ABSTRACT

Join AMD Game Engineering team members for an introduction to the AMD Ryzen™ family of processors followed by advanced optimization topics. Learn about the high-performance AMD "Zen 2" microarchitecture and profiling tools. Gain insight into code optimization opportunities and lessons learned. Examples may include C/C++, assembly, and hardware performance-monitoring counters.
Ken Mitchell is a Principal Member of Technical Staff in the Radeon™ Technologies Group/AMD ISV Game Engineering team where he focuses on helping game developers utilize AMD processors efficiently. His previous work includes automating and analyzing PC applications for performance projections of future AMD products as well as developing benchmarks. Ken studied computer science at the University of Texas at Austin.
AGENDA

- Success Stories
- “Zen 2” Architecture Processors
- AMD uProf Profiler
- Optimizations and Lessons Learned
- Contacts
SUCCESS STORIES
SUCCESS STORIES

BORDERLANDS 3
DirectX® 12

GEARS 5
DirectX® 12

WORLD WAR Z
Vulkan®
“ZEN 2” ARCHITECTURE PROCESSORS
“ZEN 2” PRODUCT EXAMPLES

NOTEBOOK

“Renoir”
AMD Ryzen™ 7 4800U 8-Core Processor

DESKTOP

“Matisse”
AMD Ryzen™ 9 3950X 16-Core Processor

HIGH END DESKTOP

“Castle Peak”
AMD Ryzen™ Threadripper™ 3990X 64-Core Processor
MICROARCHITECTURE
ADVANCES IN “ZEN 2” MICROARCHITECTURE

- +15% IPC Improvement from “Zen” to “Zen 2"
- 2x op cache capacity
- Reoptimized L1I cache
- 3rd address generation unit
- 2x FP data path width
- 2x L3 capacity
- Improved branch prediction accuracy
- Hardware optimized Security Mitigations
- Secure Virtualization with Guest Mode Execute Trap (GMET)
- Improved SMT fairness
  - for ALU and AGU schedulers
- Improved Write Combining Buffer
DATA FLOW
“RENOIR” 8 CORE PROCESSOR

AMD Ryzen™ 7 4800U, 15W TDP, 8 Cores, 16 Threads, 4.2 GHz max boost clock, 1.8 GHz base clock, integrated GPU.

* Monolithic Die. Each 4M L3 Cache has its own 32B/cycle link to the data fabric. 64b DDR4 Channel Shown.
AMD Ryzen™ 9 3950X, 105W TDP, 16 Cores, 32 Threads, 4.7 GHz max boost clock, 3.5 GHz base clock.

* Two Core Complex Die (CCD). Each CCD has two 16M L3 Cache Complexes.

* The L3 Cache Complexes within a CCD share a single link to the Data Fabric.
**“CASTLE PEAK” 64 CORE PROCESSOR**

AMD Ryzen™ Threadripper™ 3990X, 280W TDP, 64 Cores, 128 Threads, 4.3 GHz max boost clock, 2.9 GHz base clock.

* Two CCDs per Data Fabric Quadrant.
* Two Data Fabric Quadrants have Unified Memory Controllers and two have IO Hubs.

**Data Fabric Quadrant**
- 32B fetch
- 32K I-Cache 8-way
- 32B/cycle
- 32K D-Cache 8-way
- 2*32B load
- 32B/cycle
- 1*32B store
- 32B/cycle
- 16M L3
- I+D Cache 16-way
- 32B/cycle
- 512K L2
- I+D Cache 8-way
- 32B/cycle
- 32B fetch
- 32B/cycle
- 32K I-Cache 8-way
- 32B/cycle
- 32K D-Cache 8-way
- 2*32B load
- 32B/cycle
- 1*32B store
- 32B/cycle
- 16M L3
- I+D Cache 16-way
- 32B/cycle
INSTRUCTION SET
### INSTRUCTION SET EVOLUTION

| YEAR | FAMILY | PRODUCT FAMILY | ARCHITECTURE | EXAMPLE PRODUCT | CLWB | ADX | CLUSHOPT | RDSEED | SHA | SMAP | XGETBV | XSAVEC | XSAVES | AVX2 | BM12 | MOVBE | RDRND | SMEP | FSGSBASE | XSAVEOPT | BMI | FMA | FMA4 | FMA4_1 | FMA4_2 | SSE4_1 | SSE4_2 | XSAVE | SSSE3 | MONITOR | MONITORX | CLZERO | WBNOINVD | FMA3 | EMA4 | TBX | XOP |
|------|--------|----------------|--------------|-----------------|------|-----|---------|--------|-----|------|--------|--------|--------|------|-----|-------|-------|-----|----------|---------|-----|-----|------|-------|-------|-------|-------|-------|-------|--------|--------|--------|------|-----|-----|-----|
| 2019 | 17h    | “Matisse”      | “Zen2”      | Ryzen™ 9 3950X  | 1    | 1   | 1      | 1      | 1   | 1    | 1      | 1      | 1      | 1    | 1   | 1      | 1      | 1   | 1         | 1      | 1   | 1   | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1 | 0 | 0 | 0 |
| 2017 | 17h    | “Summit Ridge”, “Pinnacle Ridge” | “Zen”, “Zen+” | Ryzen™ 7 2700X  | 0    | 1   | 1      | 1      | 1   | 1    | 1      | 1      | 1      | 1    | 1   | 1      | 1      | 1   | 1         | 1      | 1   | 1   | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1 | 0 | 0 | 0 |
| 2015 | 15h    | “Carrizo”, “Bristol Ridge” | “Excavator”  | A12-9800        | 0    | 0   | 0      | 0      | 0   | 0    | 0      | 1      | 1      | 1    | 1   | 1      | 1      | 1   | 1         | 1      | 1   | 1   | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1 | 1 | 1 | 1 |
| 2014 | 15h    | “Kaveri”, “Godavari” | “Steamroller” | A10-7890K       | 0    | 0   | 0      | 0      | 0   | 0    | 0      | 0      | 0      | 0    | 1   | 1      | 1      | 1   | 1         | 1      | 1   | 1   | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1 | 1 | 1 | 1 |
| 2012 | 15h    | “Vishera”      | “Piledriver” | FX-8370         | 0    | 0   | 0      | 0      | 0   | 0    | 0      | 0      | 0      | 0    | 0   | 1      | 1      | 1   | 1         | 1      | 1   | 1   | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1 | 1 | 1 | 1 |
| 2011 | 15h    | “Zambezi”      | “Bulldozer”  | FX-8150         | 0    | 0   | 0      | 0      | 0   | 0    | 0      | 0      | 0      | 0    | 0   | 0      | 0      | 0   | 0         | 1      | 1   | 1   | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1 | 1 | 1 | 1 |
| 2013 | 16h    | “Kabini”       | “Jaguar”     | A6-1450         | 0    | 0   | 0      | 0      | 0   | 0    | 0      | 0      | 0      | 0    | 1   | 0      | 0      | 1   | 1         | 0      | 1   | 1   | 1    | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1      | 1 | 1 | 1 | 1 |
| 2011 | 14h    | “Ontario”      | “Bobcat”     | E-450           | 0    | 0   | 0      | 0      | 0   | 0    | 0      | 0      | 0      | 0    | 0   | 0      | 0      | 0   | 0         | 0      | 0   | 0   | 0    | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0 | 0 | 0 | 0 |
| 2011 | 12h    | “Llano”        | “Husky”      | A8-3870         | 0    | 0   | 0      | 0      | 0   | 0    | 0      | 0      | 0      | 0    | 0   | 0      | 0      | 0   | 0         | 0      | 0   | 0   | 0    | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0 | 0 | 0 | 0 |

