Нашим семьям и друзьям, которые во всем поддерживали нас.
Нашим читателям, которые открывают перед нами будущее.
И учителям, которые научили наших читателей читать.

Содержание

Предисловие

Недавние разработки ведущих производителей микросхем, таких как NVIDIA, со всей очевидностью показали, что будущие микропроцессоры и крупные высокопроизводительные вычислительные системы (HPC) будут гибридными (гетерогенными). В их основу будут положены компоненты двух основных типов в разных пропорциях:

- **мультиядерные и многоядерные центральные процессоры**: количество ядер будет и дальше возрастать из-за желания поместить все больше компонентов на один кристалл, не упираясь в барьер мощности, памяти и параллелизма на уровне команд;
- **специализированное оборудование и массивно-параллельные ускорители**: например, графические процессоры (GPU) от NVIDIA в последние годы превзошли стандартные CPU в производительности вычислений с плавающей точкой. Да и программировать их стало так же просто, как многоядерные GPU (если не проще).

Каким будет соотношение между этими компонентами в будущих проектах, пока не ясно, и со временем оно, скорее всего, будет меняться. Но не вызывает сомнений, что вычислительные системы нового поколения – от ноутбуков до суперкомпьютеров – будут состоять из гетерогенных компонентов. Именно такая система преодолела барьер в один петафлоп (10^{15} операций с плавающей точкой в секунду).

И тем не менее задачи, с которыми сталкиваются разработчики, вступающие в новый мир гибридных процессоров, все еще весьма сложны. Критическим частям программной инфраструктуры уже очень трудно поспевать за изменениями. В некоторых случаях производительность не масштабируется с увеличением числа ядер, потому что все большая доля времени тратится не на вычисления, а на перемещение данных. В других случаях разработка программ, оптимизированных для достижения максимального быстродействия, завершается спустя годы после выхода на рынок оборудования, и потому они оказываются устаревшими уже в момент поставки. А иногда, как в случае последних моделей GPU, программа вообще перестает работать, так как окружение изменилось слишком сильно.

Книга «Технология CUDA в примерах» посвящена рассмотрению одного из самых новаторских и эффективных решений задачи программирования массивно-параллельных процессоров, появившемуся в последние годы.

В ней вы на примерах познакомитесь с программированием на языке CUDA C, узнаете о конструкции графических процессоров NVIDIA и научитесь их эффективно использовать. Эта книга – введение в идеи параллельных вычислений, здесь

вы найдете и простые примеры, и технику отладки (поиск логических ошибок и разрешение проблем с производительностью), и более сложные темы, связанные с разработкой и использованием конкретных приложений. Все излагаемые идеи иллюстрируются примерами кода.

Эта книга – обязательное чтение для всех работающих с вычислительными системами, содержащими ускорители. Параллельные вычисления здесь рассматриваются достаточно глубоко, предлагаются подходы к решению различных возникающих задач. Особенно полезна она будет разработчикам приложений, авторам библиотек для численных расчетов, а также студентам, изучающим параллельные вычисления, и преподавателям.

Я получил большое удовольствие от чтения этой книги и почерпнул из нее кое-что новое для себя. Уверен, что и вам это предстоит.

Джек Донгарра,
заслуженный профессор,
заслуженный исследователь
университета штата Теннесси,
национальная лаборатория Оук Ридж

Вступление

В этой книге демонстрируется, как, задействовав мощь графического процессора (GPU), имеющегося в вашем компьютере, можно создавать высокопроизводительные приложения для самых разных целей. Хотя первоначально GPU предназначались для визуализации компьютерной графики на экране монитора (и по сей день используются для этой цели), позже они оказались востребованы и в других, столь же требовательных к вычислительным ресурсам областях науки, техники, финансовой математики и др. Программы для GPU, которые решают задачи, не связанные с графикой, мы будем называть программами *общего назначения* (general purpose). Для чтения этой книги необходим опыт работы с языками C или C++, но разбираться в компьютерной графике, к счастью, необязательно. Вообще не нужно! Программирование GPU просто открывает возможность по-новому – и с большой пользой – применить уже имеющиеся у вас навыки.

