
RYZEN
AMD

AMD RYZEN™ CPU OPTIMIZATION

Presented by Ken Mitchell & Elliot Kim

AMD RYZEN™ CPU OPTIMIZATION

ABSTRACT

- 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.

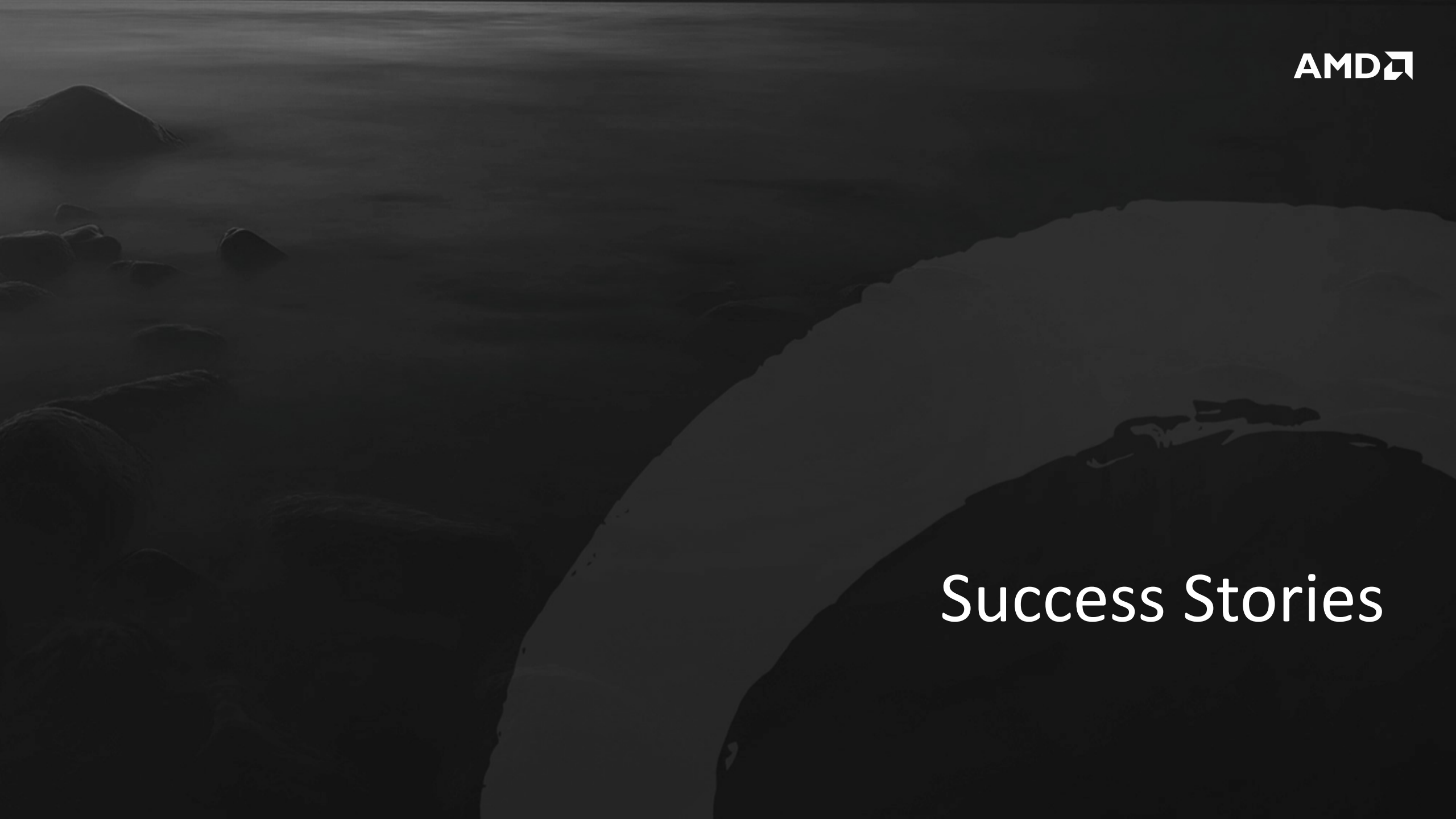
  - Ken Mitchell is a Senior Member of Technical Staff in the Radeon Technologies Group/AMD ISV Game Engineering team where he focuses on helping game developers utilize AMD CPU cores efficiently. Previously, he was tasked with automating & analyzing PC applications for performance projections of future 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

AMD

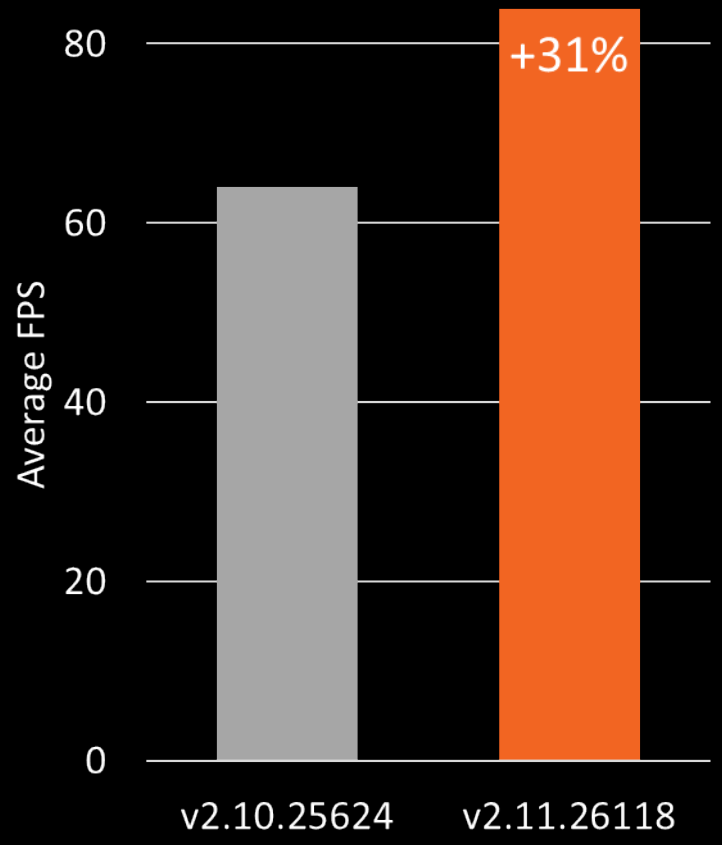
- Success Stories
- “Zen” Family Processors
- AMD µProf Profiler
- Optimizations & Lessons Learned
- Roadmap
- QA


AMD
Success Stories

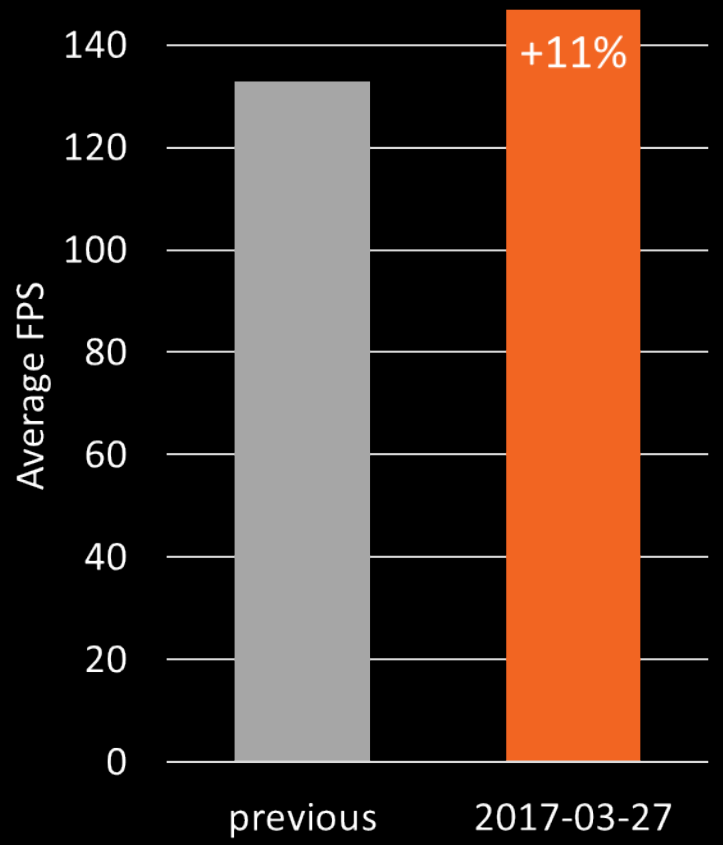
SUCCESS STORIES

1080P GAMING PERFORMANCE IMPROVEMENTS

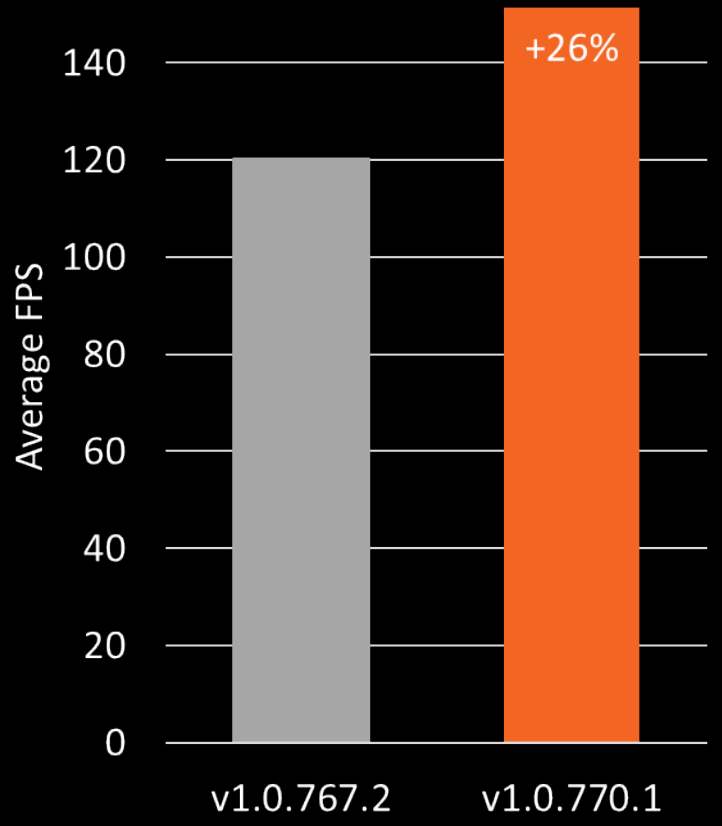
Ashes of the Singularity
1080p, High, GPU Test



