ATI X1k Series

Mike Houston, Stanford University
A Little Background
Threads

In today’s talk…

a thread is a hardware execution context

- Program counter
- Registers
- Fragment/quad identifier
- Screen space position
Single-threaded CPU

- Run one thread as fast as possible
The applications we care about have abundant parallelism.

- Graphics
- Image/media processing
- Vision
- Scientific computing
Throughput matters

- Intel Pentium D: 2 cores
Throughput matters

- Sony-Toshiba-IBM CELL: 9 cores
<table>
<thead>
<tr>
<th>Commodity parallel processors</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Pentium D</td>
<td>2 cores</td>
</tr>
<tr>
<td>AMD Opteron</td>
<td>2 cores</td>
</tr>
<tr>
<td>SUN Niagara</td>
<td>8 cores</td>
</tr>
<tr>
<td>IBM CELL</td>
<td>9 cores</td>
</tr>
<tr>
<td>NVIDIA 7800 GTX</td>
<td>24 pixel pipelines</td>
</tr>
<tr>
<td>ATI X1800 XT</td>
<td>16 pixel pipelines (16 ALUs)</td>
</tr>
<tr>
<td>ATI X1900XT/X</td>
<td>16 pixel pipelines (48 ALUs)</td>
</tr>
</tbody>
</table>
Stalls slow a thread’s execution down

- Branches, data dependencies
  - a few cycles

- Latency of memory access
  - Worst case hundreds to thousands of cycles
Universal problem: memory latency

- You want to read data from “memory”

- The data can be stored anywhere
  - Is it in a register?
  - Is it in cache?
  - Is it off chip in DRAM?
  - Is it on a different machine?

- Waiting on memory cripples performance
Streaming vs. Threading

- Key property of a parallel architecture is how it attempts to keep processing units busy

- **Hardware multi-threading**
  - Hide effect of stalls by doing something else

- **Streaming**
  - Never stall. Only allow loads from memory that can be accessed without stalling processor
Hardware multi-threading

- Multiple thread contexts per processing core
- One thread runs on a core at a time
Hardware multi-threading

- Execute one thread until stall, then switch to next thread
Hardware multi-threading

- Intel Hyper-threading: 2 threads (sort of)
- SUN Niagara: 4 threads per core (32 total)

- GPU fragment processing engines are **heavily multi-threaded**!
  - Hundreds of threads active per fragment pipeline

(Much more on this later)
Hardware multi-threading scalability

- Need enough threads to cover latency of stalls
- Need storage for all the execution contexts
- Need more contexts as compute/memory latency gap grows
- Does not decrease bandwidth requirements!
Streaming

- Limit data access to on-chip storage
  - Accessible without stalling processor
  - Introduces different address spaces

- Overlap current computation with loading of data for future computations
  - A.k.a. prefetch
  - Block transfer all data needed for upcoming computation (think DMA)
How to stream

1. Preload batch of data
2. Compute on data
3. Initiate write of results
4. Compute on next batch (which should be loaded)
Need “arithmetic intensity”

- Using data faster than it can be loaded causes stalls
Streaming summary

- 1 HW thread context per streaming core
- Num contexts scales with number of cores
  - Not number cores $\times$ latency
- Fundamentally different programming style
  - Naming working sets up front is tricky
Recap

- Streaming and threading utilize parallelism to hide memory latency
- Parallel processing creates enormous bandwidth demands (more processors)
- Stalls occur when processors request data at a rate faster than memory can provide it. Bandwidth is the scarcest resource
A closer look at GPUs
GPUs

- Rasterizing geometry generates fragments
- GPUs run shader programs on fragments
- Mental model: processor runs shader program on a stream of fragments to generate a stream of output pixels
Modern GPUs have many fragment processing cores.
GPU multi-threading

- Each fragment processor maintains hundreds of active fragment contexts (think threads)

- Constrained multi-threading
  - Groups of threads must stay in lockstep (same program counter)
GPU multi-threading (version 1)

- Change threads each cycle (round robin)

```
frag1
frag2
frag3
frag4
```

```
instr1
   ↓
   ↓
   ↓

instr2
   ↓
   ↓
   ↓

instr3
   ↓
   ↓
   ↓
```
GPU multi-threading (version 2)

- Change thread on stall (round robin)
Implications of GPU multi-threading

- Programming model defines shader invocations to be logically independent

- But effects of threaded fragment processing must be considered for performance
  - Coherence of control flow
  - Coherence of memory access
Effects of PC coherence

