AMD RYZEN™ CPU OPTIMIZATION

Presented by Ken Mitchell & Elliot Kim
Join AMD ISV Game Engineering team members for an introduction to the AMD Ryzen™ family of CPU and APU processors followed by advanced optimization topics. Learn about the “Zen” family SoCs, planned next-generation Ryzen processors, 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 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. Previously, he was tasked with automating & analyzing PC applications for performance projections of future AMD products. He studied computer science at the University of Texas at Austin.

- Elliot Kim 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. Previously, he worked as a game developer at Interactive Magic and has since gained extensive experience in 3D technology and simulations programming. He holds a BS in Electrical Engineering from Northeastern University in Boston.
AGENDA

- Success Stories
- “Zen” Family Processors
- AMD μProf Profiler
- Optimizations & Lessons Learned
- Roadmap
- QA
Success Stories
SUCCESS STORIES

1080P GAMING PERFORMANCE IMPROVEMENTS

**Ashes of the Singularity**
1080p, High, GPU Test

Average FPS

<table>
<thead>
<tr>
<th>Version</th>
<th>Previous</th>
<th>2017-03-27</th>
</tr>
</thead>
<tbody>
<tr>
<td>v2.10.25624</td>
<td>60</td>
<td>80</td>
</tr>
<tr>
<td>v2.11.26118</td>
<td></td>
<td>+31%</td>
</tr>
</tbody>
</table>

**Total War: Warhammer**
1080p, High

Average FPS

<table>
<thead>
<tr>
<th>Version</th>
<th>Previous</th>
<th>2017-03-27</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1.0.767.2</td>
<td>120</td>
<td>140</td>
</tr>
<tr>
<td>v1.0.770.1</td>
<td></td>
<td>+26%</td>
</tr>
</tbody>
</table>

**Rise of the Tomb Raider**
1080p, High

Average FPS

<table>
<thead>
<tr>
<th>Version</th>
<th>Previous</th>
<th>2017-03-27</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1.0.767.2</td>
<td>120</td>
<td>140</td>
</tr>
<tr>
<td>v1.0.770.1</td>
<td></td>
<td>+26%</td>
</tr>
</tbody>
</table>
SUCCESS STORIES
CREATIVITY IMPROVEMENTS

This routine operation has shrunk from an agonizing 22.5 seconds to a blistering 11 milliseconds.
SUCCESS STORIES

DISCLAIMER

Ashes of the Singularity
- [https://community.amd.com/community/gaming/blog/2017/03/30/amd-ryzen-community-update-2](https://community.amd.com/community/gaming/blog/2017/03/30/amd-ryzen-community-update-2)
- AMD Ryzen™ 7 1800X Processor, 2x8GB DDR4-2933 (15-17-17-35), GeForce GTX 1080 (378.92 driver), Gigabyte GA-AX370-Gaming5, Windows® 10 x64 build 1607, 1920x1080 resolution, high in-game quality preset.

Total War: Warhammer
- Testing conducted as of April 4, 2017. System configuration: AMD Ryzen™ 7 1800X, Gigabyte GA-AX370-Gaming5, 2x8GB DDR4-2933, GeForce GTX 1080 (378.92 driver), Windows 10 x64 (build 1607), 1920x1080 resolution.

Rise of the Tomb Raider
- Testing conducted as of 6/6/2017. System configuration: AMD Ryzen™ 7 1800X Processor, 2x8GB DDR4-3200 (14-14-14-36), GeForce GTX 1080 (382.33 driver), Asus Crosshair VI (BIOS 9943), Windows® 10 x64 build 1607, 1920x1080 resolution.