Total War: Warhammer
1080p, High

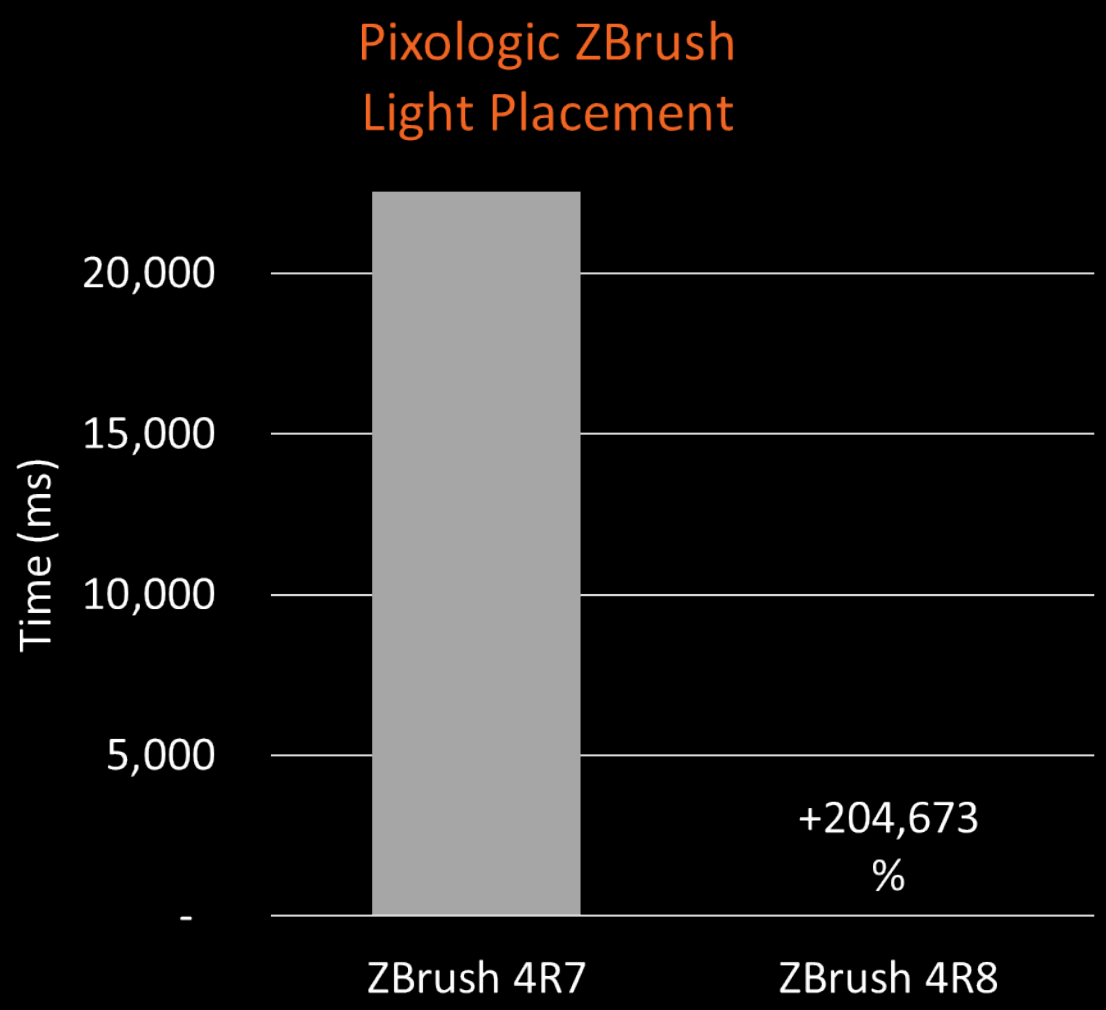


Rise of the Tomb Raider
1080p, High



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
- 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.

Rise of the Tomb Raider

- https://community.amd.com/community/gaming/blog/2017/06/23/even-more-performance-updates-for-ryzen-customers
- 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

Total War: Warhammer

- https://community.amd.com/community/gaming/blog/2017/04/06/amd-ryzen-community-update-3
- 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.

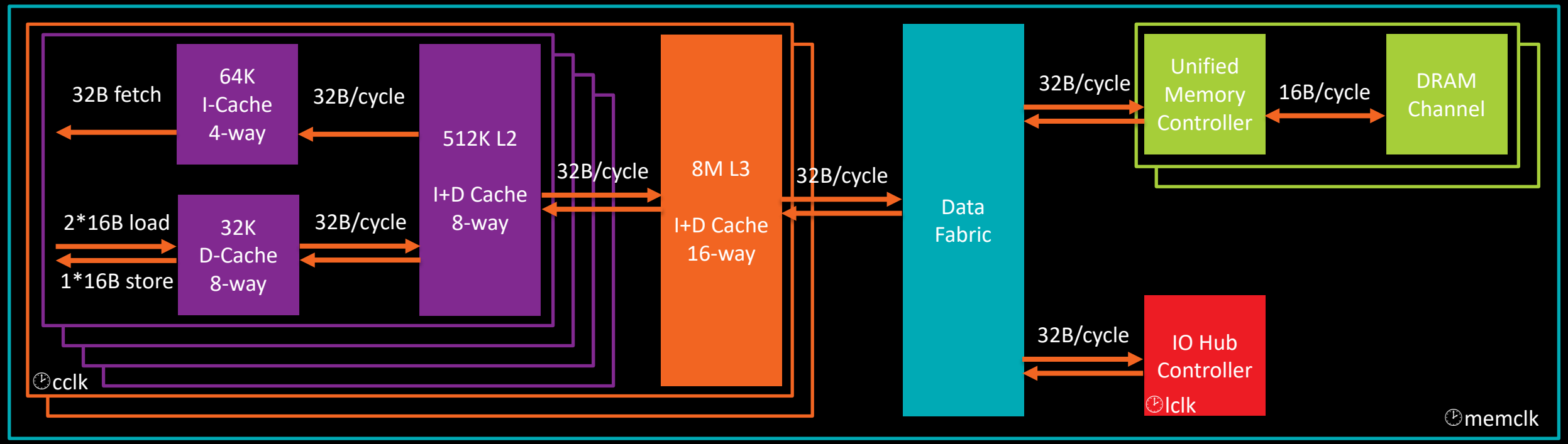
ZBrush

- https://community.amd.com/community/gaming/blog/2017/06/23/even-more-performance-updates-for-ryzen-customers
- 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.


AMD
“Zen” Family Processors

“SUMMIT RIDGE”

