CUDA architecture

Massimiliano Piscozzi

Università degli Studi di Milano

June 2008

Outline

Outline

What's CUDA?

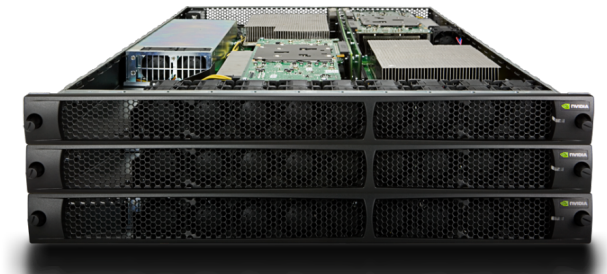
CUDA = Compute Unified Device Architecture

- GPGPU technology
  1. Hardware technology
  2. Software technology

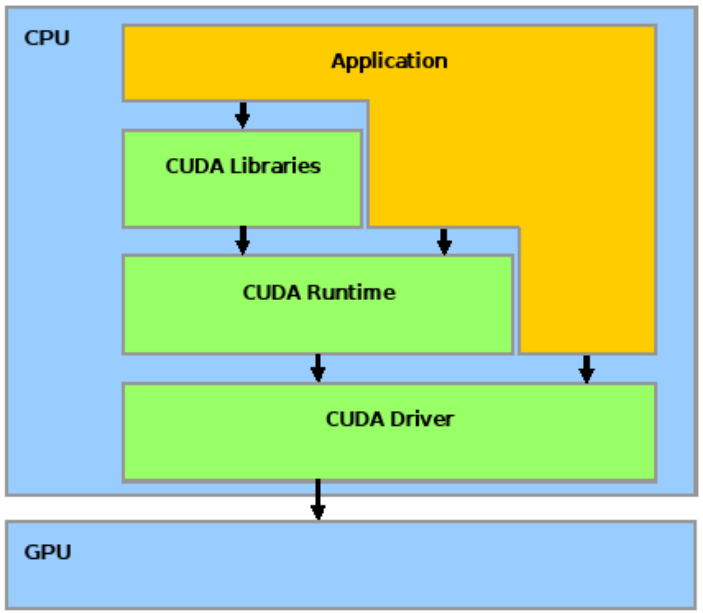
CUDA: the hardware side

- GeForce / Quadro (8-series) graphics card
  - Desktops, notebooks

- Tesla (C/D/S-870) high performance computing (HPC) solution
  - Workstations, servers, clusters



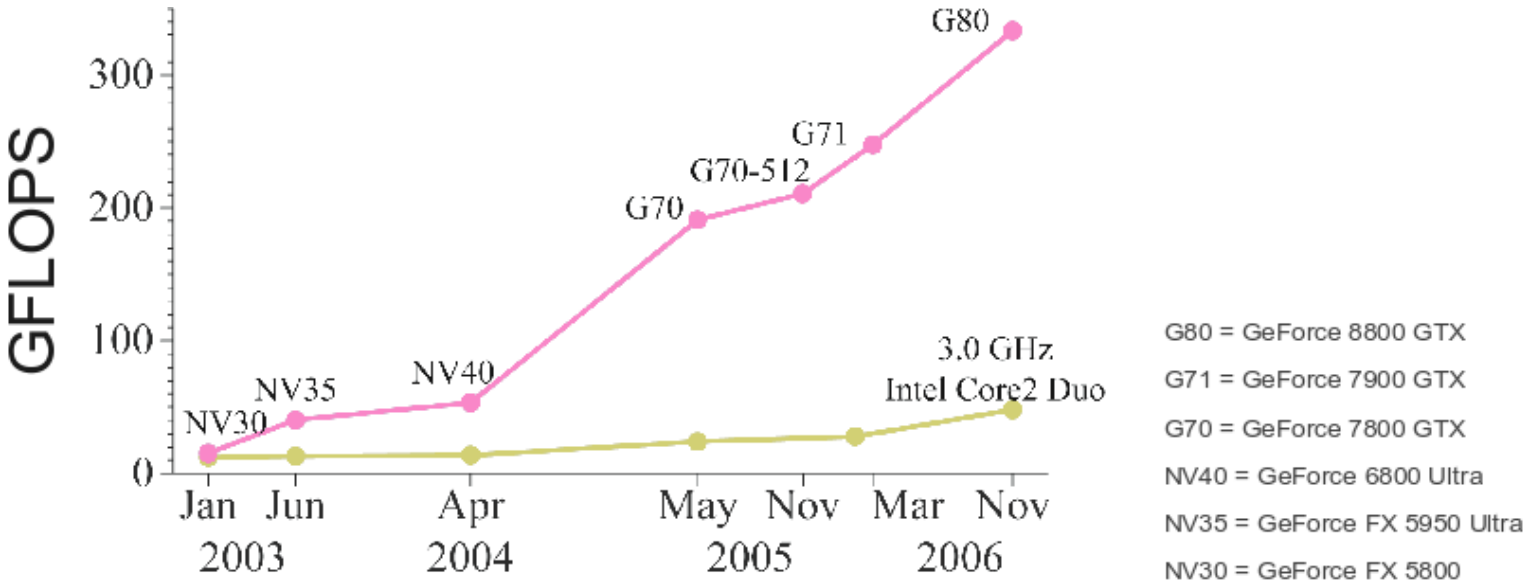
CUDA: the software side



- CUDA software stack
  1. Hardware layer
  2. Application Programming Interface (API)
     - C language extension
  3. Higher level mathematical/programming libraries
     - CUBLAS, CUFTT, CUDPP, ...

