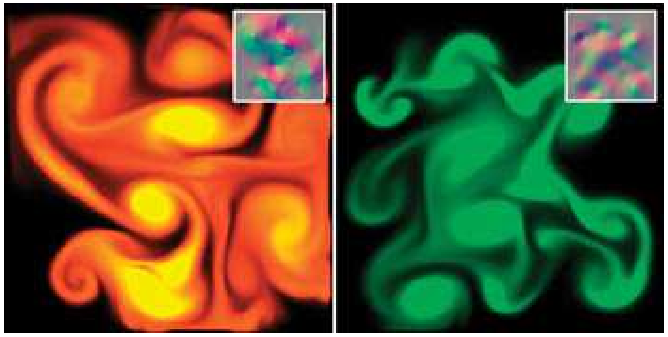
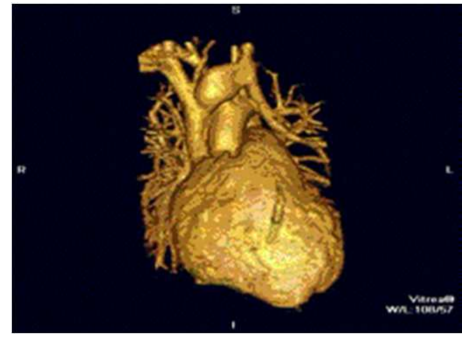
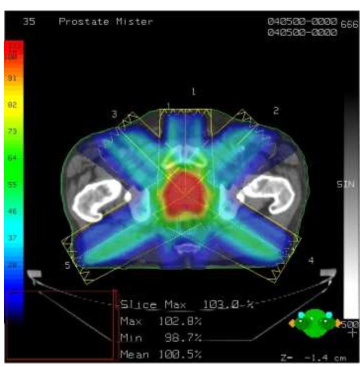
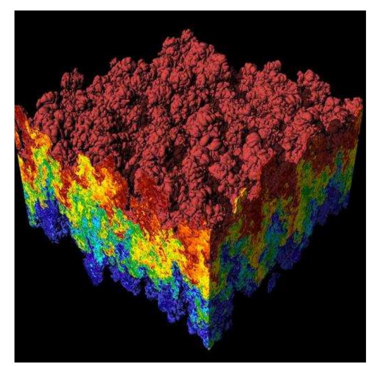




CS 179: GPU Programming

Lecture 1: Introduction





Images: http://en.wikipedia.org
http://www.pcper.com
http://northdallasradiationoncology.com/

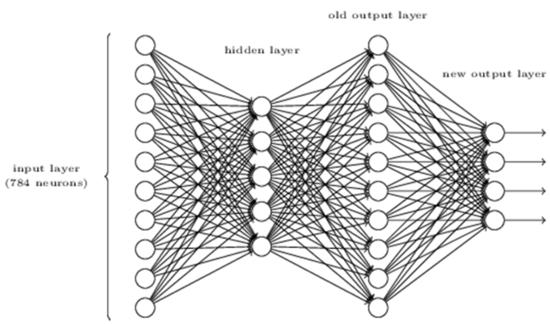
The Problem

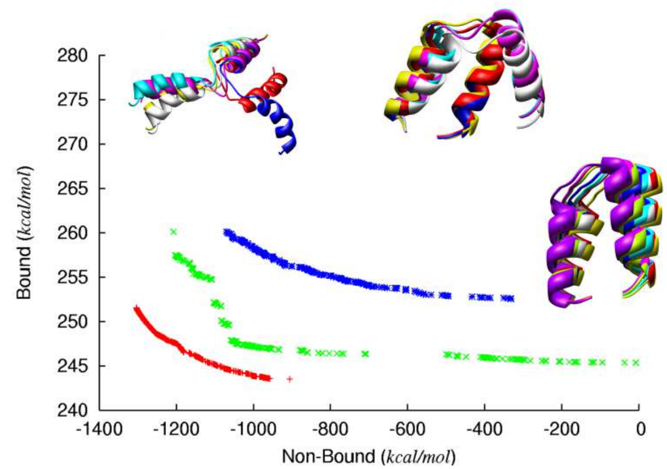
- Are our computers fast enough?

Source: XKCD Comics (http://xkcd.com/676/)

The Problem

- Are our computers *really* fast enough?






