ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

В.Л. Якушев, А.В. Филимонов, П.Ю. Солдатов

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ CUDA ДЛЯ ОПТИМИЗАЦИИ ПРЯМОГО РЕШАТЕЛЯ СЛАУ

В данной работе предложен способ ускорения прямого решателя СЛАУ с применением графических процессоров (GPU) от nVidia. Описан опыт пошагового повышения быстродействия. Рассмотрены основные проблемы, возникающие при использовании графического ускорителя, и пути их решения. Представлен способ автоматической балансировки вычислений для различных конфигураций оборудования. Приведены результаты тестирования для конечно-элементных моделей реальных строительных объектов.

GPU, CUDA, гибридные системы, умножение матриц, системы автоматизированного проектирования.

Введение

В данный момент развитие технологий вычислений общего назначения на графических процессорах (GPU) идет быстрыми темпами, причем как на аппаратном, так и на программном уровне. Благодаря большому количеству ядер и другим особенностям архитектуры использование GPU позволяет значительно увеличить производительность вычислений для некоторых задач [8, 9]. Однако для большинства задач, оптимизированных для применения на центральных процессорах (CPU), получение эффективного решения с использованием GPU является трудоемкой задачей. В данной работе рассмотрен подход, который дает возможность упросить внедрение технологии вычисления на GPU и уменьшить время выполнения выполняемых задач для гибридных систем. Суть подхода заключается в перенаправлении наиболее трудоемких функций на GPU и оптимизации взаимодействия между вычислительными устройствами. Особенности применения данного подхода были рассмотрены при оптимизации прямого решателя разреженных систем линейных алгебраических уравнений большой размерности с использованием программного обеспечения CUDA Toolkit для графических процессоров производства nVidia.

Решатель реализует разложение Холецкого, распараллелен для машин с общей памятью и поддерживает двойную точность вычислений. Решатель может использоваться в расчетных комплексах для прочностного анализа строительных сооружений [1, 2]. Необходимо отметить, что размер матрицы, задающей СЛАУ, настолько велик, что запись ее в память GPU не представляется возможной.

Решатель использует интерфейс BLAS, который представляет собой набор функций для выполнения операций линейной алгебры. При достаточно большой размерности входных массивов эти функции могут выполняться в несколько раз быстрее при использовании библиотеки CUBLAS, имеющей подобный интерфейс и входящей в комплект CUDA Toolkit [4, 7]. Однако при небольшой размерности копирование данных на видеокарту и обратно занимает слишком много времени относительно собственно выполнения операций, и существенную прибавку в скорости (в том числе на не самых мощных GPU [10]) дает только операция умножения матриц — GEMM (рис. 1).

