AMD RYZEN™ PROCESSOR SOFTWARE OPTIMIZATION

PRESENTED BY
KEN MITCHELL
ABSTRACT

• Join AMD ISV Game Engineering team members for an introduction to the AMD Ryzen™ family of processors followed by advanced optimization topics. Learn about the Ryzen™ line up of processors, profiling tools and techniques to understand optimization opportunities, and get a glimpse of the next generation of “Zen 2” x86 core architecture. Gain insight into code optimization opportunities and lessons learned with examples including C/C++, assembly, and hardware performance-monitoring counters.

• Ken Mitchell is a Senior Member of Technical Staff in the Radeon™ Technologies Group/AMD ISV Game Engineering team where he focuses on helping game developers utilize AMD CPU cores efficiently. His previous work includes automating & 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” Family Processors
• AMD μProf Profiler
• Optimizations & Lessons Learned
• Roadmap
• Questions
• Giveaway
SUCCESS STORIES
SUCCESS STORIES

World of Warcraft
v8.1 DX12
(higher is better)

DOTA 2
version 3,359; gameplay 7.21B
(higher is better)

Audacity
LAME MP3 Encode x86
(lower is better)

see disclaimer and testing details in next slide.
DISCLAIMER

• **World of Warcraft**
  - [Link](https://community.amd.com/community/gaming/blog/2019/02/08/ryzen-processors-are-now-optimized-for-both-alliance-and-horde-in-world-of-warcraft)
  - Testing done by AMD performance labs January 4, 2019 on the following system. PC manufacturers may vary configurations yielding different results. Results may vary based on driver versions used. Test configuration: AMD Ryzen™ 7 2700X Processor, 2x8GB DDR4-3200, GeForce GTX 1080 (driver 398.82), MSI B450 GAMING PLUS Socket AM4 motherboard, Samsung 850 SSD, Windows 10 x64 Pro (RS4).

• **DOTA 2**
  - Testing done by Kenneth Mitchell February 18, 2019 on the following system. PC manufacturers may vary configurations yielding different results. Results may vary based on driver versions used. Test configuration: AMD Ryzen™ 5 2400G Processor, 2x8GB DDR4-3200 (14-14-14-34), Radeon™ RX Vega 11 (driver 19.1.1 Jan20), AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1809, Radeon™ R7 SSD, 1920x1080 resolution, Best Looking. DOTA 2 version 3,359; gameplay 7.21B. Steam Beta Built Feb 15 2019, at 15:03:34.

• **Audacity Lame MP3 Encode**
  - AMD Ryzen™ 7 2700X Processor, 16GB Corsair Vengeance DDR4-3200 at 2933, NVIDIA GeForce GTX 1080Ti 11GB (driver 390.77), ASUS Crosshair VII Hero motherboard, Corsair Neutron XTi 480 SSD, Windows 10 Pro x64 RS3, fully updated as of 4/1/2017.
“ZEN” FAMILY PROCESORS
"PINNACLE RIDGE" 8 CORE PROCESSOR

- **64K I-Cache**: 4-way, 32B/cycle
- **32K D-Cache**: 8-way, 32B/cycle
- **512K L2**: I+D Cache, 8-way
- **8M L3**: I+D Cache, 16-way
- **Data Fabric**: 32B/cycle
- **Unified Memory Controller**: 32B/cycle
- **DRAM Channel**: 16B/cycle
- **IO Hub Controller**: 32B/cycle

Clock Frequencies:
- **cclk**: 4.3 GHz / 3.7 GHz
- **fclk**: 1.47 GHz (DDR4-2933)
- **Iclk**: 615 MHz

Clock Relationships:
- **fclk = uclk = memclk**
- **CCLK = 4.3 GHz**
- **ICLK = 615 MHz**
"RAVEN RIDGE" 4 CORE PROCESSOR

- **64K I-Cache** (4-way)
- **32K D-Cache** (8-way)
- **512K L2** (8-way)
- **4M L3** (16-way)

**Data Fabric**

- **Unified Memory Controller**
- **DRAM Channel**

**IO Hub Controller**

- **GFX9**
- **Media**

**Key Parameters**

- cclk 3.9 GHz / 3.6 GHz
- fclk = uclk = memclk 1.47 GHz (DDR4-2933)
- lclk 496 MHz
- sclk 1.25 GHz (11CU = 704 shaders)
• 16 cores / 32 threads
• 4 DDR Channels
  • ~64ns near memory *
  • ~105ns far memory *
• NUMA or UMA
• The AMD Ryzen™ Master Utility Game Mode may improve performance by effectively improving memory latency by restricting the system to use only die0 processors while in NUMA mode
• 64 PCIe® Gen3 lanes
• 50GB/s die-to-die bandwidth (bi-directional) *
• * RP2-21 DRAM latency for Die0 or Die1 communicating with their respective local memory pool(s): approximately 64ns with DDR4-3200. DRAM Latency for Die0 or Die1 communicating with the other die’s memory pool: approximately 105ns with DDR4-3200. Die-to-die bandwidth of the Infinity Fabric with DDR4-3200 measured at approximately 50GBps. AMD System configuration: AMD Ryzen™ Threadripper™ 2950X and 1950X, Corsair H100i CLC, 4x8GB DDR4-3200 (14-14-14-28-1T), Asus Zenith X399 Extreme (BIOS 0008), GeForce GTX 1080 Ti (driver 398.36), Windows® 10 x64 1803, Samsung 850 Pro SSD, Western Digital Black 2TB HDD. Results may vary with configuration. RP2-21
**RYZEN™ THREADRIPPER™ 32 CORE PROCESSOR**

- 32 cores / 64 threads
- 4 DDR Channels
  - ~64ns near memory *
  - ~105ns far memory *
  - NUMA only
    - Windows 10 2019H1 Insider Preview has improved NUMA support
  - The AMD Ryzen™ Master Utility Game Mode may improve performance by effectively improving memory latency by restricting the system to use only die0 processors
- 64 PCIe® Gen3 lanes
- 25GB/s die-to-die bandwidth (bi-directional) *

