Лекция 6

СУПЕРПРОЦЕССОРЫ

Лекция 6. ГРАФИЧЕСКИЕ ПРОЦЕССОРЫ

Графические процессоры *(начиная с* GeForce 8800*)* ***и процессоры*** **Cell**– «**революционные** ***средства высокопроизводительных вычислений».***

GPU (Graphics Proccesing Unit) — сопроцессор, широко применяемый в ПК как ускоритель трехмерной (**3D**) графики. Использование GPU эффективно и при выполнении обычных вычислений – подход GPGPU (General-Purpose Graphics Processing Unit – графические вычислительные устройства общего назначения)

Отличительные особенности **GPU** ***в сравнении с*** **CPU**: 1) их архитектура нацелена на увеличение скорости расчета ***текстур*** и сложных графических объектов; 2) ограниченный набор команд.

Текстура – это растровое изображение, накладываемое на поверхность для придания ей цвета, окраски, иллюзии рельефа. Нанесение текстуры позволяет воспроизвести малые объекты поверхности (шрамы на коже, складки на одежде, мелкие камни и прочие предметы на поверхности стен и почвы).

Задача системы обработки **3D-***графики* – синтез ***в «реальном времени»*** изображения из описания сцены – ее графических примитивов и характера освещения, т. е. способа отражения света каждым из объектов с учетом положения зрителя.

1. НАЧАЛЬНЫЕ СВЕДЕНИЯ

а) ЭТАПЫ СИНТЕЗА 3D-ИЗОБРАЖЕНИЯ

Для синтеза трехмерного изображения необходимо:

– ***принять решение***, какие объекты должны присутствовать в сцене (*видимые и невидимые объекты*);

– ***определить*** местоположение вершин, которые задают каждый из этих объектов;

– ***построить*** по этим вершинам грани;

– ***заполнить*** получившиеся полигоны текстурами в соответствии с *освещением, степенью детализации, учетом перспективных искажений* и т.д.

Чем тщательнее делаются такие расчеты, тем реалистичнее получается 3D-изображение.

Повышение производительности указанных рутинных операций достигается за счет разбивки их по ступеням (конвейеризации) и распараллеливания.

Число вершинных и пиксельных конвейеров – одна из их важнейших характеристик GPU.

Классический процесс конвейерного расчета трехмерного изображения показан на рисунке.

б) КОМПОНЕНТЫ КОНВЕЙЕРА

Вершинный процессор **(Vertex)** – на основании данных от CPU рассчитывает геометрию сцены и положения вершин. Производит дополнительные операции над вершинами – преобразование и освещение (Transform & Lighting, T&L). В каждый момент обрабатываются данные только об одной вершине под управлением специализированной программы – *вершинного шейдера* (Vertex Shader).

Vertex Shader – выполняет операции с вершинами по изменению их параметров и освещению (T&L). Любая вершина в 3D-модели определяется *координатами* X,Y,Z; описывается *характеристиками цвета*, *текстурными координатами* и т.п. Шейдеры вычисляют новые координаты и цвет. Они выполняют и такие операции: *деформация* и *анимация* объектов, *имитация* ткани и др.

Сборка **(Triangle)** – сборка 3D-модели. На этом этапе вершины соединяются между собой линиями, образуя *каркасную модель* – ***полигоны***.

Пиксельный процессор **(Pixel)** – определяет конечные *данные для вывода в кадровый буфер*. Работает на этапе ***растеризации*** *(разбиения объекта на отдельные пикселы)* под управлением ***пиксельного шейдера*** (Pixel Shader). Вычисляет конечное значение цвета пиксела и его Z(*«глубинное»*)-значение.

Pixel Shader – реализует операции: затенение или освещение; текстурирование – выполняет ***блок наложения текстур*** TMU; присвоение цвета и др.

ROP – блок растровых операций (Raster Operations). С использованием ***буфера глубины*** (Z-буфера) определяет и отбрасывает те пикселы, которые будут *не видны* пользователю.