ZBrush
- Testing conducted as of 6/6/2017. System configuration: AMD Ryzen™ 7 1800X Processor, 2x8GB DDR4-2400 (17-17-17-39), GeForce GTX 1080 (382.05 driver), AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1607, 1920x1080 resolution.
“Zen” Family Processors
"SUMMIT RIDGE"

**DATA FLOW**

- **cclk 4.0 GHz / 3.6 GHz**
- **memclk 1.3 GHz (DDR4-2667)**
- **lclk 615 MHz**

- **64K I-Cache 4-way**
- **512K L2**
- **8M L3**
- **16B/cycle**
- **32B/cycle**
- **2*16B load**
- **1*16B store**
- **32B fetch**
- **32B/cycle**

**Unified Memory Controller**

**DRAM Channel**

**IO Hub Controller**

**Data Fabric**

**32B/cycle**

**16B/cycle**
THREADRIPPER(TM) PROCESSOR

DATA FLOW

- cclk 4.0 GHz / 3.4 GHz
- memclk 1.3 GHz (DDR4-2667)
- lclk 727 MHz

![Diagram of the Threadripper(TM) Processor with data flow and specifications.](image-url)
cclk 3.9 GHz / 3.6 GHz
memclk 1.3 GHz (DDR4-2667)
lclk 496
sclk 1.25 GHz (11CU = 704 shaders)

1 CCX with GFX9 “Vega” & MultiMedia Hub
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.

- System configs: AMD reference motherboard(s), AMD Radeon™ R9 290X GPU, 8GB DDR4-2667 (“Zen”)/8GB DDR3-2133 (“Excavator”)/8GB DDR3-1866 (“Piledriver”), Ubuntu Linux 16.x (SPECint06) and Windows® 10 x64 RS1 (Cinebench R15).

- Results may vary.
**MICROARCHITECTURE**

**INSTRUCTION SET EVOLUTION**

<table>
<thead>
<tr>
<th>YEAR</th>
<th>FAMILY</th>
<th>PRODUCT FAMILY</th>
<th>ARCHITECTURE</th>
<th>EXAMPLE MODEL</th>
<th>ADX</th>
<th>CLFLUSHOPT</th>
<th>RDSEED</th>
<th>SHA</th>
<th>SMAP</th>
<th>XGETBV</th>
<th>XSAVE</th>
<th>XSAVES</th>
<th>AVX</th>
<th>AVX2</th>
<th>BMIM</th>
<th>RDRND</th>
<th>SMEP</th>
<th>FSGSBASE</th>
<th>XSAVEOPT</th>
<th>BI</th>
<th>FMA</th>
<th>F16C</th>
<th>AES</th>
<th>AVX</th>
<th>OSXSAVE</th>
<th>PCLMULQDQ</th>
<th>SSE4.1</th>
<th>SSE4.2</th>
<th>XSAVE</th>
<th>SSSE3</th>
<th>CLZERO</th>
<th>FMA4</th>
<th>TBM</th>
<th>XOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017</td>
<td>17h</td>
<td>“Summit Ridge”</td>
<td>“Zen”</td>
<td>Ryzen 7 1800X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2015</td>
<td>15h</td>
<td>“Carrizo”/“Bristol Ridge”</td>
<td>“Excavator”</td>
<td>A12-9800</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td>15h</td>
<td>“Kaveri”/“Godavari”</td>
<td>“Steamroller”</td>
<td>A10-7890K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>15h</td>
<td>“Vishera”</td>
<td>“Piledriver”</td>
<td>FX-8370</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>15h</td>
<td>“Zambezi”</td>
<td>“Bulldozer”</td>
<td>FX-8150</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>16h</td>
<td>“Kabini”</td>
<td>“Jaguar”</td>
<td>A6-1450</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>14h</td>
<td>“Ontario”</td>
<td>“Bobcat”</td>
<td>E-450</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>12h</td>
<td>“Llano”</td>
<td>“Husky”</td>
<td>A8-3870</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td>10h</td>
<td>“Greyhound”</td>
<td>“Greyhound”</td>
<td>Phenom II X4 955</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

+ ADX multi precision support
+ CLFLUSHOPT Flush Cache Line Optimized SFENCE order
+ RDSEED Pseudorandom number generation Seed
+ SHA Secure Hash Algorithm (SHA-1, SHA-256)
+ SMAP Supervisor Mode Access Prevention
+ XGETBV Get extended control register
+ XSAVEC, XSAVES Compact and Supervisor Save/Restore