* RP2-22 DRAM latency for Die0 or Die2 communicating with their respective local memory pool(s): approximately 64ns with DDR4-3200. DRAM Latency for Die0 or Die2 communicating with the other die’s memory pool: approximately 105ns with DDR4-3200. Die-to-die bandwidth of the Infinity Fabric with DDR4-3200 measured at approximately 25GBps. AMD System configuration: AMD Ryzen™ Threadripper™ 2990WX and 1950X, Corsair H100i CLC, 4x8GB DDR4-3200 (16-18-18), Asus Zenith X399 Extreme (BIOS 0008), GeForce GTX 1080 Ti (driver 398.36), Windows® 10 x64 1803, Samsung 850 Pro SSD, Western Digital Black 2TB HDD. Results may vary with configuration. RP2-22
MICROARCHITECTURE
ZEN SMT DESIGN OVERVIEW

- All structures available in 1T mode
- Front End Queues are round robin with priority overrides
- High throughput from SMT
- AMD Ryzen™ achieved a greater than 52% increase in IPC than previous generation AMD processors

1. Testing by AMD Performance labs. PC manufacturers may vary configurations yielding different results. System configs: AMD reference motherboard(s), AMD Radeon™ R9 290X GPU, 8GB DDR4-2667 (“Zen”), 8GB DDR3-2133 (“Excavator”), Ubuntu Linux 16.x (SPECint06) and Windows® 10 x64 RS1 (Cinebench R15). Updated Feb 28, 2017: Generational IPC uplift for the “Zen” architecture vs. “Piledriver” architecture is +52% with an estimated SPECint_base2006 score compiled with GCC 4.6 –O2 at a fixed 3.4GHz. Generational IPC uplift for the “Zen” architecture vs. “Excavator” architecture is +64% as measured with Cinebench R15 1T, and also +64% with an estimated SPECint_base2006 score compiled with GCC 4.6 –O2, at a fixed 3.4GHz. System configs: AMD reference motherboard(s), AMD Radeon™ R9 290X GPU, 8GB DDR4-2667 (“Zen”), 8GB DDR3-2133 (“Excavator”), Ubuntu Linux 16.x (SPECint_base2006 estimate) and Windows® 10 x64 RS1 (Cinebench R15). SPECint_base2006 estimates: “Zen” vs. “Piledriver” (31.5 vs. 20.7 | +52%), “Zen” vs. “Excavator” (31.5 vs. 19.2 | +64%). Cinebench R15 1T scores: “Zen” vs. “Piledriver” (130 vs. 79 both at 3.4G | +76%), “Zen” vs. “Excavator” (100 vs. 87.5 both at 4.0G | +16%). RZN-11
Family 17h added the AMD vendor specific instruction CLZERO and deprecated support for FMA4, TBM, & XOP.
FLOAT INSTRUCTIONS

- There can be a significant penalty for mixing SSE and AVX instructions when the upper 128 bits of the YMM registers contain non-zero data
  - Avoid mixing legacy SIMD and AVX instructions
  - Zero the upper 128 bits of all YMM registers before executing any legacy SIMD instructions by using the VZEROUPPER or VZEROALL instruction
  - Zero the upper 128 bits of all YMM registers after leaving a legacy SIMD section of code by using the VZEROUPPER or VZEROALL instruction
- Results from x86 processors from different vendors may not match exactly for instructions RCPPS, RCPSS, RSQRTPS, RSQRTSS which define relative error as:
  - $|\text{Relative Error}| \leq 1.5 \times 2^{-12}$
CLZERO INSTRUCTION

- AMD Vendor specific instruction.
- An atomic MOVNT streaming store type instruction that writes an entire 64B cache line full of zeros to memory and clears poisoned status.
- Intended to recover from some otherwise fatal Machine Check Architecture (MCA) errors caused by uncorrectable corrupt memory.
  - Example: kill a corrupt user process but keep the system running.
- Execution Path: Store Queue, Store Commit, Write Combining Buffer, L2, Data Fabric, Memory.
- Is non-cacheable, unlike the PowerPC DCBZ instruction.
- Use memset rather than the CLZERO intrinsic to quickly zero memory.
CACHE LATENCY

- 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 a CCX.
  - L2 tags are duplicated in the L3 for fast cache transfers within a CCX.
    - And fast probe filtering for Ryzen™ Threadripper™ and Epyc™.
- L1 capacity evictions may cause L2 capacity evictions and L3 capacity evictions.

<table>
<thead>
<tr>
<th>Level</th>
<th>Count</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>2 K uops</td>
<td>32</td>
<td>8</td>
<td>8 uops</td>
<td>NA</td>
</tr>
<tr>
<td>L1I</td>
<td>8</td>
<td>64 KB</td>
<td>256</td>
<td>4</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>8 MB</td>
<td>8192</td>
<td>16</td>
<td>64 B</td>
<td>35 clocks</td>
</tr>
</tbody>
</table>

![Cache Latency](cache-latency.png)

<table>
<thead>
<tr>
<th></th>
<th>L1D</th>
<th>L2U</th>
<th>L3U</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Ryzen™ 7 1800X</td>
<td>4</td>
<td>17</td>
<td>40</td>
</tr>
<tr>
<td>AMD Ryzen™ 7 2700X</td>
<td>4</td>
<td>12</td>
<td>35</td>
</tr>
</tbody>
</table>
REFILL FROM LOCAL DRAM

- ~64ns near memory *
  - RP2-22 DRAM latency for Die0 or Die2 communicating with their respective local memory pool(s): approximately 64ns with DDR4-3200. DRAM Latency for Die0 or Die2 communicating with the other die’s memory pool: approximately 105ns with DDR4-3200. Die-to-die bandwidth of the Infinity Fabric with DDR4-3200 measured at approximately 25GBps. AMD System configuration: AMD Ryzen™ Threadripper™ 2990WX and 1950X, Corsair H100i CLC, 4x8GB DDR4-3200 (16-18-18), Asus Zenith X399 Extreme (BIOS 0008), GeForce GTX 1080 Ti (driver 398.36), Windows® 10 x64 1803, Samsung 850 Pro SSD, Western Digital Black 2TB HDD. Results may vary with configuration. RP2-22

- CCX: Core Complex
- CCM: Cache-Coherent Master
- SDF: Scalable Data Fabric
- CAKE: Coherent AMD socKet Extender
- IFIS: Infinity Fabric Inter-Socket SerDes
- IFOP: Infinity Fabric On-Package SerDes
- CS: Coherent Slave
- UMC: Unified Memory Controller
REFILL FROM OTHER LOCAL CCX

- Refill from other CCX cost may be similar to memory latency
- CCX: Core Complex
- CCM: Cache-Coherent Master
- SDF: Scalable Data Fabric
- CAKE: Coherent AMD socKet Extender
- IFIS: Infinity Fabric Inter-Socket SerDes
- IFOP: Infinity Fabric On-Package SerDes
- CS: Coherent Slave
- UMC: Unified Memory Controller
REFILL FROM REMOTE DIE DRAM

