Архитектура и программирование потоковых многоядерных процессоров для научных расчётов

Лекция 1. Введение. Общий обзор

Кому адресован курс

- **Студенты старших курсов**
  - Полный базовый физико-математический курс
  - Владение методами вычислительной математики
  - Работа над дипломным проектом
  - Потребность в большом объёме вычислений
- **Студенты имеющие практический опыт программирования**
  - Знание языка программирования C
  - Знание основ архитектуры компьютеров
  - Владение средствами отладки и профилирования программ
  - Базовые знания операционной системы Linux

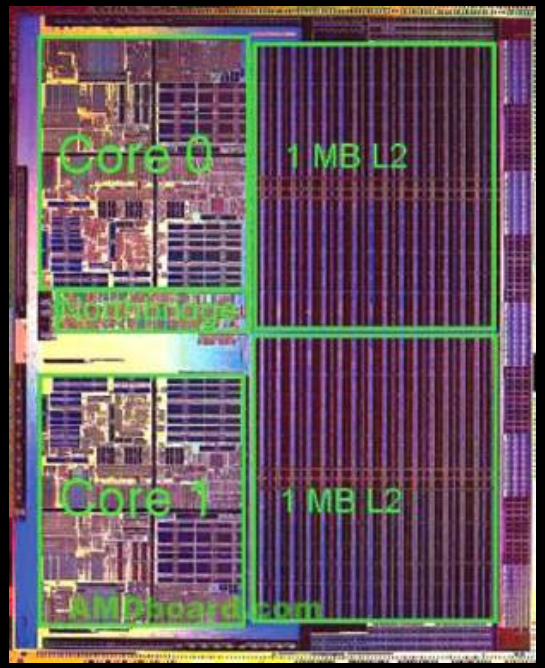
Краткое содержание курса

- **Программная модель CUDA**
  - CUDA – Compute Unified Design Architecture
- **Аппаратная реализация**
  - Реализация на потоковых графических процессорах NVIDIA
  - История развития потоковых графических процессоров
- **Библиотеки CuBLAS и CuFFT**
  - CuBLAS – CUDA Basic Linear Algebra Subpr. (основан на BLAS)
  - CuFFT – CUDA Fast Fourier Transform (основан на FFTW)
- **API**
  - Расширения языка C
  - NVCC компилятор и PTX виртуальная машина
- **Организация потока данных**
  - Различные виды памяти
  - Когерентный доступ к памяти
- **Примеры программирования**

Классификация вычислительных архитектур

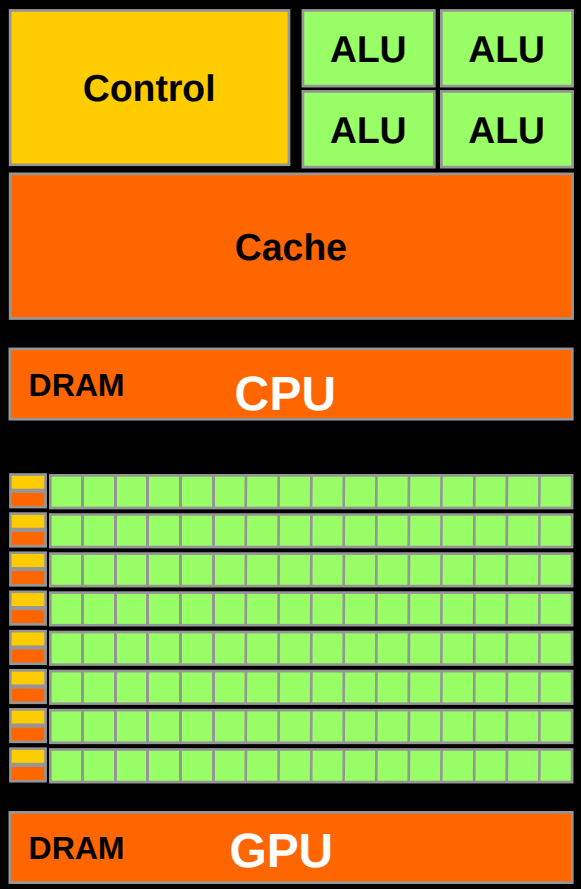
- **Предложена М. Флином (M. Flynn) в 1966 году**
  - Основана на понятии потока (stream), как последовательности комманд или данных, обрабатываемых процессором
- SISD – “Single Instruction / Single Data” Stream
  - Скалярная обработка – каждая комманда на ленте сопровождается данными для этой комманды
  - Наличие конвейера не меняет сути
- SIMD – “Single Instruction / Multiple Data” Stream
  - Векторная обработка - каждая комманда на ленте сопровождается множеством данных, над которыми эта команда выполняется
- MISD – “Multiple Instruction / Single Data” Stream
  - ??? Конвейерные системы
- MIMD – “Multiple Instruction / Multiple Data” Stream
  - Компьютеры, способные выполнять одновременно множество комманд над множеством данных ??? Распределённые GRID-структуры

