GPUs: Engines for Future High-Performance Computing

John Owens
Assistant Professor, Electrical and Computer Engineering
Institute for Data Analysis and Visualization
University of California, Davis
Parallel Processing is the Future

**Major vendors supporting multicore**
- Intel, AMD

**Excitement about IBM Cell ...**

**Hardware support for threads**

**Interest in general-purpose programmability on GPUs**

**Why?**
Outline

Technology Trends
GPUs and General Purpose Programmability
Looking Towards the Future
Microprocessor Scaling is Slowing

- 52%/year
- ps/gate 19%
- Gates/clock 9%
- Clocks/inst 18%
- 19%/year

[courtesy of Bill Dally]
Performance of P6 Architecture

Courtesy of John Hennessy
Today’s Microprocessors

- Scalar programming model with no native data parallelism
  - SSE is the exception
- Few arithmetic units - little area
- Optimized for complex control
- Optimized for low latency not high bandwidth

Pentium III - 28.1M T
Today’s Microprocessors

- Scalar programming model with no native data parallelism
  - SSE is the exception
- Few arithmetic units - little area
- Optimized for complex control
- Optimized for low latency not high bandwidth

Pentium III - 28.1M T
Future Potential is Large

- 2001: 30:1
- 2011: 1000:1

[courtesy of Bill Dally]
## NVIDIA Historicals

<table>
<thead>
<tr>
<th>Chip</th>
<th>Tapeout</th>
<th>Production</th>
<th>Mtrans</th>
<th>Core Clk</th>
<th>Fill Rate</th>
<th>Vtx Rate</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>NV3</td>
<td>Feb-96</td>
<td>Jul-96</td>
<td>4</td>
<td>100</td>
<td>100</td>
<td>-</td>
<td>RIVA 128</td>
</tr>
<tr>
<td>NV4</td>
<td>Apr-98</td>
<td>Aug-98</td>
<td>7.5</td>
<td>120</td>
<td>250</td>
<td>-</td>
<td>RIVA TNT</td>
</tr>
<tr>
<td>NV5</td>
<td>Dec-98</td>
<td>Mar-99</td>
<td>9</td>
<td>175</td>
<td>350</td>
<td>-</td>
<td>RIVA TNT2</td>
</tr>
<tr>
<td>NV10</td>
<td>May-99</td>
<td>Sep-99</td>
<td>23</td>
<td>120</td>
<td>480</td>
<td>15</td>
<td>GeForce 256</td>
</tr>
<tr>
<td>NV15</td>
<td>Dec-99</td>
<td>Mar-00</td>
<td>25</td>
<td>166</td>
<td>664</td>
<td>21</td>
<td>GeForce 2 GTS</td>
</tr>
<tr>
<td>NV20</td>
<td>Sep-00</td>
<td>Apr-01</td>
<td>57</td>
<td>200</td>
<td>800</td>
<td>25</td>
<td>GeForce3</td>
</tr>
<tr>
<td>NV25</td>
<td>Sep-01</td>
<td>Dec-01</td>
<td>63</td>
<td>300</td>
<td>1200</td>
<td>75</td>
<td>GeForce 4 Ti</td>
</tr>
<tr>
<td>NV30</td>
<td>Aug-02</td>
<td>Dec-02</td>
<td>121</td>
<td>500</td>
<td>2000</td>
<td>187.5</td>
<td>GeForce FX 5800</td>
</tr>
<tr>
<td>NV40</td>
<td>Dec-03</td>
<td>Apr-04</td>
<td>222</td>
<td>400</td>
<td>6400</td>
<td>600</td>
<td>GeForce 6800</td>
</tr>
</tbody>
</table>

**From R128/GF256 t.o.:**

- **1.67x**
- **1.71x**
- **2.27x**
Long-Term Trend: CPU vs. GPU

![Graph showing the performance of CPUs and GPUs from 1997 to 2003]
Recent GPU Performance Trends

Programmable 32-bit FP multiplies per second

- **NVIDIA NV30, 35, 40**
- **ATI R300, 360, 420**
- **Pentium 4**

**GFLOPS**
- July 01
- Jan 02
- July 02
- Jan 03
- July 03
- Jan 04

Courtesy Pat Hanrahan/David Luebke
Recent GPU Performance Trends

Programmable 32-bit FP multiplies per second

- **NVIDIA NV30, 35, 40**: 36 GB/s
- **ATI R300, 360, 420**: 25 GB/s
- **Pentium 4**: 6 GB/s

Courtesy Pat Hanrahan/David Luebke
Recent GPU Performance Trends

Programmable 32-bit FP multiplies per second

- NVIDIA NV30, 35, 40
- ATI R300, 360, 420
- Pentium 4

GFLOPS:
- 6 GB/s, $260 (P4X3)
- 25 GB/s, $199 (X800)
- 36 GB/s, $205

Costs:
- $205
- $199

Years:
- July 01
- Jan 02
- July 02
- Jan 03
- July 03
- Jan 04

Courtesy Pat Hanrahan/David Luebke
Functionality Improves Too!

