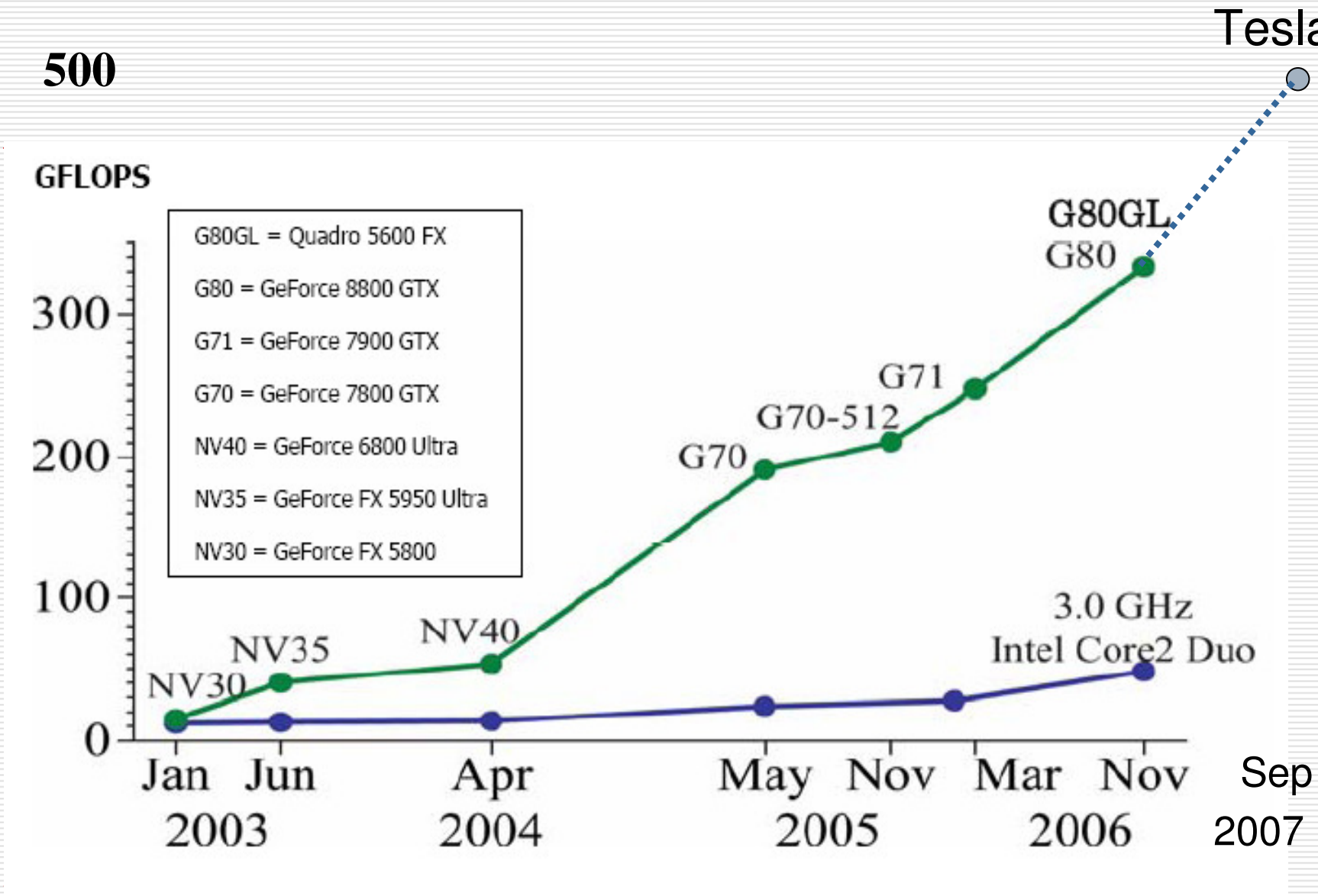
Introduction to CUDA Computing

Evan Bollig
bollig@scs.fsu.edu
428 DSL (SCS VisLab)

http://www.scs.fsu.edu/~bollig/CUDA

Outline

- Computing on the GPU (Past and Present)
- CUDA Design
- Developing with CUDA
- Applications
  - Matrix Addition
  - Matrix Multiply
  - CUFFT
  - CUBLAS
- Using CUDA outside of C/C++ (F77, Matlab)
- CUDA Resources at FSU
- Q/A Session