“Zen 2” added CLWB and the AMD vendor specific instruction WBNOINVD.
SOFTWARE PREFETCH LEVEL INSTRUCTIONS

- Loads a cache line from the specified memory address into the data-cache level specified by the locality reference T0, T1, T2, or NTA.
- If a memory fault is detected, a bus cycle is not initiated and the instruction is treated as an NOP.
- Prefetch levels T0/T1/T2 are treated identically in “Zen” & “Zen 2” microarchitectures.
- The non-temporal cache fill hint, indicated with PREFETCHNTA, reduces cache pollution for data that will only be used once. It is not suitable for cache blocking of small data sets. Lines filled into the L2 cache with PREFETCHNTA are marked for quicker eviction from the L2 and when evicted from the L2 are not inserted into the L3.
- The operation of this instruction is implementation-dependent. Prefetch fill & evict policies may differ for other processor vendors or microarchitecture generations.
“MATISSE” CACHE AND MEMORY
Cache line size is 64 Bytes.
- 2 cpu clock cycles to move a single cache line.
- L2 is inclusive of L1.
  - lines filled into L1 are also filled into L2.
- L3 is filled from L2 victims of all 4 cores within its CCX.
  - L2 tags are duplicated in its L3 for fast cache transfers within a CCX.
- L2 capacity evictions may cause L3 capacity evictions.
- “Matisse” products may have 1 or 2 CCDs.
  - Each CCD Core Complex Die (CCD) may have two CCX.
  - CCX: Core Complex (4 Cores, 8 Logical Processors, 16MB).

### CACHE LATENCY

<table>
<thead>
<tr>
<th>Level</th>
<th>Count/CCD</th>
<th>Capacity</th>
<th>Sets</th>
<th>Ways</th>
<th>Line Size</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>uop</td>
<td>8</td>
<td>4 K uops</td>
<td>64</td>
<td>8</td>
<td>8 uops</td>
<td>NA</td>
</tr>
<tr>
<td>L1I</td>
<td>8</td>
<td>32 KB</td>
<td>64</td>
<td>8</td>
<td>64 B</td>
<td>4 clocks</td>
</tr>
<tr>
<td>L1D</td>
<td>8</td>
<td>32 KB</td>
<td>64</td>
<td>8</td>
<td>64 B</td>
<td>4 clocks</td>
</tr>
<tr>
<td>L2U</td>
<td>8</td>
<td>512 KB</td>
<td>1024</td>
<td>8</td>
<td>64 B</td>
<td>12 clocks</td>
</tr>
<tr>
<td>L3U</td>
<td>2</td>
<td>16 MB</td>
<td>16384</td>
<td>16</td>
<td>64 B</td>
<td>39 clocks</td>
</tr>
</tbody>
</table>
Refills within the same CCX may be relatively low cost!
- Some operating system schedulers are CCX aware.

CCX: Core Complex (4 Cores, 8 Logical Processors, 16MB).
- IFOP: Infinity Fabric™ On-Package.
  - CCM: Cache-Coherent Master has the memory map.
  - SDF Transport Layer: Scalable Data Fabric Transport Layer.
  - CS: Coherent Slave responsible for cache coherency.
  - Electrical interface between chiplets not shown.
- UMC: Unified Memory Controller.
• Minimize refills from local DRAM.

• CCX: Core Complex (4 Cores, 8 Logical Processors, 16MB).

• IFOP: Infinity Fabric™ On-Package.
  • CCM: Cache-Coherent Master has the memory map.
  • SDF Transport Layer: Scalable Data Fabric Transport Layer.
  • CS: Coherent Slave responsible for cache coherency.
  • Electrical interface between chiplets not shown.

• UMC: Unified Memory Controller.
**REFILL FROM LOCAL DRAM**

- Minimize refills from local DRAM.
- **CCX**: Core Complex (4 Cores, 8 Logical Processors, 16MB).
- **IFOP**: Infinity Fabric™ On-Package.
  - CCM: Cache-Coherent Master has the memory map.
  - SDF Transport Layer: Scalable Data Fabric Transport Layer.
  - CS: Coherent Slave responsible for cache coherency.
  - Electrical interface between chiplets not shown.
- **UMC**: Unified Memory Controller.
REFILL FROM ANY OTHER CCX

- Refill from any other CCX cost may be similar to memory latency.
- CCX: Core Complex (4 Cores, 8 Logical Processors, 16MB).
- IFOP: Infinity Fabric™ On-Package.
  - CCM: Cache-Coherent Master has the memory map.
  - SDF Transport Layer: Scalable Data Fabric Transport Layer.
  - CS: Coherent Slave responsible for cache coherency.
  - Electrical interface between chiplets not shown.
- UMC: Unified Memory Controller.
REFILL FROM ANY OTHER CCX

- Refill from any other CCX cost may be similar to memory latency.

