УДК 519.683.4

ИСПОЛЬЗОВАНИЕ ГРАФИЧЕСКИХ УСКОРИТЕЛЕЙ И ВЫЧИСЛИТЕЛЬНЫХ СОПРОЦЕССОРОВ ПРИ РЕШЕНИИ ЗАДАЧИ ФИЛЬТРАЦИИ

К. Ю. Богачев[1], А. С. Богатый[1], А. Р. Лапин[1]

Рассматривается реализация алгоритма BiCGStab с блочным ILU(0)-предобусловливателем на графических картах компаний AMD и Nvidia, а также на вычислительном сопроцессоре Intel Xeon Phi. Приведены результаты тестирования этого алгоритма на несимметричных плохо обусловленных матрицах, возникающих при моделировании реальных месторождений с большим числом скважин. Произведено сравнение времени работы данного алгоритма на четырех системах примерно одинаковой стоимости с флагманскими продуктами компаний AMD, Nvidia и Intel.

Ключевые слова: высокопроизводительные вычисления, графические ускорители, язык CUDA, язык OpenCL.

1. Введение. Задача фильтрации вязкой сжимаемой многофазной многокомпонентной смеси сводится к решению системы дифференциальных уравнений, описанной в [1, 2]. В качестве аппроксимации по времени рассматривается наиболее часто используемая полностью неявная схема, а аппроксимация по пространственным переменным проводится методом конечных объемов. Таким образом, первоначальная задача сводится к системе нелинейных алгебраических уравнений. Решение этой системы ищется стандартным методом Ньютона, на каждом шаге которого необходимо решать линейную систему с несимметричной матрицей Якоби. В нашем случае для их решения выгоднее использовать итерационный алгоритм, не требующий настройки итерационных параметров, как это показано в [10].

Вследствие плохой обусловленности матриц этих линейных систем результат, полученный после порядка 1000 итераций, содержит значительную вычислительную погрешность и не может быть использован в качестве приемлемого ответа. Для улучшения параметров сходимости используются предобусловливатели. В настоящей статье мы рассматриваем скорость работы итерационного алгоритма BiCGStab, описанного в [3], с ILU(0)-предобусловливателем на различных системах. Отметим, что мы использовали эффективный способ параллельного построения и применения ILU(0)-предобусловливателя, рассмотренный в статьях [5, 6], который позволяет получать ускорение до 1328 раз на кластере из 512 узлов (4096 процессоров) по сравнению с последовательной версией. В рамках данной работы были приобретены и протестированы четыре различные многоядерные системы примерно одинаковой стоимости с флагманскими продуктами компаний Nvidia, ATI и Intel:

1) графическая карта Nvidia Tesla C2075 с процессором Intel i7 950, памятью DDR3 1600 MHz и отдельным видео (карта Nvidia Tesla C2075 использовалась исключительно в качестве ускорителя);

2) графическая карта ATI FirePro v9800 с процессором Intel i7 950, памятью DDR3 1600 MHz и отдельным видео (карта ATI FirePro v9800 использовалась исключительно в качестве ускорителя);

3) процессоры: два Intel Xeon E5-2680 с памятью DDR3 1600 MHz;

4) процессор Intel Xeon Phi 5110P с памятью DDR3 1053 MHz.

На текущий момент все четыре системы содержат самые производительные продукты, которые можно приобрести для рабочей станции. Для измерения производительности систем 1 и 2 версия CPU программы на C++ была портирована на CUDA и OpenCL.

2. Особенности OpenCL и CUDA. Языки программирования OpenCL и CUDA имеют Си-подобный синтаксис и используются для написания приложений для массивно-параллельных систем. Программы, написанные на OpenCL, могут работать как на картах фирмы Nvidia, так и на картах AMD, в то время как CUDA поддерживается только картами Nvidia. Структуры приложений, написанных на языках OpenCL и CUDA, схожи, поэтому приведем общие принципы устройства программы для графической карты.