Billion-pixel Gaia camera to map galaxy in 3D

The European Space Agency successfully put its Gaia satellite into orbit, with the hopes of unrolling a stunning map of the Milky Way in 3D.

by Tim Hornyak @robotopia / December 19, 2013 7:33 AM PST

2 / 0 / 2 / 0 / / more

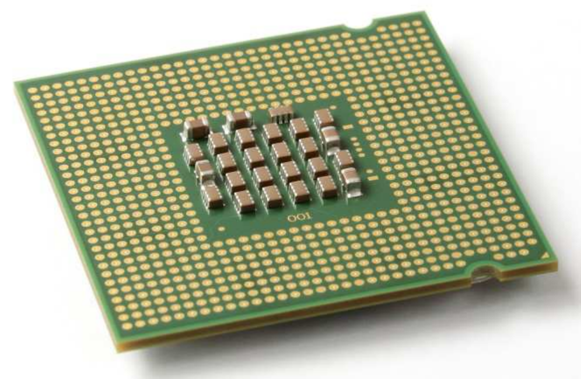
http://lauraskelton.github.io/images/posts/5deepnetworklayer.png
http://www.dmi.unict.it/nicosia/research/proteinFolding3.png
http://www.cnet.com/

The Problem

- What does it mean to “solve” a computational problem?

The CPU

- The "central processing unit"
- Traditionally, applications use CPU for primary calculations
  - Powerful, general-purpose capabilities
  - R+D -> Moore's Law!
  - Established technology



Wikimedia commons: Intel_CPU_Pentium_4_640_Prescott_bottom.jpg

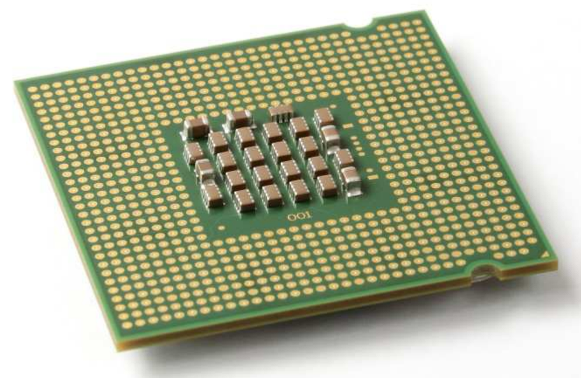
The GPU

- Designed for our “graphics”
- For “graphics problems”, much faster than the CPU!
- *What about other problems?*

This course in 30 seconds

- For certain problems, use

instead of



Images: http://www.nvidia.com,
Wikimedia commons: Intel_CPU_Pentium_4_640_Prescott_bottom.jpg

This course in 60 seconds

- GPU: Hundreds of cores!
  - vs. 2,4,8 cores on CPU
- Good for *highly parallelizable problems*:
  - Increasing speed by 10x, 100x+

Questions

- What is a GPU?
- What is a parallelizable problem?
- What does GPU-accelerated code look like?
- Who cares?

Outline

- Motivations
- Brief history
- "A simple problem"
- "A simple solution"
- Administrivia

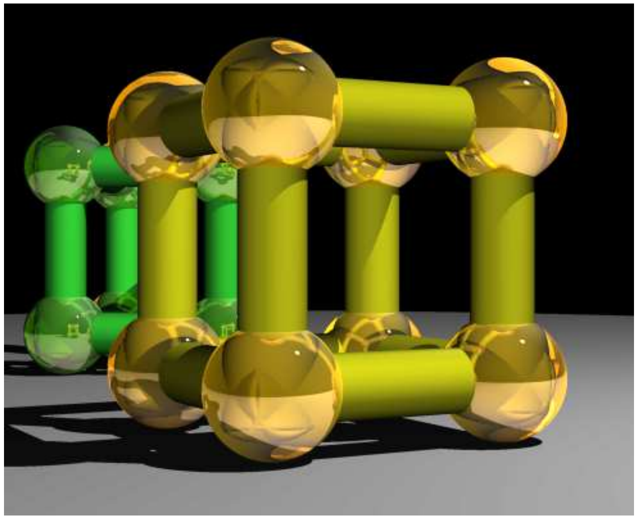
GPUs – The Motivation

- Screens!
  - 1e5 – 1e7 pixels
- Refresh rate: ~60 Hz
- Total: ~1e7-1e9 pixels/sec !

