосвязных областей [7]. Пожалуй, основным недостатком метода является большая потребность в оперативной памяти, вытекающая из высокой плотности матрицы СЛАУ, которая имеет размер $3N \times 3N$, где N – количество ГЭ.

1. Численная аппроксимация интегральных уравнений. Последовательный алгоритм

Разобьем границу трехмерной области на N граничных элементов, представляющих собой пространственные прямоугольники либо треугольники. В центре каждого ГЭ зададим локальную систему координат. Будем предполагать, что функции напряжений $\varphi_i(x)$ являются на нем постоянными. Для каждого ГЭ зададим в качестве граничного условия вектор скорости. В уравнении (3), поочередно помещая рассматриваемую точку x_0 в центры полученных элементов, получим следующий вид интегрального уравнения:

$$u(\xi_0^p)=-\frac{1}{4\pi\mu}\sum_{q=1}^{N}[\vec{\varphi}(\xi_0^q)\cdot A_{qp}\cdot\iint_{S_q}G_{ij}(r_{pq})dS_q]+\frac{1}{4}I^{\Gamma}(\xi_0^p); \tag{4}$$

$$I^{\Gamma}(\xi_0^p)=\sum_{q=1}^{N}[\vec{u}(\xi_0^q)\cdot A_{qp}\cdot\iint_{S_q}T_{ijk}(r_{pq})\cdot(0,\ 0,\ 1)dS_q]; \tag{5}$$

$$r_{pq}=A_{gq}\cdot(\xi_0^q-\xi_0^p)-(\xi_x,\ \xi_y,\ 0),$$

где $i,\ j,\ k=1,\ 2,\ 3;\ p,\ q=1,\ 2,\ 3,\ ...,\ N$;

$\xi_0^p,\ \xi_0^q$ – центры элементов с индексами p, q;

A_{qp} – матрица перехода от q-й к p-й системе координат;

A_{gq} – матрица перехода от глобальной к q-й системе координат;

S – область ГЭ, определенная в его декартовой системе координат $O\xi_x\xi_y$.

Таким образом, в результате получим систему линейных уравнений относительно неизвестных значений φ:

$$C\cdot\varphi=B; \tag{6}$$

$$C_{pq} = \frac{1}{4\pi\mu} A_{qp} \cdot \iint_{S_q} G_{ij}(r_{pq}) dS_q ; \tag{7}$$

$$B_p = -\vec{u}_p + \frac{1}{4}\sum_{q=1}^{N} [\vec{u}(\xi_0^q) \cdot A_{qp} \cdot \iint_{S_q} T_{ijk}(r_{pq}) \cdot (0,0,1) dS_q]. \tag{8}$$

После решения СЛАУ (6) скорость в любой точке x внутри области решения можно найти из выражений

$$u(x) = \sum_{q=1}^{N} A_{qg} \cdot (\vec{u}(\xi_0^q) \cdot U_q(x) - \vec{\varphi}(\xi_0^q) \cdot F_q(x)) ; \tag{9}$$

$$U_q(x) = \frac{1}{8} \iint_{S_q} G_{ij}(r_{xq}) dS_q ; \tag{10}$$

$$F_q(x) = \frac{1}{8\pi\mu} \iint_{S_q} T_{ijk}(r_{xq}) \cdot (0,0,1) dS_q ; \tag{11}$$

$$r_{xq} = A_{gq} \cdot (x - \xi_0^q) - (\xi_x, \xi_y, 0),$$

где A_{qg} – матрица перехода от *q*-й системы координат к глобальной.

2. Параллельный алгоритм построения СЛАУ

Преимуществом параллельного алгоритма построения СЛАУ (6)–(8) с помощью технологии программирования видеокарт CUDA является возможность вычислять коэффициенты матрицы C_{pq} полностью независимо друг от друга, параллельно. Что касается выражения для свободного члена B_p, то здесь необходимо применять специальную методику вычисления суммы, называемую редукцией [8].

Создадим двумерную сетку блоков размером *N*/8×*N*/8 или *N*/16×*N*/16, для каждого блока – двумерную конфигурацию нитей размерами 8×8 или 16×16. При этом для оптимизации вычислений количество блоков будем выбирать таким образом, чтобы оно было кратно степени двойки. В глобальную память видеокарты поместим массивы граничных элементов и граничных условий, а также пустые массивы для матрицы и свободного члена СЛАУ, которые и будем заполнять. Для быстрого вычисления сумм с помощью редукции выделим разделяемую память в блоках. Используя для каждой нити доступ к ее глобальным индексам (*p*, *q*) в сетке, поставим ей в соответствие и затем вычислим значение C_{pq}, которое является тензором третьего порядка. Полученное значение запишем в выходной массив с учетом особенностей используемых программных библиотек cuSOLVER или cula [9], для которых матрицу C надо предварительно транспонировать.