• ~105ns far memory *

* RP2-22 DRAM latency for Die0 or Die2 communicating with their respective local memory pool(s): approximately 64ns with DDR4-3200. DRAM Latency for Die0 or Die2 communicating with the other die’s memory pool: approximately 105ns with DDR4-3200. Die-to-die bandwidth of the Infinity Fabric with DDR4-3200 measured at approximately 25GBps. AMD System configuration: AMD Ryzen™ Threadripper™ 2990WX and 1950X, Corsair H100i CLC, 4x8GB DDR4-3200 (16-18-18), Asus Zenith X399 Extreme (BIOS 0008), GeForce GTX 1080 Ti (driver 398.36), Windows® 10 x64 1803, Samsung 850 Pro SSD, Western Digital Black 2TB HDD. Results may vary with configuration. RP2-22
V2.0 NEW FEATURES

• Remote profiling
• Thread Concurrency
• Designed to support Multiple counters using the same event but different unit masks supported
• “Assess Performance (Extended)” event based sampling profile updated
• See https://developer.amd.com/amd-uprof/

• Open-Source Register Reference For AMD Family 17h Processors (Publication #: 56255)
  • See http://support.amd.com/en-us/search/tech-docs
THREAD CONCURRENY EXAMPLE

Threads shown are hardware threads (a.k.a. logical processors)

Testing done by Kenneth Mitchell January 18, 2019 on the following system. PC manufacturers may vary configurations yielding different results. Results may vary based on driver versions used. Test configuration: AMD Ryzen™ 7 2700X Processor, 2x8GB DDR4-3200 (16-18-18-36), Radeon™ RX 580, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1809, 1920x1080 resolution.
BIOS core clock = 3.7GHz, Core Performance Boost = Disable, Global C-state Control = Disable, Wolfenstein 2 / SAS Machine / Finale / 1080p High novsync.
ASSESS PERFORMANCE (EXTENDED) EXAMPLE

Analyze issues using hardware event based sampling.

- CPU clocks
- Ret inst
- IPC
- Retired Branch Instructions PTI
- %Retired Branch Instructions Mispredicted
- Data Cache Accesses PTI
- %Demand Data Cache Miss
- %Data Cache Miss
- Data Cache Refills DRAM PTI
- Data Cache Refills CCX PTI
- Data Cache Refills L2 PTI
- Misalign Loads PTI
- ALUTokenStall PTI
- CacheableLocks PTI
- StiOther PTI
- WcbFull PTI
PERFORMANCE COUNTER DOMAINS

IC/BP: instruction cache and branch prediction

DE: instruction decode, dispatch, microcode sequencer, & micro-op cache

EX (SC): integer ALU & AGU execution and scheduling

FP: floating point

LS: load/store

L2

L3

DF: Data Fabric

UMC: Unified Memory Controller (NDA only)

IOHC: IO Hub Controller (NDA only)

rdpmc

SMN in/out
POWER MANAGEMENT

- Disabling power management features may reduce variation during AB testing
  - BIOS Settings
    - “Zen” Common Options
      - Core Performance Boost = Disable
      - Global C-state Control = Disable
  - OS Power Options Choose Power Plan
    - High Performance = selected
  - AMD Ryzen™ Master Utility
    - Control Mode = Manual
    - All Cores = Enabled
    - Set a reasonable frequency & voltage such as P0 custom default
      - Set core clock > base clock to disable boost on “Raven Ridge” processors
      - Note SMU may still reduce frequency if application exceeds power, current, thermal limits
OPTIMIZATIONS AND LESSONS LEARNED
AGENDA

• Use general guidance
• Use best practices when counting cores
• Build commands lists in parallel
• Use best practices with spinlocks
• Avoid memcpy & memset regression
• Avoid too many non-temporal streams
• Avoid false sharing
USE GENERAL GUIDANCE
**Generate better code for current and past processors.**

<table>
<thead>
<tr>
<th>Year</th>
<th>Visual Studio Changes</th>
<th>AMD Product Family (implicit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>Added nullptr keyword. Replaced VCBuild with MSBuild.</td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>Added x64 native compiler.</td>
<td>“K8”</td>
</tr>
</tbody>
</table>
• Binary compiled after applying platform:x64 shows higher performance
  • Configuration:ReleaseSSE2 uses SSE2 intrinsics for some functions
  • See http://lame.sourceforge.net/

• Performance of binary compiled with Microsoft Visual Studio 2017 v15.9.6
  • Testing done by Kenneth Mitchell February 9, 2019 on the following system. PC manufacturers may vary configurations yielding different results. Results may vary based on driver versions used. Test configuration: AMD Ryzen™ 7 2700X Processor, 2x8GB DDR4-3200 (16-18-18-36), Radeon™ RX Vega 64, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1809, 1920x1080 resolution. BIOS core clock = 3.7GHz, Core Performance Boost = Disable, Global C-state Control = Disable
  • WAV file 448MB, 44.4 minutes
TEST CPUID BEFORE CALLING INSTRUCTIONS

- AMD “Pinnacle Ridge” processors will throw an ILLEGAL_INSTRUCTION exception if AVX512 is used
- Do not use FMA4 instructions if they are not enumerated by CPUID
- More commonly, AMD “Greyhound” & “Llano” processors will throw an ILLEGAL_INSTRUCTION exception if SSSE3 is used
  - ISPC https://github.com/ispc/ispc
    - x86-x64 issue fixed November 21, 2018
    - CPU_Pentium4 issue fixed July 12, 2018
- Masked Occlusion Culling https://github.com/GameTechDev/MaskedOcclusionCulling
  - pabsd issue fixed November 2nd, 2018
- Bullet3 https://github.com/bulletphysics/bullet3/
  - _xgetbv issue fixed June 20th, 2014
- Windows® 10 x64 requires: SSE2 & PrefetchW
  - See https://docs.microsoft.com/en-us/windows-hardware/design/minimum/minimum-hardware-requirements-overview
- The AMD64 Instruction Set Architecture includes SSE2 & PrefetchW
- Windows 7 x86 does NOT require SSE2 & PrefetchW!