Преимущества и недостатки традиционной SISD архитектуры



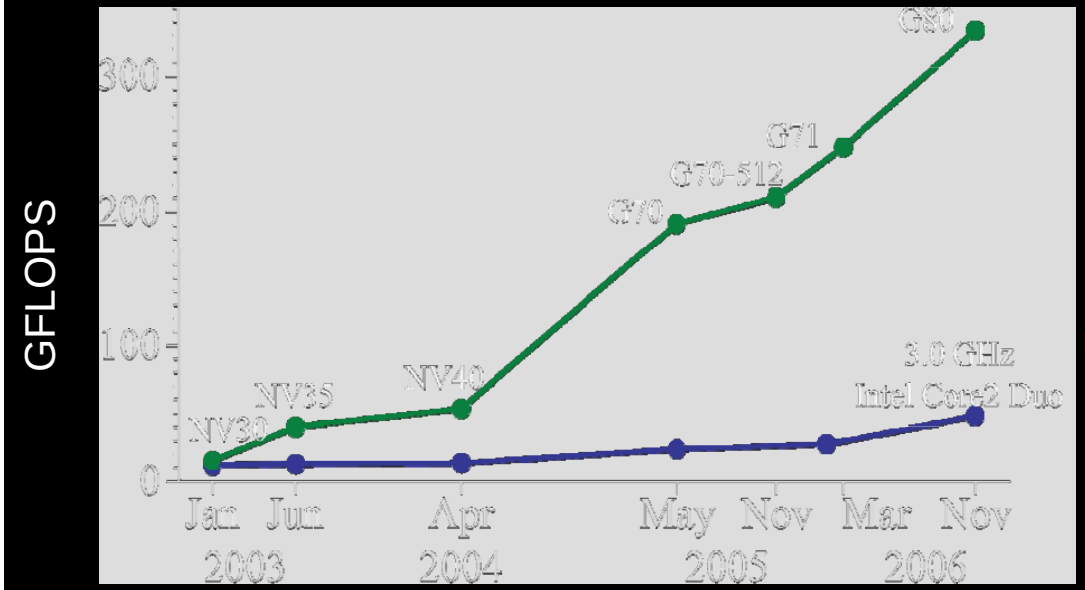
- Фото двух-ядерного кристалла AMD
- **Good: Простота ‘традиционной универсальной’ программной модели**
  - Последовательное исполнение инструкций
  - Общая память данных
  - Применение отработанных оптимизирующих компиляторов
- **Bad: Производительность ограничена скоростью доступа к памяти**
  - Применение конвейеров
  - Кэширование обращений к памяти
  - Обрабатывать другие исполняемые потоки пока этот ждёт данных
  - Блочное чтение памяти