- (*Very* approximate – orders of magnitude)

GPUs – The Motivation

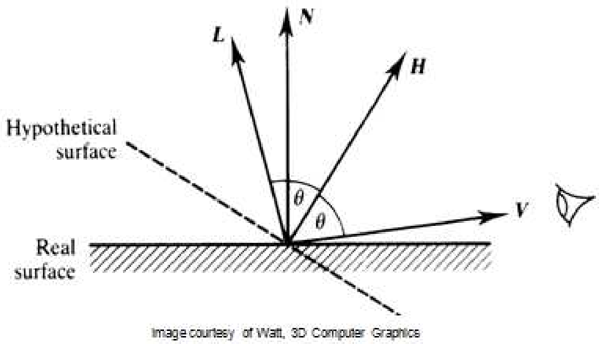
- Lots of calculations are "the same"!



- e.g. Raytracing:

Superquadric Cylinders, exponent 0.1, yellow glass balls, Barr, 1981

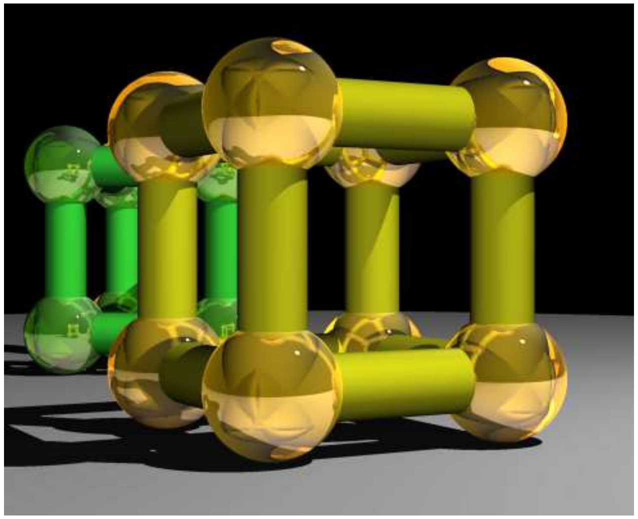
  - Goal: Trace light rays, calculate object interaction to produce realistic image



Watt, 3D Computer Graphics (from http://courses.cms.caltech.edu/cs171/)

GPUs – The Motivation

- Lots of calculations are "the same"!



Superquadric Cylinders, exponent 0.1, yellow glass balls, Barr, 1981

- e.g. Raytracing:

```
for all pixels (i,j):
    Calculate ray point and direction in 3d space
    if ray intersects object:
        calculate lighting at closest object
    store color of (i,j)
```

GPUs – The Motivation

- Lots of calculations are "the same"!

- e.g. Simple shading:

```
for all pixels (i,j):
    replace previous color with new color
    according to rules
```

GPUs – The Motivation

- Lots of calculations are "the same"!

$$T_{\mathbf{v}}\mathbf{p} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{bmatrix} = \mathbf{p} + \mathbf{v}$$

- e.g. Transformations (camera, perspective, …):

```
for all vertices (x,y,z) in scene:
   Obtain new vertex (x',y',z') = T(x,y,z)
```

