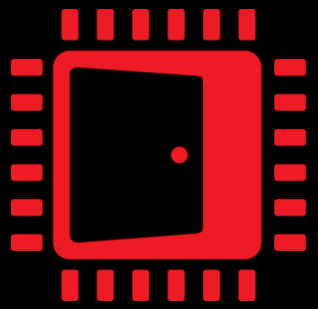


AMD
GPUOpen

GDC

AMD RYZEN™ PROCESSOR SOFTWARE OPTIMIZATION

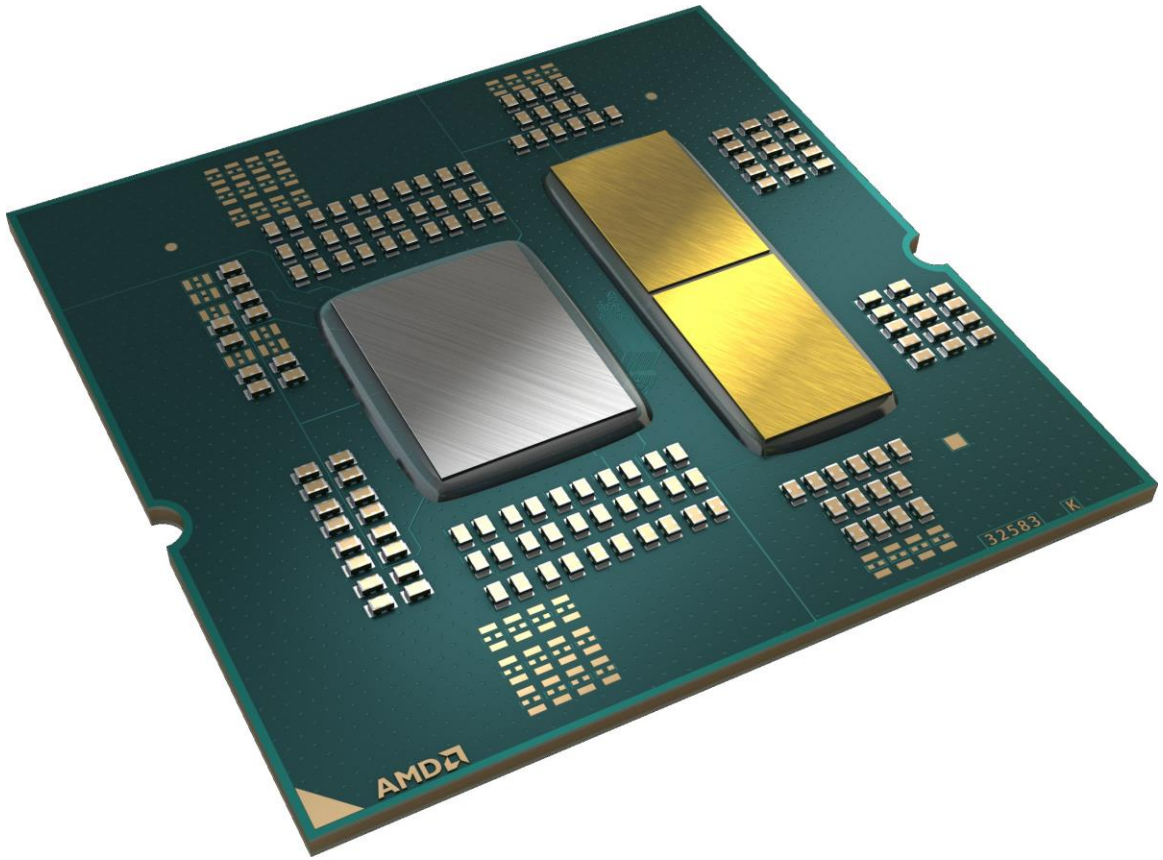
KEN MITCHELL

AMD
together we advance_

AGENDA

- Abstract
- Speaker Biography
- Products
- Data Flow
- Microarchitecture
- Best Practices
- Optimizations

ABSTRACT



- Break through CPU bottlenecks to reach higher frames-per-second!
- Dive into data flow, simultaneous multi-threading, resource sharing, instruction set evolution, cache hierarchies, and coherency.
- Unlock powerful profiling tools and application analysis techniques.
- Discover best practices and lessons learned.
- Attack valuable code optimization opportunities.
- Examples include C/C++, assembly, and hardware performance-monitoring counters.

SPEAKER BIOGRAPHY

- Ken Mitchell is a Fellow and Technical Lead in the AMD Software Performance Engineering team where he collaborates with Microsoft® Windows® and AMD engineers to optimize AMD processors for better performance-per-watt. He began working at AMD in 2005. His previous work includes helping game developers utilize AMD processors efficiently, analyzing PC applications for performance projections of future AMD products, as well as developing system benchmarks. Ken earned a Bachelor of Science in Computer Science degree at the University of Texas at Austin.

- Kenneth.Mitchell@amd.com

PRODUCTS

FORMER CODE NAMES

CPU Architecture	Mobile (Laptop)	Desktop	Workstation
“Zen 4”	**“Hawk Point” AMD Ryzen™ 8040 Series** **“Phoenix” AMD Ryzen™ 7040 Series**	**“Raphael AM5” AMD Ryzen™ 7000 Series**	**“Storm Peak”** **AMD Threadripper™ 7000 Series**
“Zen 3”	“Rembrandt” AMD Ryzen™ 6000 Series “Cezanne” AMD Ryzen™ 5000 Series	“Vermeer” AMD Ryzen™ 5000 Series	“Chagall PRO” AMD Threadripper™ 5000 Series
“Zen 2”	“Renoir” AMD Ryzen™ 4000 Series	“Matisse” AMD Ryzen™ 3000 Series	“Castle Peak” AMD Threadripper™ 3000 Series
“Zen”	“Picasso” AMD Ryzen™ 3000 Series “Raven Ridge” AMD Ryzen™ 2000 Series	“Pinnacle Ridge” AMD Ryzen™ 2000 Series “Summit Ridge” AMD Ryzen™ 1000 Series	“Threadripper” AMD Threadripper™ 1000 Series

- Table shown does not include all former code names for each CPU architecture.

“HAWK POINT” AMD RYZEN™ 8040 SERIES PROCESSORS

Mobile Model	Cores / Threads	Boost / Base Frequency	GPU Compute Units	AMD Ryzen™ AI	TDP
AMD Ryzen™ 9 8945HS	**8 / 16**	**Up to 5.2GHz / 4.0GHz**	**12**	**Yes**	**45W**
AMD Ryzen™ 7 8845HS	8 / 16	Up to 5.1GHz / 3.8GHz	12	Yes	45W
AMD Ryzen™ 7 8840U	8 / 16	Up to 5.1GHz / 3.3GHz	12	Yes	28W
AMD Ryzen™ 7 8840HS	8 / 16	Up to 5.1GHz / 3.3GHz	12	Yes	28W
AMD Ryzen™ 5 8645HS	6 / 12	Up to 5.0GHz / 4.3GHz	8	Yes	45W
AMD Ryzen™ 5 8640U	6 / 12	Up to 4.9GHz / 3.5GHz	8	Yes	28W
AMD Ryzen™ 5 8640HS	6 / 12	Up to 4.9GHz / 3.5GHz	8	Yes	28W
AMD Ryzen™ 5 8540U	6 / 12	Up to 4.9GHz / 3.2GHz	4	No	28W
AMD Ryzen™ 3 8440U	4 / 8	Up to 4.7GHz / 3.0GHz	4	No	28W

“RAPHAEL AM5” AMD RYZEN™ 7000 SERIES PROCESSORS

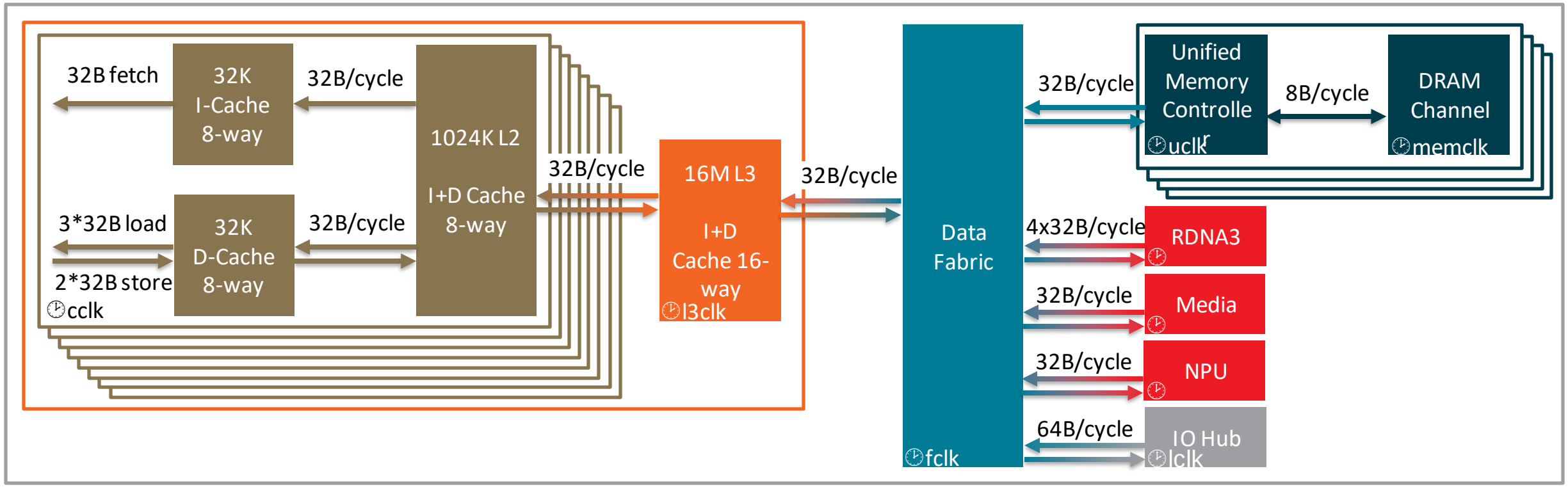
Desktop Model	Cores / Threads	Boost / Base Frequency	GPU Compute Units	AMD Ryzen™ AI	TDP
AMD Ryzen™ 9 7950X3D	16 / 32	Up to 5.7GHz / 4.2GHz	2	No	120W
AMD Ryzen™ 9 7950X	**16 / 32**	**Up to 5.7GHz / 4.5GHz**	**2**	**No**	**170W**
AMD Ryzen™ 9 7900X3D	12 / 24	Up to 5.6GHz / 4.4GHz	2	No	120W
AMD Ryzen™ 9 7900X	12 / 24	Up to 5.6GHz / 4.7GHz	2	No	170W
AMD Ryzen™ 9 7900	12 / 24	Up to 5.4GHz / 3.7GHz	2	No	65W
AMD Ryzen™ 7 7800X3D	**8 / 16**	**Up to 5.0GHz / 4.2GHz**	**2**	**No**	**120W**
AMD Ryzen™ 7 7700X	8 / 16	Up to 5.4GHz / 4.5GHz	2	No	105W
AMD Ryzen™ 7 7700	8 / 16	Up to 5.3GHz / 3.8GHz	2	No	65W
AMD Ryzen™ 5 7600X	6 / 12	Up to 5.3GHz / 4.7GHz	2	No	105W
AMD Ryzen™ 5 7600	6 / 12	Up to 5.1GHz / 3.8GHz	2	No	65W
AMD Ryzen™ 5 7500F	6 / 12	Up to 5.0GHz / 3.7GHz	0	No	65W

“STORM PEAK” AMD THREADRIPPER™ PRO 7000 WX-SERIES PROCESSORS

Workstation Model	Cores / Threads	Boost / Base Frequency	GPU Compute Units	AMD Ryzen™ AI	TDP
AMD Ryzen™ Threadripper™ PRO 7995WX	**96 / 192**	**Up to 5.1GHz / 2.5GHz**	**0**	**No**	**350W**
AMD Ryzen™ Threadripper™ PRO 7985WX	64 / 128	Up to 5.1GHz / 3.2GHz	0	No	350W
AMD Ryzen™ Threadripper™ PRO 7975WX	32 / 64	Up to 5.3GHz / 4.0GHz	0	No	350W
AMD Ryzen™ Threadripper™ PRO 7965WX	24 / 48	Up to 5.3GHz / 4.2GHz	0	No	350W
AMD Ryzen™ Threadripper™ PRO 7955WX	16 / 32	Up to 5.3GHz / 4.5GHz	0	No	350W
AMD Ryzen™ Threadripper™ PRO 7945WX	12 / 24	Up to 5.3GHz / 4.7GHz	0	No	350W

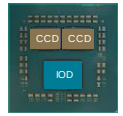
DATAFLOW

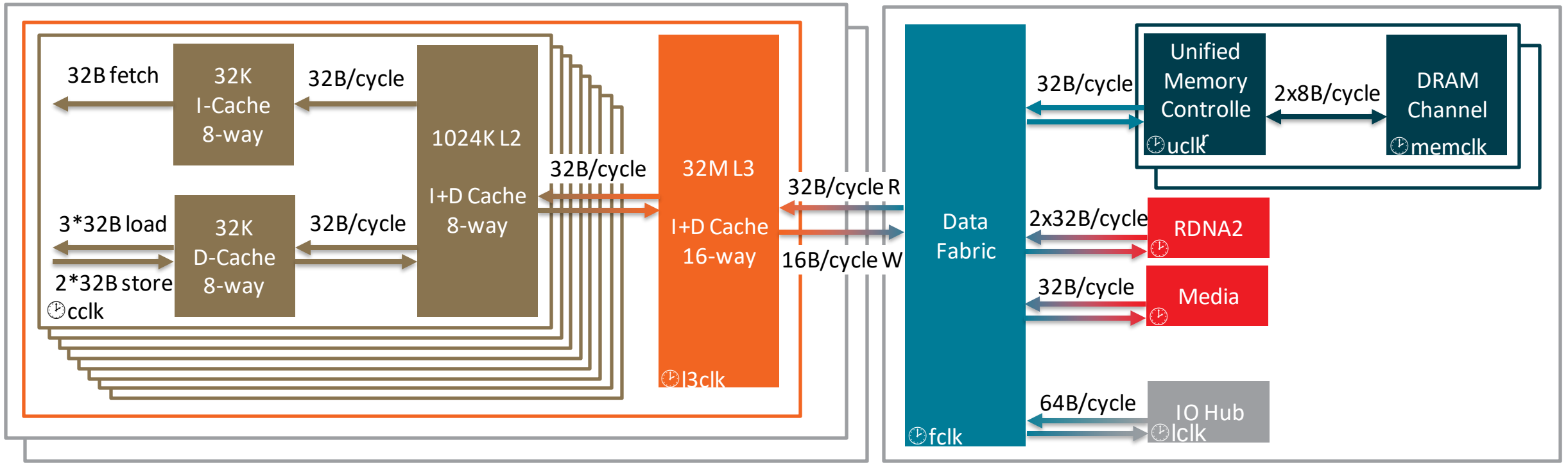
“HAWK POINT”



- AMD Ryzen™ 9 8945HS, 35-54W TDP, 8 cores, 16 threads, up to 5.2 GHz max boost clock, 4.0 GHz base clock with 2 channels of DDR5 memory.
- integrated AMD RDNA™ 3 graphics and Neural Processing Unit (NPU).

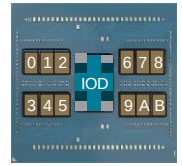
“RAPHAEL AM5”

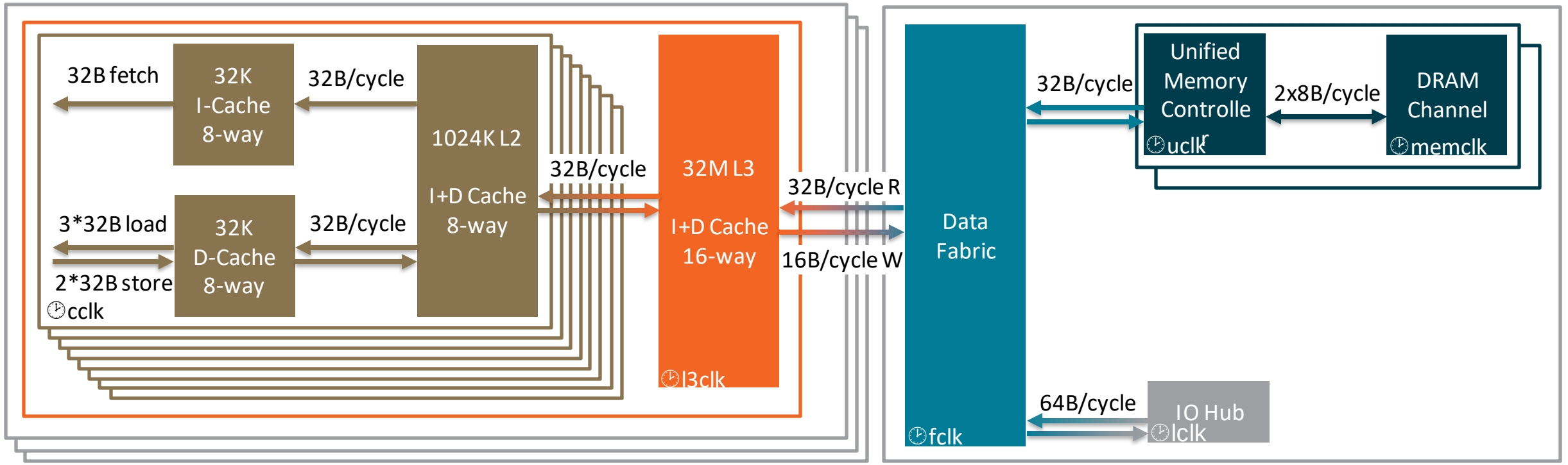




- AMD Ryzen™ 9 7950X, 170W TDP, 16 cores, 32 threads, up to 5.7 GHz max boost clock, 4.5 GHz base clock with 2 channels of DDR5 memory.
- Two Core Complex Die (CCD). Each CCD has one 32M L3 cache.

“STORM PEAK”

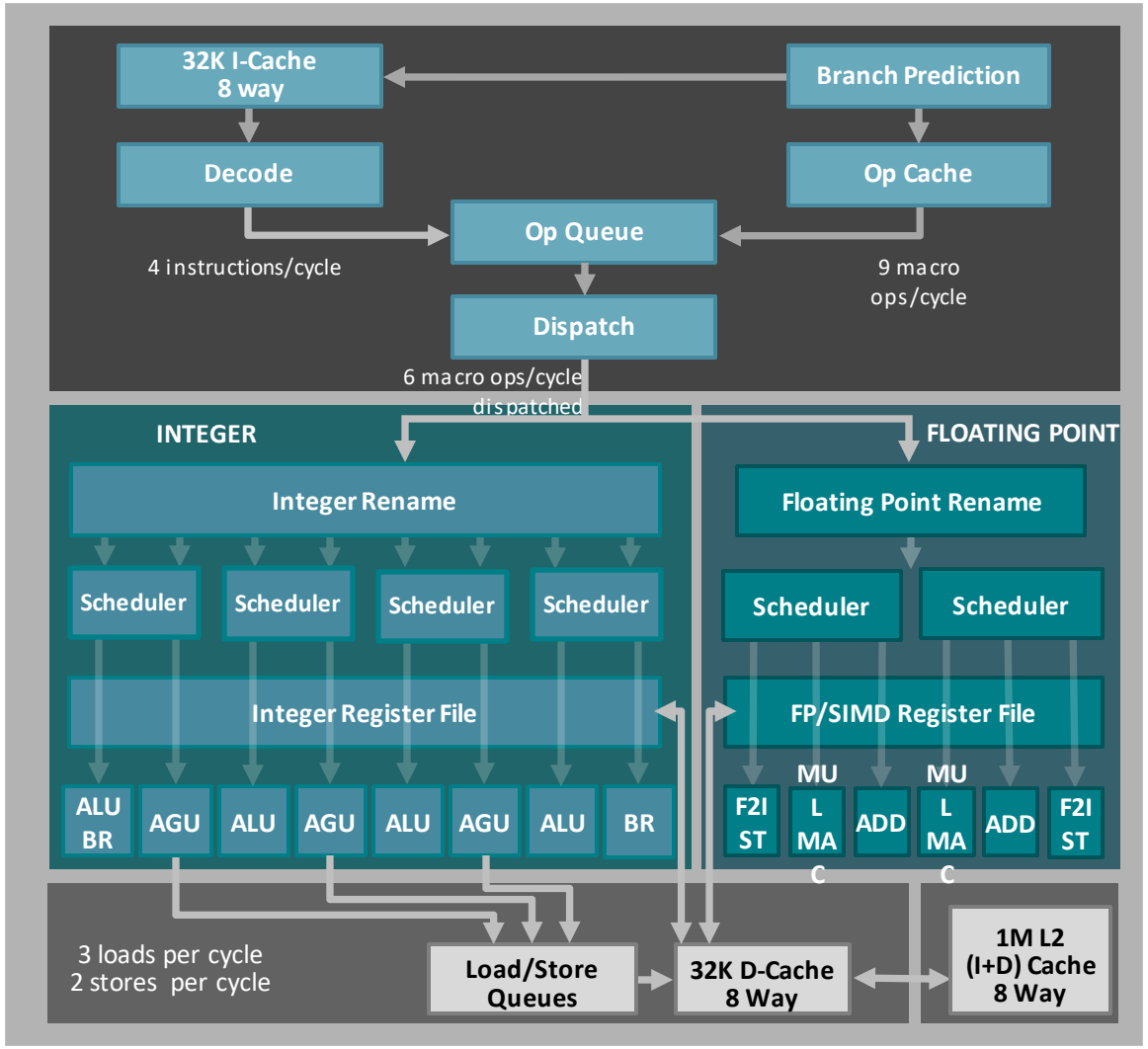




- AMD Ryzen™ Threadripper™ Pro 7995WX, 350W TDP, 96 cores, 192 threads, up to 5.1 GHz boost, 2.5 GHz base with 8 channels of DDR5 memory.
- Three CCDs per Data Fabric quadrant shown.

