Секция №1, устный
УДК 519.6

ИСПОЛЬЗОВАНИЕ GPU ДЛЯ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЕЙ, ПОЛУЧЕМЫХ В МЕТОДЕ КОНЕЧНЫХ ЭЛЕМЕНТОВ

Долгун А.А.

Институт нефтегазовой геологии и геофизики СО РАН, Новосибирск

В работе проводится сравнение скорости решения на CPU и GPU систем линейных алгебраических уравнений с разреженной матрицей больших размерностей, получающихся при использовании векторного метода конечных элементов для решения электромагнитных задач. На CPU рассматриваются собственные параллельные реализации и параллельные реализации на основе библиотеки Intel MKL. На GPU исследуются различные форматы хранения разреженных матриц.

Ключевые слова: вычисления на GPU, разреженные матрицы, метод конечных элементов

Использование численных методов для решения уравнений электромагнетизма в задачах геоэлектрики требует больших вычислительных ресурсов. Современные GPU (видеокарты) кроме ускорения вывода двумерной и трехмерной графики позволяют выполнять вычисления с плавающей точкой. При этом теоретическая производительность GPU среднего и высокого класса больше, чем производительность центрального процессора. В результате современные суперкомпьютерные системы представляют собой гибридные машины с узлами, в которых сочетаются несколько многоядерных центральных процессоров и несколько GPU. Поэтому адаптация программ к использованию вычислительных возможностей GPU становится актуальной.

Векторный метод конечных элементов является одним из наиболее эффективных при решении электромагнитных задач. Применяемые в нем векторные пространства Неделека позволяют учитывать непрерывность и скачки компонент электромагнитного поля на границах материалов. Использование конечных элементов разной формы дает возможность хорошо аппроксимировать сложную геометрию расчетной области.

Решение электромагнитных задач с помощью векторного метода конечных элементов приводит к необходимости решения систем линейных алгебраических уравнений (СЛАУ) большой размерности. Наиболее простыми, универсальными и распространенными при аппроксимации сложной геометрии являются тетраэдральные конечные элементы. В результате их использования матрица СЛАУ имеет нерегулярную разреженную структуру. В памяти обычно хранится в разреженном строчном (столбцовом) симметричном или несимметричном формате. Для решения таких СЛАУ применяются итерационные решатели на подпространствах Крылова. С точки зрения алгоритмов, основными операциями в данных решателях являются операции с векторами и умножение матрицы на вектор. Исследованию их реализации на GPU и посвящена эта работа.

Так как наибольшую распространенность на суперкомпьютерах имеют GPU производства компании NVIDIA, в данной работе рассматриваются именно они. Логически GPU представляет собой несколько потоковых мультипроцессоров (SM, Streaming Multiprocessor), каждый из которых содержит десятки исполнительных модулей – ядер CUDA. Каждое ядро CUDA выполняет по 32 потока. Каждый поток исполняет одну и ту же программу над различными частями обрабатываемых данных. Используемая модель вычислений называется SIMT (Single Instruction, Multiple Data) и ставит существенные ограничения на организацию как кода программы, так и данных, необходимую для получения высокой производительности.

Векторные операции являются «естественными» для архитектуры GPU и их реализация не представляет сложности. Поэтому в работе основное внимание направлено на сравнение алгоритмов матрично-векторного умножения для матриц разреженной структуры. В качестве тестовых используются две матрицы, полученные из конечноэлементных задач. Первой взята матрица, возникающая в задаче распространения электрического поля, генерируемого петлей, в проводящем полупространстве с объектом (рис. 1). Во временной области решается уравнение

$$\nabla \times \left(\frac{1}{\mu_0 \mu} \nabla \times \mathbf{E}(\mathbf{x}, t) \right) + \varepsilon_0 \varepsilon \frac{\partial^2 \mathbf{E}(\mathbf{x}, t)}{\partial t^2} + \sigma \frac{\partial \mathbf{E}(\mathbf{x}, t)}{\partial t} = -\frac{\partial \mathbf{J}(\mathbf{x}, t)}{\partial t}.$$