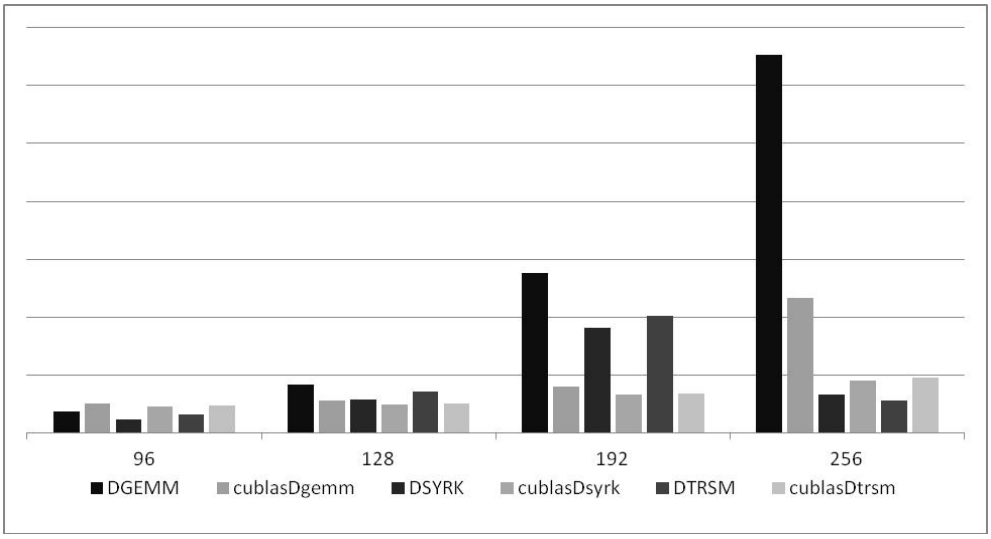

Рис. 1. Сравнение времени выполнения некоторых операций в зависимости от размера матриц с использованием библиотек BLAS и CUBLAS с двойной точностью вычислений.

Операция умножения матриц выполняется 80–90 % времени работы решателя, причем из-за особенностей алгоритма размер стороны матриц не превышает 300. Таким образом, существенно уменьшив время умножения матриц, можно существенно уменьшить и время работы решателя. Рассматриваемая операция имеет следующий вид:

$$C = \alpha AB + \beta C.$$

Внедрение CUBLAS

По тестовым примерам был определен порог значения стороны матрицы, превышение которого делает выгодным вычисления на GPU. Однако простая замена вызовов матриц без учета особенностей вычислений на GPU приведет к замедлению работы решателя, поэтому необходимо учитывать некоторые особенности работы именно этого типа вычислительных устройств. В первую очередь, необходима оптимизация однотипных операций: выделения и освобождения памяти под умножаемые матрицы в рамках одного контекста на GPU, а также работа с множителем β на центральном процессоре, поскольку выполнять операции умножения на скаляр и сложения матриц небольшой размерности на GPU неэффективно. Самую серьезную проблему создают очереди заданий: стандартными средствами на GPU нельзя запустить новое задание, пока не закончит выполняться текущее. Таким образом, время ожидания превышает время, сэкономленное на вычислениях. Если не формировать заданий для занятой видеокарты, а исполнять их сразу на CPU, то удается уменьшить время работы решателя примерно на 20 % в многопоточном режиме [3].

Однако остаются другие проблемы. Так, видеокарта оказывается недостаточно загруженной: время простоя слишком велико по отношению ко времени вычислений, и значительную часть работы GPU составляет копирование. Кроме того, порог-условие отправки умножения матриц на GPU слишком высок, и ресурсоемкие операции выполняются на более медленном в таких условиях устройстве. Также этот порог может сильно различаться в зависимости

от задачи и конфигурации оборудования. В этой связи необходимы шаги, направленные на улучшение использования видеокарты и взаимодействия между CPU и GPU.

Оптимизация умножений матриц

Существуют способы распараллеливания действий на самой видеокарте [5, 6]. Так как решатель использует OpenMP, логично применять для каждого omp-потока свой контекст на GPU с помощью CUDA streams. Отметим, что размер матриц и количество потоков позволяют не задумываться об экономии и динамическом выделении памяти. Однако, так как любое обращение к GPU обходится дорого, целесообразно записывать все входящие матрицы в одну; это позволит сэкономить еще немного времени на вызовах копирования.

Технология CUDA поддерживает асинхронное копирование, поэтому некоторые операции на видеокарте, в том числе из разных CUDA streams, могут выполняться одновременно. Для ускорения собственно копирования была применена pinned-память — фиксированный участок оперативной памяти, который может быть очень быстро помещен в память GPU. То есть необходимые данные сначала копируются в pinned-массив и только затем — на GPU. Несмотря на увеличение числа команд, использование pinned-памяти существенно уменьшило время копирования. Время выполнения умножений разными способами отражено на рис. 2.

