Would'a, CUDA, Should'a.

CUDA:

Compute Unified Device Architecture

OU Supercomputing Symposium

Highly-Threaded HPC

Jargon Alert

- This field is filled with jargon
  - CUDA != GPGPU programming
    - CUDA is a C-like language that facilitates getting your code onto the GPU
    - GPGPU programming is assembly/bit manipulation right on the hardware.
  - Big difference? You make the call.

History

- Mid 90's
  - Stream Processors at Stanford: Imagine and Merrimac
  - SIMD style of processing
  - Programming Language for Merrimac: Brook
  - Adapted to GPUs
    - http://graphics.stanford.edu/projects/brookgpu
    - http://sourceforge.net/projects/brook
- Brook adopted by ATI as a programming language for their GPUs
- NVIDIA: CUDA – some commonality with Brook

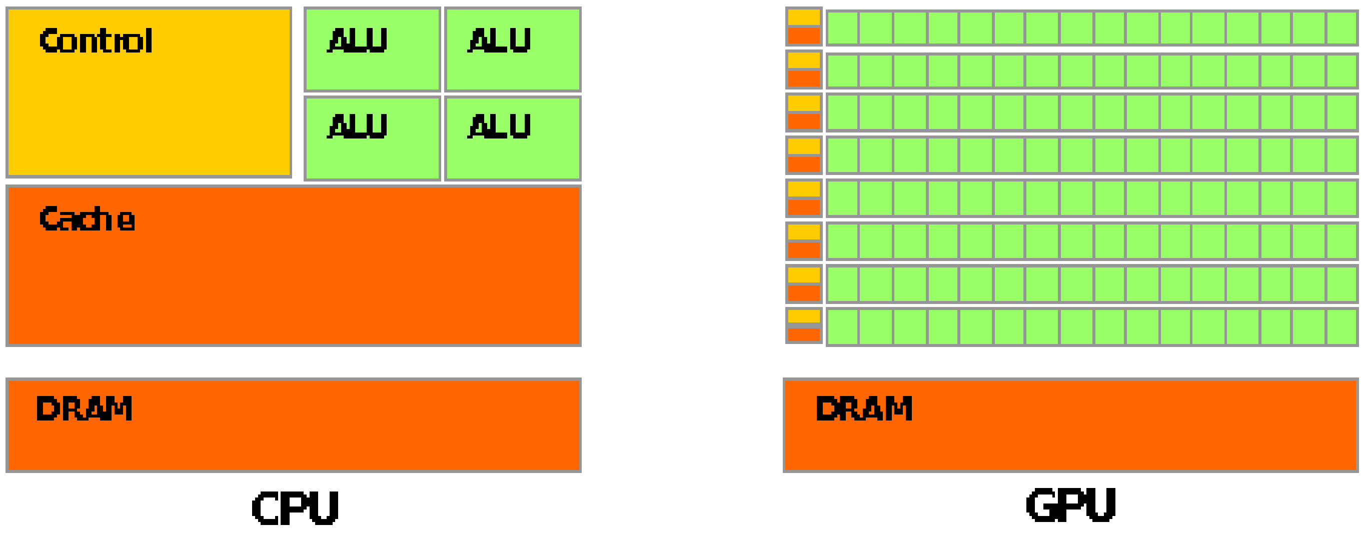
Speedup

- If you can figure out the right mapping, speedups can be extraordinary

Example Applications	URL	Speedup
Seismic Database	http://www.headwave.com	66x – 100x
Mobile Phone Antenna Simulation	http://www.accelware.com	45x
Molecular Dynamics	http://www.ks.uiuc.edu/Research/vmd	21x – 100x
Neuron Simulation	http://www.evolvedmachines.com	100x
MRI Processing	http://bic-test.beckman.uiuc.edu	245x – 415x
Atmospheric Cloud Simulation	http://www.cs.clemson.edu/~jesteel/clouds.html	50x

Source: http://www.nvidia.com/object/IO_43499.html

CPU vs. GPU Layout



Source: Nvidia CUDA Programming Guide

Buzzwords

- Kernel
  - Code that will be run in lock-step on the stream processors of the GPU
- Thread
  - An execution of a kernel with a given index. Each thread uses its index to access elements in array such that the collection of all threads cooperatively processes the entire data set.

Buzzwords

- Block
  - A group of threads.
    - Just like those dastardly pthread'ed threads, these could execute concurrently or independently and in no particular order.
    - Threads can be coordinated somewhat, using the _syncthreads() function as a barrier, making all threads stop at a certain point in the kernel before moving on en mass.

Buzzwords

- Grid
  - This is a group of (thread)blocks.
  - There's no synchronization at all between the blocks.