0:000> g
(79c.838): Illegal instruction - code c000001d (first chance)
(79c.838): Illegal instruction - code c000001d (!!! second chance !!!)
mod!foo::bar+0xb63:
  00007ff7`25a78663 660f381ed4      pabsd xmm2,xmm4
  00007ff7`25a78663 660f381ed4      pabsd xmm2,xmm4
USE BEST PRACTICES WHEN COUNTING CORES
**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.
  - But games often suffer from SMT contention on the main thread.
  - 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.
  - AMD “Bulldozer” is not a SMT design.
- Avoid core clamping.
- See [https://gpuopen.com/cpu-core-count-detection-windows/](https://gpuopen.com/cpu-core-count-detection-windows/)

```c
DWORD getDefaultThreadCount() {
    DWORD cores, logical;
    getProcessorCount(cores, logical);
    DWORD count = logical;
    char vendor[13];
    getCpuidVendor(vendor);
    if (0 == strcmp(vendor, "AuthenticAMD")) {
        if (0x15 == getCpuidFamily()) {
            // AMD "Bulldozer" family microarchitecture
            count = logical;
        } else {
            count = cores;
        }
    }
    return count;
}
```
AVOID THE 2009 AMD PROCESSOR AND CORE ENUMERATION CODE SAMPLE

- Deprecated enum.c code sample from June 30, 2009 does not work properly on AMD family 17h processors.
  - Number of cores per processor & Number of threads per processor return incorrect values.

- See [https://github.com/GPUOpen-LibrariesAndSDKs/cpu-core-counts](https://github.com/GPUOpen-LibrariesAndSDKs/cpu-core-counts)

Incorrect enum.exe output for Ryzen™ 7 2700X:

Physical Processor ID 0 has 16 cores as logical processors 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Number of active logical processors: 16
Number of active physical processors: 1
Number of cores per processor: 16
Number of threads per processor core: 1

--------------------------------------------------------------------------------

Correct output:

Number of cores per processor: 8
Number of threads per processor core: 2
AVOID SIGNED, NARROWING AFFINITY MASKS

• Avoid signed, narrowing affinity masks.
  • Otherwise, the application may crash or exhibit unexpected behavior.
  • By default, an application is constrained to a single group, a static set of up to 64 logical processors.
  • Right Shifts: For signed numbers, the sign bit is used to fill the vacated bit positions.
  • Left Shifts: If you left-shift a signed number so that the sign bit is affected, the result is undefined.
GET GETLOGICALPROCESSORINFORMATIONEX BUFFER LENGTH AT RUNTIME

• Get PSYSTEM_LOGICAL_PROCESSOR_INFORMATION_EX buffer length at runtime.
  • Otherwise, the application may crash if an insufficiently sized buffer was created at compile time.
  • See https://github.com/GPUOpen-LibrariesAndSDKs/cpu-core-counts/blob/master/windows/ThreadCount-Win7.cpp

• WinDbg Commands:
  • bp kernelbase!GetLogicalProcessorInformation ".printf "FOUND GetLogicalProcessorInformation Buffer Length 0x%x\n",poi(@rdx);"
  • bp kernelbase!GetLogicalProcessorInformationEx ".printf "FOUND GetLogicalProcessorInformationEx Buffer Length 0x%x\n",poi(@r8);"

```c
// char buffer[0x1000] /* bad assumption */
char* buffer = NULL;
DWORD len = 0;
if (FALSE == GetLogicalProcessorInformationEx( 
    RelationAll, 
    (PSYSTEM_LOGICAL_PROCESSOR_INFORMATION_EX)buffer, 
    &len)) {
    if (GetLastError() == ERROR_INSUFFICIENT_BUFFER) {
        printf("len = 0x%x\n", len);
        buffer = (char*)malloc(len);
        GetLogicalProcessorInformationEx( 
            RelationAll, 
            (PSYSTEM_LOGICAL_PROCESSOR_INFORMATION_EX)buffer, 
            &len)) {
            // ...
        }
    } 
    free(buffer);
```
**PROCESSORCOUNTLIE SHIM**

- It may not be practical for developers to fix application code bugs related to high processor count long after release.
  - Costs may include restoring source from an archive, buying more servers, hiring more engineers, coding, testing, and releasing.

- Although application updates are strongly preferred, AMD and the Microsoft Windows Compatibility team collaborated on a **last-resort** Application Compatibility Shim.

- The ProcessorCountLie shim passes limited CPU topology to the application, preventing processor count bugs in application code such as signed, narrowing affinity or insufficient buffer.
  - The updated shim in Windows® 10 version 19H1 hooks processor related APIs such as GetSystemInfo and GetLogicalProcessorInformation.
    - See the Compatibility Administrator.
    - Contact your Microsoft representative or report a problem using the Microsoft Feedback Hub.
BUILD COMMANDS LISTS IN PARALLEL
BUILD COMMAND LISTS IN PARALLEL

• Binary compiled using NumContexts WorkerThreads shows higher performance given sufficient Draw count
  • See https://github.com/Microsoft/DirectX-Graphics-Samples/tree/master/Samples/Desktop/D3D12Multithreading

• Recommend NumContexts = min(cores-1, Draws/250)

• Performance of binary compiled with Microsoft® Visual Studio 2017 v15.9.6
  • Testing done by Kenneth Mitchell February 12, 2019 on the following system. PC manufacturers may vary configurations yielding different results. Results may vary based on driver versions used. Test configuration: AMD Ryzen™ 7 2700X Processor, 2x8GB DDR4-3200 (16-18-18-36), Radeon™ RX Vega 64 (driver 19.2.1 Feb4), AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1809, 1920x1080 resolution. BIOS core clock = 3.7GHz, Core Performance Boost = Disable, Global C-state Control = Disable
USE UE4 PARALLEL RENDERING

- Be sure to re-enable parallel rendering after using debug features before you ship!
- See [https://docs.unrealengine.com/en-US/Programming/Rendering/ParallelRendering](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.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>
<tr>
<td>r.rhicmdbypass</td>
<td>0</td>
</tr>
</tbody>
</table>
USE BEST PRACTICES WITH SPINLOCKS
SUMMARY

- Spin Lock Best Practices:
  - avoid lock prefix instructions
  - use the pause instruction
  - test and test-and-set
  - alignas(64) lock variable
    - or _declspec(align(64))