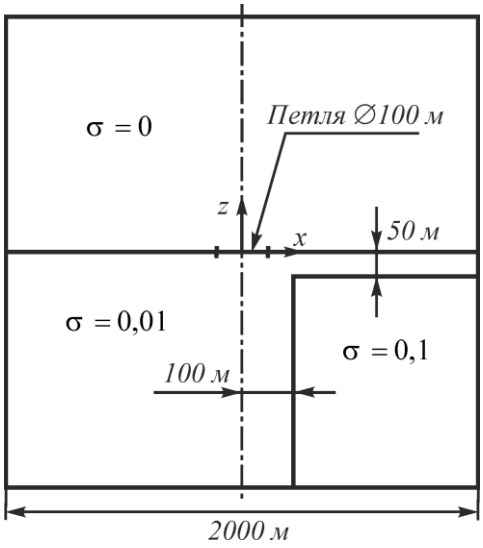

Рис. 1. Расчетная область для задачи с вещественной матрицей

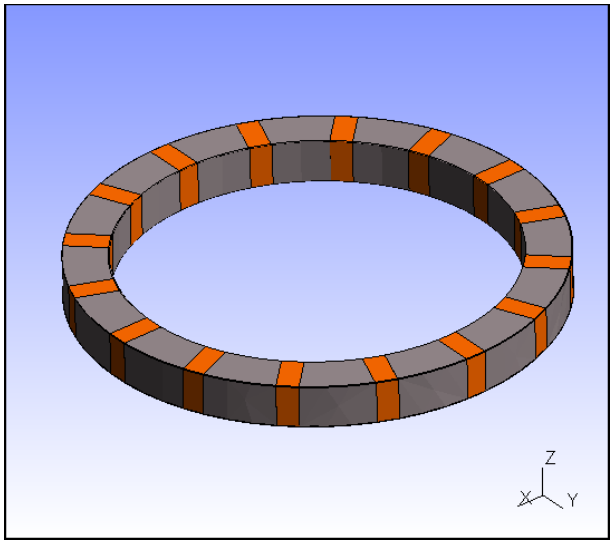

Рис. 2. Расчетная область для задачи с комплексной матрицей

Используются базисные функции первого порядка. Для каждого слоя по времени генерируется СЛАУ с вещественной симметричной матрицей. Для тестов берется матрица и правая часть с одного из слоев. Вторая матрица возникает в задаче расчета магнитного поля, создаваемого тороидальной катушкой с плоскими витками в однородной области (рис. 2). Решается уравнение

$$\nabla \times \left(\frac{1}{\mu_0 \mu} \nabla \times \mathbf{A}(\mathbf{x})\right) + (i\omega\sigma - \omega^2 \varepsilon_0 \varepsilon)\mathbf{A}(\mathbf{x}) = \mathbf{J}(\mathbf{x}).$$

(Магнитное поле вычисляется как $\mathbf{B} = \nabla \times \mathbf{A}$.) Используются базисные функции второго порядка, дающие большую заполненность матрицы, чем на первом порядке. В результате получается СЛАУ с симметричной комплексной матрицей.

Исследование производительности выполняется для умножения матрицы на вектор и процедур решения СЛАУ с обеими матрицами. Сравниваются следующие алгоритмы матрично-векторного умножения на GPU: в разреженном строчном симметричном формате (библиотека CuSparse из CUDA Toolkit), в разреженном строчном несимметричном формате (CuSparse) и в специальном эффективном на GPU гибридном формате. Гибридный формат подходит для несимметричных матриц (для симметричных хранятся оба треугольника) с примерно одинаковым количеством ненулевых элементов в строке и требует больше памяти по сравнению с разреженным строчным несимметричным форматом. Сравнению производительности различных форматов хранения для различных типов матриц посвящены работы [1] и [2]. Гибридный формат реализован самостоятельно на основе алгоритмов, взятых из [3]. Для операций с векторами на GPU применяется библиотека CuBLAS, входящая в состав CUDA Toolkit.