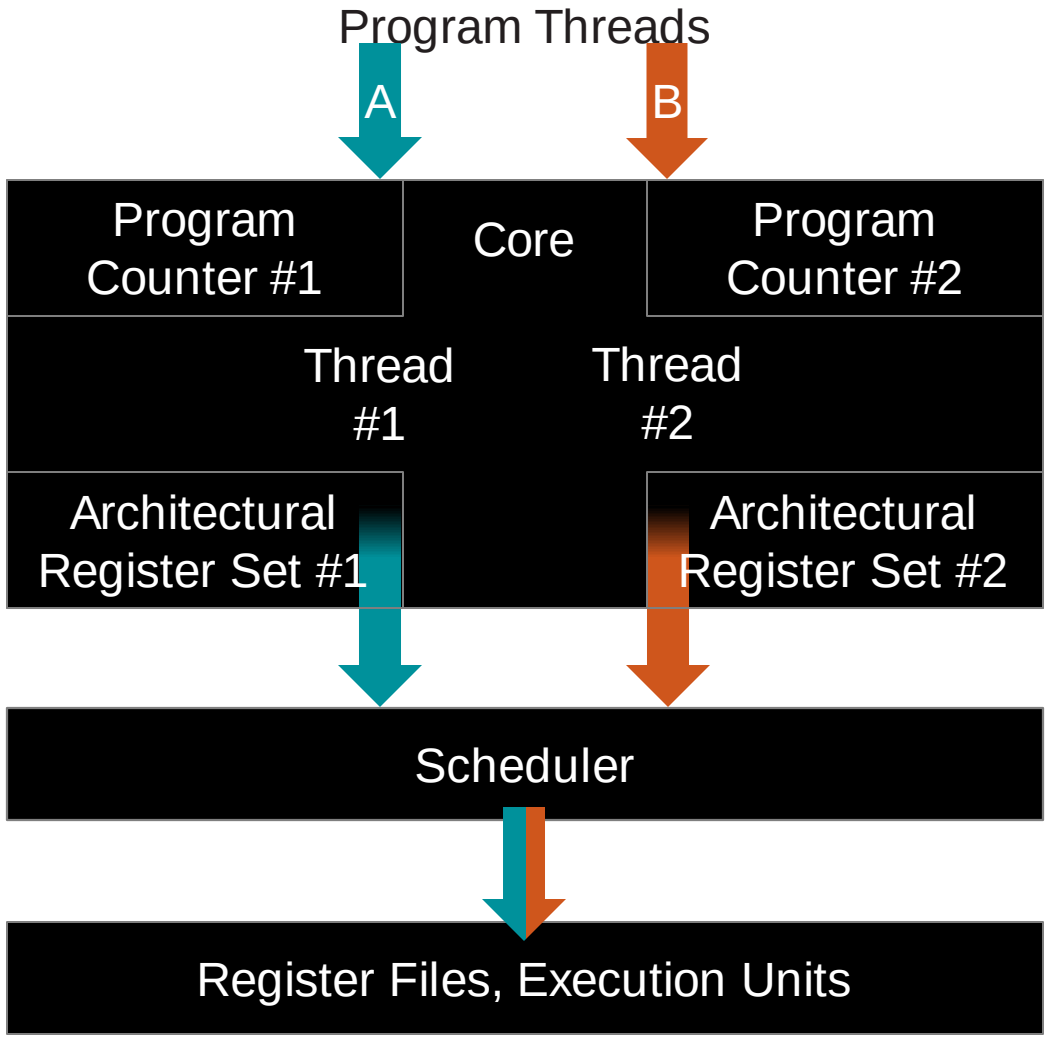
MICROARCHITECTURE

AMD “ZEN 4”



- ~13% higher IPC for desktop.
- Increased op cache from 4K to 6.75K ops.
- Increased L2 cache from 512 KB to 1024 KB.
- Improved load store.
- Improved branch prediction.
- Added AVX-512 instruction support.

SIMULTANEOUS MULTI-THREADING



- Single-threaded applications do not always occupy all resources of the processor.
- The processor can take advantage of the unused resources to execute a second thread concurrently.
- Although each thread has a program counter and architectural register set, core resources may be shared while operating in two-threaded mode.

CORE RESOURCE SHARING DEFINITIONS

Category	Definition
Competitively shared	Resource entries are assigned on demand. A thread may use all resource entries.
Watermarked	Resource entries are assigned on demand. When in two-threaded mode a thread may not use more resource entries than are specified by a watermark threshold.
Statically partitioned	Resource entries are partitioned when entering two-threaded mode. A thread may not use more resource entries than are available in its partition.

AMD “ZEN 4” CORE RESOURCE SHARING

Resource	Competitively Shared	Watermarked	Statically Partitioned
Integer Scheduler		X	
Integer Register File		X	
Load Queue		X	
Floating Point Physical Register		X	
Floating Point Scheduler		X	
Memory Request Buffers		X	
Op Queue			X
Store Queue			X
Write Combining Buffer		X	
Retire Queue			X

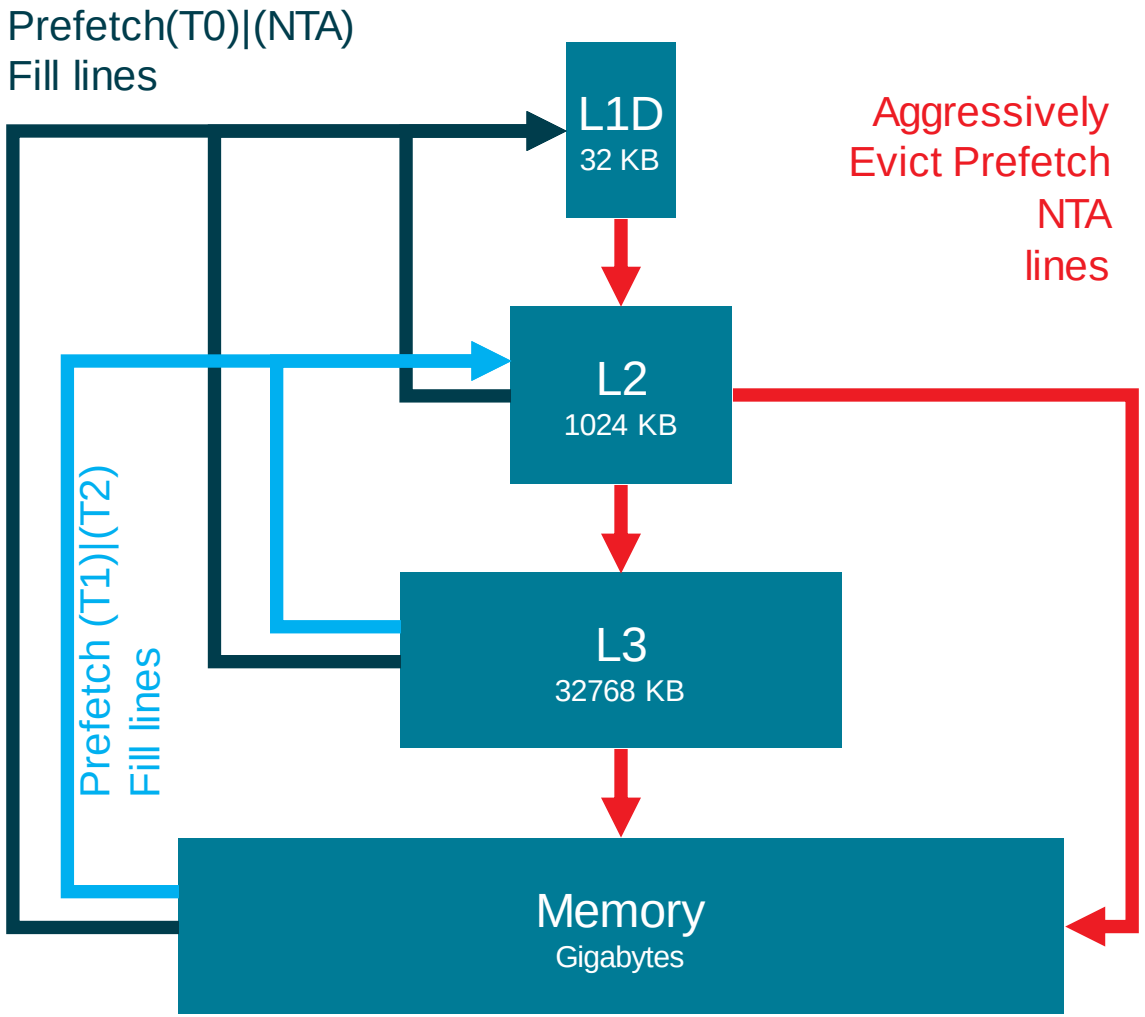
INSTRUCTION SET EVOLUTION

Core	AVX512*	GFNI	VAES	VPCLMUL	CLWB	ADX	CLFLUSHOPT	RDSEED	SHA	XGETBV	XSAVEC	XSAVES	AVX2	BMI2	MOVBE	RDRND	FSGSBASE	XSAVEOPT	BMI	FMA	F16C	AES	AVX	OSXSAVE	PCLMUL	SSE4.1	SSE4.2	XSAVE	SSSE3	MONITORX	CLZERO
AMD “Zen 4”	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AMD “Zen 3”	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AMD “Zen 2”	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AMD “Zen 1”	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
“Jaguar”	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1	1	1	1	1	0	0

AVX512 INSTRUCTION SET EVOLUTION

Core	AVX512_BF16	AVX512_VPOPCNTDQ	AVX512_BITALG	AVX512_VNNI	AVX512_VBMI2	AVX512_VBMI	AVX512VL	AVX512BW	AVX512CD	AVX512_IFMA	AVX512DQ	AVX512F
AMD “Zen 4”	1	1	1	1	1	1	1	1	1	1	1	1
AMD “Zen 3”	0	0	0	0	0	0	0	0	0	0	0	0
AMD “Zen 2”	0	0	0	0	0	0	0	0	0	0	0	0
AMD “Zen 1”	0	0	0	0	0	0	0	0	0	0	0	0
“Jaguar”	0	0	0	0	0	0	0	0	0	0	0	0

SOFTWARE PREFETCH INSTRUCTIONS



- Use Software Prefetch instructions on linked data structures experiencing cache misses.
- Use NTA on use once data.
- While in two-threaded mode, beware too many software prefetches may evict the working set of the other thread from their shared caches.
- Prefetch(T0)|(NTA) fills into L1.
- Prefetch(T1)|(T2) fills into L2.
  - **new for AMD “Zen 4”!**

HARDWARE PREFETCHERS L1

Category	Definition
L1 Stream	Uses history of memory access patterns to fetch additional sequential lines in ascending or descending order.
L1 Stride	Uses memory access history of individual instructions to fetch additional lines when each access is a constant distance from the previous.
L1 Region	Uses memory access history to fetch additional lines when the data access for a given instruction tends to be followed by a consistent pattern of other accesses within a localized region.

HARDWARE PREFETCHERS L2

Category	Definition
L2 Stream	Uses history of memory access patterns to fetch additional sequential lines in ascending or descending order.
L2 Up/Down	Uses memory access history to determine whether to fetch the next or previous line for all memory accesses.

STREAMING HARDWARE PREFETCHER

- Uses history of memory access patterns to fetch additional sequential lines in ascending or descending order.

Memory Address	0	40	80	C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0	400	440	480	4C0	500	540	580	5C0	600	640	680	6C0	700	740	780	7C0	800
Steam +1																								1	2	3	4	5	6	7	8	9	10

```
alignas(64) float a[LEN];
// …
float sum = 0.0f;
for (size_t i = 0; i < LEN; i++) {
    sum += a[i]; // streaming prefetch
}
```

STRIDE HARDWARE PREFETCHER

- Uses memory access history of individual instructions to fetch additional lines when each access is a constant distance from the previous.

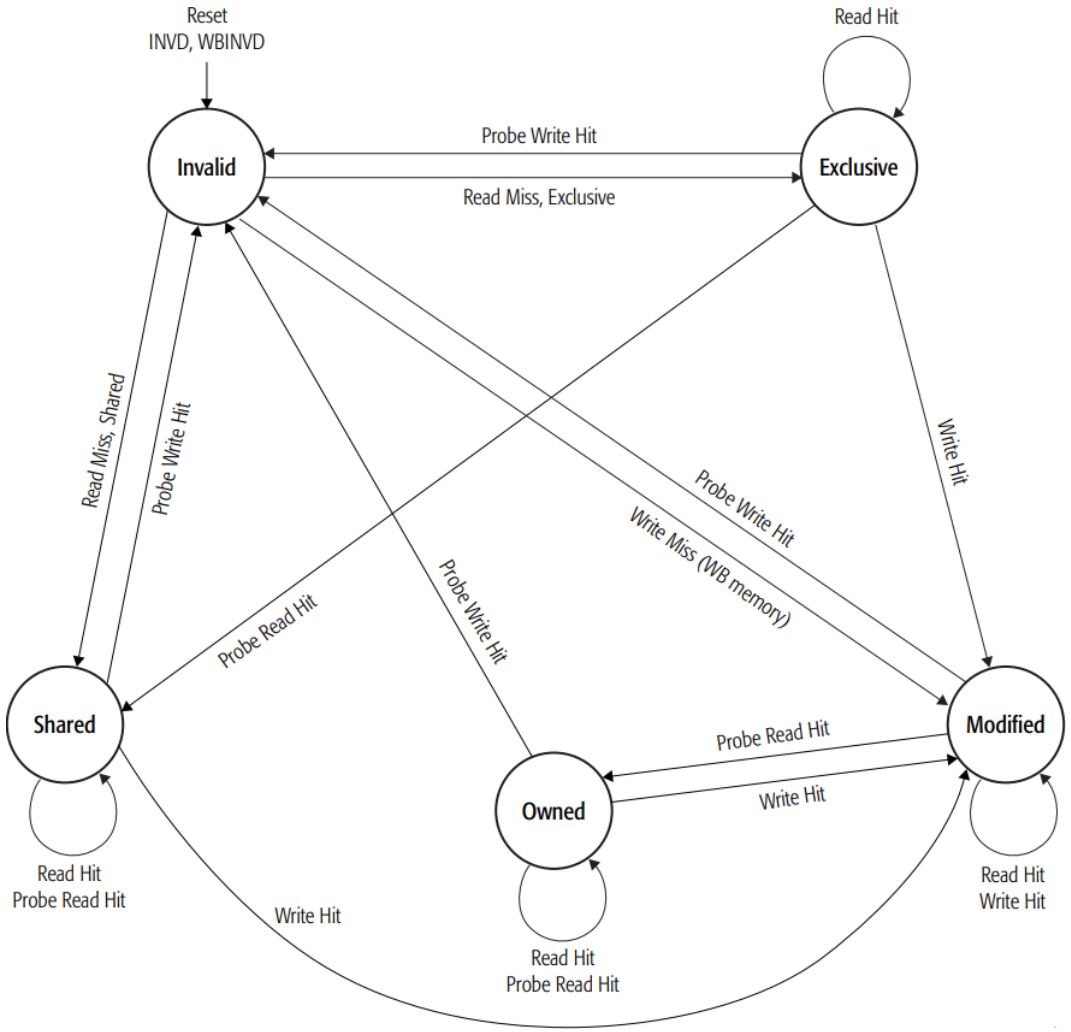
Memory Address	0	40	80	C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0	400	440	480	4C0	500	540	580	5C0	600	640	680	6C0	700	740	780	7C0	800
Stride +5													1					2					3					4					5
Stride +5																	1					2					3				4	

```
struct S { double x1, y1, z1, w1; char name[256]; double x2, y2, z2, w2; };
alignas(64) S a[LEN];
// …
double sumX1 = 0.0f, sumX2 = 0.0f;
for (size_t i = 0; i < LEN; i++) {
    sumX1 += a[i].x1; // stride prefetch 0
    sumX2 += a[i].x2; // stride prefetch 1
}
```

DESKTOP CACHE HIERARCHY EVOLUTION

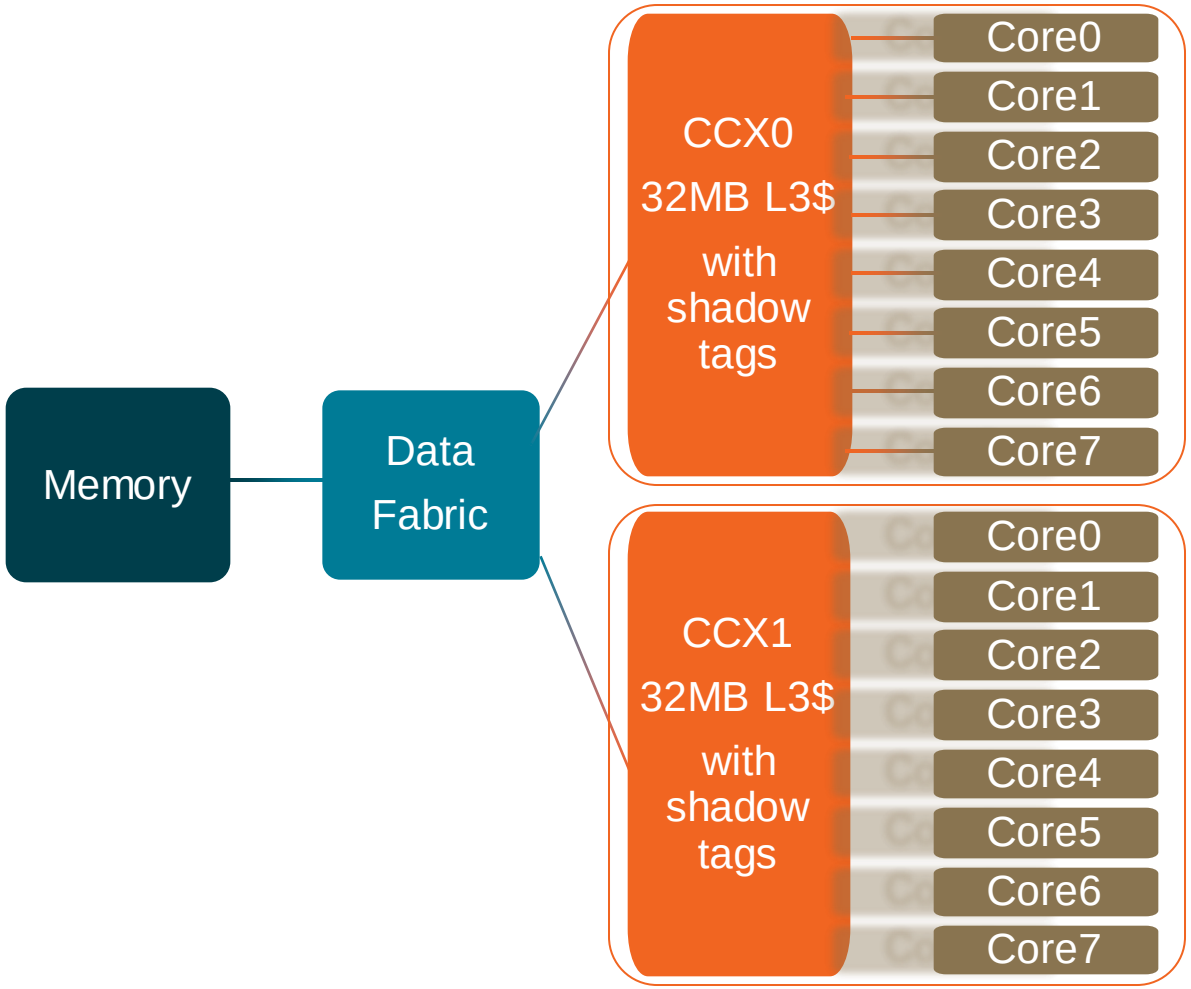
Core	uOP/Core K	L1I/Core KB	L1D/Core KB	L2/Core KB	L3/CCX MB
AMD “Zen 4”	6.75	32	32	1024	32*
AMD “Zen 3”	4	32	32	512	32*
AMD “Zen 2”	4	32	32	512	16
AMD “Zen 1”	2	64	32	512	8

