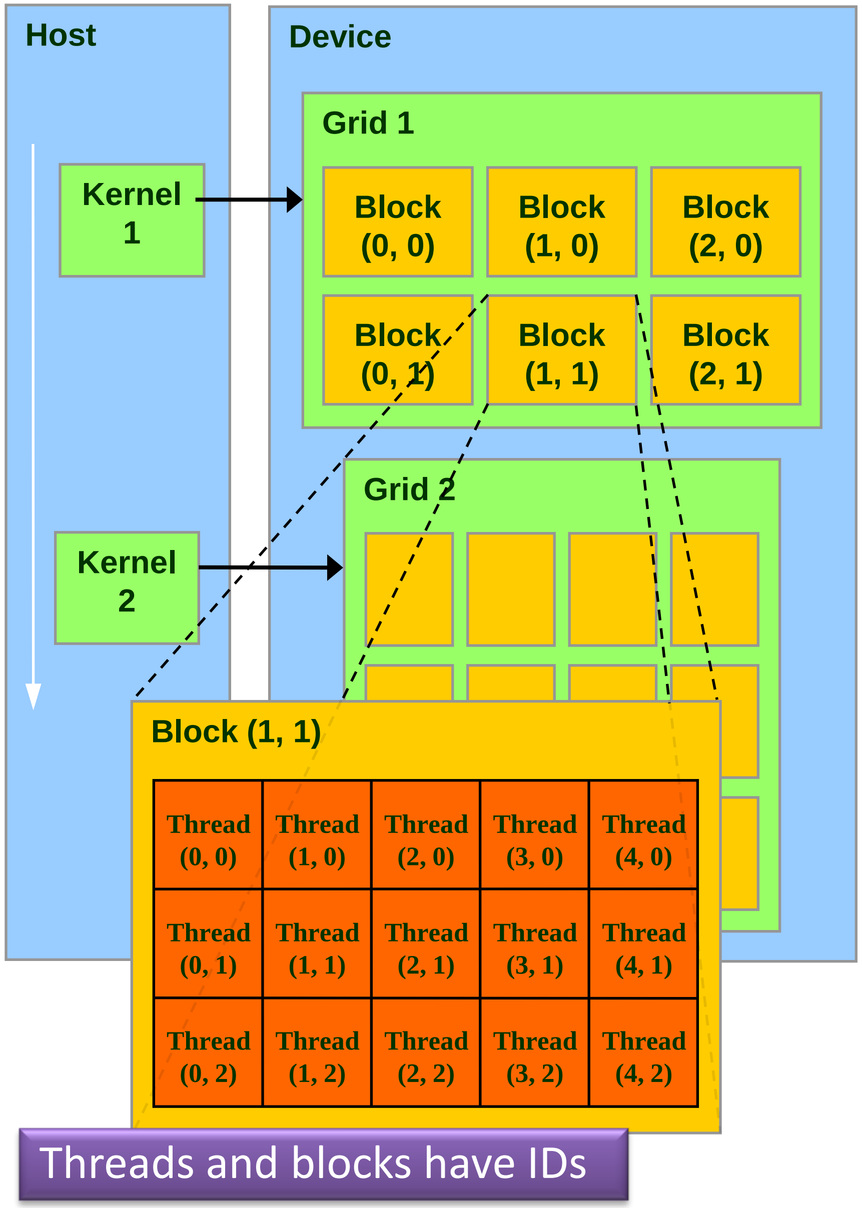
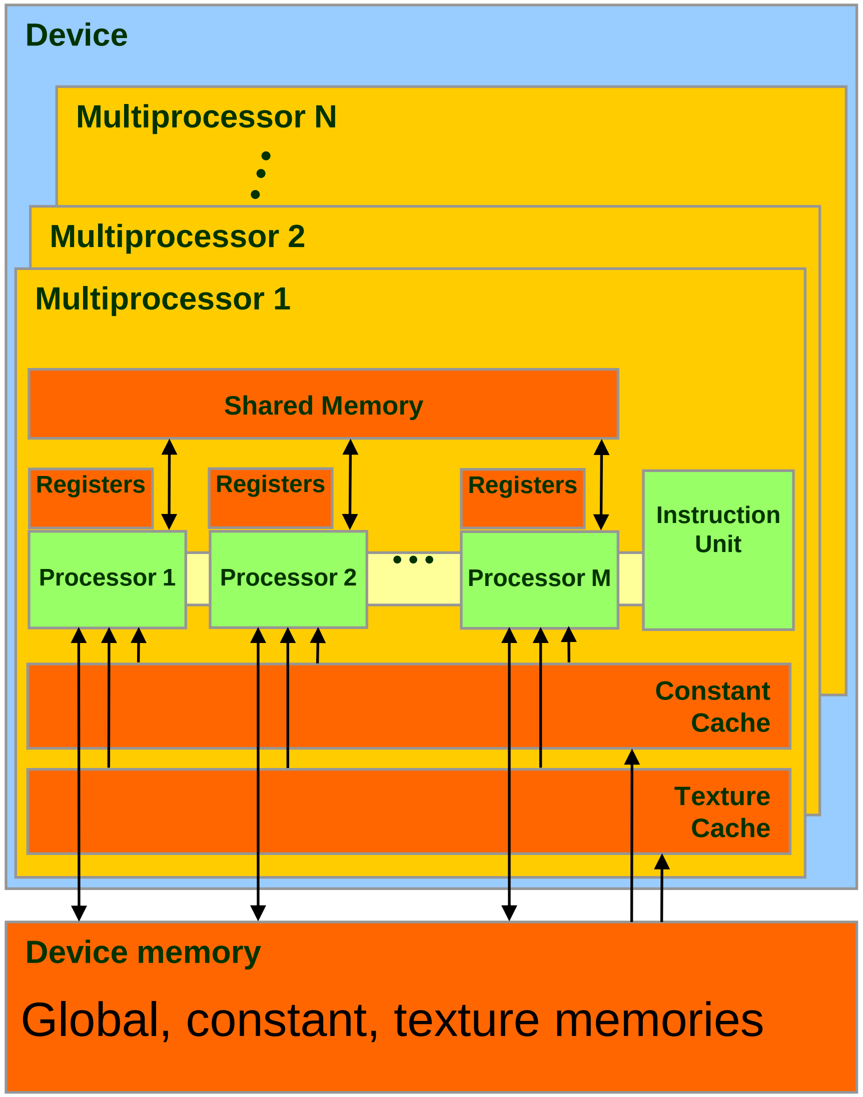
CUDA Execution Model



