CS 6068
Parallel Computing
Fall 2014

Prof. Fred Annexstein @proffreda
fred.annexstein@uc.edu
Office Hours: 10-11 MW or by appointment
Tel: 513-556-1807
Meeting: Mondays 6:00-8:50PM Swift 719

Lecture 1: Welcome

- Goals of this course
- Syllabus, policies, grading
- Blackboard Resources
- Introduction/Motivation
- Scope of the Problems in Parallel Computing
- Massive Parallelism on your …

  Cluster, Desktop, Laptop, Phone, Watch

Course Goals

Learning Outcomes:

- Students will learn the computational thinking and programming skills needed to achieve terascale computing performance, applicable in all science and engineering disciplines.

- Students will learn algorithmic design patterns for parallel computing, critical system and architectural design issues, and programming methods and analysis for parallel computing software.

Workload/Grading

The grading for the class will be based on 1 or 2 exams, 6 lab projects, some homework assignments, and a final project presentation and report.

Each student's final grade will be a weighted average – Exams and Quizzes 30%, Homework and Labs 40%, Final Presentation and Final Report 30%. Late homework will be considered on case-by-case basis.

Final Projects

- Students will be expected to work in small teams on a final project. The work for the project can be divided up between team members in any way that works best for you and the group. However, all individual work and outcomes must be clearly documented and self-assessed.

- Each team will make a presentation involving a demonstration. Also, each team will submit a final report documenting all accomplishments.

Course Materials

Udacity Videos CS334 (7 Week online course):
https://www.udacity.com/course/cs344

Recommended Textbooks:
– CUDA by Example, Addison-Wesley
– Programming on Parallel Machines by Norm Matloff (free online)
• Additional Notes will be made available on BB
• Use of Forums on BB is encouraged

Lab Equipment:
– Cuda-based Tesla server in my office
– Your own Mac or PCs with a CUDA enabled GPU

Tentative Schedule

Week 1: Introduction to Parallel Computing
Udacity Lesson 1: Introduction and the GPU Programming Model
Lab Project 1: Converting Photos from Color to Greyscale

Week 2: Parallel Architecture
Udacity Lesson 2: GPU Hardware and Parallel Communication Patterns
Lab Project 2: Gaussian filter for smooth blur

Week 3: Parallel Functional Programming
Udacity Lesson 3: Fundamental Parallel Algorithms (Reduce, Scan, Histogram)
Lab Project 3: HDR Tonemapping

Week 4: Parallel Sorting
Udacity Lesson 4: Fundamental Parallel Algorithms (Applications of Sort and Scan)
Lab Project 4: Red Eye Removal using Template Matching

Week 5: Parallel Optimizations
Udacity Lesson 5: Optimizing GPU Programs
Lab Project 5: Accelerating Histograms

Week 6: CUDA memory models; tiling, texture, and constant memory
Udacity Lesson 6: Parallel Computing Patterns
Lab Project 6: Seamless Image Compositing using Poisson Blending

Week 7: CUDA Advanced Topics: interoperability, atomics, and streams
Midterm Exam
Final Project Proposals Due

