From Graphics to Computation

David B. Kirk, NVIDIA Fellow

GPUs Today

Lessons from Graphics Pipeline

- **Throughput** is paramount
- Create, run, & retire *lots of threads* very rapidly
- Use **multithreading** to hide latency

Early Electronic Graphics Hardware

*SKETCHPAD: A Man-Machine Graphical Communication System*

Ivan Sutherland, 1963
The Graphics Pipeline

- Key abstraction of real-time graphics
- Hardware used to look like this
- One chip/board per stage
- Fixed data flow through pipeline

Modern GPUs: Unified “Virtual Pipeline”

- Vertex shaders, pixel shaders, etc. become threads running different programs on a flexible core

Fermi: The Computational GPU

Performance
- 7x Double Precision of CPUs
- IEEE 754-2008 SP & DP Floating Point

Flexibility
- Increased Shared Memory from 16 KB to 64 KB
- Added L1 and L2 Caches
- ECC on all Internal and External Memories
- Enable up to 1 TeraByte of GPU Memories
- High Speed GDDR5 Memory Interface

Usability
- Multiple Simultaneous Tasks on GPU
- 10x Faster Atomic Operations
- C++ Support
- System Calls, printf support
Dawning Nebulae

Second Fastest Supercomputer in the World

1.27 Petaflop

4640 Tesla GPUs

1000+ GPU Clusters Around the World

Programming GPUs
C for CUDA: C with a few keywords

```c
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}

// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

```c
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        y[i] = a*x[i] + y[i];
}

// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

CUDA Programming Effort / Performance

Source: MIT CUDA Course

Targeting Multiple Platforms with CUDA

- CUDA C / C++
- NVCC
  - NVIDIA CUDA Toolkit
- MCUDA
  - CUDA to Multi-core
- Ocelot
  - PTX to Multi-core
- Swan
  - CUDA to OpenCL
- Other GPUs

CUDA to OpenCL

Source: MIT CUDA Course

MCUDA: http://impact.crcf.illinois.edu/mcuda.php
Ocelot: http://code.google.com/p/ptxocelot/
Swan: http://www.multiscalelab.org/swan
Rasterization & Ray Tracing

**Rasterization**
- For each triangle
  - Find the pixels it covers
  - For each pixel: compare to closest triangle so far

**Classical Ray Tracing**
- For each pixel
  - Find the triangles that might be closest
  - For each triangle: compute distance to pixel

Mapped to massively parallel GPU through DirectX or OpenGL

Mapped to massively parallel GPU through NVIDIA OptiX

---

Why ray tracing?
- Ray tracing unifies rendering of visual phenomena
  - fewer algorithms with fewer interactions between algorithms
- Easier to combine advanced visual effects **robustly**
  - soft shadows
  - subsurface scattering
  - indirect illumination
  - transparency
  - reflective & glossy surfaces
  - depth of field
  - ...
- But: resource intensive, challenging to make fast

---

Ray tracing regimes

**Whitted 1980**
- Mirror reflections
- Perfect refractions
- Hard shadows
- 2-20 rays per pixel

**Cook 1984**
- Depth of field
- Motion blur
- Soft shadows
- Glossy reflections
- 20-200 rays per pixel

**Kajiya 1986**
- Indirect illumination
- Caustics
- Physical accuracy
- 200-10^5 rays per pixel
**Stochastic Rasterization**

- Rasterize convex hull of time-continuous triangle
- Ray trace against TCT at each pixel

**Ambient Occlusion**

- Darken pixels by % of hemisphere blocked by nearby triangles
- Compute triangle regions of influence to find affected pixels
**Workloads**

- Each GPU is designed to target a mix of known and speculative workloads
- The art of GPU design is choosing these workloads (and shipping on schedule!)

**What workloads will drive future GPUs?**

- High performance computing
- Graphics
- BIG Science
- Computational graphics

---

**Histogram**

- Distribution of colors in an image
- Image analysis for High Dynamic Range tone mapping

---

**Separable Filters**

- Depth of field, film bloom
- Subsurface scattering via texture space diffusion

---

**Separable Filters**

- Reinhard HDR tone mapping
- HDR in Valve's source engine
- Realistic Skin Rendering
  - Eugene d'Eon, David Luebke, Eric Enderton
  - In Proc. EGSR 2007 and GPU Gems 3
More Post-Processing Effects

CUDA Tessellation

- Flexible adaptive geometry generation
- Recursive subdivision

Real-time fluid effects

Complex fluid-drive motion is all around
- Car exhaust, dust storms, rolling mist, steam, smoke, fire, contrails, bubbles in water, ...

Goal: Add this level of realism to games

Problem: Turbulent motion is computationally intensive!
Solution: GPUs are computational monsters!