- Stall due to divergence of control

```c
if (x == 0)
{
    y += 2.0;
}
else
{
    y *= 2.0;
}
```
Effects of access coherence amplified

- Incoherence: cache miss by one fragment might hold up entire group of threads

- Coherence: cache miss by one fragment pre-fetches line for subsequent threads in group
3D Architecture
Architectural Highlights

Focus on Efficiency
- Reduce idle time and latency
- Improve memory access management

Shader Model 3.0
- High performance dynamic flow control
- Full speed 128-bit rendering

Image Quality
- High dynamic range rendering
- Adaptive anti-aliasing
- Improved texture filtering quality

Flexibility and Scalability
- Decoupling processing units from rigidly defined pipelines
Radeon X1800 3D Architecture

- 16 Pixel Shader Processors
  - Ultra-Threading Dispatch Processor
  - 4 Shader Cores

- 8 Vertex Shader Processors
- 16 Texture Address Units
- 16 Texture Units
- 16 Render Back-End Units
Pixel Shader Processors

Quad Pixel Shader Core

Pixel Shader Processors

Per Clock Cycle:
1 vec3 ADD + input modifier
1 scalar ADD + input modifier
1 vec3 ADD/MUL/MADD
1 scalar ADD/MUL/MADD
1 flow control instruction

Texture Address Units
1 texture address instructions per unit per clock cycle
Shader Model 3.0 Features

- **Dynamic Flow Control**
  - Branching (IF…ELSE), Looping, Subroutines

- **128-bit Floating Point Processing**
  - For pixel and vertex shaders

- **Longer Shaders**
  - Billions of instructions possible with flow control
Dynamic Flow Control

- Allows different paths through the same shader to be executed on adjacent pixels

- Provides significant optimization opportunities
  - Skip parts of a shader that don’t need to be executed (“early out”)
  - Avoid state change overhead by combining multiple related shaders into a single one
  - Allows GPU to execute CPU code more effectively

- Can interfere with parallelism
  - Redundant computation can often reverse any benefits of using flow control
Accelerating Flow Control

1. Large number of threads
2. Intelligent thread selection
3. Small thread size
4. Dedicated flow control logic
Ultra-Threading

**Sophisticated, Large Scale Multi-Threading**
- Hundreds of simultaneous threads across multiple cores
- Each thread can perform up to 6 different shader instructions on 4 pixels per clock cycle

**Small Thread Sizes**
- 16 pixels per thread in Radeon X1800
- Fine-grain parallelism

**Fast Branch Execution**
- Dedicated units handle flow control with no ALU overhead

**Large, Multi-Ported Register Arrays**
- Enables fast thread switching

---

**Ultra-Threading Benefits**
Hides texture fetch latency
Minimizes shader processor idle time and wasted cycles
Accelerates dynamic flow control with SM3.0

*Mike Houston - Stanford University Graphics Lab*
Thread Size and Dynamic Branching Efficiency

// Sample Shader
if (shadow)
{
    Process □,□
}
else
{
    Process □,□
}

Shadow Mapping

Thread Size
4x4
(16 pixels)
Efficient

Thread Size
16x16
(256 pixels)
Less Efficient

Thread Size
64x64
(4096 pixels)
Inefficient
Fast Branch Execution

- Branch Execution Unit eliminates overhead
- Shaders with flow control can execute in fewer clock cycles
128-bit Floating Point Processing

- Optimal performance without reducing precision
  - General Purpose Register Array has ample storage and read/write bandwidth
  - All shader calculations use 128-bit floating point precision, at full speed

- Effective handling of non-pixel operations
  Examples:
  - Vertex processing – render to vertex buffer
  - Parallel data processing – off-loading work from CPU

- Maintains precision in long shaders
Vertex Engine

- 8 Vertex Shader Processors
  - Each can handle 2 shader instructions per clock
  - 10 billion instructions per second

- Upgraded to support SM3.0
  - Dynamic flow control
  - 1,024 instructions (practically unlimited with flow control)
  - More temporary registers
Flexible Architecture

- Focus on shader processing
  - Primary performance indicator for games going forward

- Radeon X1000 architecture de-couples components of the rendering pipeline
  - Allows greater flexibility in GPU design
  - Number of each component can be varied independently
  - More optimal design choices for any given silicon budget