Week 8: Parallel Scalability Analysis

Week 9: Parallel Dynamic Programming; Parallel FFT

Week 10: Monte Carlo Methods

Week 11: Parallel Machine Learning

Week 12: Alternative Architectures and MapReduce

Weeks 13-14: Final Project Presentations

Categories of Parallelism

Parallelism can be exploited at many levels by computer hardware

-- CPUs have accelerators GPUs

– Within the CPU core, multiple functional units, pipelining

– Within the Chip, many cores

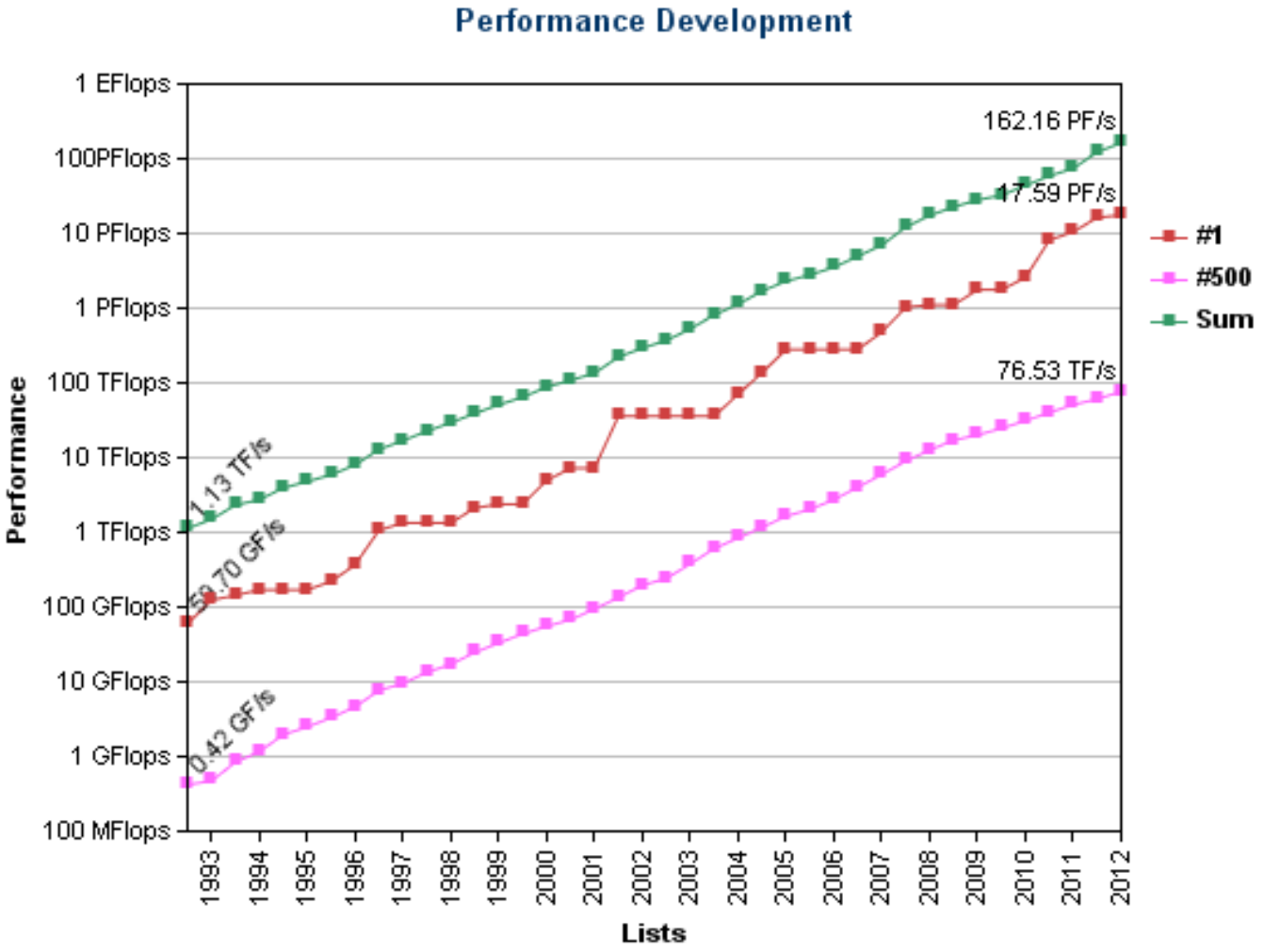
– On a node, multiple chips

– In a system, many nodes

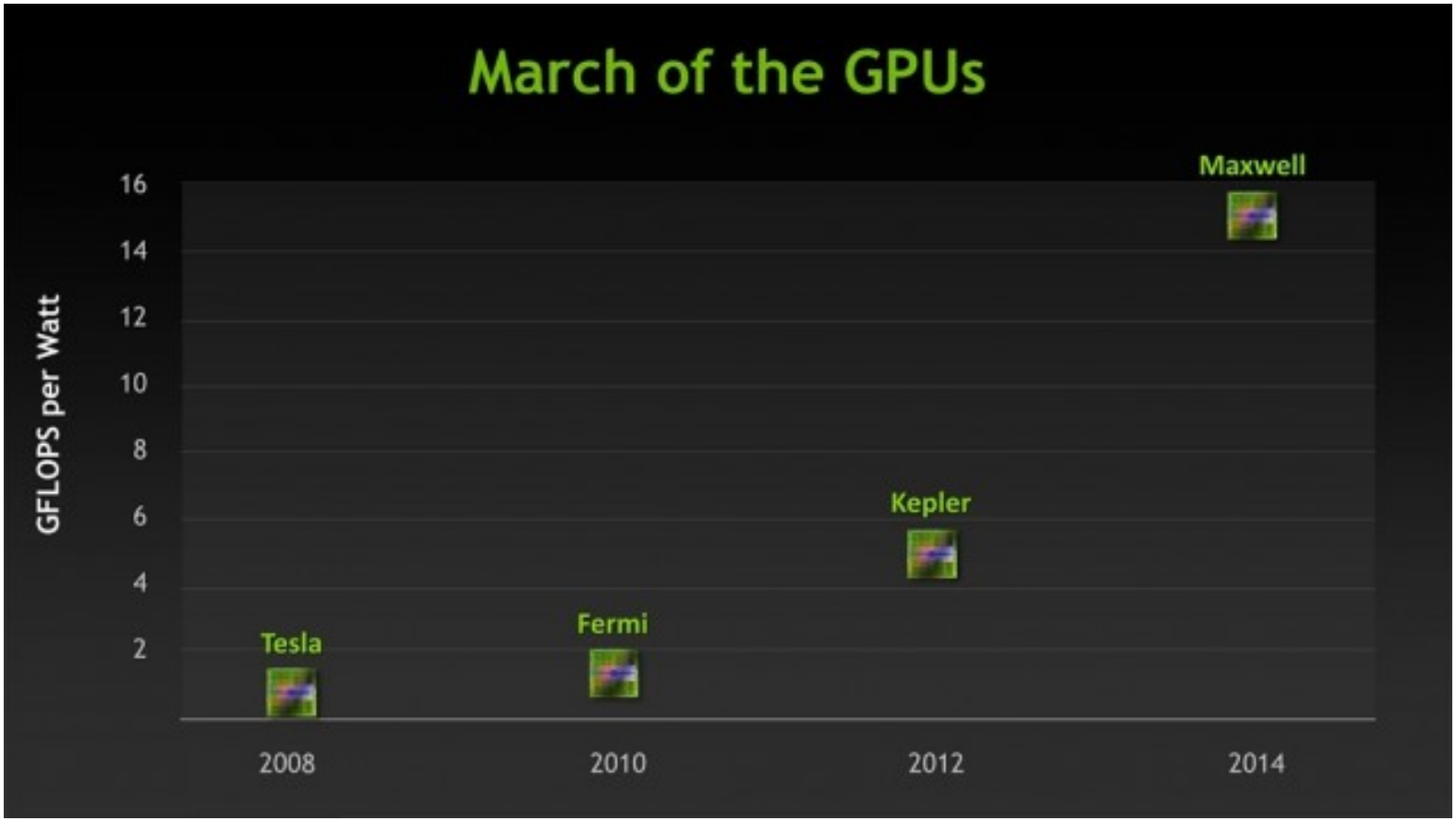
– On the grid, many systems

Supercomputer Performance

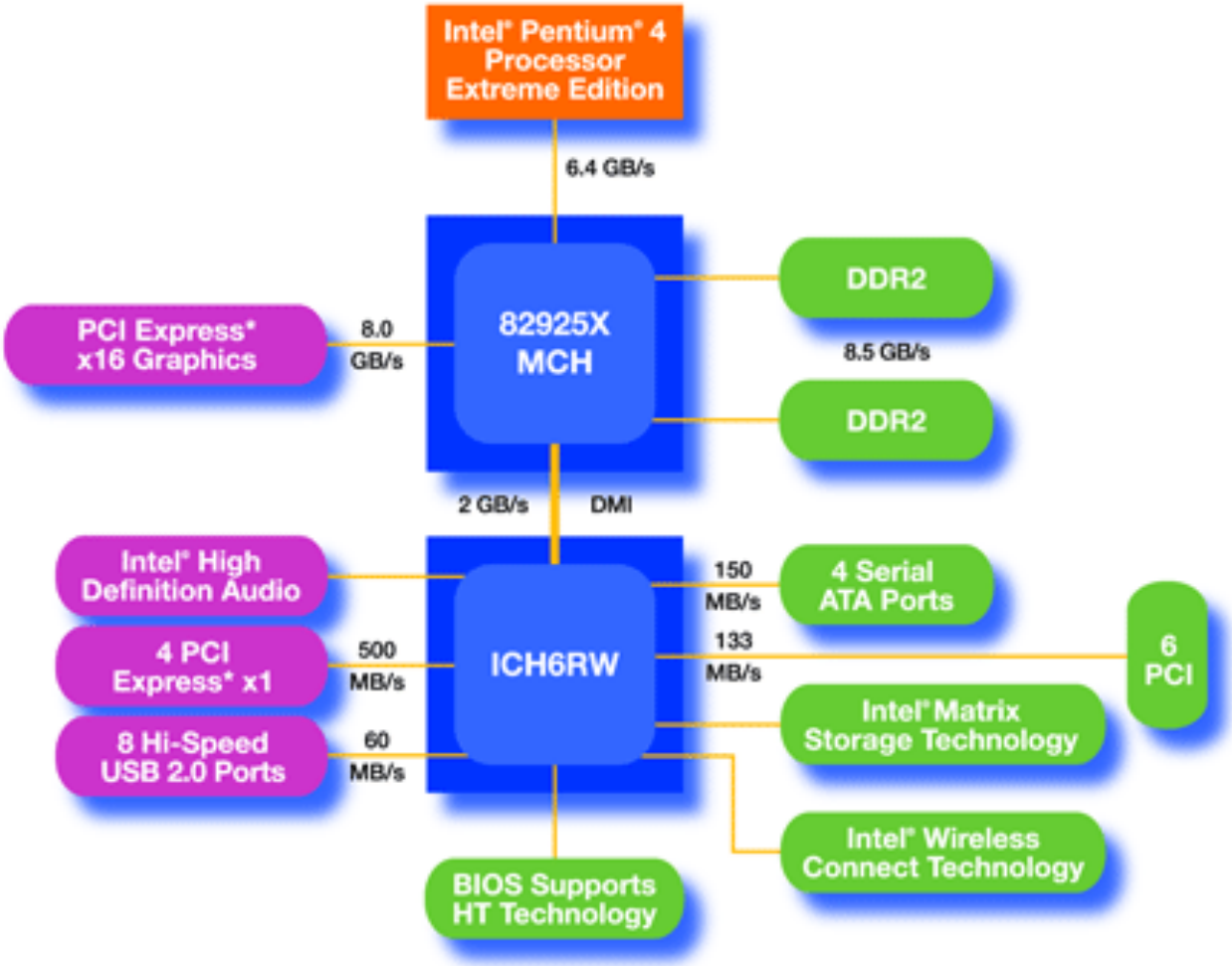
www.top500.org