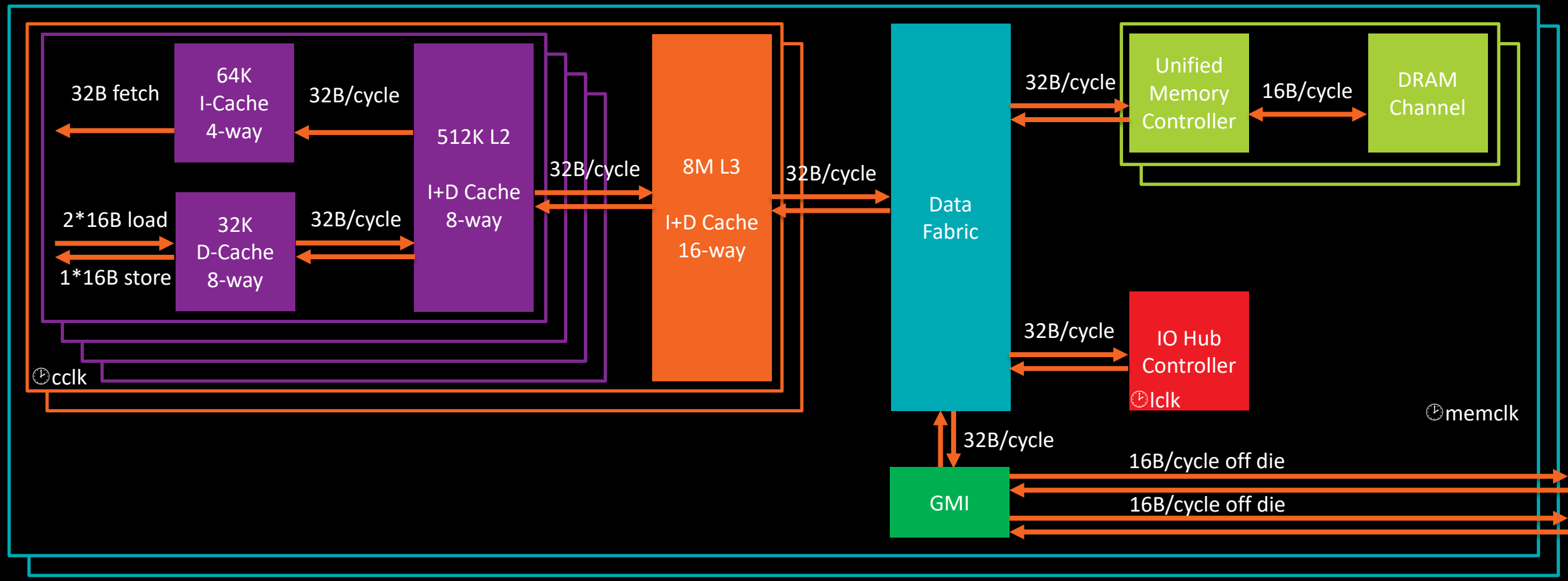
DATA FLOW



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

THREADRIPPER(TM) PROCESSOR

DATA FLOW

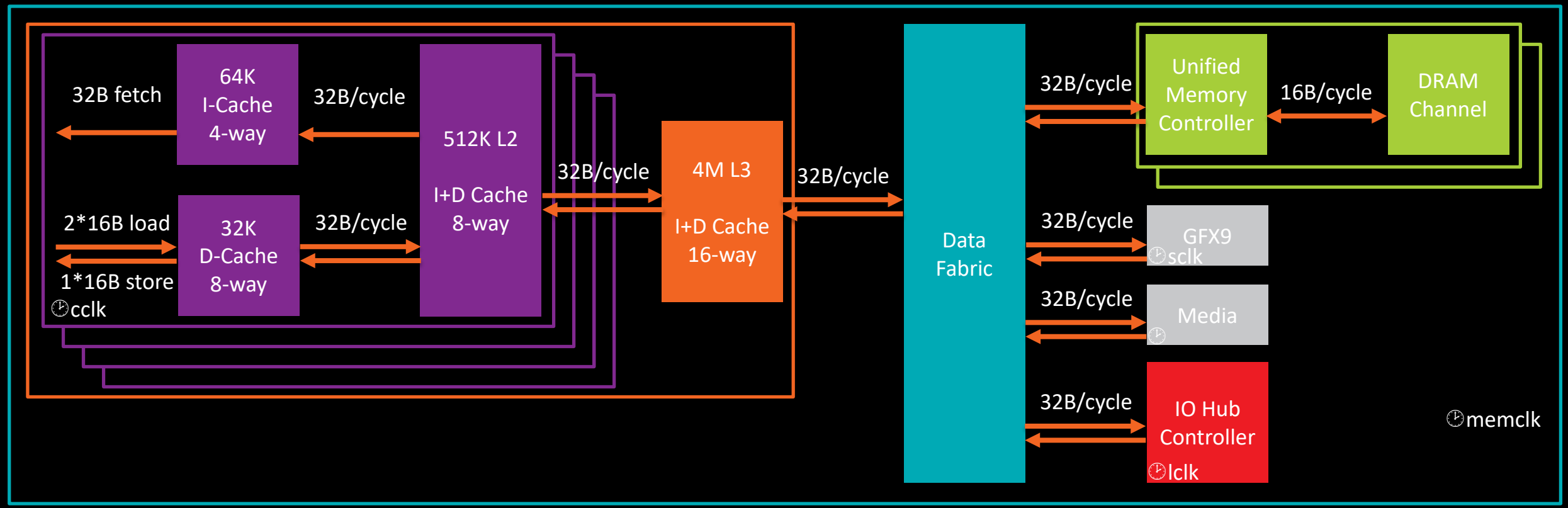


- cclk 4.0 GHz / 3.4 GHz
- memclk 1.3 GHz (DDR4-2667)
- lclk 727 MHz

- 2x Die connected by GMI

“RAVEN RIDGE”

DATA FLOW

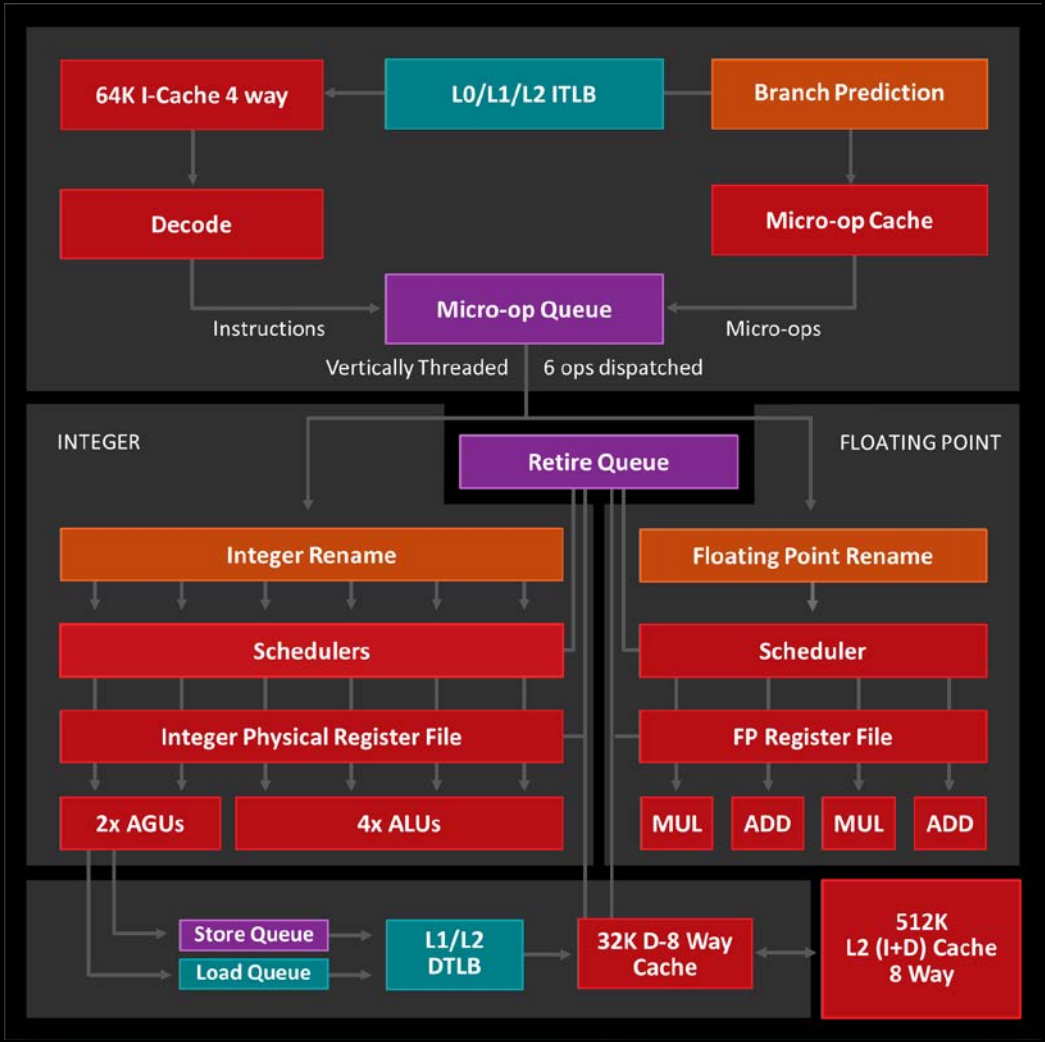


- cclk 3.9 GHz / 3.6 GHz
- memclk 1.3 GHz (DDR4-2667)
- lclk 496
- sclk 1.25 GHz (11CU = 704 shaders)

- 1 CCX with GFX9 “Vega” & MultiMedia Hub

MICROARCHITECTURE

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.
  - 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

Competitively shared structures

Competitively shared and SMT Tagged

Competitively shared with Algorithmic Priority

Statically Partitioned

MICROARCHITECTURE

INSTRUCTION SET EVOLUTION

YEAR	FAMILY	PRODUCT FAMILY	ARCHITECTURE	EXAMPLE MODEL	ADX	CLFLUSHOPT	RDSEED	SHA	SMAP	XGETBV	XSAVEC	XSAVES	AVX2	BMI2	MOVBE	RDRND	SMEP	FSGSBASE	XSAVEOPT	BMI	FMA	F16C	AES	AVX	OSXSAVE	PCLMULQDQ	SSE4.1	SSE4.2	XSAVE	SSSE3	CLZERO	FMA4	TBM	XOP
2017	17h	“Summit Ridge”	“Zen”	Ryzen 7 1800X	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	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
2014	15h	“Kaveri”/”Godavari”	“Steamroller”	A10-7890K	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	0	1	1	1
2012	15h	“Vishera”	“Piledriver”	FX-8370	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	0	1	1	1
2011	15h	“Zambezi”	“Bulldozer”	FX-8150	0	0	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	0	1	0	1
2013	16h	“Kabini”	“Jaguar”	A6-1450	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0
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	1	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
2009	10h	“Greyhound”	“Greyhound”	Phenom II X4 955	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	0