CACHE-COHERENCY PROTOCOL



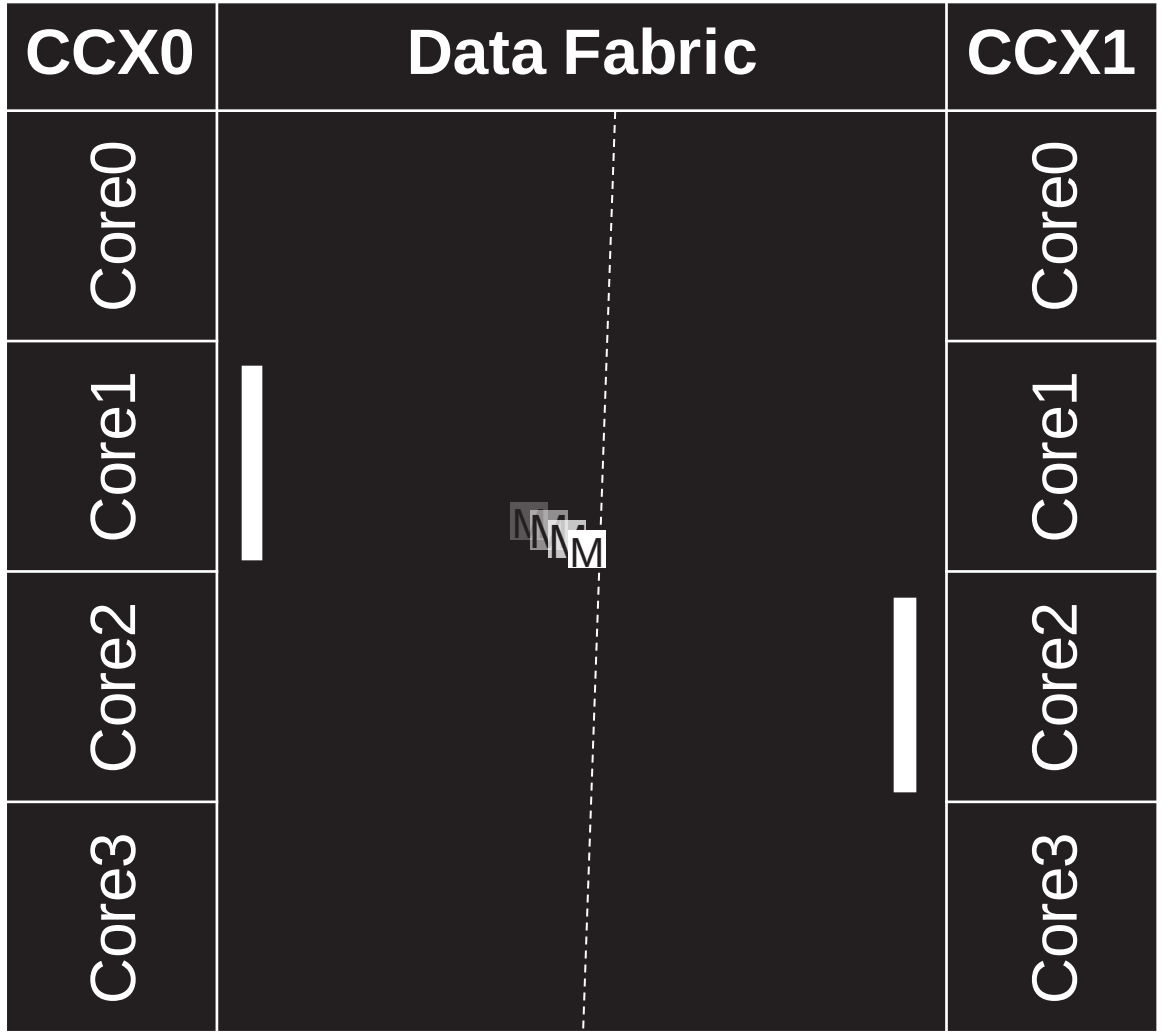
- The AMD cache-coherency protocol is MOESI (Modified, Owned, Exclusive, Shared, Invalid).
- Instruction-execution activity and external-bus transactions may change the cache's MOESI state.
- Read hits do not cause a MOESI-state change.
- **Write hits generally cause a MOESI-state change into the modified state.**
- If the cache line is already in the modified state, a write hit does not change its state.

CACHE-TO-CACHE TRANSFERS



- Each CPU Complex (CCX) has a L3 cache shared by up to eight cores.
- The L3 cache has shadow tags for each L2 cache within its complex.
- Shadow tags determine if a “fast” cache-to-cache transfer between cores within the CCX is possible.
- **Cache-coherency probe latency responses may be slower from cores in another CCX.**

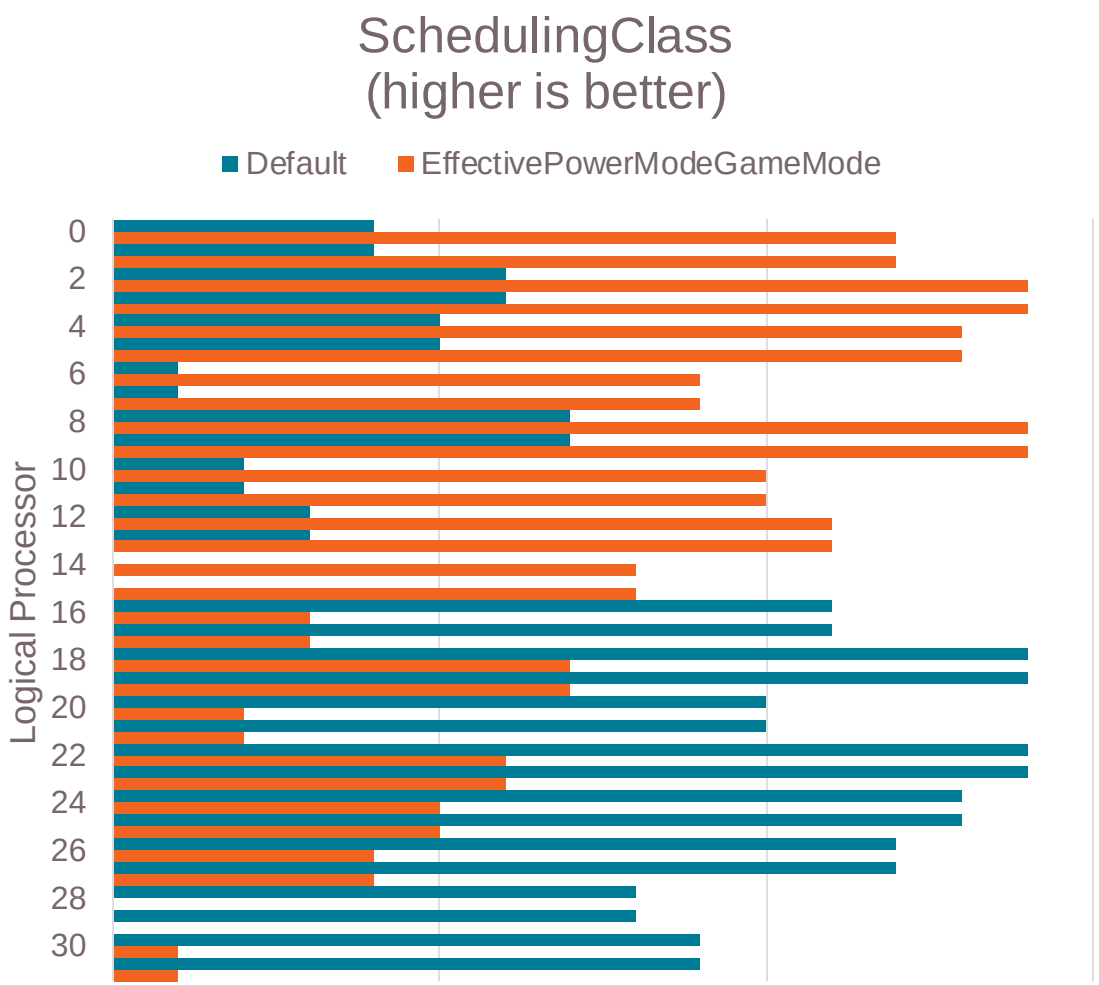
CACHE-COHERENCY EFFICIENCY



- Minimize ping-ponging modified cache lines between cores – especially in another CCX!

- Minimize using Read-Modify-Write instructions.
  - Use a single atomic add with a local sum rather than many atomic increment operations.
- Improve lock efficiency.
  - “Test and Test-and-Set” in user spin locks with a pause instruction.
  - Replace user spin locks with modern sync APIs.
- Use a memory allocator optimized for multi-threading.
  - Try mimalloc or jemalloc.

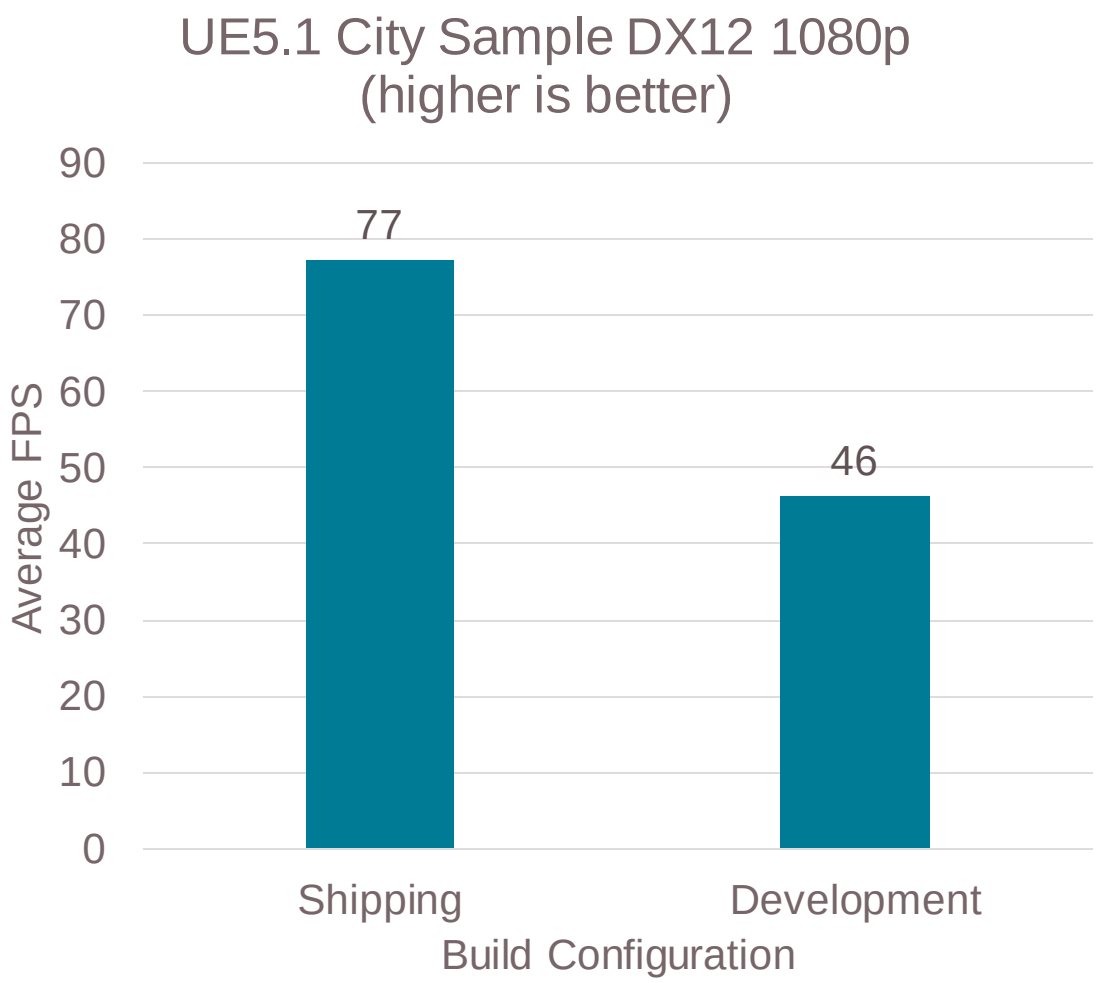
AMD “PREFERRED CORE”



- Some AMD products have cores that are faster than other cores.
- Windows® may use SchedulingClass or EfficiencyClass during thread scheduling. These values may change during runtime.
- **Thread affinity masks may interfere with thread scheduling and power management optimizations on Windows PCs.**
- Testing done by AMD performance labs January 22, 2023 on an AMD reference motherboard equipped with 16GB DDR5-6000MHz, Ryzen™ 9 7950X3D with Nvidia RTX 4090, Win11 Pro x64 22621.1105. Actual results may vary.

BEST PRACTICES

PREFER SHIPPING CONFIGURATION BUILDS FOR CPU PROFILING



- Disable debug features before you ship!
- Debug and development builds may reduce performance.
  - Stats collection may cause cache pollution.
  - Logging may create serialization points.
  - Debug builds may disable multi-threading optimizations.

- *Performance of UE4.5.1 binaries compiled with Microsoft® Visual Studio 2022 v17.4.4.*
- *Testing done by AMD technology labs, January 30, 2023 on the following system. Test configuration: AMD Ryzen™ Threadripper™ PRO 5995WX, Cooler Master MasterLiquid ML360 RGB TR4 Edition, 256GB (8 x 32GB 2R RDDR4-3200 at 24-22-22-52) memory, AMD Radeon™ RX 7900 XTX GPU with driver 23.1.1 (January 11, 2023), 2TB M.2 NVME SSD, AMD Reference Motherboard, Windows® 11 version 22H2, 1920x1080 resolution. Actual results may vary.*

DISABLE ANTI-TAMPER WHILE CPU PROFILING

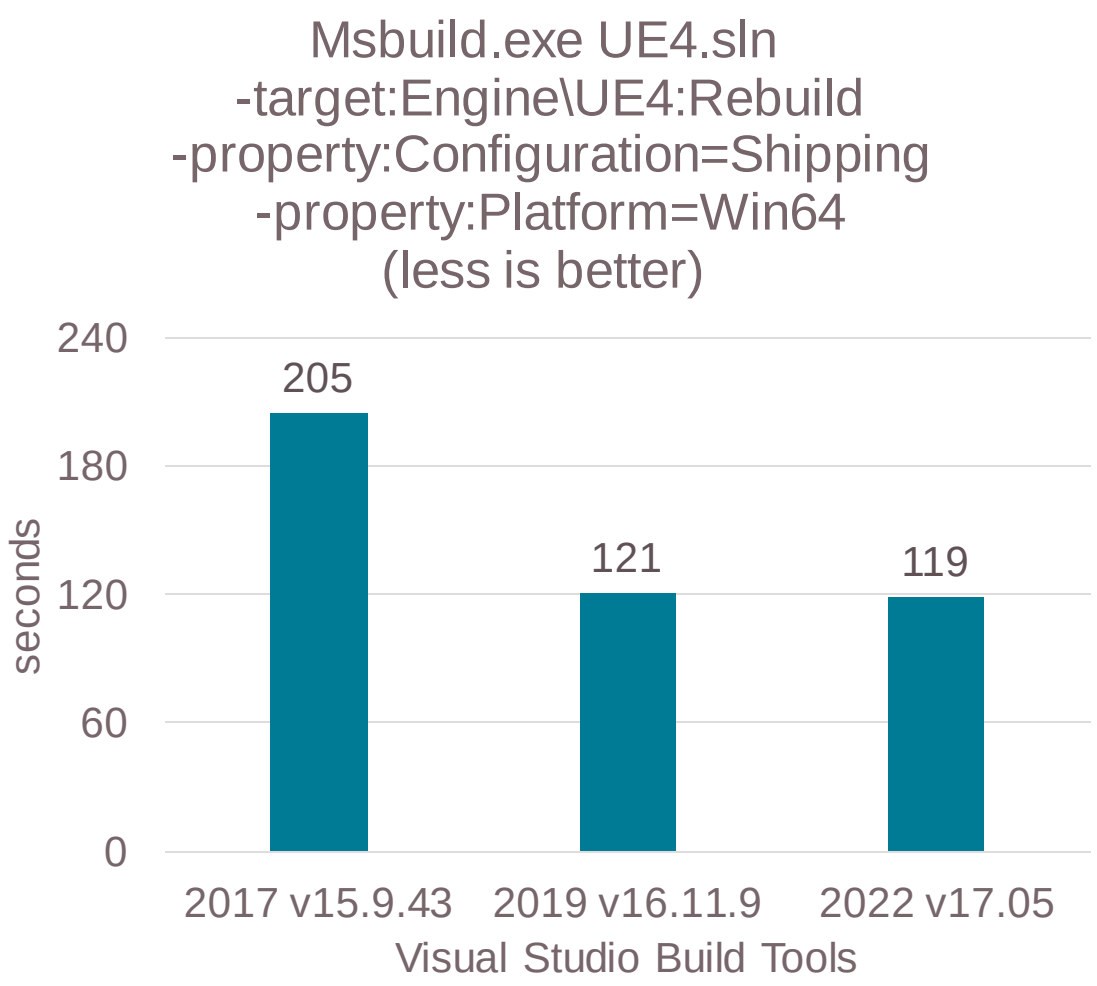
- Anti-tamper and Anti-Cheat technologies may prevent CPU debugging and profiling tools from working correctly – especially while loading and retrieving symbol information.
- Create a CPU profiling friendly build configuration similar-to the Shipping configuration but with Anti-Tamper and Anti-Cheat technologies disabled.
- Add this build as a launch option during development.
- Remove this build before release.

TEST COLD SHADER CACHE FIRST TIME USER EXPERIENCE

```
rem Run as administrator
rem Disable Steam shader pre-caching before running this script
rem Reboot after running this script to clear any shaders still in system memory

setlocal enableextensions
cd /d "%~dp0"
rmdir /s /q "%LOCALAPPDATA%\D3DSCache"
rmdir /s /q "%LOCALAPPDATA%\AMD\DX9Cache"
rmdir /s /q "%LOCALAPPDATA%\AMD\DxCache"
rmdir /s /q "%LOCALAPPDATA%\AMD\DxcCache"
rmdir /s /q "%LOCALAPPDATA%\AMD\OglCache"
rmdir /s /q "%LOCALAPPDATA%\AMD\VkCache"
rmdir /s /q "%LOCALAPPDATA%\NVIDIA\DXCache"
rmdir /s /q "%ProgramFiles(x86)%\Steam\steamapps\shadercache"
```

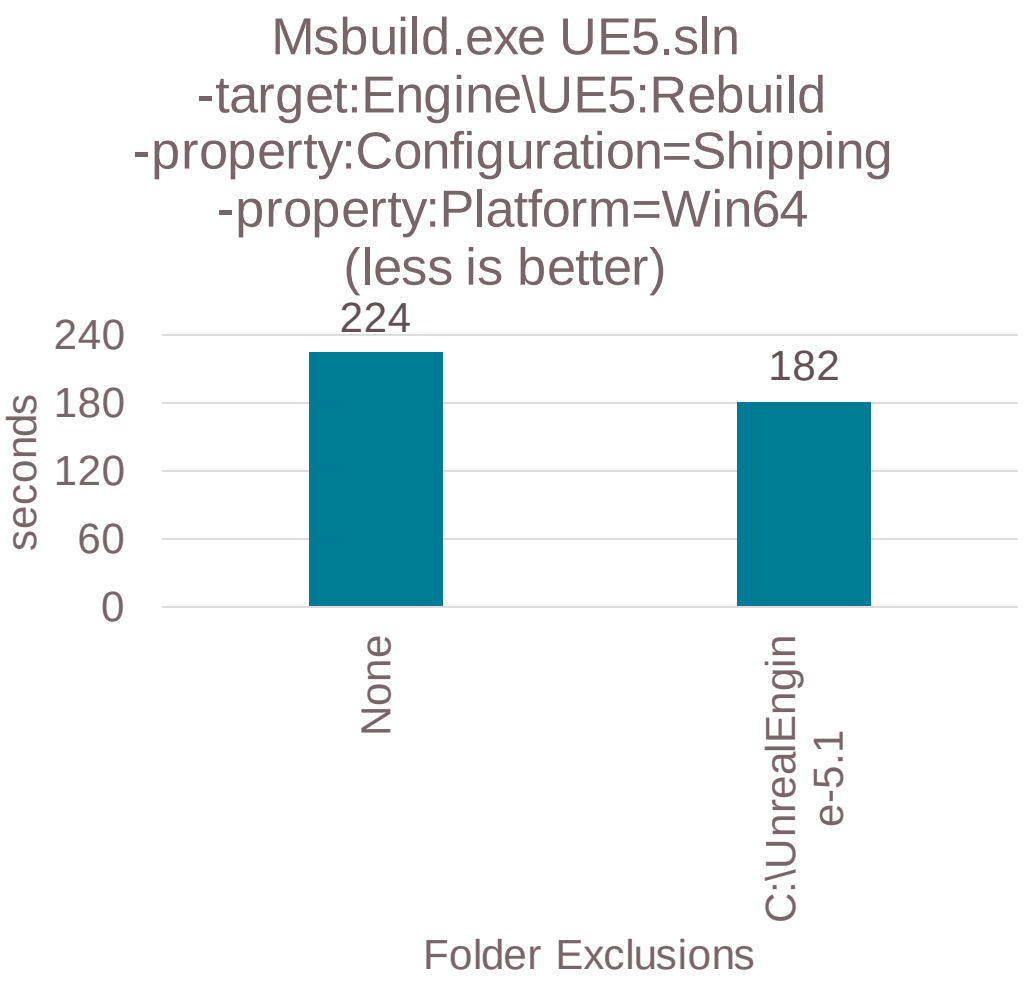
USE THE LATEST COMPILER AND WINDOWS® SDK



- Get the latest build and link time improvements.
- Get the latest library and runtime optimizations.

