Fixing the Hyperdrive: Maximizing Rendering Performance on NVIDIA GPUs

Louis Bavoil, Principal Engineer

Booth #223 - South Hall
www.nvidia.com/GDC
Full-Screen Pixel Shader

SM = Streaming Multiprocessor
TEX = Texture unit
L2 = Level 2 cache
DRAM = physical video-memory unit
CROP = Color ROP
Speed Of Light (SOL) Metrics

“SOL%” = % of Peak Performance

Top SOL% values:
- **SM**: 95%
- **TEX**: 72%
- **L2**: 72%
- **DRAM**: 34%
- **CROP**: 5%
Capturing a Frame from a DX App

Using Nsight Graphics 1.0
Press CTRL-Z, then Space
Press CTRL-Z, then Space
Profiler Result for the Whole Frame

GPU Frame Time: 3.15 ms
Measured using D3D timestamp queries

NOTE: The profiler always locks the GPU Core Clock frequency (for most deterministic results).
Profiler Result for the Whole Frame

"DrawCoarseAOPS" = 49.9% of the frame
Profiling a PerfMarker Range...
### Top SOL Units

<table>
<thead>
<tr>
<th>Top SOLs</th>
<th>SM: 94.8%</th>
<th>TEX: 72.1%</th>
<th>L2: 71.9%</th>
<th>DRAM: 34.3%</th>
<th>CROP: 5.4%</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Idle</td>
<td>0.0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSL2 Stall Cycles</td>
<td>0.1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SM Active</td>
<td></td>
<td></td>
<td>99.1%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SM Active Min/Max Delta</td>
<td></td>
<td></td>
<td>0.3%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SM Issue Utilization Per Active Cycle</td>
<td></td>
<td></td>
<td></td>
<td>95.6%</td>
<td></td>
</tr>
<tr>
<td>SM Occupancy (Active Warps Per Active Cycle)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>46.4%</td>
</tr>
</tbody>
</table>
The Peak-Perf% Analysis Method

For each “Top SOL%” unit:

1. If SOL% > 80% ⇒ (A) try removing work from this unit
   • If SM: By opportunistically skipping instructions using branches (or early depth test)
   • If SM: By moving math instructions to lookup tables
   • If TEX: By moving structured-buffer loads to constant-buffer loads, etc.

2. If SOL% < 60% ⇒ (B) try increasing the SOL% of this unit
   • By removing “idle cycles” (GPU unit is not doing any work for a % of the time)
   • By removing “stall cycles” (GPU unit has internal inefficiencies)
   • By avoiding “slow paths” if possible (e.g. 32-bit index buffers, and FP32x4 textures)

3. If SOL% in [60,80], do both (A) and (B)
Range Profiling & Async Compute

- For DX12, Nsight Frame Captures flatten all async COMPUTE queues to the main DIRECT queue.

- For understanding overlaps of async compute work with graphics work, Nsight GPU Trace can be used.
Example DX11 Workload:

Voxelization using UAV Atomics
CPU Limited?

GPU Idle: 0.0%

⇒ Not CPU limited at all
“Top SOLs”

Top SOLs: [ VPC: 25.0% | SM: 21.1% | L2: 20.6% ]

VPC = ViewPort Culling unit
SM = Streaming Multiprocessor
L2 = Level 2 Cache
“SM Active”: % of the SM cycles with at least one active warp

SM Active: 59.5%
Draw Call Count: 100

Wait For Idle (WFI) Count: 103
DX11 Driver Behavior

By default: Serialize Draw calls with bound UAV in common

Draw call #1 using UAV_0

Draw call #2 using UAV_0

GPU Wait For Idle (WFI)
DX11 Driver Behavior

Optimized: Concurrent Draw Calls

NvAPI_D3D11_BeginUAVOverlap

Draw call #1
using UAV_0

Draw call #2
using UAV_0

NvAPI_D3D11_EndUAVOverlap
# UAV-Overlap Optimization

Add NvAPI_D3D11_{Begin,End}UAVOverlap

<table>
<thead>
<tr>
<th></th>
<th>BEFORE</th>
<th>AFTER</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFI Count</td>
<td>103</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Top SOLs</td>
<td>VPC: 25.0% SM: 21.1% L2: 20.6%</td>
<td>VPC: 52.3% SM: 44.3% L2: 42.6%</td>
<td>VPC: 2.1x SM: 2.1x L2: 2.1x</td>
</tr>
<tr>
<td>SM Active%</td>
<td>59.1%</td>
<td>95.1%</td>
<td>1.6x</td>
</tr>
<tr>
<td>GPU Elapsed Time</td>
<td>0.69 ms</td>
<td>0.38 ms</td>
<td>1.8x Gain</td>
</tr>
</tbody>
</table>
The Peak-Perf% Analysis Method