- Profiling:
  - AMD uProf v2.0 "Assess Performance (Extended)" Event Based Sampling Profile
  - ALUTokenStall PTI
  - >= 3K Per Thousand Instructions is bad for top functions

namespace MyLock {
    typedef unsigned LOCK, *PLOCK;
    enum { LOCK_IS_FREE = 0, LOCK_IS_TAKEN = 1 };
    #if 0
        /* BAD */
        void Lock(PLOCK pl) {
            while (LOCK_IS_TAKEN == _InterlockedCompareExchange( \n                pl, LOCK_IS_TAKEN, LOCK_IS_FREE)) { // lock, xchg, cmp
            }
        }
    #else
        /* GOOD */
        void Lock(PLOCK pl) {
            while ((LOCK_IS_TAKEN == *pl) || \n                (LOCK_IS_TAKEN==_InterlockedExchange(pl, LOCK_IS_TAKEN))){
                _mm_pause();
            }
        }
    
    #endif
    void Unlock(PLOCK pl) {
        _InterlockedExchange(pl, LOCK_IS_FREE);
    }
} alignas(64) MyLock::LOCK gLock;
PERFORMANCE

- Binary compiled after applying best practices shows higher performance
- Performance of binary compiled with Microsoft Visual Studio 2017 v15.9.6
  - Testing done by Kenneth Mitchell January 30, 2019 on the following system. PC manufacturers may vary configurations yielding different results. Results may vary based on driver versions used. Test configuration: AMD Ryzen™ 7 2700X Processor, 2x8GB DDR4-3200 (16-18-18-36), Radeon™ RX 580, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1809, 1920x1080 resolution. BIOS core clock = 3.7GHz, Core Performance Boost = Disable, Global C-state Control = Disable

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>304</td>
<td>27</td>
</tr>
</tbody>
</table>

Use Best Practices With Spinlocks (less is better)
CODE SAMPLE

```c
#include "intrin.h"
#include "stdio.h"
#include "windows.h"
#include <chrono>
#include <numeric>
#include <thread>

#define LEN 512

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

DWORD WINAPI ThreadProcCallback(LPVOID data) {
    MyLock::Lock(&gLock);
    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++) {
        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);
    }
    printf("result: %f\n", r);
    MyLock::Unlock(&gLock);
    return 0;
}

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];
    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): %f\n", 1000.0 * time_span.count());
    delete[] threads;
    return EXIT_SUCCESS;
}
```
PROFILING BEFORE

~4K stalls per 1000 instructions
PROFILING AFTER

~0 stalls per 1000 instructions
<table>
<thead>
<tr>
<th>Line</th>
<th>Offset</th>
<th>Source Code</th>
<th>CPU clocks</th>
<th>Ret inst</th>
<th>IPC</th>
<th>ALUStall%</th>
<th>CacheMiss%</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td></td>
<td>DWORD WINAPI ThreadProcCallback(LPDVOID data) {</td>
<td>3923106</td>
<td>8758</td>
<td>0.07</td>
<td>21036.12</td>
<td>105.58</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mylock::lock(&amp;lock);</td>
<td>81</td>
<td>1</td>
<td>0.03</td>
<td>26600.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>XOR eax, eax;</td>
<td>32</td>
<td>1</td>
<td>0.02</td>
<td>28100.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RAX = 0.0F;</td>
<td>100</td>
<td>1</td>
<td>0.02</td>
<td>28100.00</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>std::fill((float*)(&amp;a[0]), (float*)(&amp;a[0] + 8), 0.0F);</td>
<td>100</td>
<td>1</td>
<td>0.02</td>
<td>28100.00</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
<td>float c = 0.0;</td>
<td>100</td>
<td>1</td>
<td>0.02</td>
<td>28100.00</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
<td>for (size_t iter = 0; iter &lt; 1000000; iter++) {</td>
<td>41</td>
<td>16</td>
<td>0.39</td>
<td>905.25</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
<td>for (int m = 0; m &lt; LNUM; m++) {</td>
<td>145</td>
<td>20</td>
<td>0.14</td>
<td>451.30</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
<td>for (int i = 0; i &lt; 4; i++) {</td>
<td>25692</td>
<td>46697</td>
<td>1.73</td>
<td>106.72</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td>for (int j = 0; j &lt; 4; j++) {</td>
<td>21043</td>
<td>25620</td>
<td>1.22</td>
<td>105.18</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
<td>for (int k = 0; k &lt; 4; k++) {</td>
<td>122220</td>
<td>71610</td>
<td>0.59</td>
<td>478.76</td>
<td>0.01</td>
</tr>
<tr>
<td>47</td>
<td></td>
<td>r += std::accumulate((float*)(&amp;a[0]), (float*)(&amp;a[0] + 8), 0.0F);</td>
<td>100</td>
<td>1</td>
<td>0.02</td>
<td>28100.00</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td>}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
<td>}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>int main(int argc, char *argv[]) {</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>using namespace std::chrono;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DISASSEMBLY SNIPPET

; BEFORE (BAD)
00000001400010C1:
xor eax,eax
lock cmpxchg dword ptr [gLock@@3IA],ecx
cmp eax,ecx
je 00000001400010C1

; AFTER (GOOD)
00000001400010C0:
cmp dword ptr [gLock@@3IA],1
je 00000001400010D9
mov eax,1
xchg eax,dword ptr [gLock@@3IA]
cmp eax,1
jne 00000001400010DD
00000001400010D9:
pause
jmp 00000001400010C0
00000001400010DD:
# INTERLOCKED INTRINSIC FUNCTIONS

<table>
<thead>
<tr>
<th>intrinsic</th>
<th>generates instructions including</th>
</tr>
</thead>
<tbody>
<tr>
<td>_InterlockedAnd</td>
<td>lock cmpxchgl</td>
</tr>
<tr>
<td>interlockedbittestandreset</td>
<td>lock btr</td>
</tr>
<tr>
<td>_InterlockedBitTestAndSet</td>
<td>lock bts</td>
</tr>
<tr>
<td>_InterlockedCompareExchange</td>
<td>lock cmpxchg</td>
</tr>
<tr>
<td>_InterlockedCompareExchange128</td>
<td>lock cmpxchg16b</td>
</tr>
<tr>
<td>_InterlockedCompareExchangePointer</td>
<td>lock cmpxchg</td>
</tr>
<tr>
<td>_InterlockedDecrement</td>
<td>lock dec</td>
</tr>
<tr>
<td>_InterlockedExchangeAdd</td>
<td>lock add</td>
</tr>
<tr>
<td>_InterlockedIncrement</td>
<td>lock inc</td>
</tr>
<tr>
<td>_InterlockedOr</td>
<td>lock cmpxchg</td>
</tr>
<tr>
<td>_InterlockedXor</td>
<td>lock cmpxchg</td>
</tr>
<tr>
<td>_InterlockedExchange</td>
<td>xchg</td>
</tr>
<tr>
<td>_InterlockedExchangePointer</td>
<td>xchg</td>
</tr>
</tbody>
</table>

Instructions generated may vary depending on compiler and optimization flags.
AVOID MEMCPY & MEMSET REGRESSION
Currently, “C:\Windows\System32\vcruntime140.dll” includes a memcpy & memset regression that can affect AMD family 17h and later processors only.

