



SIGGRAPH2007

General-Purpose Computation on Graphics Hardware

GP GPU



SIGGRAPH2007

Welcome & Overview

Mike Houston
Stanford University





Introduction

- The GPU on commodity video cards has evolved into an extremely flexible and powerful processor
 - Programmability
 - Precision
 - Power
- This tutorial will address how to harness that power for general-purpose computation

Motivation: Computational Power

- GPUs are fast...

- 3.0 GHz Intel Core2 Quad (QX6850):

- Computation: 96 GFLOPS peak
 - Memory bandwidth: 21 GB/s peak
 - Price: \$1100 (chip)

- NVIDIA GeForce 8800 GTX:

- Computation: 330 GFLOPS observed
 - Memory bandwidth: 55.2 GB/s observed
 - Price: \$550 (board)

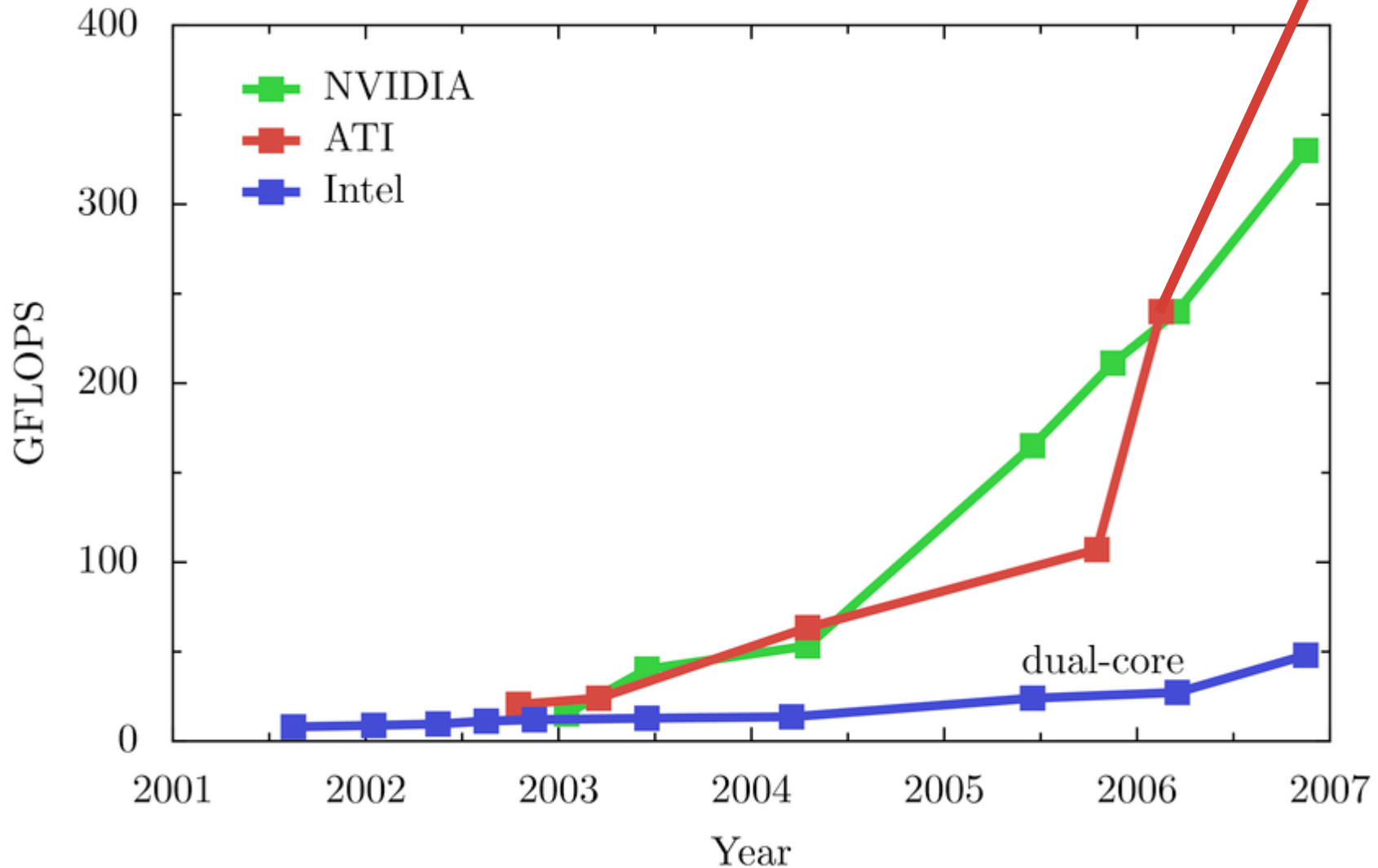
- GPUs are getting faster, faster

- CPUs: 1.4× annual growth
 - GPUs: 1.7×(pixels) to 2.3× (vertices) annual growth