BEFORE: Top SOLs: [ VPC:25.0% | SM:21.1% | L2:20.6% ]
AFTER: Top SOLs: [ VPC:52.3% | SM:44.3% | L2:42.6% ]

For each “Top SOL%” unit:

1. If SOL% > 80% ➔ (A) try removing work from this unit
2. If SOL% < 60% ➔ (B) try increasing the SOL% of this unit
   • By removing “idle cycles” (GPU unit is not doing any work for a % of the time)
   • By removing “stall cycles” (GPU unit has internal inefficiencies)
   • By avoiding “slow paths” if possible (e.g. avoiding 32-bit index buffers, and avoiding FP32x4 texture formats).

3. If SOL% in [60,80], do both (A) and (B)
Example Workload:

Drawing Tiny Triangles
Index Buffer Format = R32_UINT
With all indices >= USHORT_MAX replaced with 0

API Primitive Count: 22,657,500
Shaded Pixels: 0

Top SOLs [ PD:64.1% | VPC:46.7% | DRAM:36.2% ]

GPU Idle: 0.0%

DRAM Read Utilization: 35.9%

PD = Primitive Distributor unit
VPC = ViewPort Culling unit
“DRAM Read Utilization”: % of cycles that a DRAM read request is active
## Index-Buffer Format Optimization

### 32->16 bits per index

<table>
<thead>
<tr>
<th></th>
<th>BEFORE</th>
<th>AFTER</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Top SOLs</strong></td>
<td>PD:64.1%</td>
<td>PD:80.5%</td>
<td>PD:1.3x</td>
</tr>
<tr>
<td></td>
<td>VPC:46.7%</td>
<td>VPC:58.7%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DRAM:36.2%</td>
<td>DRAM:28.5%</td>
<td>DRAM: 0.8x</td>
</tr>
<tr>
<td><strong>DRAM Read Utilization</strong></td>
<td>36%</td>
<td>28%</td>
<td>0.78x</td>
</tr>
<tr>
<td><strong>GPU Elapsed Time</strong></td>
<td>5.09 ms</td>
<td>2.37 ms</td>
<td>2.1x Gain</td>
</tr>
</tbody>
</table>
The Peak-Perf% Analysis Method

BEFORE: Top SOLs: [ PD:64.1% | VPC:46.7% | DRAM:36.2% ]
AFTER: Top SOLs: [ PD:80.5% | VPC:58.7% | DRAM:28.5% ]

For each “Top SOL%” unit:

1. If SOL% > 80% ⇒ (A) try removing work from this unit
2. If SOL% < 60% ⇒ (B) try increasing the SOL% of this unit
   • By removing “idle cycles” (GPU unit is not doing any work for a % of the time)
   • By removing “stall cycles” (GPU unit has internal inefficiencies)
   • By avoiding “slow paths” if possible (e.g. 32-bit index buffers, and FP32x4 textures)
3. If SOL% in [60,80], do both (A) and (B)
Example Workload:
Light-Tile Culling Compute Shader
“SM Issue Utilization” < 60% AND “SM Warp Stall Barrier” > 20%

→ SM perf is limited by synchronization stalls from GroupMemoryBarrierWithGroupSync() instructions

Top SOLs [ SM:41.9% | TEX:3.4% | L2:1.8% ]

SM Issue Utilization: 42.6%
SM Warp Stall Barrier: 43.2%

SM Issue Utilization: The % of SM active cycles a SM scheduler issued at least one instruction
SM Warp Stall Barrier: % of active warps that were stalled waiting for sibling warps at a CTA barrier
BEFORE: 2-Warp Thread Groups

```c
for (uint i = groupIndex; i < lightCount; i += groupSize ) {
    CullLight(i,...)
}
```

```
for (uint i = groupIndex; i < lightCount; i += groupSize )

GroupMemoryBarrierWithGroupSync()
```

```
GroupMemoryBarrierWithGroupSync()
```

```
GroupMemoryBarrierWithGroupSync()
```

```
GroupMemoryBarrierWithGroupSync()
```

