Deep Dive:
Asynchronous Compute

Stephan Hodes
Developer Technology Engineer, AMD

Alex Dunn
Developer Technology Engineer, NVIDIA
Joint Session

**AMD**
- Graphics Core Next (GCN)
- Compute Unit (CU)
- Wavefronts

**NVIDIA**
- Maxwell, Pascal
- Streaming Multiprocessor (SM)
- Warps
Terminology

Asynchronous: Not **independent**, async work *shares* HW

Work Pairing: Items of GPU work that execute simultaneously

Async. Tax: Overhead cost associated with asynchronous compute
Async Compute → More Performance
Queue Fundamentals

3 Queue Types:
- Copy/DMA Queue
- Compute Queue
- Graphics Queue

All run asynchronously!
General Advice

- Always profile!
  - Can make or break perf
- Maintain non-async paths
  - Profile async on/off
  - Some HW won’t support async
- ‘Member hyper-threading?
  - Similar rules apply
  - Avoid throttling shared HW resources
Regime Pairing

<table>
<thead>
<tr>
<th>Good Pairing</th>
<th>Poor Pairing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics</td>
<td>Graphics</td>
</tr>
<tr>
<td>Shadow Render (Geometry limited)</td>
<td>G-Buffer (Bandwidth limited)</td>
</tr>
<tr>
<td>Compute</td>
<td>Compute</td>
</tr>
<tr>
<td>Light culling (ALU heavy)</td>
<td>SSAO (Bandwidth limited)</td>
</tr>
</tbody>
</table>

(Technique pairing doesn’t have to be 1-to-1)
- Red Flags

Problem/Solution Format

Topics:

- Resource Contention - AMD
- Descriptor heaps - NVIDIA
- Synchronization models
- Avoiding “async-compute tax”
Hardware Details - AMD

- 4 SIMD per CU

- Up to 10 Wavefronts scheduled per SIMD
  - Accomplish latency hiding
  - Graphics and Compute can execute simultaneously on same CU

- Graphics workloads *usually* have priority over Compute
Resource Contention – AMD

**Problem:** Per SIMD resources are shared between Wavefronts

SIMD executes Wavefronts (of different shaders)

- Occupancy limited by
  - # of registers
  - Amount of LDS
  - Other limits may apply...

- Wavefronts contest for caches
### Resource Contention

- Keep an eye on vector register (VGPR) count

<table>
<thead>
<tr>
<th>GCN VGPR Count</th>
<th>&lt;=24</th>
<th>28</th>
<th>32</th>
<th>36</th>
<th>40</th>
<th>48</th>
<th>64</th>
<th>84</th>
<th>&lt;=128</th>
<th>&gt;128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Waves/SIMD</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- Beware of cache thrashing!
  - Try limiting occupancy by allocating dummy LDS
Hardware Details - NVIDIA

- Compute scheduled breadth first over SMs
- Compute workloads have priority over graphics
  - Driver heuristic controls SM distribution

**Maxwell**
Static SM partitioning

**Pascal**
Dynamic SM partitioning
Problem: HW only has one – applications can create many

Switching descriptor heap could be a hazard (on current HW)

- GPU must drain work before switching heaps
- Applies to CBV/SRV/UAV and Sampler heaps
- (Redundant changes are filtered)
- D3D: Must call SetDescriptorHeap per CL!
Avoid hazard if total # descriptors (all heaps) < pool size
Driver sub-allocates descriptor heaps from large pool

Pool sizes (Kepler+):
- CBV/UAV/SRV = 1048576
- Sampler = 2048 + 2032 static + 16 driver owned
- NB. [1048575|4095] → [0xFFFFF|0xFFF] → (packed into 32-bit)
Synchronization

GPU synchronization models to consider:
- Fire-and-forget
- Handshake

CPU also has a part to play
- ExecuteCommandLists (ECLs) schedules GPU work
- Gaps between ECLs on CPU can translate to GPU
Fire-and-Forget (Sync.)

- Work beginning synchronized via fences
Fire-and-Forget (Sync.)

- Work beginning synchronized via fences
- But, some workloads vary frame-to-frame
- Variance leads to undesired work pairing
- Impacts overall frame time as bad pairing impacts performance
CPU Latency (Sync.)

- Similar situation – CPU plays a role here
Similar situation – CPU plays a role here
Game introduces latency on the CPU between ECLs
Latency translates to GPU
Leads to undesired work pairing, etc...
Handshake (Sync.)

- Synchronize begin and end of work pairing
- Ensures pairing determinism
- Might miss some asynchronous opportunity (HW manageable)
- Future proof your code!
Synchronization - Advice

CPU isn’t innocent, keep an eye on it

Two GPU synchronization models:

- Fire-and-Forget 😊
  - **Cons**: Undeterministic regime pairing
  - **Pros**: Less synchronization == more immediate performance (best case scenario)

- Handshake 😊
  - **Cons**: Additional synchronization might cost performance
  - **Pros**: Regime pairing determinism (all the time)

Synchronize for determinism (as well as correctness)
Async. Tax

Overhead cost associated with asynchronous compute

- Quantified by: \( \frac{[AC-Off(\text{ms})]}{[Serialized AC-On (\text{ms})]} \% \)
  - serialize manually via graphics API

- Can easily knock out AC gains!
Async. Tax – Root Cause

**CPU:**
- Additional CPU work organizing/scheduling async tasks
- Synchronization/ExecuteCommandLists overhead

**GPU:**
- Synchronization overhead
- A Difference in work ordering between AC-On/Off
- Different shaders used between AC-On/Off paths
- Additional barriers (cross-queue synchronization)
Async. Tax – Advice

First: determine if CPU or GPU is the bottleneck (GPUView)

CPU:
- Count API calls per frame, compare AC-On/Off for differences
- Measure differences through per-thread profiling

GPU:
- Compare GPU cost of shaders for AC-On/Off
- Inspect difference contributors
Tools

- API Timestamps: Time enable/disable async compute
- GpuView: (PTO)
GPU View #1

- Using 3D, Compute, Copy
- Frame boundaries @ Flip Queue packets
- Compute overlapping graphics per-frame
GPU View #2 - Markers

NB. Open with, ctrl + e

Description

- **Time**: GPU accurate
- **DataSize**: size in bytes of Data
- **Data**: Event name emitted
- PIXBegin/EndEvent
  - Byte Array → ASCII/Unicode
  - Manual step 😞
GPU View #3 - Events

CPU Timeline:
ID3D12Fence::Signal
• DxKrnl – SignalSynchronizationObjectFromCpu
ID3D12Fence::Wait
• DxKrnl – WaitForSynchronizationObjectFromCpu

GPU Timeline:
ID3D12CommandQueue::Signal
• DxKrnl – SignalSynchronizationObjectFromGpu
ID3D12CommandQueue::Wait
• DxKrnl – WaitForSynchronizationObjectFromGpu
Thanks \0

Questions?

@AlexWDunn - adunn@nvidia.com
Stephan.Hodes@amd.com