в) НЕДОСТАТКИ КЛАССИЧЕСКОЙ АРХИТЕКТУРЫ

Чем больше конвейеров в классических GPU, тем выше производительность. Число конвейеров должно совпадать с числом **Pixel**, числом **Vertex** и числом **TMU**.

ОСНОВНОЙ НЕДОСТАТОК классической архитектуры – ***невозможность балансировки нагрузки вершинных и пиксельных шейдеров.***

Иногда проблему позволяет решить архитектура, в которой *количество пиксельных процессоров не совпадает с количеством вершинных* (конвейер не классичен). Но … – далеко не всегда.

Пусть в GPU – **4** Vertex и **8** Pixel. При синтезе графики *с насыщенной геометрией* могут быть заняты все 4 Vertex и только **1** Pixel, а оставшиеся **7** Pixel будут бездействовать. Тогда *производительность всего* GPU *определится производительностью и количеством* Vertex. Но для графики *с насыщенными пиксельными эффектами* могут оказаться занятыми только **1** Vertex и все **8** Pixel. Здесь уже *производительность определится быстродействием и количеством* Pixel.

Ситуацию можно поправить, если вместо **4** Vertex и **8** Pixel (в сумме **12**) использовать **12** ***унифицированных процессоров***, которые могли бы выполнять как Vertex Shader, так и Pixel Shader.

г) ПЕРЕХОД НА УНИФИЦИРОВАННЫЕ ПРОЦЕССОРЫ

Унифицированные процессоры – это ***скалярные процессоры*** общего назначения. Векторная архитектура *была традиционной для* GPU из-за преобладания операций с векторами. ПРИМЕР R-G-B-A-*обработки* (компоненты R,G,B задают цвет пиксела, A – его прозрачность): Pixel Shader – операция над **2** векторами при сложении цветов, Vertex Shader – геометрические преобразования **4**x**4**-матриц.

Причина перехода от векторных к скалярным вычислениям в том, что традиционная векторная архитектура мало эффективна в случае обработки сложных смешанных шейдеров, сочетающих векторные и скалярные инструкции.

NVIDIA GeForce 8800 (ноябрь 2006) ***знаменует новую веху в развитии*** GPU – *переход к унифицированным потоковым шейдерным процессорам.* Входные данные поступают на вход унифицированного процессора и им обрабатываются. ***Выходные данные записываются в регистры***, *а затем подаются на вход другого процессора для исполнения следующей операции.*

Развитие программирования ***общего назначения*** на GPU (General Programming on GPU, GPGPU) *логически привело к возникновению технологий, нацеленных* ***на более широкий круг задач, чем растеризация***. В результате компанией Nvidia была создана технология Compute Unified Device Architecture (сокращенно – CUDA).

2. ПОДХОД GPGPU

Сегодня* GPU *из специализированных устройств* 3D-*графики постепенно превращаются в мощные графические вычислительные устройства общего назначения (general-purpose graphics processing unit,* GPGPU*). На простых тестах производительность GeForce 8800 достигает 330 GFLOPs.

Использование GPU в качестве ***вычислительного сопроцессора*** к ПК дает увеличение производительности в 5–20 раз и более. Разработчики ПО ведут работы по оптимизации своих продуктов под вычисления на базе GPU.

