CUDA

Quick overview

Reading

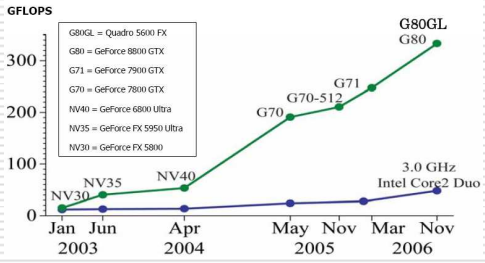
- NVIDIA CUDA. Programming Guide.
- Available online from

  http://developer.nvidia.com/object/cuda.html

About CUDA

- CUDA
  - Compute Unified Device Architecture
    - General purpose computation on comodity graphics hardware (GPUs)
  - Available for free download from the Nvidia website (drivers and SDK).
  - Availble on Nvidia Geforce 8 and Quadro FX 4600/5600 series of GPUs
    - Nvidia promises support and increased functionality on future generation GPUs
  - Available for Windows XP 32 bit and Linux 32/64 bit. Mac and Windows 64 bit/Vista support upcoming

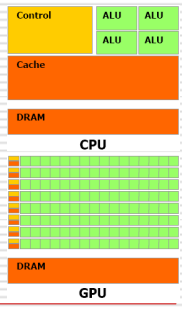
Floating point operations per second (FLOPS). GPU vs. CPU



CUDA Programming Guide Fig. 1-1.

Transistors for data processing rather than cache/flow control

- The GPU utilizes parallel computation
  - Execution on many elements concurrently
- Single Instruction – Multiple Data (SIMD) architecture, i.e. limited flow control requirements
- Memory latency hidden by computation, i.e. limited cache requirements



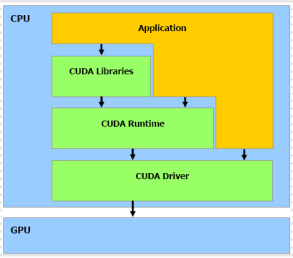
CUDA Programming Guide Fig. 1-2.

CUDA philosophy

- Up until now
  - The GPU could only be programmed through a graphics API imposing a steep learning curve an unnecessary overhead
  - The GPU memory could be read in a general way (gather) but not written generally (no scatter).
- CUDA
  - A hardware and programming model that overcomes these problems and expose the GPU as a truly generic data-parallel computing device.

Software stack

- Hardware driver
- Application Programming Interface and Runtime System
  - ***Extension to the C programming language!***
- High-level mathematical libraries
- User application



CUDA Programming Guide Fig. 1-3.

Some terminology

- GPU – compute *device*
- CPU – *host*
  - data-parallel, compute-intensive portions of applications running on the host are off-loaded onto the device

Some terminology (2)

- Both the host and the device maintain **their own** DRAM, referred to as *host memory* and *device memory*, respectively.
  - One can copy data from one DRAM to the other through optimized API calls that utilize the device's high-performance Direct Memory Access (DMA) engines.

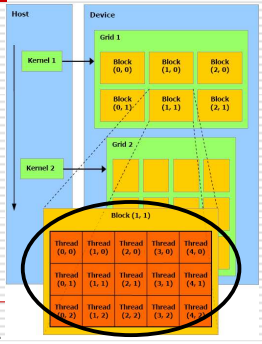
Some terminology (3)

- **kernel**
  - a portion of an application that is executed many times, but independently on different data, can be isolated into a function that is executed on the device as many different threads

Thread batching

Thread **block**

- A batch of threads that operates with a limited amount of *shared memory*
- Syncronization points used to coordinate shared memory access
- Each thread knows its 1D, 2D, or 3D *thread id*

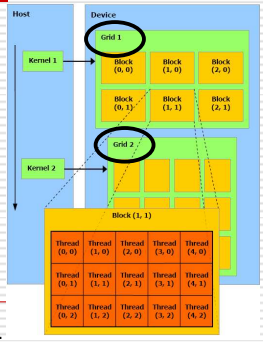


CUDA Programming Guide Fig. 2-1.