<table>
<thead>
<tr>
<th></th>
<th>Pixel Shader Processors</th>
<th>Vertex Shader Processors</th>
<th>Texture Units</th>
<th>Render Back-Ends</th>
<th>Z Compare Units</th>
<th>Max. Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radeon X1800</td>
<td>16</td>
<td>8</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>512</td>
</tr>
<tr>
<td>Radeon X1600</td>
<td>12</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>128</td>
</tr>
<tr>
<td>Radeon X1300</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>128</td>
</tr>
</tbody>
</table>
3D Architecture Comparison

Radeon X1800
- 16 Shader Processors

Radeon X1600
- 12 Shader Processors

Radeon X1300
- 4 Shader Processors

Mike Houston - Stanford University Graphics Lab
3D Architecture Comparison

- **Radeon X1800**
  - ~320M transistors
  - ~260 sq.mm@90nm
  - 16 Shader Processors

- **Radeon X1600**
  - ~160M transistors
  - ~130 sq.mm@90nm
  - 12 Shader Processors

- **Radeon X1300**
  - ~100M transistors
  - ~95 sq.mm@90nm
  - 4 Shader Processors

*Mike Houston - Stanford University Graphics Lab*
What about the X1900 (R580)?

Same architecture as X1800, but 3X the ALUs

- Triple the ALUs and raw compute for “small” transistor increase
- MUCH cheaper than more quad pipes

~380M transistors
~340 sq.mm@90nm
48 Shader Processors

Mike Houston - Stanford University Graphics Lab
What about the X1900 (R580)?

- 3X ALUs
- 3X Register File
- More complex dispatch logic
Decoupled ALU, Branch, Texture allows you to add more of different resources
3D Architecture Summary

- Ultra-Threaded Pixel Shader Engine
- Efficient Dynamic Flow Control
- Full Speed 128-bit Floating Point Processing
- Consistent Performance
- Flexible, Scalable Design
Memory Architecture
Design Goals

“Future Proof”
- Software upgradeable
- Supports different (and future) memories

Ease Physical Implementation -> Faster Clocks
- Ring Bus

Reduce Memory Traffic
- Caches
- Improved Z

Increase Random Access Performance
- More Channels
- More Banks
Ring Bus Memory Controller

Supports today’s fastest graphics memory devices
- GDDR3, 48+ GB/sec
- GDDR4, ?

512-bit Ring Bus
- Simplifies layout and enables extreme memory clock scaling

New Cache Design
- Fully Associative for more optimal performance

Improved Hyper Z
- Better compression and hidden surface removal

Programmable Arbitration Logic
- Maximizes memory efficiency
- Can be upgraded via software

Mike Houston - Stanford University Graphics Lab
Two internal 256-bit rings
- Run in opposite directions to minimize latency
- Return requested data to clients
- Memory writes use crossbar switch

Circle around the periphery of the chip
- Reduces routing complexity
- Permits higher clock speeds

One ring stop per pair of memory channels
- Linked directly to memory interface
“R4XX” Memory Controller
Extremely high wire density around memory controller

- increases routing complexity.
- power and thermal issues impedes clock speed.
Low density wiring around X1K Memory controller
X1K RING BUS
Conceptual Operation

Client makes a memory request
Client makes a memory request
X1K RING BUS
Conceptual Operation

Client makes a memory request
Client makes a memory request
Memory Controller determines the DRAM to service the request.
Memory Controller determines the DRAM to service the request.
Memory Controller determines the DRAM to service the request.
Memory Controller determines the DRAM to service the request.
The return data is placed on the Ring Bus to reach the Ring Stop nearest to the Client.
The return data is placed on the Ring Bus to reach the Ring Stop nearest to the Client.
Programmable Arbitration Logic

Prioritizes memory access requests
- Predicts impact of each request on overall performance

Uses feedback system to maximize memory and GPU efficiency

Programmable parameters
- Can be tuned via driver updates
Clients make memory requests
X1K Memory Controller Arbitration

Desperation Meter

Client Weights

Dual DRAM Sequencer

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

Memory Parameters
Memory Controller determines priority based on

Client Weights

Client desperation

Request Efficiency
X1K Memory Controller Arbitration

Desperation Meter

Client

Client

Client

Client

Client Weights

Dual DRAM Sequencer

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

Memory Parameters
X1K Memory Controller Arbitration

Desperation Meter

Client

Client

Client

Client

Client Weights

Memory Parameters

Dual DRAM Sequencer

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM
X1K Memory Controller Arbitration

