ИСПОЛЬЗОВАНИЕ РЕСУРСОВ GPU ДЛЯ ВЫПОЛНЕНИЯ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

Д.К. Боголепов

Основная идея *метакомпьютинга* состоит в использовании независимых компьютеров, объединенных в сеть, как будто они являются одной большой параллельной машиной или виртуальным суперкомпьютером. Эта идея имеет ряд преимуществ, к которым относится, например, низкая стоимость суммарных вычислительных ресурсов.

В то время как традиционные средства распределенных вычислений были разработаны для небольшого числа гомогенных сильно связанных ресурсов (системы управления кластером), быстрый рост сети Интернет позволил применять эти концепции в большем масштабе. К тому же настольные ПК в корпоративных и домашних условиях не сильно нагружены – обычно используется только одна десятая часть мощности. Поэтому возникает интерес к использованию вычислительных мощностей, которые доступны в виде свободных циклов работы центрального процессора. Эта новая парадигма стала одной из идей *грид*.

Сегодня наблюдается быстро растущий интерес к технологиям грид как со стороны научных исследователей, так и со стороны коммерческих предприятий. Как следствие, сегодня предлагается множество систем для организации распределенных вычислений, которые реализуют определенные концепции грид. К их числу можно отнести такие инструментарии, как Alchemi .NET Framework [http://www.alchemi.net], NGrid [http://ngrid.sourceforge.net], X-Com [http://x-com.parallel.ru], проект XtermWeb [http://www.xtremwebch.net]. Характерной особенностью перечисленных систем является возможность быстрой и простой организации вычислений в имеющейся компьютерной сети, а основная их задача – повышение эффективности использования доступных ресурсов.

Большинство инструментариев для организации распределенных вычислений предполагают использование ресурсов *центрального процессора*. При этом из рассмотрения исключается еще одна важная компонента практически любого ПК – *графический ускоритель* (GPU). Современные ускорители предлагают впечатляющие возможности по обработке изображений, однако они могут использоваться и для решения других задач, даже не имеющих прямого отношения к графике. Таким образом, вполне естественным шагом является привлечение вычислительных возможностей видеокарт для решения задач в распределенной грид-среде. Цель данной работы состоит в том, чтобы расширить существующие инструментарии (например, Alchemi) для использования в процессе распределенных вычислений ресурсов графического адаптера. Далее мы кратко рассмотрим основные принципы использования графических ускорителей для вычислений общего назначения и принципы организации соответствующего инструментария для выполнения распределенных вычислений.

МОДЕЛЬ ПРОГРАММИРОВАНИЯ НА ВИДЕОАДАПТЕРЕ

Существует масса проблем и ограничений при разработке программ для видеоадаптера (так называемых *шейдеров*) по сравнению с приложениями для традиционных архитектур. Так, например, все современные видеоадаптеры *не* разрешают рекурсивные вызовы функций, имеют ограничение на глубину вложенных условных операторов (*if*) и число итераций циклов. И, конечно, сама модель программирования отличается своей спецификой: любая задача, которую требуется решить на видеопроцессоре, должна быть сформулирована как *задача растеризации*. Несмотря на это, решение задачи на видеопроцессоре имеет свои преимущества перед традиционной реализацией с использованием центрального процессора.

Во-первых, при решении некоторой задачи на видеокарте (если, конечно, это не видеоигра или сложная система трехмерного моделирования) центральный процессор остается незагруженным и может использоваться для выполнения других полезных задач. Во-вторых, определенный класс задач может быть решен на видеопроцессоре значительно *быстрее*, чем на центральном процессоре (задача трассировки лучей). Современные видеоадаптеры представляют собой *массивно-параллельные процессоры* для обработки вещественных чисел в формате с плавающей запятой и содержат *сотни* (!) независимых шейдерных процессоров[1]. Таким образом, для решения на видеоадаптере подходят задачи, требующие большого числа схожих вычислений, которые могут быть выполнены независимо.

Важно отметить, что оба крупных производителя – ATI [http://www.ati.com] и NVidia [http://www.nvidia.com] – уделяют вопросу использования видеоадаптеров для вычислений общего назначения значительное внимание, предлагая разработчикам законченные программные решения (технологии Stream Computing и CUDA соответственно). Последние поколения графической аппаратуры

[1] Видеоадаптер NVidia GeForce 8800 содержит 128 шейдерных процессора, ATI 2900XT – 320 шейдерных процессоров.

полностью поддерживают 32-битные числа с плавающей точкой стандарта IEEE-754, что вполне достаточно для многих (хотя и далеко не всех) задач. Поддержка вещественных чисел двойной точности планируется в ближайшем будущем.

