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

МЕТОДИКА ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ РЕШАТЕЛЕЙ СЛАУ С ИСПОЛЬЗОВАНИЕМ ГРАФИЧЕСКИХ УСКОРИТЕЛЕЙ

Предложен простой и эффективный способ адаптации прямого решателя СЛАУ для гибридных систем. Описан опыт пошагового повышения его быстродействия. Рассмотрены основные проблемы, возникающие при использовании графического процессора (GPU), и пути их решения. Реализован способ автоматической балансировки вычислений для различных конфигураций оборудования. Приведены результаты тестирования для конечно-элементных моделей реальных строительных объектов.

Графические процессоры (GPU), параллельные вычисления, разложение Холецкого.

Введение

Решение разреженных систем линейных алгебраических уравнений большой размерности является важной вычислительной задачей при разработке наукоемких приложений. Благодаря параллельным алгоритмам и развитию архитектур центральных процессоров можно существенно оптимизировать работу решателей. Однако в настоящее время происходит развитие вычислений на графических процессорах, которые при определенных условиях имеют показатели производительности значительно выше, чем у центральных процессоров. В данной работе изучается возможность ускорения работы существующего решателя путем перенаправления части вычислений на графические вычислительные устройства.

Рассмотрен прямой разреженный решатель СЛАУ, который реализует разложение Холецкого — симметричная положительно определенная матрица A представляется в виде $A = LL^T$, где L — нижняя треугольная. При использовании прямого метода время решения практически не зависит от количества правых частей. Данная особенность является очень важной, так как в строительной механике часто возникают задачи с большим количеством правых частей [2–4]. Получению эффективного решения итерационными методами с большим количеством правых частей посвящена статья [5]. Разработанный решатель распараллелен для машин с общей памятью с помощью директив OpenMP.

Различия в идеологии и синтаксисе распараллеливания для CPU и GPU предполагают существенное изменение внутренней структуры программы для получения максимального быстродействия [1]. При создании промышленного приложения возникают дополнительные проблемы, связанные с возможностью запуска на различных конфигурациях оборудования. Следует отметить, что технологии вычислений на GPU все еще находятся в периоде становления и текущие оптимальные решения могут быстро устареть.

Возможности применения вычислений на GPU

Построение оптимального для графических вычислительных устройств параллельного алгоритма для реализации разложения Холецкого с нуля или внедрение директив OpenACC потребовало бы много ресурсов [10]. Поэтому было решено сосредоточиться на перенаправлении части вычислений с большими объемами данных на GPU [6]. При постановке работы было сформулировано дополнительное условие внесения минимальных изменений в код решателя. И решение было найдено, так как решатель использует интерфейс

BLAS (Basic Linear Algebra Subprograms). Данный интерфейс является дефакто стандартом интерфейса программирования приложений для создания библиотек, выполняющих основные операции линейной алгебры, такие как умножение векторов и матриц [11]. Таким образом, можно динамически подключить две библиотеки: BLAS, оптимизированную для CPU, и CUBLAS, оптимизированную для GPU. CUBLAS входит в комплект средств разработки CUDA Toolkit и имеет стандартный интерфейс BLAS. Для внедрения CUBLAS в работу решателя потребовалось подготовить небольшую библиотеку, которая устраняет отличия в обращении к BLAS и CUBLAS (например, в передаваемых типах данных) и корректно осуществляет передачу данных между вычислительными мощностями [8, 9]. Достоинством такого подхода является быстрая адаптация имеющихся решателей на CPU для работы с GPU и возможность настройки решателя индивидуально для каждого компьютера.

Однако при реализации данного подхода возникло несколько проблем. Например, перенаправление всех вычислительных операций из библиотеки BLAS не ускорит, а замедлит работу решателя, поскольку много времени будет тратиться на переписывание данных. Поэтому необходимо определить операции, где вычисления на GPU более производительны, чем на CPU [7, 8, 11]. Подготовка данных для GPU происходит во многих параллельных потоках, и при их передаче образуется очередь, которая практически сводит на нет распараллеливание OpenMP. Скорость копирования данных на GPU зависит от различных способов копирования и конфигурации оборудования. Оптимальный баланс, полученный на определенном примере и конфигурации оборудования, может быть нарушен при изменении конфигурации или задачи. Поэтому алгоритм должен быть способен автоматически определить оптимальное распределение заданий для различных сочетаний процессоров.

Внедрение CUBLAS

Анализ работы решателя показал, что операция перемножения матриц (DGEMM) занимает около 80–85 % времени факторизации (рис. 1). Разработаны тестовые программы сравнения скорости умножения матриц. По результатам их выполнения были определены размеры массивов, превышение которых делало выгодным использование GPU (рис. 2). Количество умножений матриц, размеры которых больше пороговых значений, оказалось достаточным, чтобы ожидать сокращения общего времени умножения матриц в несколько раз.