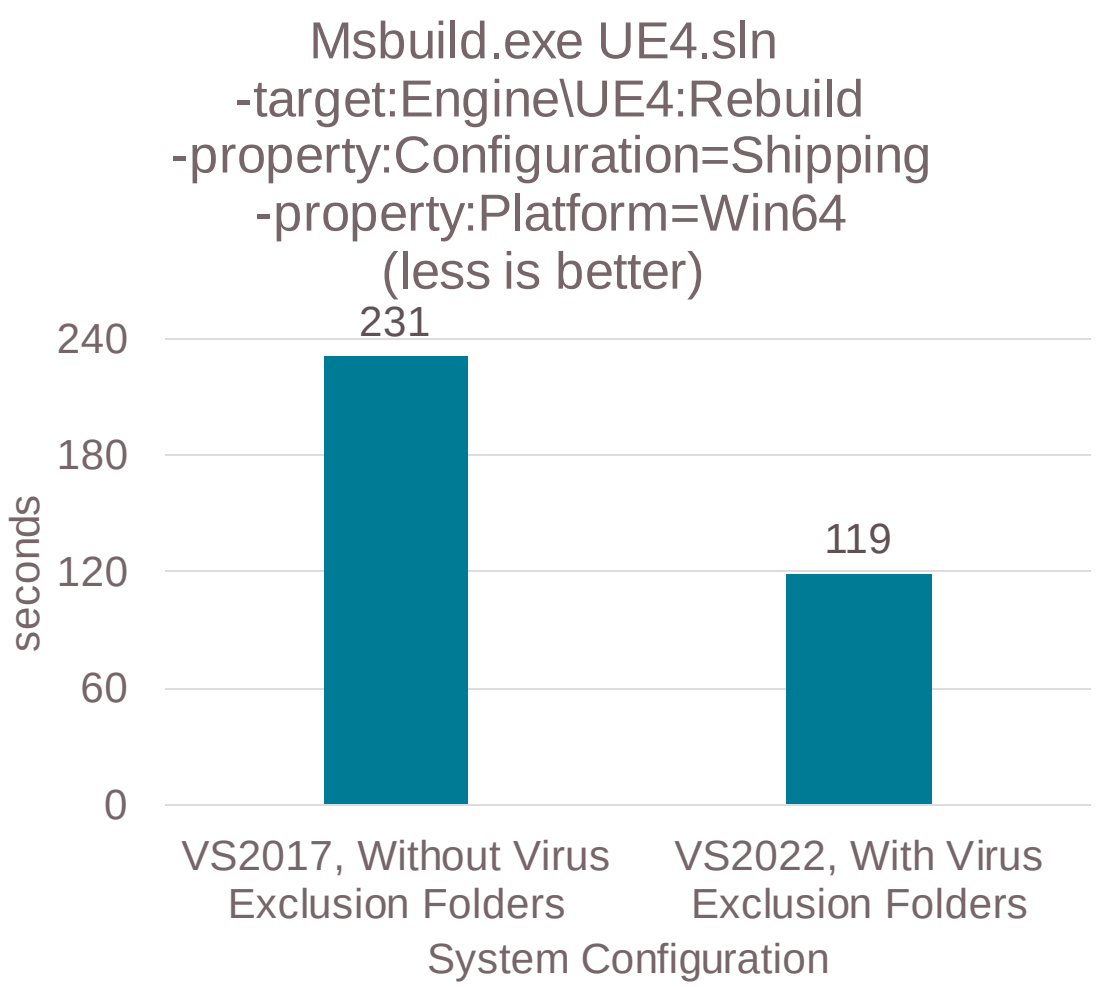
- *Performance of UE4.27.2 binaries compiled with Microsoft® Visual Studio.*
- *Testing done by AMD technology labs, February 5, 2022 on the following system. Test configuration: AMD Ryzen™ Threadripper™ PRO 5995WX, Enermax LIQTECH TR4 II series 360mm liquid cooler, 256GB (8 x 32GB 2R RDDR4-3200 at 24-22-22-52) memory, AMD Radeon™ RX 6800 XT GPU with driver 21.10.2 (October 25, 2021), 2TB M.2 NVME SSD, AMD Reference Motherboard, Windows® 11 x64 version 21H2, 1920x1080 resolution. Actual results may vary.*

ADD VIRUS AND THREAT PROTECTION EXCLUSIONS



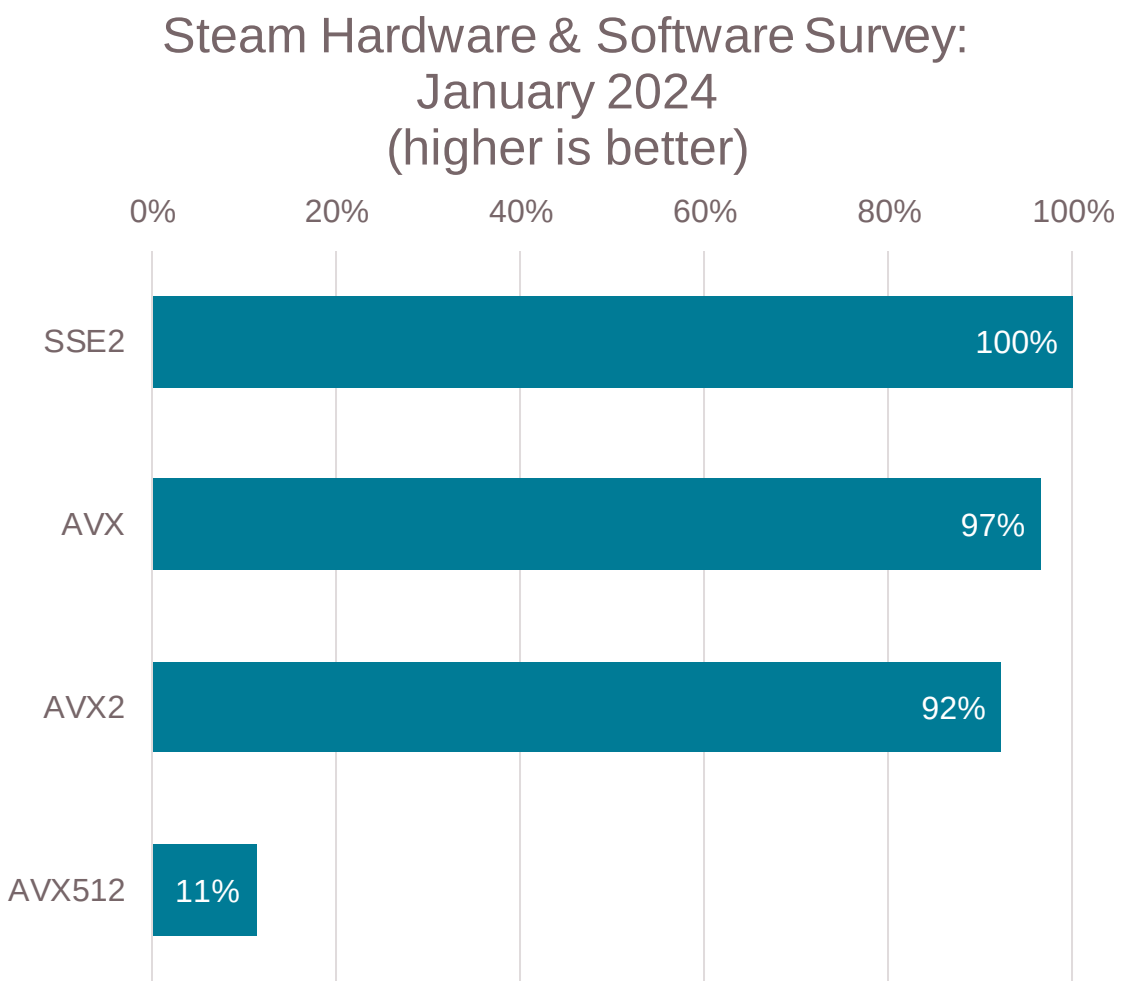
- WARNING: Not recommended for CI/CD systems. Exclusions may make your device vulnerable to threats.
- Add project folders to virus and threat protection settings exclusions for faster build times.
- Faster rebuild time after optimization!
- *Performance of UE5.1 binaries compiled with Microsoft® Visual Studio 2022 v17.4.4.*
- *Testing done by AMD technology labs, January 28, 2023 on the following system. Test configuration: AMD Ryzen™ Threadripper™ PRO 5995WX, Cooler Master MasterLiquid ML360 RGB TR4 Edition, 256GB (8 x 32GB 2R RDDR4-3200 at 24-22-22-52) memory, AMD Radeon™ RX 7900 XTX GPU with driver 23.1.1 (January 11, 2023), 2TB M.2 NVME SSD, AMD Reference Motherboard, Windows® 11 version 22H2, 1920x1080 resolution. Actual results may vary.*

REDUCE BUILD TIMES



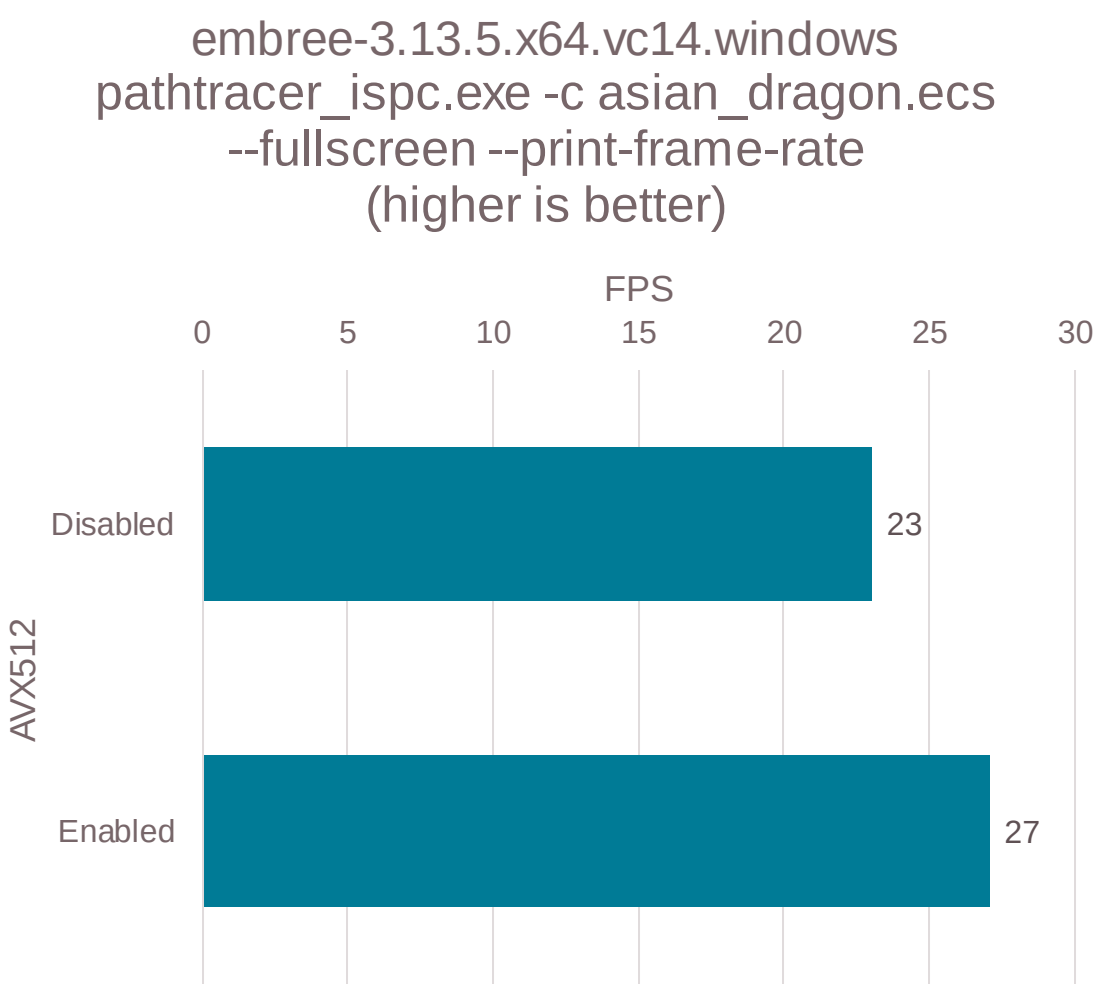
- *Performance of UE4.27.2 binaries compiled with Microsoft® Visual Studio.*
- *Testing done by AMD technology labs, February 5, 2022 on the following system. Test configuration: AMD Ryzen™ Threadripper™ PRO 5995WX, Enermax LIQTECH TR4 II series 360mm liquid cooler, 256GB (8 x 32GB 2R RDDR4-3200 at 24-22-22-52) memory, AMD Radeon™ RX 6800 XT GPU with driver 21.10.2 (October 25, 2021), 2TB M.2 NVME SSD, AMD Reference Motherboard, Windows® 11 x64 version 21H2, 1920x1080 resolution. Actual results may vary.*

USE AVX OR AVX2 IF CPU MINIMUM REQUIREMENTS ALLOW



- A binary may have better code generation using AVX or later ISA by using the Microsoft® Visual C compiler option /arch:[AVX|AVX2|AVX512].
- Minimum hardware requirements:
  - Windows 10 = SSE2
  - Windows 11 = SSE4.1
- The Windows 10 supported processor list includes AMD products which support AVX but not AVX2.
- The Windows 10 supported processor list may include products from other CPU vendors which do not support AVX.

ENABLE AVX512 IN DEVELOPMENT TOOLS



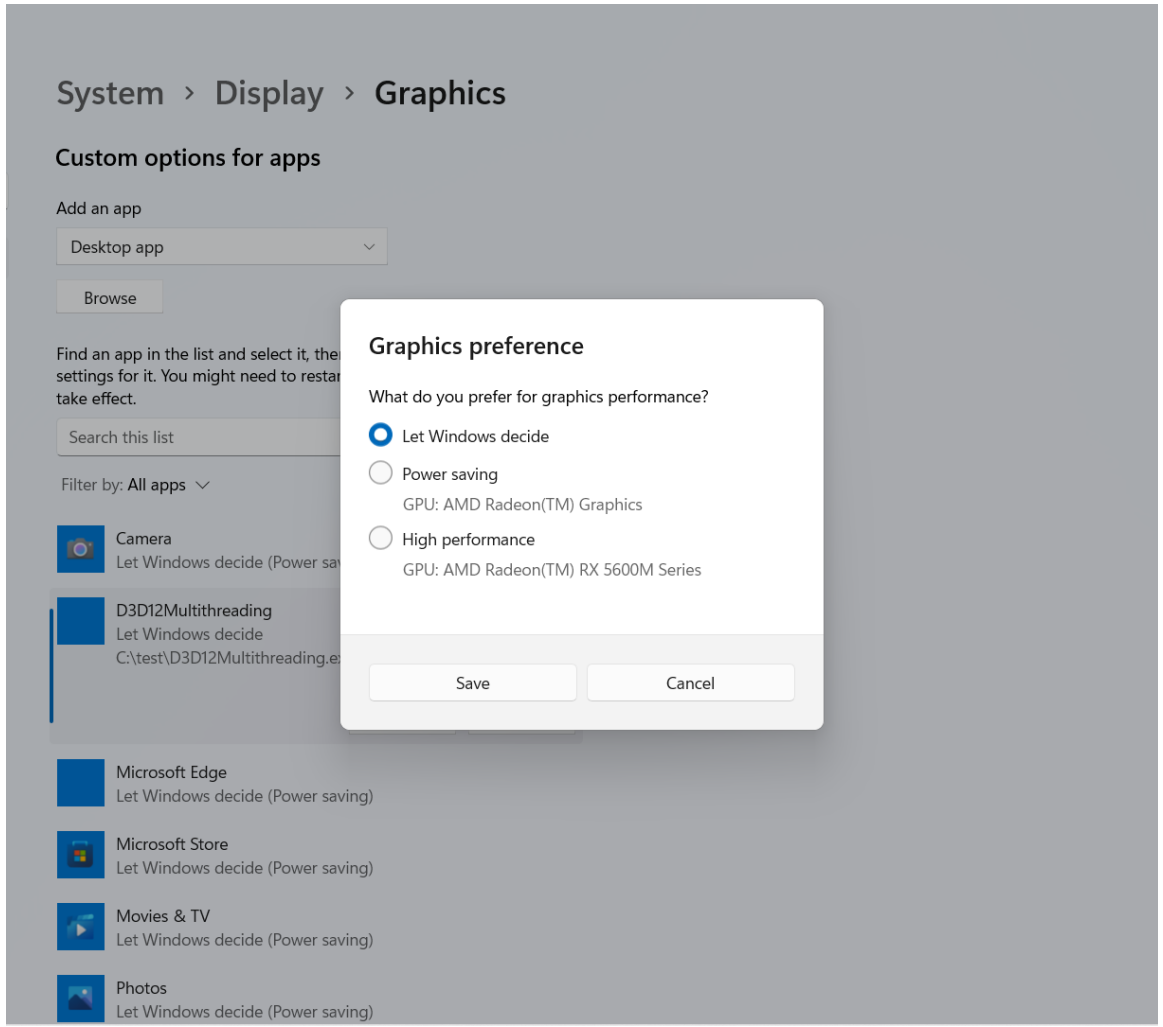
- Development tools may benefit from AVX512.
- Examples:
  - Light Baking.
  - Texture Compression.
  - Mesh to Signed Distance Fields.

- *Testing done by AMD technology labs, January 29, 2023 on the following system. Test configuration: AMD Ryzen™ 7950X, NZXT Kraken X62 cooler, 32GB (2 x 16GB DDR5-6000 30-38-38-96) memory, AMD Radeon™ RX 7900 XTX GPU with driver 23.1.1 (January 11, 2023), 2TB M.2 NVME SSD, AMD Reference Motherboard, Windows® 11 x64 build 22H2, 1920x1080 resolution. Actual results may vary.*

AUDIT CONTENT

- Ask artists to recommend profiling scenes of interest!
  - For example, an indoor dungeon, an outdoor city, an outdoor forest, large crowds, or a specific time of day.
- Run Unreal Engine MapCheck!
  - It may find some performance issues.
  - https://docs.unrealengine.com/en-US/BuildingWorlds/LevelEditor/MapErrors/index.html
- Use Unity AssetPostprocessor!
  - Enforce minimum standards.
  - https://docs.unity3d.com/Manual/BestPracticeUnderstandingPerformanceInUnity4.html
- Check stats before CPU profiling!
  - If a scene far exceeds its draw budget or has many duplicate objects, report the issue to its artists and consider profiling a different scene. Otherwise, you may risk profiling hot spots which may not be hot after the art issues are resolved.

SUPPORT HYBRID GRAPHICS



- Use IDXGIFactory6:: EnumAdapterByGpuPreference DXGI_GPU_PREFERENCE_HIGH_PERFORMANCE for game applications.
- The user may change preferences per application in Graphics settings.