Outline

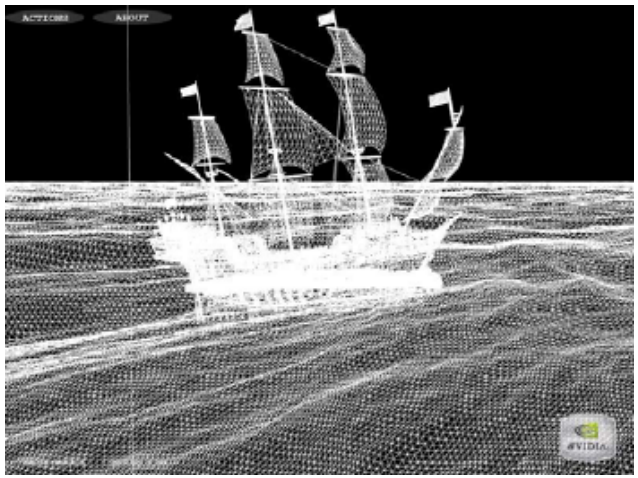
CPU vs GPU Performance



1. Why GPUs are so fast?
2. Why can't we replace CPUs with GPUs?
   - When can we use the GPU instead of the CPU?

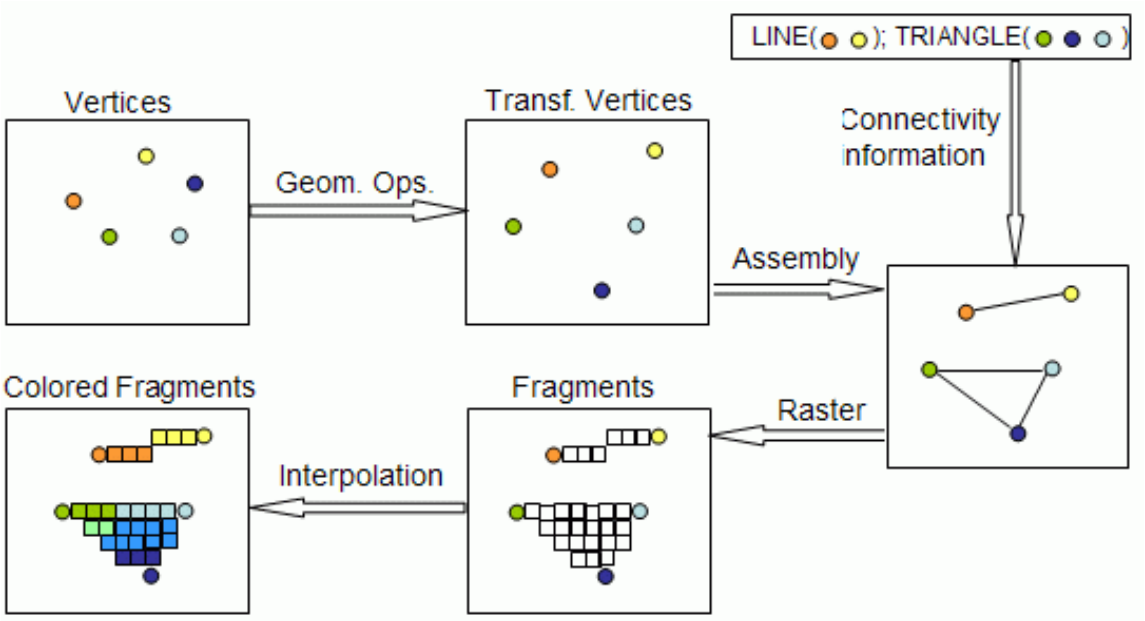
Real-time rendering

- Graphics hardware enables real-time rendering



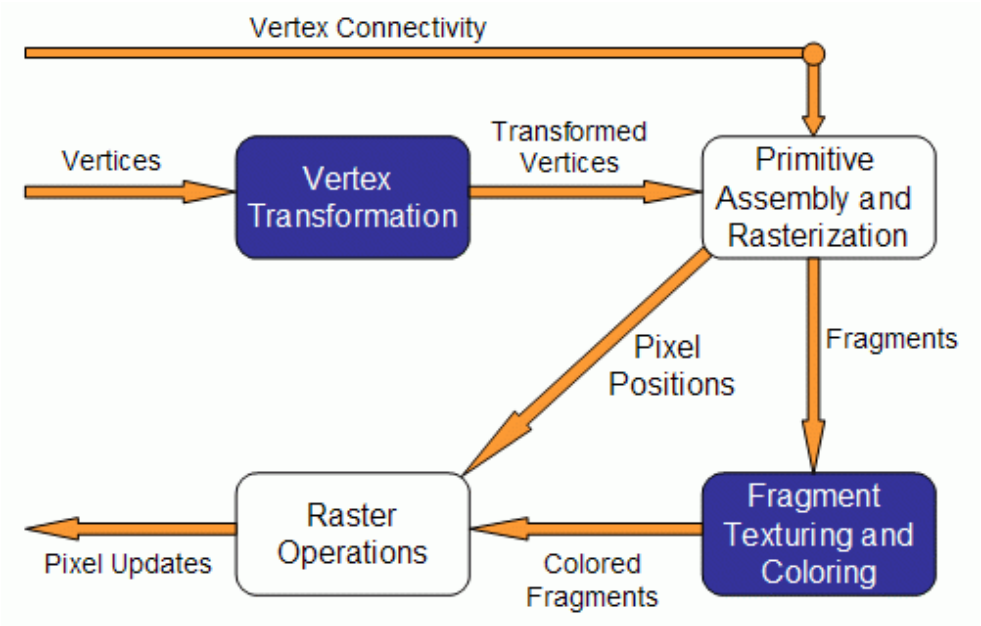
- Main goal
  - Transform a collection of **3D primitives** (triangles, lines, points) ...
  - ... into an **array of pixels**