10 years ago:
- Graphics done in software

5 years ago:
- Full graphics pipeline

Today:
- 40x geometry, 13x fill vs. 5 yrs ago
- Programmable!

Programmable, data parallel processing on every desktop

Enormous opportunity to change the way commodity computing is done!
Outline

Technology Trends

GPUs and General Purpose Programmability

Looking Towards the Future
The Rendering Pipeline

Application → Geometry → Rasterization → Composite

GPU
The Rendering Pipeline

Application

Geometry

Rasterization

Composite

GPU

Compute 3D geometry
Make calls to graphics API
The Rendering Pipeline

- Application
- Geometry
- Rasterization
- Composite

Compute 3D geometry
Make calls to graphics API

Transform geometry from 3D to 2D (in parallel)
The Rendering Pipeline

Application

Geometry

Compute 3D geometry
Make calls to graphics API

Transform geometry from 3D to 2D (in parallel)

Rasterization

Generate fragments from 2D geometry (in parallel)

Composite

GPU
The Rendering Pipeline

- **Application**
  - Compute 3D geometry
  - Make calls to graphics API

- **Geometry**
  - Transform geometry from 3D to 2D *(in parallel)*

- **Rasterization**
  - Generate fragments from 2D geometry *(in parallel)*

- **Composite**
  - Combine fragments into image

**GPU**
The Programmable Rendering Pipeline

- Application
  - Geometry (Vertex)
    - Transform geometry from 3D to 2D; *vertex programs*
  - Rasterization (Fragment)
    - Generate fragments from 2D geometry; *fragment programs*
  - Composite
    - Combine fragments into image
NVIDIA GeForce 6800 3D Pipeline

Vertex

Triangle Setup

Z-Cull

Shader Instruction Dispatch

Fragment

Fragment Crossbar

L2 Tex

Memory Partition

Memory Partition

Memory Partition

Memory Partition

Courtesy Nick Triantos, NVIDIA
Detail of a single pixel shader pipeline

SIMD Architecture
Dual Issue / Co-Issue
FP32 Computation
Shader Model 3.0

4x pipes / 2x math/pipe
“NV35 has been characterized as a 4x2 / 8x0 architecture.
NV40 is 16x1 / 32x0 architecture.”

Courtesy Nick Triantos, NVIDIA
Detail of a single pixel shader pipeline

**Texture Filter**
- Bi / Tri / Aniso
- 1 texture @ full speed
- 4-tap filter @ full speed
- 16:1 Aniso w/ Trilinear (128-tap)
- FP16 Texture Filtering

**shader Unit 1**
- 4 FP Ops / pixel
- Dual/Co-Issue
- Texture Address Calc
- Free fp16 normalize + mini ALU

**Shader Unit 2**
- 4 FP Ops / pixel
- Dual/Co-Issue
- + mini ALU

4x pipes / 2x math/pipe

“NV35 has been characterized as a 4x2 / 8x0 architecture.

**NV40 is 16x1 / 32x0 architecture.”

**SIMD Architecture**
- Dual Issue / Co-Issue
- FP32 Computation
- Shader Model 3.0

**Output**
- Shaded Fragments

*Courtesy Nick Triantos, NVIDIA*
Maximizing Shader Performance

- **Apply programs to large numbers of elements**
  - Elements are divided into batches
  - One instruction is evaluated over every element in a batch

- **Minimize register usage**

- **Avoid control flow changes**
  - Branching is possible but inefficient

- **Do a lot of work per program**
  - Remedies overhead of passes

- **Trust your compilers**
  - Combination of compile-time and runtime optimization
Programming a GPU for Graphics
Programming a GPU for Graphics

- Application specifies geometry -> rasterized
Programming a GPU for Graphics

- Application specifies geometry -> rasterized
- Each fragment is shaded with SIMD program
Programming a GPU for Graphics

