Predicting Performance
Where to begin?

You’re done writing code, now what?

Does it work?

Is it fast?

What does fast mean?

60x faster than the CPU is pretty good
• What are you leaving on the table?
• How close is it to theoretical?
Predicting Performance

It is very useful to predict theoretical performance when working on shaders.

Spreadsheets are quite useful for this:
• Compute theoretical performance
• Compute pixels per clock, etc
• Easy to see how close an implementation is to peak performance

Quite easy with CAL/Brook+ since you can get the ISA even if you use a high-level language.
Breaking down GPGPU performance
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering.
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering.
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering.
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering.

ALU  TEX  MEM  ALU
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering...

ALU  TEX  MEM  ALU  ...
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering.
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering.

ALU  TEX  MEM  ALU  ...  MEM

Theoretical performance can be calculated.
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering.

Theoretical performance can be calculated

What we know...
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering.

![Diagram showing ALU, TEX, MEM, ALU, ... MEM]

Theoretical performance can be calculated

What we know...

# of ALU, TEX, and Write operations
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering.

Theoretical performance can be calculated.

What we know...

- # of ALU, TEX, and Write operations
- GPUShaderAnalyzer great for this
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering.

Theoretical performance can be calculated.

What we know...

# of ALU, TEX, and Write operations

- GPUShaderAnalyzer great for this

What do we need to find out?
Calculating Theoretical Performance

Radeon X1900XT (R580)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks
Calculating Theoretical Performance

Radeon X1900XT (R580)
• 8:48:16:16 (VER:ALU:TEX:ROP)
• 256 bit memory bus
• 625 MHz Engine / 750 Mhz Memory Clocks
Calculating Theoretical Performance

Radeon X1900XT (R580)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:
- 1 ALU Shader
Calculating Theoretical Performance

Radeon X1900XT (R580)
• 8:48:16:16 (VER:ALU:TEX:ROP)
• 256 bit memory bus
• 625 MHz Engine / 750 Mhz Memory Clocks

Example:
• 1 ALU Shader

\[
\frac{(\#\text{pixels}) \times (\#\text{ALU instructions})}{(\text{ALU}/\text{clk}) \times (3D \text{ engine speed})}
\]
Calculating Theoretical Performance

Radeon X1900XT (R580)
• 8:48:16:16 (VER:ALU:TEX:ROP)
• 256 bit memory bus
• 625 MHz Engine / 750 Mhz Memory Clocks

Example:
• 1 ALU Shader

\[
\frac{(#\text{pixels}) \times (#\text{ALU instructions})}{(ALU/clk) \times (3D \text{ engine speed})} = \frac{(1920 \times 1088) \times (1)}{(48) \times (625 \text{ MHz})}
\]
Calculating Theoretical Performance

Radeon X1900XT (R580)
• 8:48:16:16 (VER:ALU:TEX:ROP)
• 256 bit memory bus
• 625 MHz Engine / 750 Mhz Memory Clocks

Example:
• 1 ALU Shader

\[
\frac{(#\text{pixels}) \times (#\text{ALU instructions})}{(\text{ALU/clk}) \times (3D \text{ engine speed})} = \frac{(1920 \times 1088) \times (1)}{(48) \times (625 \text{ MHz})} = 0.07 ms
\]
Calculating Theoretical Performance

Radeon X1900XT (R580)
• 8:48:16:16 (VER:ALU:TEX:ROP)
• 256 bit memory bus
• 625 MHz Engine / 750 Mhz Memory Clocks

Example:
• 1 TEX Shader
Calculating Theoretical Performance

Radeon X1900XT (R580)
• 8:48:16:16 (VER:ALU:TEX:ROP)
• 256 bit memory bus
• 625 MHz Engine / 750 Mhz Memory Clocks

Example:
• 1 TEX Shader

\[
\frac{(\#\text{pixels}) \times (\#\text{TEX instructions})}{(\text{TEX}/\text{clk}) \times (3D \text{ engine speed})}
\]
Calculating Theoretical Performance

Radeon X1900XT (R580)
• 8:48:16:16 (VER:ALU:TEX:ROP)
• 256 bit memory bus
• 625 MHz Engine / 750 Mhz Memory Clocks