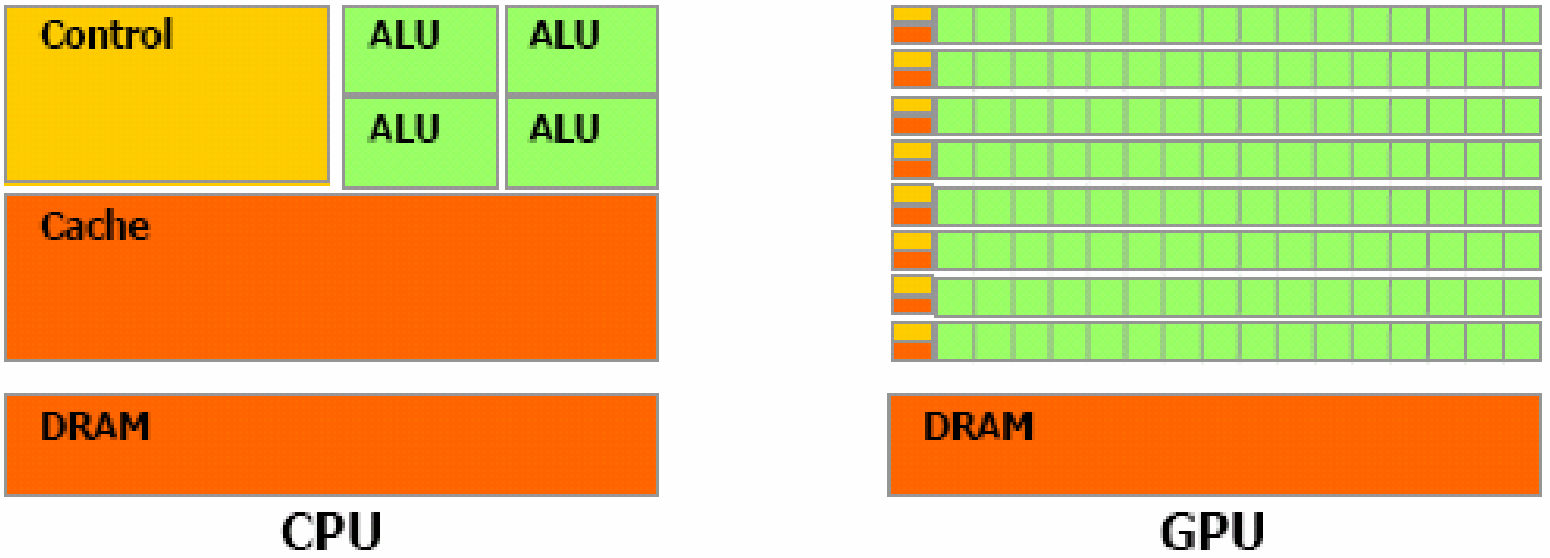
Courtesy, CUDA Programming Guide

Graphics Processor Unit (GPU)

- Generally available in most workstations
  - Easily replaceable (Small investment, large potential)
    - NVidia GeForce 8800 GT: ~$250; ~500 GFLOPS
- GPUs are specialized for highly parallel rendering tasks (data processing)
  - caching and flow control secondary (inverse is true for CPU)
  - Ideal for data-parallel computation
    - General Purpose computation on GPUs (GPGPU)

CPU vs GPU

- Processing units in green



Courtesy, CUDA Programming Guide

GPGPU Prior to Q4 2006

- GPU could only be tricked into computing using graphics commands
  - Data stored in pixels/buffers
  - Inconvenient and unintuitive
  - Requires intricate knowledge of OpenGL, GLSL, etc.
- GPU DRAM could be used for general read but not general write
  - Less flexible than CPU; less desirable to port applications

GPGPU Today: CUDA

- Compute Unified Device Architecture (CUDA)
  - New hardware design + C API for direct computing
  - Disregard graphics aspect of GPGPU
  - Geared towards scientific programming
  - Maintains interoperability with OpenGL to facilitate graphics programming
- GPU is now a “computational coprocessor”