Motivation: Computational Power



SIGGRAPH2007



An Aside: Computational Power

- *Why are GPUs getting faster so fast?*
 - Arithmetic intensity
 - The specialized nature of GPUs makes it easier to use additional transistors for computation
 - Economics
 - Multi-billion dollar video game market is a pressure cooker that drives innovation to exploit this property

Motivation: Flexible and Precise

- **Modern GPUs are deeply programmable**
 - Programmable pixel, vertex, and geometry engines
 - Solid high-level language support
- **Modern GPUs support “real” precision**
 - 32 bit floating point throughout the pipeline
 - High enough for many (not all) applications
 - Vendors committed to double precision soon
 - DX10-class GPUs add 32-bit integers

Motivation: The Potential of GPGPU

SIGGRAPH2007

- In short:
 - The power and flexibility of GPUs makes them an attractive platform for general-purpose computation
 - Example applications range from in-game physics simulation to conventional computational science
 - Goal: make the inexpensive power of the GPU available as a computational coprocessor

The Problem: Difficult To Use

- GPUs designed for & driven by video games
 - Programming model unusual
 - Programming idioms tied to computer graphics
 - Programming environment tightly constrained
- Underlying architectures are:
 - Inherently data parallel
 - Rapidly evolving (even in basic feature set!)
 - Largely secret
- Can't simply "port" CPU code!
 - Good news: it's getting better (CTM, CUDA)



The world changed...

- Since we last taught this course in 2005:
- GPGPU no longer “lunatic fringe”
- Multiple vendor GPGPU initiatives
 - NVIDIA - CUDA
 - AMD - CTM
- Multiple companies doing GPGPU
 - PeakStream (bought by Google)
 - RapidMind
 - Acceleware
- Several real applications
 - Game Physics - Havok FX
 - Molecular Dynamics - Folding@Home, NAMD, VMD