Code generated by MSVS 2015, 2017, or 2019 may link to vcruntime140.dll.

However, we have found that this regression occurs under very specific circumstances where length > 32 && length <= 128 Bytes && length is unknown at compile time.

Meanwhile, we propose a workaround.
WORKAROUND

Workaround:

• Modify Assembly Files (Copying Recommended)
  • Comment out lines shown in yellow
  • C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Tools\MSVC\14.16.27023\crt\src\x64\n    • memcpy.asm
    • memset.asm

• Assemble Object Files
  • ml64.exe -c memcpy.asm memset.asm

• Set Linker > Input > Additional Dependencies = memcpy.obj;memset.obj;…

Alternate Workaround:

• Copy vcruntime140.dll from “Microsoft Visual Studio\2019\Preview\Common7\IDE\VC\vcpackages\” to the application folder

* memcpy.asm & memset.asm code is Copyright (c) Microsoft Corporation. All rights reserved

```asm
; memcpy.asm
; line 456
XmmCopySmall:
bt __favor, __FAVOR_SMSTRG ; check if string copy should be used.
jc memcpy_repmovs
movups xmm0, [rcx + rdx] ; load deferred bytes
add rcx, 16
sub r8, 16

; memset.asm
; line 93
; Check if strings should be used
;
cmp r8, 128 ; is this a small set, size <= 128?
ja XmmSet ; if large set, use XMM set
bt __favor, __FAVOR_SMSTRG ; check if string set should be used
jnc XmmSetSmall ; otherwise, use a 16-byte block set
jmp memset_repmovs
```
PERFORMANCE

- Binary compiled after applying workaround shows higher performance
- Performance of binary compiled with Microsoft Visual Studio 2017 v15.9.6
  - Testing done by Kenneth Mitchell February 11, 2019 on the following system. PC manufacturers may vary configurations yielding different results. Results may vary based on driver versions used. Test configuration: AMD Ryzen™ 7 2700X Processor, 2x8GB DDR4-3200 (16-18-18-36), Radeon™ RX Vega 64, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1809, 1920x1080 resolution. BIOS core clock = 3.7GHz, Core Performance Boost = Disable, Global C-state Control = Disable
#include <numeric>
#include <chrono>
alignas(64) char a[32 * 1024];
alignas(64) char b[32 * 1024];
using namespace std::chrono;

void work(int size, size_t steps) {
    high_resolution_clock::time_point t0 = \
        high_resolution_clock::now();
    for (size_t i = 0; i < steps; i++) {
        memcpy(b, a, size);
    }
    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());
}

int main(int argc, char *argv[]) {
    int j = (argc > 1) ? atoi(argv[1]) : 0;
    int seed = (argc > 2) ? atoi(argv[2]) : 7;
    size_t steps = (argc > 3) ? atoll(argv[3]) : 1000000000;
    int size = (argc > 4) ? atoi(argv[4]) : 48;
    srand(seed);
    for (int i = 0; i <= sizeof(a); ++i) {
        a[i] = rand()%256;
    }
    memset(b, 0, sizeof(b));
    work(size, steps);
    printf("a[%i] = %i\n", j, a[j]);
    printf("b[%i] = %i\n", j, b[j]);
    return EXIT_SUCCESS;
}
DISASSEMBLY SNIPPET

; BEFORE (WITHOUT WORKAROUND)
; calls vcruntime140.dll!memcpy
memcpy:
jmp qword ptr [__imp_memcpy]

; AFTER (WITH WORKAROUND)
; calls memcpy within executable with workaround
memcpy:
mov r11,rcx
mov r10,rdx
cmp r8,10h
jbe 00000001400012F0
cmp r8,20h
jbe 00000001400012D0
sub rdx,rcx
jae 00000001400012A4
lea rax,[r8+r10]
cmp rcx,rax
jb 00000001400015D0
cmp r8,80h
jbe 0000000140001510
...
AVOID TOO MANY NON-TEMPORAL STREAMS
SUMMARY

• Write Combining lines are not stored in the caches. The Write Combining Buffer writes 64 byte lines to memory when full. The processor can gather writes from 8 different 64B cache lines (up to 7 from one hardware thread).

• Avoid interleaving multiple Write Combining streams to different addresses; use only one stream per hardware thread if possible. While using multiple streams, the hardware may close buffers before they are completely full, thus leading to reduced performance.

• Profiling:
  • AMD uProf v2.0 "Assess Performance (Extended)" Event Based Sampling Profile
  • WcbFull PTI
    • \( \geq 22 \) Per Thousand Instructions is bad for top functions
    • \( \geq 35 \) Per Thousand Instructions is very bad for top functions
PERFORMANCE

- Binary compiled after applying the workaround shows higher performance.
  - Testing done by Kenneth Mitchell January 31, 2019 on the following system. PC manufacturers may vary configurations yielding different results. Results may vary based on driver versions used. Test configuration: AMD Ryzen™ 7 2700X Processor, 2x8GB DDR4-3200 (16-18-18-36), Radeon™ RX 580, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1809, 1920x1080 resolution. BIOS core clock = 3.7GHz, Core Performance Boost = Disable, Global C-state Control = Disable.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>104</td>
</tr>
</tbody>
</table>