Таблица 1. Умножение вещественной матрицы на вектор (время в миллисекундах)

	gcc	Intel	k100, Intel
Умножение на CPU, последовательная реализация	44	37	52
Умножение на CPU, 2 потока	36	34	36
Умножение на CPU, 4 потока	27	25	28
Умножение на CPU, 6 потоков			27
Умножение на CPU, 8 потоков			29
Умножение на CPU, 10 потоков			33
Умножение на CPU с использованием MKL, 2 потока	25	25	32
Умножение на CPU с использованием MKL, 4 потока	20	20	25
Умножение на CPU с использованием MKL, 6 потоков			19
Умножение на CPU с использованием MKL, 8 потоков			18
Умножение на CPU с использованием MKL, 10 потоков			19
Умножение на GPU с использованием CuSparse, разреженный строчный симметричный формат	86	86	115
Умножение на GPU с использованием CuSparse, разреженный строчный несимметричный формат	13	13	10
Умножение на GPU, гибридный формат	6	6	8

Таблица 2. Умножение комплексной матрицы на вектор (время в миллисекундах)

	gcc	Intel	k100, Intel
Умножение на CPU, последовательная реализация	98	41	112
Умножение на CPU, 2 потока	54	24	63
Умножение на CPU, 4 потока	32	20	34
Умножение на CPU, 6 потоков			27
Умножение на CPU, 8 потоков			22
Умножение на CPU, 10 потоков			23
Умножение на CPU с использованием MKL, 2 потока	19	19	26
Умножение на CPU с использованием MKL, 4 потока	17	17	15
Умножение на CPU с использованием MKL, 6 потоков			15
Умножение на CPU с использованием MKL, 8 потоков			13
Умножение на CPU с использованием MKL, 10 потоков			13
Умножение на GPU с использованием CuSparse, разреженный строчный симметричный формат	38	38	39
Умножение на GPU с использованием CuSparse, разреженный строчный несимметричный формат	11	11	8
Умножение на GPU, гибридный формат	7	7	9

Время счета на GPU с использованием различных форматов хранения сравнивается также с временем счета на центральном процессоре в разреженном строчном симметричном формате. Для этого была выполнена собственная параллельная реализация матрично-векторного умножения и решателей, а также их параллельная реализация с помощью библиотеки Intel MKL (версии 11).

Запуск программ проводился на компьютере с процессором Intel Core i5-2500K 3300 MHz (4 ядра), GPU NVIDIA GeForce GTX 560Ti 1 GB, двухканальной оперативной памятью DDR3 1333 MHz. Также для сравнения был использован кластер k100 в Институте прикладной математики РАН. Узлы кластера состоят из двух процессоров Intel Xeon X5670 2930 MHz (6 ядер) и трех GPU NVIDIA Fermi C2050 3 GB, оперативная память – трехканальная DDR3 1333 MHz. На персональном компьютере применялись компиляторы gcc 4.6.3 и Intel 13.1. На k100 – Intel 12.1.3. CUDA Toolkit версии 4.2.

Вещественная матрица имеет размерность 1160270x1160270, занимает 226 Мбайт в разреженном строчном симметричном формате, 435 Мбайт в несимметричном и 462 Мбайта в гибридном. Комплексная матрица имеет размерность 327630x327630, занимает 207 Мбайт в разреженном строчном симметричном формате, 406 Мбайт в несимметричном и 523 Мбайта в гибридном.