Для каждой запущенной нити с индексами (*p*, *q*) вычислим значение $B_p = \frac{1}{4} A_{qp} \cdot \iint_S T_{ijk}(r_{pq}) \cdot (0,0,1) dS$, затем умножим его на граничное условие $\vec{u}_q$ и просуммируем полученные значения для всех нитей блока *p*-й строки, используя методику редукции. Для окончательного нахождения значения B_p в соответствии с выражением (8) отнимем от полученной суммы значение граничного условия $\vec{u}_p$.

Стоит также отметить, что ускорение расчетов, получаемое от разработанного параллельного алгоритма заполнения матрицы C, будет зависеть от используемой видеокарты с поддержкой технологии CUDA. Вычисление свободного члена СЛАУ с использованием методики редукции выполняется за $O(\log_2 n)$ операций, где n – количество запущенных нитей в блоке.

3. Использование библиотек cuSOLVER, cula для решения СЛАУ

Для решения полученной СЛАУ, которая обладает высокой плотностью, можно применять готовые программные библиотеки (например, cuSOLVER, cula), реализующие в параллельном режиме методы линейной алгебры с помощью технологии CUDA. Библиотека coSOLVER является относительно новой и не работает на видеокартах с поддержкой compute capability ниже 2.0 [9]. Поэтому для остальных устройств можно использовать библиотеку cula.

Что касается использования cuSOLVER, в официальной документации [10] показан пример решения СЛАУ. Последовательное применение функций geqrf, ormqr, trsm позволяет при помощи QR-разложения и последующих преобразований найти решение. При использовании cula достаточно воспользоваться готовой функцией culaDeviceSgesv, которая применяет методику LU-разложения. Обе библиотеки предоставляют интерфейс, работающий напрямую с памятью видеокарты, что позволяет уменьшить количество операций обмена данными между видеокартой и центральным процессором.

4. Параллельный алгоритм нахождения скорости в заданных точках

После нахождения неизвестных значений напряжений φ будем использовать выражение (9) для вычисления скоростей жидкости в наборе заданных точек. Создадим прямоугольную сетку блоков размером $N \times M$, где M – общее число точек, для которых необходимо найти вектор скорости потока. При этом количество блоков в горизонтальном направлении выберем кратным степени двойки. В глобальную память видеокарты поместим массив граничных элементов, граничных условий, найденных значений напряжений φ и массив координат точек, для которых необходимо найти скорости жидкости. Последний будет служить одновременно для записи найденных значений. Для каждой нити в блоке с координатами (q, i) поставим в соответствие и вычислим значение слагаемого суммы $A_{qg} \cdot [\vec{u}_q \cdot U_q(x_i) + \vec{\varphi}_q \cdot F_q(x_i)]$. По аналогии с вычислением свободного члена СЛАУ используем методику редукции для нахождения суммы (9).

5. Течение вязкой жидкости в трехмерной каверне

На примере расчета течения жидкости в трехмерной каверне (рис. 1 и 2) сравним временные затраты на основные алгоритмы между последовательным и параллельным аналогами для нескольких процессоров и видеокарт с поддержкой технологии программирования CUDA, характеристики которых представлены в табл. 1 и 2. Для дискретизации каверны будем использовать поочередно 20, 22, 24, 26, 28, 30 и 32 части для разбиения каждого из ее ребер, получив таким образом 2400, 2904, 3456, 4056, 4704, 5400 и 6144 граничных элемента. Поиск скоростей жидкости будем проводить в 3600 точках в центральном сечении каверны. Результаты измерений показаны в табл. 3–5. Ввиду недостаточного количества памяти видеокарт GPU_1 и GPU_4 расчеты были проведены не для всех значений.

Таким образом, для алгоритма построения СЛАУ ускорение работы программы будет следующим: при использовании видеокарты – до 200 раз, для алгоритма решения СЛАУ – до 943 раз, для алгоритма поиска вектора скорости в заданных точках – до 242 раз. При расчетах на процессорах использовалась нераспараллеленная версия алгоритма. Основное время работы программы приходится на решение СЛАУ как в последовательном, так и в параллельном режимах. Наибольшее ускорение расчетов за счет распараллеливания достигается для алгоритма решения СЛАУ, что обусловлено хорошей оптимизацией программных библиотек cuSOLVER и cula.