Чтобы использовать NVIDIA GPU для решения задач общего назначения, необходимо знать, что такое технология CUDA. Все NVIDIA GPU построены на базе *архитектуры CUDA*. Можно считать, что это та основа, на которой компания NVIDIA сконструировала GPU, умеющие решать *как* традиционные задачи рендеринга, *так и* задачи общего назначения. Для программирования GPU на основе CUDA мы будем использовать язык *CUDA C*. Как вы скоро увидите, CUDA C – это по существу язык C[1], в который введены расширения, позволяющие программировать массивно-параллельные компьютеры, каковыми являются NVIDIA GPU.

При написании книги мы ориентировались на опытных программистов на C или C++, не испытывающих затруднений при чтении и написании кода на языке C. Текст, изобилующий примерами, позволит вам быстро освоить разработанный компанией NVIDIA язык программирования CUDA C. Разумеется, мы не предполагаем, что вы уже разрабатывали крупномасштабные программные системы, писали компилятор языка C или ядро операционной системы либо досконально знаете все закоулки стандарта ANSI C. Но в то же время мы не тратим время на обзор синтаксиса языка C или таких хорошо известных библиотечных функций, как `malloc()` или `memcpy()`, поскольку считаем, что с этими вопросами читатель уже знаком.

Вы встретите ряд приемов, которые можно было бы назвать общими парадигмами параллельного программирования, но мы не ставили себе целью написать учебник по основам параллельного программирования. Кроме того, хотя мы так

---

[1] Точнее, C++. – *Прим. перев.*

или иначе демонстрируем почти все части CUDA API, эта книга не может служить заменой полному справочнику по CUDA API, и мы не углубляемся во все детали всех инструментальных средств, помогающих разрабатывать программы на CUDA C. Поэтому мы настоятельно рекомендуем читать эту книгу вместе со свободно доступной документацией, в частности *Руководством по программированию в среде NVIDIA CUDA* (NVIDIA CUDA Programming Guide) и *Наставлением по рекомендуемым приемам разработки в среде NVIDIA CUDA* (NVIDIA CUDA Best Practices Guide). Но не стоит сразу бросаться скачивать эти документы; по ходу изложения мы расскажем, что и когда нужно делать.

Ну и хватит слов, мир программирования NVIDIA GPU на CUDA C ждет вас!

Благодарности

Говорят, что, для того чтобы выпустить в свет техническую книгу, нужно семь нянек. И книга «Технология CUDA в примерах» – не исключение. Авторы обязаны многим людям, и некоторых из них хотели бы поблагодарить на этих страницах.

Ян Бак (Ian Buck), отвечающий в NVIDIA за направление разработки ПО для программирования GPU, оказал неоценимую помощь на всех этапах работы над этой книгой – от поддержки самой идеи до консультации по различным деталям. Мы также обязаны Тиму Маррею (Tim Murray), нашему неизменно улыбчивому рецензенту; то, что эта книга технически точна и читабельна, – в основном его заслуга. Большое спасибо нашему художнику Дарвину Тату (Darwin Tat), который создал потрясающую обложку и рисунки в очень сжатые сроки. Наконец, мы очень признательны Джону Парку (John Park), который помог утрясти все юридические тонкости, связанные с изданием книги.

Без помощи со стороны издательства Addison-Wesley эта книга так и осталась бы в мечтах авторов. Питер Гордон (Peter Gordon), Ким Бёдигхаймер (Kim Boedigheimer) и Джулия Нахил (Julie Nahil) выказали безмерное терпение и профессионализм, благодаря им публикация этой книги прошла без сучка и без задоринки. А стараниями выпускающего редактора Молли Шарп (Molly Sharp) и корректора Ким Уимпсетт (Kim Wimpsett) куча изобилующих ошибками документов превратилась в аккуратный томик, который вы сейчас читаете.

Часть приведенного к этой книге материала не удалось бы включить без содействия со стороны помощников. Точнее, Надим Мохаммад (Nadeem Mohammad) поведал о примерах применения CUDA, упомянутых в главе 1, а Натан Уайтхэд (Nathan Whitehead) щедро поделился кодом, включенным в различные примеры.