Avoid Too Many Non-Temporal Streams (less is better)
```c
#include <intrin.h>
#include <numeric>
#include <chrono>
#define LEN 64000
alignas(64) float a[LEN];
alignas(64) float b[LEN];

void step(float dt) {
    for (size_t i = 0; i < LEN; i += 8) {
        // x,y,z,w,vx,vy,vw
        __m128 p1 = _mm_load_ps(&a[(i + 0) % LEN]);
        __m128 v1 = _mm_load_ps(&a[(i + 4) % LEN]);
        p1 = _mm_add_ps(p1, _mm_mul_ps(v1, _mm_load_ps1(&dt)));
        _mm_stream_ps(&a[(i + 0) % LEN], p1);
        __m128 p2 = _mm_load_ps(&b[(i + 0) % LEN]);
        __m128 v2 = _mm_load_ps(&b[(i + 4) % LEN]);
        p2 = _mm_add_ps(p2, _mm_mul_ps(v2, _mm_load_ps1(&dt)));
        #if 0
        /* without workaround */
        _mm_store_ps(&b[(i + 0) % LEN], p2);
        #else
        /* with workaround */
        _mm_store_ps(&b[(i + 0) % LEN], p2);
        #endif
    }
}

int main(int argc, char *argv[]) {
    using namespace std::chrono;
    int j = (argc > 1) ? atoi(argv[1]) : 0;
    int seed = (argc > 2) ? atoi(argv[2]) : 3;
    size_t steps = (argc > 3) ? atoll(argv[3]) : 2000000;
    float dt = (argc > 4) ? (float)atof(argv[4]) : 0.001f;
    srand(seed);
    for (int i = 0; i < LEN; ++i) {
        a[i] = (float)rand() / RAND_MAX;
        b[i] = (float)rand() / RAND_MAX;
    }
    high_resolution_clock::time_point t0 = high_resolution_clock::now();
    for (size_t i = 0; i < steps; i++) {
        step(dt);
    }
    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());
    printf("a[%i] = %f\n", j, a[j]);
    printf("b[%i] = %f\n", j, b[j]);
    return EXIT_SUCCESS;
}
```
### PROFILING BEFORE

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU clocks</th>
<th>Ret inst</th>
<th>IPC</th>
<th>WebFull PTI</th>
<th>Retired Branch</th>
<th>%Retired Branch</th>
<th>Data Cache Acc</th>
<th>%Demand Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>wbcp-full-before.exe (PID 04548)</td>
<td>529207</td>
<td>313</td>
<td>1.06</td>
<td>22.84</td>
<td>33.49</td>
<td>0.07</td>
<td>88.30</td>
<td>26.52</td>
</tr>
<tr>
<td>conhost.exe (PID 7570)</td>
<td>447</td>
<td>44</td>
<td>0.72</td>
<td>2.62</td>
<td>348.97</td>
<td>0.07</td>
<td>459.67</td>
<td>2.12</td>
</tr>
</tbody>
</table>

~23 WebFull per 1000 instructions
PROFILING AFTER

~3 WcbFull per 1000 instructions
DISASSEMBLY SNIPPET

; BEFORE (WITHOUT WORKAROUND)
mulps  xmm0, xmm1
addps  xmm0, xmmword ptr [rsi+r8*4+42E40h]
movntps xmmword ptr [rsi+r8*4+42E40h], xmm0
movups xmm0, xmmword ptr [rsi+rcx*4+4640h]
muls  xmm0, xmm1
addps  xmm0, xmmword ptr [rsi+r8*4+4640h]
movntps xmmword ptr [rsi+r8*4+4640h], xmm0

; AFTER (WITH WORKAROUND)
mulps  xmm0, xmm1
addps  xmm0, xmmword ptr [rsi+r8*4+42E40h]
movntps xmmword ptr [rsi+r8*4+42E40h], xmm0
movups xmm0, xmmword ptr [rsi+rcx*4+4640h]
muls  xmm0, xmm1
addps  xmm0, xmmword ptr [rsi+r8*4+4640h]
movntps xmmword ptr [rsi+r8*4+4640h], xmm0
AVOID FALSE SHARING
SUMMARY

- False Sharing occurs when threads running on different processors, each with a local cache, modify variables that exist in the same cache line. This can reduce performance due to processor work required to maintain cache-coherency.
  - L3 is filled from L2 victims of all 4 cores within a CCX.
    - L2 tags are duplicated in the L3 for fast cache transfers within a CCX.
      - And fast probe filtering for Ryzen™ Threadripper™ and Epyc™.
- Use thread local rather than global or process shared data.
- Align and pad Thread parameters – especially synchronizations variables!
- Profiling:
  - AMD uProf v2.0 "Assess Performance (Extended)" Event Based Sampling Profile
    - Data Cache Refills CCX PTI
      - Minimize
PERFORMANCE

- Binary compiled after applying best practices shows higher performance
- Performance of binary compiled with Microsoft Visual Studio 2017 v15.9.3
  - Testing done by Kenneth Mitchell February 5, 2019 on the following system. PC manufacturers may vary configurations yielding different results. Results may vary based on driver versions used. Test configuration: AMD Ryzen™ 7 2700X Processor, 2x8GB DDR4-3200 (16-18-18-36), Radeon™ RX Vega 64, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1809, 1920x1080 resolution. BIOS core clock = 3.7GHz, Core Performance Boost = Disable, Global C-state Control = Disable

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>+679%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>365</td>
</tr>
</tbody>
</table>
# GDC19

## AMD Ryzen™ Processor Software Optimization

### March 20th, 2019

---

### CODE SAMPLE

```
#include <windows.h>
#include <chrono>
#include <numeric>
#include <thread>

using namespace std::chrono;

#define NUM_ITER 2000000000

int seed;

#if 0
/* 4 bytes */
struct ThreadData { unsigned long sum; };
#else
/* 64 bytes */
struct alignas(64) ThreadData { unsigned long sum; };
#endif

DWORD WINAPI ThreadProcCallback(void* param) {
  ThreadData *p = (ThreadData*)param;
  srand(seed);
  p->sum = 0;
  for (int i = 0; i < NUM_ITER; i++) {
    p->sum += rand() % 2;
  }
  return 0;
}

int main(int argc, char *argv[]) {
  seed = (argc > 1) ? atoi(argv[1]) : 3;
  int numThreads = std::thread::hardware_concurrency();
  HANDLE* threads = new HANDLE[numThreads];
  ThreadData* a = new ThreadData[numThreads];
  high_resolution_clock::time_point t0 = \
    high_resolution_clock::now();
  for (size_t i = 0; i < numThreads; ++i) {
    threads[i] = CreateThread(NULL, 0, ThreadProcCallback, \
      (void*) &a[i], 0, NULL);
  }
  WaitForMultipleObjects(numThreads, 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 (ms): %lf\n", 1000.0 * time_span.count());
  for (size_t i = 0; i < numThreads; ++i) {
    printf("sum[%llu] = %lu\n", i, a[i].sum);
  }
  delete[] a;
  delete[] threads;
  return EXIT_SUCCESS;
}
```
### PROFILING BEFORE