[1] Московский государственный университет им. М. В. Ломоносова, механико-математический факультет, Ленинские горы, д. 1, 119899, ГСП-1, Москва; К. Ю. Богачев, доцент, e-mail: bogachev@mech.math.msu.su; А. С. Богатый, студент, e-mail: bogatyia@gmail.com; А. Р. Лапин, студент, e-mail: lapinra@gmail.com

Обе видеокарты являются симметричными многопроцессорными системами (SMP) из многоядерных процессоров. Эти процессоры в видеокартах Nvidia называются потоковыми мультипроцессорами (Streaming Multiprocessor, SM), а в видеокартах AMD они называются Compute Units, CU. При этом Tesla C2075 содержит 14 процессоров по 32 ядра (всего 448 ядер), а FirePro v9800 содержит 20 процессоров по 80 ядер (всего 1600 ядер).

Каждый поток, работающий на видеокарте, принадлежит определенной группе потоков. Синхронизация потоков между собой, не требующая значительного времени, возможна только внутри одной группы потоков. Это связано с тем, что одна группа потоков работает только на одном процессоре, поэтому синхронизация потоков, работающих на разных процессорах, является времязатратной операцией. В видеокартах один процессор может обрабатывать несколько групп потоков одновременно, поэтому для повышения производительности желательно запускать число групп, кратное числу процессоров (для Tesla C2075 это 8 групп на процессор — всего 112 групп, для FirePro v9800 это 3 группы на процессор — всего 60 групп).

Кроме того, для повышения производительности необходимо подбирать оптимальный размер группы, при этом максимальный размер группы определяется видеокартой (1024 для Tesla C2075 и 256 для FirePro v9800).

В языках OpenCL и CUDA имеется несколько основных видов памяти: общая память; память, разделяемая группой потоков; локальные переменные потока.

Общая память является самой объемной (4 Гб у FirePro v9800 и 6 Гб у Tesla C2075), но в то же время самой медленной. Чтобы скрыть латентность общей памяти, необходимо заводить число потоков, значительно превышающее количество ядер на видеокарте. Разделяемая память работает на порядки быстрее, но ее размер существенно меньше (32 Кб у FirePro v9800 и 64 Кб у Tesla C2075); доступ к ней имеют только потоки из одной группы. Если к какому-либо участку общей памяти происходят частые обращения, то для большей эффективности требуется скопировать этот участок в разделяемую память.

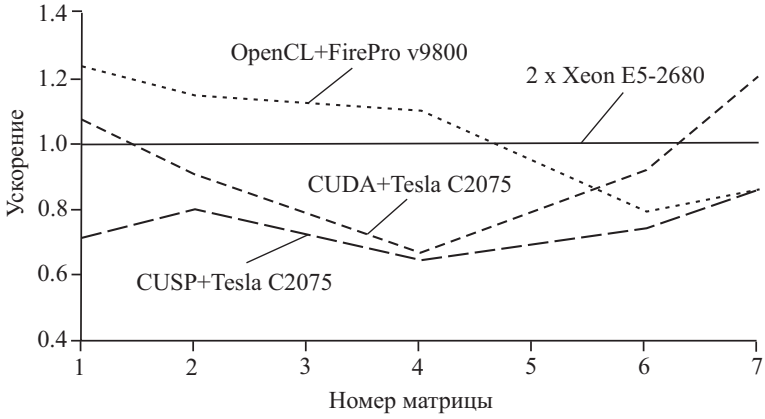

Рис. 1. Ускорение относительно 2 x Xeon E5-2680

Кроме того, каждый из этих языков программирования имеет ряд индивидуальных особенностей, про которые можно узнать в [8, 9].

Таким образом, код параллельной программы, работающий на CPU, при переносе на GPU требует значительной реорганизации с учетом предоставляемых видов памяти, возможных группировок потоков и их синхронизации.