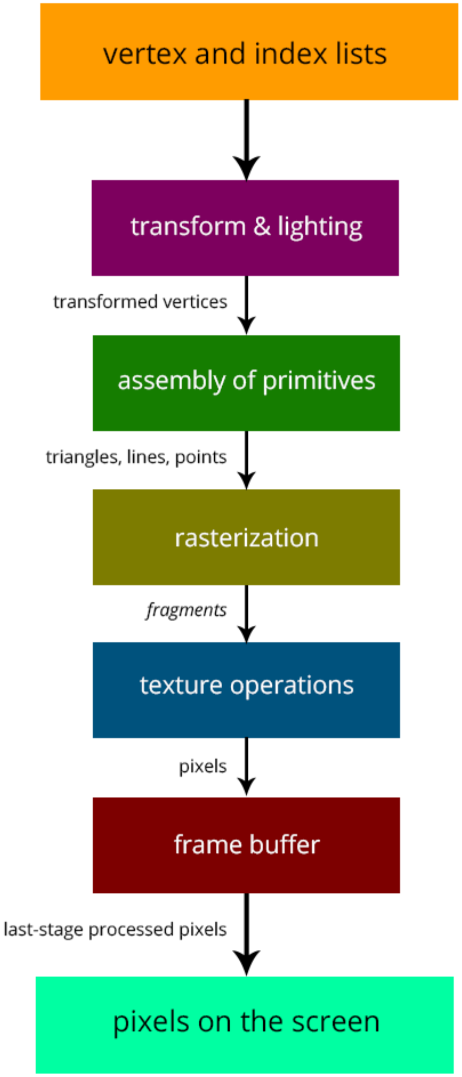
Outline

- Motivations
- Brief history
- “A simple problem”
- “A simple solution”
- This course

GPUs – Brief History

- Fixed-function pipelines
  - Pre-set functions, limited options

http://gamedevelopment.tutsplus.com/articles/the-end-of-fixed-function-rendering-pipelines-and-how-to-move-on--cms-21469
Source: Super Mario 64, by Nintendo



GPUs – Brief History

- Shaders
  - Could implement one's own functions!
  - GLSL (C-like language)
  - Could "sneak in" general-purpose programming!

http://minecraftsix.com/glsl-shaders-mod/

GPUs – Brief History

- CUDA (Compute Unified Device Architecture)
  - General-purpose parallel computing platform for NVIDIA GPUs
- OpenCL (Open Computing Language)
  - General heterogenous computing framework
- …

- Accessible as extensions to C! (and other languages…)

GPUs Today

- “General-purpose computing on GPUs” (GPGPU)

Demonstrations

Outline

- Motivations
- Brief history
- “A simple problem”
- “A simple solution”
- This course

A simple problem…

- Add two arrays
  - A[] + B[] -> C[]

- On the CPU:

```
float *C = malloc(N * sizeof(float));
for (int i = 0; i < N; i++)
    C[i] = A[i] + B[i];
```

  - Operates sequentially… can we do better?

A simple problem…

- On the CPU (multi-threaded, pseudocode):

```
(allocate memory for C)
Create # of threads equal to number of cores on processor
(around 2, 4, perhaps 8)
(Indicate portions of A, B, C to each thread...)

...

In each thread,
For (i from beginning region of thread)
    C[i] <- A[i] + B[i]
    //lots of waiting involved for memory reads, writes, ...
Wait for threads to synchronize...
```

  – Slightly faster – 2-8x (slightly more with other tricks)

A simple problem…

- How many threads? How does performance scale?

- Context switching:
  - High penalty on the CPU
  - Low penalty on the GPU

A simple problem...

- On the GPU:

```
(allocate memory for A, B, C on GPU)
Create the “kernel” – each thread will perform one (or a few)
additions
    Specify the following kernel operation:

    For (all i‘s assigned to this thread)
            C[i] <- A[i] + B[i]

Start ~20000 (!) threads
Wait for threads to synchronize...
```

GPU: Strengths Revealed

- Parallelism / lots of cores
- Low context switch penalty!
  - We can "cover up" performance loss by creating more threads!

Outline

- Motivations
- Brief history
- "A simple problem"
- "A simple solution"
- This course

GPU Computing: Step by Step

- Setup inputs on the host (CPU-accessible memory)
- Allocate memory for inputs on the GPU
- Allocate memory for outputs on the host
- Allocate memory for outputs on the GPU
- Copy inputs from host to GPU
- Start GPU kernel
- Copy output from GPU to host

- (Copying can be asynchronous)

The Kernel

- Our “parallel” function
- Simple implementation

```
__global__ void
cudaAddVectorsKernel(float * a, float * b, float * c) {
    //Decide an index somehow
    c[index] = a[index] + b[index];
}
```

Indexing

- Can get a block ID and thread ID within the block:
  - Unique thread ID!

```
__global__ void
cudaAddVectorsKernel(float * a, float * b, float * c) {
    unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    c[index] = a[index] + b[index];
}
```