Desperation Meter

Client

Client Weights

Dual DRAM Sequencer

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

Memory Parameters
X1K Memory Controller Arbitration

Desperation Meter

Client

Client Weights

Dual DRAM Sequencer

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

Memory Parameters
X1K Memory Controller Arbitration

Desperation Meter

Client

Client

Client

Client Weights

Dual DRAM Sequencer

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

Memory Parameters
X1K Memory Controller Arbitration

Desperation Meter

Client

Client

Client

Client Weights

Memory Parameters

Dual DRAM Sequencer

DRAM

DRAM

DRAM

DRAM

DRAM
X1K Memory Controller Arbitration

Desperation Meter

Client

Client Weight

Dual DRAM Sequencer

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

Memory Parameters
X1K Memory Controller Arbitration

Desperation Meter

Client

Client

Client

Client Weights

Memory Parameters

Dual DRAM Sequencer

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM

DRAM
X1K memory controller is a sophisticated programmable device.
X1K memory controller is a sophisticated programmable device

Client weights are programmable per application basis
X1K memory controller is a sophisticated programmable device

Client weights are programmable per application basis

Memory request efficiency is programmable per application basis as well.
Memory Channels - 4x Improvement in Random Access over X850

Radeon X1800
- 8x32-bit channels
- 8 Banks Per Dram

Radeon X850
- 4x64-bit channels
- 4 banks Per Dram
Fully Associative Caches

- Cache lines can map to any location in external memory
- Earlier designs used Direct Mapped & N-Way Associative Caches
- Could only access limited blocks of external memory

Texture, Color, Z & Stencil caches are all now fully associative

- Reduces memory bandwidth requirements
- Minimizes cache contention stalls
- Optimized game performance
- Gains up to 25% clock for clock in fill/bandwidth bound cases

Mike Houston - Stanford University Graphics Lab
Cache Performance

X1800 Z Cache Misses Relative to X850

Cache Misses

Battlefield 2
Far Cry
3DMark05 GT1
X850 Baseline

Mike Houston - Stanford University Graphics Lab
Cache Performance

X1800 Texture Cache Misses Relative to X850

- Battlefield 2
- Far Cry
- HL2: Lost Coast
- X850 Baseline
Z-Buffer improvements

**Improved Hierarchical Z Buffer**
- Detects and discards hidden pixels before shading
- Important in scenes with heavy overdraw (i.e. overlapping objects)
- New technique uses floating point for improved precision
- Catches up to 60% more hidden pixels than X850

**Improved Z Compression**
- Z Buffer data is typically the largest user of memory bandwidth
- Bandwidth can be reduced by up to 8:1 using lossless compression
- New method achieves higher compression ratios more often
Memory Controller Performance

- New technology benefits most apparent in most bandwidth-demanding situations
  - High resolutions (1600x1200 and up)
  - Anti-Aliasing (4x and 6x modes, Adaptive AA)
  - Anisotropic Filtering (8x and 16x Quality AF modes)
  - HDR

- Extended modes
  - Fetch4: fetch 4 samples from depth map at once (R580)
  - Scatter
X1800 Die shot
GPGPU and the ATI X1800
GPGPU on the ATI X1800

- 32-bit floating point ("IEEE inspired")
  - Simplifies precision issues in applications
- Long programs
  - We can now handle larger applications
  - 512 static instructions
  - Effectively unlimited dynamic instructions
- Branching and Looping
  - No performance cliffs for dynamic branching and looping
  - Fine branch granularity: ~16 fragments
- Faster upload/download
  - 50-100% increase in PCIe bandwidth over last generation
GPGPU on the ATI X1800, cont.

- Advanced memory controller
  - Latency hiding for streaming reads and writes to memory
    - With enough math ops you can hide all memory access!
  - Large bandwidth improvement over previous generation

- Scatter support (a[i] = x)
  - Arbitrary number of float outputs from fragment processors
  - Uncached reads and writes for register spilling

- F-Buffer
  - Support for linearizing datasets
  - Store temporaries “in flight”
GPGPU on the ATI X1800, cont.