+ 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

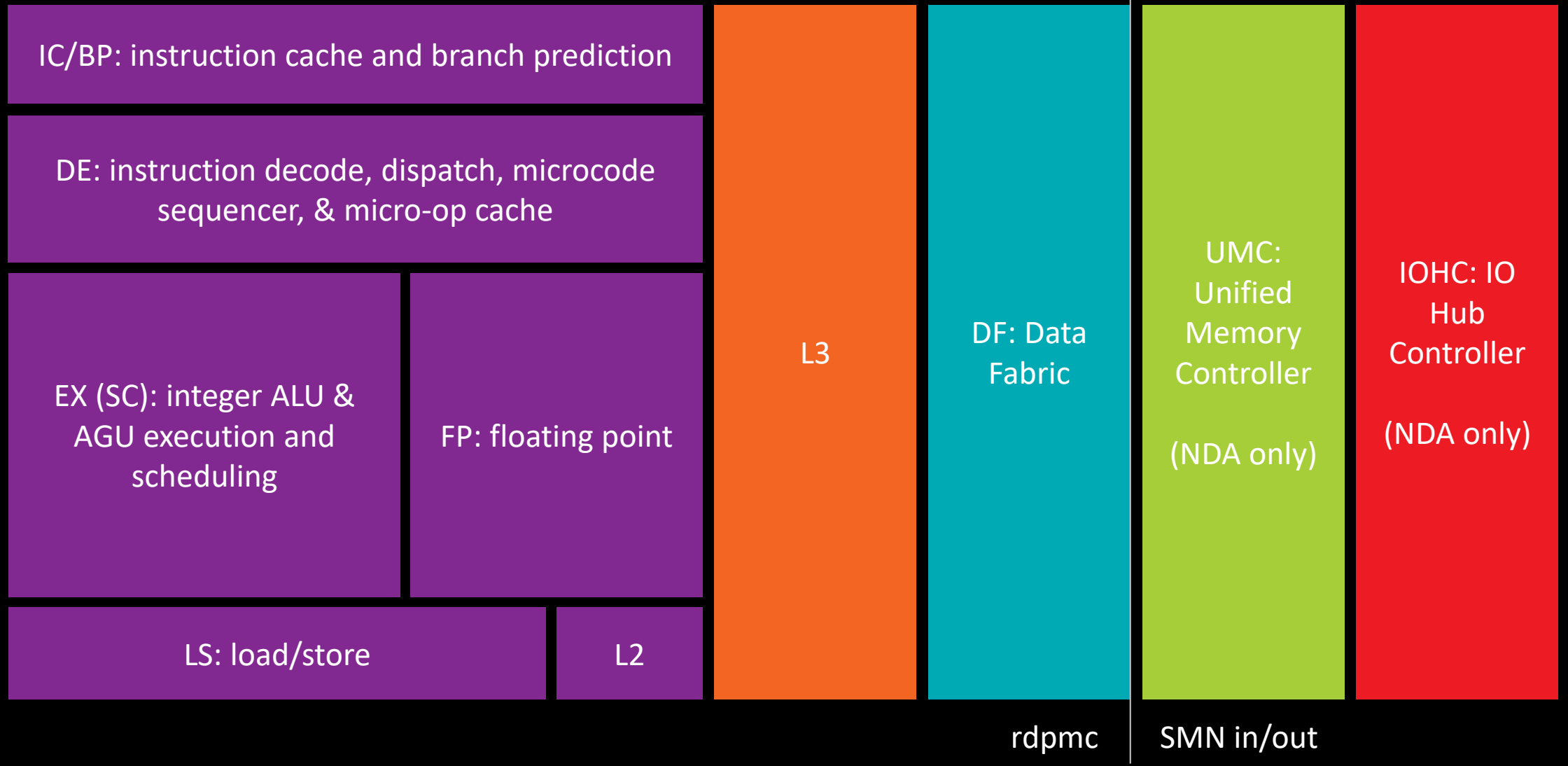
AMD PROFILER

- 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

MICROARCHITECTURE

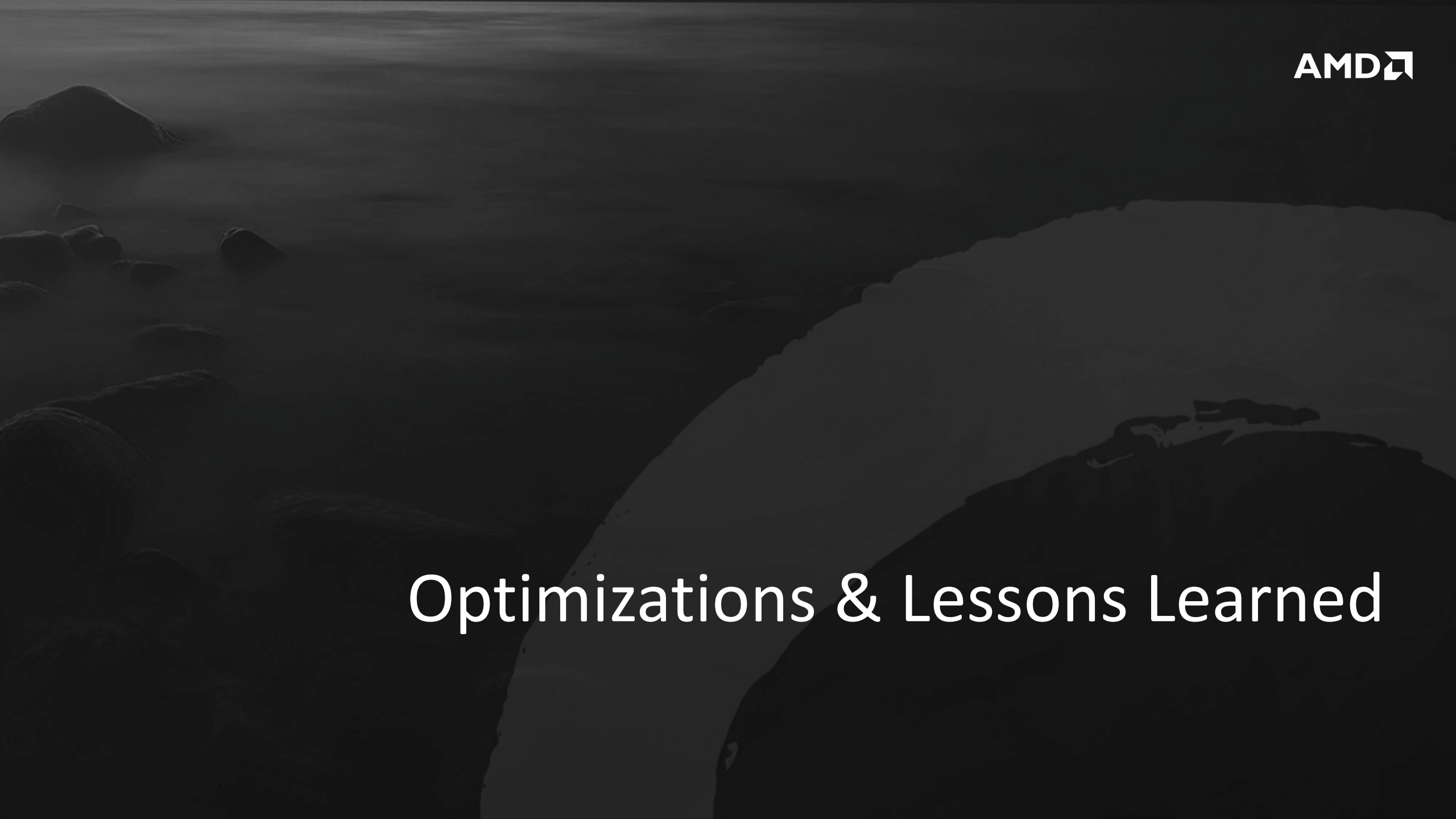
PERFORMANCE MONITORING COUNTER DOMAINS



POWER MANAGEMENT

FIXED FREQUENCY

- 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


AMD
Optimizations & Lessons Learned

AGENDA – OPTIMIZATIONS & LESSONS LEARNED

AMD

- 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


AMD
Use Microsoft® Visual Studio 2017

USE MICROSOFT® VISUAL STUDIO 2017

FOR OPTIMAL CODE GENERATION

AMD

- 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

USE MICROSOFT® VISUAL STUDIO 2017

FOR OPTIMAL CODE GENERATION

AMD

MSVS 2015

```
1. movss    DWORD PTR T[0], xmm1
2. movss    DWORD PTR T[4], xmm2
3. movss    DWORD PTR T[8], xmm3
4. movss    DWORD PTR T[12], xmm4
5. movups xmm0, DWORD PTR T[0]
```

MSVS 2017

```
1. movss    DWORD PTR T[0], xmm1
2. movups xmm0, DWORD PTR T[0]
3. movss    DWORD PTR T[4], xmm2
4. movss    DWORD PTR T[8], xmm3
5. movss    DWORD PTR T[12], xmm4
```


AMD
Use best practices when counting cores

USE ALL PHYSICAL CORES

AMD

- 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/

```
// 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;
    }
  }
  return count;
}
```

GET GETLOGICALPROCESSORINFORMATIONEX BUFFER LENGTH AT RUNTIME

AMD

- Brief:
  - 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

```
// 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);
}
```

AVOID SIGNED, NARROWING AFFINITY MASKS

AMD

- 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.
    - See https://msdn.microsoft.com/en-us/library/336xbhcz.aspx
- Example:
  - Bad
    - popcnt = 0
    - mask b[32] = 0x0000000000000000
  - Good
    - popcnt = 64
    - mask b[32] = 0x0000000100000000

```
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
```


AMD
Avoid Too Many Non-Temporal Streams

AVOID TOO MANY NON-TEMPORAL STREAMS

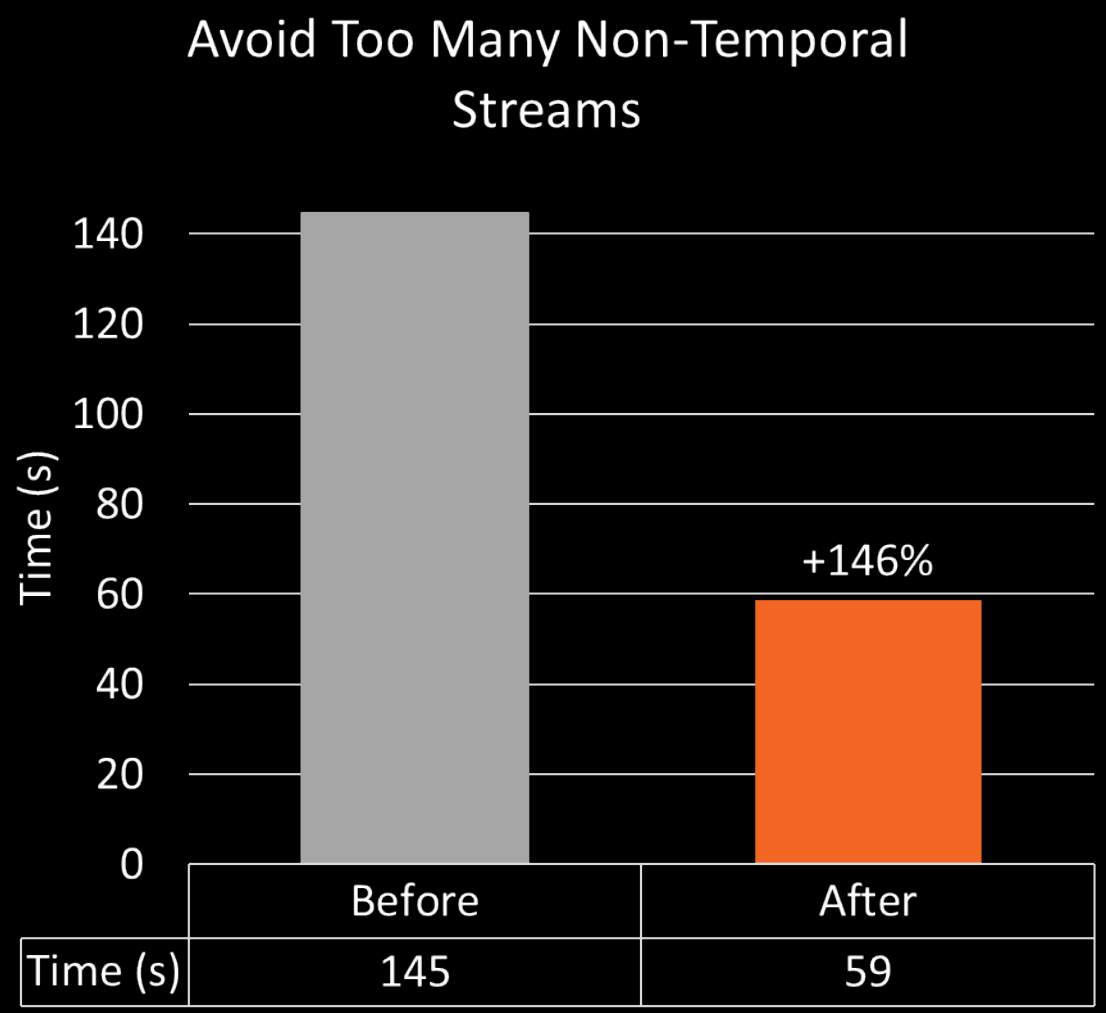
SUMMARY

- 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

AVOID TOO MANY NON-TEMPORAL STREAMS

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 very
- Binary compiled after applying the workaround shows higher performance.