Результаты умножения вещественной матрицы на вектор с использованием собственной параллельной реализации на CPU, параллельной реализации с помощью MKL и трех форматов на GPU представлены в Табл. 1. В столбцах приведено время выполнения для персонального компьютера с компиляторами gcc, Intel и кластера k100 с компилятором Intel. Аналогичные результаты для комплексной матрицы приведены в Табл. 2. В Табл. 3 показано время решения СЛАУ с вещественной матрицей методом minres, в Табл. 4 – СЛАУ с комплексной матрицей двухуровневым решателем [4].

Таблица 3. Решение СЛАУ с вещественной матрицей методом minres (время в секундах)

	gcc	Intel	k100, Intel
minres на CPU, последовательная реализация	679	597	814
minres на CPU, 2 потока	468	423	482
minres на CPU, 4 потока	458	429	366
minres на CPU, 6 потоков			370
minres на CPU, 8 потоков			406
minres на CPU, 10 потоков			443
minres на CPU, MKL, 2 потока	526	527	579
minres на CPU, MKL, 4 потока	470	472	419
minres на CPU, MKL, 6 потоков			382
minres на CPU, MKL, 8 потоков			390
minres на CPU, MKL, 10 потоков			394
minres на GPU, гибридный формат	99	98	125

Таблица 4. Решение СЛАУ с комплексной матрицей двухуровневым решателем (время в секундах)

	gcc	Intel	k100, Intel
Двухуровневый решатель на CPU, последовательная реализация	328	101	275
Двухуровневый решатель на CPU, 2 потока	166	56	156
Двухуровневый решатель на CPU, 4 потока	94	77	103
Двухуровневый решатель на CPU, 6 потоков			79
Двухуровневый решатель на CPU, 8 потоков			70
Двухуровневый решатель на CPU, 10 потоков			83
Двухуровневый решатель на GPU, гибридный формат	24	20	36

По результатам времени выполнения, приведенным в таблицах, можно сделать следующие выводы:

- Использование компилятора Intel позволяет получить немного более быстрый код для вещественных чисел и значительно более быстрый для комплексных.
- Матрично-векторное умножение из библиотеки MKL быстрее собственной реализации, но преимущество теряется при использовании процедур MKL для написания решателя. Этого следовало ожидать, так как реализация решателя с помощью MKL предполагает последовательный код с вызовом процедур MKL, выполняющихся параллельно. В собственной параллельной реализации потоки выполняются все время счета с необходимой синхронизацией. В результате количество синхронизаций меньше. Возможно также, что в MKL потоки запускаются и останавливаются при каждом вызове параллельной процедуры.
- Распараллеливание на CPU позволяет получить ускорение не более чем в 2-3 раза. При увеличении количества потоков более некоторого оптимального числа (4-6-8) идет увеличение времени решения.
- Матрично-векторное умножение на GPU в разреженном строчном симметричном формате дает плохие результаты. Несимметричный формат значительно быстрее, но требует в 2 раза больше памяти.
- Матрично-векторное умножение на GPU GeForce GTX 560Ti в гибридном формате в 1.5-2 раза быстрее, чем в несимметричном разреженном. На k100 оба формата сравнимы.
- Решение СЛАУ на GPU в 5 и более раз быстрее последовательной реализации на CPU и 2-3 раза параллельной.

Литература

1. Bell N., Garland M. Efficient sparse matrix-vector multiplication on CUDA // NVIDIA Technical Report NVR-2008-004. NVIDIA Corporation, 2008.
2. Bell N., Garland M. Implementing sparse matrix-vector multiplication on throughput-oriented processors // Proc. Conf. HPC Networking, Storage and Analysis (SC09), ACM, pp. 1-11, 2009.
3. Bell N., Garland M. CUSP: Generic parallel algorithms for sparse matrix and graph computations. http://code.google.com/p/cusp-library/
4. Nechaev O., Shurina E., Botchev M. Multilevel iterative solvers for the edge finite element solution of the 3D Maxwell equation // Computers & Mathematics with Applications. 2008. Vol. 55, №10, pp. 2346–2362.