- Application specifies geometry -> rasterized
- Each fragment is shaded with SIMD program
- Shading can use values from texture memory
Programming a GPU for Graphics

- Application specifies geometry -> rasterized
- Each fragment is shaded with SIMD program
- Shading can use values from texture memory
- Image can be used as texture on future passes
Programming a GPU for GP Programs
Programming a GPU for GP Programs

- Draw a screen-sized quad
Programming a GPU for GP Programs

- Draw a screen-sized quad
- Run a SIMD program over each fragment
Programming a GPU for GP Programs

- Draw a screen-sized quad
- Run a SIMD program over each fragment
- “Gather” is permitted from texture memory
Programming a GPU for GP Programs

- Draw a screen-sized quad
- Run a SIMD program over each fragment
- “Gather” is permitted from texture memory
- Resulting buffer can be treated as texture on next pass
GPUs are fast (why?) ...

**Characteristics of computation permit efficient hardware implementations**
- High amount of parallelism ...
- ... exploited by graphics hardware
- High latency tolerance and feed-forward dataflow ...
- ... allow very deep pipelines
- ... allow optimization for bandwidth not latency

**Simple control**
- Restrictive programming model

**Competition between vendors**
... but GPU programming is hard

Must think in graphics metaphors
Requires parallel programming (CPU-GPU, task, data, instruction)
Restrictive programming models and instruction sets
Primitive tools
Rapidly changing interfaces
**Challenge: Programming Systems**

- **Programming Model**
  - High-Level Abstractions/Libraries
  - Low-Level Languages
  - Compilers

**Performance Analysis Tools**

**CPU**
- Scalar
  - STL, GNU SL, MPI, ...
  - C, Fortran, ...
  - gcc, vendor-specific, ...
  - gdb, vtune, Purify, ...
  - Lots
  → *applications*

**GPU**
- Stream? Data-Parallel?
  - Brook, Scout, sh, Glift
  - GLSL, Cg, HLSL, ...
  - Vendor-specific
  - Shadesmith, NVPerfHUD
  → *kernels*
Brook: General-Purpose Streaming Language

Stream programming model
- Treats GPU as streaming coprocessor
- Streams enforce data parallel computing
- Kernels encourage arithmetic intensity
- Streams and kernels explicitly specified

C with stream extensions

Open-source: www.sf.net/projects/brook/

Ian Buck et al., “Brook for GPUs: Stream Computing on Graphics Hardware”, Siggraph 2004
Virtual representation of memory: N-D array, stack, hash table, queue, ...

Physical representation of memory: 1D array
<table>
<thead>
<tr>
<th>Virtual representation of memory: N-D array, stack, hash table, queue, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstractions provided by library (STL, Boost, Glift ...)</td>
</tr>
<tr>
<td>Physical representation of memory: 1D array</td>
</tr>
</tbody>
</table>
Glift: Data Structures for GPUs (Lefohn)

Virtual representation of memory: N-D array, stack, hash table, queue, ...

Abstractions provided by library (STL, Boost, Glift ...)

Physical representation of memory: 1D array

Why?

- Algorithms expressed in natural data domain
- Algorithms separate from data structures
- Applications and library can be written separately
- Libraries promote reuse
- Vendors prefer higher level of abstraction
- Allow more complex applications!
Glift: Data Structures for GPUs (Lefohn)

Virtual Stacks

Abstract

Why?
- Algorithms
- Algorithms
- Applications
- Libraries
- Vendors prefer higher level of abstraction
- Allow more complex applications!
// Color map potential temperature from blue
// (cold) to red (hot) using hsVa() at all
// locations where there is water.
where (land == 0)
  image = hsVa(240 - 240 * norm(pt), 1, 1, 1);
else
  image = 0; // black
GPGPU Application Research

Image processing [Johnson/Frank/Vaidya, LLNL]
Alternate graphics pipelines [Purcell, Carr, Coombe]
Visual simulation [Harris]
Volume rendering [Kniss, Krüger]
Level set computation [Lefohn, Strzodka]
Numerical methods [Bolz, Krüger, Strzodka]
Molecular dynamics [Buck]
Databases [Sun, Govindaraju]
...
GPGPU Algorithm Research

Sorting
- “Oblivious” sorts ... sorting networks, bitonic sort

Searching
- Parallel searches are efficient
- Binary search is common (O(log n) runtime but uniform cost across elements)

Scatter
- Not supported on graphics hardware!
- Emulate using gather

Stream compaction
- Parallel algorithms (Hillis and Steele) quite useful!
What Runs Well on GPUs?