3. Описание алгоритма. Матрицы, возникающие при моделировании месторождений с большим количеством скважин, являются сильно разреженными, асимметричными (но имеют симметричный портрет) и плохо обусловленными. Из-за плохой обусловленности матриц алгоритм BiCGStab почти никогда не сходится за примерно 1000 итераций (при большом количестве итераций накопленная погрешность не позволяет получить решение с нужной точностью), поэтому возникает необходимость использования эффективного предобусловливателя.

4. Итерационный алгоритм. В итерационном алгоритме BiCGStab (подробное описание можно найти в [3]) используются только операции вида $\overline{y} = \alpha\,\overline{x} + \beta$, $\overline{y} = A\,\overline{x}$, $\alpha = (\overline{x}, \overline{x})$. Эти операции легко распараллеливаются и эффективно ложатся на архитектуру GPU.

Несмотря на то что матрицы, возникающие при моделировании, плохо обусловлены, было получено несколько матриц, на которых BiCGStab сходится без предобусловливателя. Кроме нашей реализации этого алгоритма на GPU мы добавили сравнение с временем работы библиотеки CUSP от Nvidia. Для тестирования библиотеки матрицы сначала переводились из блочного msr-формата, описанного в [5], в HYB-формат (заявлен как наиболее эффективный для SpMV-операций), а потом загружались на GPU. Время преобразования формата и копирования матрицы на GPU не учитывалось (рис. 1).

Отметим, что вычисления производились в типе double, и на всех системах алгоритм делал почти одинаковое число итераций. Характеристики матриц можно найти в табл. 1. Как видно из рис. 1, наши

реализации алгоритма BiCGStab на CUDA и OpenCL на всех приведенных матрицах работают быстрее реализации из библиотеки CUSP, а на некоторых матрицах обгоняют 2 x Xeon E5-2680, что говорит о том, что алгоритм был перенесен с CPU достаточно эффективно.

Во многих статьях утверждалось, что итерационный алгоритм значительно ускоряется при переходе на GPGPU-вычисления, однако мы наблюдали лишь незначительное изменение времени его работы. Мы считаем, что это связано со следующими факторами.

1. Часто сравнивают производительность последовательного алгоритма на CPU с параллельным алгоритмом на GPU, в то время как у нас все вычисления на двухпроцессорной системе велись в 32 потока (16 реальных ядер в системе и еще 16 от технологии hyperthreading). Вычисления в 32 потока оказались быстрее последовательной версии в 16 раз.

2. Часто в сравнениях не пишут производительность памяти, которая играет ключевую роль в данной задаче. Мы использовали память DDR3 1600 MHz, которая значительно быстрее устаревшей памяти DDR2 677 MHz.

3. Часто сравнение производится с давно устаревшими моделями процессоров. Мы использовали самый новый процессор компании Intel, имеющий стоимость, сравнимую со стоимостью вычислительных систем на основе графических процессоров, и четыре канала памяти.

Таблица 1

№ матрицы	Число уравнений	Число ненул. элементов	Макс. число ненул. элементов в строке	№ матрицы	Число уравнений	Число ненул. элементов	Макс. число ненул. элементов в строке
1	2702850	26985356	14	24	1749732	22905936	56
2	575025	10849131	21	25	510750	10410048	42
3	589580	7622080	80	26	815932	10972520	106
4	1393869	26825013	48	27	432796	5033320	14
5	4970010	79366986	54	28	44872	583520	20
6	730728	13851342	21	29	203272	2646256	18
7	182682	3454722	21	30	679262	8763844	44
8	364002	4802308	30	31	1028625	20160621	117
9	7256967	148722255	27	32	172710	3213180	27
10	43461	873999	96	33	748776	9751272	80
11	382580	4802024	36	34	281457	4970583	72
12	2348076	44292186	30	35	895212	18392508	30
13	27000	543240	24	36	175760	2055024	14
14	2400402	29471964	56	37	499322	5967940	94
15	759746	7621772	14	38	65864	816968	24
16	766098	15492708	51	39	568006	6678900	72
17	205650	2798004	98	40	1391786	16968996	28
18	1353710	18144132	60	41	1965948	36503964	135
19	1121196	22110966	69	42	199490	2596300	22
20	186886	2101340	18	43	4863578	63273260	96
21	702405	14487201	21	44	116412	2070468	24
22	512740	6274024	34	45	63846	634444	10
23	2715768	34318624	130	46	565086	7147916	14