Power Constrained Parallelism



Example of Physical Reality Behind GPU Processing

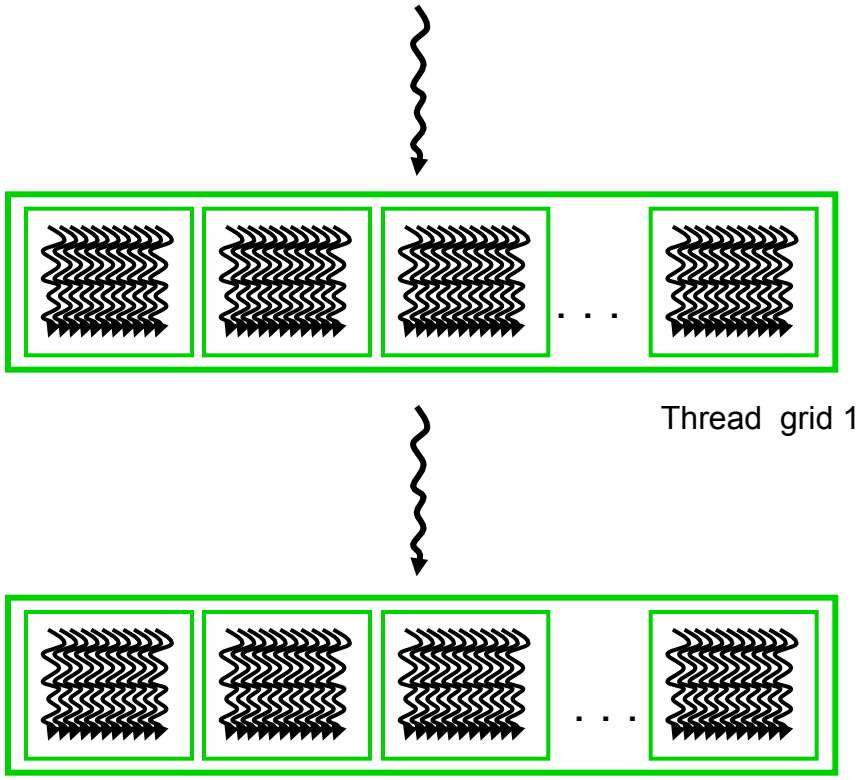


CUDA – CPU/GPU

- Integrated host+device app C program
  - Serial or modestly parallel parts in host C code
  - Highly parallel parts in device SPMD kernel C code

Serial Code (host)

Parallel Kernel (device)
KernelA<<< nBlk, nTid >>>(args);