Альтернатива классическому подходу



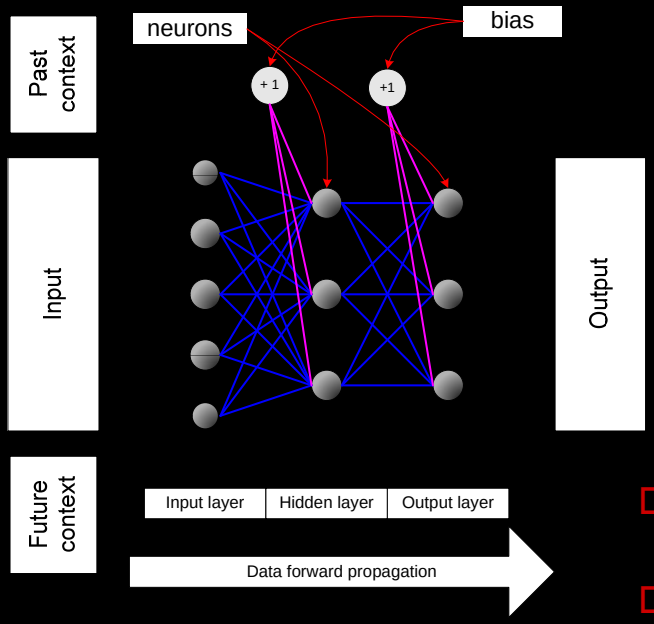
- Строить систему расчитанную на **cache-miss** а не на **cache-hit**
- Вместо кэша и сложных АЛУ отдадим эту площадь кристала под упрощённые АЛУ, имеющие общую память на кристалле
- Good: Латентность доступа к памяти устанавливает **однократную систематическую задержку** между потоком исходных данных и результатов
- Bad: **Программист обязан** тщательно рассчитывать **размещение алгоритма** на исполняющих элементах / продумывать **стратегии доступа к памяти**

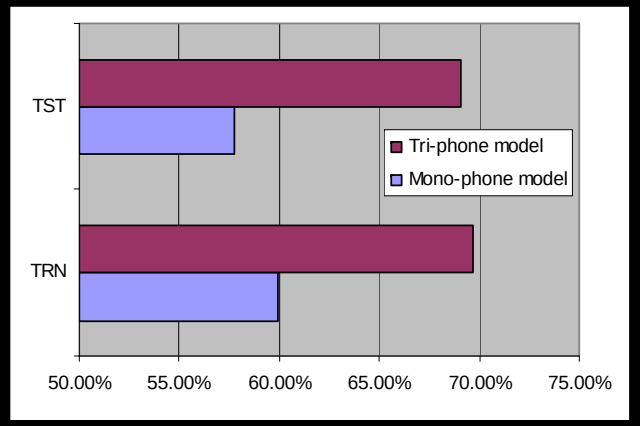
Есть ли выигрыш? – ДА!



- Где он наблюдается?
- Где высоко количество операций на одно чтение исходных данны
- Где одна программа исполняется для массивов данных (**SIMD**)
- Аппаратная реализация традиционно ориентирована на обработку графических данных
  - Данные – матрицы
  - Отсутствие ветвлений
  - Каждый из результатов вычисляется из подмножества данных
  - Плавающая точка

Пример. Тренировка MLP





- Тренировка MLP акустической модели для распознавателей речи
- Объём тренировочных данных для одной эпохи ~ **100 часов речи** (8kHz)
- Сеть **150x1000x60 сигма-нейроны**
- Реализован классический алгоритм backpropagation
- **X10 ускорение вычислений по сравнению с CPU (3 часа 30 мин.)**