AVOID TOO MANY NON-TEMPORAL STREAMS

CODE SAMPLE

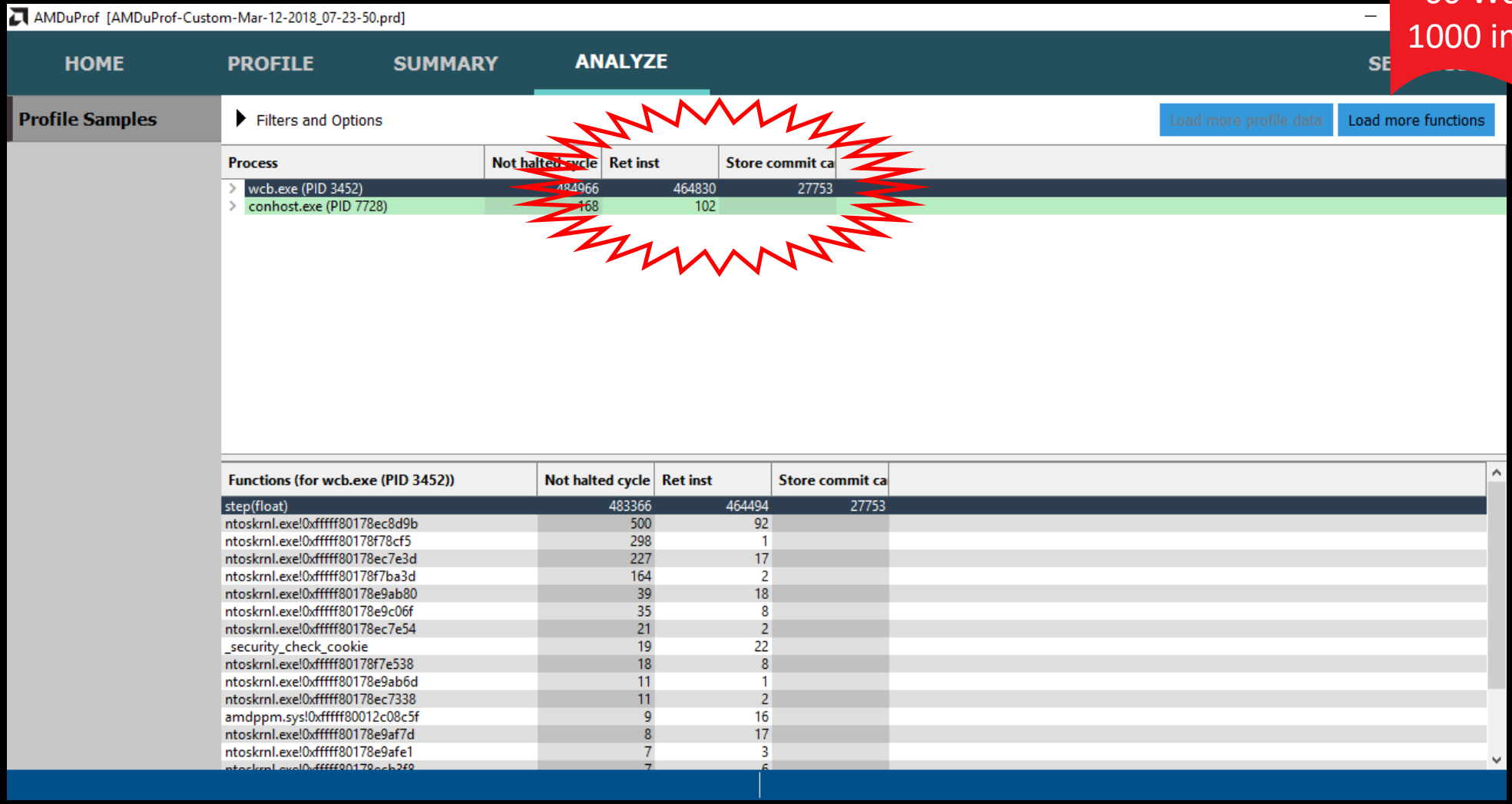
```
#include "stdafx.h"
#include <intrin.h>
#include <numeric>
#include <chrono>
#define LEN 64000
alignas(64) float a[LEN];
alignas(64) float b[LEN];

void step(float dt) {
    for (size_t i = 0; i < LEN; i += 8) {
        // x,y,z,w,vx,vy,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
    }
}
```

```
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;
}
```