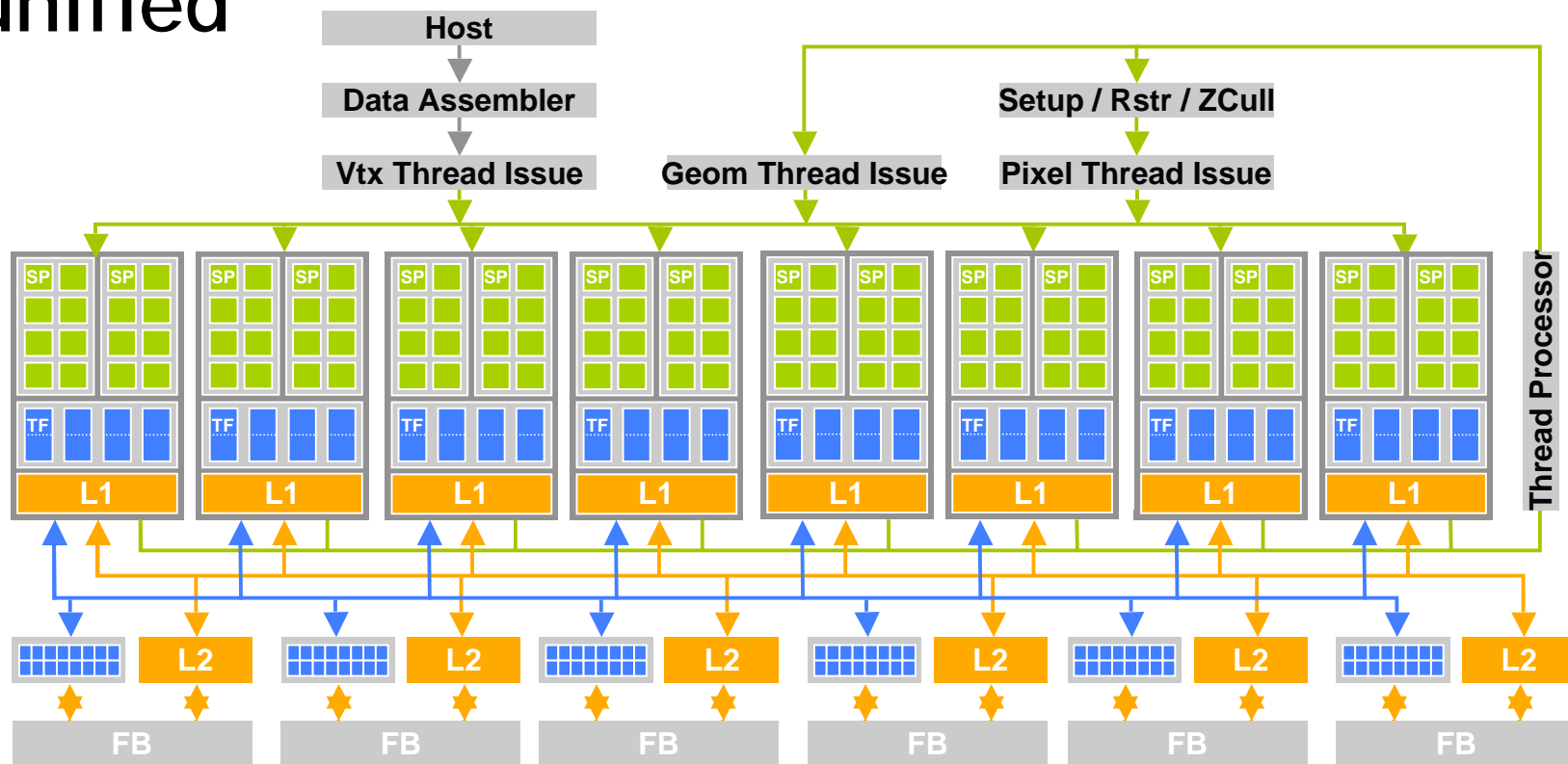
What can you do other than render?

- Large matrix/vector operations (BLAS)
- Protein Folding (Molecular Dynamics)
- Finance modeling
- FFT (SETI, signal processing)
- Raytracing
- Physics Simulation [cloth, fluid, collision,...]
- Sequence Matching (Hidden Markov Models)
- Speech/Image Recognition (Hidden Markov Models, Neural nets)
- Databases
- Sort/Search
- Medical Imaging (image segmentation, processing)
- And many, many, many more...
- See GPGPU.org for more examples



The way we think about things for GPGPU

- A pool of parallel processors
- Vertex/fragment/geometry processors now unified



The Importance of Data Parallelism



SIGGRAPH2007

- GPUs are designed for graphics
- Graphics processes *independent* vertices & pixels
 - Temporary registers are zeroed
 - No shared or static data
 - No read-modify-write buffers
- Data-parallel processing
 - GPUs architecture is ALU-heavy
 - Multiple vertex/pixel pipelines
 - For example, GeForce 8800 GTX has 128 ALUs, Radeon HD2900 has 320
 - Hide memory latency (with more computation)



Arithmetic Intensity

- Arithmetic intensity
 - ops per word transferred
 - Computation / bandwidth
- Best to have *high* arithmetic intensity
- Ideal GPGPU apps have
 - Large data sets
 - Amenable to streaming memory access
 - Lots of parallelism
 - High independence between data elements



Stream Processing

- **Streams**

- Collection of records requiring similar computation
 - Vertex positions, Voxels, FEM cells, etc.
- Provide data parallelism

- **Kernels**

- Functions applied to each element in stream
 - transforms, PDE, ...
- Few dependencies between stream elements
 - Encourage high arithmetic intensity