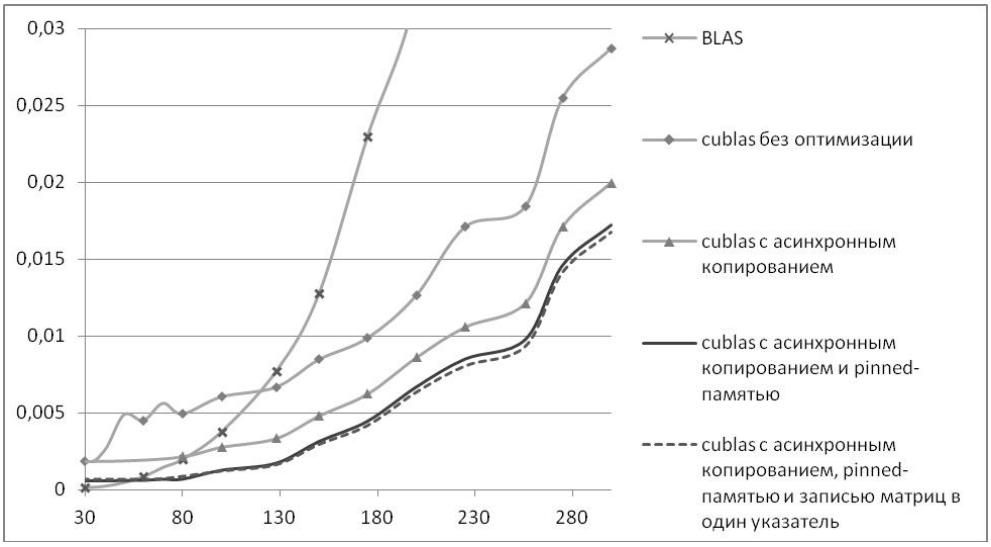

Рис. 2. Сравнение различных способов умножений

Таким образом, с помощью вышеописанных усовершенствований удалось понизить порог отправки задания на GPU и увеличить скорость копирования.

Результаты тестирования

Для тестирования были подобраны модели строительных объектов из практики ЦНИИСК им. Кучеренко. Тестирование проводилось на оборудовании низкого ценового диапазона. Время работы решателя уменьшилось на большинстве задач примерно в 1,5 раза. Результаты для каждой задачи приведены в табл. 1.

Таблица 1

Результаты тестирования. Конфигурация 1: Intel Core i7 3400 MHz (4 ядра, 8 потоков), nVidia GeForce GTX550Ti. Конфигурация 2: Intel Pentium III Xeon, 2666 MHz (2 ядра, 4 потока), nVidia GeForce GTS450

Номер задачи	Конфигурация 1			Конфигурация 2		
	CPU	CPU+GPU	Соотн.	CPU	CPU+GPU	Соотн.
1	64	36	1,78	143	80	1,78
2	108	64	1,69	244	147	1,69
3	267	128	2,09	389	247	2,09
4	127	76	1,67	282	167	1,67
5	53	39	1,36	118	91	1,30
6	44	30	1,47	99	67	1,48
7	29	19	1,53	66	40	1,65
8	127	90	1,41	286	172	1,66

Также было проведено тестирование на машине с графическим ускорителем высокого класса. Несмотря на то, что удалось добиться ускорения примерно в 2,5 раза, результат нельзя признать удовлетворительным, поскольку на данном оборудовании алгоритм будет выполняться всего в два-три раза быстрее, чем на обычном персональном компьютере, однако их стоимость отличается в десятки раз. Можно сделать вывод о том, что загрузить GPU топкласса методом, описанным в работе, не получается, и для эффективной работы с Tesla необходим другой подход, в том числе использование ядра для умножения матриц отличного от имеющегося в библиотеке CUBLAS. Результаты тестирования на Tesla приведены в табл. 2.