5. Предобусловливатель. Мы рассматривали предобусловливатели ILU-класса. Существует три основных метода их распараллеливания.

Первый метод — это выделение компонент связности в матрице и параллельный расчет всех компонент одновременно, метод подробно описан в [7]. Почти во всех матрицах линейных систем, полученных при моделировании месторождений, была лишь одна компонента связности, и первый метод распараллеливания оказался к ним неприменим.

Второй метод носит название Block-Jacobi. Недостатком этого метода, отмеченным в [7], является то, что при увеличении числа блоков разбиения качество предобусловливателя заметно ухудшается.

Третий метод — метод Капорина–Коньшина разбиения матрицы на блоки с перекрытиями, который был предложен для решения линейных систем с несимметричными матрицами в [4, 6]. Этот метод построения ILU(0)-предобусловливателя, подробно описанный в [5, 6], мы решили перенести на GPU.

Выбранный метод распараллеливания является эффективным и позволяет получать ускорение относительно последовательной версии до 1328 раз на кластере из 512 узлов (4096 процессоров). Процедуры построения и применения предобусловливателя на GPU аналогичны соответствующим функциям, реализованным для CPU. Каждая часть предобусловливателя обрабатывается соответствующей группой

потоков. Однако с увеличением числа частей в этом алгоритме растет и размер используемой памяти. На карте Tesla C2075 с 6 Гб памяти ее хватает максимум на 170 кусков разбиения с перекрытиями на линейной системе с 7 миллионами неизвестных.

Была выбрана следующая реализация: матрица разбивается на несколько кусков с перекрытиями (оптимальным количеством оказалось 112 у Tesla C2075 (в 8 раз больше числа Streaming Multiprocessors) и 64 у FirePro v9800), а каждая часть считается независимо от остальных. Внутри каждого куска разбиения строки должны обрабатываться последовательно, поэтому ускорения можно добиться только за счет распараллеливания процедуры обработки строки. Это удалось реализовать за счет возможности обработки строки группой потоков. При этом, благодаря блочной структуре хранения матрицы, каждый блок строки также может обрабатываться параллельно. В то же время нет возможности задействовать все ресурсы видеокарты из-за малого количества ненулевых элементов в каждой строке.

6. Результаты. Параметры сравнения:

1) на четырех системах одинаковой стоимости использовались все ядра: 60 на Intel Xeon Phi 5110P, 16 на 2 x Intel Xeon E5-2680, 448 на Tesla C2075 и 1600 на ATI FirePro v9800;

2) измерялось время работы всего алгоритма, описанного выше (BiCGStab с ILU(0)-предобусловливателем);

3) вычисления производились в типе double, а время копирования матриц на видеокарту не учитывалось.

Результаты сравнения приведены на рис. 2.

