

# CUDA КАТИТСЯ ПК?

## Перспективы параллельных вычислений на GPU от Nvidia

Среди особенностей графических плат Nvidia, помимо поддержки DirectX 10, аппаратных средств воспроизведения HD-видео, аппаратного физического движка PhysX и функций стереоскопии, появившихся в последнем поколении GPU, непременно упоминается и о таинственном CUDA. Эта технология в общем-то не связана напрямую с графикой, а используется для параллельных вычислений. Как именно, какие преимущества получают разработчики, есть ли аналогичные технологии — обо всем этом мы расспросили Юрия Уральского, ведущего специалиста по технологиям компании Nvidia.



Юрий Уральский, Nvidia

— Давайте начнем с азов. Что такое CUDA?

— CUDA — это сокращение от Compute Unified Device Architecture, то есть унифицированная архитектура компьютерных вычислений. Это комплекс аппаратно-программных средств, который позволяет запускать произвольный код на графическом процессоре (GPU). Идея использования графического процессора для вычислений общего назначения появилась около пяти лет назад. В 2003 г. даже сформировалось комьюнити, которое хотело реализовать возможности таких вычислений. Соответствующие методы получили название GPGPU — General-purpose Computing on GPU). Когда CUDA еще не существовала, разработчики использовали GPU через стандартный графический API — Open GL или DirectX. Однако чтобы запустить процесс вычислений на GPU, необходимо было рисовать какую-то геометрию, использовать возможности графического API полностью. Это было не очень удобно или даже очень неудобно.

— В чем же преимущества GPU-вычислений?

— Даже до появления CUDA было очевидно, что возможности GPU превосходят CPU для определенного рода задач. В первую очередь тех, которые предполагают высокую степень параллелизма. Например, решение дифференциальных уравнений, задач из об-

ласти гидродинамики, обработки изображений и т.п. Мы увидели, что это направление очень популярно, и попытались проанализировать все проблемы, с которыми сталкивались разработчики. Ответом стала технология CUDA. Выход ее первых версий состоялся в начале 2007 г. Тогда и появился SDK, который был призван облегчить разработчикам доступ к вычислительным ресурсам графических процессоров.

— Можно ли говорить о том, что на популярность концепции GPGPU повлияли недостатки x86-архитектуры в плане вычислений?

— Пожалуй, не совсем. Архитектура x86 ориентируется прежде всего на центральные процессоры, где универсальность играет более важную роль, чем производительность. GPU как процессор радикально отличается от традиционной CPU-архитектуры и сориентирован на высокопроизводительные вычисления. CUDA пытается быть для GPU тем же самым, что и набор x86-инструкций для центральных процессоров — единой стабильной программной моделью и платформой для разработки программ, ориентированной на вычисления. В другой существующей сегодня архитектуре Cell пока нет стабильной программной модели, на сегодняшний день — это предмет исследований. А CUDA уже реально используется в ряде коммерческих приложений.

— Каковы ключевые отличия между вычислениями на CPU и GPU?

— На протяжении последних 10–15 лет традиционные CPU наращивали производительность при выполнении последовательного кода путем усложнения архитектуры. Анализируя код, они пытались увеличить его эффективность на этапе исполнения. Причем код продолжал оставаться непараллельным. Продолжалось увеличение размеров кеш-памяти, наращивание тактовой частоты. Однако сегодня возможности дальнейшего наращивания скорости выполнения последовательного кода практически подошли к концу. Единственным выходом является паралле-

лизм. Именно поэтому GPU становится все более и более популярным как вычислитель. При условии правильного написания кода под GPU можно получить большую производительность на квадратный миллиметр кристалла, единицу потраченной энергии, доллар и т.п. Мы часто при дизайне GPU оперируем показателями экономической энергоэффективности. С точки зрения этих параметров удельная производительность графических процессоров намного выше, чем у CPU. Они используют совершенно другой подход к наращиванию производительности: вместо того чтобы усложнять последовательный код, графический процессор использует массивы, или «море» вычислителей, которые работают параллельно, но, как правило, на более низкой частоте. Такая архитектура кардинально отличается, и CUDA — программно-аппаратная платформа, которая позволяет разработчикам использовать потенциал этих массивов, при этом применяя стандартные средства разработки.