Logical graphics pipeline



- Very close to the graphics libraries (OpenGL, DirectX) pipeline
  - Vertex transformation $\approx$ perspective projection
  - Fragment operations $\approx$ shaders evaluation

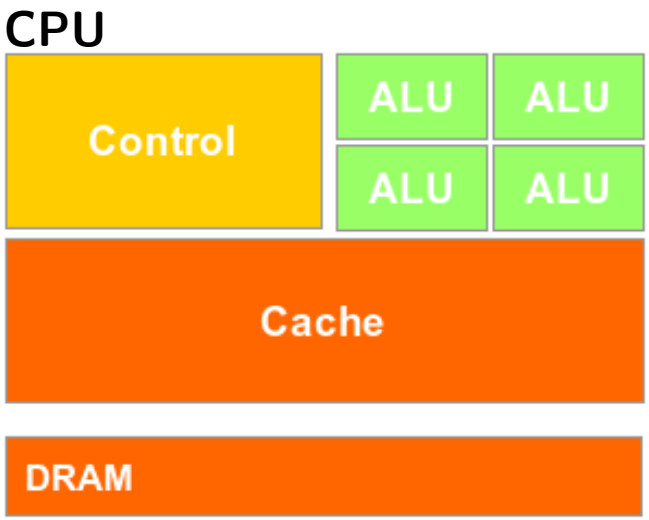
Semi-fixed graphics pipeline

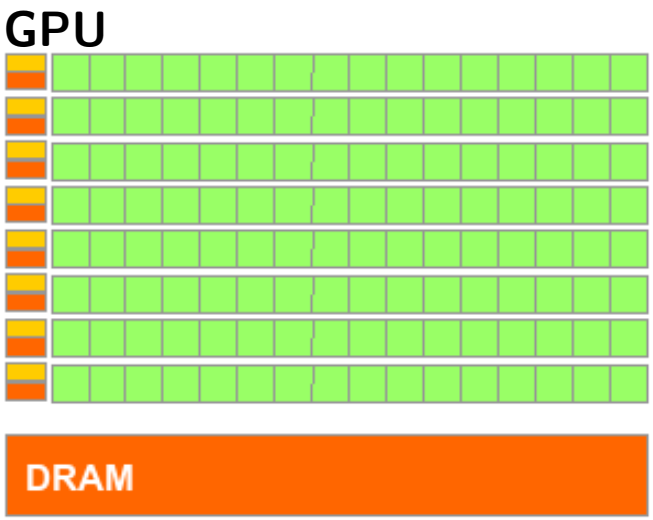


- Two programmable units
  1. Vertex processor
  2. Fragment processor
- The raster unit is fixed

CPU vs GPU architecture

- **GPUs** are specialized for **highly parallel**, **compute-intensive** computation
  1. Same computation = lower requirement for flow control
  2. Arithmetic intensity (memory access latency hiding) *vs* big data caches