Calling the Kernel

```
void cudaAddVectors(const float* a, const float* b, float* c, size){
    //For now, suppose a and b were created before calling this function

    // dev_a, dev_b (for inputs) and dev_c (for outputs) will be
    // arrays on the GPU.

    float * dev_a;
    float * dev_b;

    float * dev_c;

    // Allocate memory on the GPU for our inputs:
    cudaMalloc((void **) &dev_a, size*sizeof(float));
    cudaMemcpy(dev_a, a, size*sizeof(float), cudaMemcpyHostToDevice);

    cudaMalloc((void **) &dev_b, size*sizeof(float)); // and dev_b
    cudaMemcpy(dev_b, b, size*sizeof(float), cudaMemcpyHostToDevice);


    // Allocate memory on the GPu for our outputs:
    cudaMalloc((void **) &dev_c, size*sizeof(float));
```

Calling the Kernel (2)

```
    //At lowest, should be 32
    //Limit of 512 (Tesla), 1024 (newer)
    const unsigned int threadsPerBlock = 512;

    //How many blocks we'll end up needing
    const unsigned int blocks = ceil(size/float(threadsPerBlock));

    //Call the kernel!
    cudaAddVectorsKernel<<<blocks, threadsPerBlock>>>
        (dev_a, dev_b, dev_c);

    //Copy output from device to host (assume here that host memory
    //for the output has been calculated)

    cudaMemcpy(c, dev_c, size*sizeof(float), cudaMemcpyDeviceToHost);

    //Free GPU memory
    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);
}
```

Summary

- For many highly parallelizable problems…
  - GPU offers massive performance increase!

- *Making difficult problems easy*
- *Putting impossible problems within reach*

Outline

- Motivations
- Brief history
- “A simple problem”
- “A simple solution”
- This course

This Course

- General topics:
  - GPU computing /parallelization
    - Audio, linear algebra, medical engineering, machine learning, finance, …
  - CUDA (parallel computing platform)
  - Libraries, optimizations, etc
- Prerequisites:
  - C/C++ knowledge

Administrivia

- Course Instructors/TA's:
  - Kevin Yuh (kyuh@caltech.edu)
  - Eric Martin (emartin@caltech.edu)
- CS179: GPU Programming
  - Website: http://courses.cms.caltech.edu/cs179/
- Overseeing Instructor:
  - Al Barr (barr@cs.caltech.edu)
- Class time:
  - ANB 107, MWF 3:00 PM

Course Requirements

- Option 1:
  - Homework:
    - 7 assignments
    - Each worth 10% of grade
    - Due Wednesdays, ~~5 PM~~ 3 PM (chg'd 4/3/2015)
  - Final project:
    - 3-week project
    - 30% of grade

Course Requirements

- Option 2:
  - Homework:
    - 5 assignments
    - Each worth 10% of grade
    - Due Wednesdays, ~~5 PM~~ 3 PM (chg'd 4/3/2015)
  - Final project:
    - 5-week project
    - 50% of grade
  - Difference: Exchange sets 6,7 for more time on project

Projects

- Topic – your choice!
- Project scale
  - 5-week projects: Significantly more extensive
- Solo or pairs
  - Expectations set accordingly
- Idea generation:
  - Keep eyes open!
  - Talk to us
  - We hope to bring guests!

Administrivia

- Collaboration policy:
  - Discuss ideas and strategies freely, but all code must be your own
  - “50 foot rule” (in spirit) – don’t consult your code when helping others with their code

Administrivia

- Office Hours: Located in ANB 104
  - Kevin: Mondays, 9-11 PM
  - Eric: Tuesdays, 7-9 PM
- Extensions on request
  - Talk to TAs

Machines

- Primary machines (multi-GPU, remote access):
  - haru.caltech.edu
  - mako.caltech.edu (pending)
- E-mail me your preferred username!
- Change your password
  - Separately on each machine (once mako is up)
  - Use *passwd* command

Machines

- Secondary (CMS) machines:
  - mx.cms.caltech.edu
  - minuteman.cms.caltech.edu
- Use your CMS login
- Not all assignments work here!

Machines

- Alternative: Use your own! (Harder):
  - Must have an NVIDIA CUDA-capable GPU
  - Virtual machines won't work!
    - Exception: Machines with I/O MMU virtualization and certain GPUs
  - Special requirements for:
    - Hybrid/optimus systems
    - Mac/OS X
- Setup is difficult! (But we have some instructions)
- May need to modify assignment makefiles

Final remarks for the day…

"Three RAAF FA-18 Hornets in formation after refueling" by U.S. Air Force photo by Senior Airman Matthew Bruch -

Welcome to the course!