Невозможно не поблагодарить и тех, кто прочел первые черновики и прислал полезные отзывы, в том числе Женевьеву Брид (Genevieve Breed) и Курта Уолла (Kurt Wall). Многие работающие в NVIDIA программисты оказали бесценную техническую помощь, в частности Марк Хэйргроув (Mark Hairgrove) скрупулезно проштудировал книгу в поисках всех и всяческих ошибок – технических, типографских и грамматических. Стив Хайнс (Steve Hines), Николас Вилт (Nicholas Wilt) и Стивен Джоунс (Stephen Jones) консультировали нас по отдельным разделам CUDA, помогая пролить свет на нюансы, которые иначе авторы могли бы оставить без внимания. Благодарим также Рандима Фернандо (Randima Fernando), который помог запустить этот проект, и Майкла Шидловски (Michael Schidlowsky) за упоминание Джейсона в своей книге.

И конечно, раздел «Благодарности» немыслим без выражения горячей признательности семьям, которые поддерживали нас во время работы и без которых эта книга никогда не состоялась бы. А особенно мы хотели бы поблагодарить наших любящих родителей, Эдварда и Кэтлин Кэндрот и Стивена и Хелен Сандерс. Спасибо также нашим братьям Кеннету Кэндроту и Кори Сандерсу. Спасибо за неустанную заботу и поддержку.

Об авторах

Джейсон Сандерс – старший инженер-программист в группе CUDA Platform в компании NVIDIA. Работая в NVIDIA, он принимал участие в разработке первых версий системы CUDA и в работе над спецификацией OpenCL 1.0, промышленном стандарте гетерогенных вычислений. Джейсон получил степень магистра по компьютерным наукам в Калифорнийском университете в Беркли, где опубликовал результаты исследований по вычислениям на графических процессорах. Он также получил степень бакалавра по электротехнике и электронике в Принстонском университете. До поступления на работу в NVIDIA он работал в ATI Technologies, Apple и Novell. Когда Джейсон не пишет книги, он любит играть в футбол и заниматься фотографированием.

Эдвард Кэндрот – старший инженер-программист в группе CUDA Algorithms в компании NVIDIA. Вот уже больше 20 лет он занимается оптимизацией кода и повышением производительности различных программ, в том числе Photoshop и Mozilla. Кэндрот работал в Adobe, Microsoft и Google, а также консультировал различные компании, включая Apple и Autodesk. В свободное от кодирования время он любит играть в World of Warcraft или отведывать восхитительные блюда в Лас-Вегасе.

Глава 1. Почему CUDA? Почему именно теперь?

Еще сравнительно недавно на параллельные вычисления смотрели как на «экзотику», находящуюся на периферии информатики. Но за последние несколько лет положение кардинально изменилось. Теперь практически каждый честолюбивый программист *вынужден* изучать параллельное программирование, если хочет добиться успеха в компьютерных дисциплинах. Быть может, вы взяли эту книгу, еще сомневаясь в важности той роли, которую параллельное программирование играет в компьютерном мире сегодня и будет играть в будущем. В этой вводной главе мы обсудим современные тенденции в области разработки оборудования, лежащие в основе ПО, которое предстоит создавать нам, программистам. И надеемся убедить вас в том, что «параллельно-вычислительная революция» *уже* произошла и что, изучая CUDA C, вы делаете правильный шаг, который позволит занять подобающее место среди тех, кто будет писать высокопроизводительные приложения для гетерогенных платформ, содержащих центральные и графические процессоры.

1.1. О чем эта глава

Прочитав эту главу, вы:

- узнаете о возрастающей роли параллельных вычислений;
- получите представление об истории вычислений с помощью GPU и технологии CUDA;
- познакомитесь с некоторыми успешными приложениями на базе CUDA.

1.2. Век параллельной обработки

За последние годы в компьютерной индустрии произошло немало событий, повлекших за собой масштабный сдвиг в сторону параллельных вычислений. В 2010 году почти все компьютеры потребительского класса будут оборудованы многоядерными центральными процессорами. С появлением дешевых двухъядерных нетбуков и рабочих станций с числом ядер от 8 до 16 параллельные вычисления перестают быть привилегией экзотических суперкомпьютеров и мейнфреймов. Более того, даже в мобильные телефоны и портативные аудиоплееры производители постепенно начинают встраивать средства параллельной обработки, стремясь наделить их такой функциональностью, которая была немыслима для устройств предыдущего поколения.