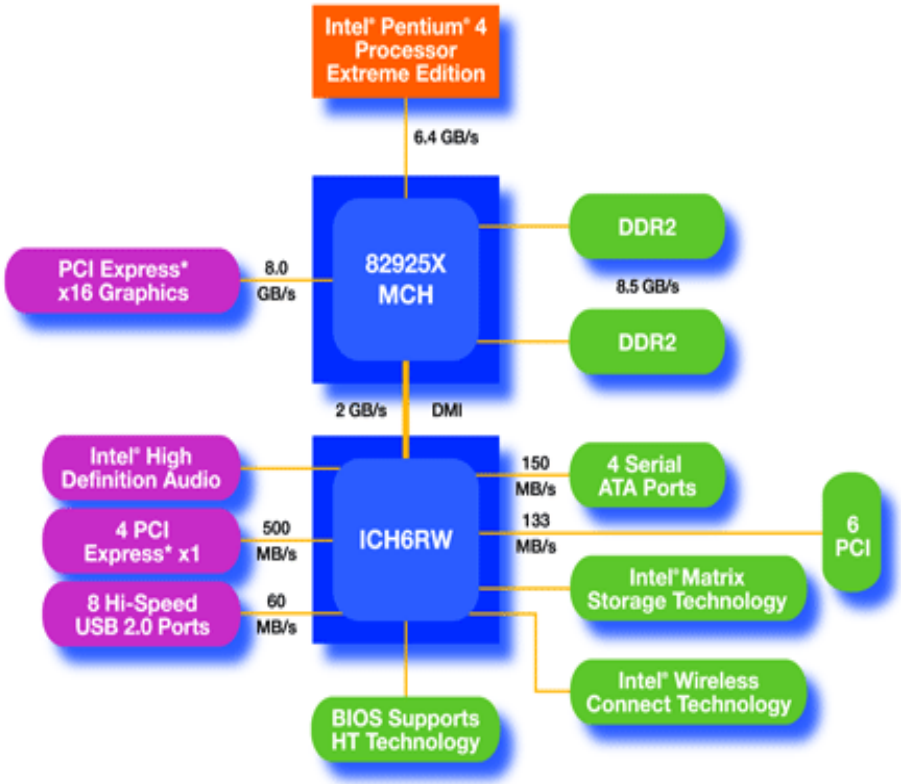




SIMD (Single Instruction Multiple Data) architecture

Outline

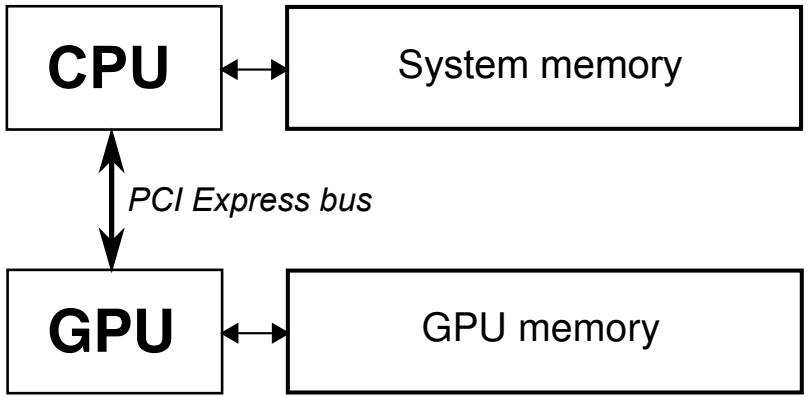
Computer organization



- The **graphics card** can be used as a **coprocessor**

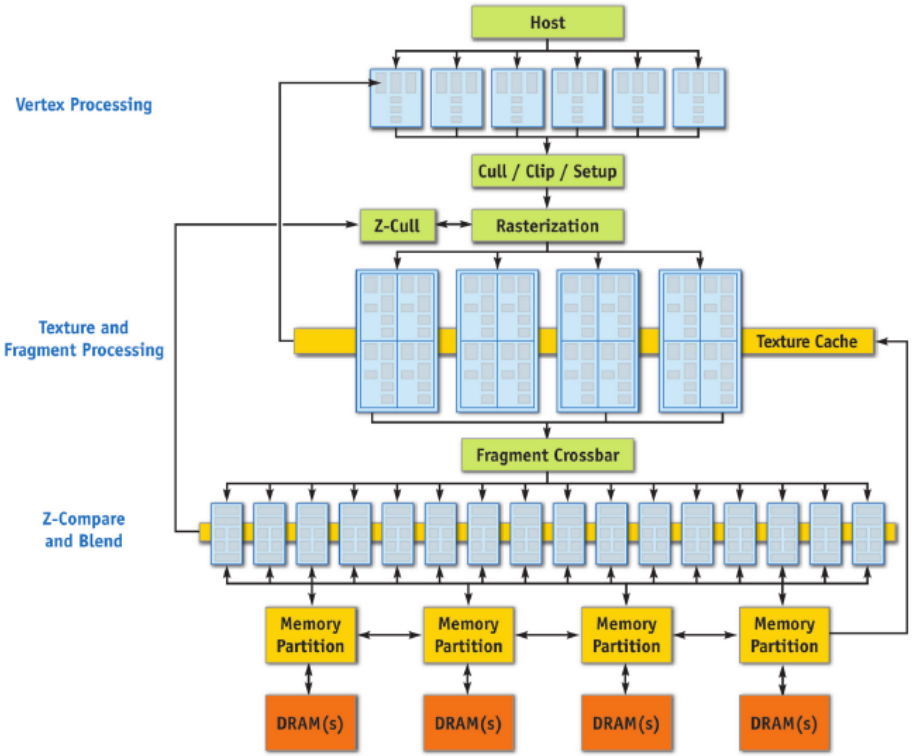
CPU-GPU cooperation

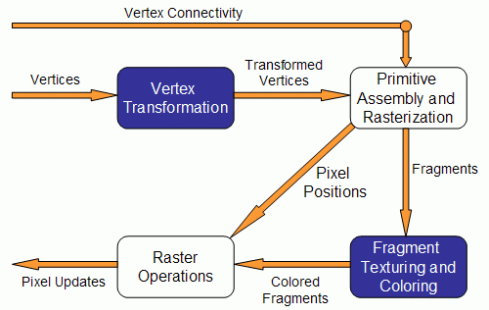
- CPU-GPU communication via **PCI Express** bus
- CPU and GPU each have their own memory spaces



- **CPU** is the **host**, **GPU** is the **device**
  1. CPU sends data to the GPU
  2. GPU processes data
  3. CPU copies data back from the GPU