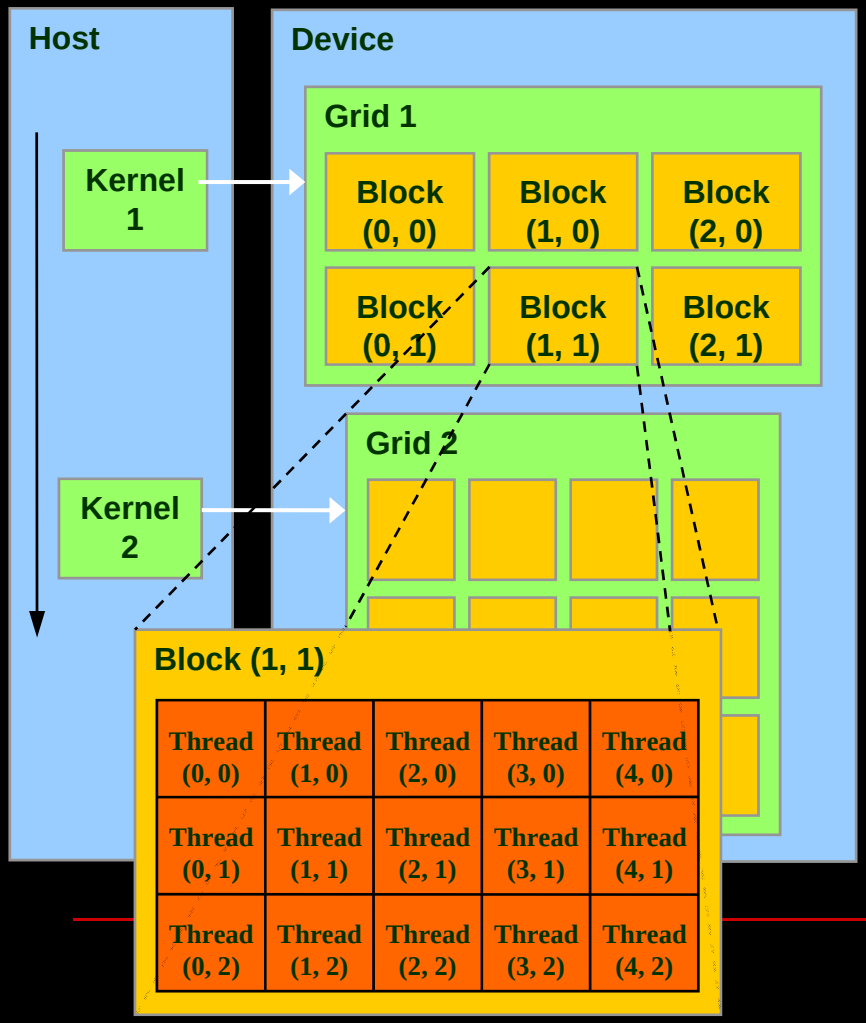
CUDA = Compute Unified Design Architecture

- Многократное ускорение в каждой конкретной задаче достигается в результате усилий программиста
- Существовали ранние попытки использования графических карт для научных рассчётов
- Проблема – алгоритмы должны были быть реализованы на специальном **“шейдерном” языке** (shade language)
- Shade Language разработан и подходит для графики
- Для облегчения работы с алгоритмами общего назначения (**GPGPU=General Purpose computing on Graphical Processing Units**) компания NVIDIA выдвинула инициативу создания **аппаратно/программной архитектуры общего назначения**