- *Testing done by AMD performance labs January 24, 2022 on a Dell G5 15 SE laptop equipped with, 16GB DDR4-3200MHz, Ryzen™ 9 4900H with Radeon™ RX 5600M, Win11 Pro x64 22000.434.*

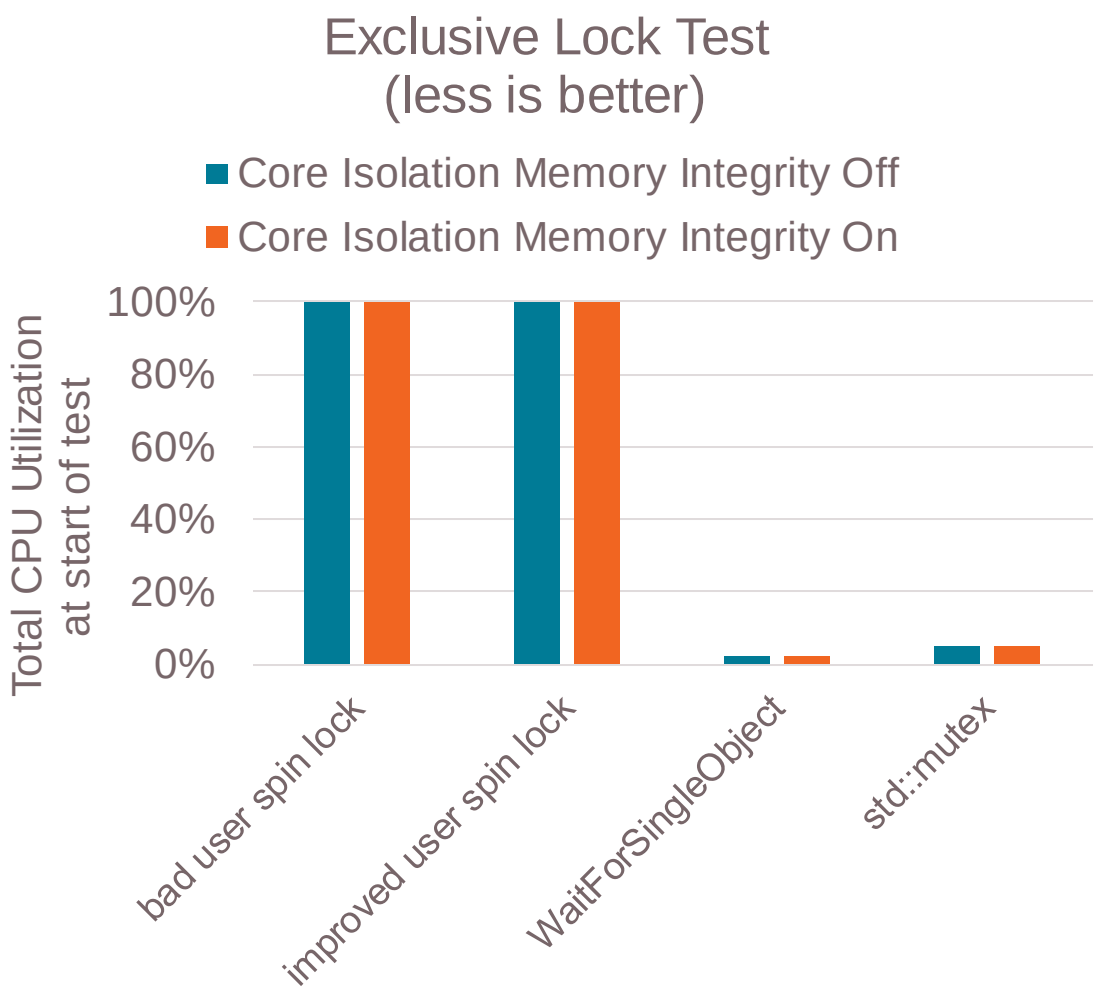
USE PREFERRED VIDEO AND AUDIO CODECS

- Prefer H264 video and AAC audio codecs as recommended by the Unreal Engine Electra Media Player.
- Hardware accelerated codecs may increase hours of battery life and reduce CPU work.
- AMD Radeon™ graphics devices released since 2022 no longer accelerate WMV3 decoding.
- See amd.com product specifications for supported rendering formats.

OPTIMIZATIONS

SYNC APIS

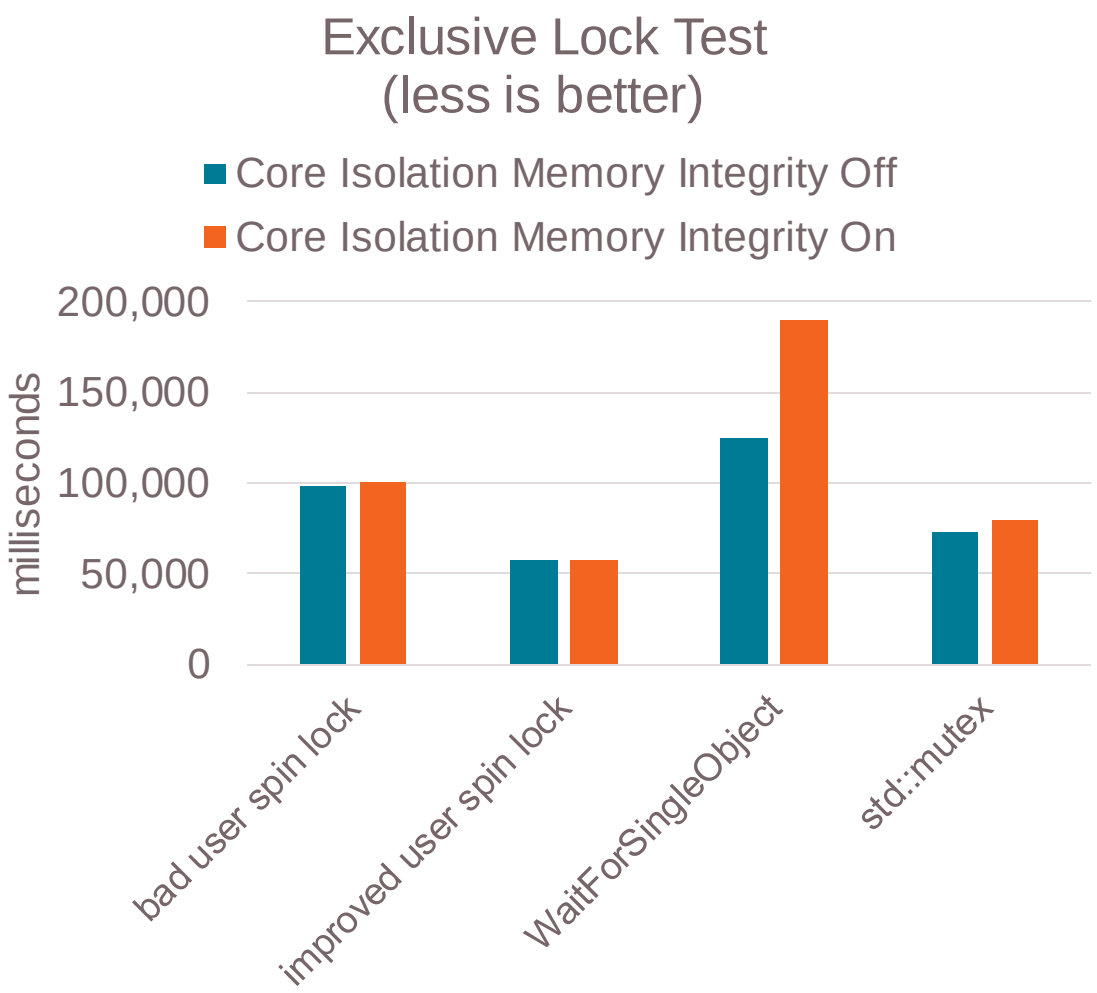
USE MODERN SYNC APIS



- Avoid user spin locks that starve payload work on other ready threads, consume excessive power, and drain laptop batteries.

- *Performance of binaries compiled with Microsoft® Visual Studio 2022 v17.8.6.*
- *Testing done by AMD technology labs, January 6, 2024 on the following system. Test configuration: AMD Ryzen™ Threadripper™ 7995WX 96-Cores, NZXT Kraken 360 cooler, 256GB (8 x 32GB RDDR5-4800 memory, AMD Radeon™ RX 580 GPU with 31.0.12027.9001 (March 20, 2023), 1TB M.2 NVME SSD, AMD Reference Motherboard, Windows® 11 x64 version 23H2, 1920x1080 resolution. Actual results may vary.*

USE MODERN SYNC APIS



- Prefer std::mutex which has good performance and low CPU utilization.
- Legacy sync APIs like WaitForSingleObject may rely on expensive syscall instructions.
- Modern sync APIs like std::mutex may rely on lower-cost mwaitx instructions that can execute at any privilege level.

- *Performance of binaries compiled with Microsoft® Visual Studio 2022 v17.8.6.*
- *Testing done by AMD technology labs, January 6, 2024 on the following system. Test configuration: AMD Ryzen™ Threadripper™ 7995WX 96-Cores, NZXT Kraken 360 cooler, 256GB (8 x 32GB RDDR5-4800 memory, AMD Radeon™ RX 580 GPU with 31.0.12027.9001 (March 20, 2023), 1TB M.2 NVME SSD, AMD Reference Motherboard, Windows® 11 x64 version 23H2, 1920x1080 resolution. Actual results may vary.*

USE MODERN SYNC APIS: SHARED CODE

```
#include "intrin.h"
#include <chrono>
#include <numeric>
#include <thread>
#include <vector>
#include <mutex>
#include <Windows.h>
#define LEN 128

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

```
int main(int argc, char* argv[]) {
    using namespace std::chrono;
    float b0 = (argc > 1) ? strtof(argv[1], NULL) : 1.0f;
    float c0 = (argc > 2) ? strtof(argv[2], NULL) : 2.0f;
    std::fill((float*)b, (float*)(b + LEN), b0);
    std::fill((float*)c, (float*)(c + LEN), c0);
    size_t num_threads = \
         GetMaximumProcessorCount(ALL_PROCESSOR_GROUPS);
    wprintf(L"num_threads: %llu\n", num_threads);
    std::vector<std::thread> threads = {};
    auto t0 = high_resolution_clock::now();
    for (size_t i = 0; i < num_threads; ++i) {
        threads.push_back(std::thread(fn));
    }
    for (size_t i = 0; i < num_threads; ++i) {
        threads[i].join();
    }
    auto t1 = high_resolution_clock::now();
    wprintf(L"time (ms): %lli\n", \
        duration_cast<milliseconds>(t1 - t0).count());
    return EXIT_SUCCESS;
}
```

USE MODERN SYNC APIS: BAD USER SPIN LOCK

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

MyLock::LOCK gLock;
```

```
void fn() {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
        MyLock::Lock(&gLock);
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    for (int k = 0; k < 4; k++)
                        a[m][i][j] += b[m][i][k] * c[m][k][j];
        r += std::accumulate((float*)a, \
            (float*)(a + LEN), 0.0f);
        MyLock::Unlock(&gLock);
    }
    wprintf(L"result: %f\n", r);
}
```

USE MODERN SYNC APIS: IMPROVED USER SPIN LOCK

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

alignas(64) MyLock::LOCK gLock;
```

```
void fn() {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
        MyLock::Lock(&gLock);
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    for (int k = 0; k < 4; k++)
                        a[m][i][j] += b[m][i][k] * c[m][k][j];
        r += std::accumulate((float*)a, \
            (float*)(a + LEN), 0.0f);
        MyLock::Unlock(&gLock);
    }
    wprintf(L"result: %f\n", r);
}
```

USE MODERN SYNC APIS: WAITFORSINGLEOBJECT

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

HANDLE hMutex;

int main(int argc, char* argv[]) {
    hMutex = CreateMutex(NULL,FALSE,NULL);
    // otherwise main is the same as before.
    // ...
}
```

```
void fn() {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
        WaitForSingleObject(hMutex, INFINITE);
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    for (int k = 0; k < 4; k++)
                        a[m][i][j] += b[m][i][k] * c[m][k][j];
        r += std::accumulate((float*)a, \
            (float*)(a + LEN), 0.0f);
        ReleaseMutex(hMutex);
    }
    wprintf(L"result: %f\n", r);
}
```

USE MODERN SYNC APIS: STD::MUTEX

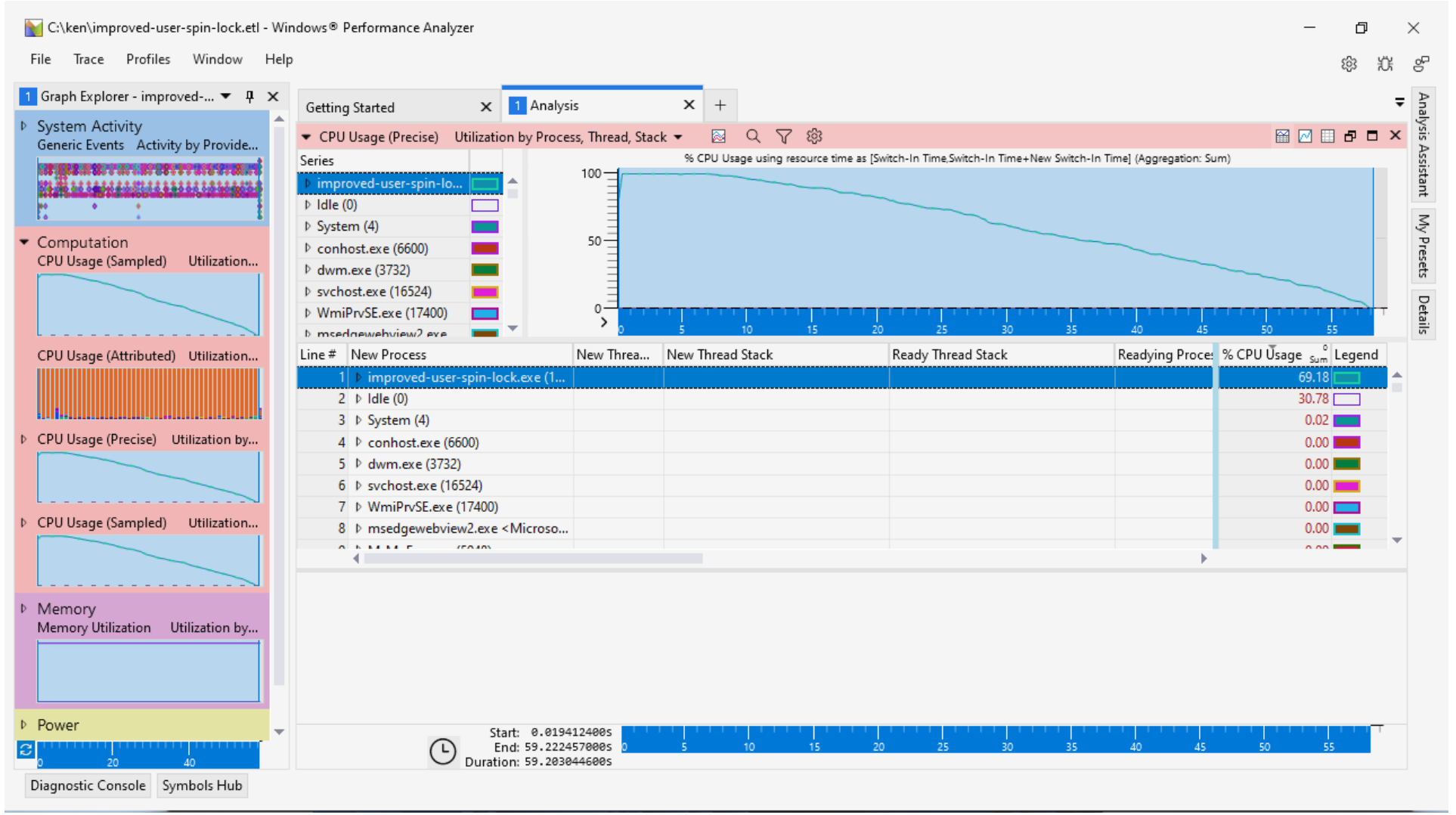
```
// MyLock not required. Let the OS do the work!
std::mutex mutex;
```

```
void fn() {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
       mutex.lock();
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                    for (int k = 0; k < 4; k++)
                        a[m][i][j] += b[m][i][k] * c[m][k][j];
        r += std::accumulate((float*)a, \
            (float*)(a + LEN), 0.0f);
        mutex.unlock();
    }
    wprintf(L"result: %f\n", r);
}
```

USE MODERN SYNC APIS

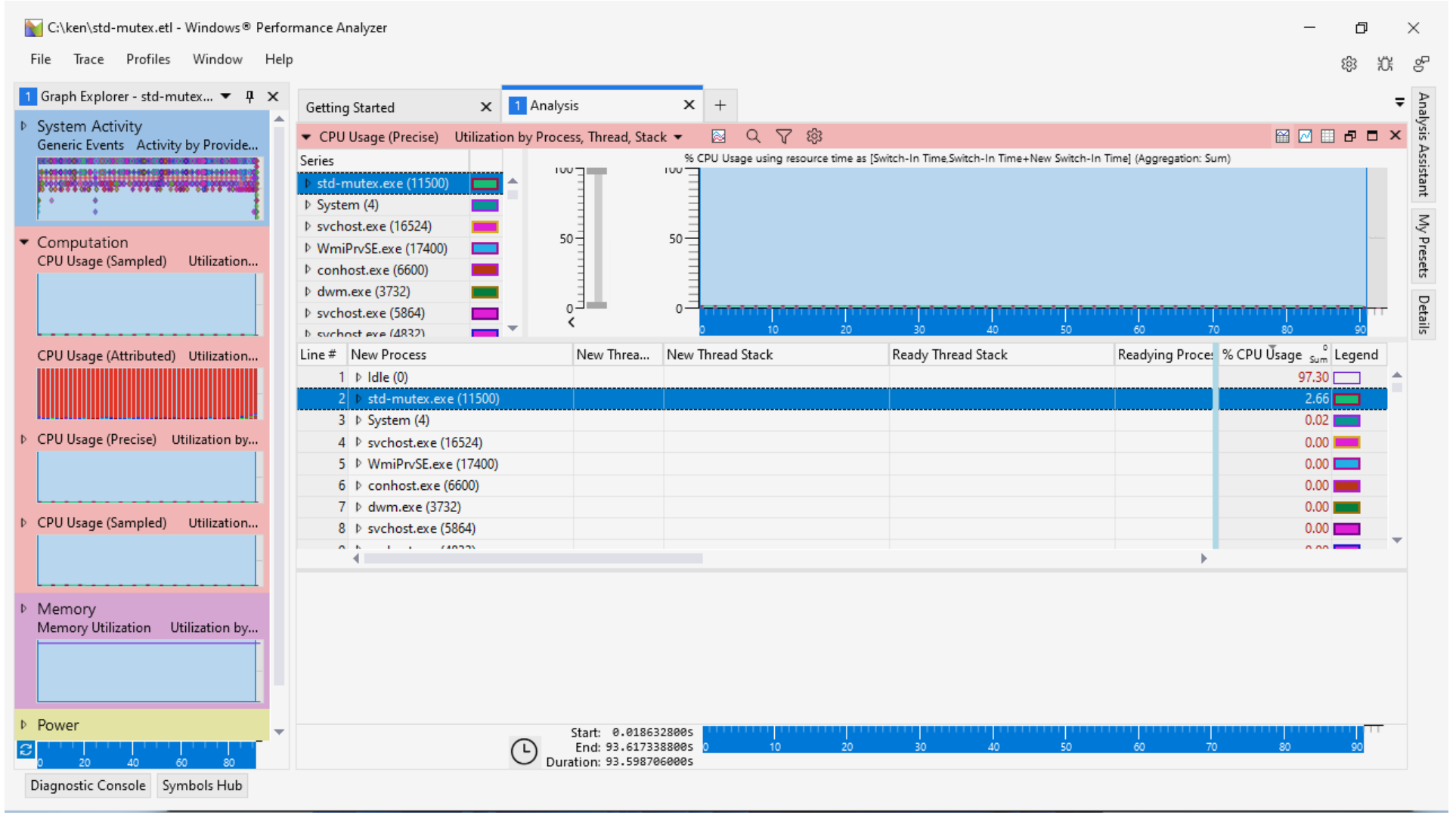
- **Prefer functions using mwaitx**
  - std::mutex
  - std::shared_mutex
  - AcquireSRWLockExclusive
  - AcquireSRWLockShared
  - SleepConditionVariableSRW
  - SleepConditionVariableCS
  - EnterCriticalSection
- **Avoid functions using syscall**
  - WaitForSingleObject
  - WaitForMultipleObjects