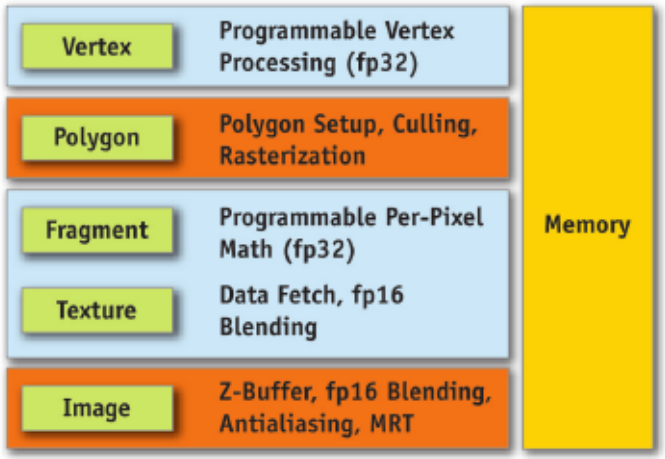
GeForce 6-series architecture

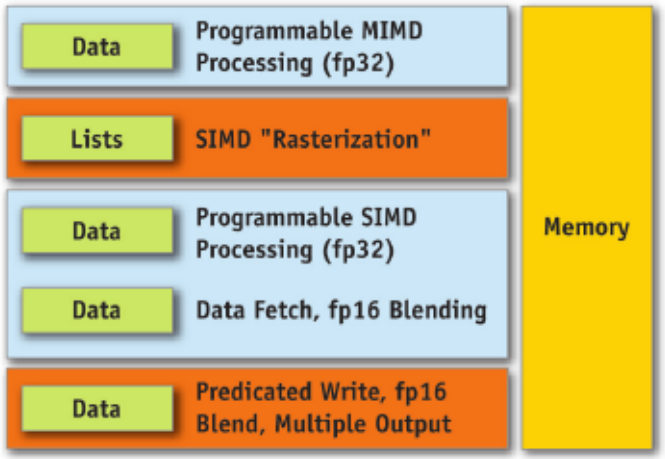




- Specialized units (SIMD architecture)
  1. Vertex processors
  2. Fragment processors
- GPU memory interface up to 35 GB/s

Legacy GPGPU approach

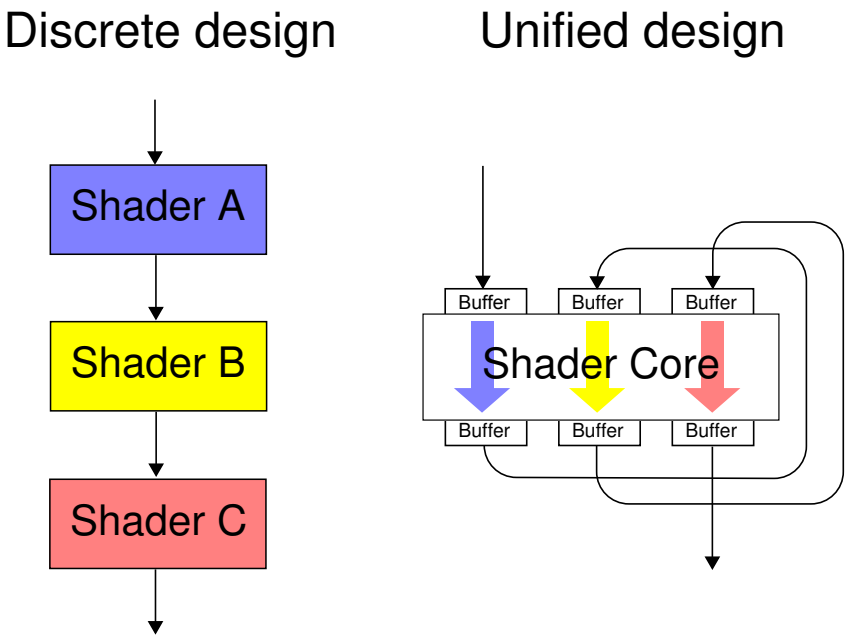




- General purpose applications must be mapped on the graphical pipeline
  1. GPGPU algorithms = multi-pass rendering
  2. Algorithms written using shading languages (GLSL, CG, ...)

