Multi-Adapter Integrated + Discrete GPUs

Allen Hux, Intel

Legal Notices and Disclaimers

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Intel, Core and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

Agenda

Opportunity: Integrated + Discrete

D3D12 Multi-Adapter Background

Practical Asymmetrical Multi-GPU

Results

Conclusion & Call to Action

References

Integrated Graphics Opportunity

- Many gaming PCs have both integrated and discrete GPUs
- Usually the integrated is idle
- Integrated graphics is a lot of compute!
  - … and, we have an extension that can help extract more performance
- However, D3D12 multi-adapter has many pitfalls

For a class of algorithms, there is a recipe for tapping the integrated GPU for more performance with modest engineering effort.

D3D12 Multi-Adapter Support

2 ways D3D supports Multiple GPUs:

1. LDA: Linked Display Adapter
   - Appears as one adapter (D3D Device) with multiple nodes
   - Transparently copy or use* resources across/between nodes
   - Typically “symmetrical” i.e. identical GPUs

2. Explicit Multiple Adapter
   - Cross-Adapter Shared resources with many restrictions
   - May be “asymmetrical” – this is what we’re doing

Multi-Adapter Approaches

Share Rendering: Split Frame, Alternate Frame, Checkerboard

- Low ROI for asymmetric GPUs

Post-Processing: CMAA, SSAO, Camera effects…

- Requires crossing PCI bus twice

- Occlusion Culling, Physics, AI
  - Producer-consumer
  - Even better when running async from rendering

Multi-Adapter Approaches

Share Rendering: Split Frame, Alternate Frame, Checkerboard

- Low ROI for asymmetric GPUs

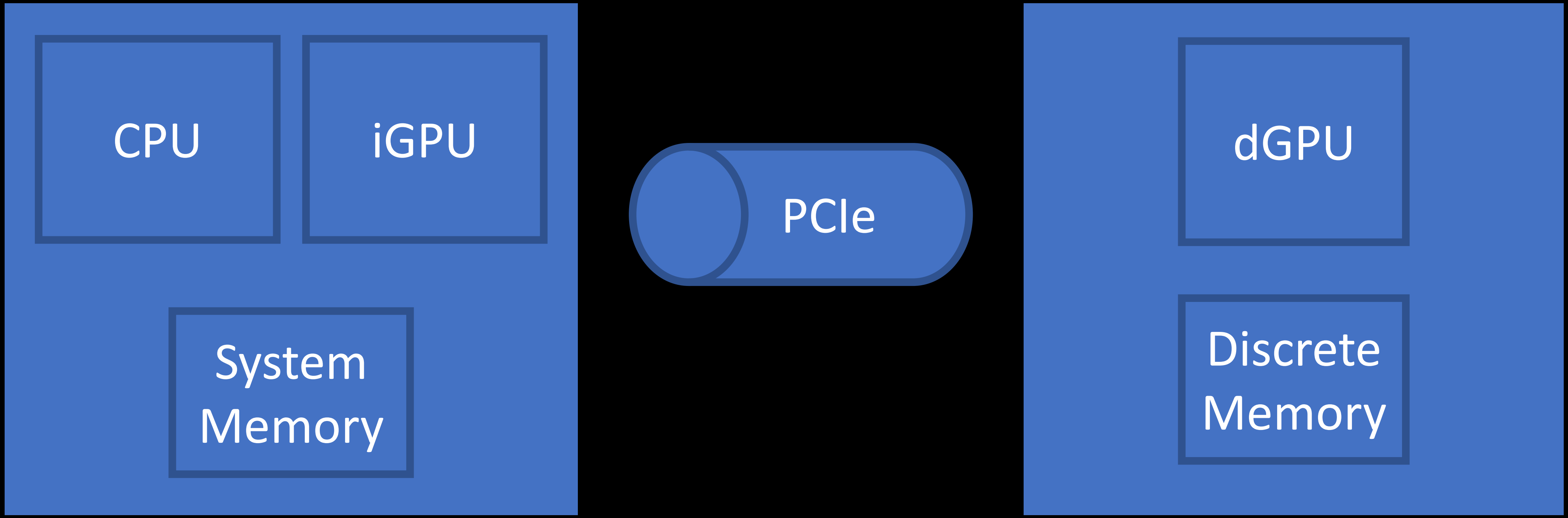
Post-Processing: CMAA, SSAO, Camera effects…

- Requires crossing PCI bus twice

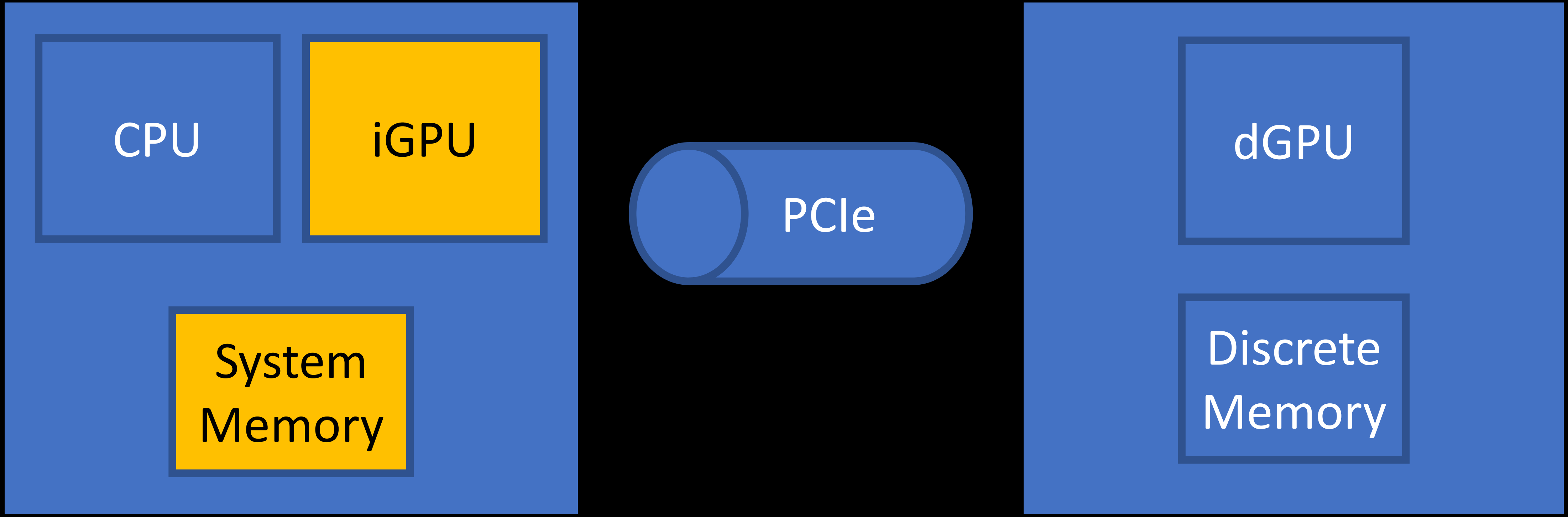
- Occlusion Culling, Physics, AI
  - Producer-consumer
  - Even better when running async from rendering