Разработчики ПО испытывают все более настоятельную потребность в освоении разнообразных платформ и технологий параллельных вычислений, чтобы предложить все более требовательным и знающим пользователям новаторские и функционально насыщенные решения. Время командной строки уходит, настает время многопоточных графических интерфейсов. Сотовые телефоны, умеющие только звонить, вымирают, им на смену идут телефоны, умеющие воспроизводить музыку, заходить в Интернет и поддерживать GPS-навигацию.

1.2.1. Центральные процессоры

В течение 30 лет одним из основных методов повышения производительности бытовых компьютеров было увеличение тактовой частоты процессора. В первых персональных компьютерах, появившихся в начале 1980-х годов, генератор тактовых импульсов внутри CPU работал на частоте 1 МГц или около того. Прошло 30 лет – и теперь тактовая частота процессоров в большинстве настольных компьютеров составляет от 1 до 4 ГГц, то есть примерно в 1000 быстрее своих прародителей. Хотя увеличение частоты тактового генератора – далеко не единственный способ повышения производительности вычислений, он всегда был наиболее надежным из всех.

Однако в последние годы производители оказались перед необходимостью искать замену этому традиционному источнику повышения быстродействия. Из-за фундаментальных ограничений при производстве интегральных схем уже невозможно рассчитывать на увеличение тактовой частоты процессора как средство получения дополнительной производительности от существующих архитектур. Ограничения на потребляемую мощность и на тепловыделение, а также быстро приближающийся физический предел размера транзистора заставляют исследователей и производителей искать решение в другом месте.

Тем временем суперкомпьютеры, оставаясь в стороне от мира потребительских компьютеров, на протяжении десятков лет добивались повышения производительности сходными методами. Производительность применяемых в них процессоров росла столь же быстрыми темпами, как в персональных компьютерах. Однако, помимо впечатляющего повышения быстродействия одного процессора, производители суперкомпьютеров нашли и другой источник роста общей производительности – увеличение *числа* процессоров. Самые быстрые современные суперкомпьютеры насчитывают десятки и сотни тысяч процессорных ядер, работающих согласованно. И, глядя на успехи суперкомпьютеров, естественно задаться вопросом: может быть, не гнаться за повышением производительности одного ядра, а поместить в персональный компьютер несколько таких ядер? При таком подходе мощность персональных компьютеров можно наращивать и дальше, не пытаясь любой ценой увеличить тактовую частоту.

В 2005 году, столкнувшись с ростом конкуренции на рынке и имея не так уж много вариантов выбора, ведущие производители CPU стали предлагать процессоры с двумя вычислительными ядрами вместо одного. В последующие

годы эта тенденция продолжилась выпуском CPU с тремя, четырьмя, шестью и восьмью ядрами. Эта так называемая *мультиядерная революция* знаменовала колоссальный скачок в развитии рынка потребительских компьютеров.

Сегодня весьма затруднительно купить настольный компьютер с одним-единственным процессорным ядром. Даже самые дешевые маломощные CPU содержат не менее двух ядер. Ведущие производители CPU уже анонсировали планы выпуска CPU с 12 и 16 ядрами, лишний раз подтвердив, что настало время параллельных вычислений.

1.3. Развитие GPU-вычислений

По сравнению с традиционным конвейером обработки данных в центральном процессоре, выполнение вычислений общего характера в графическом процессоре (GPU) – идея новая. Да и сама концепция GPU появилась сравнительно недавно. Однако мысль о том, чтобы производить вычисления в графическом процессоре, не так нова, как может показаться.

1.3.1. Краткая история GPU

Мы уже говорили об эволюции центральных процессоров в плане увеличения тактовой частоты и числа ядер. А тем временем в области обработки графики произошла настоящая революция. В конце 1980 – начале 1990-х годов рост популярности графических операционных систем типа Microsoft Windows создал рынок для процессоров нового типа. В начале 1990-х годов пользователи начали покупать ускорители двумерной графики для своих ПК. Эти устройства позволяли аппаратно выполнять операции с растровыми изображениями, делая работу с графической операционной системой более комфортной.