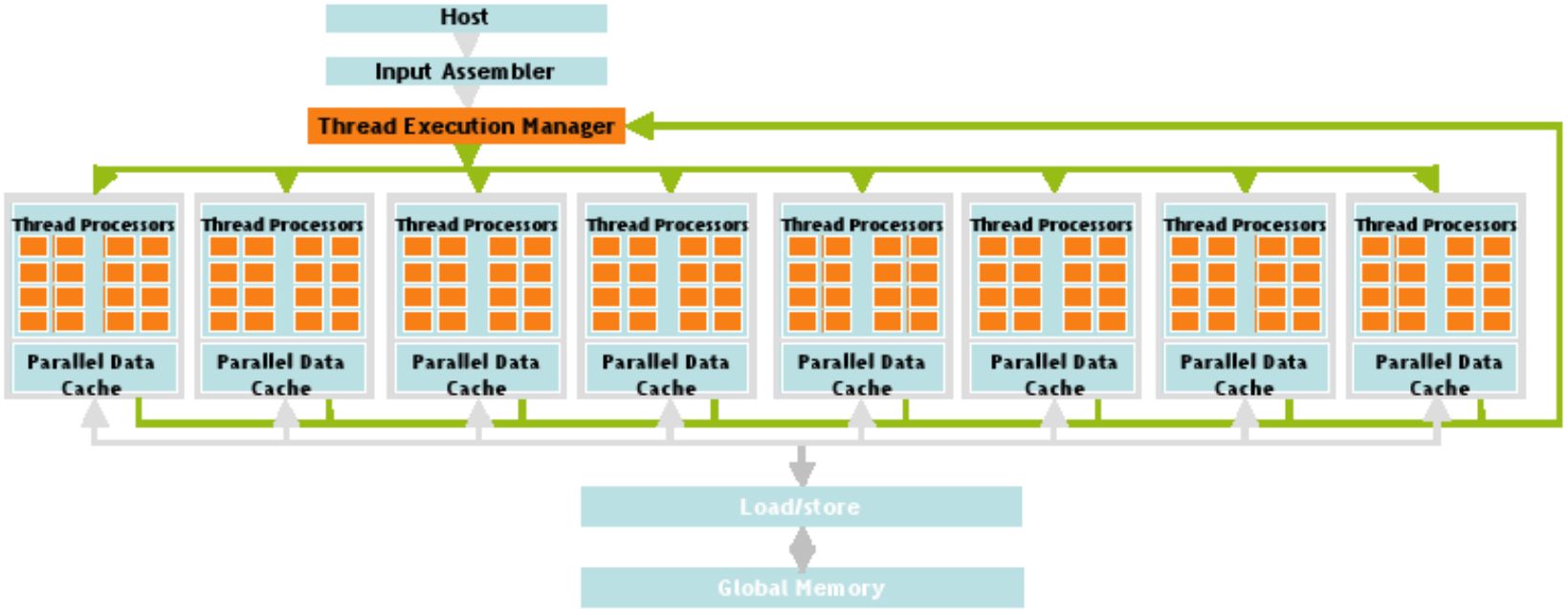
The unified architecture

- *Specialized units* vs *general purpose processors*



- **Unified design**
  - Better workload balancing
  - (More) independent of the logical pipeline

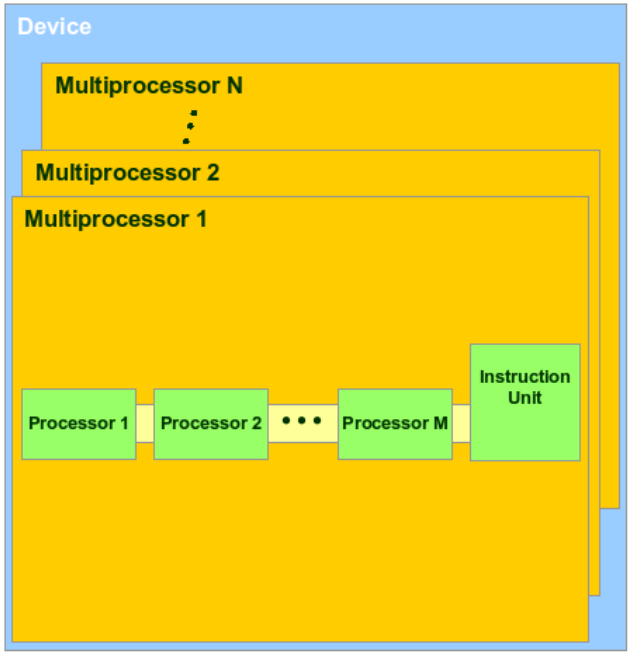
GeForce 8-series architecture



- General purpose multiprocessors (SIMD architecture)
  - No Vertex / Fragment specialization
- GPU memory interface up to 90 GB/s

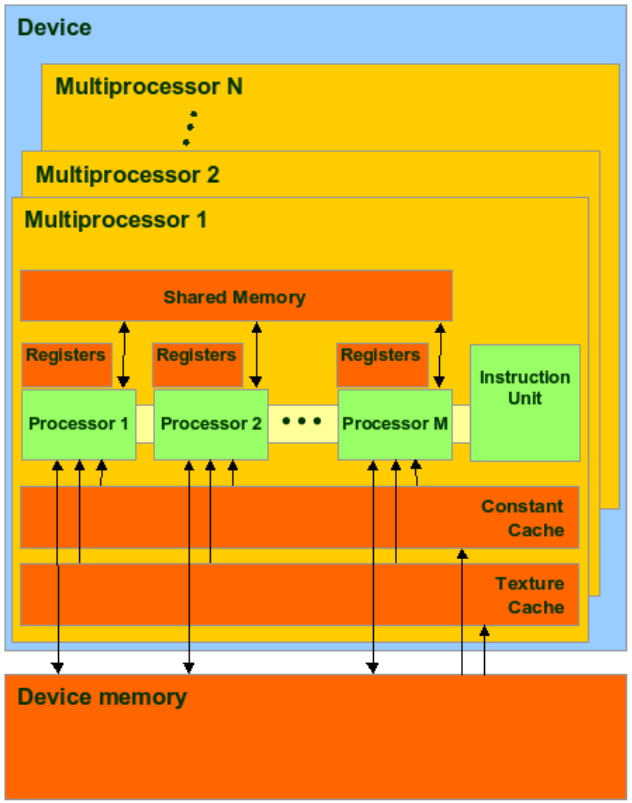
Multiprocessors

- Up to 16 multiprocessors per card
- Each multiprocessor can execute a *warp* of threads
  - Lightweight threads
  - *Warp* = 32 threads
- SIMD thread execution