Таблица 2

Результаты тестирования. Конфигурации 1 и 2 аналогичны табл. 1. Конфигурация 3: Intel Xeon X5680 3300 MHz (6 ядер, 12 потоков), nVidia Tesla M2090

Номер задачи	Конфигурация 3			Сравнение с конфиг. 1	Сравнение с конфиг. 2
	CPU	CPU+GPU	Соотн.		
1	59	25	2,36	1,44	3,20
2	99	37	2,68	1,73	3,97
3	141	53	2,66	2,42	4,66
4	115	44	2,61	1,73	3,80
5	43	28	1,54	1,39	3,25
6	40	22	1,82	1,36	3,05
7	28	16	1,75	1,19	2,50
8	120	48	2,5	1,88	3,58

Заключение

Представленный подход позволяет уменьшить время работы решателя примерно в полтора раза. Удалось избежать модификации исходного кода решателя, т.к. основные изменения были реализованы на уровне библиотеки BLAS, которая анализировала и перенаправляла часть вычислений на графический процессор. В дальнейшем будет предпринят поиск более эффективного ядра для перемножения матриц на GPU, в частности будут опробованы batched-функции. Следует отметить, что с выходом обновлений программного обеспечения от nVidia могут появиться дополнительные возможности для оптимизации приложений. Помимо поиска более эффективных решений для GPU производства nVidia, планируется адаптация метода для графических ускорителей других производителей.

ЛИТЕРАТУРА

1. *Якушев В.Л., Назаров Ю.П., Жук Ю.Н., Симбиркин В.Н., Филимонов А.В.* Решение большеразмерных задач статики и динамики конструкций методом конечных элементов. Супервычисления и математическое моделирование. Труды XII междунар. семинара / Под ред. Р.М. Шагалиева. Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2011. С. 407–413.
2. *Якушев В.Л., Симбиркин В.Н., Филимонов А.В.* Сейсмический режим поиска собственных форм колебаний в программном комплексе STARK ES // Вестн. кибернетики. Тюмень: Изд-во ИПОС СО РАН, 2012. № 11. С. 151–157.
3. *Якушев В.Л., Филимонов А.В., Солдатов П.Ю.* Методика повышения эффективности решателей СЛАУ с использованием графических ускорителей // Вестн. кибернетики. Тюмень: Изд-во ИПОС СО РАН, 2013. № 12. С. 169–173.
4. *CUBLAS* Library User Guide // NVIDIA Corporation [Электрон. ресурс]. Режим доступа: http://developer.nvidia.com/.
5. *CUDA* C Best Practices Guide // NVIDIA Corporation [Электрон. ресурс]. Режим доступа: http://docs.nvidia.com/.
6. *CUDA* C Programming Guide // NVIDIA Corporation [Электрон. ресурс]. Режим доступа: http://docs.nvidia.com/.
7. *CUDA* Toolkit 5.0 Performance Report [Электрон. ресурс]. Режим доступа: https://developer.nvidia.com/content/cuda-5-performance-report-now-available/.
8. *Cullinan C., Wyant C., Frattesi T.* Computing Performance Benchmarks among CPU, GPU, and FPGA [Электрон. ресурс]. Режим доступа: http://www.wpi.edu/.
9. *General-Purpose* Computation on Graphics Hardware [Электрон. ресурс]. Режим доступа: http://www.gpgpu.org/.
10. *Tan G., Li L., Triechle S., Phillips E., Bao Y., Sun N.* Fast implementation of DGEMM on Fermi GPU // Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, New York, NY, USA , P. 35:1–35:11.

V.L. Yakushev, A.V. Filimonov, P.Yu. Soldatov

Direct SLAE solver optimization by CUDA

The authors suggest technique of accelerating direct SLAE solver by implementing computations on nVidia graphics processing units. Features of GPU use are enumerated, bottle necks are identified, different options of performing tasks on GPU were tested to achieve better performance for hybrid CPU-GPU systems. The solver was tested on both middle and top level GPUs, the results are provided.

GPU, CUDA, hybrid CPU-GPU, general matrix multiplication, computer aided design.