- CCX: Core Complex (4 Cores, 8 Logical Processors, 16MB).
- IFOP: Infinity Fabric™ On-Package.
  - CCM: Cache-Coherent Master has the memory map.
  - SDF Transport Layer: Scalable Data Fabric Transport Layer.
  - CS: Coherent Slave responsible for cache coherency.
  - Electrical interface between chiplets not shown.
- UMC: Unified Memory Controller.
AMDUPROF PROFILER
NEW IN V3.2

**THREAD CONCURRENCY**
- Scaled chart for Threadripper™

**FLAME GRAPH**
- Sorted call stacks

**SYMBOLS**
- Improved symbol path support
7-zip 19.00 x64 benchmark “7z.exe b” shown.

Testing done by AMD technology labs, February 9, 2019 on the following system. Test configuration: AMD Ryzen™ Threadripper™ 3970X Processor, AMD Wraith Ripper™ Cooler, 64GB (4 x 16GB DDR4-3200 at 22-22-22-52) memory, Radeon™ RX 580 GPU with driver 20.1.3 (January 17, 2020), 2TB M.2 NVME SSD, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1909, 1920x1080 resolution. Actual results may vary.

You may need to run AMDuProf as administrator to see this advanced option.
Horizontal: normalized inclusive samples.
Vertical: call stack
Color: Module Name
### AMD RYZEN™ PROCESSOR SOFTWARE OPTIMIZATION

#### May 15, 2020

![AMDProfiler](C:/Users/amd/AMDu...-02-2019_17-30-20.db)

#### Home

- **Profile Samples**
- **Call Graph Samples**

#### Profile

**Filters and Options**

**View** | **Overall assessment** | **Group By** | **Process** | **Show cour**
--- | --- | --- | --- | ---
**Process**

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU clocks</th>
<th>IPC</th>
<th>DC miss rate</th>
<th>Misalign rate</th>
<th>Mispredict rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScimarkStable.exe (PID 22696)</td>
<td>269079</td>
<td>1.70</td>
<td>0.01</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>Load Modules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScimarkStable.exe</td>
<td>248299</td>
<td>1.80</td>
<td>0.01</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>[Sys] ntoskrnl.exe</td>
<td>19030</td>
<td>0.27</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>[Sys] ucrtbase.dll</td>
<td>365</td>
<td>3.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>[Sys] hal.dll</td>
<td>202</td>
<td>1.41</td>
<td>0.24</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>[Sys] ntdll.dll</td>
<td>70</td>
<td>0.46</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>[Sys] afld.sys</td>
<td>18</td>
<td>0.72</td>
<td>0.03</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>[Sys] atikmdag.sys</td>
<td>8</td>
<td>0.33</td>
<td>0.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Sys] amdppm.sys</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Analyze

**Search**: Type function name...

<table>
<thead>
<tr>
<th>Functions (for ScimarkStable.exe (PID 22696))</th>
<th>CPU clocks</th>
<th>IPC</th>
<th>DC miss rate</th>
<th>Misalign rate</th>
<th>Mispredict rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOR_execute</td>
<td>52352</td>
<td>0.65</td>
<td>0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random_nextDouble</td>
<td>52099</td>
<td>1.04</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SparseCompFlow_matmult</td>
<td>46034</td>
<td>2.78</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LU_factor</td>
<td>45505</td>
<td>2.41</td>
<td>0.02</td>
<td>0.06</td>
<td>0.00</td>
</tr>
<tr>
<td>FFT_transform_internal</td>
<td>25954</td>
<td>3.28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MonteCarlo_integrate</td>
<td>17268</td>
<td>1.04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ntoskrnl.exe (0xffffffff80000000c121)</td>
<td>12731</td>
<td>0.36</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>FFT_bitreverse</td>
<td>5612</td>
<td>1.77</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ntoskrnl.exe (0xffffffff80000000c9f52)</td>
<td>1400</td>
<td>0.57</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ntoskrnl.exe (0xffffffff80000000c1148)</td>
<td>1148</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Settings

**Load more profile data** | **Load more functions**

**Exclude** | **Include**
I recommend this profile.
I recommend enabling Call Stack Sampling (CSS) with Frame Pointer Omission (FPO) for Flame Graph Analysis.
A 5 second delay may allow you to change the foreground window before profiling starts.

I often collect 30 or 60 seconds of samples.
Enable loading from the Microsoft Symbol Server – especially if you have not defined _NT_SYMBOL_PATH.
OPTIMIZATIONS AND LESSONS LEARNED
TOPICS

- General Guidance
- Use Best Practices with Scalability
- Verify Parallel DX12 Pipeline State Creation
- Verify Parallel DX12 Command List Generation
- Use Best Practices with Locks
- Reorder Hot Struct Members
- Use Prefetch Level while iterating std::vector<T*>
GENERAL GUIDANCE
USE THE LATEST COMPILER & SDK

- “Zen 2” recommended compiler flags:
  - /GL /arch:AVX2 /MT /fp:fast /favor:blend
- JeMalloc may benefit some applications.
  - See http://jemalloc.net/

<table>
<thead>
<tr>
<th>Year</th>
<th>Visual Studio Changes</th>
<th>AMD Products (implicit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017</td>
<td><strong>Update v15.9.14 and later may improve AMD Ryzen™ memcpy/memset performance.</strong></td>
<td>“Summit Ridge”</td>
</tr>
</tbody>
</table>
USE A SUPPORTED INSTRUCTION SET

- Using `/arch:AVX` or `/arch:AVX2` may improve code gen of inline code.
  - `memcpy` & `memset` may be inline if the length is known at compile time.

- AVX is supported on many systems and growing over time.

- AVX512 is not supported by AMD processors and was present on less than 1% of users with Intel processors.

- Source: AMD User Experience Program Users Survey including 4 Million systems sampled from January 2019 to October 2019.
USE ALL PHYSICAL CORES

- This advice is specific to AMD processors and is not general guidance for all processor vendors.
- Generally, applications show SMT benefits and use of all logical processors is recommended.
- However, games often suffer from SMT contention on the main or render threads during gameplay.
  - One strategy to reduce this contention is to create threads based on physical core count rather than logical processor count.
  - Profile your application/game to determine the ideal thread count.
  - Recommend game options to:
    - Set Max Thread Pool Size
    - Force Thread Pool Size
    - Force SMT
    - Force Single NUMA Node (implicitly Group)
  - Avoid setting thread pool size as a constant.

See [https://gpuopen.com/cpu-core-count-detection-windows/](https://gpuopen.com/cpu-core-count-detection-windows/)

```c
DWORD get_default_thread_count() {
    DWORD cores, logical;
    get_processor_count(cores, logical);
    DWORD count = logical;
    char vendor[13];
    get_cpuid_vendor(vendor);
    if (0 == strcmp(vendor, "AuthenticAMD")) {
        if (0x15 == get_cpuid_family()) {
            // AMD "Bulldozer" family microarchitecture
            count = logical;
        } else {
            count = cores;
        }
    }
    return count;
}
```
DISABLE DEBUG FEATURES BEFORE YOU SHIP

While investigating open issues, developers may submit change requests which enable debug features on Test and Shipping configurations. These debug features may greatly reduce performance due to disabling multi-threading, cache pollution from STATS, and increased serialization from logging.