CUDA общие положения

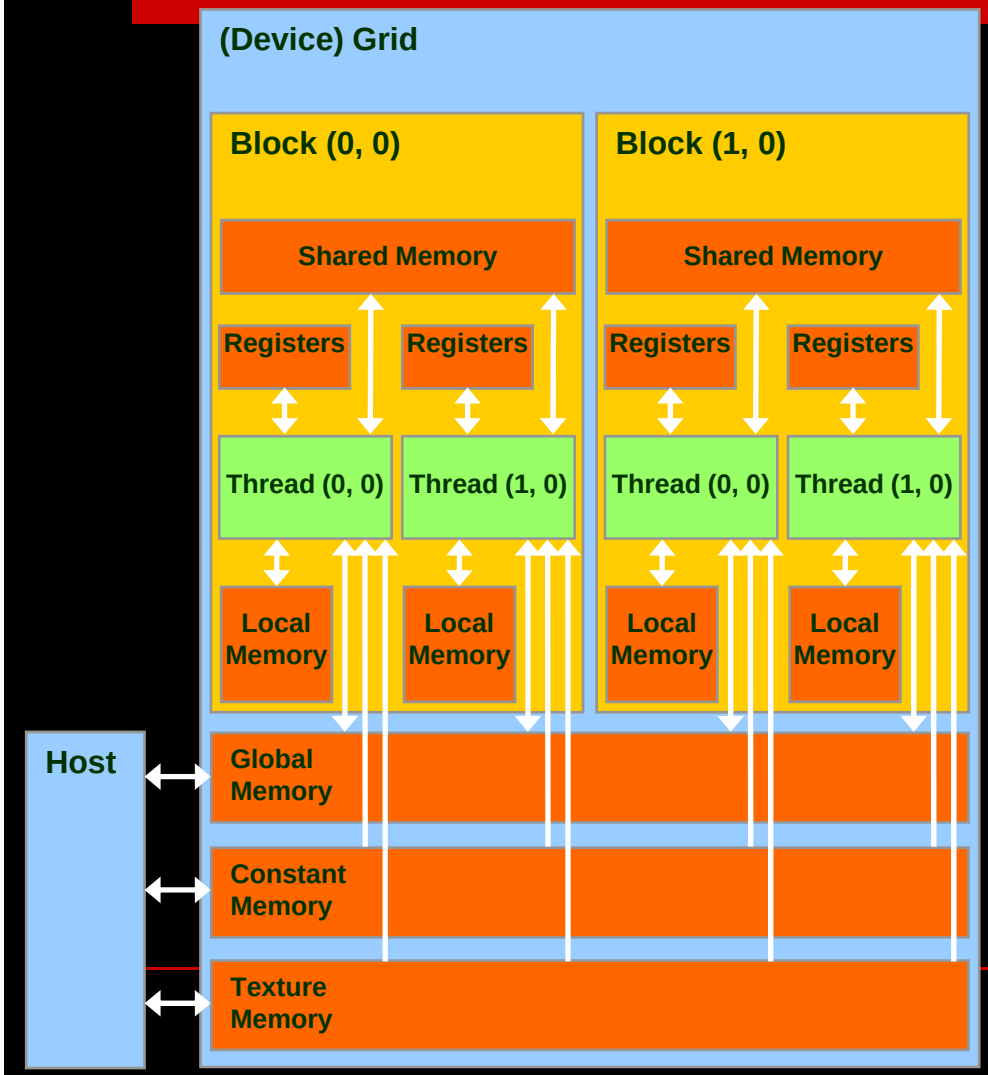
- GPU – **сопроцессор** для CPU (**хоста**)
- У GPU есть собственная память (**device memory**)
- GPU способен одновременно обрабатывать множество процессов (**threads**) данных одним и тем же алгоритмом
- Для осуществления расчётов при помощи GPU хост должен осуществить запуск вычислительного ядра (**kernel**), который определяет конфигурацию GPU в вычислениях и способ получения результатов (алгоритм)
- Процессы GPU (в отличие от CPU) очень просты и многочисленны (**~ 1000 для полной загрузки GPU**)