Best for
Integrated
+
Discrete

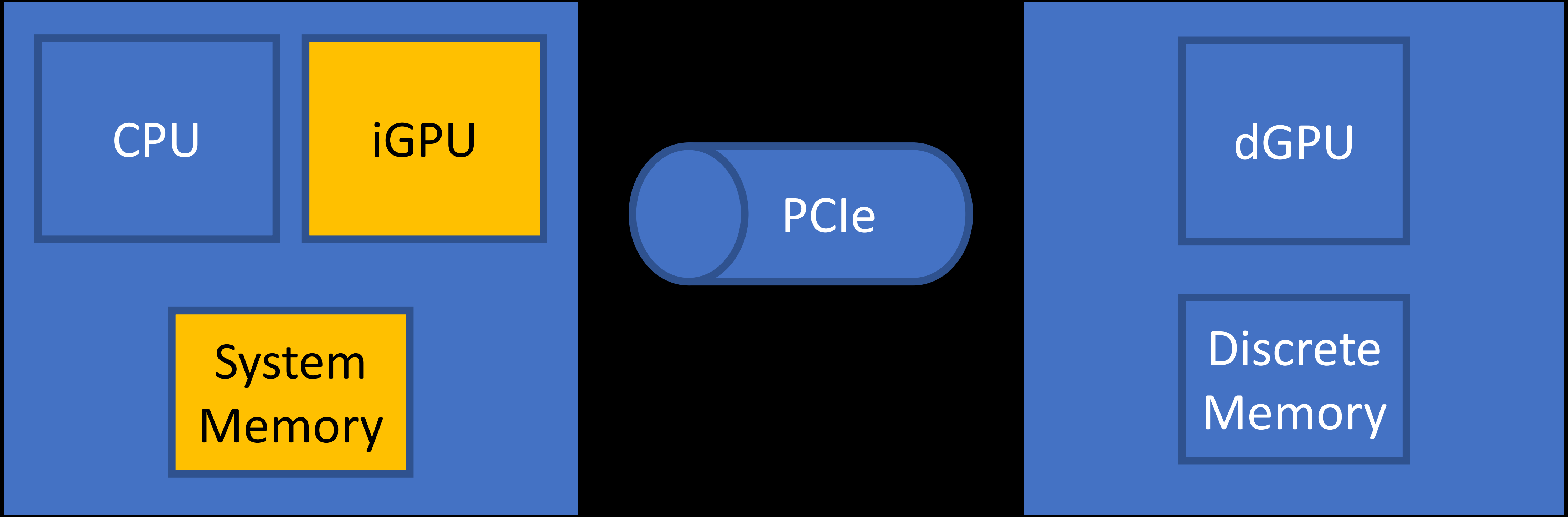
Platform Overview



Platform Overview



Platform Overview



integrated graphics memory *is* system memory
iGPU can use cross-adapter shared resources with little/no penalty

Driving Workload: Microsoft D3D12 n-body particle sim

Uses Async Compute

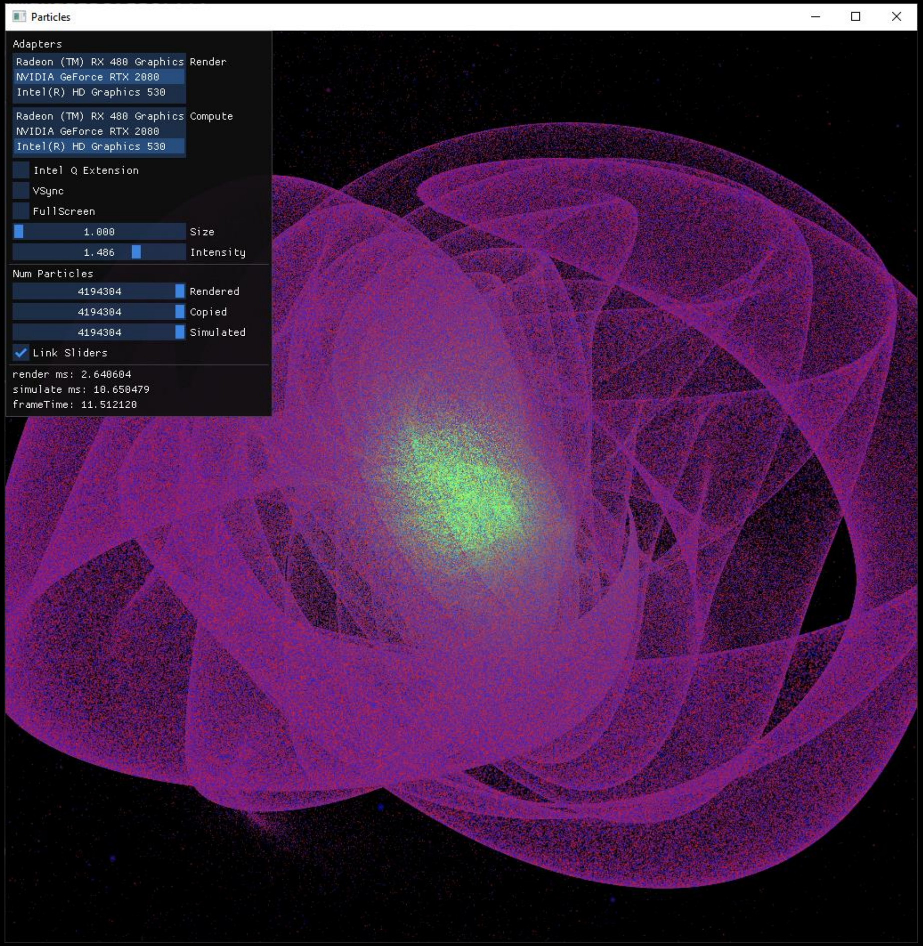
- Separate Render, Compute Queues

Modifications for this talk:

- multi-adapter
- Only 1 gravity source
    - O(n) instead of O(n^2)

Caveat: atypical graphics workload:

- All alpha + geometry shader, No depth



Particles

```
Adapters
Radeon (TM) RX 480 Graphics  Render
NVIDIA GeForce RTX 2080
Intel(R) HD Graphics 530

Radeon (TM) RX 480 Graphics  Compute
NVIDIA GeForce RTX 2080
Intel(R) HD Graphics 530

□ Intel Q Extension
□ VSync
□ FullScreen
            1.000            Size
            1.486            Intensity
Num Particles
           4194304           Rendered
           4194304           Copied
           4194304           Simulated
☑ Link Sliders
render ms: 2.640604
simulate ms: 10.650479
frameTime: 11.512120
```

Agenda

Opportunity: Integrated + Discrete

D3D12 Multi-Adapter Background

Practical Asymmetrical Multi-GPU

Results

Conclusion & Call to Action

References

D3D12 Cross-Adapter Resources

- Resources are allocated by D3D Device -> bound to adapter
- How to move data between adapters?