Some Unreal Engine settings to verify include:
- Build.h #define FORCE_USE_STATS and #define STATS
  - See 4.24/Engine/Source/Runtime/Core/Public/Misc/Build.h
- Parallel Rendering CVARS
  - See 4.24/Engine/Source/Runtime/RHI/Private/RHICommandList.cpp
  - See https://docs.unrealengine.com/en-US/Programming/Rendering/ParallelRendering

<table>
<thead>
<tr>
<th>Command</th>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r.rhicmdbypass</td>
<td>0</td>
</tr>
<tr>
<td>r.rhicmdusedeferredcontexts</td>
<td>1</td>
</tr>
<tr>
<td>r.rhicmduseparallelalgorithms</td>
<td>1</td>
</tr>
<tr>
<td>r.rhithread.enable</td>
<td>1</td>
</tr>
</tbody>
</table>
USE BEST PRACTICES WITH SCALABILITY
OPTIMIZE SCALABILITY FOR INTEGRATED GRAPHICS

- Goal: >= 60 FPS Average at 720p 100% Very Low
- Try:
  - Use DXGI_FORMAT_R11G11B10_FLOAT rather than DXGI_FORMAT_R16G16B16A16_FLOAT
  - Reduce shadow map quality
  - Reduce volumetric fog quality
  - Disable Ambient Occlusion
- For Unreal Engine
  - r.SceneColorFormat
  - r.AmbientOcclusionLevels=0
USE PROPER VIDEO MEMORY BUDGET FOR APU

- AGS SDK 5.4
  - Added isAPU flag.
    - If true, set the video memory budget to sharedMemoryInBytes for APU (AMD Accelerated Processing Unit with integrated graphics).
    - If false, set the video memory budget to localMemoryInBytes for discrete GPU.
  - Example:
    - unsigned long long memory_budget = (device.isAPU)? device.sharedMemoryInBytes: device.localMemoryInBytes;
  - See https://gpuopen.com/ags-sdk-5-4-improves-handling-video-memory-reporting-apus/
VERIFY PARALLEL DX12 PIPELINE STATE CREATION
VERIFY PARALLEL DX12 PIPELINE STATE CREATION

- Game shows parallel DX12 Pipeline State Creation.
- Performance of binary compiled with:
  - Microsoft® Visual Studio 2019 v16.4.5.
  - UnrealEngine-4.24.2-release from https://github.com/EpicGames/UnrealEngine
- Testing done by AMD technology labs, February 18, 2020 on the following system. Test configuration: AMD Ryzen™ 9 3950X Processor, AMD Wraith Prism Cooler, 16GB (2 x 8GB DDR4-3200 at 22-22-22-52) memory, Radeon™ VII GPU with driver 20.1.4 (January 24, 2020), 2TB M.2 NVME SSD, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1909, 1920x1080 resolution. Actual results may vary.
UE4.24 PARALLELIZED DX12 PIPELINE STATE CREATION 😊

Before

After

Hello Parallelism! 😊
TEST COLD SHADER CACHE

- Using a cold shader cache may simplify verifying if D3D12.dll!CDevice::CreatePipelineState was called in parallel.
- Install the Windows® SDK Windows® Performance Toolkit. Add the GPUView folder to the PATH.
- Applications and games may vary configurations of shader caches on disk yielding different results.
- Results may vary based on GPU vendor & driver versions used.

```plaintext
rmdir /s /q "%LOCALAPPDATA%\D3DSCache"
rmdir /s /q "%LOCALAPPDATA%\AMD\DxCache"
rmdir /s /q "%LOCALAPPDATA%\AMD\VkCache"
rmdir /s /q "%LOCALAPPDATA%\AMD\GLCache"
call log.cmd
pushd "C:\WindowsNoEditor"
start InfiltratorDemo.exe -dx12
popd
timeout.exe /t 10
call log.cmd
```
Cold shader cache shown.

Add CPU Usage (Precise).

Add Flame Graph, Find all D3D12.dll!CDDevice::CreatePipelineState.

See parallelism highlighted in CPU Usage (Precise). This is easiest to find using a cold shader cache.
Warm shader cache shown.

See parallelism highlighted in CPU Usage (Precise).
VERIFY PARALLEL DX12 COMMAND LIST GENERATION
VERIFY PARALLEL DX12 COMMAND LIST GENERATION

- Game shows parallel DX12 Command List Generation.
- Performance of binary compiled with:
  - Microsoft® Visual Studio 2019 v16.4.5.
  - UnrealEngine-4.24.2-release from https://github.com/EpicGames/UnrealEngine
- Testing done by AMD technology labs, February 13, 2020 on the following system. Test configuration: AMD Ryzen™ 9 3950X Processor, AMD Wraith Prism Cooler, 16GB (2 x 8GB DDR4-3200 at 22-22-22-52) memory, Radeon™ VII GPU with driver 20.1.4 (January 24, 2020), 2TB M.2 NVME SSD, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1909, 1920x1080 resolution. Actual results may vary.
Run: InfiltratorDemo.exe -dx12
Run as admin: timeout.exe /t 5
call log.cmd
timeout.exe /t 3
call log.cmd

Open merged.etl using the Windows® Performance Analyzer.

Zoom to single frame using Present markers.

Move CPU Column next to Task Name then filter and expand CommandList.
USE BEST PRACTICES WITH LOCKS
SUMMARY

- Use modern OS synchronization APIs
  - Recommended:
    - std::mutex
    - std::shared_mutex
    - SRWLock
    - EnterCriticalSection
  - May allow more efficient scheduling and longer battery life
SUMMARY

• Otherwise, for user spin locks:
  • Use the pause instruction
  • Alignas(64) lock variable
  • Test and test-and-set
  • Avoid lock prefix instructions
  • The OS may be unaware that threads are spinning; scheduling efficiency and battery life may be lost
  • Use spin locks only if held for a very short time
“ZEN 1” PERFORMANCE

- Binaries compiled using best practices show improved performance.
- Performance of binary compiled with Microsoft Visual Studio 2019 v16.4.5.
“Zen 2” improved SMT fairness for ALU schedulers.
  - This helps mitigate bad user spin lock code.