Вычислительная конфигурация GPU



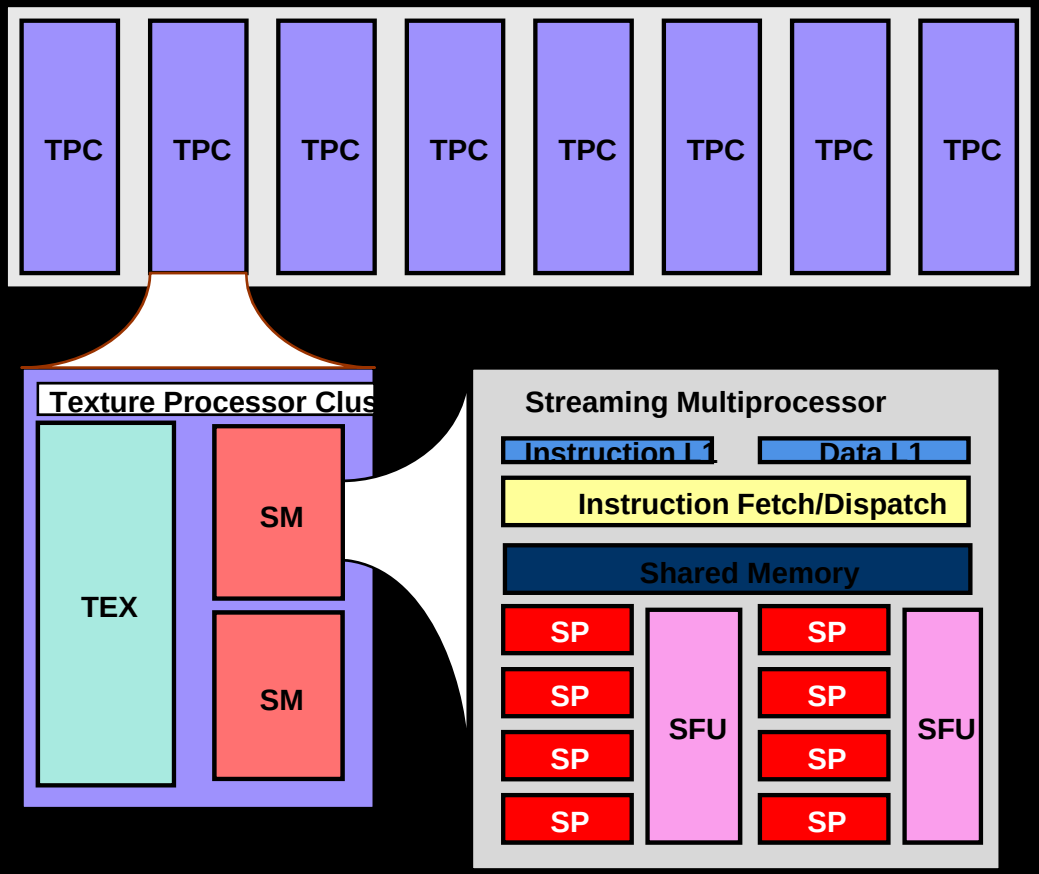
- Процессы объединяются в блоки (**blocks**), внутри которых они имеют общую память (**shared memory**) и синхронное исполнение
- Блоки объединяются в сетки (**grids**)
  - Нет возможности предсказать очерёдность запуска блоков в сетки
  - Между блоками нет и не может быть (см. выше) общей памяти

Модель памяти GPU



- GPU может читать
  - Constant Memory
  - Texture Memory
- GPU может читать/писать
  - Global Memory
- Каждый из процессов ч/п
  - Local Memory
  - Registers
- Каждый из процессов внутри блока ч/п
  - Shared memory
  - **Gather/Scatter MA pattern**
- Хост имеет возможность читать/писать:
  - Global Memory
  - Constant Memory
  - Texture Memory

Аппаратная архитектура GPU



- SPA - Streaming Processor Array
- TPC - Texture Processor Cluster
- SM - Streaming Multiprocessor
  - Multi-threaded processor core
  - Fundamental processing unit for CUDA thread block
- SP - Streaming Processor
  - Scalar ALU for a single CUDA thread

Пример.
Параметры GPU GeForce 8800GTX

- Programming model:
  - Максимум 512 процессов в блоке (512x512x64)
  - Максимальные размеры сетки (65535x65535)
  - Максимальный размер вычислительного ядра ~ 2 млн. инструкций

- HW architecture:
  - 16 мультипроцессоров (Streaming Multiprocessors - SM) / 128 потоковых процессоров (Streaming Processors)
  - Вплоть до 8 блоков исполняются одновременно на каждом SM
  - Вплоть до 24 варпов (**warps***) исполняются одновременно на каждом SM
  - Вплоть до 768 процессов исполняются одновременно на каждом SM
  - Количество регистров на SM - 8192
  - 16k общей памяти на SM / 16 банков
  - 64k памяти констант (кэшируется по 8k на SM)
  - 32-разрядная IEEE float арифметика

- *warp = часть блока, исполняемая на SM в SIMD виде

Итоги лекции

- **В результате лекции студенды должны приобрести:**
  - Понимание целей, организации и общей программы данного курса
  - Понимание общей схемы организации вычислений GPGPU и её отличий от классической SISD схемы
  - Понимание назначения CUDA и наиболее общих понятий этой архитектуры