Example:
• 1 TEX Shader

\[
\frac{(\#\text{pixels}) \times (\#\text{TEX instructions})}{(\text{TEX}/\text{clk}) \times (3D \text{ engine speed})} = \frac{(1920 \times 1088) \times (1)}{(16) \times (625 \text{ Mhz})}
\]
Calculating Theoretical Performance

Radeon X1900XT (R580)
• 8:48:16:16 (VER:ALU:TEX:ROP)
• 256 bit memory bus
• 625 MHz Engine / 750 Mhz Memory Clocks

Example:
• 1 TEX Shader

\[
\frac{(\#\text{pixels}) \times (\#\text{TEX instructions})}{(\text{TEX/clk}) \times (3D \text{ engine speed})} = \frac{(1920 \times 1088) \times (1)}{(16) \times (625 \text{ Mhz})} = 0.21 \text{ ms}
\]
Calculating Theoretical Performance

Radeon X1900XT (R580)
- \(8:48:16:16\) (\text{VER:ALU:TEX:ROP})
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:
- Memory Performance - 1 Byte in 1 Byte out (Copy)
Calculating Theoretical Performance

Radeon X1900XT (R580)
• 8:48:16:16 (VER:ALU:TEX:ROP)
• 256 bit memory bus
• 625 MHz Engine / 750 Mhz Memory Clocks

Example:
• Memory Performance - 1 Byte in 1 Byte out (Copy)

\[
\frac{(#pixels \times (in + out \text{ bits per pixel}) \times (bus \times (memory \text{ speed}))}{(bus \times (memory \text{ speed}))}
\]
Calculating Theoretical Performance

Radeon X1900XT (R580)
• 8:48:16:16 (VER:ALU:TEX:ROP)
• 256 bit memory bus
• 625 MHz Engine / 750 Mhz Memory Clocks

Example:
• Memory Performance - 1 Byte in 1 Byte out (Copy)

\[
\frac{\text{(\#pixels)} \times (\text{in + out bits per pixel})}{(\text{bus}) \times (\text{memory speed})} = \frac{(1920 \times 1088) \times (16 \text{ bits})}{(256) \times (750 \text{ Mhz x 2DDR})}
\]
Calculating Theoretical Performance

Radeon X1900XT (R580)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:
- Memory Performance - 1 Byte in 1 Byte out (Copy)

\[
\frac{(#\text{pixels}) \times (in + out \text{ bits per pixel})}{(bus) \times (memory \text{ speed})} = \frac{(1920 \times 1088) \times (16 \text{ bits})}{(256) \times (750 \text{ Mhz} \times 2 \text{ DDR})} = 0.085 \text{ ms}
\]
Calculating Theoretical Performance

Overall Theoretical Performance
• $\text{max}(\text{ALU}, \text{TEX}, \text{Memory})$
  – each operation happens in parallel
• $\text{max}(0.07, 0.21, 0.085)$
• 0.21 ms - Texture bound
  – Texture units is the limitation
Calculating Theoretical Performance

Overall Theoretical Performance
• $\max(\text{ALU,TEX,Memory})$
  – each operation happens in parallel
• $\max(0.07, 0.21, 0.085)$
• 0.21 ms - Texture bound
  – Texture units is the limitation

Remember, this is only a starting point!
• ALU and TEX calculation is reasonable
  – Actually usually very close
• Memory assumes peak
  – depends on access pattern, etc
• Conditional operations can complicate things
## Decoder Ring

<table>
<thead>
<tr>
<th>GPU</th>
<th>Engine Clock</th>
<th>Memory Clock</th>
<th>Memory Width</th>
<th>ALUs</th>
<th>TEX</th>
<th>ROPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R600</td>
<td>750</td>
<td>850</td>
<td>512</td>
<td>64</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>HD2900</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R610</td>
<td>700</td>
<td>1100</td>
<td>64</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>HD2600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R630</td>
<td>800</td>
<td>1100</td>
<td>128</td>
<td>24</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>HD2400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R670</td>
<td>750</td>
<td>1150</td>
<td>256</td>
<td>64</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>HD3870</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Decoder Ring

<table>
<thead>
<tr>
<th>GPU</th>
<th>Engine Clock</th>
<th>Memory Clock</th>
<th>Memory Width</th>
<th>ALUs</th>
<th>TEX</th>
<th>ROPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R600</td>
<td>750</td>
<td>850</td>
<td>512</td>
<td>64</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>HD2900</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R610</td>
<td>700</td>
<td>1100</td>
<td>64</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>HD2600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R630</td>
<td>800</td>
<td>1100</td>
<td>128</td>
<td>24</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>HD2400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R670</td>
<td>750</td>
<td>1150</td>
<td>256</td>
<td>64</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>HD3870</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How long would a 1 ALU shader that outputs 1 byte take on R610? What’s the bottleneck?
Decoder Ring