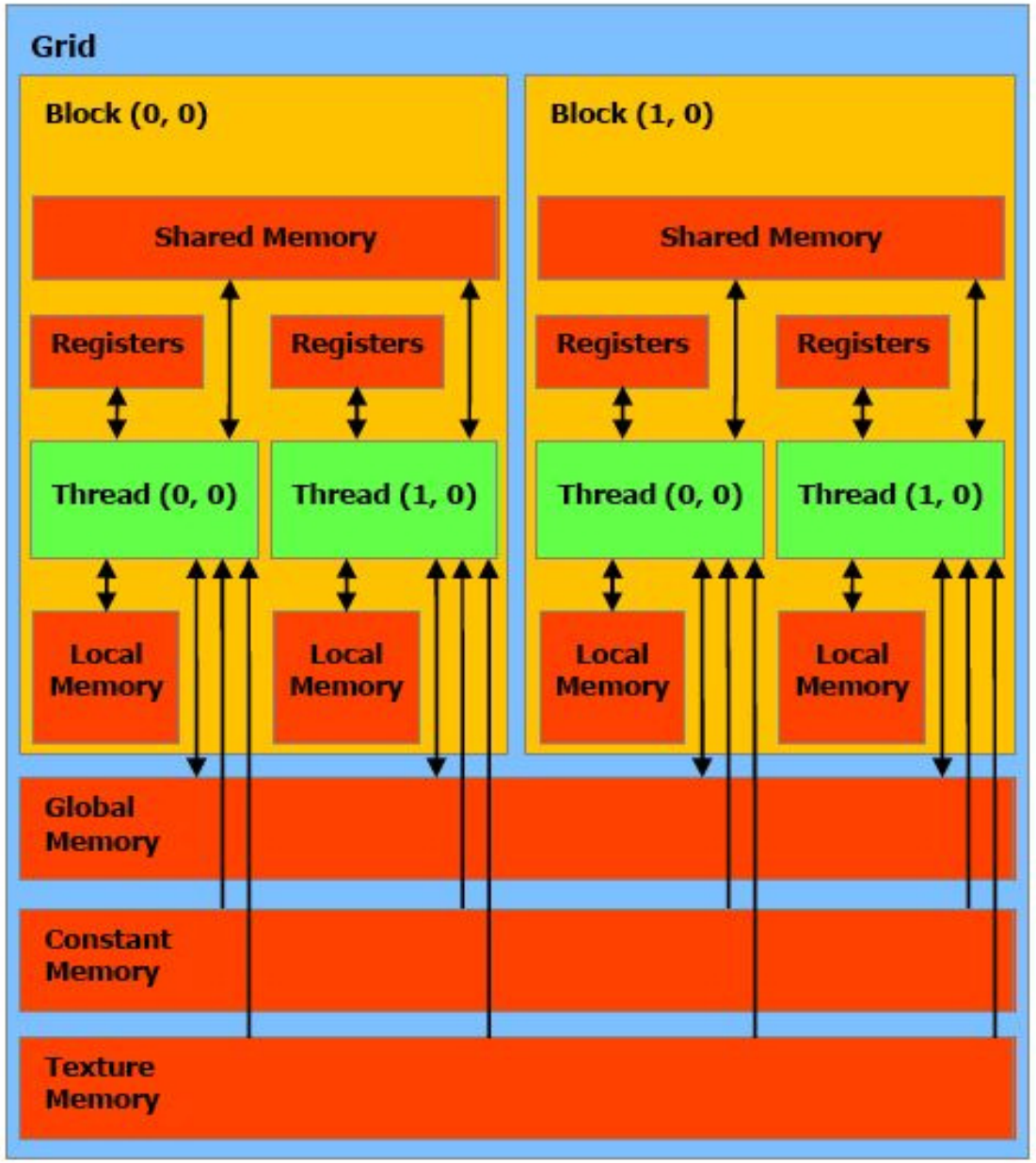
Programming CUDA

Each multiprocessor has a Single Instruction, Multiple Data architecture (SIMD)

*At any given clock cycle, **each** processor of **each** multiprocessor executes the same instruction, but operates on different data.*