+ CLZero Zero Cache Line

- FMA4
- TBM
- XOP
AMD μProf Profiler
AMD µProf Profiler

- v1.1 Changes
  - New GUI for Performance and Power Profiling analysis.
    - Filter available events
    - Group by Process, Module, or Thread
  - Unified CLI for Performance and Power Profiling
  - Energy consumption analysis of Process, Thread, Module, Function and Source line.
  - L3 and DF PMC counter support (Windows CLI only)
  - Recommend using 250,000 CPU Clock Cycle Interval while adding Custom Counters to Profile
    - Makes math easy for Per Thousand Instructions (PTI) & Per Thousand Clocks (PTC)
  - 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
PERFORMANCE MONITORING 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
Disabling power management features may reduce variation during AB testing.

- **BIOS Settings**
  - “Zen” Common Options
    - Custom Core Pstates (or use Ryzen Master)
      - Disable all except Pstate0
    - 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
  - Core Performance Boost = Disable
  - Global C-state Control = Disable

- High Performance power scheme
Optimizations & Lessons Learned
Use Microsoft® Visual Studio 2017
Use best practices when counting cores
Avoid too many non-temporal streams
Use best practices with spinlocks
Avoid false sharing
Use Microsoft® Visual Studio 2017
USE MICROSOFT® VISUAL STUDIO 2017
FOR OPTIMAL CODE GENERATION

Brief:
- A ST/ST/LD stall code gen issue was fixed in MSVS 2017

Example:
- Found in some C++ with SSE2 intrinsics.
- Allegedly found in some C++ with DirectXMath XMVectorSwizzle.

Profiling:
- Custom CPU Profile > Events by Hardware Source >
  - [024] Bad Status 2 > [1] StliOther
    - > 5 per thousand instructions is bad
  - [035] Store to Load Forward
  - [076] Cycles not in Halt
  - [0C0] Instructions Retired
<table>
<thead>
<tr>
<th></th>
<th>MSVS 2015</th>
<th></th>
<th>MSVS 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>movss DWORD PTR T[0], xmm1</td>
<td>1.</td>
<td>movss DWORD PTR T[0], xmm1</td>
</tr>
<tr>
<td>2.</td>
<td>movss DWORD PTR T[4], xmm2</td>
<td>2.</td>
<td>movups xmm0, DWORD PTR T[0]</td>
</tr>
<tr>
<td>3.</td>
<td>movss DWORD PTR T[8], xmm3</td>
<td>3.</td>
<td>movss DWORD PTR T[4], xmm2</td>
</tr>
<tr>
<td>4.</td>
<td>movss DWORD PTR T[12], xmm4</td>
<td>4.</td>
<td>movss DWORD PTR T[8], xmm3</td>
</tr>
<tr>
<td>5.</td>
<td>movups xmm0, DWORD PTR T[0]</td>
<td>5.</td>
<td>movss DWORD PTR T[12], xmm4</td>
</tr>
</tbody>
</table>
Use best practices when counting cores
**USE ALL PHYSICAL CORES**

**Brief:**

- **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.
  - Always 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
// This advice is specific to AMD processors and is not general guidance for all processor vendors
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;
        }
    } else {
        count = cores;
    }
    return count;
}
```
Brief:

- Get `GetPSYSTEM_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

```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) {
        buffer = (char*)malloc(len);
        if (GetLogicalProcessorInformationEx(
            RelationAll,
            (PSYSTEM_LOGICAL_PROCESSOR_INFORMATION_EX)buffer,
            &len)) {
            // ...
        }
    }
    free(buffer);
}
```
Brief:

- 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.

Example:

- Bad
  - popcnt = 0
  - mask b[32] = 0x0000000000000000

- Good
  - popcnt = 64
  - mask b[32] = 0x00000000010000000

```c
DWORD lps = 64; // logical processors
DWORD_PTR p = 0xffffffffffffffff; // process mask
int b = 32; // bit index