GPUs win when …

- *Limited data reuse*

<table>
<thead>
<tr>
<th></th>
<th>Memory BW</th>
<th>Cache BW</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4 3GHz</td>
<td>6 GB/s</td>
<td>44 GB/s</td>
</tr>
<tr>
<td>NV GF 6800</td>
<td>36 GB/s</td>
<td>--</td>
</tr>
</tbody>
</table>

- *High arithmetic intensity:* Defined as math operations per memory op
  - Attacks the memory wall - are all mem ops necessary?
- *Common error:* Not comparing against optimized CPU implementation
Outline

Technology Trends
GPU and General Purpose Programmability
Looking Towards the Future
Challenge: Programming Systems

Programming Model

High-Level Abstractions/Libraries

Low-Level Languages

Compilers

Performance Analysis Tools

CPU
Scalar
STL, GNU SL, MPI, ...
C, Fortran, ...
gcc, vendor-specific, ...
gdb, vtune, Purify, ...
Lots
→ applications

GPU
Stream? Data-Parallel?
Brook, Scout, sh, Glift
GLSL, Cg, HLSL, ...
Vendor-specific
Shadesmith, NVPerfHUD
None
→ kernels
Challenge: Mobile/Embedded Market

Why?
- UI, messaging/screen savers, navigation, gaming (location based)

Typical specs (cell-phone class):
- 200-800k gates, ~100 MHz, ~100 mW
- 1-10M vtx/s, 100+M frags/s

What’s important?
- Visual quality
- Power-efficient (ops/W)
  - Avoid memory accesses, unified shaders ...
- Low cost
Challenge: Power

Desktop:
- Double-width cards
- Workstation power supplies; draw power from motherboard

Mobile:
- Batteries improving 5-10% per year
- Ops/W most important
Grand Challenges

Architecture: Increase features and performance without sacrificing core mission

Interfaces: Abstractions, APIs, programming models, languages
- Many approaches needed
- Goal?: C programs compiling to dynamically-balanced CPU-GPU clusters
- Academic and research community

Applications: Killer app needed!
Our Big-Picture Goal

- IBM Cell
  Game Developers
- Multicore x86
  Internal, External
- GPUs
  Internal, Academia
- Stream Processors
  Internal, Academia
Our Big-Picture Goal

- IBM Cell Game Developers
- Multicore x86 Internal, External
- GPUs Internal, Academia
- Stream Processors Internal, Academia

Abstraction & Generalization - Academia
Our Big-Picture Goal

- IBM Cell
  - Game Developers

- Multicore x86
  - Internal, External

- GPUs
  - Internal, Academia

- Stream Processors
  - Internal, Academia

Abstraction & Generalization - Academia

Education
Conclusion
A highly parallel language used by non-experts.

Power of notation
Good:
    make it easier to express yourself
Better:
    hide stuff you don't care about
Best:
    hide stuff you do care about

Give the language a purpose.
Rob Pike on Languages

Conclusion
A highly parallel language used by non-experts.

Power of notation
Good:
make it easier to express yourself
Better:
hide stuff you don't care about
Best:
hide stuff you do care about

Give the language a purpose.
Conclusions

The future of the desktop is parallel
  • We just don’t know what kind of parallel

A case for the GPU
  • Interaction with the world is visual
  • GPUs have a well-established programming model
  • NVIDIA shipped 100M units last year
Acknowledgements

Nick Triantos, Craig Kolb (NVIDIA)
Mark Segal (ATI)
Kari Pulli (Nokia)
Aaron Lefohn (UC Davis)
Ian Buck, Tim Purcell, Pat Hanrahan, Bill Dally (Stanford)
Pat McCormick (Los Alamos)

Funding: DOE Office of Science, Los Alamos National Laboratory, ChevronTexaco, UC MICRO, UC Davis
For more information ...

GPGPU home: http://www.gpgpu.org/
  • Mark Harris, UNC/NVIDIA

GPU Gems (Addison-Wesley)
  • Vol 1: 2004; Vol 2: 2005
  • Vol 2 has NVIDIA GeForce 6800

Conferences: Siggraph, Graphics Hardware, GP^2
  • Course notes: Siggraph ‘04/‘05, IEEE Visualization ‘04

University research: Caltech, CMU, Duisberg, Illinois, Purdue, Stanford, SUNY Stonybrook, Texas, TU München, Utah, UBC, UC Davis, UNC, Virginia, Waterloo