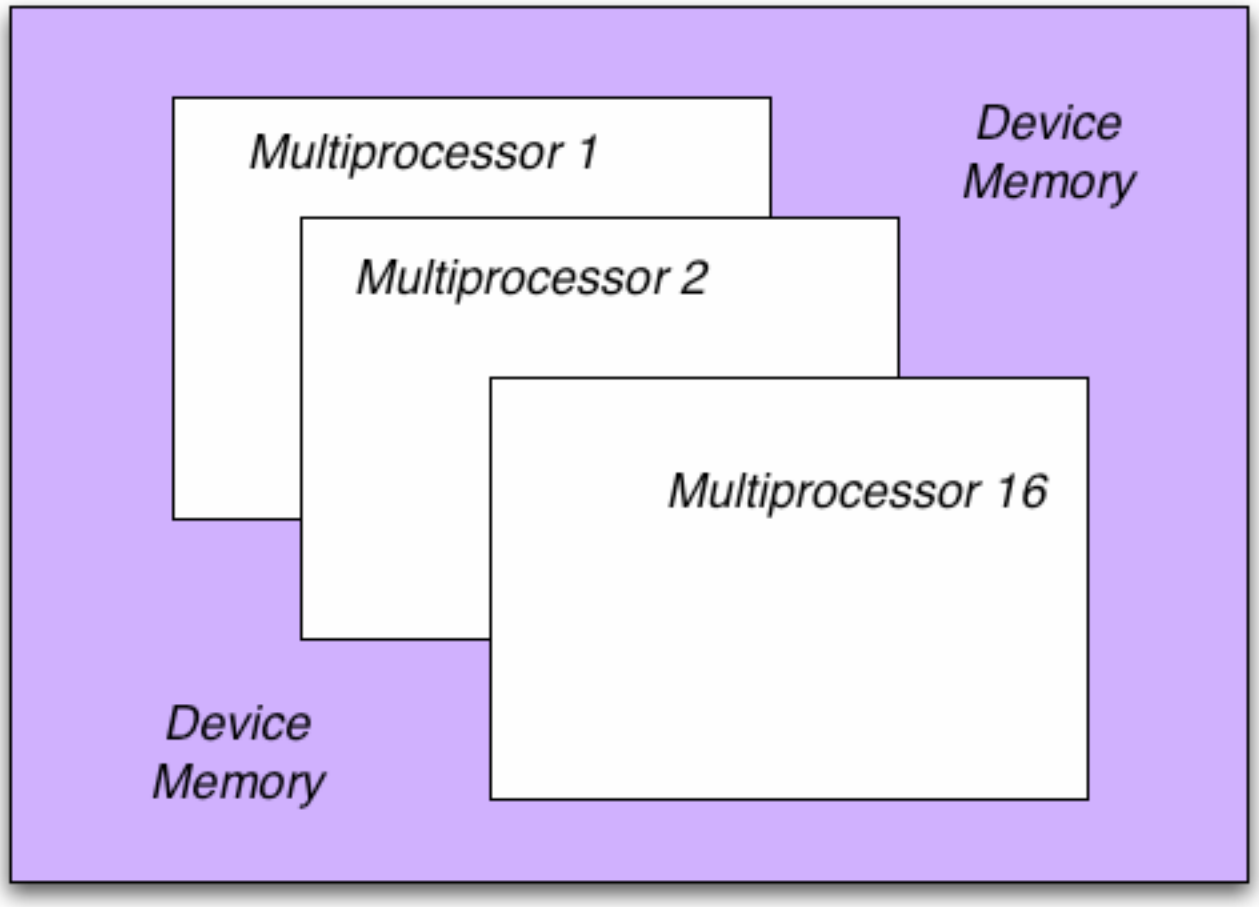
Organized Execution

- A *kernel* is executed by a batch of threads arranged in *blocks*.
  - A *block* of threads share memory and can be synchronized
  - A single multiprocessor executes a *warp* (subset of block) in SIMD
- Thread *blocks* are grouped as *Grids*
  - *Blocks* cannot share memory
  - Used for single execution of kernel