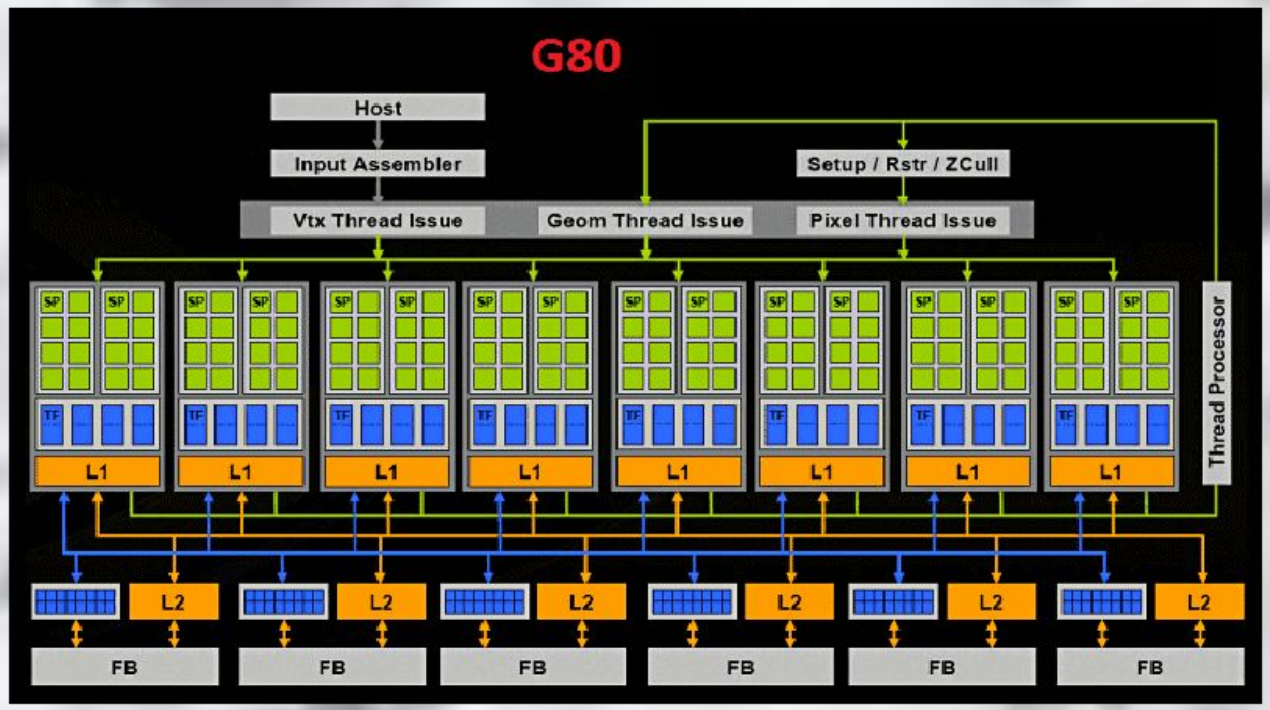
В начале 2009 появились П*суперК* Arbyte SC на базе процессоров Intel нового поколения и GPU NVIDIA Tesla. ***Установлена эффективность*** их использования в области инженерных расчетов, обработки данных геологоразведки, медиаданных, аэрогидродинамики, теплофизики и др.

ПРИМЕР: Персональный суперкомпьютер NVIDIA®Tesla™ S1070 в компактном корпусе 1U, 960 ядер, пиковая производительность – 4 TFLOPs, потребляемая мощность – 700 Вт. ***Использует технологию параллельных вычислений*** NVIDIA CUDA™. Цена – менее 10000 $.

Fermi –одна из последних архитектур GPU с большим количеством процессорных ядер, которая может использоваться в качестве ***мощных акселераторов*** Host при выполнении трудоемких вычислений. На ее основе разрабатываются ***гибридные кластеры*** с использованием как процессорных, так и графических ядер. Они обладают сравнительно низким энергопотреблением, малыми стоимостями внедрения и обслуживания при на порядок большей производительности, чем в обычном кластере.

а) АРХИТЕКТУРА G80 (GeForce 8800)

Начало выпуска карточек на основе G80 ($6818*10^6$ транзисторов) – 2006г. *Это – первая GPU, с унифицированной шейдерной архитектурой, в которой все операции выполняются на скалярных унифицированных (потоковых) процессорах* (streaming processors, SPs). В SP входит блок вычислений с плавающей запятой, целочисленный блок и АЛУ. G80 состоит из 128 SP, объединенные в 16 SMs (Symmetrical Multiprocessors), или 8 TPC (Texture Processing Unit) – по 2 SMs в каждом.