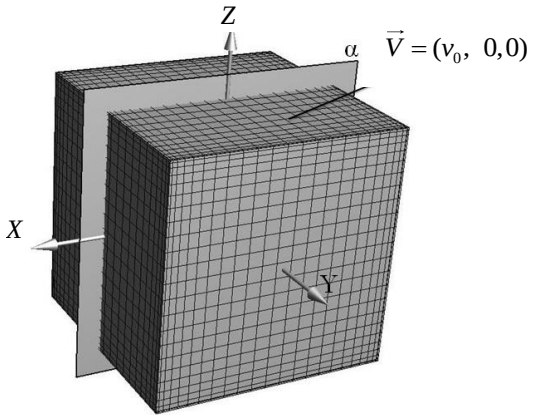

Рис. 1. Гранично-элементная сетка каверны. За счет движения верхней грани куба создается поле течения жидкости внутри него. Поле скорости ищется в центральном сечении α

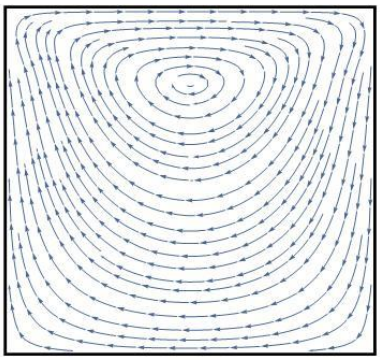

Рис. 2. Траектории движения частиц жидкости в центральном сечении α каверны

Таблица 1

Краткая спецификация процессоров

Процессор (архитектура)	CPU_1	CPU_2	CPU_3	CPU_4	CPU_5
	Intel® Core™ i3-530	Intel® Core™ i5-4200M	Intel® Core™ i7-4710HQ	Intel® Core™ E8600	Intel® Core™ i7-4770K
Количество ядер (потоков)	2(2)	2(4)	4(8)	2(2)	4(8)
Частота, ГГц	2,93	2,5	2,5	3,33	3,5

Таблица 2

Краткая спецификация видеокарт с поддержкой CUDA

Видеокарта (архитектура)	GPU_1	GPU_2	GPU_3	GPU_4	GPU_5
	GeForce GT 240	GeForce GT 755M	GeForce GTX 850M	GeForce GTX 750Ti	GeForce GTX 670
Частота ядра, МГц	550	980	936	1020	915
Скорость заполнения текстур, миллиард текселов в секунду	17,6	31,4	35,0	40,8	102,5
Количество памяти, Гб	0,5-1	2	2	1-2	2
Тип памяти	GDDR3 GDDR5	GDDR5	DDR3 GDDR5	GDDR5	GDDR5
Интерфейс памяти, бит	128	128	128	128	256
Пропускная способность памяти, Гб/с	57,6	86,4	80	86,4	192,2
CUDA Cores, шт.	96	384	640	640	1344

Таблица 3

Сравнение временных затрат для алгоритма построения СЛАУ, с

Кол-во ГЭ	2400	2904	3456	4056	4704	5400	6144
CPU_1	13,52	19,69	27,71	38,16	51,53	69,27	88,05
CPU_2	10,80	13,33	18,.64	25,59	34,74	46,39	71,86
CPU_3	7,87	11,51	16,59	22,03	29,97	40,37	52,46
CPU_4	9,88	14,36	20,61	27,94	37,78	50,16	65,20
CPU_5	6,80	10,08	13,95	19,51	25,67	36,03	45,19
GPU_1	0,28	0,41	1,05	–	–	–	–
GPU_2	0,25	0,33	0,47	0,64	0,86	1,13	1,47
GPU_3	0,11	0,17	0,24	0,32	0,44	0,57	0,73
GPU_4	0,09	0,11	0,17	0,27	0,36	–	–
GPU_5	0,08	0,10	0,14	0,19	0,26	0,33	0,44

Таблица 4

Сравнение временных затрат для алгоритма решения СЛАУ, с

Кол-во ГЭ	2400	2904	3456	4056	4704	5400	6144
CPU_1	932,5	1637,4	2751,4	4463,1	6933,2	10 535,8	15 536,3
CPU_2	499,0	785,0	1328,6	2212,3	3355,2	5361,0	8120,8
CPU_3	396,0	699,0	1164,9	1904,7	3002,1	4480,7	6700,0
CPU_4	589,9	1044,9	1756,3	2858,1	4432,4	6734,0	10 038,1
CPU_5	336,6	616,9	1048,4	1643,5	2561,6	3888,2	5834,3
GPU_1	2,66	4,48	29,85	–	–	–	–
GPU_2	4,70	6,55	9,79	14,94	21,58	32,02	44,37
GPU_3	3,88	4,62	6,54	9,58	13,87	20,44	31,79
GPU_4	1,90	2,86	3,98	5,66	8,28	–	–
GPU_5	2,13	3,04	4,30	6,20	8,49	12,13	16,47