Thread grid 1

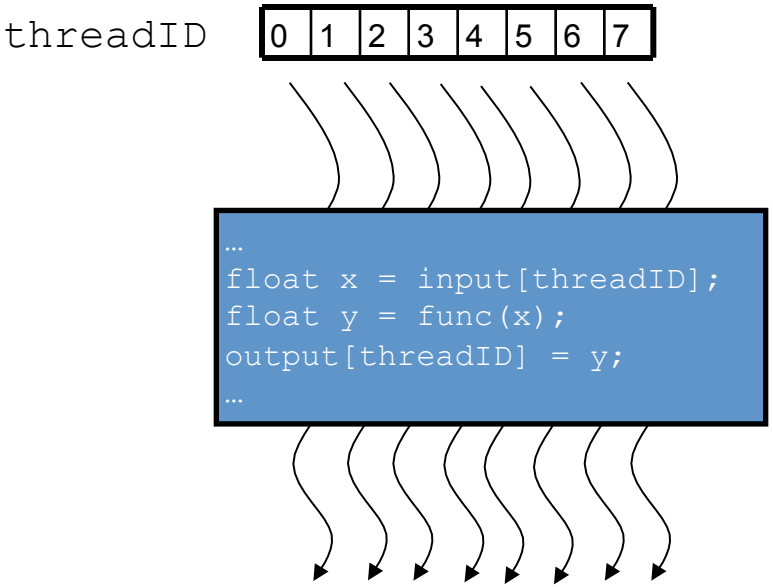
Serial Code (host)

Parallel Kernel (device)
KernelB<<< nBlk, nTid >>>(args);

Thread grid 2

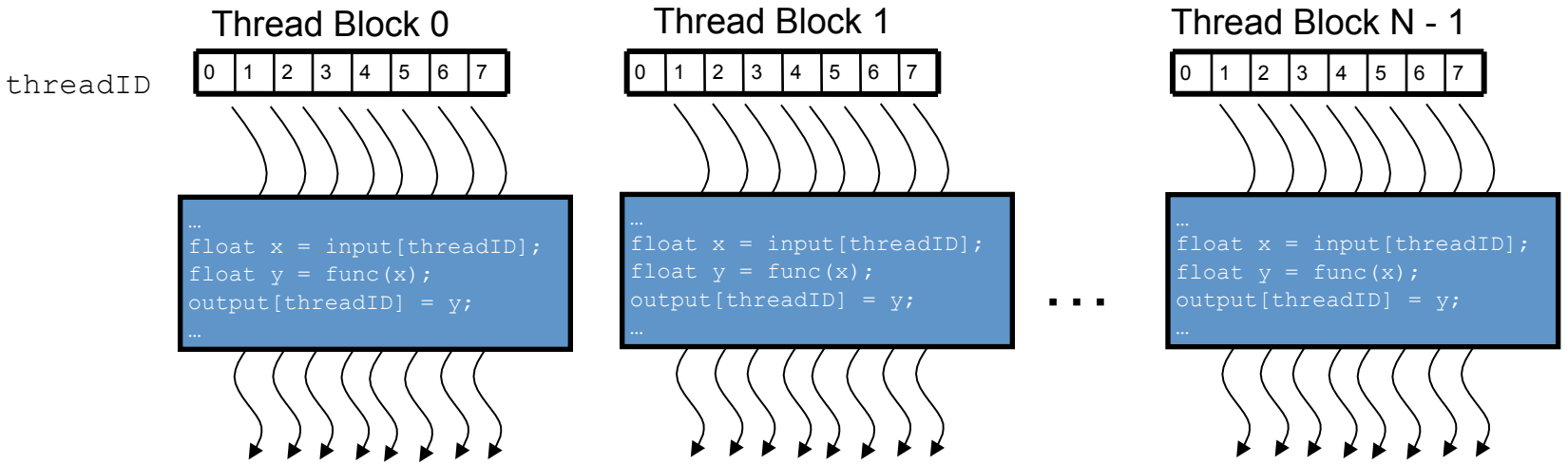
Arrays of Parallel Threads

- A CUDA kernel is executed by an array of threads
  - All threads run the same code (SPMD)
  - Each thread has an ID that it uses to compute memory addresses and make control decisions



