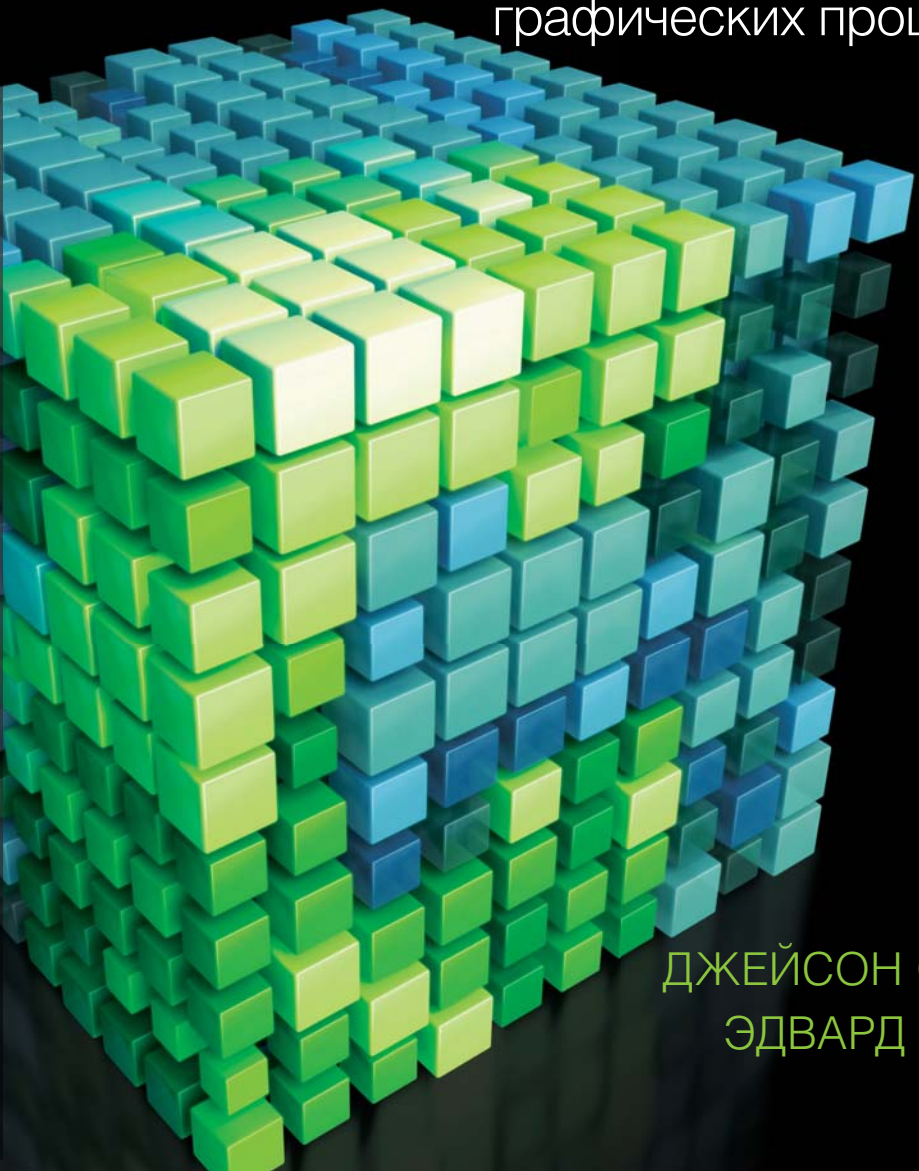
NVIDIA

ТЕХНОЛОГИЯ CUDA В ПРИМЕРАХ

Введение в программирование графических процессоров



ДМК ПРЕСС ИЗДАТЕЛЬСТВО

ДЖЕЙСОН САНДЕРС
ЭДВАРД КЭНДРОТ

УДК 004.3'144: 004.383.5CUDA
ББК 32.973.26-04
С18

Сандерс Дж., Кэндрот Э.
С18 Технология CUDA в примерах: введение в программирование графических процессоров: Пер. с англ. Слинкина А. А., научный редактор Боресков А. В. – М.: ДМК Пресс, 2015. – 232 с.: ил.
ISBN 978-5-97060-297-3

CUDA – вычислительная архитектура, разработанная компанией NVIDIA и предназначенная для разработки параллельных программ. В сочетании с развитой программной платформой архитектура CUDA позволяет программисту задействовать невероятную мощь графических процессоров для создания высокопроизводительных приложений, включая научные, инженерные и финансовые приложения.