Typical latency for launching kernel and transfering ~5 parameters: 0.02ms on "hot wire". First call always much slower – GPU requires warm-up!

GPU(Device) is a coprocessor to the CPU(Host)
Cpu launches execution of a kernel on GPU
Kernel – function executed by each thread of the GPU. Accepts arguments.
Kernel has the following configuration parameters (specified at compile time):
Grid size =#Blocks to be launched by the kernel call.
Thread Block size = # Threads grouped in an execution unit called "block".
•1 Thread Block is executed on 1 SM
•1 SM can execute multiple thread blocks simultaneously. Max number of blocks to run sim. determined by several parameters (number of threads in block, shared memory usage, register usage) . Total max number of blocks to be run by 1 kernel is however determined by device capabilities.
•Threads are scheduled in units called warps (1 warp = 32 threads).
Total number of threads for execution = grid size*block size

CUDA Hardware Architecture



G80 Series has 16 Streaming Multiprocessors (SM), each with:

- 8 Streaming Processors (ALUs)
- 1 Instruction Unit
- 16KB Shared Memory (on chip, fast access)
- A set of 32-bit registers (8KB Register File)

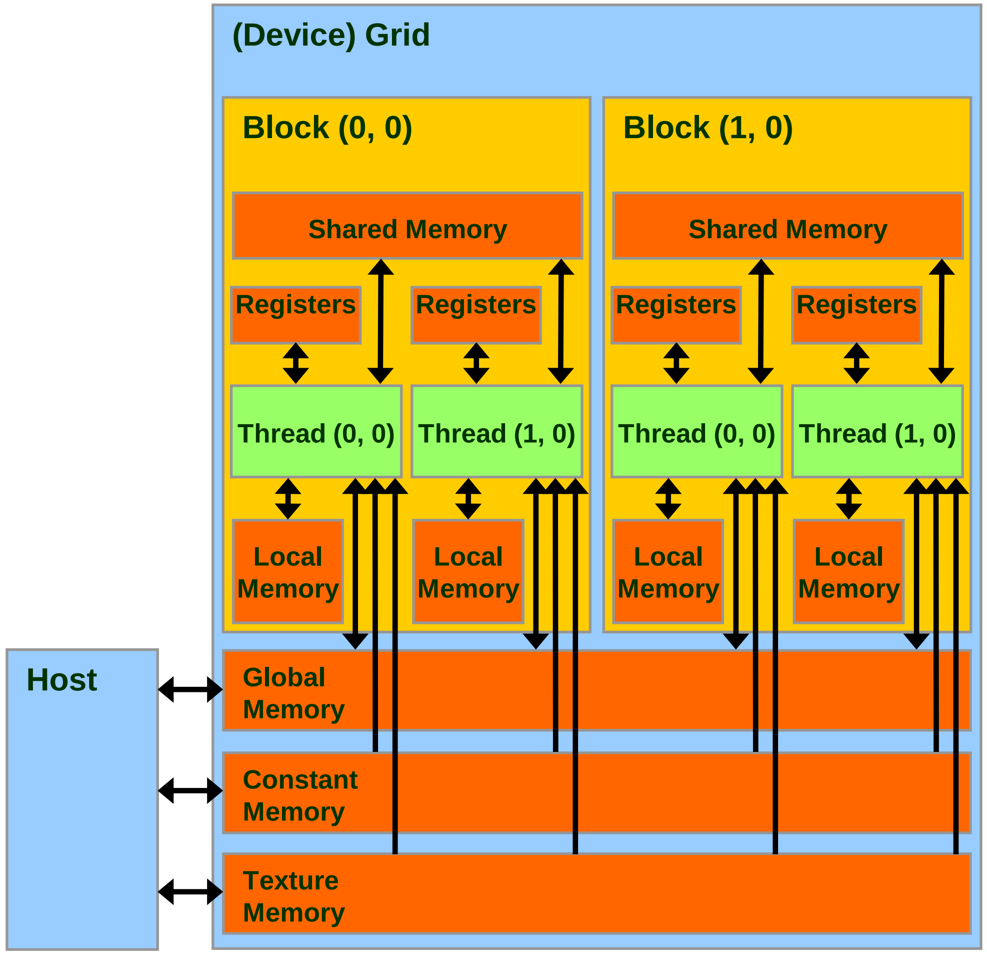
<u>GeForce 8800 Characteristics:</u>

Device 0: "GeForce 8800 Ultra“	
Total amount of global memory:	**804978688 bytes**
Total amount of constant memory:	65536 bytes
Total amount of shared memory per block:	**16384 bytes**
Total number of registers available per block:	8192 bytes
Warp size: 32 Threads	
Maximum total number of threads per block:	**512**
Maximum sizes of each dimension of a block:	512x512x64
Maximum sizes of each dimension of a grid:	65535x65535x1
Maximum memory pitch:	262144 bytes
Texture alignment:	256 bytes
Clock rate:	1512000 kilohertz

Each thread block of a grid is split into warps, each gets executed by one multiprocessor (SM). Each thread block is executed by one multiprocessor, so that the shared memory space resides in the on-chip shared memory.

A multiprocessor can execute multiple blocks concurrently: Shared memory and registers are partitioned among the threads of all concurrent blocks. So, decreasing shared memory usage (per block) and register usage (per thread) increases number of blocks that can run concurrently

CUDA Memory Architecture Overview



Global memory