AVOID TOO MANY NON-TEMPORAL STREAMS

PROFILING BEFORE WORKAROUND

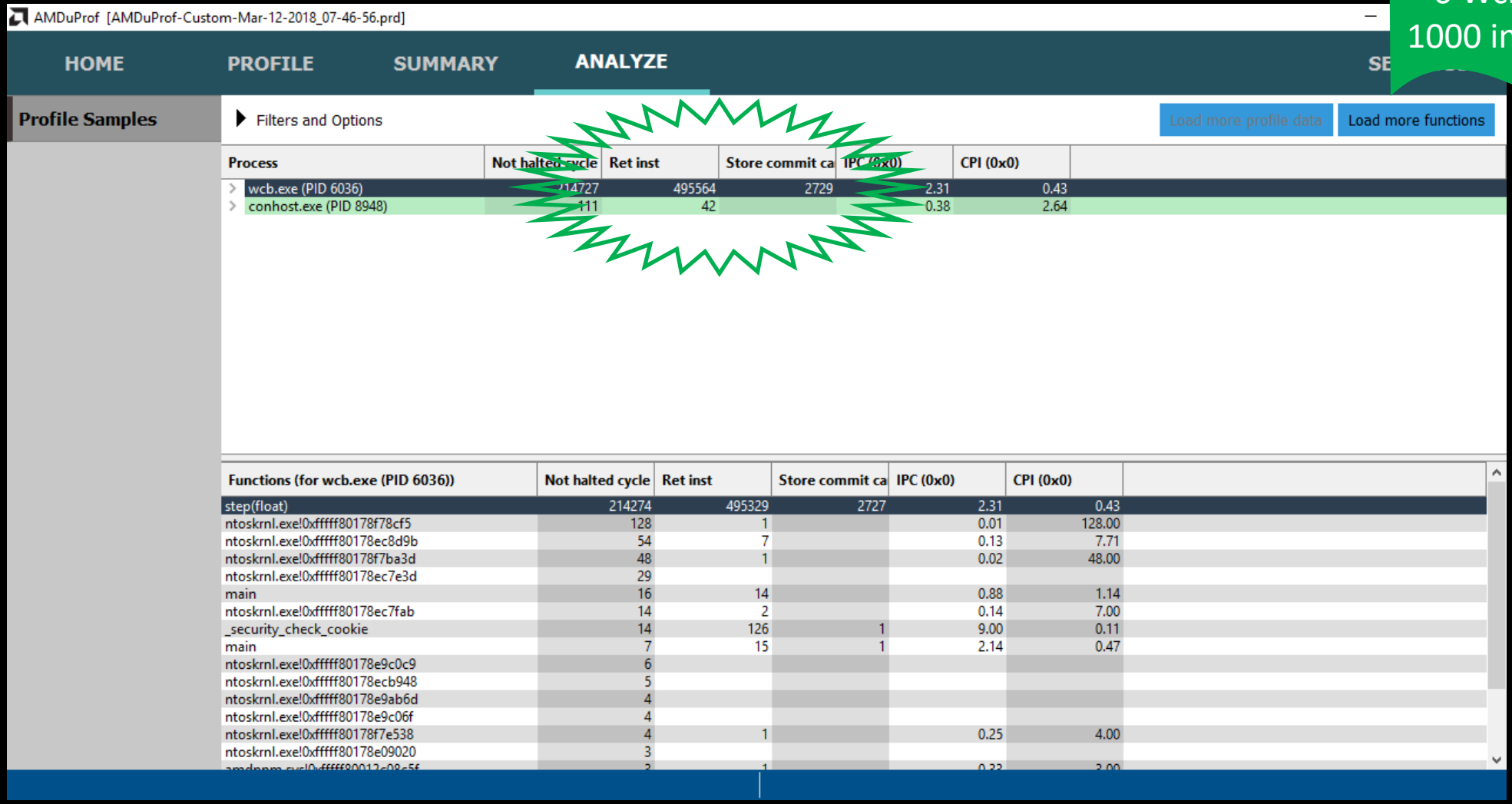
60 WcbFull per 1000 instructions



AVOID TOO MANY NON-TEMPORAL STREAMS

PROFILING AFTER WORKAROUND

6 WcbFull per 1000 instructions



AVOID TOO MANY NON-TEMPORAL STREAMS

DISASSEMBLY SNIPPET OF STEP FUNCTION

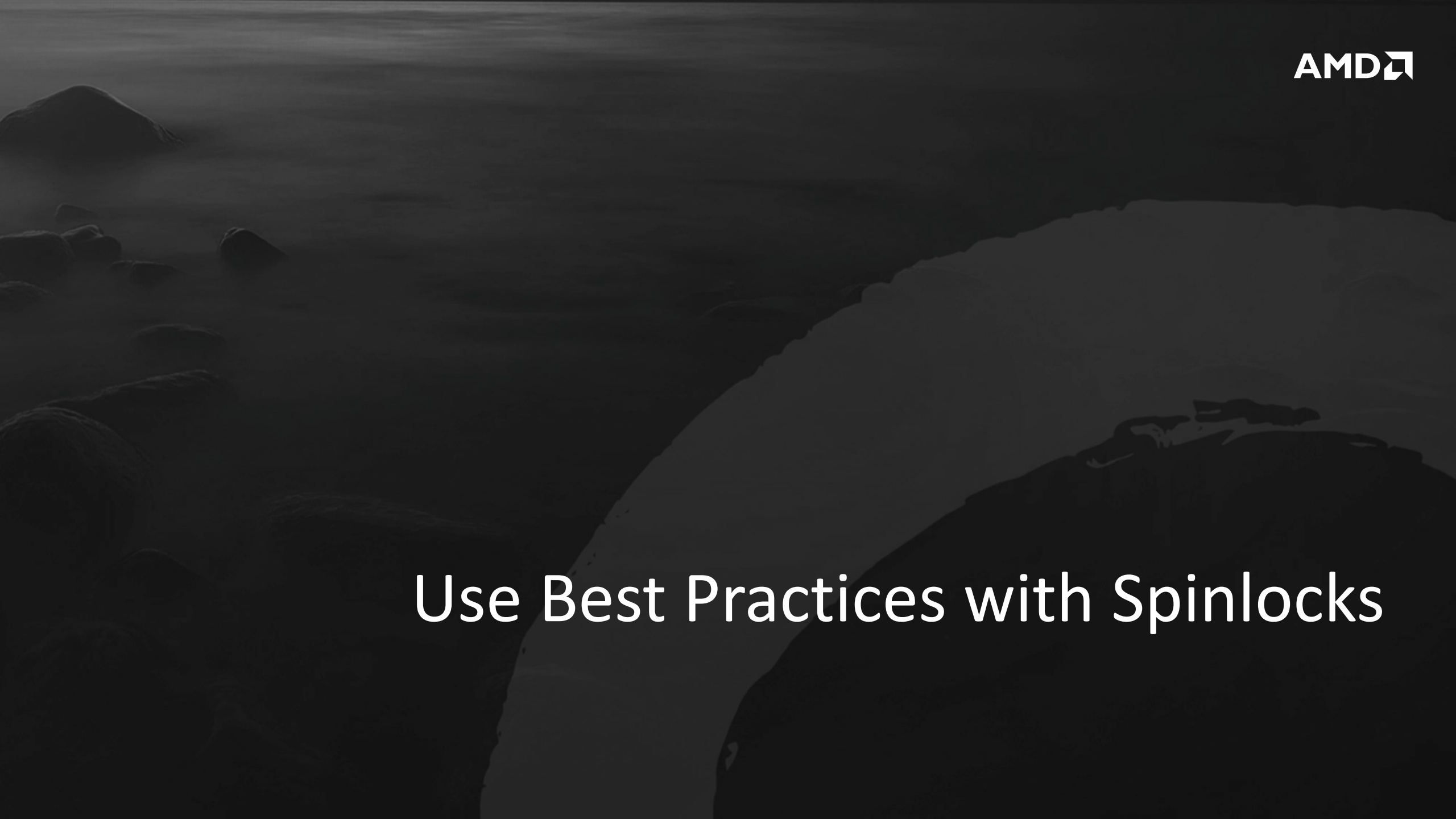
AMD

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
```


AMD
Use Best Practices with Spinlocks

USE BEST PRACTICES WITH SPINLOCKS

AMD

- 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

```
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;
```

USE BEST PRACTICES WITH SPINLOCKS

CODE SAMPLE

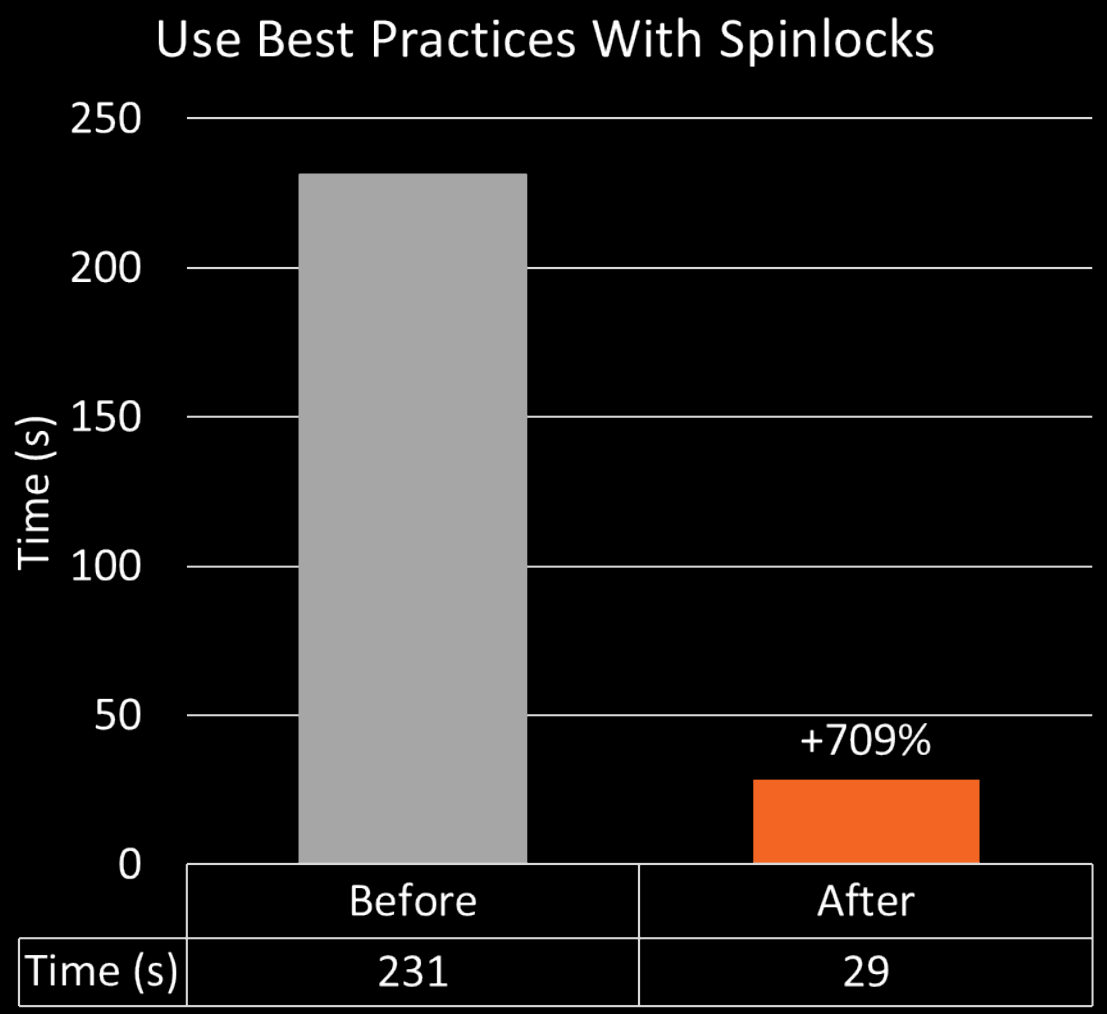
```
#include "intrin.h"
#include "stdio.h"
#include "windows.h"
#include <chrono>
#include <numeric>
#include <thread>
#define LEN 512
alignas(64) float b[LEN][4][4];
alignas(64) float c[LEN][4][4];
DWORD WINAPI ThreadProcCallback(LPVOID data) {