Programming massively parallel processors



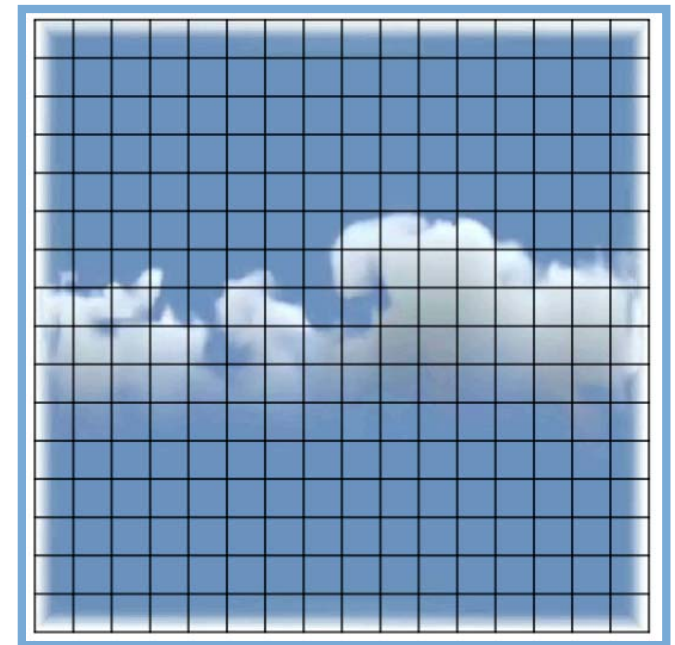
SIGGRAPH2007

- **We are now scaling cores, not clock frequency**
 - How do you run on 100s of cores?
- **Stream programming model**
 - Efficient use of many cores
 - Efficient use of memory bandwidth
 - Scales to large numbers of cores
- **GPUs are already 100s of cores**
 - What you learn here will help you code for massively parallel processors



Example: Simulation Grid

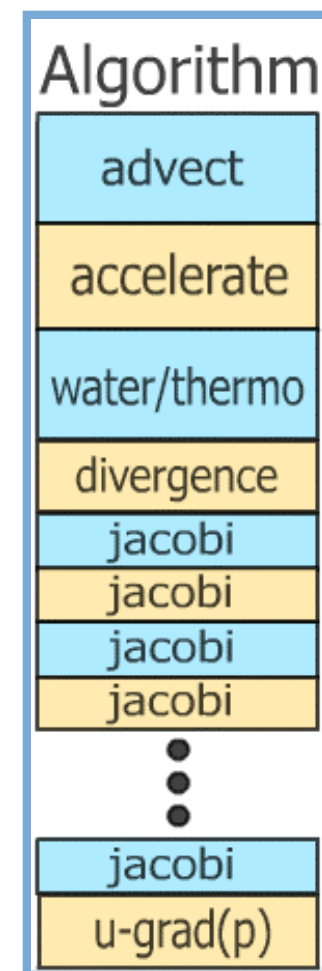
- **Common GPGPU computation style**
 - Arrays represent computational grids = streams
- **Many computations map to grids**
 - Matrix algebra
 - Image & volume processing
 - Physically-based simulation
 - Global illumination
 - Ray tracing, photon mapping, radiosity
- **Non-grid streams can be mapped to grids**





Stream Computation

- **Grid Simulation algorithm**
 - Made up of steps
 - Each step updates entire grid
 - Must complete before next step can begin
- **Grid is a stream, steps are kernels**
 - Kernel applied to each stream element



Cloud simulation algorithm

Kernels



SIGGRAPH2007

advection

```
for (int j = 1; j < height - 1; ++j)
{
  for (int i = 1; i < width - 1; ++i)
  {
    // get velocity at this cell
    Vec2f v = grid(x, y);

    // trace backwards along velocity field
    float x = (i - (v.x * timestep / dx));
    float y = (j - (v.y * timestep / dy));

    grid(x, y) = grid.bilerp(x, y);
  }
}
```

C++

```
void advection(float2 uv : WPOS,
              out float4 xNew : COLOR,

              uniform float dt, // timestep
              uniform float dx, // grid scale
              uniform samplerRECT u, // velocity
              uniform samplerRECT x) // state
{
  // trace backwards along velocity field
  float2 pos = uv - dt * f2texRECT(u, uv) / dx;

  xNew = f4texRECT.bilerp(x, pos);
}
```