Memory architecture

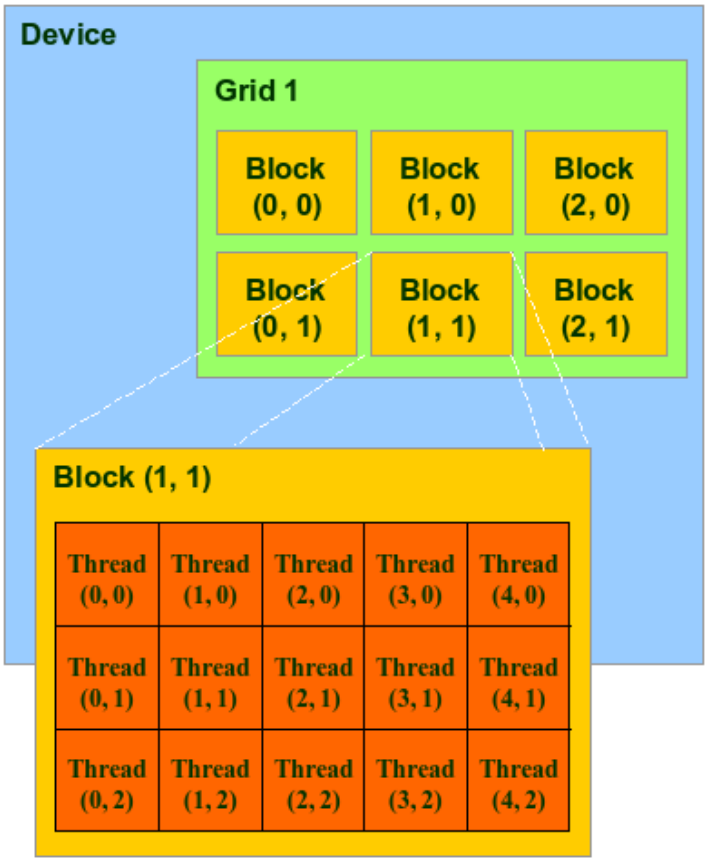
- One set of local 32-bit **registers** per-processor
- A **shared memory**, shared by all the processors
- A read-only **constant cache**, to speeds-up reads from the constant memory space
- A read-only **texture cache**,to speeds-up reads from the constant memory space



Outline

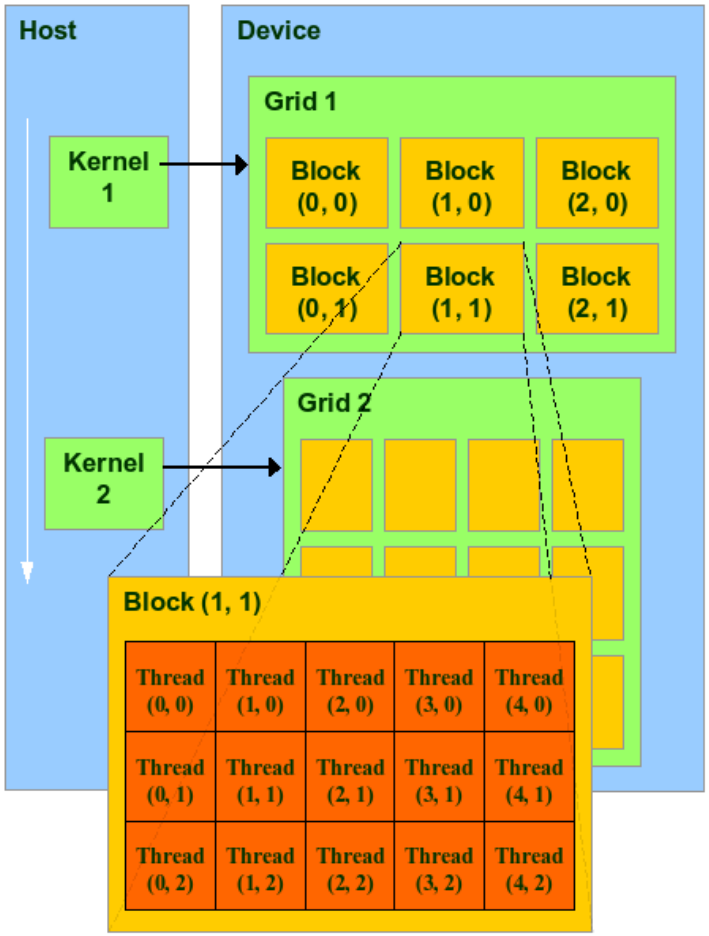
Blocks and grids

- **Block** = one-, two- or three-dimensional array of threads
- **Grid** = one-, two- or three-dimensional array of blocks