Книга написана двумя старшими членами команды по разработке программной платформы CUDA. Новая технология представлена в ней с точки зрения программиста. Авторы рассматривают все аспекты разработки на CUDA, иллюстрируя изложение работающими примерами. После краткого введения в саму платформу и архитектуру CUDA, а также беглого обзора языка CUDA C, начинается подробное обсуждение различных функциональных возможностей CUDA и связанных с ними компромиссов. Вы узнаете, когда следует использовать то или иное средство и как писать программы, демонстрирующие поистине выдающуюся производительность.

Издание предназначено для программистов, а также будет полезно инженерам, научным работникам и студентам вузов.

УДК 004.3'144: 004.383.5CUDA
ББК 32.973.26-04

ISBN 978-0-13-138768-3 (анг.)
ISBN 978-5-97060-297-3 (рус.)

Содержание

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-

лельных вычислений. Прочитав эту книгу, вы научитесь писать код на CUDA C. Мы расскажем о специализированных расширениях языка C и программных интерфейсах (API), которые NVIDIA создала для программирования GPU. *Не* предполагается знакомство со стандартами OpenGL или DirectX, а также какие-либо знания в области компьютерной графики.

Мы не останавливаемся на основах программирования на C, поэтому не можем порекомендовать эту книгу людям, совсем незнакомым с программированием. Некоторые знания в области параллельного программирования были бы полезны, но мы *не предполагаем*, что вы когда-либо занимались параллельным программированием по-настоящему. Все необходимые термины и концепции, относящиеся к параллельному программированию, объясняются прямо в тексте. Не исключено даже, что вы столкнетесь с ситуацией, когда допущения, сделанные благодаря знанию традиционного параллельного программирования, оказываются неверными в применении к программированию GPU. Так что в действительности единственным обязательным условием для чтения этой книги является наличие какого-то опыта программирования на языке C или C++.

В следующей главе мы расскажем, как подготовить свой компьютер для GPU-вычислений – установить все необходимые аппаратные и программные компоненты. После этого можно будет приступить к изучению CUDA C. Если вы уже имеете опыт работы с CUDA C или уверены, что система настроена для разработки на CUDA C, то можете сразу перейти к главе 3.

Глава 2. Приступая к работе

Надеемся, что глава 1 пробудила в вас желание поскорее приступить к изучению CUDA C. Поскольку подход, принятый в этой книге, – обучение на примерах, то вам понадобится среда разработки. Конечно, можно было бы постоять на обочине и понаблюдать за происходящим, но думается, что вы получите больше удовольствия и дольше сохраните интерес, если вмешаетесь и, не откладывая в долгий ящик, начнете экспериментировать с CUDA C. Исходя из этого предположения, мы в этой главе расскажем о том, какие аппаратные и программные компоненты понадобятся, чтобы приступить к работе. Спешим порадовать – все программное обеспечение бесплатно, так что деньги можете потратить на то, что вам по душе.

2.1. О чем эта глава

Прочитав эту главу, вы:

- скачаете все необходимые для работы с этой книгой программные компоненты;
- настроите среду для компиляции кода на CUDA C.

2.2. Среда разработки

Прежде чем отправляться в путешествие, необходимо настроить среду, в которой вы будете вести разработку на CUDA C. Для этого понадобятся:

- графический процессор, поддерживающий архитектуру CUDA;
- драйвер устройства NVIDIA;
- комплект средств разработки CUDA (CUDA Toolkit);
- стандартный компилятор языка C.

И сейчас мы подробнее обсудим каждое из этих условий.

2.2.1. Графические процессоры, поддерживающие архитектуру CUDA

К счастью, найти графический процессор, построенный на базе архитектуры CUDA, совсем несложно, потому что таковыми являются все NVIDIA GPU, начиная с выпущенной в 2006 году карты GeForce 8800 GTX. NVIDIA регулярно выпускает новые GPU на базе архитектуры CUDA, поэтому приведенный ниже

список подходящих GPU наверняка неполон. Но так или иначе, все перечисленные в нем GPU поддерживают CUDA.

Полный список имеется на сайте NVIDIA по адресу www.nvidia.com/cuda, хотя можно без опаски считать, что любой из недавно выпущенных (начиная с 2007 года) GPU с графической памятью объемом не менее 256 Мб годится для разработки и исполнения кода на языке CUDA C.