Рис. 1. Схема работы процедур решателя

В многопоточном режиме образование очередей заданий для GPU снижает быстродействие работы решателя, поскольку отправить новое задание на GPU можно только после завершения исполнения текущего. Быстрое умножение не покрывает времени ожидания и копирования, и улучшить время

работы решателя не удается. В случае, если алгоритм распределения не формирует заданий для GPU, если GPU уже занят, можно добиться незначительного ускорения. Эффект от работы GPU будет нивелирован относительно медленной работой ядер CPU с оставшимися в их распоряжении большими объемами информации, но общее время работы все же уменьшится.

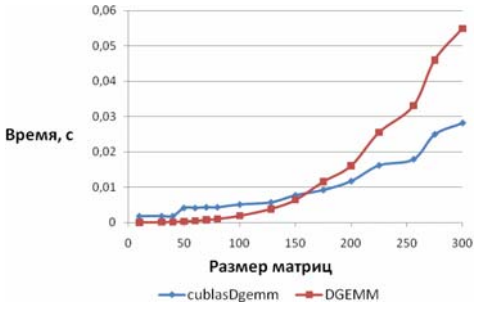

Рис. 2. Время выполнения умножения матриц в зависимости от размера (CPU Intel Core i7 3400 MHz (4 ядра, 8 потоков), GPU nVidia GeForce GTX 550Ti)

Отдельное внимание посвящено работе с выделением памяти. Было установлено, что выделять и освобождать участок памяти на GPU для каждого набора переменных менее продуктивно, чем при инициализации устройства выделить под указатели память, размер которой сопоставим с доступной глобальной памятью на конкретном GPU. В этом случае копирование матриц производится каждый раз в один и тот же участок памяти без его освобождения. Освобождение памяти производится по завершении работы устройства. Также установлено, что при необходимости выполнения операций с подматрицами передаваемых матриц выгоднее переписать матрицу полностью, а не только нужную для вычислений часть, поскольку при этом не тратится лишнее время на обращения к копированию.

Для настройки алгоритма, проверки правильности и оценки эффективности его работы был подобран ряд конечно-элементных моделей проектируемых строительных объектов из практики ЦНИИСК им. В.А. Кучеренко. Численные эксперименты были проведены для различных конфигураций оборудования. Результаты тестирования решения, а именно время расчета для каждой модели в различных режимах работы, приведены в табл. Все данные получены для двойной точности вычислений.

Результаты тестирования (CPU Intel Core i7 3400 MHz (4 ядра, 8 потоков), GPU nVidia GeForce GTX 550Ti)

Номер задачи	Кол-во степеней свободы	Время выполнения факторизации, с					
		Однопоточный режим			Восьмипоточный режим		
		CPU	CPU+GPU	Ускорение, %	CPU	CPU+GPU	Ускорение, %
1	921 600	266	119	44,7	78	48	38,5
2	4 870 800	206	99	52	51	43	15,7
3	2 534 446	183	90	50,8	44	35	20,5
4	397 950	548	304	44,5	128	97	24,2
5	889 890	124	79	36,3	29	23	20,7
6	2 428 323	441	268	39,2	107	82	23,6
7	2 545 314	5657	2466	56,4	1350	937	30,6

Как видно из табл., на данный момент удалось добиться почти двукратного ускорения факторизации в режиме работы одного ядра CPU и уменьшить

время работы примерно на 20 % при использовании OpenMP. Доля времени факторизации, затраченная на умножение матриц, сократилась примерно до 60 % (рис. 3), при этом большую часть времени вычисления проводятся на GPU (включая время копирования данных).

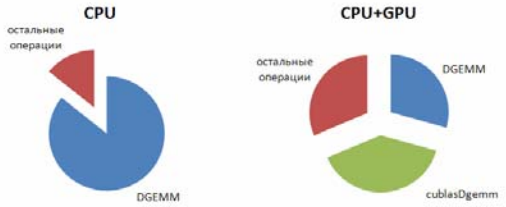

Рис. 3. Диаграмма распределения общего времени факторизации

Возникшие трудности

Для некоторых задач добиться ускорения работы решателя не удается в силу особенностей решения СЛАУ, например из-за низкого заполнения множителей разложения Холецкого. Однако, прежде чем приступить к универсализации алгоритма, следует оптимизировать работу GPU. Запуск профилировщика GPU для тестового примера умножения матриц, аналог которого встроен в решатель, показывает, что загрузить видеокарту должным образом не получается, так как слишком мало ядер участвует в вычислениях, и соотношение времени вычислений к общему времени работы GPU мало (рис. 4). Несмотря на экономию времени за счет выделения и освобождения памяти, умножение матриц в работе решателя также может выполняться гораздо быстрее. Поэтому можно сделать вывод о неэффективности библиотеки, формирующей и передающей задания для GPU, и необходимости ее усовершенствования.