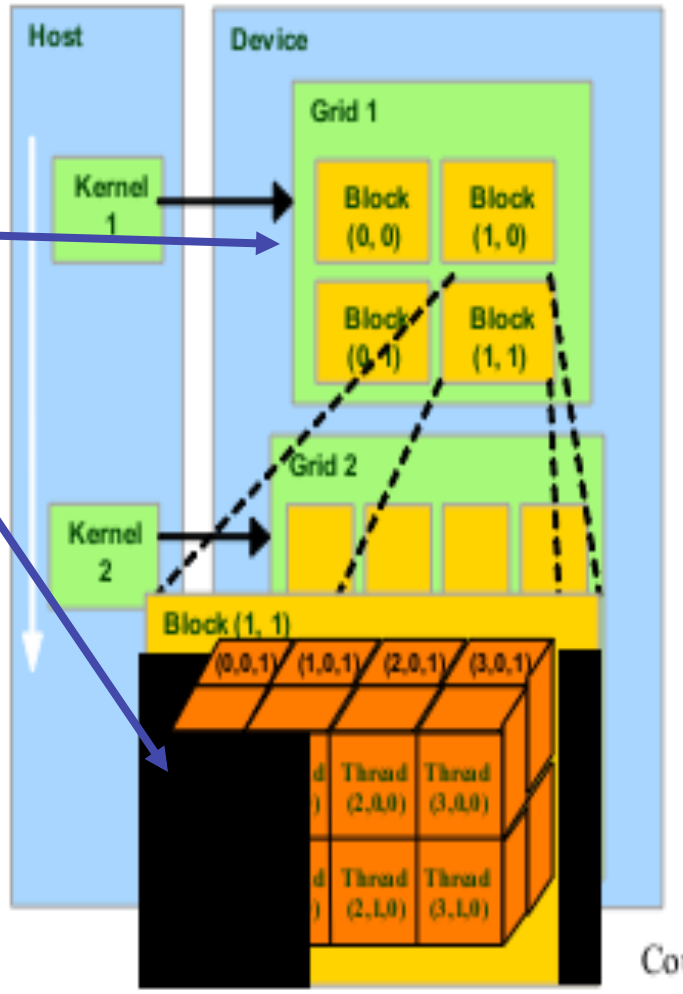
Thread Blocks: Scalable Cooperation

- Divide monolithic thread array into multiple blocks
  - Threads within a block cooperate via shared memory, atomic operations and barrier synchronization
  - Threads in different blocks cannot cooperate



Block IDs and Thread IDs

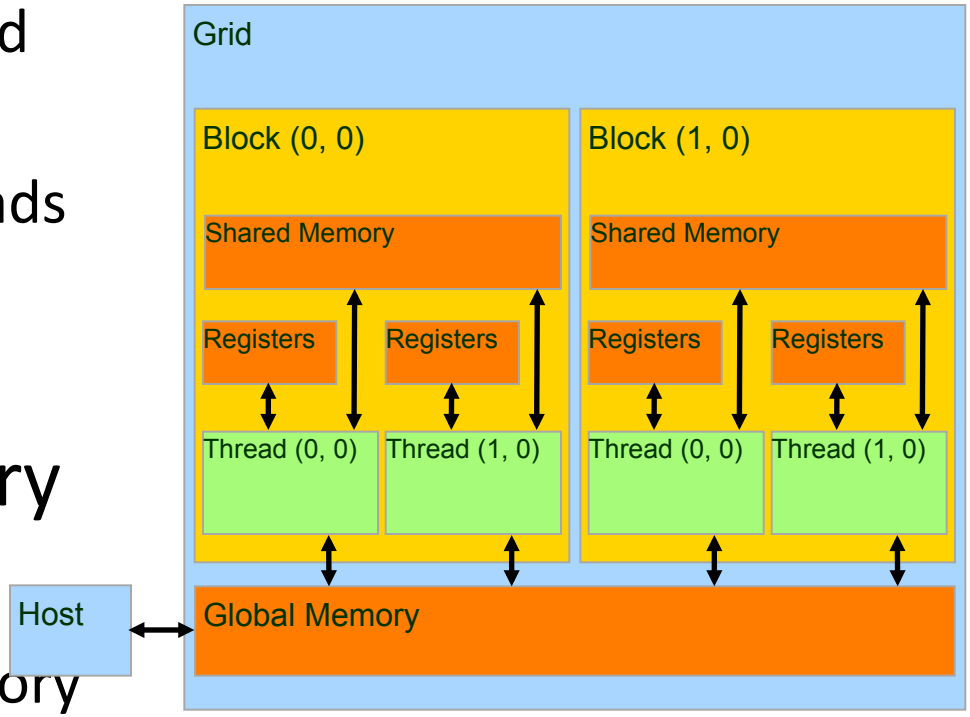
- Each thread uses IDs to decide what data to work on
  - Block ID: 1D or 2D
  - Thread ID: 1D, 2D, or 3D
- Simplifies memory addressing when processing multidimensional data
  - Image processing
  - Solving PDEs on volumes