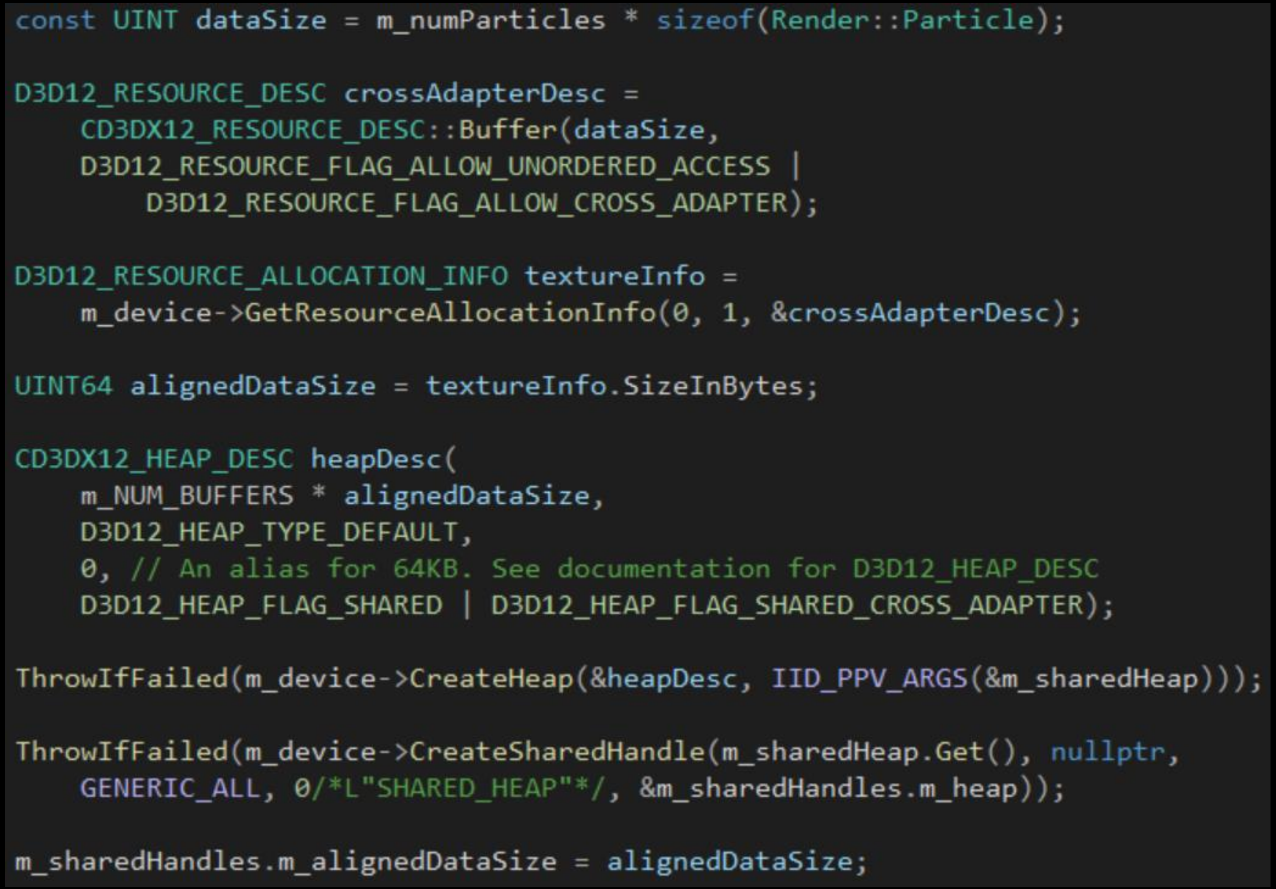
D3D12 Cross-Adapter Resources

- Resources are allocated by D3D Device -> bound to adapter
- How to move data between adapters?

- Shared, Cross-Adapter Resources
- Must be *Placed* in a Cross Adapter Shared Heap

Heap Creation 1, 2, 3

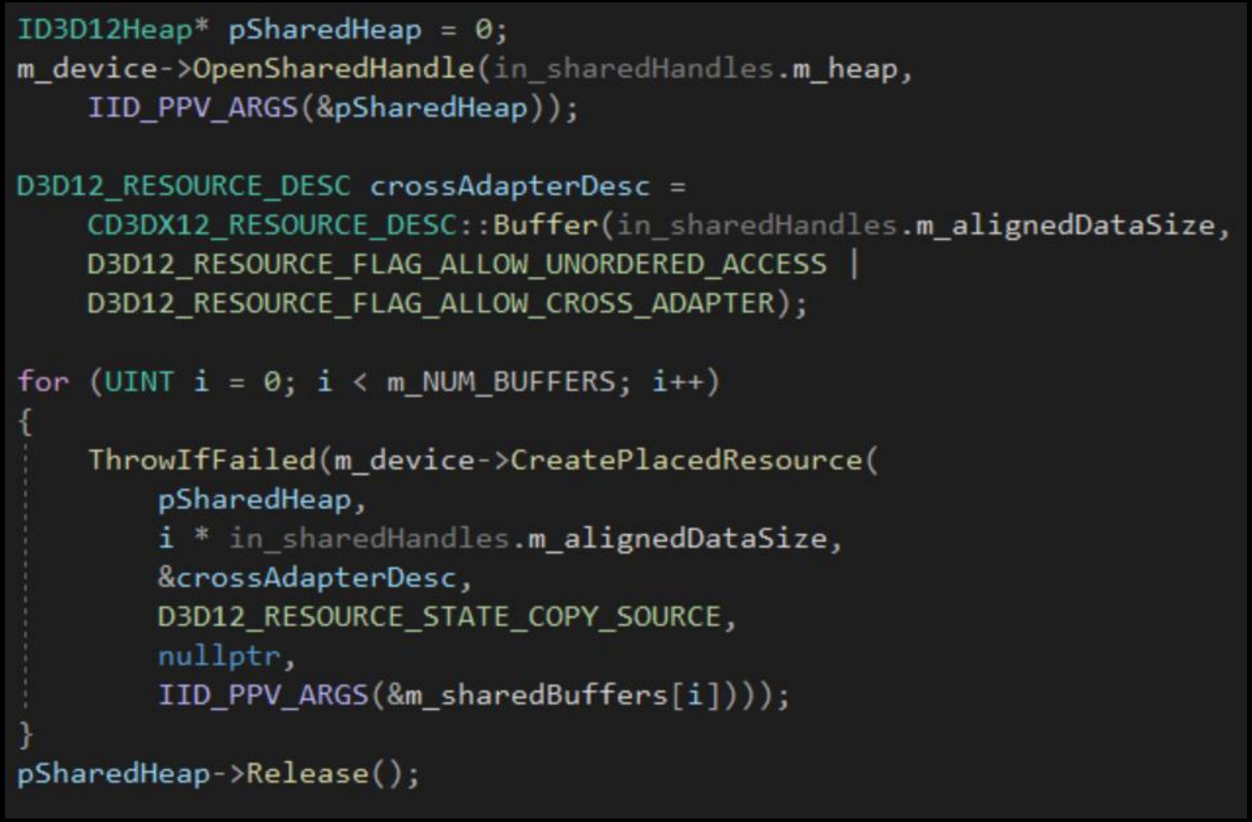
1. Get aligned data size
2. Create shared heap on any device
3. Create handle



```
const UINT dataSize = m_numParticles * sizeof(Render::Particle);

D3D12_RESOURCE_DESC crossAdapterDesc =
    CD3DX12_RESOURCE_DESC::Buffer(dataSize,
    D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS |
        D3D12_RESOURCE_FLAG_ALLOW_CROSS_ADAPTER);

D3D12_RESOURCE_ALLOCATION_INFO textureInfo =
    m_device->GetResourceAllocationInfo(0, 1, &crossAdapterDesc);

UINT64 alignedDataSize = textureInfo.SizeInBytes;

CD3DX12_HEAP_DESC heapDesc(
    m_NUM_BUFFERS * alignedDataSize,
    D3D12_HEAP_TYPE_DEFAULT,
    0, // An alias for 64KB. See documentation for D3D12_HEAP_DESC
    D3D12_HEAP_FLAG_SHARED | D3D12_HEAP_FLAG_SHARED_CROSS_ADAPTER);

ThrowIfFailed(m_device->CreateHeap(&heapDesc, IID_PPV_ARGS(&m_sharedHeap)));

ThrowIfFailed(m_device->CreateSharedHandle(m_sharedHeap.Get(), nullptr,
    GENERIC_ALL, 0/*L"SHARED_HEAP"*/, &m_sharedHandles.m_heap));

m_sharedHandles.m_alignedDataSize = alignedDataSize;
```

Cross-Adapter Resource Creation

- Open handle on 2nd device
- Both adapters: create placed resources within the cross-adapter heap
- Use same alignment and size



```
ID3D12Heap* pSharedHeap = 0;
m_device->OpenSharedHandle(in_sharedHandles.m_heap,
    IID_PPV_ARGS(&pSharedHeap));

D3D12_RESOURCE_DESC crossAdapterDesc =
    CD3DX12_RESOURCE_DESC::Buffer(in_sharedHandles.m_alignedDataSize,
    D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS |
    D3D12_RESOURCE_FLAG_ALLOW_CROSS_ADAPTER);

for (UINT i = 0; i < m_NUM_BUFFERS; i++)
{
    ThrowIfFailed(m_device->CreatePlacedResource(
        pSharedHeap,
        i * in_sharedHandles.m_alignedDataSize,
        &crossAdapterDesc,
        D3D12_RESOURCE_STATE_COPY_SOURCE,
        nullptr,
        IID_PPV_ARGS(&m_sharedBuffers[i])));
}
pSharedHeap->Release();
```