WINDOWS PERFORMANCE ANALYZER – SPIN LOCK



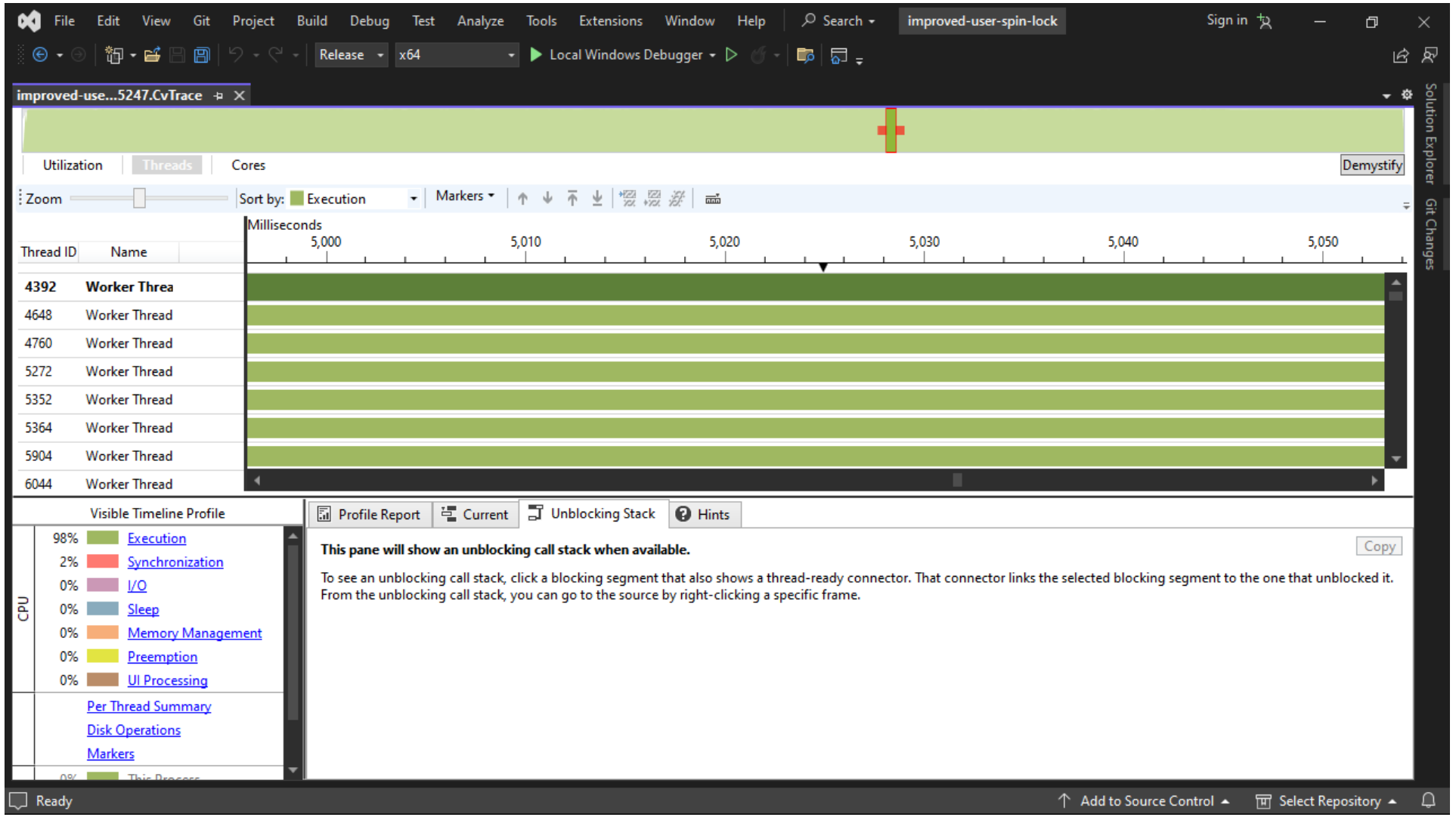

```
wpr.exe –start cpu
rem run test
pause
wpr.exe –stop log.etl
```

WINDOWS PERFORMANCE ANALYZER – STD::MUTEX

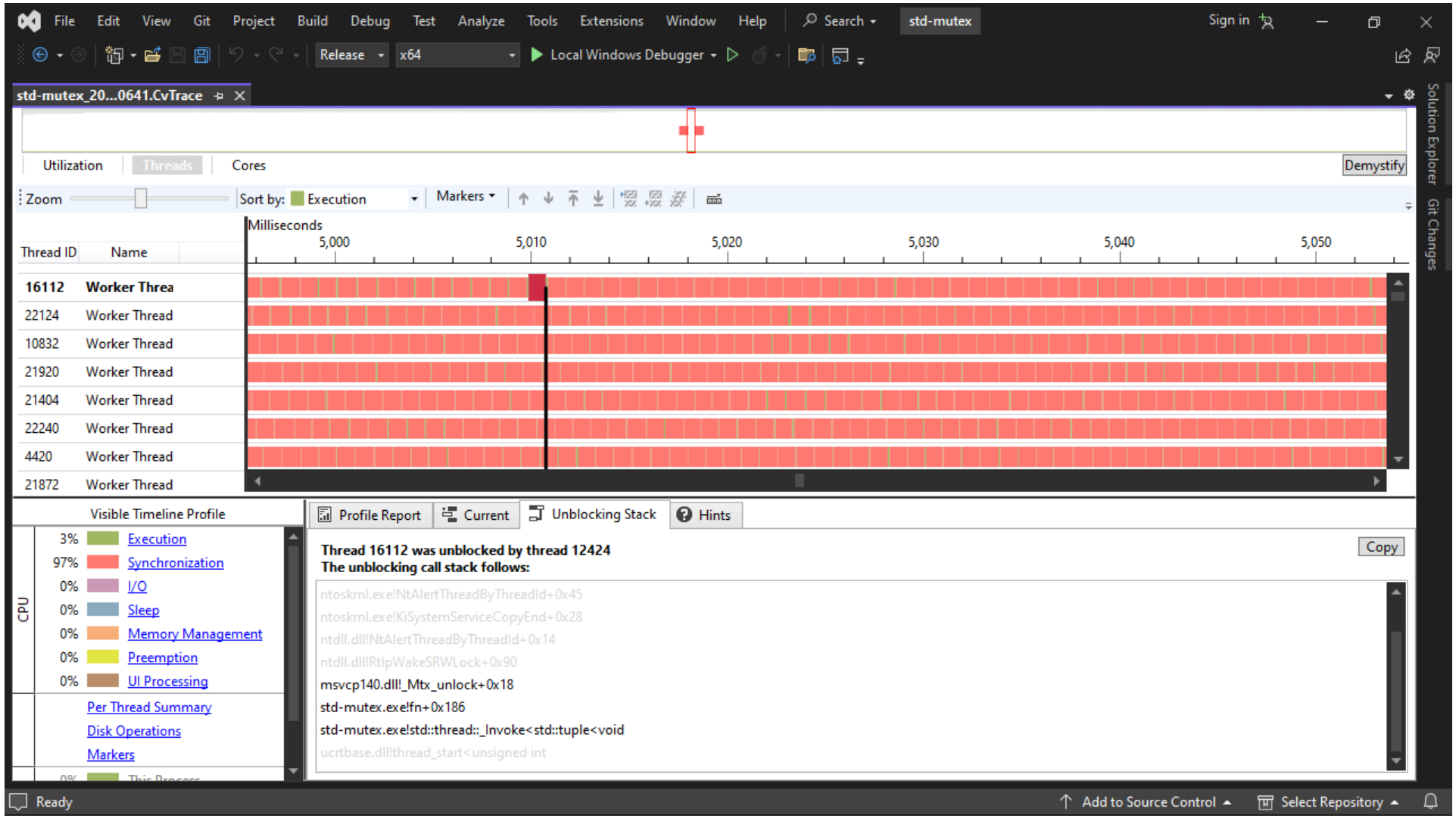



```
wpr.exe –start cpu
rem run test
pause
wpr.exe –stop log.etl
```

VISUAL STUDIO CONCURRENCY VISUALIZER – SPIN LOCK



VISUAL STUDIO CONCURRENCY VISUALIZER – STD::MUTEX



THREADING

TUNE THREAD POOL SIZE FOR INITIALIZATION AND GAME PLAY

- ***This advice is specific to AMD processors and is not general guidance for all processor vendors.***

- Profile your game to determine the optimal thread pool size for both game ***initialization*** and ***play***.

- Utilizing all logical processors in SMT dual-thread mode may benefit game ***initialization***.

- Utilizing only physical cores, each in single-thread, mode may benefit game ***play***.
  - for systems with at least 8 AMD Ryzen™ CPU cores.

- See the core count code sample at https://gpuopen.com/learn/cpu-core-counts/.

AVOID HARD AFFINITY MASKS ON PC

- Hard affinity masks interfere with OS power management and thread scheduling.
- *CPU Sets* provide APIs to declare application affinity in a 'soft' manner that is compatible with OS power management.

Minimum OS	Affinity Type	Function
Windows XP	hard	SetThreadAffinityMask
Windows 7	hard	SetThreadGroupAffinity
Windows 10	soft	SetThreadSelectedCpuSets
Windows 11	soft	SetThreadSelectedCpuSetMasks

My thread hard affinity = none

	t0	t1	t2	t3
CPU0	My thread	Other app	idle	idle
CPU1	idle	My thread	My thread	My thread

My thread hard affinity = CPU0

	t0	T1	t2	t3
CPU0	My thread	Other app	My thread	My thread
CPU1	idle	Idle	idle	idle

My thread soft affinity = CPU0

	t0	T1	t2	t3
CPU0	My thread	Other app	My thread	My thread
CPU1	idle	My thread	idle	idle

BEWARE OF PRIORITY BOOST

- Thread Priority describes the order in which threads are scheduled.
- Each thread has a dynamic priority.
- The system boosts the dynamic priority under certain conditions.
- Temporary priority-boosted threads may switch-in before threads intended to be higher priority by the developer.
- This feature can be disabled using SetProcessPriorityBoost and SetThreadPriorityBoost.

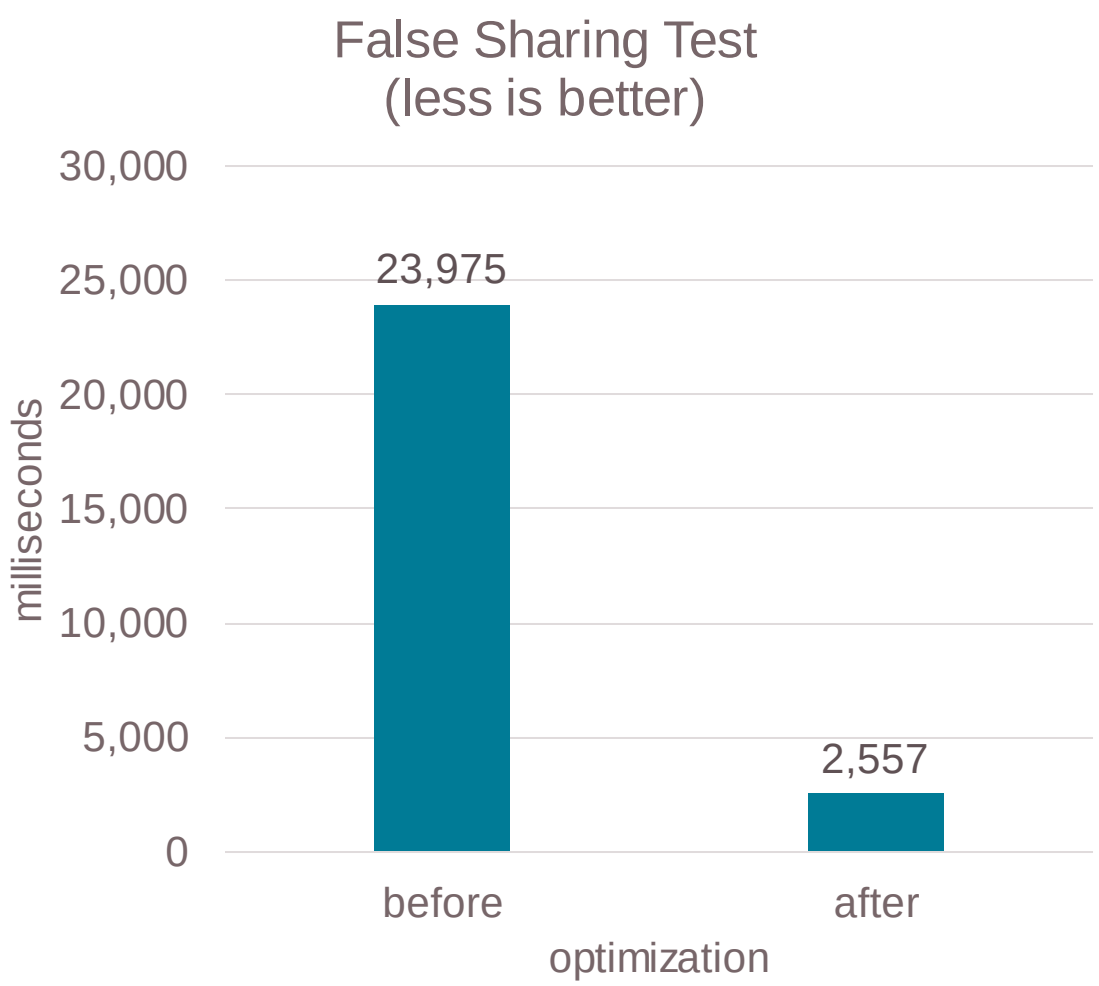
BASE PRIORITY AT THREAD PRIORITY LEVEL	NORMAL_ PRIORITY_ CLASS	ABOVE_ NORMAL_ PRIORITY_ CLASS
THREAD_PRIORITY_IDLE	1	1
THREAD_PRIORITY_LOWEST	6	8
THREAD_PRIORITY_BELOW_NORMAL	7	9
THREAD_PRIORITY_NORMAL	**8**	**10**
THREAD_PRIORITY_ABOVE_NORMAL	**9**	**11**
THREAD_PRIORITY_HIGHEST	**10**	**12**
THREAD_PRIORITY_TIME_CRITICAL	15	15

DATA ACCESS

ALIGN MEMCPY SOURCE AND DESTINATION POINTERS

- Update the compiler for the latest memcpy, memset, and other C runtime optimizations!
- Memcpy behavior is undefined if dest and src overlap, but the compiler may generate Rep Move String instructions which have defined overlapping behavior.
- Alignas(64) may allow faster rep movs microcode.
- Alignas(4096) may reduce store-to-load conflicts and benefit probe filtering on some processors.
  - PMCx024 LsBadStatus2 StliOther counts store-to-load conflicts where a load was unable to complete due to a non-forwardable conflict with an older store.
- Aligning to the bit_floor may provide a good balance of cache hits and alignment:
  - std::clamp(std::bit_floor(count), 4, 4096);

AVOID FALSE SHARING



- "False sharing" may occur when two or more cores modify different data within the same cache line.
- This microbenchmark showed its execution time reduced by about 90% after optimization using alignas(64)!

- *Performance of binaries compiled with Microsoft® Visual Studio 2022 v17.8.6.*
- *Testing done by AMD technology labs, January 6, 2024 on the following system. Test configuration: AMD Ryzen™ Threadripper™ 7995WX 96-Cores, NZXT Kraken 360 cooler, 256GB (8 x 32GB RDDR5-4800 memory, AMD Radeon™ RX 580 GPU with 31.0.12027.9001 (March 20, 2023), 1TB M.2 NVME SSD, AMD Reference Motherboard, Windows® 11 x64 version 23H2, 1920x1080 resolution. Actual results may vary.*

AVOID FALSE SHARING

```
#include <chrono>
#include <numeric>
#include <thread>
#include <vector>
#include <Windows.h>

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

using namespace std::chrono;
#define NUM_ITER 100000000

void fn(ThreadData* p, size_t seed) {
    srand(static_cast<unsigned int>(seed));
    p->sum = 0;
    for (int i = 0; i < NUM_ITER; i++)
        p->sum += rand() % 2;
}
```

```
int main(int argc, char* argv[]) {
    size_t num_threads = \
        GetMaximumProcessorCount(ALL_PROCESSOR_GROUPS);
    wprintf(L"num_threads: %llu\n", num_threads);
    ThreadData* a = static_cast<ThreadData*>(_aligned_malloc(
        num_threads * sizeof(ThreadData), 64));
    if (nullptr == a)
        return EXIT_FAILURE;
    std::vector<std::thread> threads = {};
    auto t0 = high_resolution_clock::now();
    for (size_t i = 0; i < num_threads; ++i)
        threads.push_back(std::thread(fn, &a[i], i));
    for (size_t i = 0; i < num_threads; ++i)
        threads[i].join();
    auto t1 = high_resolution_clock::now();
    wprintf(L"time (ms): %lli\n",
        duration_cast<milliseconds>(t1 - t0).count());
    for (size_t i = 0; i < num_threads; ++i)
        wprintf(L"sum[%llu] = %lu\n", i, (*(a + i)).sum);
    _aligned_free(a);
    return EXIT_SUCCESS;
}
```

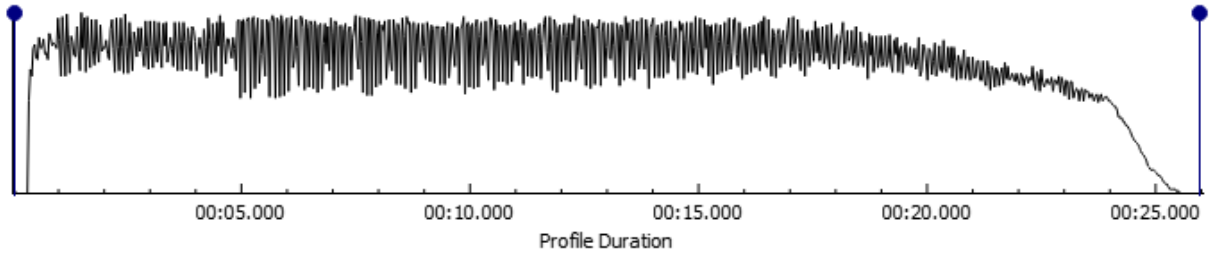
AMDuProf - [C:\Users\user\AppDa...b-09-2024_12-10-02]