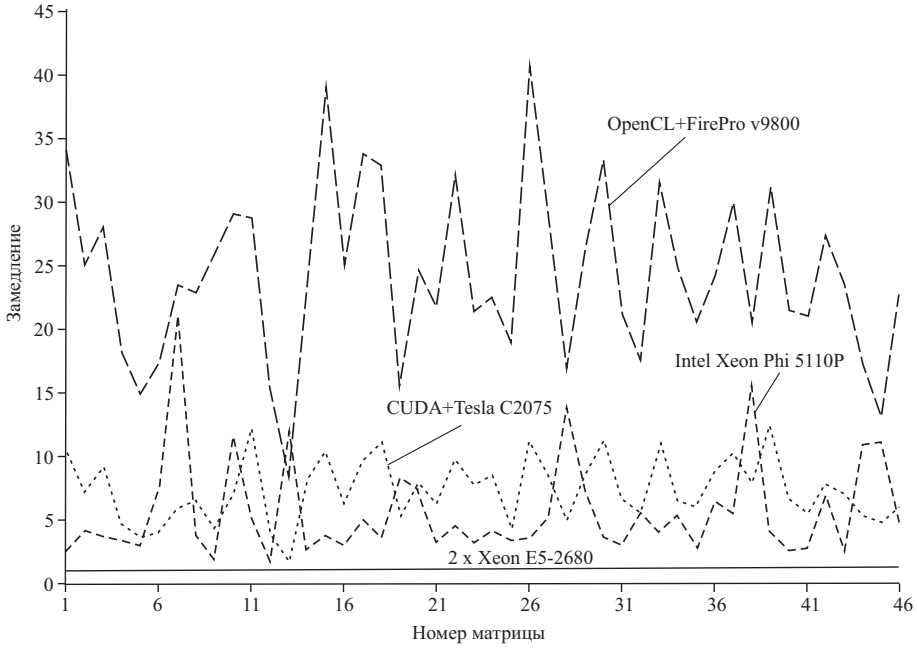

Рис. 2. Замедление относительно 2 x Xeon E5-2680

Кроме того, благодаря тому, что код на OpenCL можно без изменений запускать на видеокартах Nvidia, мы провели сравнение производительности видеокарт на нашем алгоритме (рис. 3).

Из этого рисунка видно, что для нашей задачи лучше подходит видеокарта Tesla C2075 — даже один и тот же код на OpenCL работает на ней в два раза быстрее.

На рис. 3 нет сравнения производительности с библиотекой CUSP по следующим причинам.

Таблица 2

№ матрицы	Ускорение ILU(0) относительно BAINV
26	1.77
28	3.97
45	1.45

1. Вследствие плохой обусловленности рассматриваемых матриц метод BiCGStab с предобусловливателем nonsym_bridson_ainv (использовались стандартные параметры) расходился на большинстве матриц.

2. Так как некоторые матрицы получаются огромными (до 3 Гб в mtx-формате), то памяти на Tesla C2075 не хватало для работы предобусловливателей из библиотеки CUSP.

Сказанное характеризует сложность вычислений, проводимых при моделировании месторождений. В табл. 2 приведены результаты на некоторых матрицах, для которых алгоритм из CUSP сошелся.