Cross-Adapter Resource Restrictions

- Textures have many restrictions
  - Row-major alignment, displayable with format limitations
  - Possible, and maybe someday efficient
- Focus on Buffers – good for async compute scenarios

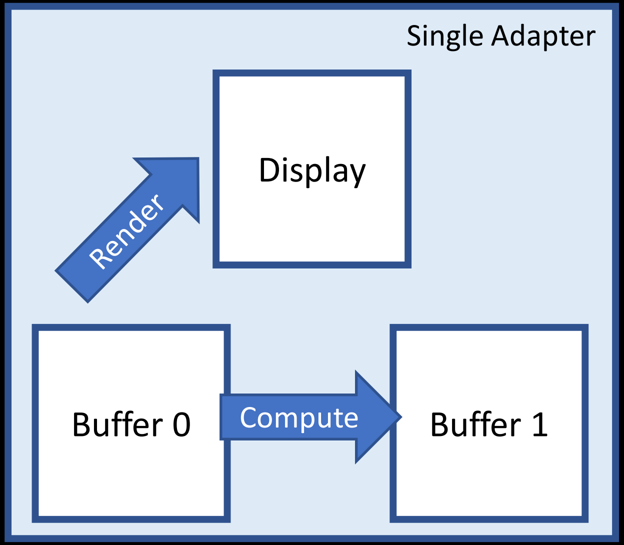
Async compute in particle sample

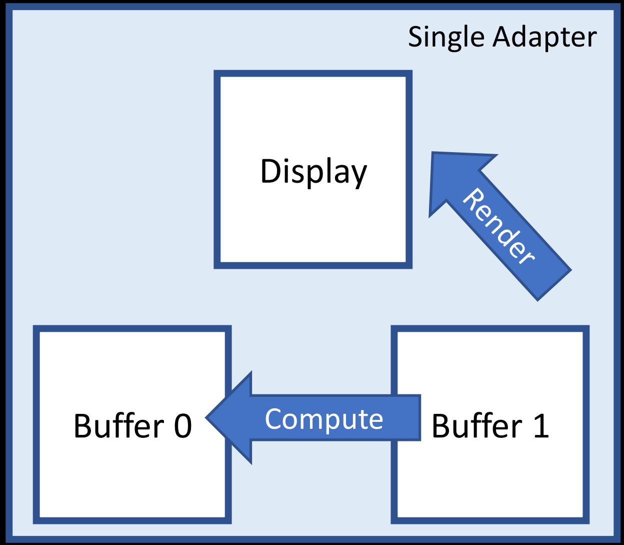
Ping-pong buffers hold source state, destination state

Read initial state
Compute next state
Render results

Parallelized by rendering prior state while computing next state (async compute)

buffer count matches swap chain length



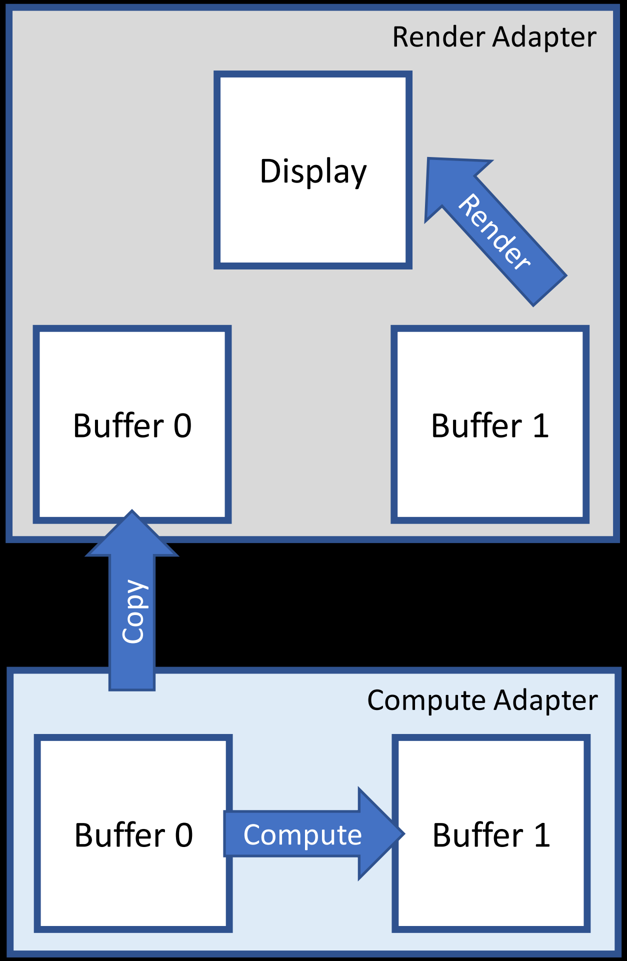


Multi-Adapter: Add copy stage

Idea: each adapter ping-pongs

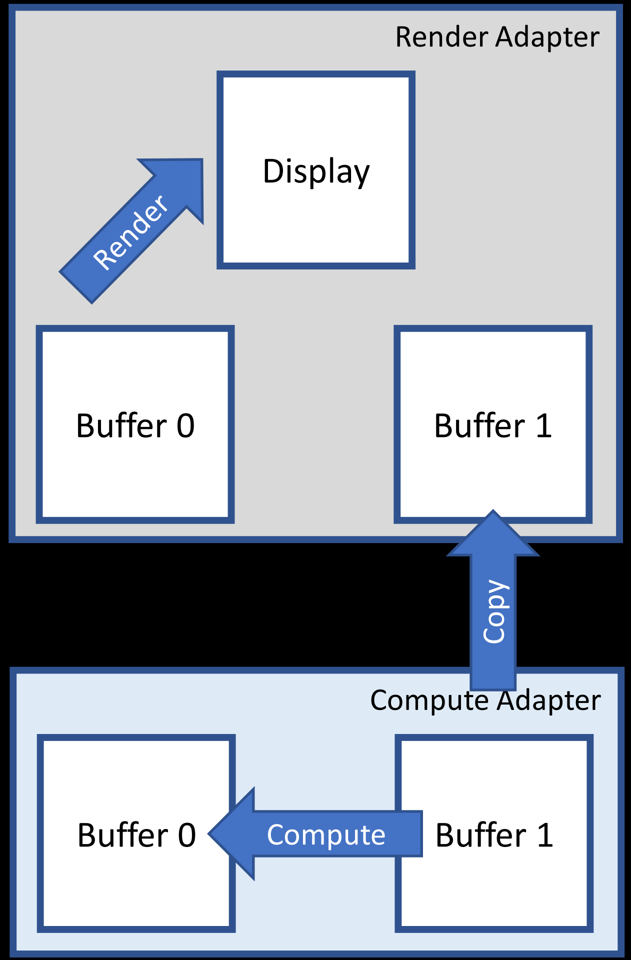
Forms a parallel pipeline:

- 2 readers of state n
- compute state n+1
- render state n-1



Multi-Adapter: Pong

Swap buffers each frame



Resource Allocations

Render buffers:

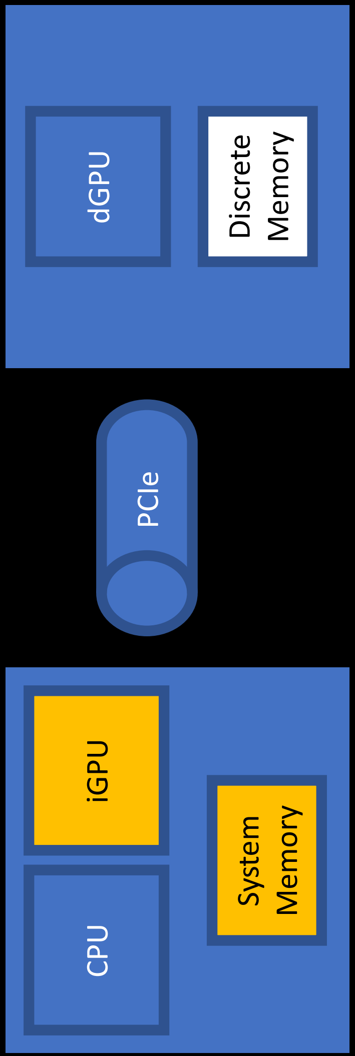
Local adapter heap
Default, Committed

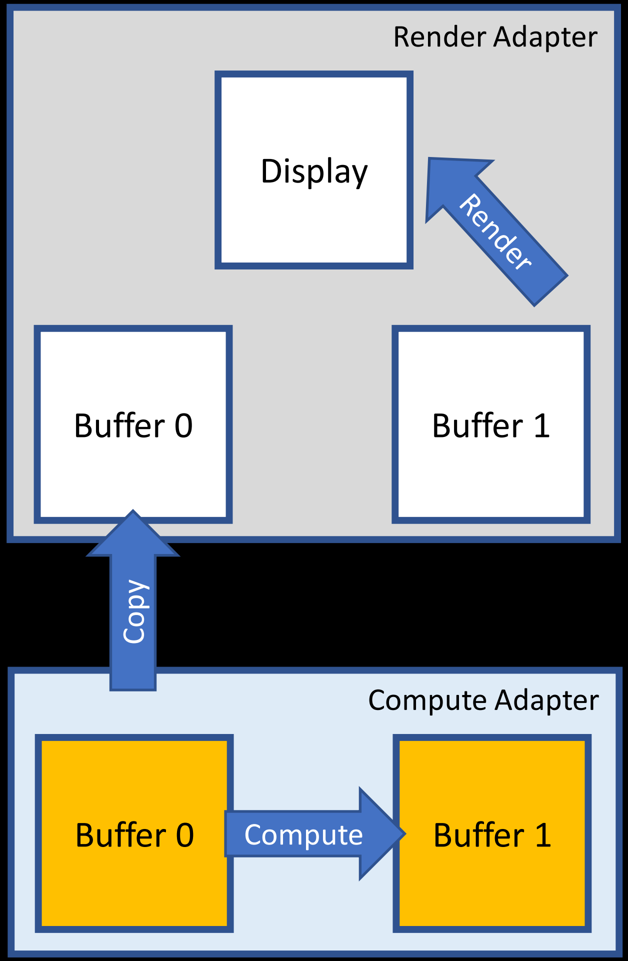
Adapter discrete memory
Adapter preferred layout

Compute buffers:

Cross-adapter heap
Cross-adapter, Placed

CPU memory
Linear layout





Copy Queue

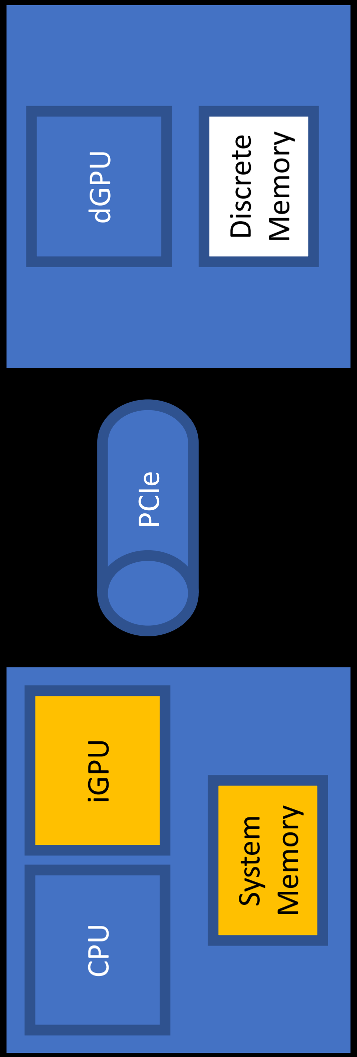
Key Insight:

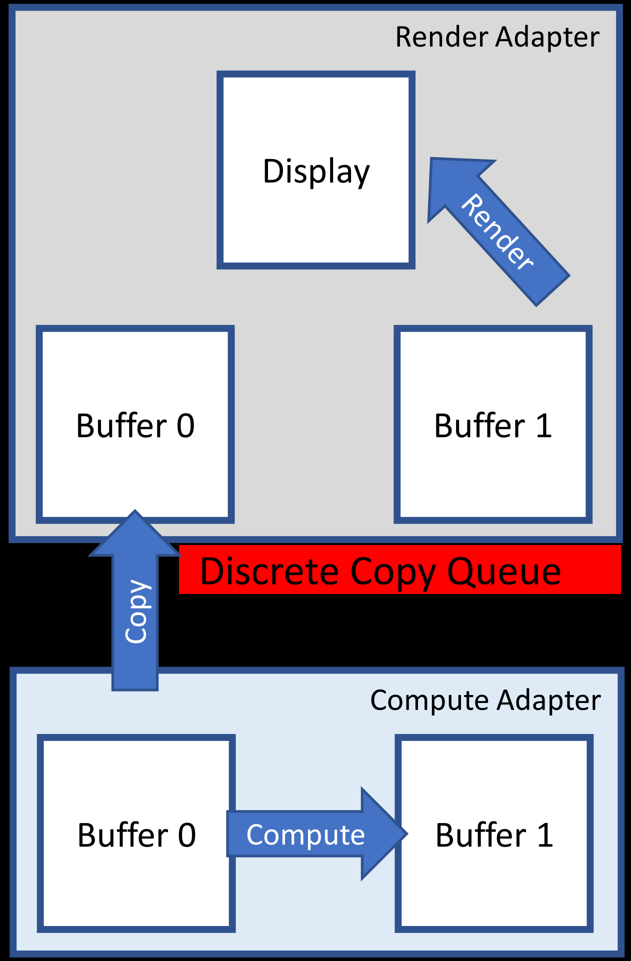
Copy Queue on Discrete Adapter

Copy from system memory to discrete

Integrated memory *is* system mem, so this is a logical arrangement.

Explicit copy stage relaxes timing



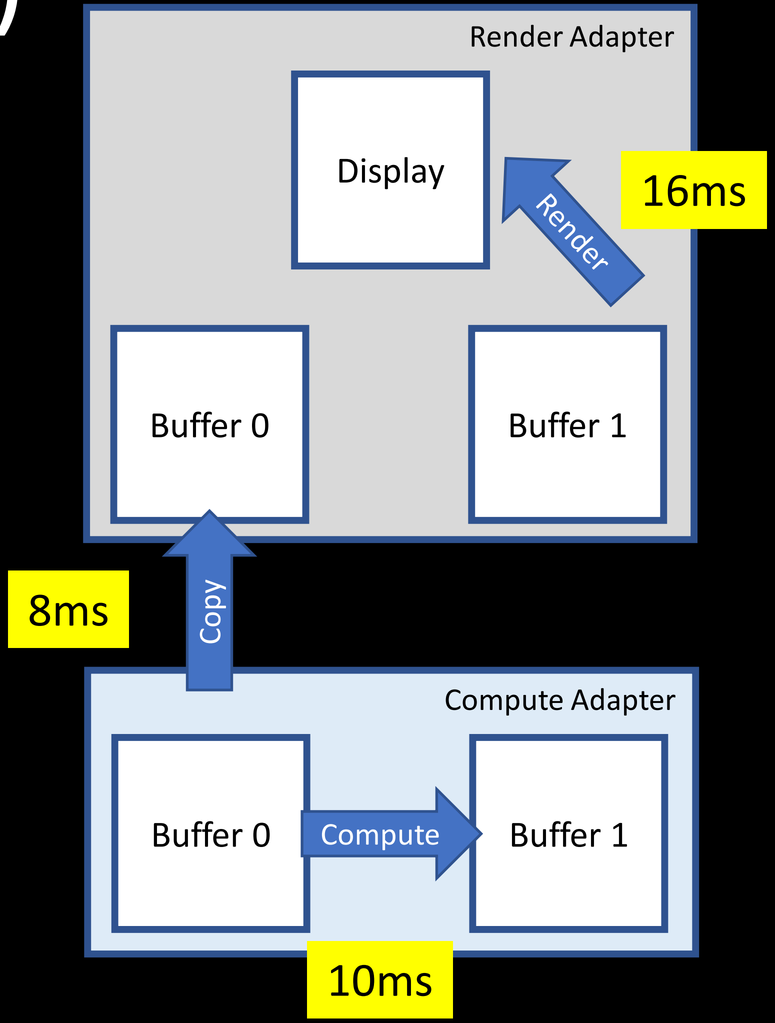


Note: actually used CopyBufferRegion because aligned data size may be padded

Render Time: Max(compute, render, copy)