— Какие языки программирования сегодня поддерживает CUDA?

— В настоящее время мы поддерживаем язык Си, поскольку считаем, что он более распространен, и чтобы завоевать доверие разработчиков. Задача CUDA достаточно четко очерчена. Мы не говорим, что можем ускорить все существующие приложения — браузеры, текстовые редакторы, игры. В первую очередь нам интересны приложения, богатые обработкой данных, которая хорошо ложится на массивно-параллельные архитектуры. Спектр областей применения программной модели CUDA довольно широк — от компьютерных игрушек до суперкомпьютеров и кластеров (уже создаются подобные GRID-кластеры, использующие GPU). Наша цель — создать экосистему, комьюнити разработчиков, которые употребили бы платформу CUDA для разработки приложений.

— Можно ли считать OpenCL вашим основным конкурентом в сфере GPU-вычислений?

— Ни в коем случае. OpenCL не будет привязан к одной платформе. Этот стандарт поддерживают многие вендоры, включая и нас. Но в рамках CUDA. В процессе разработки данного стандарта мы тесно сотрудничали с Apple, рабочей группой, и многие идеи, впервые реализованные в CUDA, впоследствии перешли в OpenCL. Это означает, что разработчики, которые пишут софт на CUDA, в общем-то, не столкнутся с серьезными проблемами при портировании программ на Open CL.

Впрочем, нельзя сказать, что программы на уровне кода полностью совместимы. Программные модели CUDA и OpenCL очень похожи и в то же время не похожи. Наша технология более тесно интегрирована с языком Си. Если перед разработчиком стоит задача использовать в разработке CUDA, то он может сосредоточиться на том участке кода, где проводятся 90% вычислений, и переписать конкретную функцию, не трогая остальное. В случае же с OpenCL потребуется значительно больше переделок.

Общее то, что и в том, и в другом случае алгоритм вычислений придется менять. Разработчикам не стоит рассчитывать на то, что, взяв существующий код и скопировав его, они получат сколь-нибудь существенную прибавку в скорости. Чтобы полностью реализовать потенциал массивно-параллельного кода, надо будет перестроить алгоритм. Это и есть основной подводный камень, с которым так или иначе придется столкнуться...

— А что Вы можете сказать о желании Intel вытеснить с рынка производи-

телей GPU и реализовать соответствующие функции на базе процессоров традиционных архитектур?

— В настоящее время нам сложно комментировать подобные анонсы. Сперва хотелось бы увидеть что-то конкретное. Как только мы увидим соответствующее железо, протестируем его — тогда можно будет что-то конкретно обсуждать. Говорить о чем-то сейчас не имеет смысла. Кроме анонсов на бумаге, ничего нет. У нас ситуация достаточно простая — со дня анонса CUDA мы поставили клиентам порядка 110–120 млн процессорных микросхем. Это значительный рынок, есть достаточно большое число устройств, на которых могут работать параллельные программы. Выкинуть с рынка такой объем будет достаточно тяжело. Мы тоже на месте не стоим и продолжаем развивать эту технологию.

Мы хотим сделать CUDA стабильной средой для программирования и обеспечить поддержку всех будущих версий нашей технологии — приложения, сделанные, скажем, на CUDA 2.0, смогут запускаться и в будущих версиях. Еще одна особенность, встроенная в платформу CUDA, — это возможность масштабирования. Написав программу для одного семейства наших графических чипов, разработчик может быть уверен, что она запустится и на более мощном GPU и обеспечит более быстрое выполнение.

— Назовите наиболее интересные проекты, сделанные на CUDA.