- •Main means of communicating with PC
- •R/W Data between host and device
- •Contents visible to all threads

Texture and Constant Memories
- • Constants initialized by host

Shared memory
- •Private to thread block.
- • Access can be syncronised to avoid data hazards.

Local Memory
- •private to each thread but slower than registers, or shared memory

Each thread can:

Read/write per-thread registers/local memory

Read/write per-block shared memory

Read/write per-grid global memory

Read only per-grid constant/texture memory

The threads can be synchronized on the thread block level, but can not be syncronised on the grid level!!!

Memory	Location	Cached	Access	Latency	Who
Local	Off-chip -DRAM	No	Read/write	100s of cycles	One thread
Shared	On-chip	N/A - resident	Read/write	1 cycle (same as registers!)	All threads in a block
Global	Off-chip	No	Read/write	100s of cycles	All threads + host
Constant	Off-chip	Yes	Read	100s of cycles	All threads + host
Texture	Off-chip	Yes	Read	100s of cycles	All threads + host

Starting CUDA Development

- Analyze Matrix Multiplication Example: http://courses.ece.uiuc.edu/ece498/al1/lectures/lecture2%20cuda%20fall%202007.ppt
- Install Visual Studio and CUDA Development Pack (Driver+SDK+Toolkit), available here: http://www.nvidia.com/object/cuda_get.html
- (!)Read release notes to start-up (contains installation instructions and how to start)
- To create your first CUDA project follow these steps (from release-notes):
  - 1. Copy the content of "NVIDIA CUDA SDK\projects\template" to a directory of your own "NVIDIA CUDA SDK\projects\myproject“
  - 2. Edit the filenames of the project to suit your needs
  - 3. Edit the *.sln, *.vcproj and source files. Just search and replace all occurences of "template" with "myproject".
  - 4. Build the release, debug, emurelease, and/or emudebug configurations using myproject.sln or myproject_vc7.sln.
  - 5. Run myproject.exe from the release, debug, emurelease, or emudebug directories located in "NVIDIA CUDA SDK\bin\win32\[release|debug|emurelease|emudebug]". (It should print "Test PASSED".)
- To understand CUDA Architecture and start developing read CUDA Programming Guide 1.1.
- Occupancy Analysis:
  - Compile CUDA project with –keep
  - Review .cubin file
  - Fill-in the shared memory usage and number of threads per block into occupancy calculator