#if 0
/* BAD */
int x = p; // signed, narrowing!
int count = 0;
while (x != 0) { // never zero!
    while (x > 0) { // false!
        count += (x & 1);
        x >>= 1; // fill bits with sign!
    }
    printf("popcnt = %i\n", count); // 0
    printf("mask b[%i] = 0x%p\n", b, (1 << b)); // undefined, 0?
}
#else
/* GOOD */
DWORD_PTR x = p;
int count = 0;
while (x != 0) {
    count += (x & 1);
    x >>= 1;
}
printf("popcnt = %i\n", count);
printf("mask b[%i] = 0x%p\n", b, (1ULL << b));
#endif
```
Avoid Too Many Non-Temporal Streams
Brief:
- Avoid interleaving multiple streams to different addresses; use only one stream if possible.
- While using multiple streams, the hardware may close buffers before they are completely full, thus leading to reduced performance.

Profiling:
- Custom CPU Profile > Events by Hardware Source >
  - [076] Cycles not in Halt
  - [0C0] Instructions Retired
  - [037] Store Commit Cancels 2 > [0] StCommitCancelWcbFull
    - 35 per thousand retired instructions is bad
Performance of binary compiled with Microsoft Visual Studio 2017 v15.5.2

- Testing conducted as of 3/11/2018. System configuration: AMD Ryzen™ 7 1800X Processor, 2x8GB DDR4-3200 (16-18-18-36), GeForce GTX 1080, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1709, 1920x1080 resolution. BIOS core clock = 3.6GHz, Core Performance Boost = Disable, Global C-state Control = Disable

- Results may vary

Binary compiled after applying the workaround shows higher performance.

Avoid Too Many Non-Temporal Streams

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>145</td>
<td>59</td>
</tr>
</tbody>
</table>

+146%
AVOID TOO MANY NON-TEMPORAL STREAMS
CODE SAMPLE

```c
#include "stdafx.h"
#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_stream_ps(&b[(i + 0) % LEN], p2);
        #else
            /* with workaround */
            _mm_store_ps(&b[(i + 0) % LEN], p2);
        #endif
    }
}
```

```c
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 < steps; i++) {
        step(dt);
    }
    return EXIT_SUCCESS;
}
```

AVOID TOO MANY NON-TEMPORAL STREAMS

PROFILING BEFORE WORKAROUND
AVOID TOO MANY NON-TEMPORAL STREAMS

PROFILING AFTER WORKAROUND

<table>
<thead>
<tr>
<th>Process</th>
<th>Not halted cycle</th>
<th>Ret inst</th>
<th>Store commit ca</th>
<th>IPC (0x0)</th>
<th>CPI (0x0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcb.exe (PID 6036)</td>
<td>49329</td>
<td>214727</td>
<td>46564</td>
<td>2,720</td>
<td>2.31</td>
</tr>
<tr>
<td>x86 host.exe (PID 8948)</td>
<td>42</td>
<td>11</td>
<td>5</td>
<td>0.38</td>
<td>2.04</td>
</tr>
</tbody>
</table>

6 WcbFull per 1000 instructions
AVOID TOO MANY NON-TEMPORAL STREAMS

DISASSEMBLY SNIPPET OF STEP FUNCTION

Before Workaround

1. movups xmm0,xmmword ptr [rsi+rcx*4+43E40h]
2. addps xmm0,xmmword ptr [rsi+r8*4+43E40h]
3. movntps xmmword ptr [rsi+r8*4+43E40h],xmm0
4. movups xmm0,xmmword ptr [rsi+rcx*4+5640h]
5. mulps xmm0,xmm1
6. addps xmm0,xmmword ptr [rsi+r8*4+5640h]
7. movntps xmmword ptr [rsi+r8*4+5640h],xmm0

After Workaround

1. mulps xmm0,xmm1
2. addps xmm0,xmmword ptr [rsi+r8*4+43E40h]
3. movntps xmmword ptr [rsi+r8*4+43E40h],xmm0
4. movups xmm0,xmmword ptr [rsi+rcx*4+5640h]
5. mulps xmm0,xmm1
6. addps xmm0,xmmword ptr [rsi+r8*4+5640h]
7. movups xmmword ptr [rsi+r8*4+5640h],xmm0
Use Best Practices with Spinlocks
USE BEST PRACTICES WITH SPINLOCKS

Brief:
- Use the pause instruction
- test, then test-and-set
- alignas(64) lock variable
  - or _declspec( align(64) )

Profiling:
- Custom CPU Profile > Events by Hardware Source >
  - [076] Cycles not in Halt
  - [0C0] Instructions Retired
  - [0AF] Dynamic Tokens Dispatch Stall Cycles 0 >
  - [4] ALU tokens total unavailable
    - 10K per thousand Instructions is bad