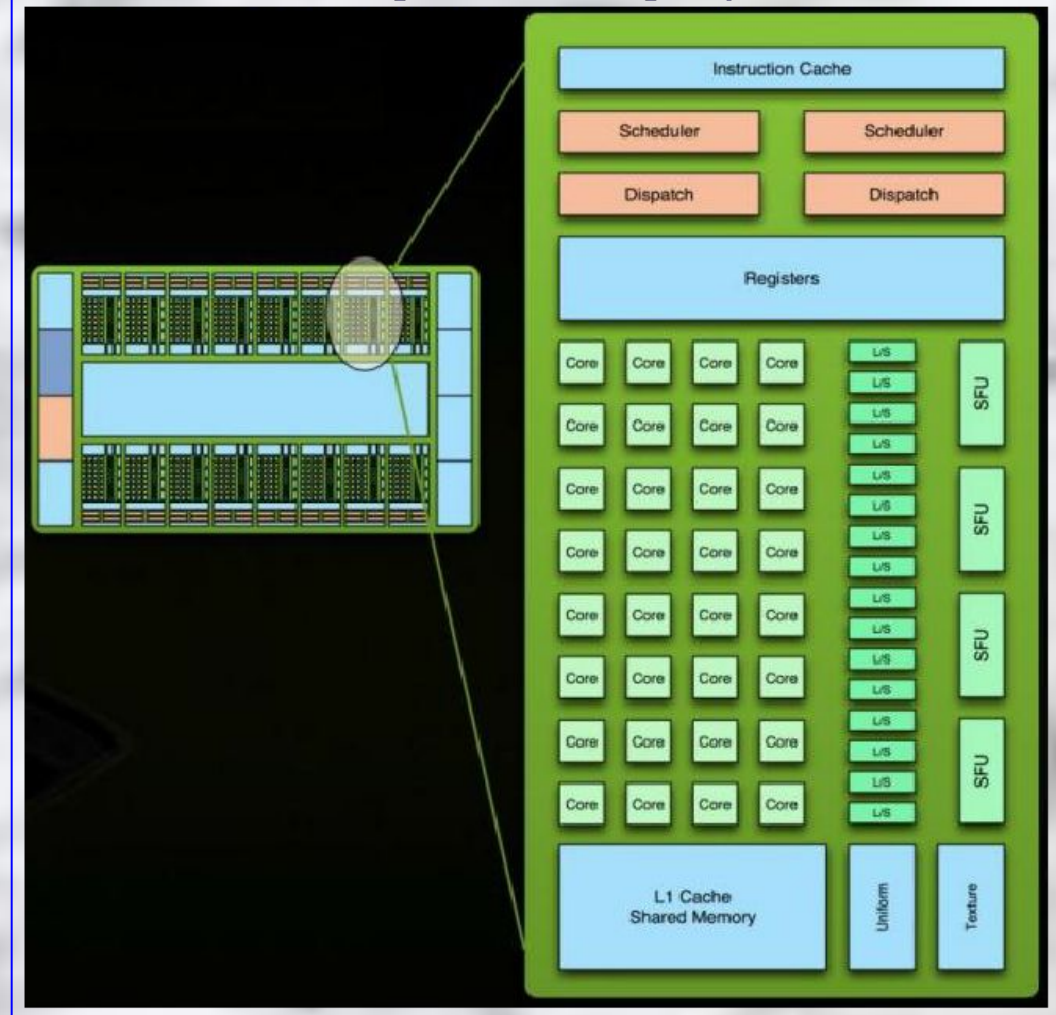
Каждый TPC содержит 16KB shared memory, read-only L1-кэш на 16KB. В каждом SM есть *планировщик* ***варпов***, блок отправки, регистровый блок, 8 блоков текстурной фильтрации и 4 текстурных блока, используемых для операций sin, cos и exp. Он используется и для операций с плавающей запятой.