Binaries compiled using best practices show improved performance.

Performance of binary compiled with Microsoft Visual Studio 2019 v16.4.5.

Testing done by AMD technology labs, February 13, 2020 on the following system. Test configuration: AMD Ryzen™ 7 3700X Processor, AMD Wraith Prism Cooler, 16GB (2 x 8GB DDR4-3200 at 22-22-22-52) memory, Radeon™ RX 5700 XT GPU with driver 20.1.4 (January 24, 2020), 512GB M.2 NVME SSD, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1909, 1920x1080 resolution. Actual results may vary.
**“ZEN 2” PERFORMANCE**

- Binaries compiled using best practices shows improved idle.
- Performance of binary compiled with Microsoft Visual Studio 2019 v16.4.5.
- Testing done by AMD technology labs, February 13, 2020 on the following system. Test configuration: AMD Ryzen™ 7 3700X Processor, AMD Wraith Prism Cooler, 16GB (2 x 8GB DDR4-3200 at 22-22-22-52) memory, Radeon™ RX 5700 XT GPU with driver 20.1.4 (January 24, 2020), 512GB M.2 NVME SSD, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1909, 1920x1080 resolution. Actual results may vary.
PROFILING

- Use AMD uProf to find possible user spin locks
  - AMD uProf v3.2 "Assess Performance (Extended)" Event Based Sampling Profile
    - ALUTokenStall PTI
      - >= 3K Per Thousand Instructions is bad for top functions
    - Replace user spin locks with modern OS synchronization APIs when possible. Otherwise, use best practices.

- Use Microsoft Windows® Performance Analyzer to find call stacks using OS synchronization APIs
  - rem Recommend using public Microsoft symbol server
    - rem _NT_SYMBOL_PATH=srv*http://msdl.microsoft.com/download/symbols
  - rem “–start gpu –start video” wpr profiles are useful for game analysis for short durations
  - wpr.exe –setprofint 1221 –start power –filemode
  - test.exe
  - wpr.exe –stop log.etl
EXAMPLES SHARED CODE

#include "intrin.h"
#include "stdio.h"
#include "windows.h"
#include <chrono>
#include <numeric>
#include <thread>
#include <mutex>
#include <shared_mutex>

#define LEN 512
alignas(64) float b[LEN][4][4];
alignas(64) float c[LEN][4][4];

int main(int argc, char* argv[]) {
    using namespace std::chrono;
    float b0 = (argc > 1) ? strtof(argv[1], NULL) : 1.0f;
    float c0 = (argc > 2) ? strtof(argv[2], NULL) : 2.0f;
    std::fill((float*)b, (float*)(b + LEN), b0);
    std::fill((float*)c, (float*)(c + LEN), c0);
    int num_threads = std::thread::hardware_concurrency();
    HANDLE* threads = new HANDLE[num_threads];

    float b0 = (argc > 1) ? strtof(argv[1], NULL) : 1.0f;
    float c0 = (argc > 2) ? strtof(argv[2], NULL) : 2.0f;
    std::fill((float*)b, (float*)(b + LEN), b0);
    std::fill((float*)c, (float*)(c + LEN), c0);
    int num_threads = std::thread::hardware_concurrency();
    HANDLE* threads = new HANDLE[num_threads];

    high_resolution_clock::time_point t0 = 
        high_resolution_clock::now();

    for (size_t i = 0; i < num_threads; ++i) {
        threads[i] = CreateThread(NULL, 
            0, ThreadProcCallback, NULL, 0, NULL);
    }
    WaitForMultipleObjects(num_threads, 
        threads, TRUE, INFINITE);

    high_resolution_clock::time_point t1 = 
        high_resolution_clock::now();
    duration<double> time_span = 
        duration_cast<duration<double>>(t1 - t0);
    printf("time (milliseconds): %lf\n", 1000.0 * time_span.count());
    delete[] threads;
    return EXIT_SUCCESS;
}
EXAMPLE 1 BAD USER SPIN LOCK

```c
namespace MyLock {
    typedef unsigned LOCK, * PLOCK;
    enum { LOCK_IS_FREE = 0, LOCK_IS_TAKEN = 1 };
    void Lock(PLOCK pl) {
        while (LOCK_IS_TAKEN == \
            _InterlockedCompareExchange( \n                pl, LOCK_IS_TAKEN, LOCK_IS_FREE)) {
        }
    }
    void Unlock(PLOCK pl) {
        _InterlockedExchange(pl, LOCK_IS_FREE);
    }
}
```

Warning! Not best practices for spin lock.

```c
alignas(64) MyLock::LOCK gLock;
DWORD WINAPI ThreadProcCallback(LPVOID data) {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
        MyLock::Lock(&gLock);
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    for (int k = 0; k < 4; k++)
                        a[m][i][j] += b[m][i][k] * c[m][k][j];
        r += std::accumulate((float*)a, \
            (float*)(a + LEN), 0.0f);
        MyLock::Unlock(&gLock);
    }
    printf("result: %f\n", r);
    return 0;
}
```
Warning! 3K ALU Token Stalls PTI!
Warning! 26K ALU Token Stalls PTI!
Warning! No pause instruction in spin loop! 😁
EXAMPLE 2 IMPROVED USER SPIN LOCK

```cpp
namespace MyLock {
    typedef unsigned LOCK, * PLOCK;
    enum { LOCK_IS_FREE = 0, LOCK_IS_TAKEN = 1 };  
    void Lock(PLOCK pl) { 
        while ((LOCK_IS_TAKEN == *pl) || 
            (LOCK_IS_TAKEN == \
            _InterlockedExchange(pl, LOCK_IS_TAKEN))) { 
            _mm_pause();
        }
    }
    void Unlock(PLOCK pl) {
        _InterlockedExchange(pl, LOCK_IS_FREE);
    }
}

alignas(64) MyLock::LOCK gLock;
DWORD WINAPI ThreadProcCallback(LPVOID data) {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
        MyLock::Lock(&gLock);
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    for (int k = 0; k < 4; k++)
                        a[m][i][j] += b[m][i][k] * c[m][k][j];
        r += std::accumulate((float*)a, \
            (float*)(a + LEN), 0.0f);
        MyLock::Unlock(&gLock);
    }
    printf("result: \f\n", r);
    return 0;
}
```

Good! Applied best practices for spin lock.
Good! Low ALU Token Stalls PTI.
Good! Low ALU Token Stalls PTI.
Good! Pause instruction in spin loop.
std::mutex mutex;