Calculate near-field fluid on grid
Fluid velocities drive particle motions

1. Calculate Fluid Velocities on Regular Grid
2nd-Order Accurate CUDA Multigrid Solver

2. Interpolate Fluid Velocities onto Particles
3D Interpolation in CUDA

3. Advance Particles
CUDA Particle System

4. Render Particles
CUDA - OpenGL Interop

APEX Turbulence
Interactive CFD Solution + Volume Rendering

Making Science Better, not just Faster

Increasing Number of CUDA Applications
**Images of sodium in the brain**
- Sodium is one of the most regulated substances in human tissues
- Any significant shift in sodium concentration signals cell death
- Much less abundant than water in human tissues, about 1/2000
- Very large number of samples are needed for good SNR
- Requires high-quality reconstruction, currently considered impractical

Thanks: Wen-mei Hwu

---

**Enables study of brain-cell viability before anatomic changes occur in stroke and cancer treatment.**
- Drastic improvement of timeliness of treatment decision
- Minutes for stroke and days for oncology.

---

**Reconstructing MR Images**
- Spiral scan data + Gridding + FFT:
  - Fast scan, fast reconstruction, good images
  - Can become real-time with about 10X speedup.

*Based on Fig 1 of Lustig et al., Fast Spiral Fourier Transform for Iterative MR Image Reconstruction, IEEE Int'l Symp. on Biomedical Imaging, 2004*
Reconstructing MR Images

Spiral scan data + LS
Superior images at expense of significantly more computation; several hundred times slower than gridding. Traditionally considered impractical!

Summary of Spiral Scan LS Results

<table>
<thead>
<tr>
<th>Reconstruction</th>
<th>Q</th>
<th>F^2/d</th>
<th>Run Time (m)</th>
<th>GFLOP</th>
<th>Run Time (m)</th>
<th>GFLOP</th>
<th>Linear Solver (m)</th>
<th>Recon. Time (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gridding + FFT (CPU, DP)</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0.39</td>
</tr>
<tr>
<td>LS (CPU, DP)</td>
<td>4609.0</td>
<td>0.3</td>
<td>518.0</td>
<td>0.4</td>
<td>1.59</td>
<td>519.59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LS (CPU, SP)</td>
<td>2678.7</td>
<td>0.5</td>
<td>342.3</td>
<td>0.7</td>
<td>1.61</td>
<td>343.91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LS (G80, Naïve)</td>
<td>260.2</td>
<td>5.1</td>
<td>41.0</td>
<td>5.4</td>
<td>1.65</td>
<td>42.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LS (G80, CMem)</td>
<td>72.0</td>
<td>18.6</td>
<td>9.8</td>
<td>22.8</td>
<td>1.57</td>
<td>11.37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LS (G80, CMem, SFU)</td>
<td>13.6</td>
<td>98.2</td>
<td>2.4</td>
<td>92.2</td>
<td>1.60</td>
<td>4.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LS (G80, CMem, SFU, Exp/layout)</td>
<td>7.5</td>
<td>178.9</td>
<td>1.5</td>
<td>144.5</td>
<td>1.69</td>
<td>3.19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

357X, in machine set up time. 228X 108X, LS now a practical method!

High-Throughput Computing = Futuristic Biology

- in-silico screening of drugs
- mastering diseases
- personalized medicine

Thanks: Lorena Barba

In-silico Drug Screening

- Weed out inactive compounds
- Rank “drug candidates” for given targets
- Example:
  - CERN grid — 300,000 potential drugs against avian flu screened
  - 2000 computers, 4 weeks!
  - 4 years cpu-time

protein aggregation

a process critical in
- some degenerative diseases (e.g., Parkinson’s): aggregates abnormal
- drug production: aggregates undesirable
time scale of the process:
- in vitro: up to days!
- impossible for molecular dynamics
Electrostatic Interactions Play a Crucial Role

- Classical molecular dynamics:
  - very detailed ... but too expensive at large scale!
- Alternative: continuum model of surrounding water
  - don’t care what the H2O molecules do
  - model as a continuum dielectric
  - leads to a boundary integral equation (BIE) problem
- Fast algorithm, well-suited for GPU:
  - fast multipole method, solves BIE in O(N) ops

As in Many Computation-hungry Applications

- Three-step approach:
  1. Restructure the mathematical formulation
  2. Innovate at the algorithm level
  3. Tune core software for hardware architecture

Vision—predictive biology, faster, cheaper, accurate

- Drug screening:
  - Few weeks, on a (say) 32-node GPU cluster >> safe drug to market
- Protein aggregation:
  - Get physics right + 100x larger simulation >> understand Parkinson’s
- Analogy:
  - circuit design : it is all done digitally and verified; the circuit works!
  - if it didn’t work, it would be too costly for many consumer electronics

Conclusion: Three Options

- “Accelerate” Legacy Algorithms and Applications
  - Use libraries, recode existing apps
  => good work for domain scientists (minimal CS required)
- Rewrite / Create new Approaches
  - Opportunity for clever algorithmic thinking
  => good work for computer scientists (minimal domain knowledge required)
- Rethink Numerical Methods & Algorithms
  - Potential for biggest performance advantage
  => Interdisciplinary: requires CS and domain insight
  => Exciting time to be a computational scientist
Key GPU Workloads

- Computational graphics (don’t forget DirectXn)
- Scientific and numeric computing
- Image processing – video & images
- Computer vision / Computational Photography
- Speech & natural language
- Data mining & machine learning

Final Thoughts – Education

- We should teach parallel computing in CS 1 or CS 2
  - Computers don’t get faster, just wider
  - Manycore is the future of computing… and graphics

Which goes faster on large data?

ALL Students need to understand this! Early!