<table>
<thead>
<tr>
<th>GPU</th>
<th>Engine Clock</th>
<th>Memory Clock</th>
<th>Memory Width</th>
<th>ALUs</th>
<th>TEX</th>
<th>ROPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R600</td>
<td>750</td>
<td>850</td>
<td>512</td>
<td>64</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>HD2900</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R610</td>
<td>700</td>
<td>1100</td>
<td>64</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>HD2600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R630</td>
<td>800</td>
<td>1100</td>
<td>128</td>
<td>24</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>HD2400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R670</td>
<td>750</td>
<td>1150</td>
<td>256</td>
<td>64</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>HD3870</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How long would a 2 ALU shader that outputs 1 byte take on R610? What’s the bottleneck?
The Value of System / Remote Memory
Radial Correction

System Requirements:
• Capture video from one or more cameras.
• Transfer images to GPU
• Convert bayer-pattern images to RGB images
• Remove lens distortions
• Return to host for further processing

System needs to use limited power
• Mobile GPU

Want to minimize correction time
• Images further processed in a large real-time system
Naive Implementation

- Input images (system memory)
- Output images (system memory)
Naive Implementation

Input images (system memory) → ??? → Output images (system memory)
Naive Implementation

Input images (system memory) → ??? → Output images (system memory)

Input images (GPU memory) → copy

University of Central Florida
Naive Implementation

- Input images (system memory)
- Output images (system memory)
- Input images (GPU memory)
- Intermediate (GPU memory)

Copy

Debayer kernel

???
Naive Implementation

- **Input images** (system memory)
- **Output images** (system memory)
- **Input images** (GPU memory)
- **Output images** (GPU memory)
- **Intermediate** (GPU memory)

- **Copy**
- **Debayer kernel**
- **Undistortion kernel**
Naive Implementation

Input images (system memory) → ??? → Output images (system memory)

Input images (GPU memory) ↓ copy

Debayer kernel

Intermediate (GPU memory)

Undistort kernel

Output images (GPU memory) ↑ copy
Faster Implementation

Input images (system memory) -> ??? -> Output images (system memory)
Faster Implementation

Input images (system memory)

Debayer kernel

Intermediate (GPU memory)

Output images (system memory)
Faster Implementation

Input images (system memory) → ??? → Output images (system memory)

Debayer kernel → Intermediate (GPU memory) → Undistort kernel
Faster Implementation

Accessing system memory directly is **1.8x faster** than naive implementation!
(2.74 ms vs. 4.96 ms)
Caveat: System Memory

Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics
Caveat: System Memory

Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics
Caveat: System Memory

Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics
Caveat: System Memory

Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics
Caveat: System Memory

Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics
Caveat: System Memory

Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics
Caveat: System Memory

Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics

Not a system memory “friendly” traversal
“Forcing” Raster pattern

“Forcing” the rasterizer to be more friendly with system memory traversal

- Use strips of geometry (CTM can directly stamp quads)
“Forcing” Raster pattern

“Forcing” the rasterizer to be more friendly with system memory traversal

- Use strips of geometry (CTM can directly *stamp* quads)
Effect of “Forcing” Raster pattern

2048x2048 float32x4 “copy” shader
• Reads input in local GPU memory, writes to system memory
• RD580 with an R580

Full screen quad time = 45.51 ms
• ~1.5 GB/sec readback

“Raster-Blocks” time = 26.53 ms
• ~2.5 GB/sec readback

Technique could also be used to optimize shaders with non-standard to local memory accesses

Need to be aware of how threads are assigned to wavefronts / warps / vectors
Effect of “Forcing” Raster pattern

2048x2048 float32x4 “copy” shader
• Reads input in local GPU memory, writes to system memory
• RD580 with an R580

Full screen quad time = 45.51 ms
• ~1.5 GB/sec readback

“Raster-Blocks” time = 26.53 ms
• ~2.5 GB/sec readback

Technique could also be used to optimize shaders with non-standard to local memory accesses

Need to be aware of how threads are assigned to wavefronts / warps / vectors