DWORD WINAPI ThreadProcCallback(LPVOID data) {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
        mutex.lock();
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    for (int k = 0; k < 4; k++)
                        a[m][i][j] += b[m][i][k] * c[m][k][j];
        r += std::accumulate((float*)a, (float*)(a + LEN), 0.0f);
        mutex.unlock();
    }
    printf("result: %f\n", r);
    return 0;
}
Found msvcp140.dll

mtx_do_lock.
EXAMPLE 4 STD::SHARED_MUTEX

// MyLock not required. Let the OS do the work!

```c
std::shared_mutex mutex;
DWORD WINAPI ThreadProcCallback(LPVOID data) {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
        mutex.lock();
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    for (int k = 0; k < 4; k++)
                        a[m][i][j] += b[m][i][k] * c[m][k][j];
        r += std::accumulate((float*)a, 
                                (float*)(a + LEN), 0.0f);
        mutex.unlock();
    }
    printf("result: %f\n", r);
    return 0;
}
```
The compiler has substituted std::shared_mutex lock() for SRWLock.
EXAMPLE 5 SRWLOCK

// MyLock not required. Let the OS do the work!

SRWLOCK lock;

DWORD WINAPI ThreadProcCallback(LPVOID data) {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
        AcquireSRWLockExclusive(&lock);
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    for (int k = 0; k < 4; k++)
                        a[m][i][j] += b[m][i][k] * c[m][k][j];
        r += std::accumulate((float*)a,
                             (float*)(a + LEN), 0.0f);
        ReleaseSRWLockExclusive(&lock);
    }
    printf("result: %f\n", r);
    return 0;
}

int main(int argc, char* argv[]) {
    InitializeSRWLock(&lock);
    //
Found ntdll.dll
RtlAcquireSRWLock
Exclusive.
EXAMPLE 6 CRITICAL SECTION

// MyLock not required. Let the OS do the work!

CRITICAL_SECTION cs;
DWORD WINAPI ThreadProcCallback(LPVOID data) {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0f;
    for (size_t iter = 0; iter < 100000; iter++) {
        EnterCriticalSection(&cs);
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    for (int k = 0; k < 4; k++)
                        a[m][i][j] += b[m][i][k] * c[m][k][j];
        r += std::accumulate((float*)a, \
                (float*)(a + LEN), 0.0f);
        LeaveCriticalSection(&cs);
    }
    printf("result: %f\n", r);
    return 0;
}

int main(int argc, char* argv[]) {
    InitializeCriticalSection(&cs);
    // ...
Found ntdll.dll RtlpEnterCriticalSection. 😊
HANDLE hMutex;

DWORD WINAPI ThreadProcCallback(LPVOID data) {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
        WaitForSingleObject(hMutex, INFINITE);
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    for (int k = 0; k < 4; k++)
                        a[m][i][j] += b[m][i][k] * c[m][k][j];
        r += std::accumulate((float*)a, (float*)(a + LEN), 0.0f);
        ReleaseMutex(hMutex);
    }
    printf("result: %f\n", r);
    return 0;
}

int main(int argc, char* argv[]) {
    hMutex = CreateMutex(NULL, FALSE, NULL);
    // MyLock not required. Let the OS do the work!
    // ...
    return 0;
}
Found kernelbase.dll WaitForSingleObject. Recommending investigating if replacing WaitForSingleObject with SRWLock or similar is possible in these call stacks.
Recommended modern OS synchronization APIs:
- std::mutex
- std::shared_mutex
- SRWLock
- EnterCriticalSection
REORDER HOT STRUCT MEMBERS
SUMMARY

- Use AMDuProf to find plateaus of hot functions where there are many Data Cache refills from DRAM.
- If the hot function includes a loop which indirectly accesses struct data members spread over many cache lines, try reordering the struct.

```c
#if 0
/* bad */
struct S { double x, y, z, w; char name[256]; double s, t, u, v;};
#else
/* good */
struct S { double x, s, y, z, w; char name[256]; double t, u, v;};
#endif
for (size_t i = 0; i < list.size(); i++) {
    const S* e = list[i];
    foo(e->x);
    bar(e->s);
    ...}
```
• 12% faster after optimization!
• Performance of binaries compiled with Microsoft Visual Studio 2019 v16.4.5.
• Testing done by AMD technology labs, February 15, 2020 on the following system. Test configuration: 3rd Gen AMD Ryzen™ Engineering Sample manufactured with 4MB L3 Cache, AMD Wraith Prism Cooler, 16GB (2 x 8GB DDR4-3200 at 22-22-22-52) memory, NVidia® GeForce® RTX 2080 Ti GPU with driver 441.87 (December 24, 2019), 2TB M.2 NVME SSD, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1909, 1920x1080 resolution. Actual results may vary.
Found three large plateaus at getGlobalPose using many CPU clocks.
These same functions have many refills from DRAM.
No loop iteration found in `getGlobalPose`. But there are many cache accesses where misses are refilled from DRAM.
Loading mPxActor doesn't appear to use many CPU Clocks. But loading mLocalPose used many CPU Clocks and shows many refills from DRAM.