```cpp
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(
        pl, LOCK_IS_TAKEN, LOCK_IS_FREE)) { // lock, xchg, cmp
    }
}
#else
/* GOOD */
void Lock(PLOCK pl) {
    while (1) {
        while (LOCK_IS_TAKEN == *pl)
            _mm_pause();
        if (LOCK_IS_FREE==_InterlockedExchange(pl, LOCK_IS_TAKEN))
            break;
    }
#endif
void Unlock(PLOCK pl) {
    _InterlockedExchange(pl, LOCK_IS_FREE);
}
}
alignas(64) MyLock::LOCK gLock;
```
```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;
}
```
Performance of binary compiled with Microsoft Visual Studio 2017 v15.5.2

- Testing conducted as of 3/11/2018. System configuration: AMD Ryzen™ 7 1800X Processor, 2x8GB DDR4-3200 (16-18-18-36), GeForce GTX 1080, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1709, 1920x1080 resolution. BIOS core clock = 3.6GHz, Core Performance Boost = Disable, Global C-state Control = Disable
- Results may vary

Binary compiled after applying best practices shows higher performance.
USE BEST PRACTICES WITH SPINLOCKS

PROFILING BAD BINARY

11K stalls per 1000 instructions
USE BEST PRACTICES WITH SPINLOCKS

PROFILING GOOD BINARY

0 stall per 1000 instructions
USE BEST PRACTICES WITH SPINLOCKS

DISASM

Bad
1. xor eax,eax
2. lock cmpxchg dword ptr [?gLock@@3IA],ecx
3. cmp eax,ecx
4. je 00000001400010C0

Good
1. cmp dword ptr [?gLock@@3IA],1
2. jne 00000001400010DB
3. nop dword ptr [rax]
4. pause
5. cmp dword ptr [?gLock@@3IA],1
6. je 00000001400010D0
7. mov eax,1
8. xchg eax,dword ptr [?gLock@@3IA]
9. test eax,eax
10. jne 00000001400010C0
Avoid false sharing
AVOID FALSE SHARING

Brief
- Minimize accesses to the same cache line from multiple threads.
- Use thread local storage rather than process shared data whenever possible.

Profiling
- Custom CPU Profile > Events by Hardware Source >
  - [076] Cycles not in Halt
  - [0C0] Instructions Retired
  - [043] Data Cache Refills from System
    - [1] LS_MABRESP_LCL_CACHE
    - [4] LS_MABRESP_RMT_CACHE, Hit in cache; Remote CCX and Home Node on different die
AVOID FALSE SHARING

PERFORMANCE

- Performance of binary compiled with Microsoft Visual Studio 2017 v15.5.2
  - Testing conducted as of 3/11/2018. System configuration: AMD Ryzen™ 7 1800X Processor, 2x8GB DDR4-3200 (16-18-18-36), GeForce GTX 1080, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1709, 1920x1080 resolution. BIOS core clock = 3.6GHz, Core Performance Boost = Disable, Global C-state Control = Disable
  - Results may vary

- Binary compiled after using Thread Local Storage shows higher performance. This binary avoids false sharing.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>407</td>
<td>53</td>
</tr>
</tbody>
</table>

Avoid False Sharing

+669%
# Avoid False Sharing