Таблица 5

Сравнение временных затрат для алгоритма поиска вектора скорости в заданных точках, с

Кол-во ГЭ	2400	2904	3456	4056	4704	5400	6144
CPU_1	19,27	22,99	27,63	32,24	37,24	42,66	48,51
CPU_2	12,25	14,67	17,38	20,39	23,61	31,87	30,93
CPU_3	10,81	13,08	15,44	18,46	20,97	24,63	27,46
CPU_4	13,74	16,61	19,72	23,11	26,84	30,91	35,05
CPU_5	9,58	11,31	13,64	15,64	18,82	21,06	24,02
GPU_1	0,38	0,46	0,55	–	–	–	–
GPU_2	0,25	0,30	0,36	0,42	0,48	0,56	0,65
GPU_3	0,13	0,15	0,18	0,22	0,24	0,29	0,33
GPU_4	0,09	0,11	0,14	0,16	0,19	–	–
GPU_5	0,08	0,09	0,11	0,13	0,16	0,17	0,20

Заключение

Полученные результаты показывают, что за счет применения параллельных алгоритмов можно существенно увеличить скорость вычислений, связанных с моделированием трехмерного течения Стокса, что может быть особенно полезно в динамических задачах, когда на каждом временном шаге необходимо осуществлять численное решение задачи (1)–(3). Что касается представленных алгоритмов используемого метода, то основной проблемой как с точки зрения

потребления памяти, так и скорости работы является решение СЛАУ. В дальнейшем планируется провести оптимизацию реализаций параллельных алгоритмов построения СЛАУ и поиска скоростей в заданных точках. Также необходима разработка новых, более быстрых методов уменьшения размерности СЛАУ и их решения.

Список литературы

1. Сандерс, Дж. Технология CUDA в примерах: введение в программирование графических процессоров / Дж. Сандерс, Э. Кэндрот ; пер. с англ. А.А. Слинкина. – М. : ДМК Пресс, 2011. – 232 с.

2. Середин, Э.Н. Фильтрация и корреляционная обработка изображений с помощью технологии CUDA / Э.Н. Середин, Б.А. Залесский // Весці НАН Беларусі. Сер. фіз.-мат. навук. – 2015. – № 1. – С. 106–116.

3. Brebbia, C.A. Boundary Element Methods in Engineering / C.A. Brebbia. – Berlin : Springer-Verlag, 1982. – 649 p.

4. Pozrikidis, C. Boundary Integral and Singularity Methods for Linearized Viscous Flow / C. Pozrikidis. – Cambridge : Cambridge University Press, 1992. – 259 p.

5. Lisicki, M. Four approaches to hydrodynamic Green's functions – the Oseen tensors / M. Lisicki ; Institute of Theoretical Physics. – Warsaw, 2013.

6. Cheng, A. H.-D. Heritage and early history of the boundary element method / A. H.-D. Cheng, D.T. Cheng. – Elsevier Science, 2003. – P. 268–302.

7. Katsikadelis, J. Boundary Elements: Theory and Applications / J. Katsikadelis. – Elsevier Science, 2002. – 430 p.

8. Боресков, А.В. Основы работы с технологией CUDA / А.В. Боресков, А.А. Харламов. – М. : ДМК Пресс, 2010. – 232с.

9. CUDA Toolkit Documentation [Electronic resource]. – 2015. – Mode of access : http://docs.nvidia.com/cuda/index.html. – Date of access : 21.11.2015.

10. CUDA Toolkit Documentation [Electronic resource]. – 2015. – Mode of access : http://docs.nvidia.com/cuda/cusolver/index.html#ormqr-example1. – Date of access : 22.11.2015.

Поступила 13.01.2016

[1]*Гданьский политехнический университет,*
Польша, ул. Нарутовича, 11-12
e-mail: dpribytok@tut.by

[2]*Объединенный институт проблем*
информатики НАН Беларуси,
Минск, Сурганова, 6
e-mail: eduard.seredin@tut.by

D.G. Pribytok, E.N. Seredin

PARALLEL ALGORITHM FOR THREE-DIMENSIONAL STOKES FLOW SIMULATION USING BOUNDARY ELEMENT METHOD

Parallel computing technique for modeling three-dimensional viscous flow (Stokes flow) using direct boundary element method is presented. The problem is solved in three phases: sampling and construction of system of linear algebraic equations (SLAE), its decision and finding the velocity of liquid at predetermined points. For construction of the system and finding the velocity, the parallel algorithms using graphics CUDA cards programming technology have been developed and implemented. To solve the system of linear algebraic equations the implemented software libraries are used. A comparison of time consumption for three main algorithms on the example of calculation of viscous fluid motion in three-dimensional cavity is performed.