Courtesy: NDVIA

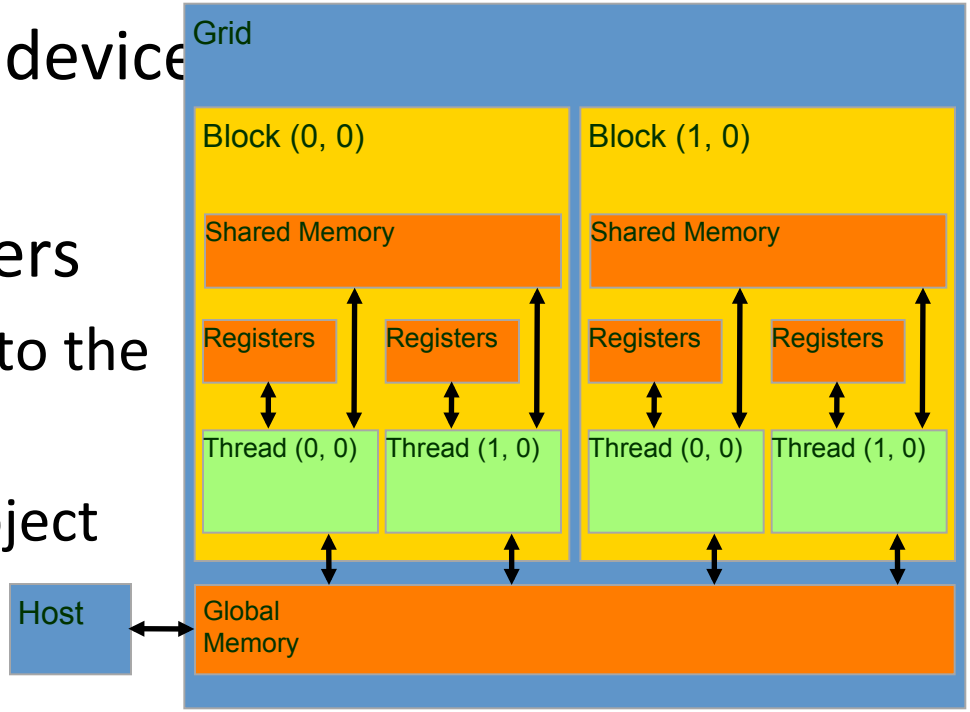
CUDA Memory Model Overview

- Global memory
  - Main means of communicating
  - R/W data between host and device
  - Contents visible to all threads
  - Long latency access, high bandwidth
- Focus on global memory for now
  - Constant and texture memory alternatives that will come later



CUDA Device Memory Allocation

- cudaMalloc()
  - Allocates object in the device Global Memory
  - Requires two parameters
    - Address of a pointer to the allocated object
    - Size of of allocated object
- cudaFree()
  - Frees object from device Global Memory
    - Pointer to freed object



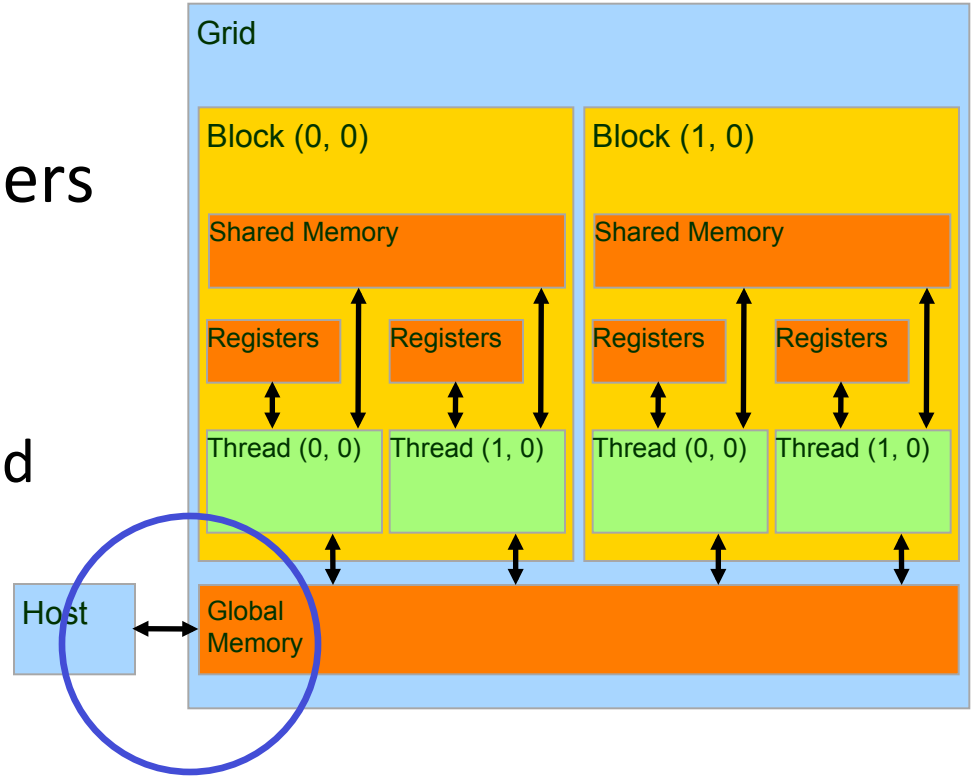
CUDA Device Memory Allocation (cont.)