Thread batching

Grid of thread blocks

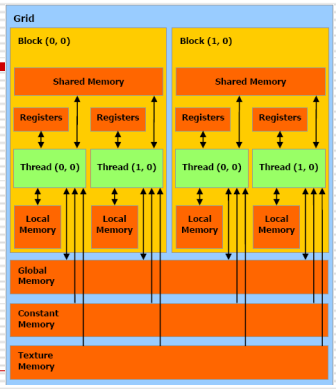
- A set of blocks of the same size that execute the same kernel
- The number of threads in a single block is limited. Many more threads available in a grid
- *No inter-thread communication!!!*
- Each block is identified by its 1D, 2D, or 3D *block id*



CUDA Programming Guide Fig. 2-1.

Memory model

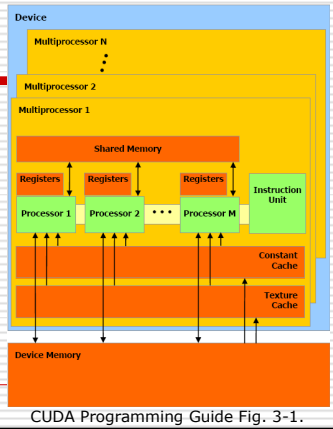
- Read-write per-thread *registers*
- Read-write per-thread *local memory*
- Read-write per-block *shared memory*
- Read-write per-grid *global memory*
- Read-only per-grid *constant memory*
- Read-only per-grid *texture memory*



CUDA Programming Guide Fig. 2-2.

Hardware implementation

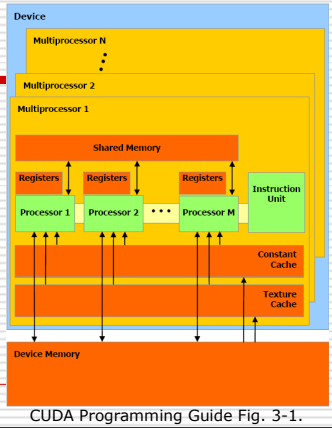
- A Set of SIMD multiprocessors with on-chip shared memory



CUDA Programming Guide Fig. 3-1.

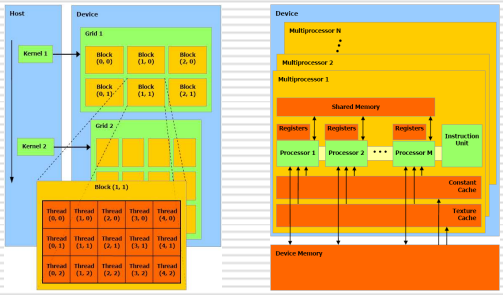
Hardware implementation

- SIMD behaviour through groups of threads called **warps**
- One or more thread blocks are executed on each multi-processor using **time-slicing**
- The issue order of the warps within a block is undefined
- The issue order of the blocks within a grid of thread blocks is undefined



CUDA Programming Guide Fig. 3-1.

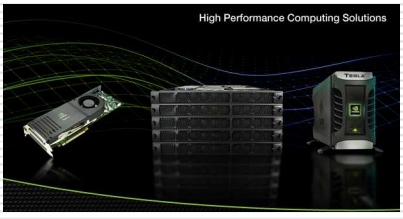
In Summary



CUDA Programming Guide Fig. 2-1. CUDA Programming Guide Fig. 3-1.

Single- and multi-devices available

- One, two, or four "plug-in" devices



www.nvidia.com

To put it short

CPU

```
For x=0 to image size
  For y=0 to image size
    <<
   compute(x,y) >>
  end
end
```

GPU

```
<< compute(threadIdx) >>
```

Debugging

- Release
- Debug
- emuRelease
- emuDebug

- **You cannot overestimate the importance of the debug facilities**