Блок задач (потоков) выполняется на SM ***частями***, или пулами, называемыми **warp** (***варп***). Логически, **warp** – *минимальное объединение потоков, внутри которого все потоки выполняются* одновременно в стиле SIMD, т.е. во всех потоках внутри **warp** одновременно может выполняться ***только одна инструкция***.

3. FERMI - АРХИТЕКТУРА– GF100/GT300

а) ОБЩАЯ ХАРАКТЕРИСТИКА

Известная как GT300 или GF100, архитектура Ферми – одна из последних и наиболее совершенных продукций NVIDIA. Предыдущая GT200-архитектура существенно изменена. Значительно улучшены как программное обеспечении, так и аппаратная часть.



Компоненты Fermi: ***Графич. процессорн. кластеры*** (4 GPC), потоковые мультипроцессоры (SM), контроллеры памяти.

Первая версия Fermi- GPU включала $3*10^9$ транзисторов и 512 ядер (512 SP). Это в 2 раза больше, чем у GT200.

Последняя версия содержит 512 SP в 16 SMs (32 SP внутри каждого), или **4 GPC** (по 4 SMs).

Специальное ПО *назначает*, *поддерживает* и *контролирует* планирование блоков потоков во всех SM.

б) GPC В ФЕРМИ

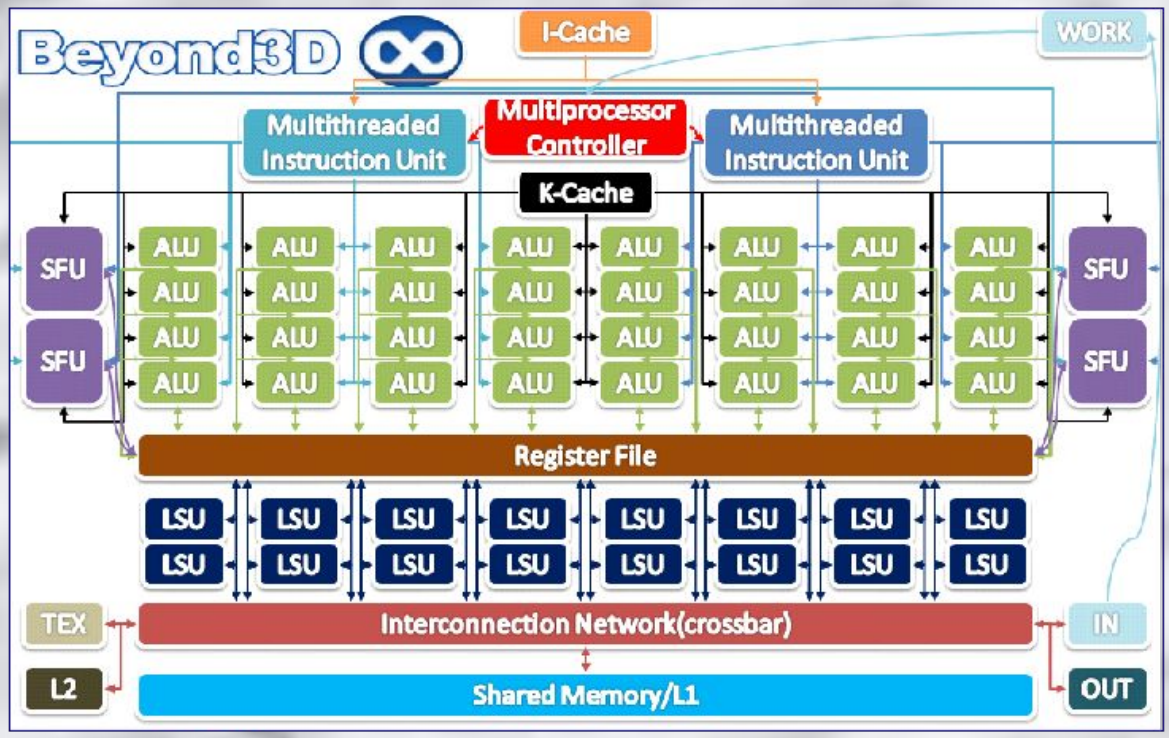
Каждый SP имеет АЛУ и блок операций с плав. точкой (FPU), *оба конвейерные*. FPU выдает результаты за 2 такта.