Many refills from CCX

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU clocks</th>
<th>Ret inst</th>
<th>IPC</th>
<th>Data Cache Clocks</th>
<th>%Demand</th>
<th>%Data Cache Miss</th>
<th>Data Cache Refills</th>
<th>Data Cache Refills DRAM PTI</th>
<th>Data Cache Refills CCX PTI</th>
<th>Data Cache Refills L2 PTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>rand</td>
<td>6104510</td>
<td>6733082</td>
<td>0.31</td>
<td>258.83</td>
<td>1.62</td>
<td>1.62</td>
<td>3641</td>
<td>265555</td>
<td>3450</td>
<td></td>
</tr>
<tr>
<td>PthreadCreateThread(oid)</td>
<td>20198</td>
<td>0.99</td>
<td>386.15</td>
<td>2.56</td>
<td>1.21</td>
<td>2.03</td>
<td>560.56</td>
<td>90.47</td>
<td>162.38</td>
<td>15.23</td>
</tr>
<tr>
<td>pthread_mutex_lock</td>
<td>141354</td>
<td>3031</td>
<td>0.35</td>
<td>315.84</td>
<td>3.01</td>
<td>4.03</td>
<td>733.96</td>
<td>92.5</td>
<td>153.2</td>
<td>31.4</td>
</tr>
<tr>
<td>pthread_rwlock_rdlock</td>
<td>6956</td>
<td>1342</td>
<td>0.20</td>
<td>402.35</td>
<td>1.86</td>
<td>5.35</td>
<td>62.05</td>
<td>95.3</td>
<td>183.1</td>
<td>23.7</td>
</tr>
<tr>
<td>pthread_rwlock_wrlock</td>
<td>3551</td>
<td>28.8</td>
<td>0.04</td>
<td>753.39</td>
<td>13.90</td>
<td>13.00</td>
<td>84.57</td>
<td>16</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>NvmmnsGetSmmDispatchNoLock</td>
<td>3694</td>
<td>99</td>
<td>0.02</td>
<td>1672.89</td>
<td>21.98</td>
<td>25.54</td>
<td>18</td>
<td>70</td>
<td>18</td>
<td>70</td>
</tr>
<tr>
<td>NvmmnsGetSmmDispatchNoLock</td>
<td>1793</td>
<td>777</td>
<td>0.44</td>
<td>295.44</td>
<td>9.47</td>
<td>16.32</td>
<td>107</td>
<td>21</td>
<td>70</td>
<td>18</td>
</tr>
</tbody>
</table>
### PROFILING AFTER

![Graph showing performance metrics](image)

Few refills from CCX

<table>
<thead>
<tr>
<th>CPU clocks</th>
<th>Ret inst</th>
<th>IPC</th>
<th>Data Cache Accesses</th>
<th>%Demand Data</th>
<th>%Data Cache Miss</th>
<th>Data Cache Refills</th>
<th>Data Cache Refills CCX PTI</th>
<th>Data Cache Refills L2 PTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>199089</td>
<td>102324</td>
<td>2.03</td>
<td>649152</td>
<td>92.4%</td>
<td>6.5%</td>
<td>2.9%</td>
<td>4.5%</td>
<td>0.1%</td>
</tr>
</tbody>
</table>

- AMD Ryzen™ Processor Software Optimization
- March 20th, 2019
SOURCE BEFORE
### AMD Ryzen™ Processor Software Optimization

**March 20th, 2019**

![Image of AMD Ryzen™ Processor Software Optimization](image_url)

### Filters
- [false sharing off cp]

<table>
<thead>
<tr>
<th>Line</th>
<th>Offset</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td>struct ThreadData { unsigned long sum; };</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>struct alignas(64) ThreadData { unsigned long sum; };</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>DWORD WINAPI ThreadProcCallback(void* param) {</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>ThreadData <em>p = (ThreadData</em>)param;</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>srand(2);</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>p-&gt;sum = 8;</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>for (int i = 0; i &lt; NUM_THREADS; i++) {</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>p-&gt;sum += rand() % 2;</td>
</tr>
</tbody>
</table>
| 20   |        | }

---

**Function List:**
- [0x1076 - 0x10bd] ThreadProcCallback(void *)

**Data Cache Access**
- CPU Clocks: 234.54, 22.24, 285.25, 0.02, 0.02
- Ret Inst: 22.24, 285.56, 0.02, 0.02, 1, 100
- IPC: 285.25, 0.02, 0.02
- Data Cache Access: 9, 100

**Source Code**
- `call word [5*0x00021649+0x003200]`
- `and eax,0x00000019h`
- `shr ax,16`
- `dec ax`
- `add [rbx],eax`

**Context**
- AMD Ryzen™ Processor Software Optimization

---

**References:**
- [AMD Ryzen™ Processor Software Optimization](image_url)
- [AMD Ryzen™ Processor Software Optimization](image_url)

---

**Activate Windows**
- Go to Settings to activate Windows.
DISASSEMBLY SNIPPET

; BEFORE (BAD)

0000000140001180:

    mov    qword ptr [rsp+28h],rsi
    lea    r8,[?ThreadProcCallback@@YAKPEAX@Z]
    mov    r9,rbp
    mov    dword ptr [rsp+20h],esi
    xor    edx,edx
    xor    ecx,ecx
    call   qword ptr [__imp_CreateThread]
    mov    qword ptr [r12+rdi*8],rax

    add    rbp,4
    inc    rdi
    cmp    rdi,r14
    jb     0000000140001180

; AFTER (GOOD)

0000000140001180:

    mov    qword ptr [rsp+28h],rsi
    lea    r8,[?ThreadProcCallback@@YAKPEAX@Z]
    mov    r9,rbp
    mov    dword ptr [rsp+20h],esi
    xor    edx,edx
    xor    ecx,ecx
    call   qword ptr [__imp_CreateThread]
    mov    qword ptr [r15+rdi*8],rax

    add    rbp,40h
    inc    rdi
    cmp    rdi,r14
    jb     0000000140001180
ROADMAP
2019 AMD CLIENT LINEUP

CONTINUOUS INNOVATION LEADERSHIP

Roadmap subject to change
3rd Generation AMD Ryzen™ Desktop Processor

- New Zen2 core microarchitecture
- New 7nm process technology
- Higher instructions-per-cycle (IPC)
- Doubled floating point width to 256-bit
- Doubled load/store bandwidth
- PCIe® Gen4 Ready
- AM4 Desktop Infrastructure
- See https://www.amd.com/en/events/ces
- See https://www.amd.com/en/events/next-horizon
QUESTIONS
THANK YOU
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

© 2019 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.