Для разъяснения принципов организации вычислений на видеоадаптерах обратимся, например, к задаче сложения двух матриц большой размерности. А именно, пусть A и B – матрицы размера $n \times m$ и требуется вычислить матрицу $C = A + B$. Вполне очевидно, что матрицы A и B легко представить в виде прямоугольных текстур размера $n \times m$ в формате с плавающей точкой (такие текстуры уже несколько лет поддерживаются аппаратурой).

Для инициирования вычисления матрицы C следует отрисовать прямоугольник в окне размера $n \times m$. При этом, конечно, для пользователя данное изображение не имеет смысла, поэтому вывод лучше осуществлять не на экран, а в буфер кадра (в память видеоадаптера). В процессе отрисовки растеризатор сгенерирует ровно $n \times m$ пикселей, каждый из которых будет обработан фрагментным шейдером. Заметим, что отдельные пиксели будут обрабатываться независимо на множестве шейдерных блоков (сотни независимых “потоков”).

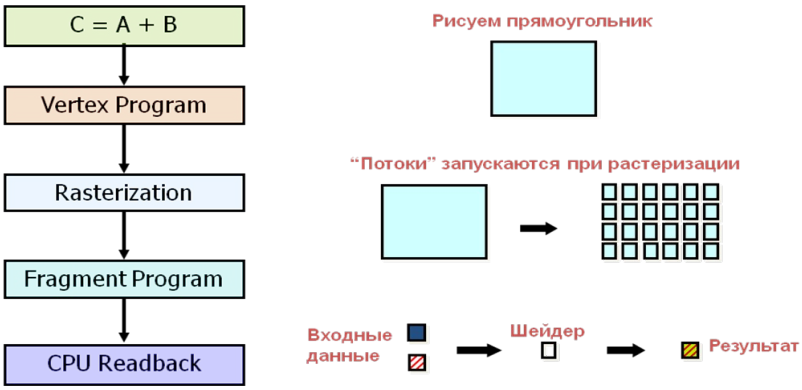


Рис.1. Процесс решения задачи сложения двух матриц. Из презентации “CUDA: Новая архитектура для вычислений на GPU” (КРИ 2008)

Все вычисления выполняются во фрагментном шейдере. На вход фрагментного шейдера поступают координаты обрабатываемого пикселя (обычно они отнормированы и принимают значение в промежутке [0,1]), которые можно интерпретировать как координаты соответствующего элемента матрицы. Кроме того, шейдер имеет доступ к текстурам (которые используются для хранения матриц A и B). Таким образом, в качестве цвета шейдер может вернуть сумму выборок из текстур с матрицами A и B. Важно отметить, что если рендеринг производится в текстуру в формате с плавающей точкой, то рассчитанные фрагментным шейдером компоненты цвета пикселя *не усекаются* по отрезку [0,1] (данное поведение вводит расширение). Иными словами, в качестве цвета можно возвращать произвольный четырехкомпонентный вещественный вектор (в рассматриваемом примере нас интересует лишь одна его компонента).

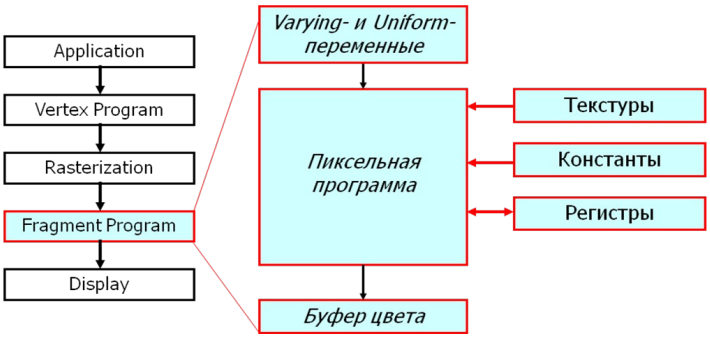


Рис.2. Все полезные вычисления выполняются во фрагментном шейдере. Из презентации “CUDA: Новая архитектура для вычислений на GPU” (КРИ 2008)