Каждый SM имеет 64KB SRAM (у GT200 – 16KB) как L1-кэш (16 KB для Shared и 48KB для кэш *или обратно*).

В отличие от G80 и GT200, L1 *подобен кэш* CPU и *пригоден не только для чтения*. L2-кэш 768KB подключен ко всем SM как резервные копии L1. Оба кэша (L1 и L2) поддерживают обратную запись и запись через глобальную память.

Наличие L2-кэш и деление глобальной памяти на ряд банков уменьшает число конфликтов.

в) АРХИТЕКТУРА SM



В одном SM:

– 32 *конвейерных* АЛУ. Каждый АЛУ – *до* 48 *потоков*, имеет доступ к 1024 32-бит. РОНам (128 КВ на SM).

– Распределение РОН между потоками – *автоматическое* (до 63 RG на поток). ***Регистры – не вполне адресные***.

– 4 блока *специального назначения* (SFU);

– 16 блоков *загрузки/хранения* (LSU);

– 2 блока *управления командами* (multi threaded instruction unit). В каждом из них – *планировщик задач* и *кэш инструкций*;

– 64КВ SRAM (48/16КВ – делится между L1-кэш и Shared memory);

– 64КБ read-only *константный/равномерный* кэш;

– 4 *текстурных блоков поддержки* – 12КВ *текстурный* кэш.

– Контроллер мультипроцессора – *аккумулирует* и *упаковывает* задачи в ***варп***, *распределяет* ***варпы*** между SP, *выделяет* RG и Shared потокам в ***варпе***. После чего *инициирует выполнение* ***варпа***. По *завершении* обработки ***варпа*** *распаковывает* результаты и *освобождает* ресурсы.

г) ВОПРОСЫ ЭФФЕКТИВНОГО ИСПОЛЬЗОВАНИЯ АППАРАТНЫХ РЕСУРСОВ GPU

Основной целью **GPGPU** является использование GPU для ускорения неграфических приложений. Теперь GPU – это массивно-параллельный потоковый процессор с поддержкой вычислений *с плавающей запятой* с 32- или 64-битной точностью. Он имеет гибкую модель программирования с такими системами, как CUDA, OpenCL и т.д., обладает высокой пропускной способностью памяти.

Но не следует забывать, что GPU *были изначально ориентированы на выполнение графических операций*, т.е. на параллельную обработку множества независимых пикселов изображения при минимальном вмешательстве CPU. Поэтому не для всех приложений будет получен выигрыш в производительности при их реализации на GPGPU. Идеальное применение GPGPU – ***обработка однородных массивов независимых данных***.

Оптимизация параллельного кода GPU не является тривиальной задачей. Есть несколько общих стратегий оптимизации эффективного использования аппаратных ресурсов. Детали ***аппаратно-зависимых*** стратегий меняются с каждой новой версией архитектуры. ***Программист должен учитывать детали архитектуры и потоковой модели для того, чтобы использовать эти стратегии для достижения хорошей производительности.***

Размещение в памяти, размеры и форма блоков обрабатываемых данных, конфигурация иерархии кэш-памяти и модель доступа нитей параллельных вычислений к глобальной памяти,

– в понимании взаимодействия всех этих факторов кроется ключ к повышению производительности GPGPU.

ТЕМУ

– ПРОЦЕССОР CELL –

ПРОРАБОТАТЬ САМОСТОЯТЕЛЬНО