Thread Group
AFTER: 1-Warp Thread Groups

1 Warp (32 Threads)

GroupMemoryBarrierWithGroupSync()

for (uint i = groupIndex;
    i < lightCount;
    i += groupSize )
{
    CullLight(i, ...)
}

GroupMemoryBarrierWithGroupSync()
For single-warp thread groups, barrier instructions are free on NVIDIA GPUs.
# Thread-Group Size Reduction:

64 threads -> 32 threads

<table>
<thead>
<tr>
<th></th>
<th>BEFORE</th>
<th>AFTER</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top SOL</td>
<td>SM:41.9%</td>
<td>SM:73.7%</td>
<td>SM:1.76x</td>
</tr>
<tr>
<td>SM Issue Utilization</td>
<td>42.6%</td>
<td>76.6%</td>
<td>1.80x</td>
</tr>
<tr>
<td>SM Warp Stall on Barriers</td>
<td>43.2%</td>
<td>0.0%</td>
<td>0.0x</td>
</tr>
<tr>
<td>SM Occupancy (Active Warps)</td>
<td>34.3</td>
<td>31.2</td>
<td>0.91x</td>
</tr>
<tr>
<td>GPU Elapsed Time</td>
<td>1.10 ms</td>
<td>0.33 ms</td>
<td>3.3x Gain</td>
</tr>
</tbody>
</table>
For each “Top SOL%” unit (from high to low SOL%):

1. If SOL% > 80%  ➔ (A) try removing work from this unit

2. If SOL% < 60%  ➔ (B) try increasing the SOL% of this unit
   - By removing “idle cycles” (GPU unit is not doing any work for a % of the time)
   - By removing “stall cycles”: SM Warp Stalls on Shared-Memory Barriers
   - By avoiding “slow paths” if possible (e.g. 32-bit index buffers, and FP32x4 textures)

3. If SOL% in [60,80], do both (A) and (B)
Example Workload:
Ray-Marched SSAO
Full-Screen Pixel Shader
with per-pixel jittering of ray directions

1440p, 8 rays per pixel, stride=4 pixels

GPU: GTX 1080
Ray-Marched SSAO Full-Screen Pixel Shader

Top SOLs: L2: 80.3% | SM: 56.0% | TEX: 37.0% | DRAM: 1.6% | CROP: 0.5%

TEX Hit Rate: 67.0%

- Workload is L2 bandwidth limited due to poor TEX hit rate
Ray-Marched SSAO
Full-Screen Pixel Shader

SM Issue Utilization: The % of SM active cycles a SM scheduler issued at least one instruction

SM Issue Utilization: 55.7%

Top SOLs [ L2:80.3% | SM:56.0% | TEX:37.0% | DRAM:1.6% | CROP:0.5% ]
“SM Issue Utilization” < 60% AND “SM Warp Stall Long Scoreboard” > 20%

⇒ SM perf is TEX-latency limited

“SM Warp Stall Long Scoreboard”:
% of active warps that were stalled waiting for a scoreboard dependency on a TEX operation
Our Solution:
“Interleaved Rendering”

Render each sampling pattern separately, using \textit{downsampled} input textures
Assumption:

Interleaved Sampling Patterns

NxN sampling patterns interleaved on screen

Typical sampling strategy for SSAO, SSDO, SSR, etc.

Per-pixel jitter seed fetched from a tiled “jitter texture”
STEP 1: Deinterleave Input

Full-Resolution Input Texture

1 Draw call with 4xMRTs

Half-Resolution 2D Texture Array

Width = \text{iDivUp}(W, 2)
Height = \text{iDivUp}(H, 2)
STEP 2: Jitter-Free Sampling

Input: Texture Array A (slices 0,1,2,3)

1 Draw

Output: Texture Array B (slices 0,1,2,3)
STEP 3: Interleave Results

1 Draw call
With 1 Tex2DArray fetch per pixel
Full-Screen Pixel Shader
AO GPU Time: 6.77 ms
Interleaved Rendering (3 Steps)
AO GPU Time: 0.10 + 5.04 + 0.08 = 5.22 ms [27% gain]
## Interleaved Rendering Optimization