GeForce GTX 480	GeForce 8300 mGPU	Quadro FX 5600
GeForce GTX 470	GeForce 8200 mGPU	Quadro FX 4800
GeForce GTX 295	GeForce 8100 mGPU	Quadro FX 4800 for Mac
GeForce GTX 285	Tesla S2090	Quadro FX 4700 X2
GeForce GTX 285 for Mac	Tesla M2090	Quadro FX 4600
GeForce GTX 280	Tesla S2070	Quadro FX 3800
GeForce GTX 275	Tesla M2070	Quadro FX 3700
GeForce GTX 260	Tesla C2070	Quadro FX 1800
GeForce GTS 250	Tesla S2050	Quadro FX 1800
GeForce GT 220	Tesla M2050	Quadro FX 580
GeForce G210	Tesla C2050	Quadro FX 570
GeForce GTS 150	Tesla S1070	Quadro FX 470
GeForce GT 130	Tesla S1060	Quadro FX 380
GeForce GT 120	Tesla S870	Quadro FX 370
GeForce G100	Tesla C870	Quadro FX 370 Low Profile
GeForce 9800 GX2	Tesla D870	Quadro CX
GeForce 9800 GTX+	Продукты Quadro Mobile	Quadro NVS 450
GeForce 9800 GTX		Quadro NVS 420
GeForce 9800 GT	Quadro FX 3700M	Quadro NVS 295
GeForce 9600 GSO	Quadro FX 3600M	Quadro NVS 290
GeForce 9600 GT	Quadro FX 2700M	Quadro Plex 2100 D4
GeForce 9500 GT	Quadro FX 1700M	Quadro Plex 2100 D2
GeForce 9400GT	Quadro FX 1600M	Quadro Plex 2100 S4
GeForce 8800 Ultra	Quadro FX 770M	Quadro Plex 1000 Model IV
GeForce 8800 GTX	Quadro FX 570M	Продукты GeForce mobile
GeForce 8800 GTS	Quadro FX 370M	
GeForce 8800 GT	Quadro FX 360M	GeForce GTX 280M
GeForce 8800 GS	Quadro NVS 320M	GeForce GTX 260M
GeForce 8600 GTS	Quadro NVS 160M	GeForce GTS 260M
GeForce 8600 GT	Quadro NVS 150M	GeForce GTS 250M
GeForce 8500 GT	Quadro NVS 140M	GeForce GTS 160M
GeForce 8400 GS	Quadro NVS 135M	GeForce GTS 150M
GeForce 9400 mGPU	Quadro NVS 130M	GeForce GT 240M
GeForce 9300 mGPU	Quadro FX 5800	GeForce GT 230M
GeForce GT 130M	GeForce 9700M GTS	GeForce 9200M GS
GeForce G210M	GeForce 9700M GT	GeForce 9100M G
GeForce G110M	GeForce 9650M GS	GeForce 8800M GTS

GeForce G105M	GeForce 9600M GT	GeForce 8700M GT
GeForce G102M	GeForce 9600M GS	GeForce 8600M GT
GeForce 9800M GTX	GeForce 9500M GS	GeForce 8600M GS
GeForce 9800M GT	GeForce 9500M G	GeForce 8400M GT
GeForce 9800M GTS	GeForce 9300M GS	GeForce 8400M GS
GeForce 9800M GS	GeForce 9300M G	

2.2.2. Драйвер устройства NVIDIA

NVIDIA поставляет системное ПО, которое позволяет программам взаимодействовать с оборудованием, поддерживающим CUDA. Если NVIDIA GPU был установлен правильно, то, скорее всего, это ПО уже находится на вашей машине. Но всегда полезно проверить, не появились ли более свежие версии драйверов, поэтому мы рекомендуем зайти на сайт www.nvidia.com/cuda и щелкнуть по ссылке *Download Drivers* (Скачать драйверы). Выберите вариант, соответствующий вашей графической карте и операционной системе, в которой вы намереваетесь вести разработку.

Выполнив все инструкции по установке, вы получите систему с новейшим системным ПО компании NVIDIA.

2.2.3. Комплект средств разработки CUDA Development Toolkit

Имея GPU с поддержкой CUDA и драйвер устройства NVIDIA, вы уже можете исполнять откомпилированный код на CUDA. Это означает, что, скачав какое-нибудь приложение на CUDA, вы сможете выполнить его на своем графическом процессоре. Однако мы предполагаем, что вы хотите не только запускать чужой код, иначе зачем было покупать эту книгу? Если вы собираетесь разрабатывать код для GPU NVIDIA на языке CUDA C, то понадобится дополнительное программное обеспечение. Впрочем, как мы и обещали, оно не будет стоит вам ни копейки.