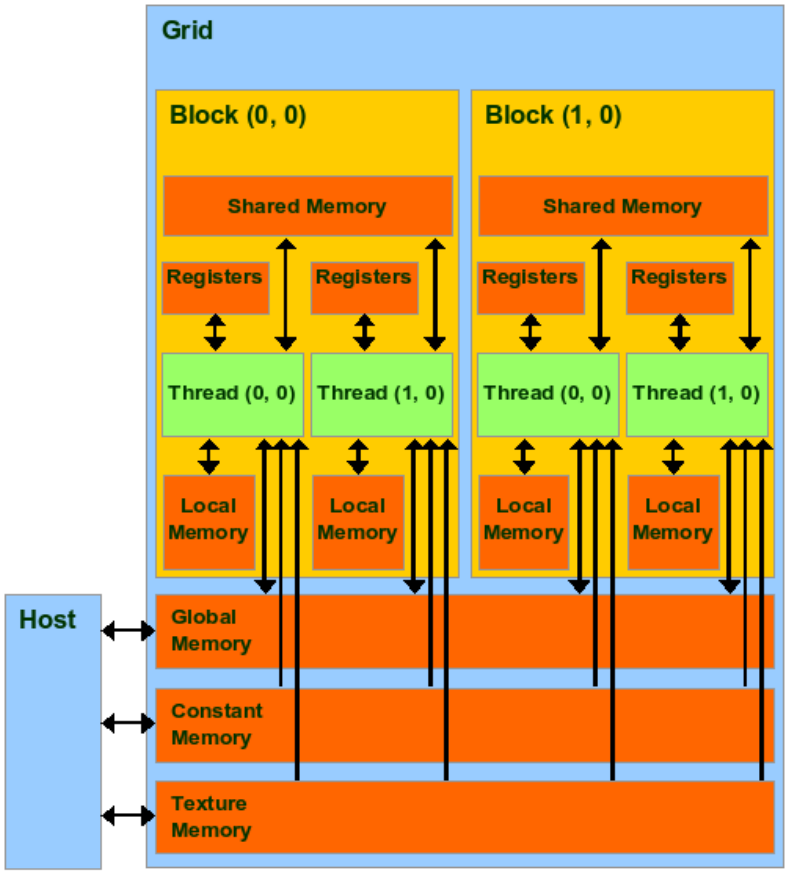


Threads batching

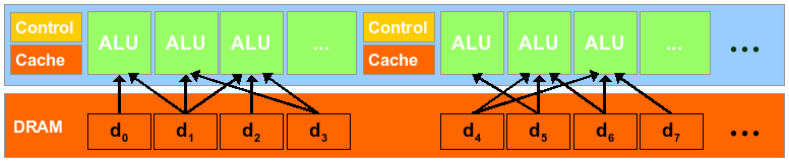
- A block is processed by only one multiprocessor
  - Each block is split into warps (consecutive IDs)
- Several blocks can be processed by the same multiprocessor concurrently
  - Registers and shared memory
- No-synchronization mechanism between blocks



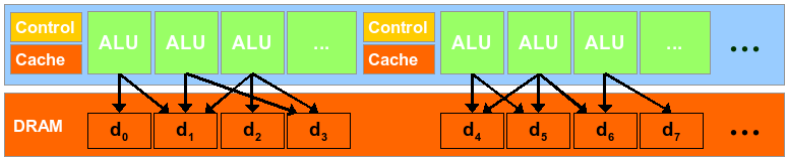
Programming model



Gather and scatter

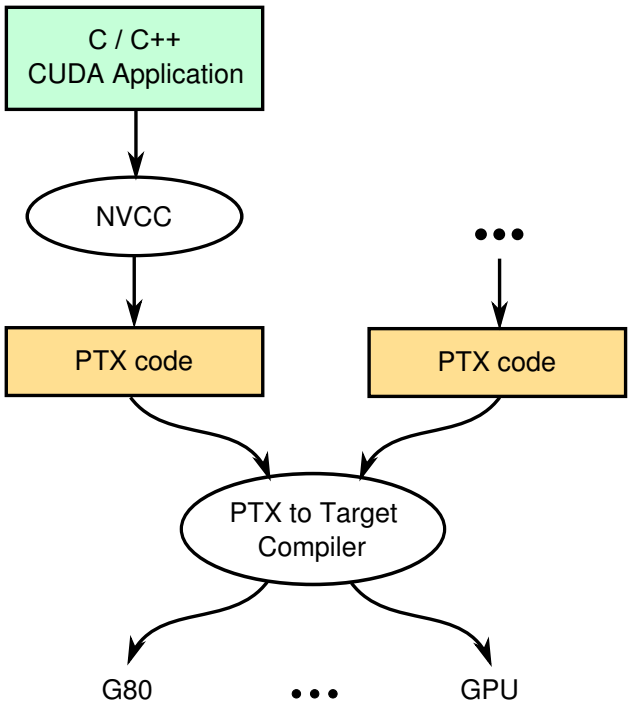


Gather



Scatter

PTX Code



1. CUDA Applications written in (extended) C language
2. **NVCC**: NVIDIA CUDA compiler based on **Open64**
3. PTX = Parallel Thread eXecution

Extended C language

- Explicit GPU memory allocation (only from CPU!)
  - cudaMalloc(...)
  - cudaFree()
- Memory copy between host and device
  - cudaMemcpy(...)
  - cudaMemcpy2D(...)

- Function execution on GPU
  - __global__ void myKernelFunction(...);
- Explicit shared memory allocation
  - __shared__ int mySharedVariable;
- Kernel launch (CPU → GPU)
  - myKernelFunc <<< gridSize, blockSize, sharedMem >>>