<table>
<thead>
<tr>
<th>AO KERNEL</th>
<th>BEFORE</th>
<th>AFTER</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top SOLs</td>
<td>L2:80.3%</td>
<td>L2:11.3%</td>
<td>L2:0.14x</td>
</tr>
<tr>
<td></td>
<td>SM:56.0%</td>
<td>SM:78.8%</td>
<td>SM:1.4x</td>
</tr>
<tr>
<td></td>
<td>TEX:37.0%</td>
<td>TEX:32.4%</td>
<td>TEX:0.9x</td>
</tr>
<tr>
<td>TEX Hit Rate</td>
<td>67%</td>
<td>93%</td>
<td>1.4x</td>
</tr>
<tr>
<td>SM Issue Utilization</td>
<td>56%</td>
<td>73%</td>
<td>1.3x</td>
</tr>
<tr>
<td>SM Warp Stall Long Scoreboard</td>
<td>48%</td>
<td>28%</td>
<td>0.6x</td>
</tr>
</tbody>
</table>
2x Partial Loop Unrolling

Before

do
{
    // Fetch Sample_1
    // Calculate RayXYZ_1
    // Advance Ray

} while ( ... );

After

do
{
    // Fetch Sample_1
    // Fetch Sample_2
    // Calculate RayXYZ_1
    // Advance Ray
    // Calculate RayXYZ_2
    // Advance Ray

} while ( ... );
# 2x Partial Loop Unrolling

<table>
<thead>
<tr>
<th></th>
<th>BEFORE</th>
<th>AFTER</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Top SOLs</strong></td>
<td>SM: 78.8%</td>
<td>SM: 88.6%</td>
<td>SM: 1.1x</td>
</tr>
<tr>
<td></td>
<td>TEX: 32.4%</td>
<td>TEX: 37.4%</td>
<td>TEX: 1.2x</td>
</tr>
<tr>
<td></td>
<td>L2: 11.3%</td>
<td>L2: 9.9%</td>
<td>L2: 0.9x</td>
</tr>
<tr>
<td><strong>SM Issue Utilization</strong></td>
<td>73%</td>
<td>84%</td>
<td>1.15x</td>
</tr>
<tr>
<td><strong>SM Warp Stall on Long Scoreboard</strong></td>
<td>28%</td>
<td>12%</td>
<td>0.43x</td>
</tr>
<tr>
<td><strong>SM Occupancy (Active Warps)</strong></td>
<td>39.0</td>
<td>33.8</td>
<td>0.87x</td>
</tr>
<tr>
<td><strong>GPU Elapsed Time</strong></td>
<td>5.04 ms</td>
<td>4.53 ms</td>
<td>11% Gain</td>
</tr>
</tbody>
</table>
For each “Top SOL%” unit:

1. If SOL% > 80% ⇒ (A) try removing work from this unit
   - Reduce the number of TEX→L2 requests by improving the TEX hit rate
2. If SOL% < 60% ⇒ (B) try increasing the SOL% of this unit
   - By removing “idle cycles” (GPU unit is not doing any work for a % of the time)
   - By removing “stall cycles” (GPU unit has internal inefficiencies)
   - By avoiding “slow paths” if possible (e.g. 32-bit index buffers, and FP32x4 textures)
3. If SOL% in [60,80], do both (A) and (B)
For each “Top SOL%” unit:

1. If SOL% > 80% ➔ (A) try removing work from this unit
2. If SOL% < 60% ➔ (B) try increasing the SOL% of this unit
   - By removing “idle cycles” (GPU unit is not doing any work for a % of the time)
   - **By removing “stall cycles”: SM Warp Stalls on TEX dependencies**
   - By avoiding “slow paths” if possible (e.g. 32-bit index buffers, and FP32x4 textures)
3. If SOL% in [60,80], do both (A) and (B)
DX12 Advanced Topic:

Binding SRV Descriptors
The TSL1 & TSL2 Caches

SRV descriptor contains texture metadata (type, dimensions, format, etc)

SM  ↔  TEX (+TSL1)  ↔  TSL2 (L1.5 cache)  ↔  L2

If SRV desc or sampler desc not in TEX/L1

If SRV desc or sampler desc not in TSL2
Top SOLs

DRAM: 36.7% | SM: 31.7% | L2: 27.6% | ...