```cpp
#include <windows.h> <chrono> <numeric> <thread>
using namespace std::chrono;
#define NUM_ITER 2000000000
struct Data {
    unsigned long sum;
};
struct ThreadData {
    int seed;
    Data data;
};
DWORD WINAPI ThreadProcCallback(void* param)
{
    ThreadData *p = (ThreadData*)param;
    srand(p->seed);
    #if 0 /* process shared data */
    p->data.sum = 0;
    for (int i = 0; i < NUM_ITER; i++) {
        p->data.sum += rand() % 2;
    }
    #else /* thread local data */
    int local_sum = 0;
    for (int i = 0; i < NUM_ITER; i++) {
        local_sum += rand() % 2;
    }
    p->data.sum = local_sum;
    #endif
    return 0;
}
int main(int argc, char *argv[]) {
    int 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) {
        a[i].seed = seed;
        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].data.sum);
    }
    delete[] a;
    delete[] threads;
    return EXIT_SUCCESS;
}
```
AVOID FALSE SHARING
PROFILING PROCESS SHARED DATA

Many DC refills
### AVOID FALSE SHARING
PROFILING THREAD LOCAL STORAGE

#### Near 0 DC refills

<table>
<thead>
<tr>
<th>Process</th>
<th>Not halted cycle</th>
<th>Ret inst</th>
<th>DC refills</th>
</tr>
</thead>
<tbody>
<tr>
<td>falsesharing.exe (PID 6012)</td>
<td>341442</td>
<td>639048</td>
<td>20</td>
</tr>
</tbody>
</table>

#### Functions (for falsesharing.exe (PID 6012))

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Not halted cycle</th>
<th>Ret inst</th>
<th>DC refills</th>
</tr>
</thead>
<tbody>
<tr>
<td>KernelBase.dll!0x67bd6077c41</td>
<td>341442</td>
<td>639048</td>
<td>20</td>
</tr>
<tr>
<td>usrcase!0x67bd6077f3e20</td>
<td>247566</td>
<td>529952</td>
<td>1</td>
</tr>
<tr>
<td>usrcase!0x67bd6077decf</td>
<td>214898</td>
<td>433177</td>
<td>1</td>
</tr>
<tr>
<td>usrcase!0x67bd6033f00</td>
<td>175456</td>
<td>244057</td>
<td>1</td>
</tr>
<tr>
<td>usrcase!0x67bd6033f10</td>
<td>165827</td>
<td>134018</td>
<td>1</td>
</tr>
<tr>
<td>ThreadingBase.dll!0x67bd6033f0</td>
<td>177027</td>
<td>109135</td>
<td>1</td>
</tr>
</tbody>
</table>
AVOID FALSE SHARING
DISASM SNIPPET OF THREADPROCCALLBACK

1. call qword ptr [\texttt{\_imp\_rand}]
2. and eax, 80000001h
3. jge 00000001400010A5
4. dec eax
5. or eax, 0FFFFFFFEh
6. inc eax
7. add dword ptr [rbx+4], eax
8. sub rdi, 1
9. jne 0000000140001091

Frequently accessed process shared data in innermost loop

Minimized access to process shared data by using thread local data

1. call qword ptr [\texttt{\_imp\_rand}]
2. and eax, 80000001h
3. jge 00000001400010A5
4. dec eax
5. or eax, 0FFFFFFFEh
6. inc eax
7. add ebx, eax
8. sub rdi, 1
9. jne 0000000140001091
10. mov dword ptr [rsi+4], ebx
RYZEN 2018 ROLL-OUT PLAN

Premium Desktop
- RYZEN 1st Generation CPU
- RYZEN THREADRIPPER 1st Generation
- RYZEN PRO 1st Generation
- RYZEN Desktop APU
- RYZEN 2nd Generation CPU
- RYZEN THREADRIPPER 2nd Generation
- RYZEN PRO 2nd Generation

Premium Mobile
- RYZEN Mobile APU
- RYZEN Mobile APU

Roadmap subject to change
Thank You!
CONTACT

Ken Mitchell
- Kenneth.Mitchell@amd.com
- @kenmitchellken

Elliot Kim
- Elliot.Kim@amd.com
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

© 2018 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.