Примерно в то же время – на протяжении всех 1980-х годов – компания Silicon Graphics, работавшая в области профессионального оборудования и ПО, стремилась вывести трехмерную графику на различные рынки, в том числе приложения для правительства и министерства обороны, визуализация при решении научно-технических задач, а также инструменты для создания впечатляющих кинематографических эффектов. В 1992 году Silicon Graphics раскрыла программный интерфейс к своему оборудованию, выпустив библиотеку OpenGL. Silicon Graphics рассчитывала, что OpenGL станет стандартным, платформенно независимым методом написания трехмерных графических приложений. Как и в случае параллельной обработки и новых CPU, приход новых технологий в потребительские приложения – всего лишь вопрос времени.

К середине 1990-х годов спрос на потребительские приложения с трехмерной графикой резко увеличился, что подготовило условия для двух весьма существенных направлений разработки. Во-первых, выход на рынок игр шутеров от первого лица (First Person Shooter, FPS), таких как Doom, Duke Nukem 3D и Quake, знаменовал начало гонки за создание все более и более реалистичных трехмерных сцен для игр на ПК. Хотя в конечном итоге 3D-графика проникнет практически

во все компьютерные игры, популярность только нарождающихся «стрелялок» от первого лица существенно ускорила внедрение 3D-графики в компьютеры потребительского класса. В то же время такие компании, как NVIDIA, ATI Technologies и 3dfx Interactive, начали выпускать доступные по цене графические ускорители, способные заинтересовать широкую публику. Все это закрепило за 3D-графикой место на рынке перспективных технологий.

Выпуск компанией NVIDIA карты GeForce 256 еще больше расширил возможности графического оборудования для потребительских компьютеров. Впервые вычисление геометрических преобразований и освещения сцены стало возможно производить непосредственно в графическом процессоре, что позволило создавать еще более визуально привлекательные приложения. Поскольку обработка преобразований и освещения уже входили неотъемлемой частью в графический конвейер OpenGL, выход GeForce 256 ознаменовал начало этапа реализации все большего и большего числа компонентов графического конвейера аппаратно.

А с точки зрения параллельных вычислений, выпуск в 2001 году серии GeForce 3 представляет, пожалуй, самый важный прорыв в технологии производства GPU. Это была первая микросхема, в которой был реализован тогда еще новый стандарт Microsoft DirectX 8.0. Этот стандарт требовал, чтобы совместимое оборудование включало возможность программируемой обработки вершин и пикселов (шейдинга). Впервые разработчики получили средства для частичного контроля над тем, какие имено вычисления будут проводиться на GPU.

1.3.2. Ранние этапы GPU-вычислений

Появление на рынке GPU с программируемым конвейером привлекло внимание многих исследователей к возможности использования графического оборудования не только для рендеринга изображений средствами OpenGL или DirectX. В те первые годы вычисления с помощью GPU были чрезвычайно запутанными. Поскольку единственным способом взаимодействия с GPU оставались графические API типа OpenGL и DirectX, любая попытка запрограммировать для GPU произвольные вычисления была подвержена ограничениям, налагаемым графическим API. Поэтому исследователи старались представить задачу общего характера как традиционный рендеринг.

По существу, GPU начала 2000-х годов предназначались для вычисления цвета каждого пикселя на экране с помощью программируемых арифметических устройств, *пиксельных шейдеров* (pixel shader). В общем случае пиксельный шейдер получает на входе координаты `(x, y)` точки на экране и некоторую дополнительную информацию, а на выходе должен выдать конечный цвет этой точки. В качестве дополнительной информации могут выступать входные цвета, текстурные координаты или иные атрибуты, передаваемые шейдеру на этапе его выполнения. Но поскольку арифметические действия, производимые над входными цветами и текстурами, полностью контролируются программистом, то, как заметили исследователи, в качестве «цветов» могли выступать *любые* данные.

Если на вход подавались числовые данные, содержащие не цвета, то ничто не мешало программисту написать шейдер, выполняющий с ними произвольные вычисления. GPU возвращал результат как окончательный «цвет» пикселя, хотя на деле цветом оказывался результат запрограммированных вычислений над входными данными. Результат можно было считать в программу, а GPU было безразлично, как именно он интерпретируется. Таким образом, программист «обманом» заставлял GPU проделать вычисления, не имеющие никакого отношения к рендерингу, подавая их под личиной стандартной задачи рендеринга. Трюк остроумный, но уж больно неестественный.