Hierarchy

- *Grids* map to GPUs
- *Blocks* map to the MultiProcessors (MP)
  - Blocks are never split across MPs, but MPs can have multiple blocks
- *Threads* map to Stream Processors (SP)
- *Warps* are groups of (32) threads that execute simultaneously

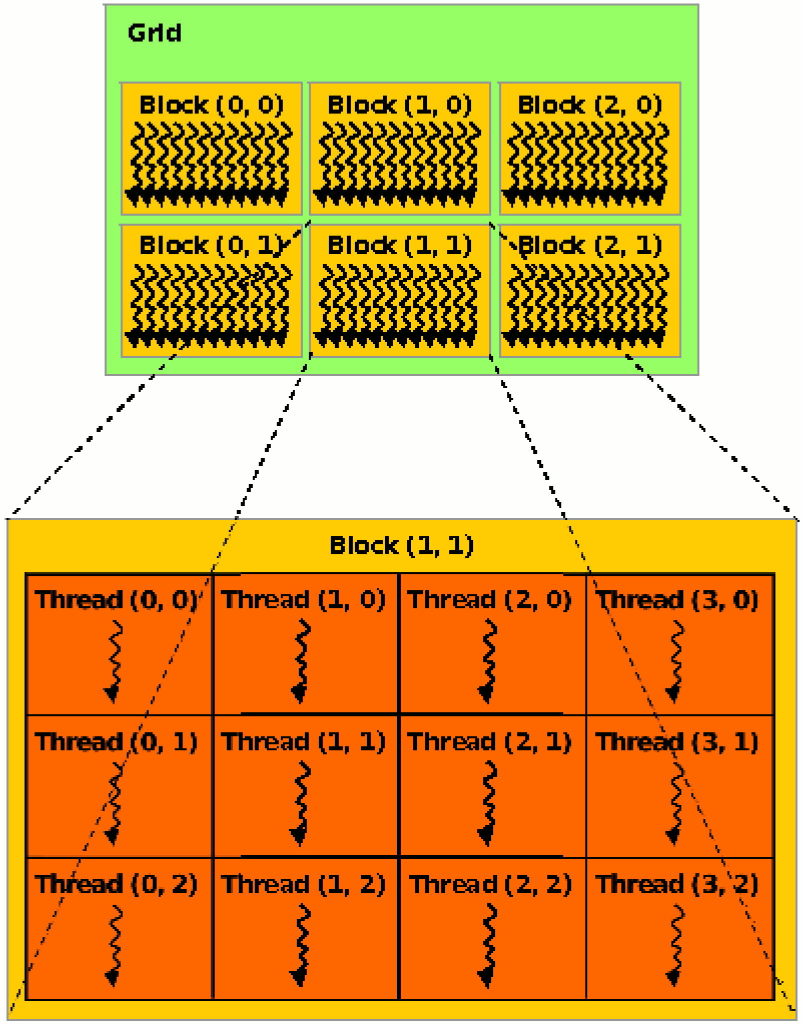


Image Source: Nvidia CUDA Programming Guide

Getting your feet wet

- In your home directory on LittleFe, there's a subdirectory called "CUDA-Examples"
- Inside of this directory, pull up the file `movearrays.cu`
- "`.cu`" is the convention for "CUDA"
- Source: Dr. Dobb's Journal, hpc-high-performance-computing, 207200659

Getting your feet wet

- Looking at the code:
  - Looks a lot like C,
    - Yay!
    - Boo!
  - Notice the `cudaNAME` calls
    - `cudaMalloc,cudaMemcpy,cudaFree`
  - This code simply creates memory locally, creates memory “remotely,” copies memory from the host to the device, copies memory from the device to the device, then copies memory back from the device to the host

Compiling

- `nvcc movearrays.cu -o movearrays`
- Run it with `./movearrays`
- Wow! Did you see that output!

- Look at `movearrays2.cu`
  - Example showing data movement, and our first look at arithmetic operations on the device
  - Compare to arithmetic operations on host

Coding it up

- If you like C, you're golden at this point.
  - Looking at the code, you see that the function call that “does stuff” on the device, `incrementArrayOnDevice()` is declared as `__global__ void`
  - Notice also the decomposition declarations with blocksize and the number of blocks.

Coding it up

- blockIdx.x is a built-in variable in CUDA that returns the blockId in the x axis of the block that is executing this block of code
- threadIdx.x is another built-in variable in CUDA that returns the threadId in the x axis of the thread that is being executed by this stream processor in this particular block

Coding it up

- How do you know where to break?
  http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls
- Each *multiprocessor* has the ability to actively process *multiple blocks* at one time
- How many depends on the number of registers per thread and how much shared memory per block is required by a given kernel. E.g. "It depends!"

Coding it up

- Getting CUDA working:
  - http://www.nvidia.com/cuda
    - Visit the download area
    - Download the CUDA driver
    - Also download the CUDA toolkit
      - Optionally want the CUDA SDK, meaning you really want the CUDA SDK
  - Installation of the driver is relatively straightforward
    - Windows XP/Vista
    - Linux x86/x86_64

Coding it up

- Getting CUDA working:
  - Even if you don't have a supported NVidia card, you can download the toolkit.
    - Toolkit includes an NVidia emulator
    - This means that you can use it in a classroom setting, where students can have universal tools to develop off-host.
      - debug
      - debug-emu
      - release
      - release-emu

Coding it up

- Getting CUDA working:
  - More CUDA than you can shake a “fill out the surveys” stick at:
    - /opt/cuda-sdk/projects
    - /opt/cuda-sdk/bin/linux/release