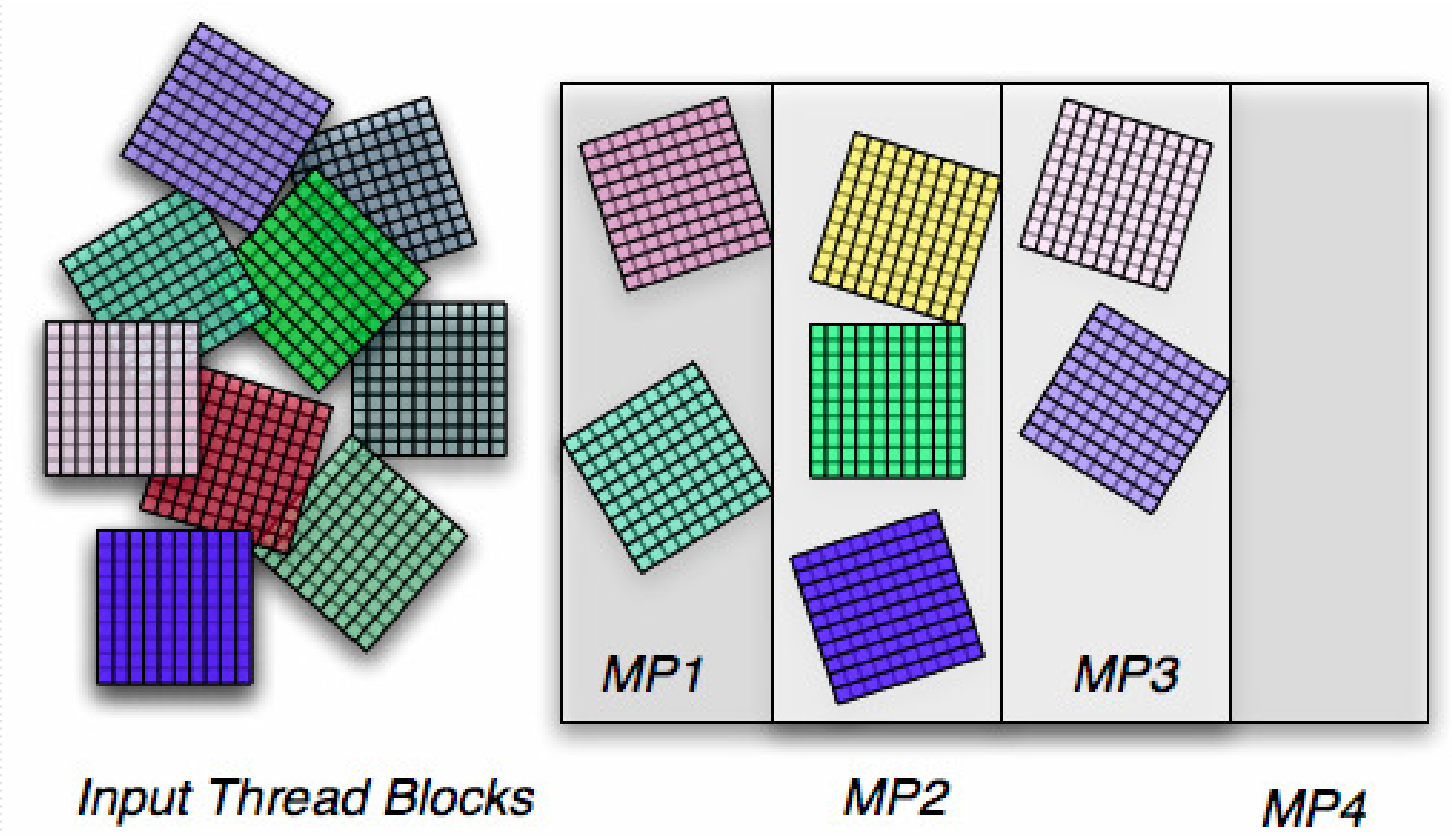


Courtesy, CUDA Programming Guide

Hardware Design



Treatment of Input Blocks



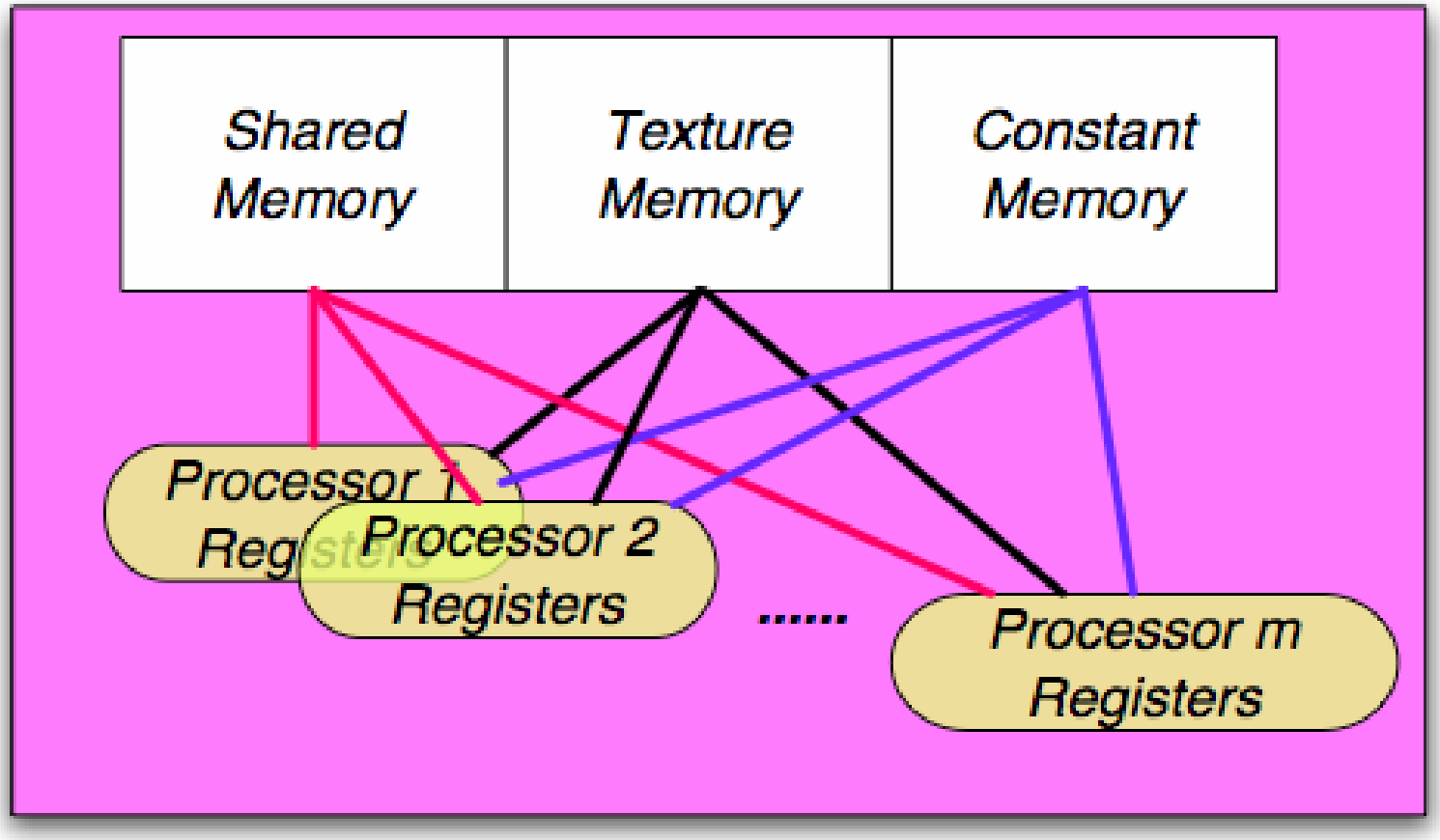
Hardware Comparison

	Number of multiprocessors	Device Memory
GeForce 8800 Ultra	16	768 MB
GeForce 8800 GTX	16	768 MB
GeForce 8800 GTS	12	640 MB
GeForce 8600 GTS	4	256 MB
GeForce 8600 GT	2	256 MB
GeForce 8500 GT	2	256 MB
Quadro FX 5600	16	1.5 GB
Quadro FX 4600	12	768 MB
Tesla C870	16	1.5 GB
Tesla D870	2x16	3 GB
Tesla S870	4x16	6 GB