Поскольку скорость выполнения арифметических операций на GPU была очень высока, результаты первых экспериментов прочили GPU-вычислениям блестящее будущее. Однако модель программирования была слишком ограничивающей для формирования критической массы разработчиков. Имели место жесткие ограничения на ресурсы, поскольку программа могла получать входные данные только в виде горстки цветов и текстурных блоков. Существовали серьезные ограничения на то, как и в какое место памяти можно записывать результаты, поэтому алгоритмы, для которых требовалась возможность записывать в произвольные ячейки памяти (с разбросом, scatter) на GPU не могли быть запущены. Кроме того, было почти невозможно предсказать, как конкретный GPU поведет себя в отношении чисел с плавающей точкой (и будет ли вообще их обрабатывать), поэтому для большинства научных расчетов GPU оказывался непригоден. Наконец, в процессе разработки неизбежно случается, что программа выдает неправильные результаты, не завершается или попросту подвешивает компьютер, но никакого сколько-нибудь приемлемого способа отладки кода, исполняемого GPU, не существовало.

Более того, всякий человек, которого эти ограничения не напугали и который все-таки хотел использовать GPU для выполнения вычислений общего назначения, должен был выучить OpenGL или DirectX, так как никакого другого способа взаимодействия с GPU не было. А это означает, что не только данные следует хранить в виде графических текстур и вычисления над ними выполнять путем вызова функций OpenGL или DirectX, но и сами вычисления записывать на специальных языках графического программирования – *шейдерных языках*. Необходимость работать в жестких рамках ограничений на ресурсы и способы программирования да при этом еще изучать компьютерную графику и шейдерные языки – и все только для того, чтобы в будущем воспользоваться вычислительной мощью GPU, – оказалась слишком серьезным препятствием на пути к широкому признанию технологии.

1.4. Технология CUDA

Лишь спустя пять лет после выпуска серии GeForce 3 наступил расцвет GPU-вычислений. В ноябре 2006 года NVIDIA торжественно объявила о выпуске первого в истории GPU с поддержкой стандарта DirectX 10, GeForce 8800 GTX. Он был построен на архитектуре CUDA. Она включала несколько новых компонен-

тов, предназначенных исключительно для GPU-вычислений и призванных снять многие ограничения, которые препятствовали полноценному применению прежних графических процессоров для вычислений общего назначения.

1.4.1. Что такое архитектура CUDA?

В отличие от предыдущих поколений GPU, в которых вычислительные ресурсы подразделялись на вершинные и пиксельные шейдеры, в архитектуру CUDA включен унифицированный шейдерный конвейер, позволяющий программе, выполняющей вычисления общего назначения, задействовать любое арифметически-логическое устройство (АЛУ, ALU), входящее в микросхему. Поскольку NVIDIA рассчитывала, что новое семейство графических процессоров будет использоваться для вычислений общего назначения, то АЛУ были сконструированы с учетом требований IEEE к арифметическим операциям над числами с плавающей точкой одинарной точности; кроме того, был разработан набор команд, ориентированный на вычисления общего назначения, а не только на графику. Наконец, исполняющим устройствам GPU был разрешен произвольный доступ к памяти для чтения и записи, а также доступ к программно-управляемому кэшу, получившему название *разделяемая память* (shared memory). Все эти средства были добавлены в архитектуру CUDA с целью создать GPU, который отлично справлялся бы с вычислениями общего назначения, а не только с традиционными задачами компьютерной графики.

1.4.2. Использование архитектуры CUDA

Однако усилия NVIDIA, направленные на то, чтобы предложить потребителям продукт, одинаково хорошо приспособленный для вычислений общего назначения и для обработки графики, не могли ограничиться разработкой оборудования, построенного на базе архитектуры CUDA. Сколько бы новых возможностей для вычислений ни включала NVIDIA в свои микросхемы, все равно единственным способом доступа к ним оставался OpenGL или DirectX. И, значит, пользователи по-прежнему должны были маскировать вычисления под графические задачи и оформлять их на шейдерном языке типа GLSL, входящего в OpenGL, или Microsoft HLSL.