Frame time is determined by the long pole of three parallel stages

Example shown: 16ms



Resource Creation

Compute Adapter

```
for (UINT i = 0; i < m_NUM_BUFFERS; i++)
{
    ThrowIfFailed(m_device->CreatePlacedResource(
        m_sharedHeap.Get(),
        i * alignedDataSize,
        &crossAdapterDesc,
        D3D12_RESOURCE_STATE_UNORDERED_ACCESS,
        nullptr,
        IID_PPV_ARGS(&m_positionBuffers[i])));
}
```

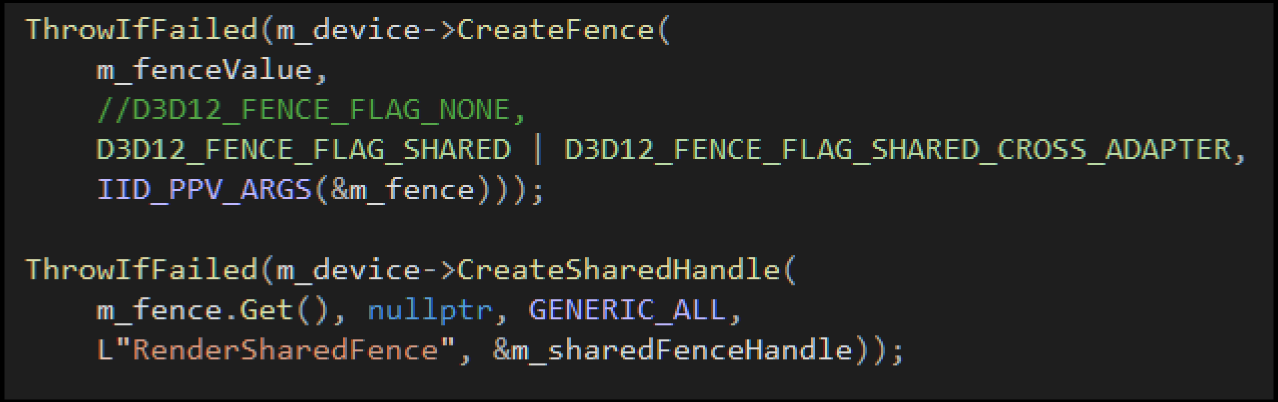
Render Adapter

```
for (UINT i = 0; i < m_NUM_BUFFERS; i++)
{
    ThrowIfFailed(m_device->CreatePlacedResource(
        m_sharedHeap.Get(),
        i * alignedDataSize,
        &crossAdapterDesc,
        D3D12_RESOURCE_STATE_COPY_SOURCE,
        nullptr,
        IID_PPV_ARGS(&m_positionBuffers[i])));
}
```

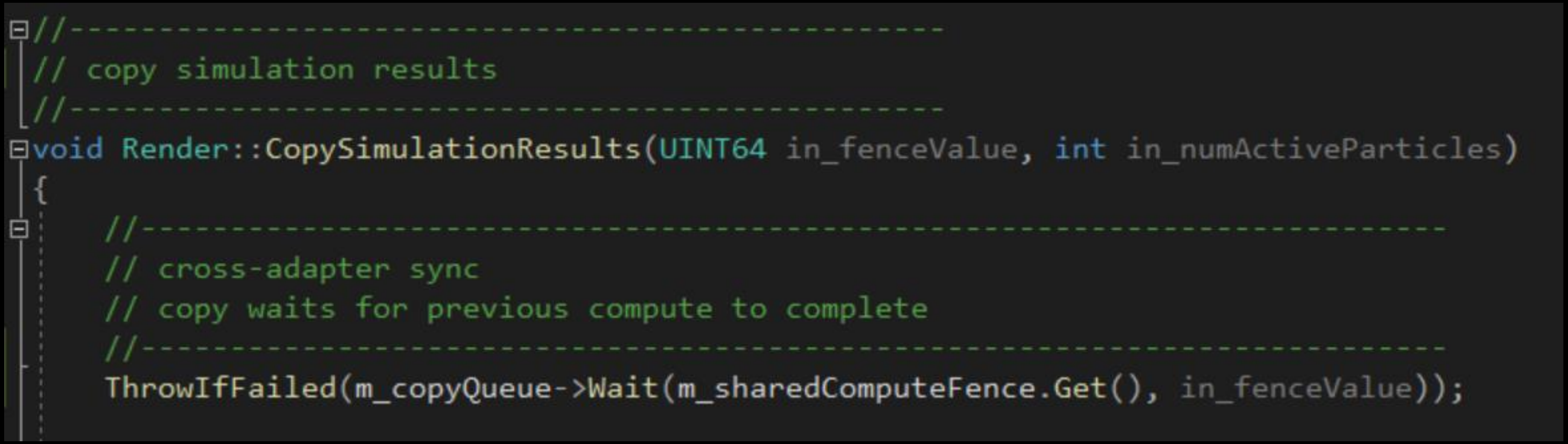
Cross-Adapter Synchronization

- Share fence handles once

- Pass event values per-frame



```
ThrowIfFailed(m_device->CreateFence(
    m_fenceValue,
    //D3D12_FENCE_FLAG_NONE,
    D3D12_FENCE_FLAG_SHARED | D3D12_FENCE_FLAG_SHARED_CROSS_ADAPTER,
    IID_PPV_ARGS(&m_fence)));

ThrowIfFailed(m_device->CreateSharedHandle(
    m_fence.Get(), nullptr, GENERIC_ALL,
    L"RenderSharedFence", &m_sharedFenceHandle));
```



```
//------------------------------------------------
// copy simulation results
//------------------------------------------------
void Render::CopySimulationResults(UINT64 in_fenceValue, int in_numActiveParticles)
{
    //----------------------------------------------------------------------------
    // cross-adapter sync
    // copy waits for previous compute to complete
    //----------------------------------------------------------------------------
    ThrowIfFailed(m_copyQueue->Wait(m_sharedComputeFence.Get(), in_fenceValue));
```

Synchronization Cycle of Life

Compute waits on Copy Fence

Cross Adapter

Copy waits on Compute Fence

Cross Adapter

AND render fence

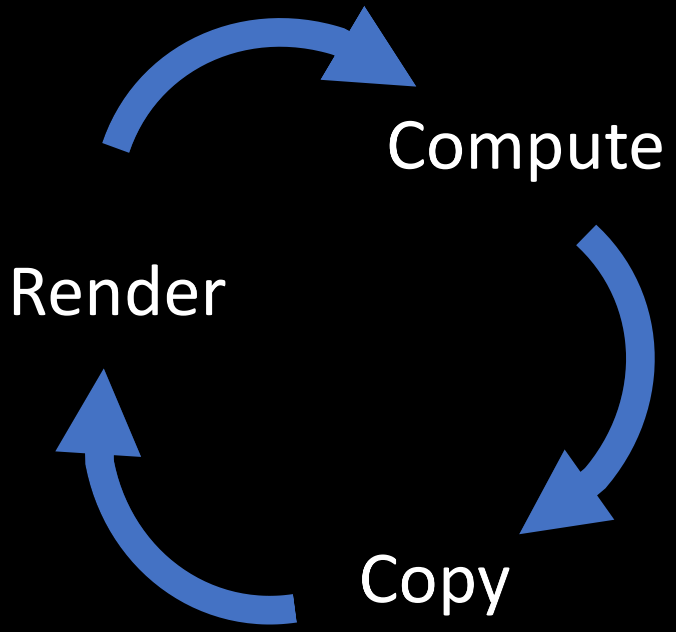
Cross Engine

Render waits on Copy Fence

Cross Engine

CPU waits on render fence

(which covers all fences)