    MyLock::Lock(&gLock);
    alignas(64) float a[LEN][4][4];
    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): %lf\n", \
        1000.0 * time_span.count());
    delete[] threads;
    return EXIT_SUCCESS;
}
```

USE BEST PRACTICES WITH SPINLOCKS

PERFORMANCE

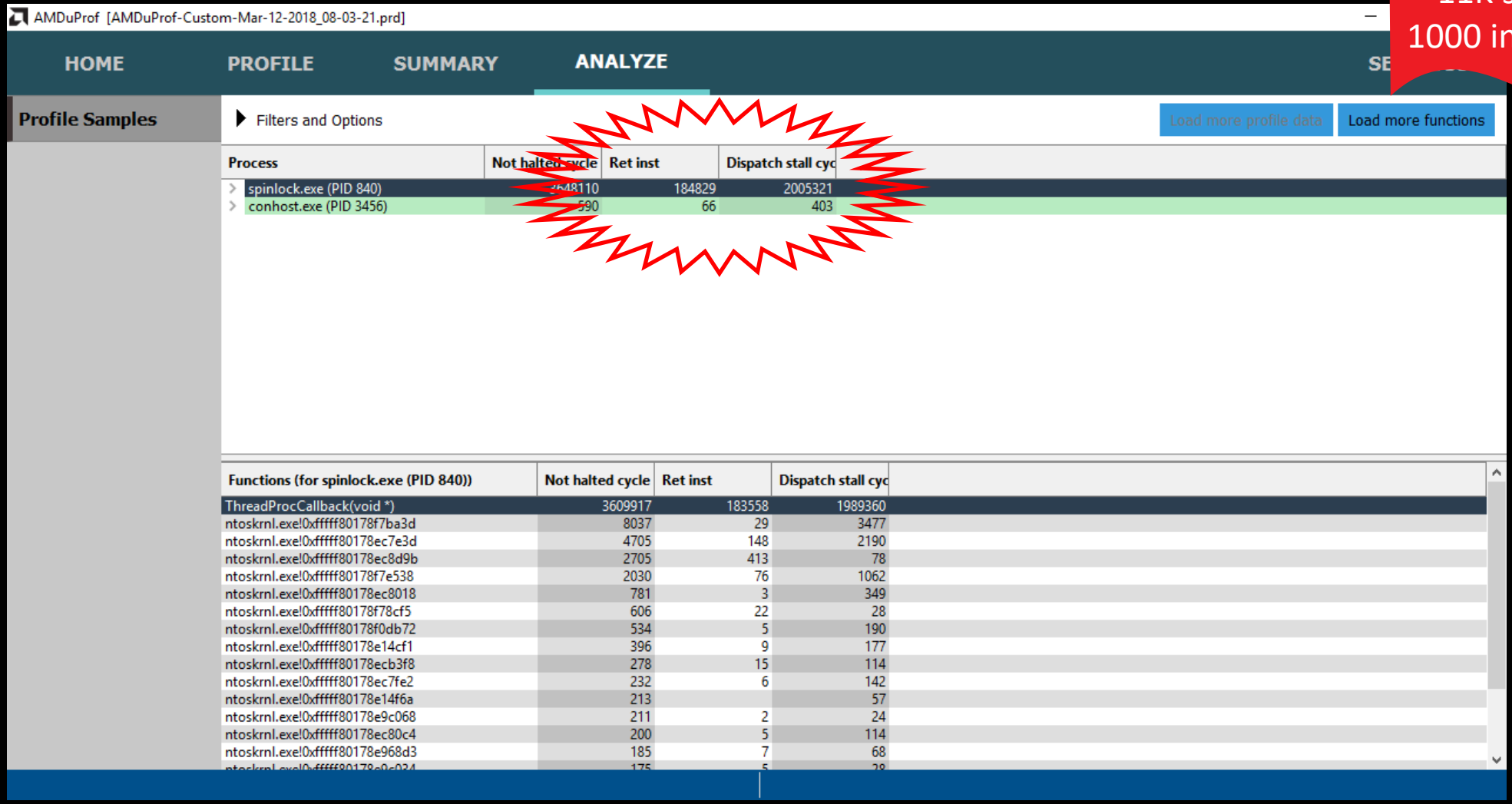
- Performance of binary compiled with Microsoft Visual Studio 2017 v15.5.2
  - Testing conducted as of 3/11/2018. System configuration: AMD Ryzen™ 7 1800X Processor, 2x8GB DDR4-3200 (16-18-18-36), GeForce GTX 1080, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 1709, 1920x1080 resolution. BIOS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



USE BEST PRACTICES WITH SPINLOCKS

PROFILING BAD BINARY

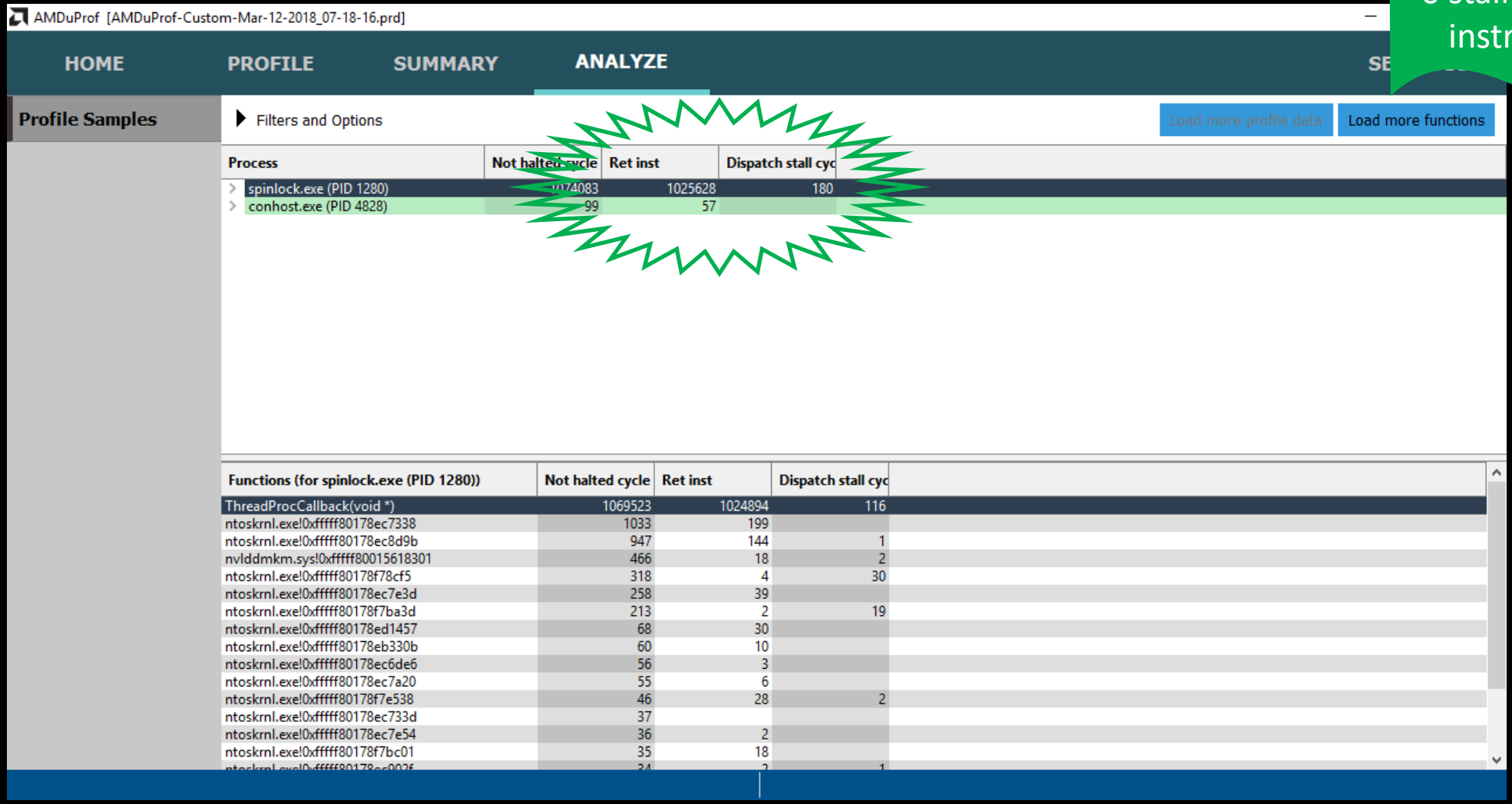
11K stalls per 1000 instructions



USE BEST PRACTICES WITH SPINLOCKS

PROFILING GOOD BINARY

0 stall per 1000 instructions



USE BEST PRACTICES WITH SPINLOCKS

DISASM

AMD

Bad