- Flexibility
  - Unlimited texture reads
  - Unlimited dependent texture reads
  - 32 hardware registers per fragment

- 512MB memory support
  - Larger datasets without going to system memory
Image Quality
Image Quality Feature Highlights

HDR with Anti-Aliasing
Adaptive Anti-Aliasing
Improved Texture Filtering Quality
3Dc+

Image from Splinter Cell 3: Chaos Theory

Mike Houston - Stanford University Graphics Lab
High Dynamic Range (HDR)

- Dynamic Range defines ratio between highest and lowest value that can be represented
  - More bits of data = greater dynamic range
  - Floating point has much greater range than integer
  - Dynamic range examples:
    - 8-bit integer – 256:1
    - 10-bit integer – 1,024:1
    - 16-bit integer – 65,536:1
    - 16-bit floating point – 2.2 trillion:1

- Most displays can only recognize values between 0 and 255 (i.e. 8 bits per color component)
  - Requires tone mapping to preserve detail
  - Light bloom and lens flare effects can help convey a sense of high brightness

- High Dynamic Range (HDR) rendering takes advantage of color formats with greater dynamic range
  - Produces more realistic images
High Dynamic Range (HDR)

- Shaders can add a light bloom or glow effect around very bright objects for increased realism.
- Takes advantage of >8 bits per color component.

Brightness:

- 10,000
- 1,000
- 255
- 196
- 128

Maximum possible brightness using 8-bit integer color.

High brightness (represented using light glow effect).
HDR Example

Image from Far Cry

High Dynamic Range
Adaptive Anti-Aliasing

- Combines image quality of supersampling with speed of multisampling
- Most visible improvement on partly transparent surfaces (foliage, fences, grates, etc.)
- Works seamlessly with all other anti-aliasing technologies
Adaptive Anti-Aliasing

Images from Half Life 2 by Valve Software

Mike Houston - Stanford University Graphics Lab
Anti-Aliasing Technology

- High Performance 6x MSAA
- Sparse Sample Patterns
- Gamma Correct Blending
- Adaptive AA
- Temporal AA
- 14x Super AA (Crossfire)
16X AF with Area-Aniso algorithm
High quality normal map compression
- Up to 4:1
- Works on any two-channel texture format

Latest version adds support for single-channel textures
- 2:1 compression
- Useful for luminance maps, shadow maps, material properties, HDR textures, and more
**Architectural Summary**

**Focus on Efficiency**
- Reduce idle time and latency
- Improve memory access management

**Shader Model 3.0 “Done Right”**
- High performance dynamic flow control
- Full speed 128-bit rendering

**Driving Image Quality Forward**
- Advanced high dynamic range rendering
- Adaptive anti-aliasing
- Improved texture filtering quality

**Flexibility and Scalability**
- Decoupling processing units from rigidly defined pipelines
GPUBench
GPUBench

- Open source: gpubench.sf.net
- Provides significant detail about GPU subsystem performance
- Lots of tests with lots of knobs
- What can we already do?
  - Find texture latencies
  - Instruction issue rates
  - Branch efficiency
  - Early-Z
  - Cache sizes, layouts, associativity
  - ...
- Precision tests
- GL and DX (new) runtimes
Texture Fetch Bandwidth

- **6800 Ultra**
- **X1800XT**
- **7800GTX**
- **X1900XTX**
Compute Rates - ATI

ATI X1800XT

~80GFlops (MAD)
~120GFlops (ADD+MAD)

ATI X1900XT

~240GFlops (MAD)
~360GFlops (ADD+MAD)

Mike Houston - Stanford University Graphics Lab
Texture Fetch Latency – Cache Hit

Nvidia 6800 Ultra

ATI X1800XT

Nvidia 7800GTX

ATI X1900XTX
When 32-bit isn’t…

Nvidia 7800GTX

ATI X1K
Acknowledgements

- ATI
  - Raja Koduri
  - Eric Demers
  - Joe Macri
  - Mark Segal
  - Bob Drebin
  - Mark Leather

- Stanford Graphics Lab (Brook and GPUBench)
  - Kayvon Fatahalian (thanks for the background slides!)
  - Tim Foley
  - Daniel Horn
  - Jeremy Sugerman
  - Ian Buck
  - Tim Purcell
Where to learn more

- ATI whitepapers
- Beyond3D
  - Very good technical analysis and explanations
- GPGPU.org
- GPUBench
  - Write more tests
  - Tweak current tests
- Internships!
  - Both ATI and Nvidia have great programs
Questions?

mhouston@graphics.stanford.edu

http://graphics.stanford.edu/~mhouston