Подробнее об этом мы будем говорить в следующей главе, но уже сейчас отметим, что поскольку приложения, написанные на CUDA C, исполняются на двух разных процессорах, то понадобятся два компилятора. Один будет компилировать код для GPU, второй – код для CPU. NVIDIA предлагает только компилятор кода, предназначенного для GPU. Как и драйвер устройства, комплект средств разработки *CUDA Toolkit* можно скачать со страницы по адресу http://developer.nvidia.com/object/gpucomputing.html. Она показана на рис. 2.1.

Вас снова попросят выбрать платформу: 32- или 64-разрядную версию Windows XP, Windows Vista, Windows 7, Linux или Mac OS. Из предлагаемых продуктов необходимо скачать только CUDA Toolkit – без него вы не сможете собрать приведенные в этой книге примеры. Кроме того, мы рекомендуем, хотя это и необязательно, скачать примеры, содержащиеся в комплекте GPU Computing SDK;

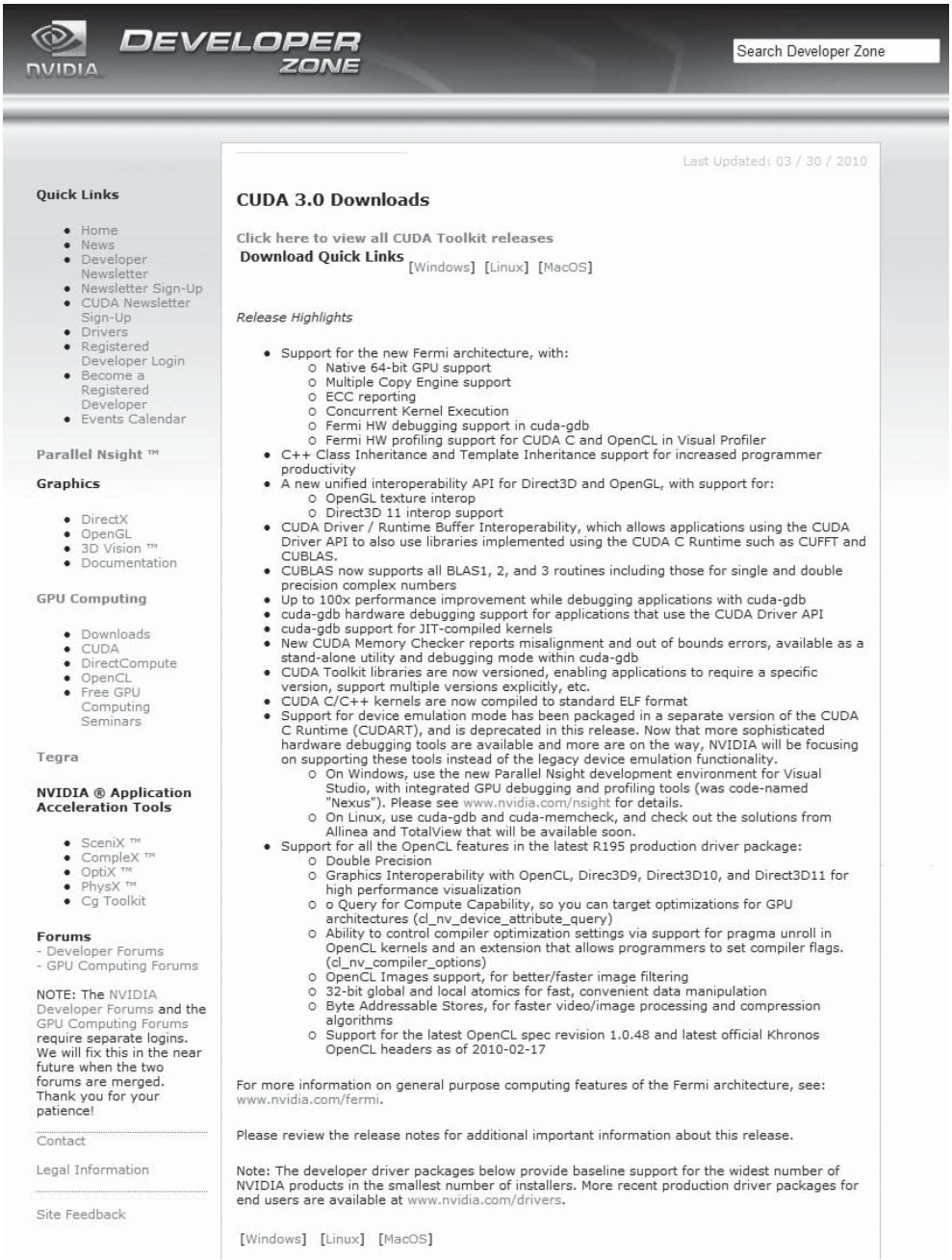


Рис. 2.1. Страница загрузки CUDA