<table>
<thead>
<tr>
<th>Line</th>
<th>Offset</th>
<th>Source Code</th>
<th>CPU clocks</th>
<th>Retained</th>
<th>IPC</th>
<th>Retained Branch</th>
<th>%Retained Branch</th>
<th>Data Cache Access</th>
<th>%Data Cache Miss</th>
<th>Data Cache Refill</th>
<th>Data Cache Refill</th>
<th>Data Cache Refill</th>
<th>Data Cache Refill</th>
<th>Data Cache Refill</th>
<th>Data Cache Refill</th>
<th>Data Cache Refill</th>
<th>Data Cache Refill</th>
</tr>
</thead>
<tbody>
<tr>
<td>137</td>
<td>0</td>
<td>return center;</td>
<td></td>
<td>315</td>
<td>397</td>
<td>0.82</td>
<td>24.19</td>
<td>458.28</td>
<td>3.52</td>
<td>2.61</td>
<td>0.13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>0</td>
<td>if (mPxActor == NULL) {</td>
<td></td>
<td>70892</td>
<td>9221</td>
<td>0.89</td>
<td>24.90</td>
<td>458.31</td>
<td>4.21</td>
<td>2.67</td>
<td>0.47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>0</td>
<td>mPxActor-&gt;GetGlobalPose();</td>
<td></td>
<td>99556</td>
<td>13322</td>
<td>0.99</td>
<td>25.13</td>
<td>426.39</td>
<td>4.21</td>
<td>2.57</td>
<td>0.47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>0</td>
<td>mPxActor-&gt;GetLocalPose();</td>
<td></td>
<td>16</td>
<td>16</td>
<td>1.14</td>
<td>27.56</td>
<td>2.85</td>
<td>250.99</td>
<td>5.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>0</td>
<td>Test mPxActor</td>
<td></td>
<td>1</td>
<td>2</td>
<td>2.00</td>
<td>150.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>0</td>
<td>return mLocalPose;</td>
<td></td>
<td>1306</td>
<td>1650</td>
<td>0.71</td>
<td>27.56</td>
<td>0.15</td>
<td>411.77</td>
<td>6.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>0</td>
<td>else</td>
<td></td>
<td>29</td>
<td>179</td>
<td>0.81</td>
<td>31.28</td>
<td>3.05</td>
<td>391.00</td>
<td>5.57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>0</td>
<td>return mPxActor-&gt;GetLocalPose();</td>
<td></td>
<td>1904</td>
<td>2050</td>
<td>0.78</td>
<td>27.56</td>
<td>0.15</td>
<td>411.77</td>
<td>6.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>0</td>
<td>}</td>
<td></td>
<td>29</td>
<td>179</td>
<td>0.81</td>
<td>31.28</td>
<td>3.05</td>
<td>391.00</td>
<td>5.57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Found loop calling `getGlobalPose` in the call stack.
mMaterialOffset has refills from DRAM in hot loop too.
mSurfaceMaterialId & mMaterialId have refills from DRAM in hot loop too.
CODE SAMPLE

// Before Reorder Optimization

protected:
  void clear();
...

  Compound *mParent;
  PxRigidActor *mPxActor;
  PxTransform mLocalPose;
  PxConvexMesh *mPxConvexMesh;

  PxBounds3 mBounds;
  mutable float mVolume;
  mutable bool mVolumeDirty;
  PxVec3 mMaterialOffset;
  float mTexScale;
  int mModelIslandNr;
...

// After Reorder Optimization

protected:
  // Reorder Begin
  PxRigidActor* mPxActor;
  PxTransform mLocalPose;
  PxConvexMesh mPxConvexMesh;
  PxBounds3 mBounds;
  mutable float mVolume;
  mutable bool mVolumeDirty;
  PxVec3 mMaterialOffset;
  float mTexScale;
  int mModelIslandNr;
  // Reorder End

  void clear();
  void finalize();
  void updateBounds();
  void updatePlanes();
...
Before optimization hot data spread over many cache lines.

WinDbg
dt KaplaDemo!Convex
+0x000 __VFN_table : Ptr64
+0x008 mScene : Ptr64 physx::fracture::base::SimScene
...+0x0f8 mPxActor : Ptr64 physx::PxRigidActor
+0x100 mLocalPose : physx::PxTransform
...+0x148 mMaterialOffset : physx::PxVec3
...+0x160 mMaterialId : Int4B
+0x164 mSurfaceMaterialId : Int4B
... After optimization hot data spread on one cache line if struct aligned.

WinDbg
dt KaplaDemo!Convex
+0x000 __VFN_table : Ptr64
+0x008 mPxActor : Ptr64 physx::PxRigidActor
+0x010 mLocalPose : physx::PxTransform
+0x02c mMaterialOffset : physx::PxVec3
+0x038 mSurfaceMaterialId : Int4B
+0x03c mMaterialId : Int4B
+0x040 mScene : Ptr64 physx::fracture::base::SimScene
...
ACCESS PATTERN BEFORE OPTIMIZATION

- Contiguous array of pointers (8 Bytes) to data structures (372 Bytes each) with similar byte offset accesses.

<table>
<thead>
<tr>
<th>0 Pointer</th>
<th>1 Pointer</th>
<th>2 Pointer</th>
<th>3 Pointer</th>
<th>4 Pointer</th>
<th>5 Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
</tr>
</tbody>
</table>

Byte Offset

-1 - 0 - 1

0 64 128 192 256 320 384

-1 - 0 - 1

0 64 128 192 256 320 384

-1 - 0 - 1

0 64 128 192 256 320 384

-1 - 0 - 1

0 64 128 192 256 320 384

-1 - 0 - 1

0 64 128 192 256 320 384

-1 - 0 - 1

0 64 128 192 256 320 384
ACCESS PATTERN AFTER OPTIMIZATION

- Contiguous array of pointers (8 Bytes) to data structures (372 Bytes each) with similar byte offset accesses.

<table>
<thead>
<tr>
<th>0 Pointer</th>
<th>1 Pointer</th>
<th>2 Pointer</th>
<th>3 Pointer</th>
<th>4 Pointer</th>
<th>5 Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>0</th>
<th>64</th>
<th>128</th>
<th>192</th>
<th>256</th>
<th>320</th>
<th>384</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>192</td>
<td>256</td>
<td>320</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td>320</td>
<td>384</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>384</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ACCESS PATTERN AFTER OPTIMIZATION

<table>
<thead>
<tr>
<th>0 Pointer</th>
<th>1 Pointer</th>
<th>2 Pointer</th>
<th>3 Pointer</th>
<th>4 Pointer</th>
<th>5 Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>0</th>
<th>64</th>
<th>128</th>
<th>192</th>
<th>256</th>
<th>320</th>
<th>384</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>192</td>
<td>256</td>
<td>320</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td>320</td>
<td>384</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>384</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ACCESS PATTERN AFTER OPTIMIZATION

<table>
<thead>
<tr>
<th>0 Pointer</th>
<th>1 Pointer</th>
<th>2 Pointer</th>
<th>3 Pointer</th>
<th>4 Pointer</th>
<th>5 Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
<td>Data Structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>0</th>
<th>64</th>
<th>128</th>
<th>192</th>
<th>256</th>
<th>320</th>
<th>384</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>192</td>
<td>256</td>
<td>320</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td>320</td>
<td>384</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>384</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Before Optimization.
After Optimization. Plateaus are relatively smaller for updateTransformations.
Before Optimization.
After Optimization. Plateaus are relatively smaller for updateTransformations.
Before Optimization.
After Optimization.

IPC is higher at lines of interest.
Before Optimization.
After Optimization.