Courtesy, CUDA Programming Guide

Single Multiprocessor

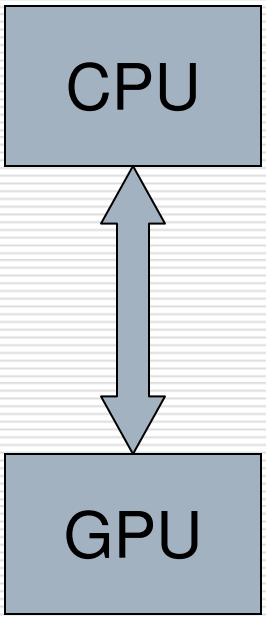
Efficient memory access

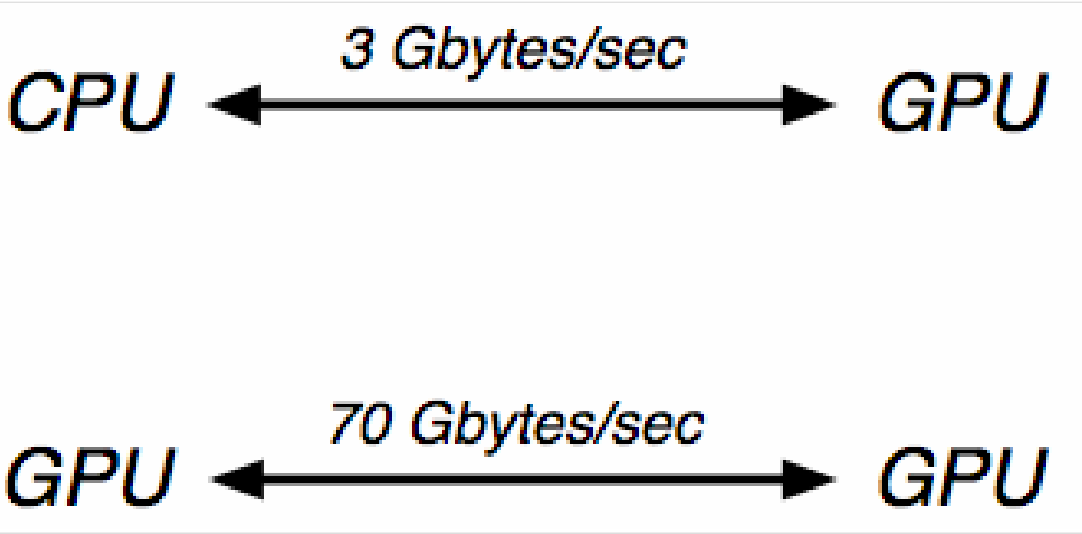


CUDA: Memory Hierarchy

- CPU O(1000)
- GPU global O(100)
- GPU shared O(1)
- GPU local O(1)
- GPU registers O(1)

Memory Bandwidth





CUDA on GeForce 8800

- # Multiprocessors: depends on card
- Each Multiprocessor: 8 processors
- Warp size: 32
- Max # threads per block: 512, # warps per block: 16
- Registers per MP: 8192 (faster access)
- Shared memory: 16,000 bytes per MP (fast access)
- Constant memory: 64,000 bytes + 8 kB cache per multiprocessor (fast access)
- Max concurrent blocks that can run concurrently on MP: 8
- Max # warps that can run concurrently on MP: 24
- Max # threads that can run concurrently on MP: 768
- Max kernel size: 2 million instructions

Developing with CUDA

- Four ways to use CUDA:
  - Driver API (prefix "cu")
    - Low level, harder to program, but offers more flexibility (we will avoid this today)
  - Runtime API (prefix "cuda")
    - Easy to use, great for code readability
  - CUBLAS (prefix "cublas")
    - implementation of BLAS routines (implicit calls to Runtime API)
  - CUFFT (prefix "cufft")
    - implementation of FFTW (http://www.fftw.org/) (impicit calls to Runtime API)

Compiling

- CUDA provided “nvcc”
  - Use instead of gcc
  - nvcc –O3 –o <exe> <input>
    -I/usr/local/cuda/include
    –L/usr/local/cuda/lib –lcudart

Specifying function types

- __device__
  - Device only
- __host__
  - Host only
- __global__
  - declares a *Kernel*
  - Must have void return type
- Example:

```
__host__ int* HostOnlyFunc (int param);
```

Specifying Variable Types

- Use inside device functions:
  - __device__
    - Global device memory
  - __shared__
    - Shared memory space of thread block
  - __constant__
    - Constant memory space (fast read)
- CUDA Intrinsic Vector Types (use anywhere):
  - [u|][char|short|int|long|float][1|2|3|4]
  - i.e. uchar4 = struct { char x, y, z, w };
  - Build your own in normal C fashion

Built-in Variables

- Predefined variables that allow per-thread execution
  - gridDim (dim3)
  - blockIdx (uint3)
  - blockDim (dim3)
  - threadIdx (uint3)
- Values are determined at runtime by execution configuration
  - __global__ function is called with <<< gridDim, blockDim >>> between function name and parameters

Applications 1: Matrix Addition (CPU)

```
void add_matrix_cpu (float *a, float *b, float *c, int N)
{
    int i, j, index;
    for (i=0;i<N;i++) {
        for (j=0;j<N;j++) {
                index =i+j*N;
                c[index]=a[index]+b[index];
        }
    }
}
void main()
{
.....
add_matrix(a,b,c,N);
}
```