Чтобы охватить максимальное количество разработчиков, NVIDIA взяла стандартный язык C и дополнила его несколькими новыми ключевыми словами, позволяющими задействовать специальные средства, присущие архитектуре CUDA. Через несколько месяцев после выпуска GeForce 8800 GTX открыла доступ к компилятору нового языка CUDA C. Он стал первым языком, специально разработанным компанией по производству GPU с целью упростить программирование GPU для вычислений общего назначения.

Помимо создания языка для программирования GPU, NVIDIA предлагает специализированный драйвер, позволяющий использовать возможности массивно-параллельных вычислений в архитектуре CUDA. Теперь пользователям нет

нужды изучать программные интерфейсы OpenGL или DirectX или представлять свои задачи в виде задач компьютерной графики.

1.5. Применение CUDA

Со времени дебюта в 2007 году на языке CUDA C было написано немало приложений в различных отраслях промышленности. Во многих случаях за счет этого удалось добиться повышения производительности на несколько порядков. Кроме того, приложения, работающие на графических процессорах NVIDIA, демонстрируют большую производительность в расчете на доллар вложенных средств и на ватт потребленной энергии по сравнению с реализациями, построенными на базе одних лишь центральных процессоров. Ниже описаны несколько направлений успешного применения языка CUDA C и архитектуры CUDA.

1.5.1. Обработка медицинских изображений

За последние 20 лет резко возросло количество женщин, страдающих раком груди. Но благодаря неустанным усилиям многих людей в последние годы интенсифицировались исследования, направленные на предотвращение и лечение этого страшного заболевания. Конечная цель состоит в том, чтобы диагностировать рак груди на ранней стадии, дабы избежать губительных побочных эффектов облучения и химиотерапии, постоянных напоминаний, которые оставляет хирургическое вмешательство, и летальных исходов в случаях, когда лечение не помогает. Поэтому исследователи прилагают все силы, чтобы отыскать быстрые, точные способы с минимальным вмешательством для диагностики начальных симптомов рака груди.

У маммограммы, одного из лучших современных методов ранней диагностики рака груди, есть несколько существенных ограничений. Необходимо получить два или более изображений, причем пленку должен проявить и интерпретировать опытный врач, способный распознать потенциальную опухоль. Кроме того, частые рентгеновские исследования сопряжены с риском облучения пациентки. После тщательного изучения врачам нередко требуется изображение конкретного участка – и даже биопсия, – чтобы исключить возможность рака. Такие ложноположительные результаты приводят к дорогостоящим дополнительным исследованиям и вызывают у пациента ненужный стресс в ожидании окончательного заключения.

Ультразвуковые исследования безопаснее рентгеновских, поэтому врачи часто назначают их в сочетании с маммографией при диагностике и лечении рака груди. Однако у традиционного УЗИ груди тоже есть свои ограничения. Попытки решить эту проблему привели к образованию компании TechniScan Medical Systems. TechniScan разработала многообещающую методику трехмерного ультразвукового сканирования, но ее решение не было внедрено в практику по очень простой причине: нехватка вычислительных мощностей. Проще говоря, вычисления, необходимые для преобразования собранных в результате УЗИ данных в трехмерное

изображение, занимают слишком много времени, поэтому признаны чрезмерно дорогими для клинического применения.

Появление первого GPU компании NVIDIA, основанного на архитектуре CUDA, вкупе с языком программирования CUDA C стало той платформой, на которой TechniScan смогла претворить мечты своих основателей в реальность. В системе ультразвукового сканирования Svara для получения изображения груди пациентки применяются ультразвуковые волны. В системе используются два процессора NVIDIA Tesla C1060, обрабатывающие 35 Гб данных, собранных за 15 минут сканирования. Благодаря вычислительной мощности Tesla C1060 уже через 20 минут врач может рассматривать детализированное трехмерное изображение груди пациентки. TechniScan рассчитывает на широкое внедрение системы Svara, начиная с 2010 года.

1.5.2. Вычислительная гидродинамика

В течение многих лет проектирование эффективных винтов и лопаток оставалось черной магией. Невероятно сложное движение воздуха и жидкости, обтекающих эти устройства, невозможно исследовать на простых моделях, а точные модели оказываются слишком ресурсоемкими с вычислительной точки зрения. Лишь самые мощные суперкомпьютеры могли предложить ресурсы, достаточные для обсчета численных моделей, требуемого для разработки и проверки конструкции. Поскольку лишь немногие организации имеют доступ к таким машинам, проектирование подобных механизмов находится в застое.