- Code example:
  - Allocate a 64 * 64 single precision float array
  - Attach the allocated storage to Md
  - “d” is often used to indicate a device data structure

```
TILE_WIDTH = 64;
float* Md
int size = TILE_WIDTH * TILE_WIDTH * sizeof(float);
cudaMalloc((void**)&Md, size);
    //cast  Md  as generic void ptr
  // return value reports errors from API call
cudaFree(Md);
```

CUDA Host-Device Data Transfer

- cudaMemcpy()
  - memory data transfer
  - Requires four parameters
    - Pointer to destination
    - Pointer to source
    - Number of bytes copied
    - Type of transfer
      - Host to Host
      - Host to Device
      - Device to Host
      - Device to Device



- Asynchronous transfer (not supported in current h/w)

CUDA Host-Device Data Transfer (cont.)

- Code example:
  - Transfer a 64 * 64 single precision float array
  - M is in host memory and Md is in device memory
  - cudaMemcpyHostToDevice and cudaMemcpyDeviceToHost are symbolic constants

```
cudaMemcpy(Md, M, size, cudaMemcpyHostToDevice);

cudaMemcpy(M, Md, size, cudaMemcpyDeviceToHost);
```

CUDA Programming Design Pattern

- Code pattern for stub functions
  - Input parameters from host memory
  - Declare corresponding device memory local variables
  - Allocate device memory with cudaMalloc()
  - Transfer data to device with cudaMemcpy()
  - Invoke kernel function (discussed next time)
  - Transfer back to host the results of kernel using cudaMemcpy()
  - Free device memory using cudaFree()

Cuda Kernel Code

```
#include <iostream>

// Kernel:
__global__ void square(float* numbers)
{
    // get the array coordinate:
    unsigned int x  = blockIdx.x * blockDim.x + threadIdx.x;

    // square the number:
    numbers[x] = numbers[x] * numbers[x];
}
```

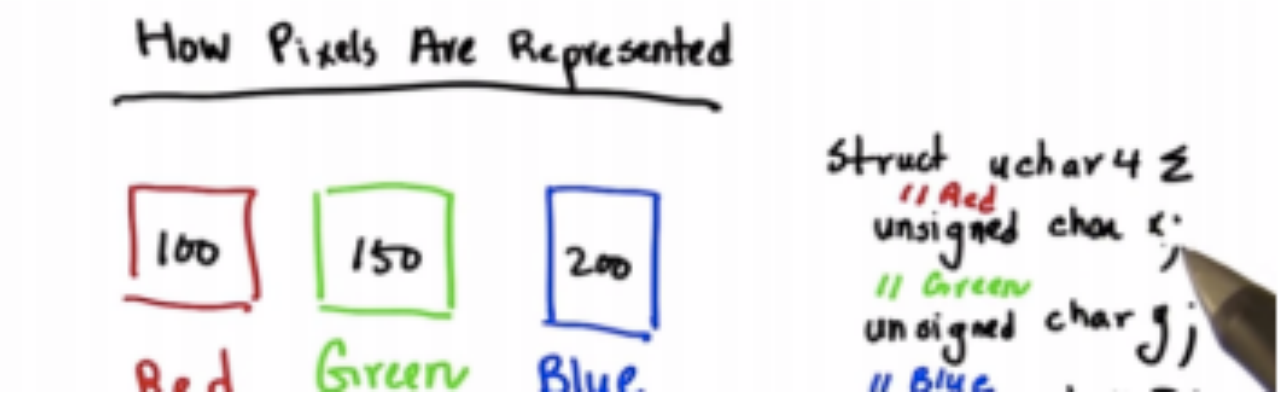
Calling Kernel Code from main.c

```
16  {
17      const unsigned int N = 100; // N numbers in array
18
19      float data[N];      // array that contains numbers to be squared
20      float squared[N];   // array to be filled with squared numbers
21
22      // number to be squared will be the index:
23      for(unsigned i=0; i<N; i++)
24          data[i] = static_cast<float>(i);
25
26      // allocate memory on CUDA device:
27      float* pDevData;        // pointer to the data on the CUDA Device
28      cudaMalloc((void**)&pDevData, sizeof(data));
29
30      // copy data to CUDA device:
31      cudaMemcpy(pDevData, &data, sizeof(data), cudaMemcpyHostToDevice);
32
33      // execute kernel function on GPU:
34      square<<<10, 10>>>(pDevData);
```

Udacity cs344

- https://www.udacity.com/course/cs344
- https://www.udacity.com/wiki/cs344
- https://github.com/udacity/cs344

Homework #1


How Pixels Are Represented
100
150
200
Red
Green
Blue
Struct uchar4 {
// Red
unsigned char
// Green
unsigned char
// Blue