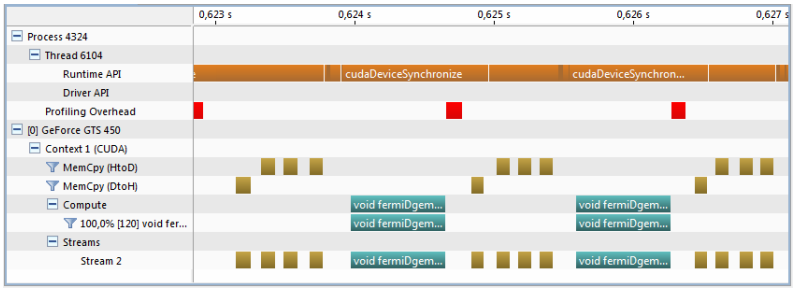

Рис. 4. Данные профилировки работы GPU на тестовом примере

Среди методов устранения обозначенных проблем рассматриваются способы распараллеливания внутри видеокарты, в частности cuda streams. С их помощью можно реализовать асинхронное копирование. Однако в данном случае результат будет сильно зависеть от архитектуры GPU и от версии установленного ПО. Также возможно написание собственного ядра для перемножения небольших матриц. Другое направление для изучения — использование специальных видов памяти, позволяющих быстро осуществлять передачу больших объемов информации в оба направления, например pinned-память. Все подходы основаны в первую очередь на оптимизации взаимодействия между CPU и GPU — ключевом факторе в работе гибридной системы. Другими словами, необходимо стремиться к максимальному использованию пропускной способности канала передачи данных.

Заключение

Предполагается, что данный решатель будет использоваться в программных комплексах, при этом значительных затрат на новую технику со стороны проектных компаний не потребуется. Поэтому тестирование данного решателя производилось на оборудовании низкого и среднего ценового диапазона. Получено уменьшение времени работы решателя на 20–30 % при использовании графического вычислительного устройства без изменений существующего кода. Данная методика повышения эффективности решения может дать лучший результат при использовании асинхронных операций и оптимизации копирования данных на графическом ускорителе.

ЛИТЕРАТУРА

1. *Сандерс Дж., Кэндрот Э.* Технология CUDA в примерах: Введение в программирование графических процессоров / Пер. с англ. А.А. Слинкина; Науч. ред. А.В. Боресков. М.: ДМК Пресс, 2011.

2. *Якушев В.Л., Жук Ю.Н., Симбиркин В.Н., Филимонов А.В.* Реализация методов расчета для большеразмерных задач строительной механики в программном комплексе STARK ES // Вестн. кибернетики. Тюмень: Изд-во ИПОС СО РАН, 2011. № 10. С. 109–116.

3. *Якушев В.Л., Симбиркин В.Н., Филимонов А.В.* Решение большеразмерных задач строительной механики методом конечных элементов в программном комплексе STARK ES // Теория и практика расчета зданий, сооружений и элементов конструкций. Аналитические и численные методы: Сб. тр. междунар. науч.-практ. конф. М., 2010. С. 516–526.

4. *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.

5. *Якушев В.Л., Симбиркин В.Н., Филимонов А.В. и др.* Решение плохообусловленных симметричных СЛАУ для задач строительной механики параллельными итерационными методами // Междунар. суперкомпьютер. конф.: Научный сервис в сети Интернет: Экзафлопсное будущее: Сб. тр. междунар. науч.-практ. конф. 2011. С. 333–342.

6. *Якушев В.Л., Филимонов А.В., Новиков П.А., Солдатов П.Ю.* Создание эффективных решателей для GPU // Науч.-практ. конф. с междунар. участием с элементами науч. шк. для молодежи: Высокопроизводительные вычисления на графических процессорах: Тез. докл. Пермь, 2012. С. 85–87.

7. *Bell N., Garland M.* Efficient sparse matrix-vector multiplication on CUDA: NVIDIA Technical Report NVR-2008-004 // NVIDIA Corporation. 2008.

8. *CUBLAS* Library User Guide // NVIDIA Corporation [Электрон. ресурс]. Режим доступа: http://developer.nvidia.com.

9. *GPUmat* User Guide [Электрон. ресурс]. Режим доступа: http://gp-you.org.

10. *Hogg J. D., Reid J. K., Scott J. A.* Design of a Multicore Sparse Cholesky Factorization Using DAGs: STFC Technical Report RAL-TR-2009-027 // Science and Technology Facilities Council. 2009.

11. *Tan G., Li L., Triechle S. et al.* Fast implementation of DGEMM on Fermi GPU // Proceedings of 2011 Intern. Conf. for High Performance Computing, Networking, Storage and Analysis, ACM. New York, NY, USA, 2011. P. 35:1–35:11.

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

Methods to increase the efficiency of the SLAE solvers using graphics accelerators

The article suggests a simple and effective method of adapting the SLAE direct solver for hybrid systems, describing the experience of a step-by-step increase of its operation speed. It considers major problems arising when using the graphics processing unit (GPU), and ways of their solution. Subject to implementation being a method of automatic balancing of computations for different configurations of equipment. The paper cites test results for the finite element models of real construction objects.

Graphics processing units (GPU), parallel computations, Cholesky factorization.