Кэмбриджский университет, продолжая великие традиции, заложенные Чарльзом Бэббиджем, является полигоном для активных исследований в области параллельных вычислений. Д-р Грэхем Пуллан и д-р Тобиас Брэндвик из «многоядерной группы» правильно оценили потенциал архитектуры CUDA для беспрецедентного ускорения гидродинамических расчетов. Первоначальная прикидка показала, что персональная рабочая станция, оборудованная GPU, уже способна достичь приемлемой производительности. Впоследствии небольшой кластер, собранный из GPU, с легкостью обставлял куда более дорогие суперкомпьютеры, подтвердив предположение о том, что GPU компании NVIDIA отлично подходят для решения интересующих их задач.

Для исследователей из Кэмбриджа колоссальный выигрыш в производительности, полученный за счет использования CUDA C, оказался больше, чем простое дополнение к ресурсам их суперкомпьютера. Наличие многочисленных дешевых вычислительных устройств позволило ученым проводить эксперименты и быстро получать их результаты. А когда результат численного эксперимента доступен через несколько секунд, возникает обратная связь, приводящая в конечном итоге к прорыву. В результате применение дешевых GPU-кластеров принципиально изменило подход к проведению исследований. Почти интерактивное моделирование открыло новые возможности для новаторских идей в области, которая раньше страдала от застоя.

1.5.3. Науки об окружающей среде

Естественным следствием быстрой индустриализации глобальной экономики является увеличивающая потребность в потребительских товарах, которые не загрязняют окружающую среду. Озабоченность в связи с изменением климата, неуклонно растущие цены на топливо и увеличение концентрации загрязняющих веществ в воде и воздухе – все это остро поставило вопрос о косвенном ущербе от промышленных выбросов. Моющие и чистящие средства давно уже стали необходимыми, но потенциально вредными потребительскими продуктами ежедневного пользования. Поэтому многие ученые начали искать, как сократить пагубное влияние моющих средств на окружающую среду, не снижая их эффективности. Но получить что-то из ничего – задача не из простых.

Основными компонентами чистящих средств являются поверхностно-активные вещества (ПАВ). Именно их молекулы определяют чистящую способность и текстуру стиральных порошков и шампуней, но они же оказывают разрушающее воздействие на окружающую среду. Эти молекулы сцепляются с грязью, а затем соединяются с водой, так что ПАВ смывается вместе с грязью. Традиционное измерение чистящей способности нового вещества требует длительных лабораторных исследований различных комбинаций материалов и загрязнений. Неудивительно, что процедура оказывается медленной и дорогой.

Университет Темпл работает совместно с промышленным гигантом, компанией Procter & Gamble, над моделированием взаимодействия молекул ПАВ с грязью, водой и другими материалами. Внедрение компьютерных моделей позволило не только ускорить традиционный подход, но и расширить диапазон исследований, включив многочисленные вариации окружающих условий, – дело, совершенно немыслимое в прошлом. Исследователи из университета Темпл воспользовались GPU-ускоренной программой моделирования Highly Optimized Object Oriented Many-particle Dynamics (HOOMD – высокооптимизированная объектно-ориентированная многочастичная динамика), разработанной в лаборатории Эймса при министерстве энергетики. Распределив моделирование между двумя GPU NVIDIA Tesla, они сумели достичь производительности, эквивалентной суперкомпьютеру Cray XT3 с 128 процессорными ядрами или IBM BlueGene/L с 1024 процессорами. Увеличив количество Tesla GPU, они уже могут моделировать взаимодействия ПАВ в 16 раз быстрее, чем на старых платформах. Поскольку архитектура CUDA помогла сократить время исчерпывающего моделирования с нескольких недель до нескольких часов, то в ближайшем будущем должны появиться новые продукты, одновременно более эффективные и менее вредные для окружающей среды.

1.6. Резюме

Компьютерная индустрия стоит на пороге революции, связанной с массовым переходом к параллельным вычислениям, а язык CUDA C, разработанный компанией NVIDIA, пока что является одним из самых успешных языков для парал-