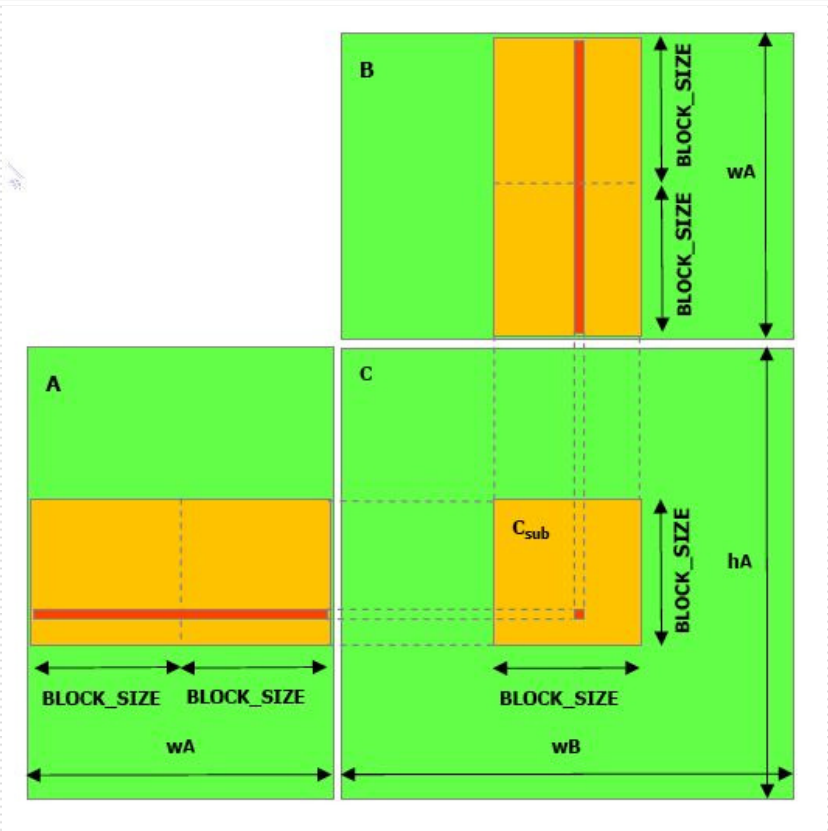
Courtesy, Ian Buck, “Programming CUDA” (S05), Supercomputing ‘07

Applications 1: Matrix Addition (CUDA)

```
__global__ void add_matrix_gpu(float *a, float *b,
                               float *c, int N)
{
    int i=blockIdx.x*blockDim.x+threadIdx.x;
    int j=blockIdx.y*blockDim.y+threadIdx.y;
    int index =i+j*N;
    if( i <N && j <N) c[index]=a[index]+b[index];
}
void main()
{
    dim3 dimBlock (blocksize,blocksize);
    dim3 dimGrid (N/dimBlock.x,N/dimBlock.y);
    add_matrix_gpu<<<dimGrid,dimBlock>>>(a,b,c,N);
}
```

Courtesy, Ian Buck, “Programming CUDA” (S05), Supercomputing ‘07

Applications 2: Matrix Multiply



- http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf
- Page 70 (PDF Page 83)
- Illustrates use of shared memory and __syncthreads()