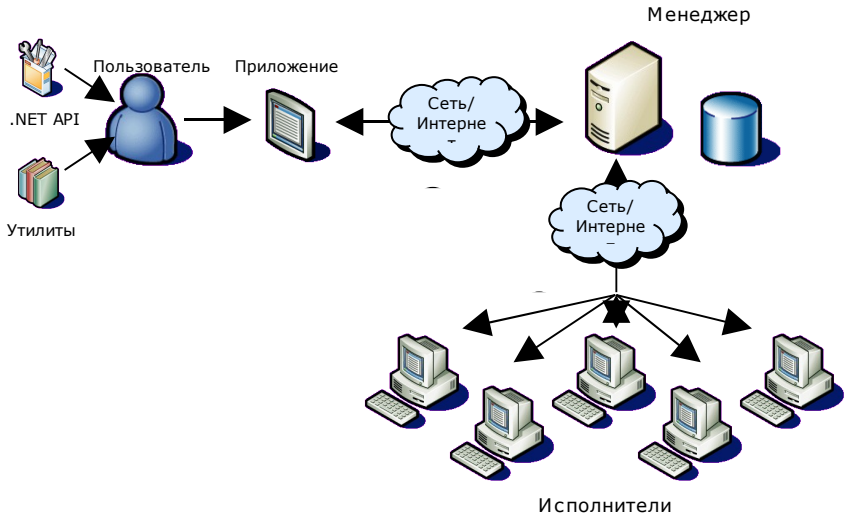
Заключительный этап решения задачи состоит в загрузке результирующей текстуры из памяти видеоадаптера в память центрального процессора (это действие является “бутылочным горлышком” многих задач, т.к. сильно нагружает системную шину) и чтении данной текстуры в массив вещественных чисел (в нем и будут храниться значения вычисленной матрицы C).

Подводя итоги, можно сказать, что задачу сложения матриц можно неплохо приспособить к архитектуре графического адаптера. Это объясняет тот факт, что одними из первых задач общего назначения, которые были решены на видеоадаптере, были задачи сложения и умножения матриц, а также всевозможные действия над векторами.

ПРИНЦИП РАБОТЫ ПЛАНИРУЕМОГО ИНСТРУМЕНТАРИЯ

Несмотря на то, что понятие грид может быть дано на интуитивно простом уровне, практическая реализация такой инфраструктуры сталкивается с множеством трудностей. Основные проблемы связаны с разнородностью, надежностью, разработкой приложений, планированием, управлением ресурсами и безопасностью. Технология Microsoft .NET может быть использована для решения большинства перечисленных проблем. Отметим, в частности, поддержку удаленного исполнения (.NET Remoting и веб-сервисы), многопоточности, безопасности, асинхронного программирования, удаленного доступа к данным, управления исполнением и поддержку нескольких языков программирования. Все это делает инструментарий Microsoft .NET удобной платформой для разработки промежуточного программного обеспечения грид.

В вычислительной грид-сети, построенной на базе рассматриваемого инструментария, можно выделить следующие три основные компоненты: *Менеджер* (*Manager*), *Исполнитель* (*Executor*) и *Пользователь* (*User*).



Как видно из представленной схемы, инструментарий построен по клиент-серверной схеме. Для объединения некоторой совокупности компьютеров в вычислительную сеть, нужно выбрать один узел, который будет играть роль сервера, и установить на него Менеджер. На один или несколько компьютеров сети устанавливаются клиенты – Исполнители, которые настраиваются на работу с Менеджером. При этом если компьютер снабжен производительным видеускорителем, то он может использоваться для выполнения вычислений (в противном случае будет задействован только центральный процессор).

Пользователи могут разрабатывать, выполнять и осуществлять мониторинг грид-приложений, используя специальный .NET API и инструментальные средства, включенные в инструментарий. Любое приложение (*грид-приложение*), разработанное для инструментария, представляет собой совокупность *грид-потоков*, которые концептуально очень похожи на “обычные” потоки. С технической точки зрения грид-поток представляет собой некоторый модуль (экземпляр класса), который содержит порцию данных для обработки и соответствующий программный код и пересылается по сети для выполнения на удаленном Исполнителе. При этом разработчик может создавать *два* типа грид-потоков: для исполнения на центральном процессоре и для исполнения на графическом адаптере. В последнем случае необходима также разработка программы для видеоадаптера – *шейдера*.

После того, как пользователь отправляет грид-приложение на выполнение главному Менеджеру, последний получает некоторое число грид-потоков, ожидающих своего исполнения в грид-сети. Затем Менеджер планирует обработку грид-потоков на доступных Исполнителях и контролирует их дальнейшее выполнение. После завершения вычислений на узел Пользователя пересылаются результаты обработки.

ЛИТЕРАТУРА:

1. General-Purpose Computation Using Graphics Hardware
   http://www.gpgpu.org
2. Расширение ARB_texture_float.
   http://www.opengl.org/registry/specs/ARB/texture_float.txt
3. Расширение ARB_color_buffer_float.
   http://www.opengl.org/registry/specs/ARB/color_buffer_float.txt
4. Расширение ARB_color_buffer_float.
   http://www.opengl.org/registry/specs/ARB/color_buffer_float.txt