```
1.  xor          eax,eax
2.  lock cmpxchg dword ptr [?gLock@@3IA],ecx
3.  cmp          eax,ecx
4.  je           0000000140010C0
```

Good

```
1.  cmp          dword ptr [?gLock@@3IA],1
2.  jne          0000000140010DB
3.  nop          dword ptr [rax]
4.  pause
5.  cmp          dword ptr [?gLock@@3IA],1
6.  je           0000000140010D0
7.  mov          eax,1
8.  xchg         eax,dword ptr [?gLock@@3IA]
9.  test         eax,eax
10. jne          0000000140010C0
```


AMD
Avoid false sharing

AVOID FALSE SHARING

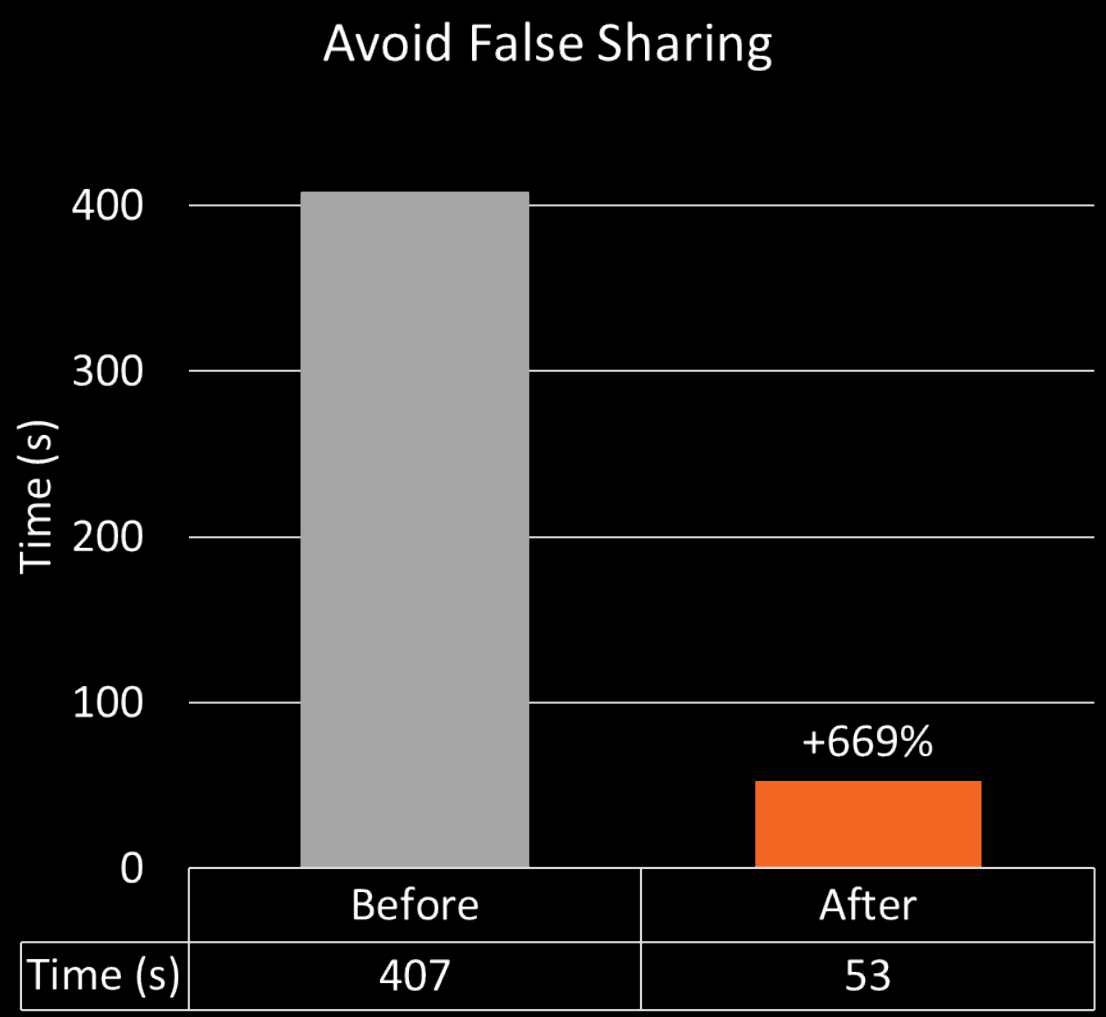
AMD

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



AVOID FALSE SHARING

CODE SAMPLE

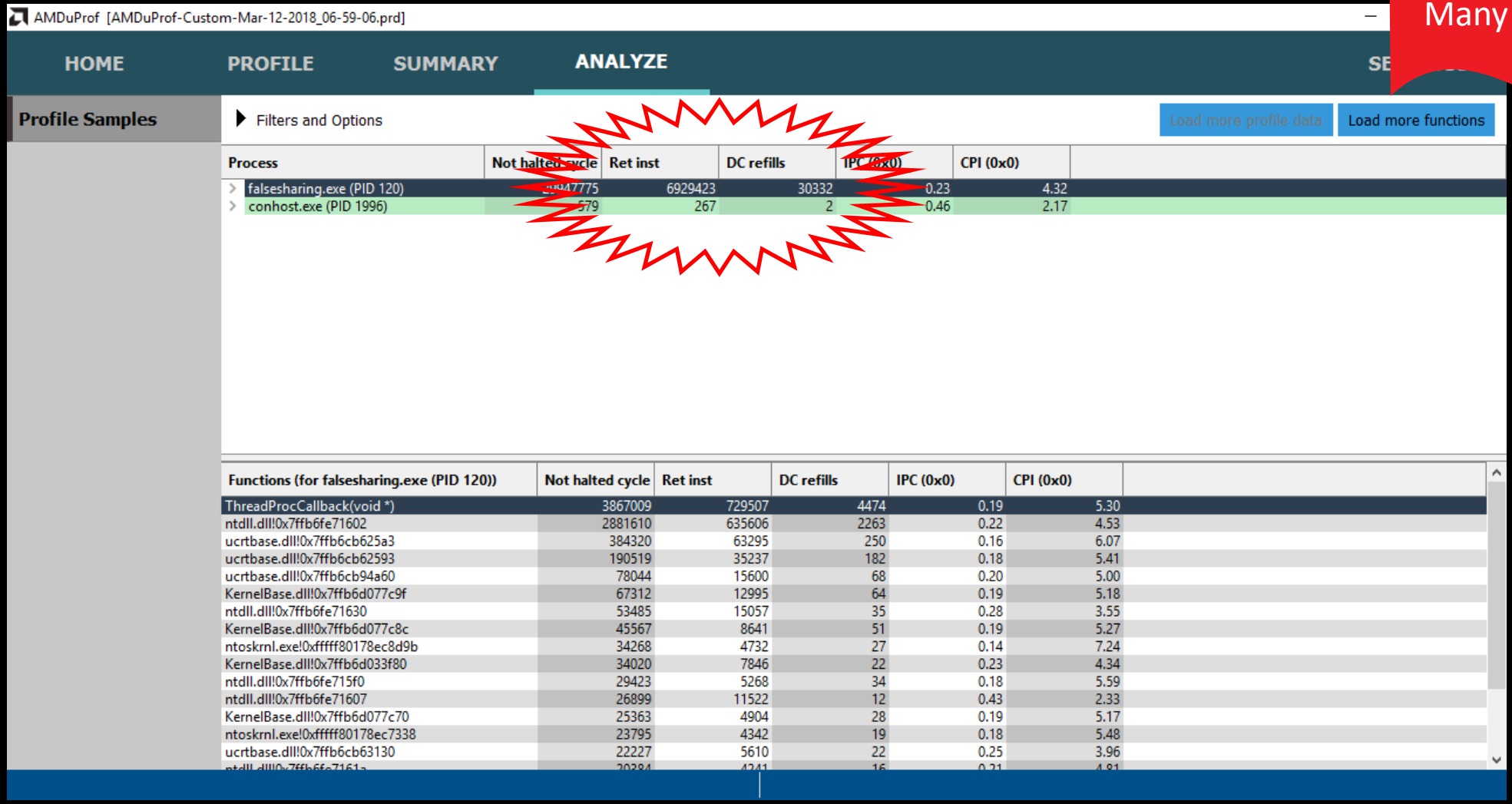
```
/* #include <windows.h> <chrono> <numeric> <thread> */
using namespace std::chrono;
#define NUM_ITER 2000000000
struct Data {
    unsigned long sum;
};
struct ThreadData {
    int seed;
    Data data;
};
DWORD WINAPI ThreadProcCallback(void* param) {
    ThreadData *p = (ThreadData*)param;
    srand(p->seed);
#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

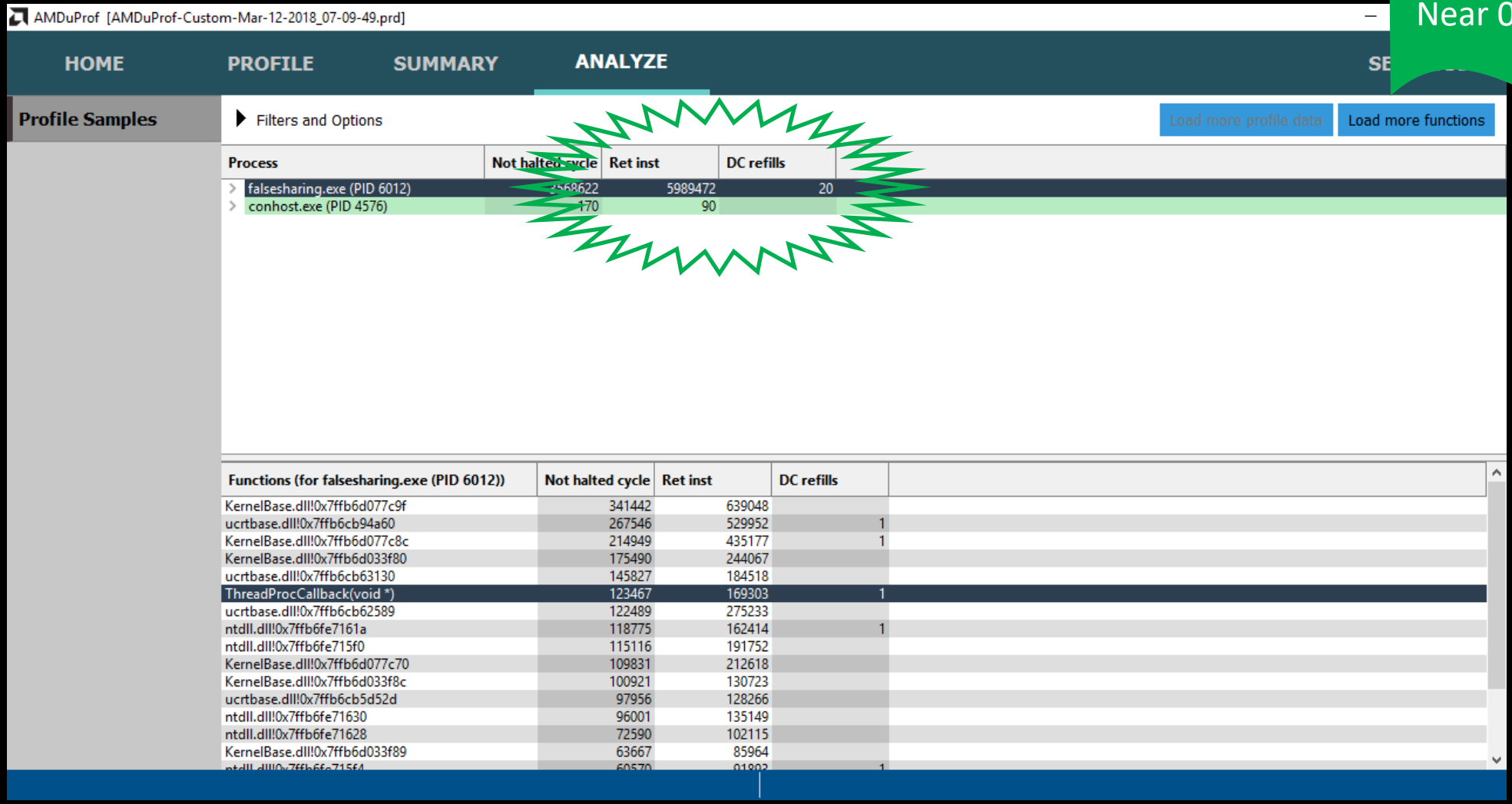
Many DC refills



AVOID FALSE SHARING

PROFILING THREAD LOCAL STORAGE

Near 0 DC refills



AVOID FALSE SHARING

DISASM SNIPPET OF THREADPROCCALLBACK

Frequently accessed process shared data in innermost loop

```
1.  call    qword ptr [__imp_rand]
2.  and     eax,80000001h
3.  jge     0000000140010A5
4.  dec     eax
5.  or      eax,0FFFFFFFEh
6.  inc     eax
7.  add     dword ptr [rbx+4],eax
8.  sub     rdi,1
9.  jne     0000000140001091
```

Minimized access to process shared data by using thread local data

```
1.  call    qword ptr [__imp_rand]
2.  and     eax,80000001h
3.  jge     0000000140010A5
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
```


AMD
Roadmap

RYZEN

2018 ROLL-OUT PLAN

	2017	2018 Q1	2018 Q2	2018 2H
Premium Desktop	RYZEN 1st Generation CPU RYZEN THREADRIPPE 1st Generation RYZEN PRO 1st Generation	RYZEN Desktop APU	RYZEN 2nd Generation CPU	RYZEN THREADRIPRER 2nd Generation RYZEN PRO 2nd Generation
Premium Mobile	RYZEN 7 5 Mobile APU	RYZEN 3 Mobile APU	RYZEN PRO Mobile APU	

Roadmap subject to change


AMD
QA


AMD
Thank You!

CONTACT

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

DISCLAIMER & 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


AMD