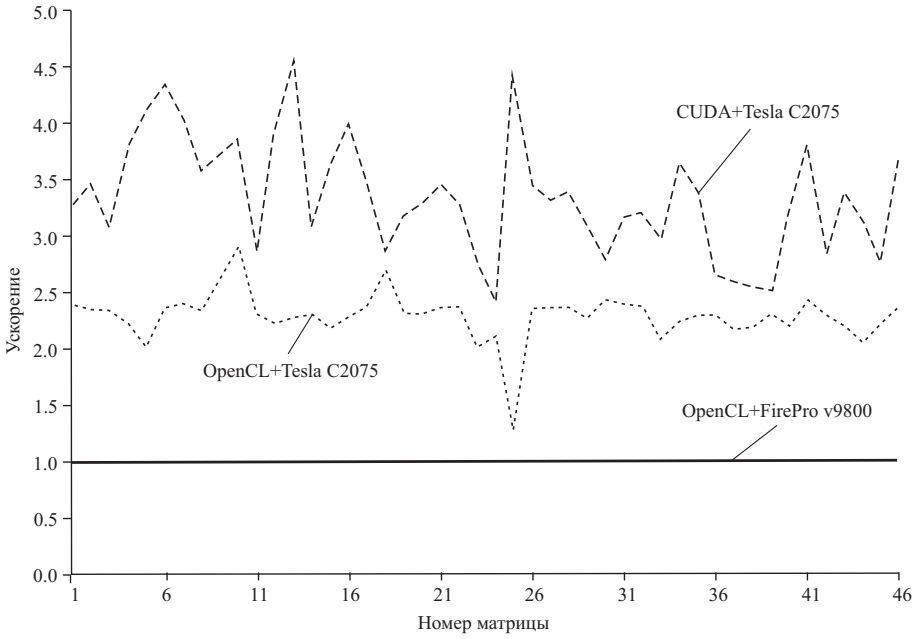

Рис. 3. Ускорение относительно OpenCL+FirePro v9800

7. Заключение. В настоящей статье рассмотрен алгоритм BiCGStab с ILU(0)-предобусловливателем, реализованный на различных многоядерных системах (Nvidia Tesla C2075, ATI FirePro v9800, 2×Intel Xeon E5-2680 и Intel Xeon Phi 5110P), имеющих на данный момент примерно одинаковую стоимость. Эффективность алгоритма проверялась на несимметричных, разреженных и плохо обусловленных матрицах, возникающих при моделировании реальных месторождений с большим числом скважин. Время пересылки матриц с хост-системы на GPU не учитывалось.

Несмотря на весь потенциал GPGPU-вычислений, на данном алгоритме обе видеокарты уступают в производительности последнему поколению процессоров Intel. При этом для того, чтобы устройства GPU показывали свою пиковую производительность в задаче фильтрации вязкой сжимаемой многофазной многокомпонентной смеси, необходимо менять не только алгоритм построения предобусловливателя, но и сам итерационный алгоритм. Еще одним препятствием к переходу на GPGPU-вычисления является объем памяти видеокарт, которого пока не хватает для моделирования крупных месторождений.

СПИСОК ЛИТЕРАТУРЫ

1. *Aziz K., Settari A.* Petroleum reservoir simulation. London: Applied Science Publishers, 1979.
2. *Chen Z., Huan G., Ma Y.* Computational methods for multiphase flows in porous media. Philadelphia: SIAM, 2006.
3. *Saad Y.* Iterative methods for sparse linear systems. Philadelphia: SIAM, 2003.
4. *Капорин И.Е., Коньшин И.Н.* Параллельное решение симметричных положительно-определенных систем на основе перекрывающегося разбиения на блоки // Журн. вычисл. математ. и матем. физ. 2001. **41**, № 4. 515–528.
5. *Богачев К.Ю., Жабицкий Я.В.* Блочные предобусловливатели класса ILU для задач фильтрации многокомпонентной смеси в пористой среде // Вестн. Моск. ун-та. Сер. 1. Математика. Механика. 2009. № 5. 19–25.
6. *Богачев К.Ю., Жабицкий Я.В.* Метод Капорина–Коньшина параллельной реализации блочных предобусловливатей для несимметричных матриц в задачах фильтрации многокомпонентной смеси в пористой среде // Вестн. Моск. ун-та. Сер. 1. Математика. Механика. 2010. № 1. 46–52.
7. *Li R., Saad Y.* GPU-accelerated preconditioned iterative linear solvers. Minneapolis: Minnesota Supercomputer Institute (University of Minnesota), 2010.
8. *Scarpino M.* OpenCL in action. Westampton: Manning Publications, 2011.
9. Nvidia CUDA C Best Practices Guide (http://developer.nvidia.com/cuda/nvidia-gpu-computing-documentation).
10. *Богачев К.Ю., Жабицкий Я.В., Климовский А.А., Миргасимов А.Р., Семенко А.Е.* Сравнение итерационных методов решения разреженных систем линейных уравнений в задачах фильтрации на вычислительных системах с распределенной памятью // Вычислительные методы и программирование. 2011. **12**. 74–76.

Поступила в редакцию
28.05.2013