PROFILE SUMMARY ANALYZE MEMORY

Function Hotspots

Grouped Metrics

IMIX

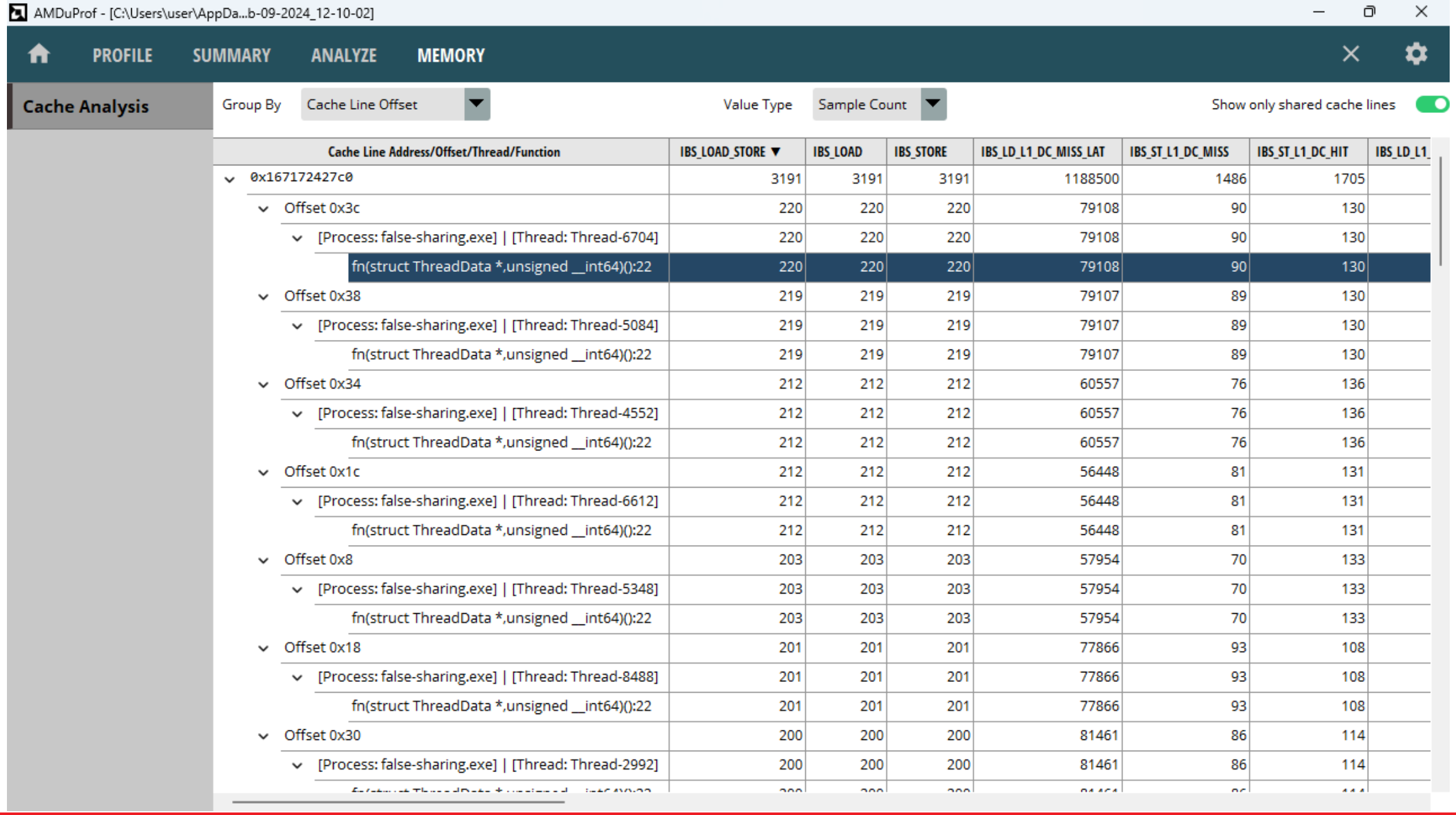


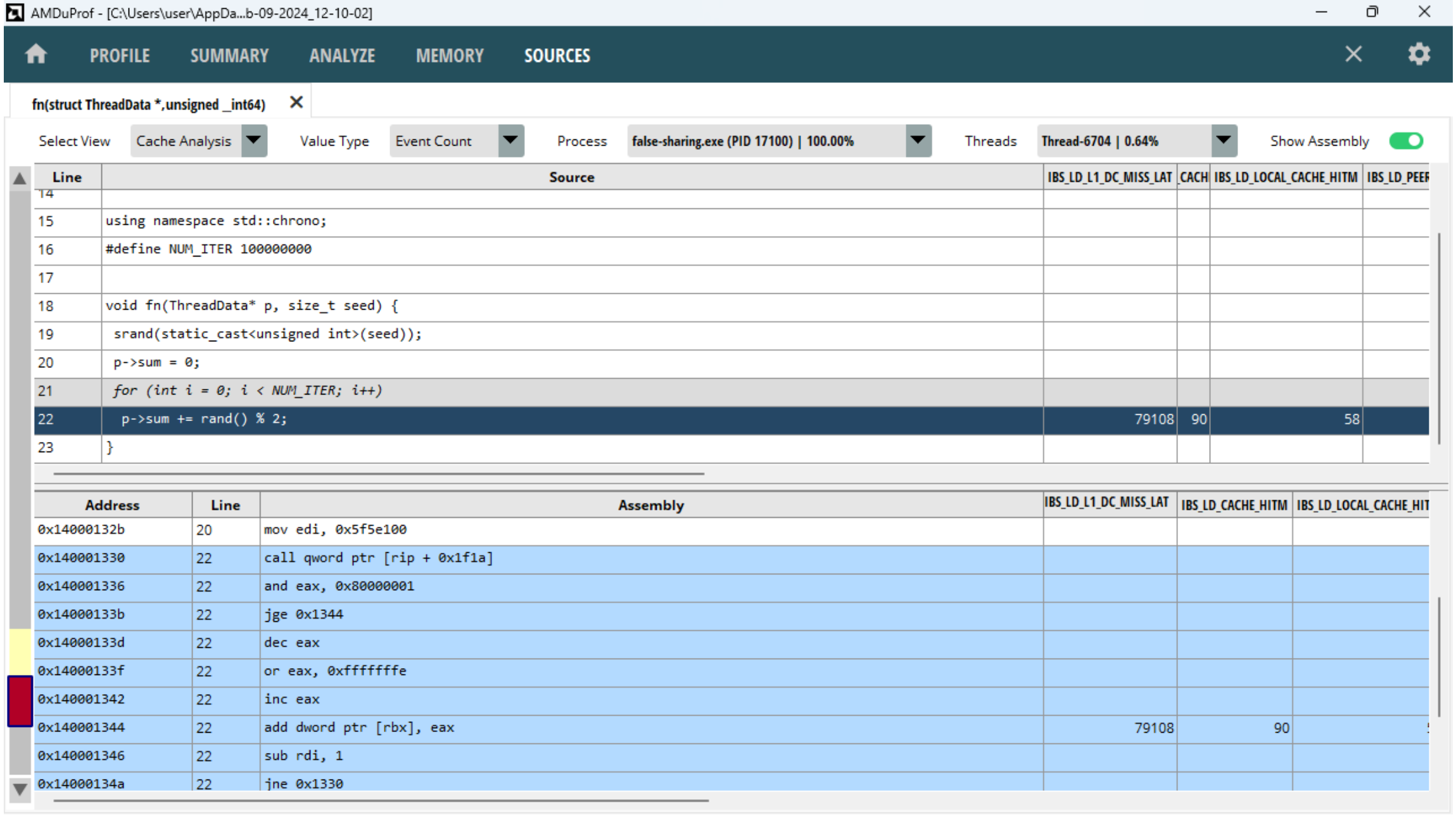
Select Metric: Thread Concurrency

Duration (ms): 0.00 | 26074.00 | Apply | Reset

Select View: Cache Analysis | Value Type: Event Count | System Modules: Include | Exclude | Search for functi... | Search

Functions	Modules	IBS_LD_L1_DC_MISS_LAT ▼	IBS_LD_CACHE_HITM	IBS_LD_LOCAL_CACHE_HITM	IBS_LD_PEER_CACHE_HITM	IBS_LD_RM
fn(struct ThreadData *,unsigned __int64)	false-sharing.ex	12342521	15731	10254	5477	
_acrt_getptd	ucrtbase.dll	3848	0	0	0	
KiExecuteAllDpcs	ntoskrnl.exe	2515	4	0	4	
PpmPerfSnapUtility	ntoskrnl.exe	2419	5	2	3	
rand	ucrtbase.dll	2120	0	0	0	
KiDpcInterrupt	ntoskrnl.exe	1353	0	0	0	
ExpInterlockedPushEntrySList	ntoskrnl.exe	1090	0	0	0	
wil_details_FeatureStateCache_GetCachedFeatureEnat	ntoskrnl.exe	646	0	0	0	
PoGetFrequencyBucket	ntoskrnl.exe	640	1	0	1	
KeReleaseSemaphoreEx	ntoskrnl.exe	549	1	0	1	
ComputeHyperThreadedProcessorEnergyUsingMsr	amdppm.sys	454	1	0	1	
FltpPerformPreCallbacksWorker	fltmgr.sys	419	0	0	0	
PpmPerfRecordUtility	ntoskrnl.exe	312	5	5	0	
PpmPerfSelectDomainStates	ntoskrnl.exe	112	0	0	0	
KiSwapContext	ntoskrnl.exe	104	1	1	0	
EtwEventEnabled	ntoskrnl.exe	80	0	0	0	

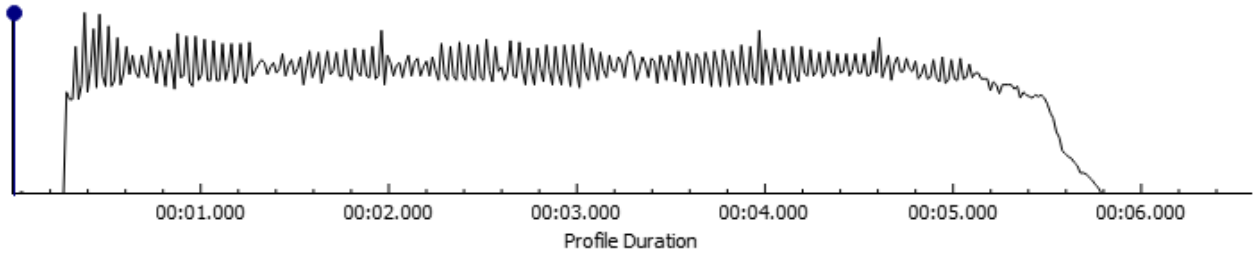

AMDuProf - [C:\Users\user\AppDa...b-09-2024_12-10-02]
PROFILE
SUMMARY
ANALYZE
MEMORY
Cache Analysis
Group By
Cache Line Offset
Value Type
Sample Count
Show only shared cache lines
Cache Line Address/Offset/Thread/Function
IBS_LOAD_STORE ▼
IBS_LOAD
IBS_STORE
IBS_LD_L1_DC_MISS_LAT
IBS_ST_L1_DC_MISS
IBS_ST_L1_DC_HIT
IBS_LD_L1_
0x167172427c0 3191 3191 3191 1188500 1486 1705
Offset 0x3c 220 220 220 79108 90 130
[Process: false-sharing.exe] | [Thread: Thread-6704] 220 220 220 79108 90 130
fn(struct ThreadData *,unsigned __int64)():22 220 220 220 79108 90 130
Offset 0x38 219 219 219 79107 89 130
[Process: false-sharing.exe] | [Thread: Thread-5084] 219 219 219 79107 89 130
fn(struct ThreadData *,unsigned __int64)():22 219 219 219 79107 89 130
Offset 0x34 212 212 212 60557 76 136
[Process: false-sharing.exe] | [Thread: Thread-4552] 212 212 212 60557 76 136
fn(struct ThreadData *,unsigned __int64)():22 212 212 212 60557 76 136
Offset 0x1c 212 212 212 56448 81 131
[Process: false-sharing.exe] | [Thread: Thread-6612] 212 212 212 56448 81 131
fn(struct ThreadData *,unsigned __int64)():22 212 212 212 56448 81 131
Offset 0x8 203 203 203 57954 70 133
[Process: false-sharing.exe] | [Thread: Thread-5348] 203 203 203 57954 70 133
fn(struct ThreadData *,unsigned __int64)():22 203 203 203 57954 70 133
Offset 0x18 201 201 201 77866 93 108
[Process: false-sharing.exe] | [Thread: Thread-8488] 201 201 201 77866 93 108
fn(struct ThreadData *,unsigned __int64)():22 201 201 201 77866 93 108
Offset 0x30 200 200 200 81461 86 114
[Process: false-sharing.exe] | [Thread: Thread-2992] 200 200 200 81461 86 114


AMDuProf - [C:\Users\user\AppDa...b-09-2024_12-10-02]
PROFILE
SUMMARY
ANALYZE
MEMORY
SOURCES
fn(struct ThreadData *,unsigned _int64)
Select View
Cache Analysis
Value Type
Event Count
Process
false-sharing.exe (PID 17100) | 100.00%
Threads
Thread-6704 | 0.64%
Show Assembly
Line
Source
IBS_LD_L1_DC_MISS_LAT
CACH
IBS_LD_LOCAL_CACHE_HITM
IBS_LD_PEER
15 using namespace std::chrono;
16 #define NUM_ITER 100000000
17
18 void fn(ThreadData* p, size_t seed) {
19 srand(static_cast<unsigned int>(seed));
20 p->sum = 0;
21 for (int i = 0; i < NUM_ITER; i++)
22 p->sum += rand() % 2;
79108
90
58
23 }
Address
Line
Assembly
IBS_LD_L1_DC_MISS_LAT
IBS_LD_CACHE_HITM
IBS_LD_LOCAL_CACHE_HIT
0x14000132b 20 mov edi, 0x5f5e100
0x140001330 22 call qword ptr [rip + 0x1f1a]
0x140001336 22 and eax, 0x80000001
0x14000133b 22 jge 0x1344
0x14000133d 22 dec eax
0x14000133f 22 or eax, 0xfffffffe
0x140001342 22 inc eax
0x140001344 22 add dword ptr [rbx], eax
79108
90
0x140001346 22 sub rdi, 1
0x14000134a 22 jne 0x1330

AMDuProf - [C:\Users\user\AppDa...b-09-2024_13-09-21]

PROFILE SUMMARY ANALYZE MEMORY

Function Hotspots
Grouped Metrics
IMIX



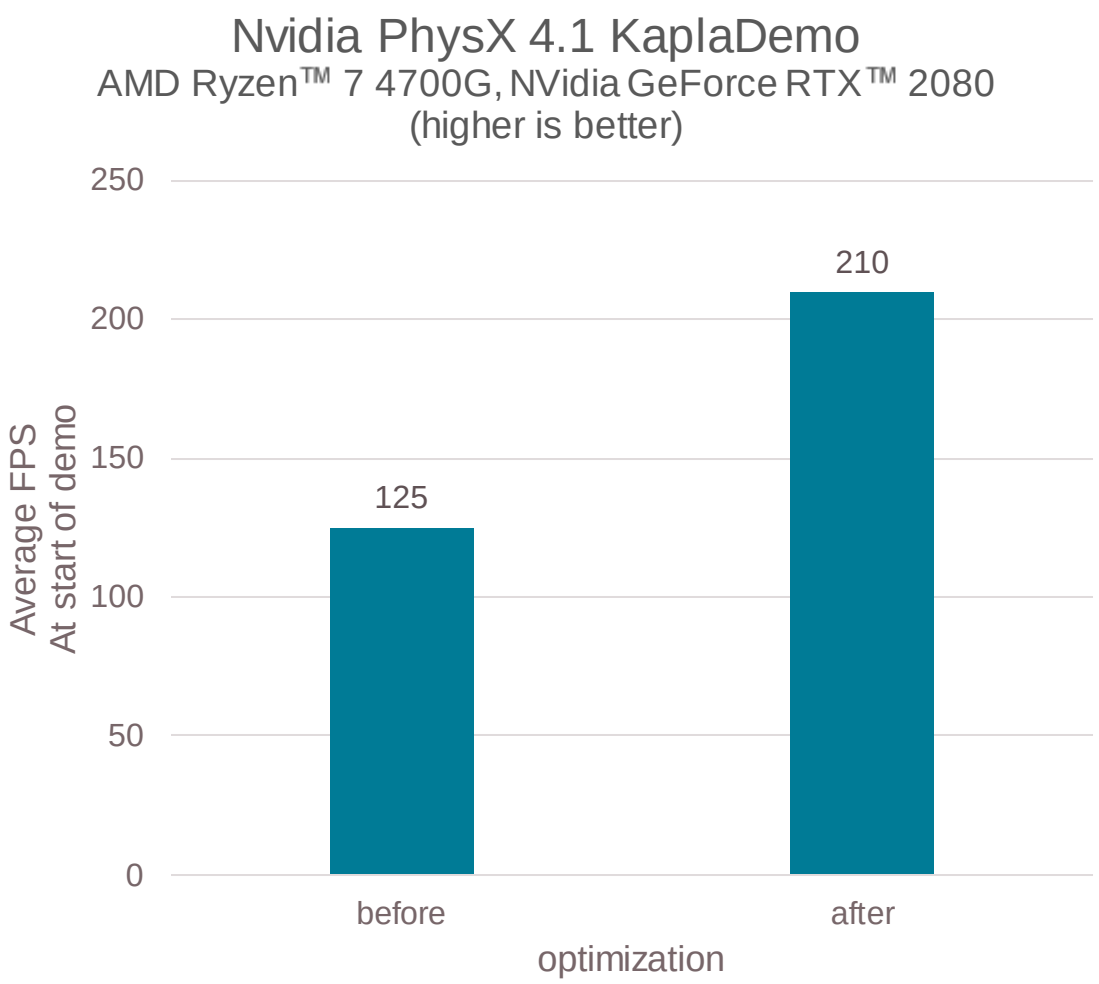
Select Metric: Thread Concurrency

Duration (ms): 0.00 6601.00 Apply Reset

Select View: Cache Analysis Value Type: Event Count System Modules: Include Exclude Search for functi... Search

Functions	Modules	IBS_LD_L1_DC_MISS_LAT ▼	IBS_LD_CACHE_HITM	IBS_LD_LOCAL_CACHE_HITM	IBS_LD_PEER_CACHE_HITM	IBS_LD_RMT_CAC
RtlFlsGetValue	ntdll.dll	17516	0	0	0	
fn(struct ThreadData *,unsigned __int64)	false-sharing.ex	10412	0	0	0	
FlsGetValue	kernelbase.dll	6443	0	0	0	
RtlSetLastWin32Error	ntdll.dll	4572	0	0	0	
rand	ucrtbase.dll	2876	0	0	0	
GetLastError	kernelbase.dll	2679	0	0	0	
PpmPerfSnapDeliveredPerformance	ntoskrnl.exe	1150	0	0	0	
PpmUpdateTimeAccumulation	ntoskrnl.exe	695	0	0	0	
ComputeHyperThreadedProcessorEnergyUsingMsr	amdppm.sys	623	0	0	0	
SnapEnergyCountersAndTimestamp	amdppm.sys	516	1	0	1	
PpmSnapPerformanceAccumulation	ntoskrnl.exe	473	0	0	0	
PpmPerfRecordUtility	ntoskrnl.exe	100	1	1	0	
PpmPerfSnapUtility	ntoskrnl.exe	45	1	1	0	
_security_check_cookie	ntdll.dll	30	0	0	0	
HalpTimerClockInterrupt	ntoskrnl.exe	20	0	0	0	
KeAccumulateTicks	ntoskrnl.exe	18	0	0	0	

USE SOFTWARE PREFETCH INSTRUCTIONS FOR LINKED DATA



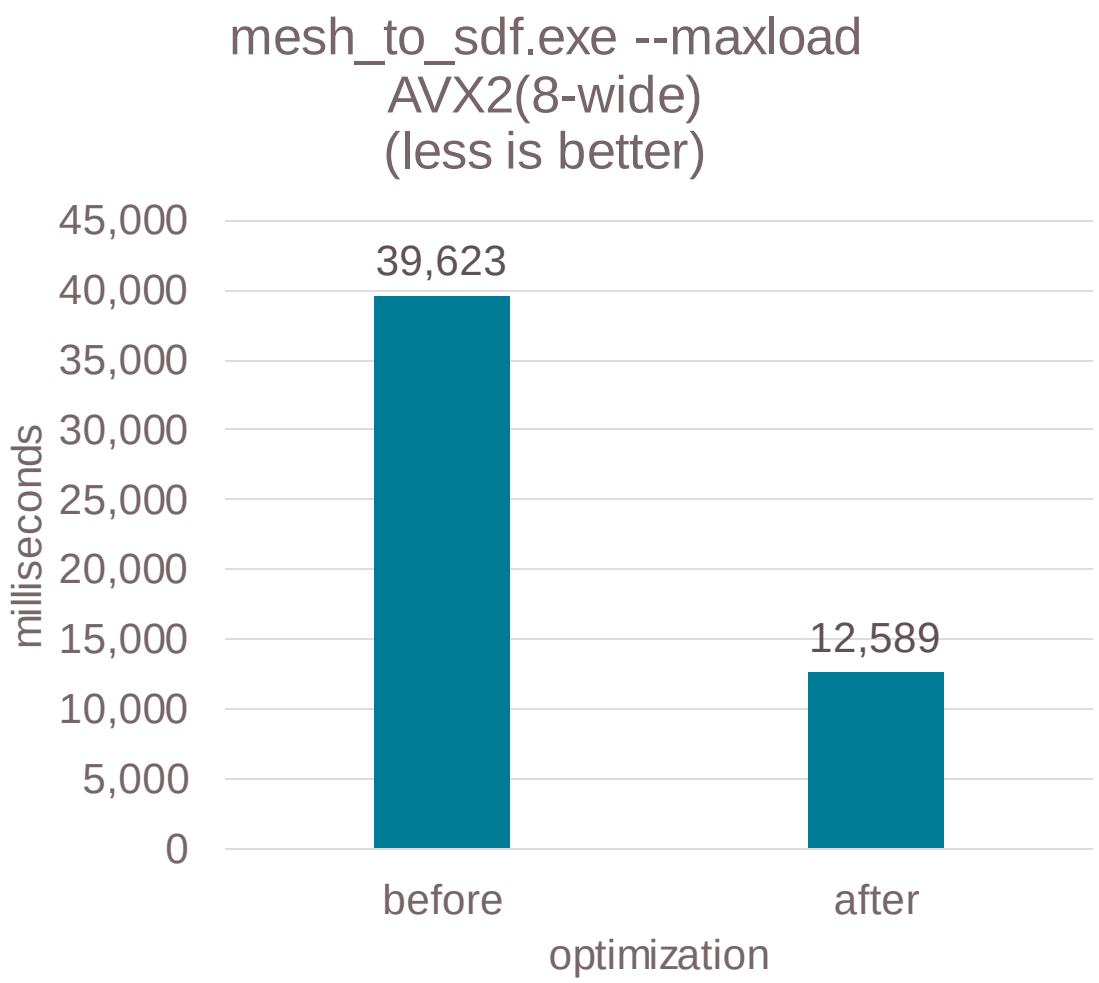
- Over 60% faster after optimization!
- *Performance of binaries compiled with Microsoft® Visual Studio 2019 v16.8.3.*
- *Testing done by AMD technology labs, January 4, 2021 on the following system. Test configuration: AMD Ryzen™ 7 4700G, AMD Wraith Spire Cooler, 16GB (2 x 8GB DDR4-3200 at 22-22-22-52) memory, NVidia GeForce RTX™ 2080 GPU with driver 460.89 (December 15, 2020), 512GB M.2 NVME SSD, AMD Ryzen™ Reference Motherboard, Windows® 10 x64 build 20H2, 1920x1080 resolution. Actual results may vary*

USE SOFTWARE PREFETCH INSTRUCTIONS FOR LINKED DATA

```
// Copyright (c) 2021 NVIDIA Corporation. All rights reserved
// ConvexRenderer.cpp from https://github.com/NVIDIAGameWorks/PhysX/tree/4.1/physx
void ConvexRenderer::updateTransformations()
{
  for (int i = 0; i < (int)mGroups.size(); i++) {
    ConvexGroup *g = mGroups[i];
    if (g->texCoords.empty())
      continue;
    float* tt = &g->texCoords[0];
    for (int j = 0; j < (int)g->convexes.size(); j++) {
      const Convex* c = g->convexes[j];
#if defined(APPLY_OPTIMIZATION)
      int distance = 4; // TODO find ideal number
      size_t future = (j + distance) % g->convexes.size();
      _mm_prefetch(0x0F8 + (char*)(g->convexes[future]), _MM_HINT_NTA); // mPxActor
      _mm_prefetch(0x100 + (char*)(g->convexes[future]), _MM_HINT_NTA); // mLocalPose
      _mm_prefetch(0x148 + (char*)(g->convexes[future]), _MM_HINT_NTA); // mMaterialOffset.x
      _mm_prefetch(0x14C + (char*)(g->convexes[future]), _MM_HINT_NTA); // mMaterialOffset.y
      _mm_prefetch(0x150 + (char*)(g->convexes[future]), _MM_HINT_NTA); // mMaterialOffset.z
      _mm_prefetch(0x164 + (char*)(g->convexes[future]), _MM_HINT_NTA); //mSurfaceMaterialId
      _mm_prefetch(0x160 + (char*)(g->convexes[future]), _MM_HINT_NTA); // mMaterialId
#endif
```

```
PxMat44 pose(c->getGlobalPose());
      float* mp = (float*)pose.front();
      float* ta = tt;
      for (int k = 0; k < 16; k++) {
        *(tt++) = *(mp++);
       }
      PxVec3 matOff = c->getMaterialOffset();
      ta[3] = matOff.x;
      ta[7] = matOff.y;
      ta[11] = matOff.z;
      int idFor2DTex = c->getSurfaceMaterialId();
      int idFor3DTex = c->getMaterialId();
      const int MAX_3D_TEX = 8;
      ta[15] = (float)(idFor2DTex*MAX_3D_TEX + idFor3DTex);
    }
    glBindTexture(GL_TEXTURE_2D, g->matTex);
    glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, g->texSize,
      g->texSize, GL_RGBA, GL_FLOAT, &g->texCoords[0]);
    glBindTexture(GL_TEXTURE_2D, 0);
  }
}
```

AVOID PENALTY FOR MIXING SSE AND AVX INSTRUCTIONS



- For AMD “Zen 2” and “Zen 3” CPUs, there is a significant penalty for mixing SSE and AVX instructions when the upper 128 bits of the YMM registers contain non-zero data.
- Benchmark execution time was reduced by over 60% after a VZeroUpper optimization.
- *Performance of binaries compiled with Microsoft® Visual Studio 2022 v17.8.6.*
- *Testing done by AMD technology labs, February 8, 2024 on the following system. Test configuration: AMD Ryzen™ Threadripper™ PRO 5995WX, Enermax LIQTECH TR4 II series 360mm liquid cooler, 256GB (8 x 32GB 2R RDDR4-3200 at 24-22-22-52) memory, AMD Radeon™ RX 6700 XT GPU with driver 24.1.1 (January 11, 2024), 1TB M.2 NVME SSD, AMD Reference Motherboard, Windows® 11 x64 version 23H2, 1920x1080 resolution. Actual results may vary.*

AVOID PENALTY FOR MIXING SSE AND AVX INSTRUCTIONS

- Use “SSE_AVX_STALLS” PMCx00E Floating Point Dispatch Faults > 0 to find code which may be missing VZeroUpper or VZeroAll instructions during AVX to SSE and SSE to AVX transitions.