IPC is higher at lines of interest.
BUT CAN WE DO BETTER?
YES!
USE PREFETCH LEVEL WHILE ITERATING STD::VECTOR<T*>
SUMMARY

- Use AMDuProf to find hot functions where there are many Data Cache refills from DRAM.
- If many refills from DRAM are while iterating std::vector<T*>, try using the _mm_prefetch intrinsic to improve performance.
- The distance to prefetch and the prefetch level <NTA|T0> may require some tuning.
- Try prefetch distance 4 and prefetch level NTA.
- For public & friend private data, try using _mm_prefetch((char*)(&v[future]->data), _MM_HINT_NTA);
- For private data, use WinDbg "dt <T>" to determine the offset of the hot data then try using _mm_prefetch(offset + (char*)(v[future]), _MM_HINT_NTA);

// Example
struct S { double x, y, z, w; char name[256]; double s, t, u, v;};

int main() {
    std::vector<S*> v;
    // initialize v
    for (size_t i = 0; i < v.size(); i++) {
        size_t distance = 4; // TODO find ideal number
        size_t future = (i + distance) % v.size();
        _mm_prefetch((char*)((char*)&(v[future]->x), _MM_HINT_NTA);
        _mm_prefetch((char*)&(v[future]->s), _MM_HINT_NTA);
        foo(v[i]->x);
        bar(v[i]->s);
    }
}
PERFORMANCE

- 60% faster after optimization!
- Performance of binaries compiled with Microsoft Visual Studio 2019 v16.4.5.
- Testing done by AMD technology labs, February 15, 2020 on the following system. Test configuration: 3rd Gen AMD Ryzen™ Engineering Sample manufactured with 4MB L3 Cache, AMD Wraith Prism Cooler, 16GB (2 x 8GB DDR4-3200 at 22-22-22-52) memory, NVidia® GeForce® RTX 2080 Ti GPU with driver 441.87 (December 24, 2019), 2TB M.2 NVME SSD, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1909, 1920x1080 resolution. Actual results may vary.
void ConvexRenderer::updateTransformations() {
    for (int i = 0; i < (int)mGroups.size(); i++) {
        ConvexGroup *g = mGroups[i];
        if (g->texCoords.empty())
            continue;
        float* tt = &g->texCoords[0];
        for (int j = 0; j < (int)g->convexes.size(); j++) {
            const Convex* c = g->convexes[j];
            #if 1
            int distance = 4; // TODO find ideal number
            size_t future = (j + distance) % g->convexes.size();
            __mm_prefetch(0x0F8 + (char*)(g->convexes[future]), __MM_HINT_NTA); // mPxActor
            __mm_prefetch(0x100 + (char*)(g->convexes[future]), __MM_HINT_NTA); // mLocalPose
            __mm_prefetch(0x148 + (char*)(g->convexes[future]), __MM_HINT_NTA); // mMaterialOffset.x
            __mm_prefetch(0x14C + (char*)(g->convexes[future]), __MM_HINT_NTA); // mMaterialOffset.y
            __mm_prefetch(0x150 + (char*)(g->convexes[future]), __MM_HINT_NTA); // mMaterialOffset.z
            __mm_prefetch(0x164 + (char*)(g->convexes[future]), __MM_HINT_NTA); // mSurfaceMaterialId
            __mm_prefetch(0x168 + (char*)(g->convexes[future]), __MM_HINT_NTA); // mMaterialId
            #endif
            PxMat44 pose(c->getGlobalPose());
            float* mp = (float*)pose.front();
            float* ta = tt;
            for (int k = 0; k < 16; k++) {
                *(tt++) = *(mp++);
            }
            PxVec3 matOff = c->getMaterialOffset();
            ta[3] = matOff.x;
            ta[7] = matOff.y;
            int idFor2DTex = c->getSurfaceMaterialId();
            int idFor3DTex = c->getMaterialId();
            const int MAX_3D_TEX = 8;
            ta[15] = (float)(idFor2DTex*MAX_3D_TEX + idFor3DTex);
        }
        glBindTexture(GL_TEXTURE_2D, g->matTex);
        glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, g->texSize, g->texSize, GL_RGBA,
                        GL_FLOAT, &g->texCoords[0]);
        glBindTexture(GL_TEXTURE_2D, 0);
    }
}
Before Optimization.
After Optimization. Plateaus are relatively smaller for `updateTransformations`. 
Before Optimization.
After Optimization. Plateaus are relatively smaller for `updateTransformations`.
Before Optimization.
After Optimization.
IPC is higher at lines of interest.
Before Optimization.
After Optimization.

IPC is higher at lines of interest.
After Optimization. 

IPC is high at prefetchnta lines with sufficient samples,
THANK YOU
FURTHER READING

OPTIMIZATION GUIDE

Software Optimization Guide for AMD Family 17h Models 30h and Greater Processors

https://developer.amd.com/resources/developer-guides-manuals/

THE PATH TO “ZEN 2”

https://www.slideshare.net/AMD/the-path-to-zen-2

AGS SDK 5.4

https://gpuopen.com/ags-sdk-5-4-improves-handling-video-memory-reporting-apus/
CONTACT INFORMATION

• Front Row:
  • Ken Mitchell
    • Team Lead/Platform Specialist
    • Kenneth.Mitchell@amd.com
  • John Hartwig
    • Engine/Scalability Specialist
    • John.Hartwig@amd.com

• Back Row:
  • Elliot Kim
    • Physics/Simulation Specialist
    • Elliot.Kim@amd.com
DISCLAIMER AND ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2020 Advanced Micro Devices, Inc. All rights reserved. AMD, Ryzen™, and the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Vulkan and the Vulkan logo are registered trademarks of Khronos Group Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation. PCIe is a registered trademark of PCI-SIG. Other names are for informational purposes only and may be trademarks of their respective owners.
DISCLAIMER AND ATTRIBUTION (MORE)

“Code Sample excerpted on slides 92 and 109 are modifications Copyright (c) 2020 Advanced Micro Devices, Inc. All Rights Reserved. Copyright (c) 2019 NVIDIA Corporation. All rights reserved. Code Sample is licensed subject to the following: “Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of NVIDIA CORPORATION nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.”
DISCLAIMER AND ATTRIBUTION (MORE)

- Slide 10: Claim: 15% IPC Improvement
  - AMD “Zen 2” CPU-based system scored an estimated 15% higher than previous generation AMD “Zen” based system using estimated SPECInt®_base2006 results. SPEC and SPECint are registered trademarks of the Standard Performance Evaluation Corporation. See www.spec.org. GD-141

- Slides 47, 48, 50, 51, 53, 54, 72, 74, 76, 78, and 80 Windows Performance Analyzer used with permission from Microsoft