— У каждого разработчика свой критерий интересности. Для одних это компьютерная игра, для других — модуль для MatLab, который тоже поддер-

живает вычисления на CUDA. На нашей страничке Cuda Zone (ее адрес — [www.nvidia.com/object/cuda\\_home.html](http://www.nvidia.com/object/cuda_home.html)) собрана очень подробная информация о приложениях, которые используют GPU-вычисления. Среди них проекты гипердинамики, моделирования частиц и т.п. Среди корпоративных приложений — пакет PowerDirector7 Ultra, TMPGEnc 4 Xpress, Photoshop и т.п. Сегодня набирают популярность высокопроизводительные вычисления на базе GRID-сетей на базе GPU (примеры — SETI@Home, «облачный» сервис Hooree и др.). То есть наша технология не ограничивается одними лишь десктопными приложениями. Чем шире область применения GPU, тем нам интереснее.

— Каковы ваши усилия по продвижению CUDA, что называется, в массы?

— Нами разработан курс по программированию на CUDA. Он читается в разных университетах США и Европы. В этом году мы подготовили аналогичные лекции и для российских вузов. Они читаются, например, в МГУ. Курс рассчитан на один семестр. Его посещали не только студенты, но и сторонние разработчики. [Мы будем продолжать вести этот курс в МГУ — его преподает Борисов, в планах которого, кстати, сделать учебник по технологии CUDA.] Возможно появление аналогичных курсов и в других российских университетах. Если кто-то проявит желание читать курс по CUDA, то милости просим, мы всегда готовы помочь в организации обучения и открыты для всех, кто заинтересован в технологии CUDA. ■

## CUDA: технические детали

Графический процессор, по сути, является сопроцессором для CPU — он оснащен собственной памятью и может выполнять вычисления одновременно с ним. Каждый GPU архитектурно намного проще, чем CPU. В его задачи входит и считывание пересылаемых центральным хост-процессором данных из локальной, текстурной памяти, и чтение-запись глобальной памяти. Отдельные потоки получают доступ к локальной памяти и памяти регистров; при необходимости их можно объединять в блоки с общей разделяемой памятью, а блоки — в сетки. Программная модель CUDA, о которой говорилось в интервью, может либо использовать библиотеки BLAS (Basic Linear Algebra Subprograms — базовые подпрограммы линейной алгебры), CUFFT (Fastest Fourier Transform — библиотека быстрого преобразования Фурье), что, с одной стороны, достаточно просто, но не так эффективно, либо вызывать низкоуровневые функции CUDA API (в Windows это динамическая библиотека cudart.dll). Последний способ дает больший контроль над приложением, но сложнее при кодировании. С точки зрения разработчика для этого требуются видеокарты с поддержкой технологии CUDA, со-

ответствующие драйверы, а также инструментальный CUDA Toolkit и CUDA SDK, доступные как в среде Windows, так и в Linux и MacOS X.

Программа для CUDA — это обычный Си-код с тремя макродополнениями: `__global__`, `__device__` и `__host__` (с двумя подчеркиваниями в начале и в конце слова). Например: `__global__ void functionA (type val)`. Функция, которая объявлена с помощью первого тега, исполняется на GPU, но запускается с центрального хост-процессора, а с помощью второго тега — полностью обрабатывается графическим чипом. На эти функции, естественно, накладывается ряд ограничений. Они, к примеру, не могут содержать рекурсивные вызовы, объявлять статические переменные оператором `static` и получать переменное число аргументов. Проектируя функцию, разработчик должен учитывать специфику GPU, включая и то, что обращение к глобальной и локальной памяти графического процессора — это довольно медленная операция. В связи с этим алго-

ритм в целом нужно разделить на отдельные модули или блоки таким образом, чтобы стандартный код обрабатывал весь массив. Сами данные при этом также стоит разделить на несколько непересекающихся блоков. Для дальнейшей оптимизации придется задуматься об оптимальном размещении промежуточных данных в регистрах или общей памяти и следить при этом за тем, чтобы массив регистров не переполнялся. Для управления памятью в API предусмотрен ряд функций, позволяющих выделить, переместить и освободить блок. Остается добавить, что параллельный код на CUDA при помощи специального компилятора преобразуется во внутренний RTX-модуль (Parallel Thread Execution) — один из примеров приведен на рисунке.