- Optimization 1:
  - Use the /arch:AVX compiler flag.
  - AVX is supported by 97% of users according to the January 2024 Steam Hardware & Software Survey.

- Optimization 2:
  - Return a __m256 value using pass-by-reference in the function parameter list rather than the function return type.

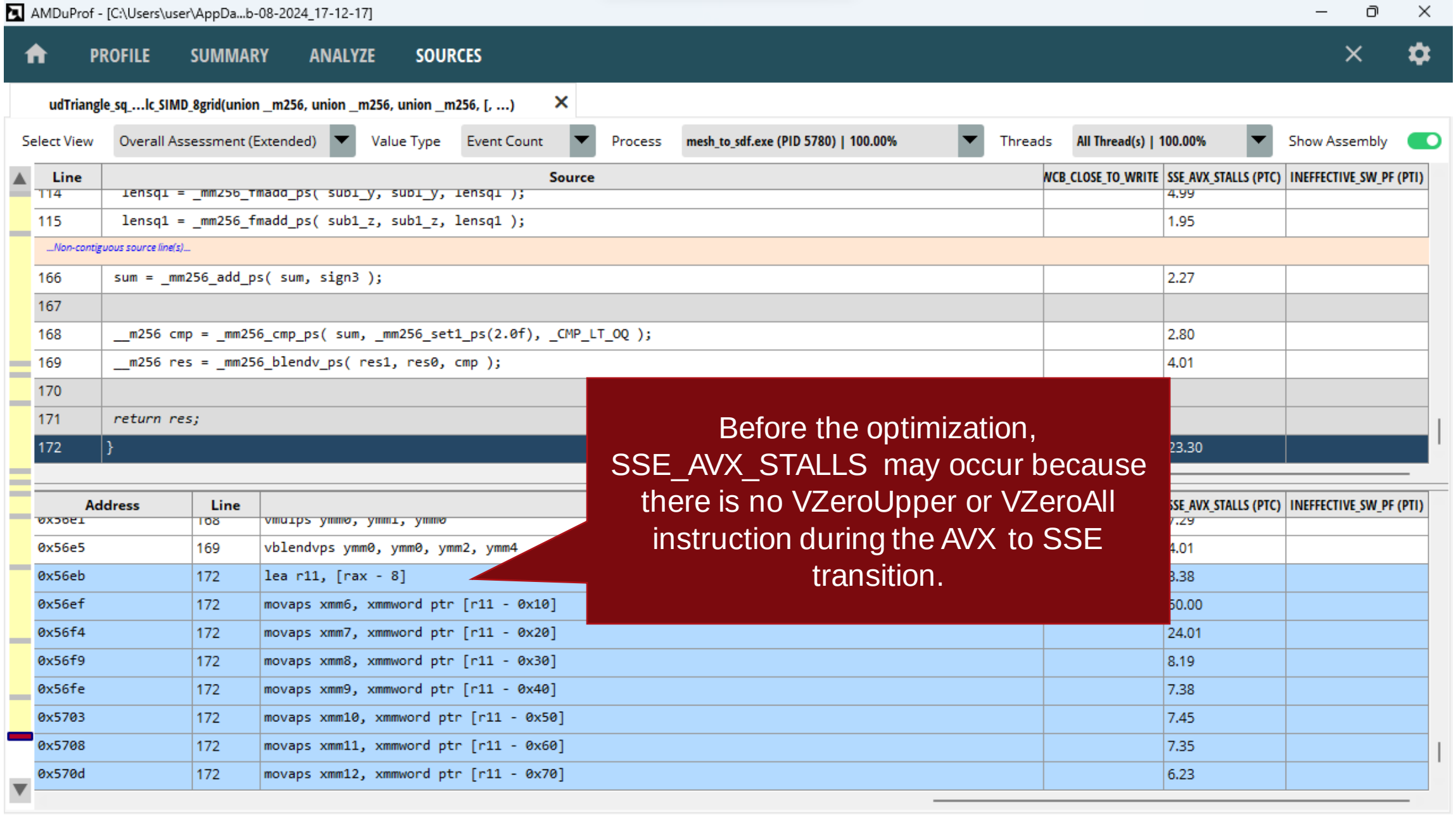
- Optimization 3:
  - Use __forceinline on the function definition.

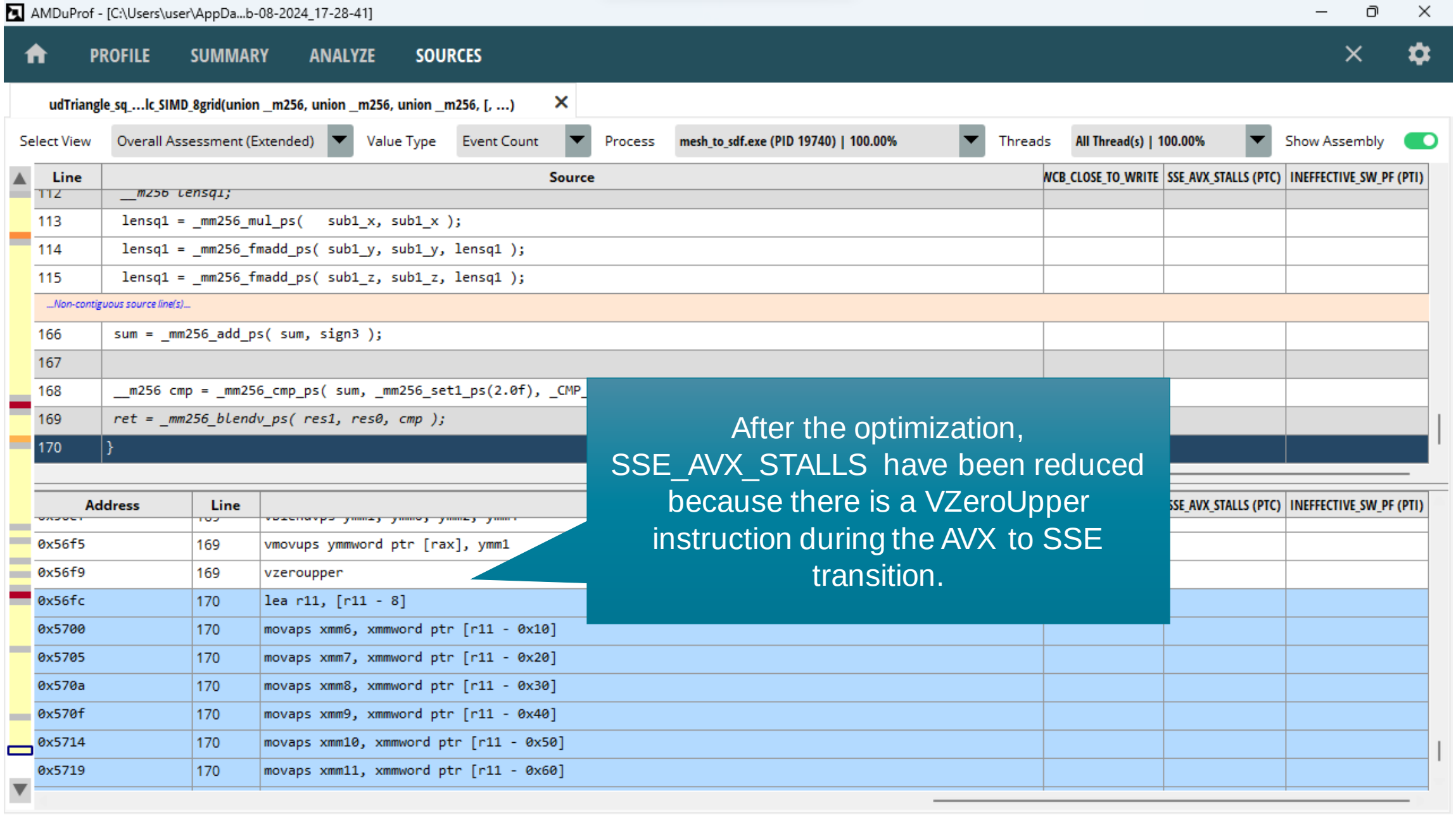
AVOID PENALTY FOR MIXING SSE AND AVX INSTRUCTIONS

```
// Before  Optimization
__m256 udTriangle_sq_precalc_SIMD_8grid(
    const __m256 p_x, const __m256 p_y,
    const __m256 p_z, const tri_precalc_t &pc )
{
    // ...
    __m256 res = _mm256_blendv_ps( res1, res0,
        cmp );

    return res;
}
```

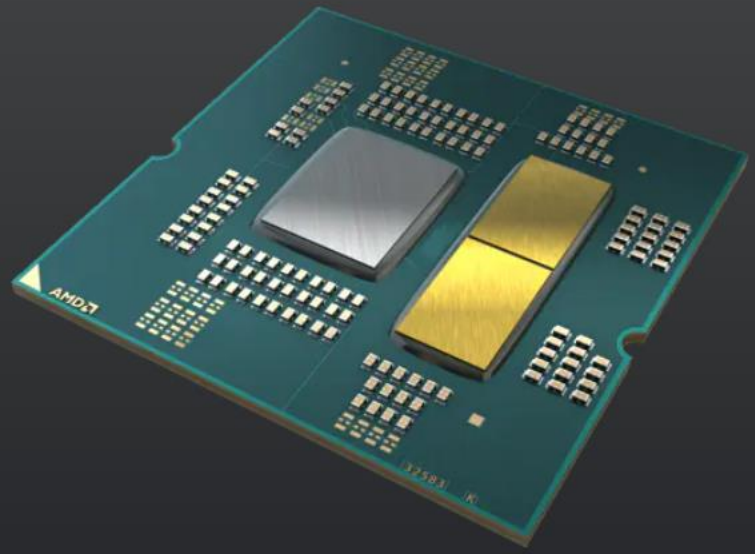
```
// After Optimization
void udTriangle_sq_precalc_SIMD_8grid(
    const __m256 p_x, const __m256 p_y,
    const __m256 p_z, const tri_precalc_t& pc,
    __m256 &ret )
{
    // ...
    ret = _mm256_blendv_ps( res1, res0,
        cmp );
}
```


AMDuProf - [C:\Users\user\AppDa...b-08-2024_17-12-17]
PROFILE
SUMMARY
ANALYZE
SOURCES
udTriangle_sq_...lc_SIMD_8grid(union _m256, union _m256, union _m256, [, ...)
Select View
Overall Assessment (Extended)
Value Type
Event Count
Process
mesh_to_sdf.exe (PID 5780) | 100.00%
Threads
All Thread(s) | 100.00%
Show Assembly
Line
Source
WCB_CLOSE_TO_WRITE
SSE_AVX_STALLS (PTC)
INEFFECTIVE_SW_PF (PTI)
114 lensq1 = _mm256_fmadd_ps(sub1_y, sub1_y, lensq1);
4.99
115 lensq1 = _mm256_fmadd_ps(sub1_z, sub1_z, lensq1);
1.95
...Non-contiguous source line(s)...
166 sum = _mm256_add_ps(sum, sign3);
2.27
167
168 __m256 cmp = _mm256_cmp_ps(sum, _mm256_set1_ps(2.0f), _CMP_LT_OQ);
2.80
169 __m256 res = _mm256_blendv_ps(res1, res0, cmp);
4.01
170
171 return res;
172 }
23.30
Address
Line
SSE_AVX_STALLS (PTC)
INEFFECTIVE_SW_PF (PTI)
0x56e5 169 vblendvps ymm0, ymm0, ymm2, ymm4
4.01
0x56eb 172 lea r11, [rax - 8]
0x56ef 172 movaps xmm6, xmmword ptr [r11 - 0x10]
0x56f4 172 movaps xmm7, xmmword ptr [r11 - 0x20]
24.01
0x56f9 172 movaps xmm8, xmmword ptr [r11 - 0x30]
8.19
0x56fe 172 movaps xmm9, xmmword ptr [r11 - 0x40]
7.38
0x5703 172 movaps xmm10, xmmword ptr [r11 - 0x50]
7.45
0x5708 172 movaps xmm11, xmmword ptr [r11 - 0x60]
7.35
0x570d 172 movaps xmm12, xmmword ptr [r11 - 0x70]
6.23
Before the optimization, SSE_AVX_STALLS may occur because there is no VZeroUpper or VZeroAll instruction during the AVX to SSE transition.


AMDuProf - [C:\Users\user\AppDa...b-08-2024_17-28-41]
PROFILE
SUMMARY
ANALYZE
SOURCES
udTriangle_sq_...lc_SIMD_8grid(union _m256, union _m256, union _m256, [, ...)
Select View
Overall Assessment (Extended)
Value Type
Event Count
Process
mesh_to_sdf.exe (PID 19740) | 100.00%
Threads
All Thread(s) | 100.00%
Show Assembly
Line
Source
WCB_CLOSE_TO_WRITE
SSE_AVX_STALLS (PTC)
INEFFECTIVE_SW_PF (PTI)
112 __m256 lensq1;
113 lensq1 = _mm256_mul_ps(sub1_x, sub1_x);
114 lensq1 = _mm256_fmadd_ps(sub1_y, sub1_y, lensq1);
115 lensq1 = _mm256_fmadd_ps(sub1_z, sub1_z, lensq1);
...Non-contiguous source line(s)...
166 sum = _mm256_add_ps(sum, sign3);
167
168 __m256 cmp = _mm256_cmp_ps(sum, _mm256_set1_ps(2.0f), _CMP_
169 ret = _mm256_blendv_ps(res1, res0, cmp);
170 }
Address
Line
0x56f5 169 vmovups ymmword ptr [rax], ymm1
0x56f9 169 vzeroupper
0x56fc 170 lea r11, [r11 - 8]
0x5700 170 movaps xmm6, xmmword ptr [r11 - 0x10]
0x5705 170 movaps xmm7, xmmword ptr [r11 - 0x20]
0x570a 170 movaps xmm8, xmmword ptr [r11 - 0x30]
0x570f 170 movaps xmm9, xmmword ptr [r11 - 0x40]
0x5714 170 movaps xmm10, xmmword ptr [r11 - 0x50]
0x5719 170 movaps xmm11, xmmword ptr [r11 - 0x60]
After the optimization, SSE_AVX_STALLS have been reduced because there is a VZeroUpper instruction during the AVX to SSE transition.

DO YOU WANT TO KNOW MORE?

AMD RYZEN Performance guide



Design faster. Render faster. Iterate faster.

Our **AMD Ryzen**™ Performance Guide will help guide you through the optimization process with a collection of tidbits, tips, and tricks which aim to support you in your performance quest.

Tools

SOFTWARE OPTIMIZATION GUIDES AT AMD.COM

CPU Architecture	Publication No.	Name
AMD “Zen 4”	57647	Software Optimization Guide for the AMD "Zen4" Microarchitecture
AMD “Zen 3”	56665	Software Optimization Guide for AMD EPYC™ 7003 Processors (formerly Software Optimization Guide for AMD Family 19h Processors)
AMD “Zen 2”	56305	Software Optimization Guide for AMD EPYC™ 7002 Processors (formerly Software Optimization Guide for AMD Family 17h Models 30h and Greater Processors)
AMD “Zen 1”	55723	Software Optimization Guide for AMD Family 17h Processors

John.Hartwig@amd.com

Kenneth.Mitchell@amd.com

Design faster. Render faster. Iterate faster.

DISCLAIMER AND NOTICES

Disclaimer 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. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. 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. THIS INFORMATION IS PROVIDED 'AS IS." 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 NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, 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.

AMD is not responsible for any electronic virus or damage or losses therefrom that may be caused by changes or modifications that you make to your system, including but not limited to antivirus software. Changes to your system configurations and settings, including but not limited to antivirus software, is done at your sole discretion and under no circumstances will AMD be liable to you for any such changes. You assume all risk and are solely responsible for any damages that may arise from or are related to changes that you make to your system, including but not limited to antivirus software.

AMD, the AMD Arrow logo, Ryzen™, Threadripper™, Radeon™, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies. Microsoft, Windows, and Visual Studio are registered trademarks of Microsoft Corporation in the US and/or other countries. Unreal® is a trademark or registered trademark of Epic Games, Inc. in the United States of America and elsewhere. NVIDIA is a trademark and/or registered trademark of NVIDIA Corporation in the U.S. and/or other countries. Steam is a trademark and/or registered trademark of Valve Corporation. PCIe is a registered trademark of PCI-SIG.

AMD products or technologies may include hardware to accelerate encoding or decoding of certain video standards but require the use of additional programs/applications.

DISCLAIMER AND NOTICES

- Code sample on slide 70 is modified.
- Copyright (c) 2024 NVIDIA Corporation. All rights reserved. Code Sample is licensed subject to the following: “Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of NVIDIA CORPORATION nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.”
- MeshToSDF, Copyright 2024 Mikkel Gjoel under MIT License. https://github.com/pixelmager/MeshToSDF
- Infiltrator Demo and City Sample use the Unreal® Engine. Unreal® is a trademark or registered trademark of Epic Games, Inc. in the United States of America and elsewhere.
- Unreal® Engine, Copyright 1998 – 2024, Epic Games, Inc. All rights reserved.
- Intel® Embree is released as Open Source under the Apache 2.0 license.

DISCLAIMER AND NOTICES

- Claim “Zen 4” average 13% IPC uplift compared to “Zen 3” desktop processors
  - RPL-005: Testing as of 15 August, 2022, by AMD Performance Labs using the following hardware: AMD AM5 Reference Motherboard with AMD Ryzen™ 7 7700X with G.Skill DDR5-6000C30 (F5-6000J3038F16GX2-TZ5N) with AMD EXPO™ loaded, AMD AM4 Reference Motherboard with AMD Ryzen™ 7 5800X and DDR4-3600C16. Processors fixed to 4GHz frequency with 8C16 enabled and evaluated with 22 different workloads. ALL SYSTEMS configured with NXZT Kraken X63, open air test bench, Radeon™ RX 6950XT (driver 22.7.1 Optional), Windows® 11 22000.856, AMD Smart Access Memory/PCIe® Resizable Base Address Register (“ReBAR”) ON, Virtualization-Based Security (VBS) OFF. Results may vary.
- Design faster. Render faster. Iterate faster. Create more, faster with AMD Ryzen™ processors
  - Testing by AMD Performance Labs as of September 23, 2020 using a Ryzen™ 9 5950X and Intel Core i9-10900K configured with DDR4-3600C16 and NVIDIA GeForce RTX 2080 Ti. Results may vary. R5K-039
- The information contained herein is for informational purposes only, and is subject to change without notice. Timelines, roadmaps, and/or product release dates shown in these slides are plans only and subject to change. “Navi”, “Vega”, “Polaris”, “Zen, “Zen+”, "Zen 2", "Zen 3", and "Zen 4" are codenames for AMD architectures, and are not product names. GD-122