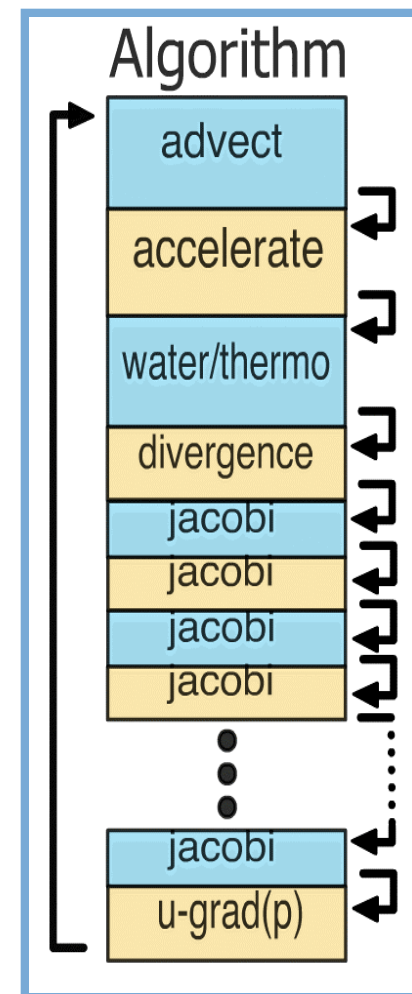
Cg

loop body / algorithm step = kernel



Feedback

- Each algorithm step depends on the results of previous steps
- Each time step depends on the results of the previous time step





Invoking Computation

- **Must invoke computation at each output**
 - In graphics terminology
 - Just draw geometry!
 - Most common GPGPU invocation is a full-screen quad
 - In newer terminology
 - Define domain over output
 - Execute for_all loop over domain



Course goals

- A detailed introduction to general-purpose computing on graphics hardware
- We emphasize:
 - Core computational building blocks
 - Strategies, tools, and analysis for programming GPUs
 - Tips & tricks, perils & pitfalls of GPU programming
- Application examples to bring it all together



Course Prerequisites

- Tutorial intended to be accessible to any savvy computer scientist
- **Helpful but not required: familiarity with**
 - Interactive 3D graphics APIs and graphics hardware
 - Data-parallel algorithms and programming
- **Target audience**
 - Researchers interested in GPGPU research
 - Developers interested in incorporating GPGPU techniques into their work
 - Developers interested in the stream programming model
 - Attendees wishing a survey of this exciting field

Speakers



SIGGRAPH2007

- In order of appearance:

- Mike Houston, Stanford University
- John Owens, University of California Davis
- Naga Govindaraju, Microsoft Research
- Mark Harris, NVIDIA
- Justin Hensley, AMD
- Cyril Zeller, NVIDIA
- Aaron Lefohn, Neoptica
- Jens Krüger, Technische Universität München
- Simon Green, NVIDIA

Schedule



SIGGRAPH2007

- 8:30 **Introduction** **Houston**
Tutorial overview, GPGPU programming
- 8:50 **GPU Architecture Overview** **Owens**
How GPUs work
- GPU Building Blocks** **Owens**
- 9:15 **Data-Parallel Algorithms**
Reduce, scan, scatter/gather, sort, and search
- 9:45 **Sort and Search** **Govindaraju**
Sorting networks & specializations, searching

10:15 **Break**



Schedule



SIGGRAPH2007

Languages & Programming Environments

10:30 GPGPU Languages

Accelerator, Brook, RapidMind

Houston

10:55 CUDA

GeForce 8800, Compute Unified Driver Architecture

Harris

11:35 CTM

CTM, Data Parallel Virtual Machine

Hensley

12:15 Lunch



Schedule



SIGGRAPH2007

High Performance GPGPU

- | | | |
|------|--|----------------|
| 1:45 | Performance Analysis & Arch Insights
<i>GPUBench, architectural models for programming</i> | Houston |
| 2:05 | CUDA
<i>GeForce 8800, Compute Unified Driver
Architecture</i> | Zeller |
| 2:35 | CTM
<i>CTM, Data Parallel Virtual Machine</i> | Hensley |

Schedule



SIGGRAPH2007

Applications

3:05 GPGPU and Rasterization

Lefohn

Image processing, depth of field, shadows, relighting

3:30 Break

3:45 GPGPU and Raytracing

Krüger

GPU raytracing and hybrid systems

4:10 Geometric Computing

Govindaraju

Interactive collision detection

Schedule



SIGGRAPH2007

Physics

4:30 GPU Flow

Krüger

4:50 Game Physics

Green

Conclusion

5:15 Question-and-answer session

All

5:30 Wrap!

Course Evaluations



SIGGRAPH2007

http://www.siggraph.org/courses_evaluation

4 Random Individuals will win an ATI Radeon™ HD2900XT



GP GPU

AMD
Smarter Choice