Courtesy, CUDA Programming Guide

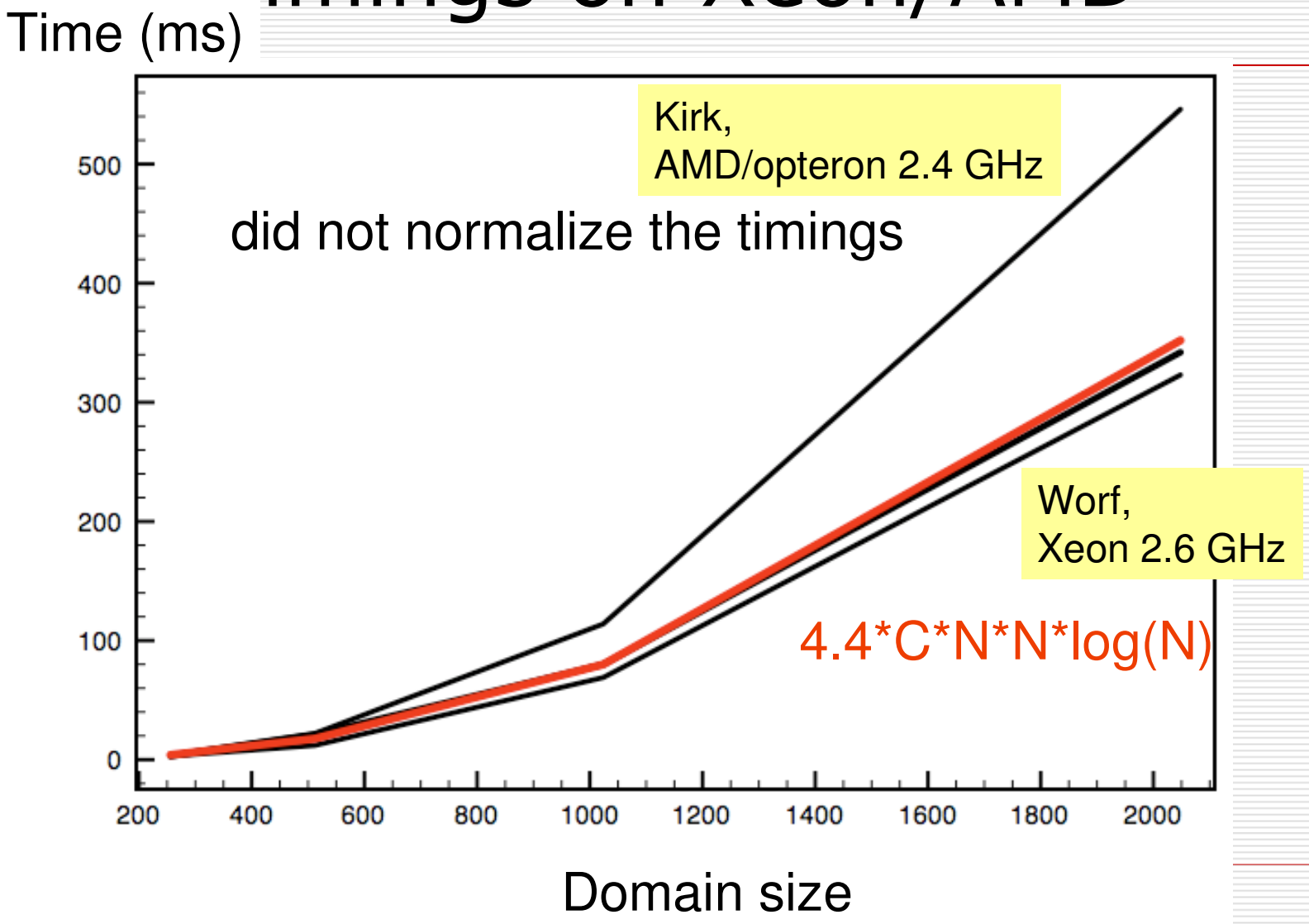
Applications 3: CUBLAS

- http://developer.download.nvidia.com/compute/cuda/1_1/CUBLAS_Library_1.1.pdf
  - Page 4 (PDF Page 8)
- Single precision implementations of most popular BLAS functions (i.e. sgemm, sgemv and saxpy)

Applications 4: CUFFT

- http://developer.download.nvidia.com/compute/cuda/1_1/CUFFT_Library_1.1.pdf
- SimpleCUFFT (From CUDA SDK)
  - http://people.scs.fsu.edu/~bollig/CUDA/simpleCUFFT.cu

FFT Timings on Xeon/AMD



FFT Timings (GPU)

Size	FFT (ms)	Host to device (ms)
256	0.3	1
512	1	3.8
1024	3	12.1
2048	20	47

On CPU: 2048x2048: 380 ms, gain: x20

Using CUDA outside of C/C++

- Matlab
  - http://developer.nvidia.com/object/matlab_cuda.html
  - nvcc and mex compilers used to integrate FFT routines as Matlab commands
  - Parallel execution in an otherwise serial environment
  - VisLab has port of plug-in for Octave

Using CUDA outside of C/C++

- C functions can be called from Fortran
  - developers can easily replace calls to BLAS primitives with CUBLAS
  - http://developer.download.nvidia.com/compute/cuda/1_1/CUBLAS_Library_1.1.pdf
    - Page 12 (PDF Page 15)

Hardware Availability (SCS)

- VisLab Machines (Room 428 DSL)
  - Linux (x86_64)
    - Worf: 8800 GTX
    - Kirk: Tesla C870
    - Uhura, Bones: 8800 GT
  - Windows XP (32-bit)
    - Borg: Quadro FX 5600
    - Reflectance (PowerWall): Quadro FX 5600
- Classroom and Hallway Machines
  - Linux (x86)
    - class[02:20], hallway-[a:f]: Quadro FX 570
- We are working on a CUDA condor pool

Hardware Availability (FSU)

- Math cluster: 20 processors with 20 graphics cards (including 4 Tesla cards)
  - First purchase complete, will be active soon
  - No details for access yet

Recap

- CUDA is a combination of new hardware and software (i.e. nvcc, Runtime API, etc.)
- The C API allows direct control over computation on GPU
- Developers no longer need to understand graphics programming to compute on a GPU
- CUDA executes many threads in SIMD fashion
- Many applications are simple modification to existing code

Useful Resources to Get Started

- NVIDIA CUDA Documentation
  - http://www.nvidia.com/object/cuda_develop.html
- NVIDIA CUDA SDK Examples
- NVIDIA Developer Forums:
  - http://developer.nvidia.com/page/home.html
- GPU Gems 3
  - http://developer.nvidia.com/object/gpu-gems-3.html
- SCS VisLab Twiki
  - http://www.scs.fsu.edu/twiki/bin/view/Computing/VisCluster
- GPGPU
  - http://www.gpgpu.org/data/history.shtml