Intel Extension:
Command Queue Throttle

Maintains performance even when load is inconsistent

E.g. integrated not 100% active, waiting on discrete

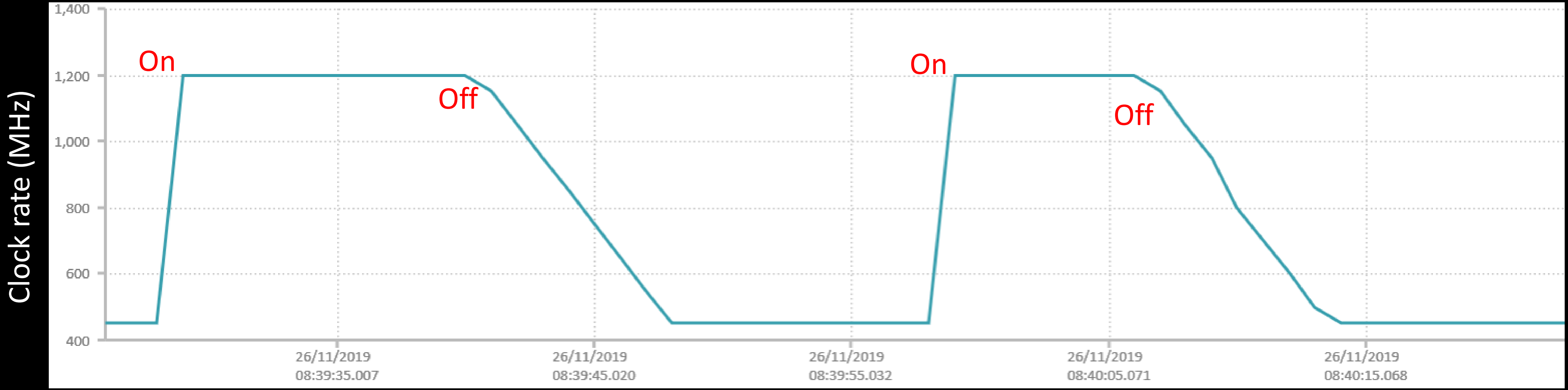
All public Intel drivers support the extension:

D3D12_COMMAND_QUEUE_THROTTLE_MAX_PERFORMANCE

Header file from your friendly Intel contact

Intel Throttle Extension

below: toggling extension on/off



- When integrated idles between commands, clock rate drops
- With extension enabled, commands are executed full-speed
- Your mileage may vary, but this is another tool you can try