GPU Idle

9.6%

TSL2 Stall Cycles

3.4%
Typical DX12 SRV Binding Pattern

Draw call 1
- SRV 1
- SRV 2
- SRV 3

Draw call 2
- SRV 1
- SRV 7
- SRV 3

2 Draw Calls with same Root Signature
Typical DX12 SRV Binding Pattern

Non-Shader-Visible SRV Descriptor Heap

Shader-Visible SRV Descriptor Heap

CopyDescriptorsSimple

SetGraphicsRootDescriptorTable

SRV 1
SRV 2
SRV 3
SRV 4
SRV 5
SRV 6
SRV 7

[0] SRV 1
[1] SRV 2
[2] SRV 3
[3] SRV 1
[4] SRV 7
[5] SRV 3

SRV 1
SRV 2
SRV 3
SRV 1
SRV 7
SRV 3
The Problem: Redundant Heap Entries

TSL1 & TSL2 caches use **heap indices** as tags

- Redundant entries in the shader-visible heap ➔ TSL1 & TSL2 cache thrashing 😞
Solution #1: Split SRV Ranges

CopyDescriptorsSimple

SetGraphicsRootDescriptorTable

SRV 1
SRV 2
SRV 3
SRV 4
SRV 5
SRV 6
SRV 7

[0] SRV 1
[1] SRV 2
[2] SRV 3
[3] SRV 7

SRV 1
SRV 2
SRV 3
SRV 7
SRV 3

**Solution #2: Shader SRV Indexing**

SetGraphicsRootDescriptorTable

- Dynamically index SRV descriptor in shaders using per-draw-call indices stored in a Root CBV
Split SRV Ranges vs Shader SRV Indexing

- **Shader SRV Indexing**
  - 😊 **Unique SRVs** in shader-visible descriptor heap
  - 😊 No CopyDescriptorsSimple calls used
  - 😞 Slight SM overhead (extra registers & instructions injected by driver)

- **Split SRV Ranges**
  - 😞 CopyDescriptorsSimple CPU overhead
  - 😞 SetGraphicsRootDescriptorTable CPU & GPU overhead
  - 😊 Can use the same shader byte code on DX12 & DX11
DX12 Advanced Topic:
Pixel Shader Barriers
Pixel Shader Barriers (PSBs)

- PSB == lightweight WFI (Wait For Idle) for PS-to-PS dependencies.
  - Hardware command available on Maxwell and beyond.
  - Used automatically by our driver on DX11.

- On DX12, used in ResourceBarrier Transition calls with:
  - StateBefore = D3D12_RESOURCE_STATE_RENDER_TARGET
  - StateAfter = D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE

- All other transitions map to full-pipeline WFIs.
# ResourceBarrier Flag Optimization

<table>
<thead>
<tr>
<th>POST-PROCESSING CHAIN</th>
<th>BEFORE</th>
<th>AFTER</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top SOLs</td>
<td>TEX:35.4%</td>
<td>TEX:40.5%</td>
<td>TEX:1.1x</td>
</tr>
<tr>
<td></td>
<td>L2:33.3%</td>
<td>L2:38.3%</td>
<td>L2:1.2x</td>
</tr>
<tr>
<td></td>
<td>SM:29.9%</td>
<td>DRAM:36.1%</td>
<td>DRAM:1.2x</td>
</tr>
<tr>
<td>Wait For Idle Count</td>
<td>44</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Pixel Shader Barrier Count</td>
<td>0</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>GPU Elapsed Time</td>
<td>0.39 ms</td>
<td>0.29 ms</td>
<td>26% Gain</td>
</tr>
</tbody>
</table>
Conclusion

- **Nsight Graphics 1.0**
  - Makes it easier to export frames to C++ and build them as EXE
  - Exposes powerful hardware metrics in the Range Profiler

- **Blog post for more details:**
  - “The Peak-Performance Analysis Method for Optimizing Any GPU Workload”

- **Demo of Nsight Graphics at NVIDIA Expo Booth**
Questions?

Louis Bavoil
lbavoil@nvidia.com