Agenda

Opportunity: Integrated + Discrete

D3D12 Multi-Adapter Background

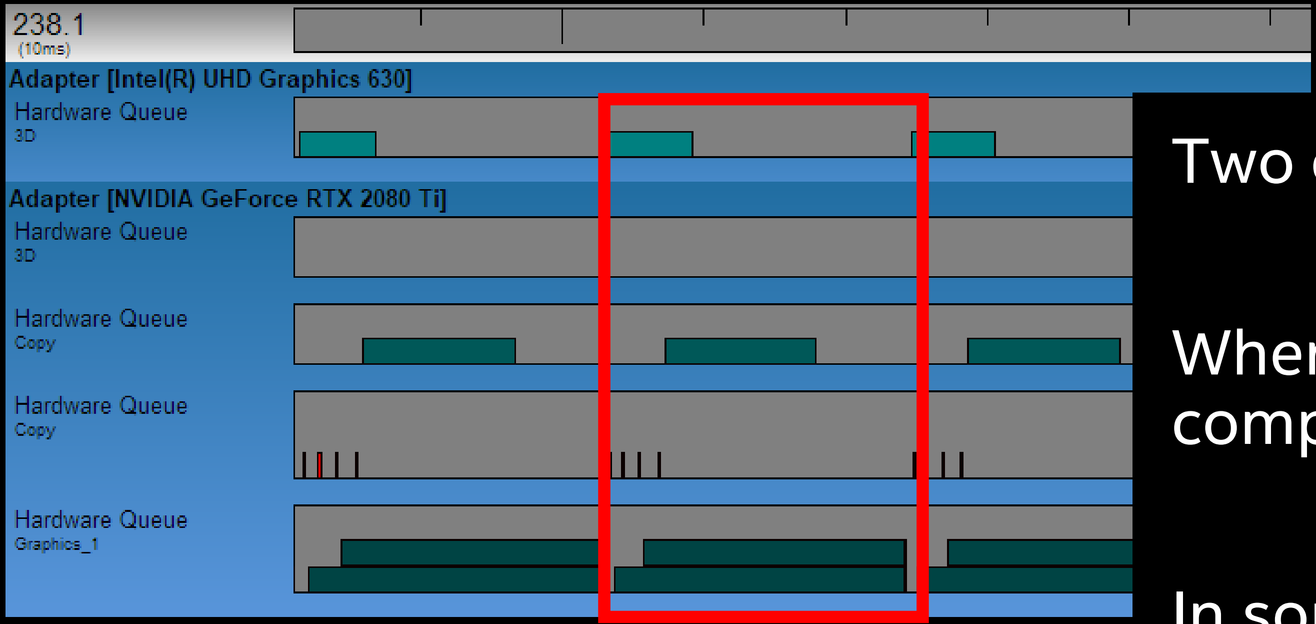
Practical Asymmetrical Multi-GPU

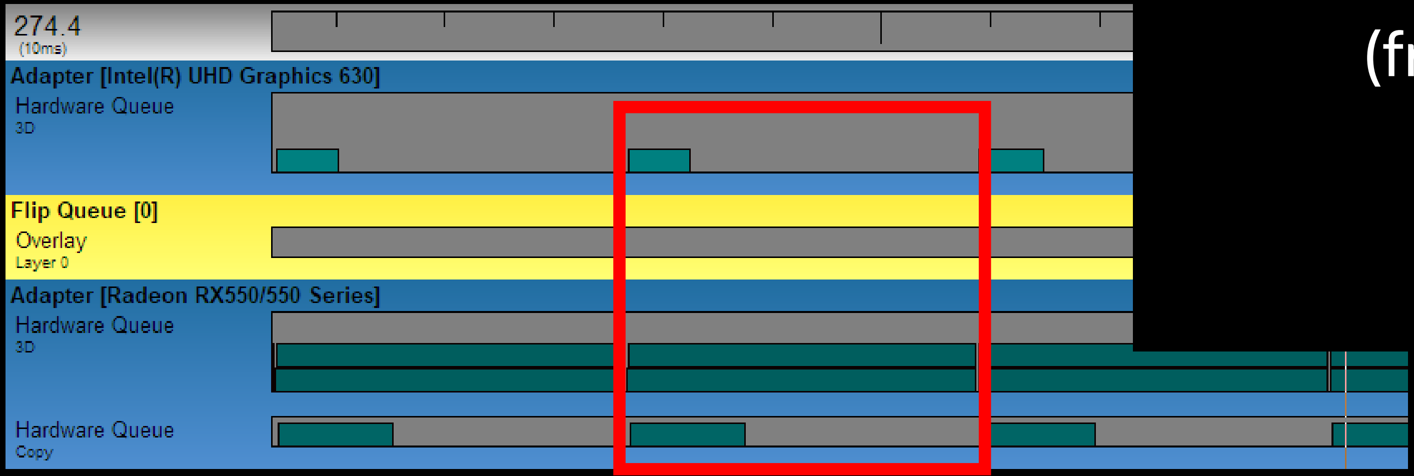
Results

Conclusion & Call to Action

References

GpuView Shows Stages Run in Parallel



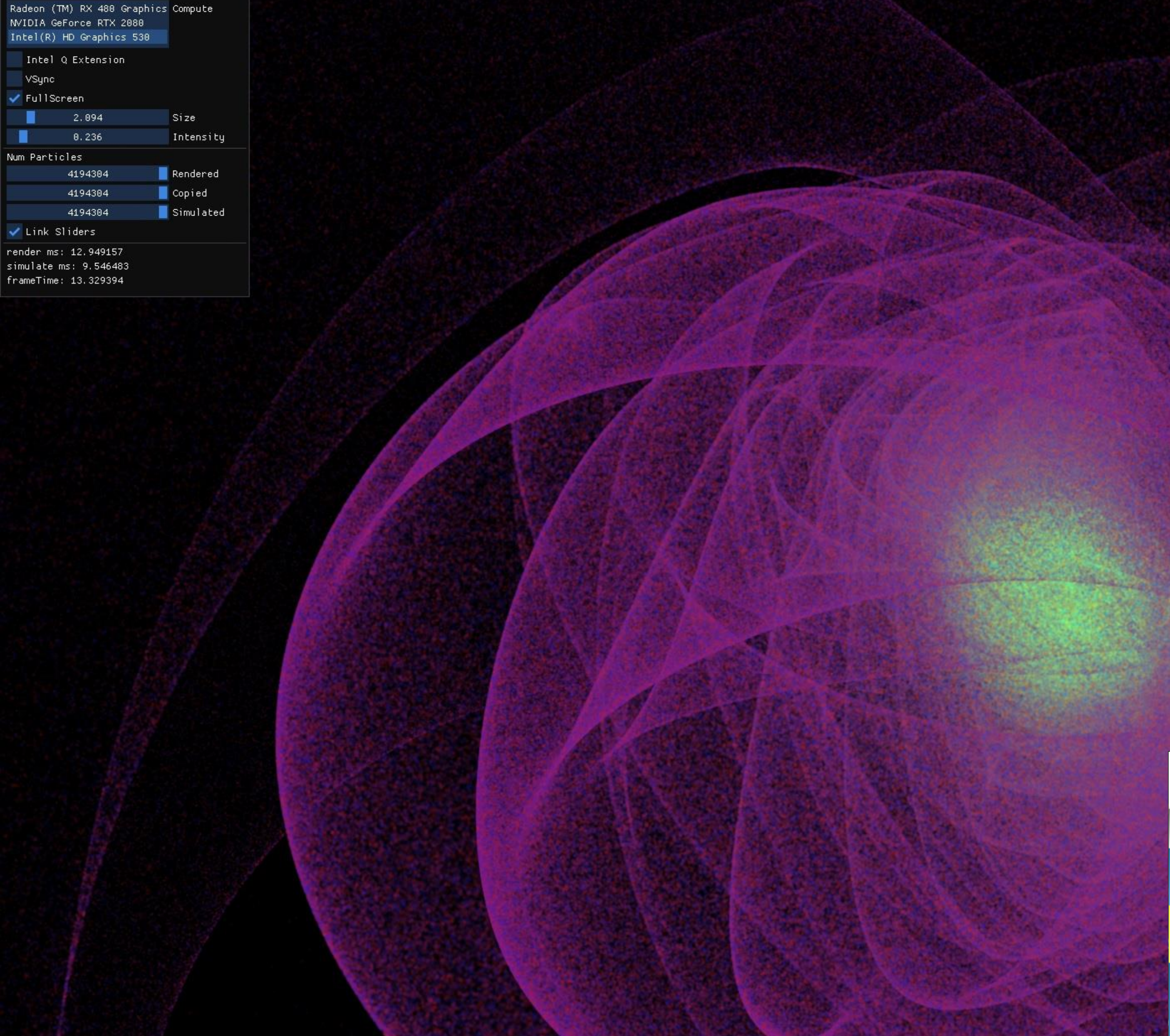


Two examples, full-screen application

When 3D is dominant, there is time for compute and copy

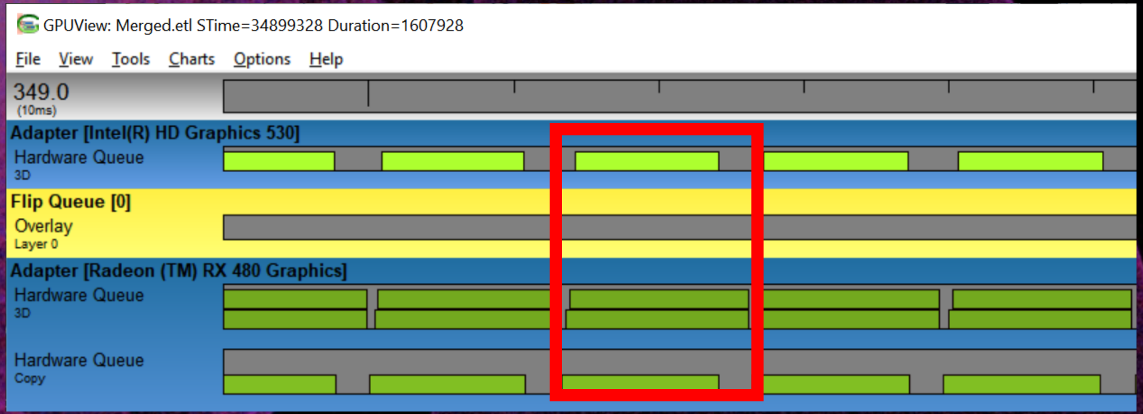
In some scenarios, Copy can dominate

(frame time > render + compute)



```
Radeon (TM) RX 480 Graphics Compute
NVIDIA GeForce RTX 2080
Intel(R) HD Graphics 530
□ Intel Q Extension
□ VSync
☑ FullScreen
2.094 Size
0.236 Intensity
Num Particles
4194304 Rendered
4194304 Copied
4194304 Simulated
☑ Link Sliders
render ms: 12.949157
simulate ms: 9.546483
frameTime: 13.329394
```

Queues and Adapters run in-sync


GPUView: Merged.etl STime=34899328 Duration=1607928
File View Tools Charts Options Help
349.0
(10ms)
Adapter [Intel(R) HD Graphics 530]
Hardware Queue
3D
Flip Queue [0]
Overlay
Layer 0
Adapter [Radeon (TM) RX 480 Graphics]
Hardware Queue
3D
Hardware Queue
Copy

4M particles Intel HD 530 + AMD RX 480

Agenda

Opportunity: Integrated + Discrete

D3D12 Multi-Adapter Background

Practical Asymmetrical Multi-GPU

Results

Conclusion & Call to Action

References

Observations

This technique is best when:

- Render GPU is saturated
- Pure producer-consumer (data crosses bus only once)
- Task can be completely offloaded (no collaboration)
- Render not waiting (pipeline has room to breathe)
- Best: compute allowed to take > 1 frame

Many async compute tasks fit this pattern

Be aware of PCIe bandwidth

- Gen3 x16: 16GB/s
- 4M particles, one float4 each: 64MB
- 16GB / 64MB = 256Hz maximum frame rate
  - Some GPUs/configs are x8: half bandwidth

Keep data transfer size as low as possible!

Splitting data buffers by usage has perf benefit

Low Code Complexity

Essentially an enhancement of async compute

Simplifies transition barriers (vs. single adapter)

- Copy Queue benefits from Implicit State Transitions
  - No transitions to/from COPY_DEST or COPY_SOURCE
- Each Adapter/Queue views resource exactly as it needs it
  - No transitions between UAV or SRV

Little specific cross-adapter code

- Shared Resources from only one adapter
- Share fence(s)

Call to Action

This recipe works for more than particles

Could be physics, mesh deformation, AI, shadows

Many async compute tasks fit this pattern

Check for Intel integrated graphics!

Agenda

Opportunity: Integrated + Discrete

D3D12 Multi-Adapter Background

Practical Asymmetrical Multi-GPU

Results

Conclusion & Call to Action

References

References

- Intel® Devmesh
- Multi-Adapter-Particles Sample code on Github

- Microsoft® n-Body Gravity Sample
- GPUOpen nBody Async Sample