AMD EPYC AMD INSTINCT AMD RYZEN AMD RADEON

AMD RYZEN™ PROCESSOR SOFTWARE OPTIMIZATION

KEN MITCHELL

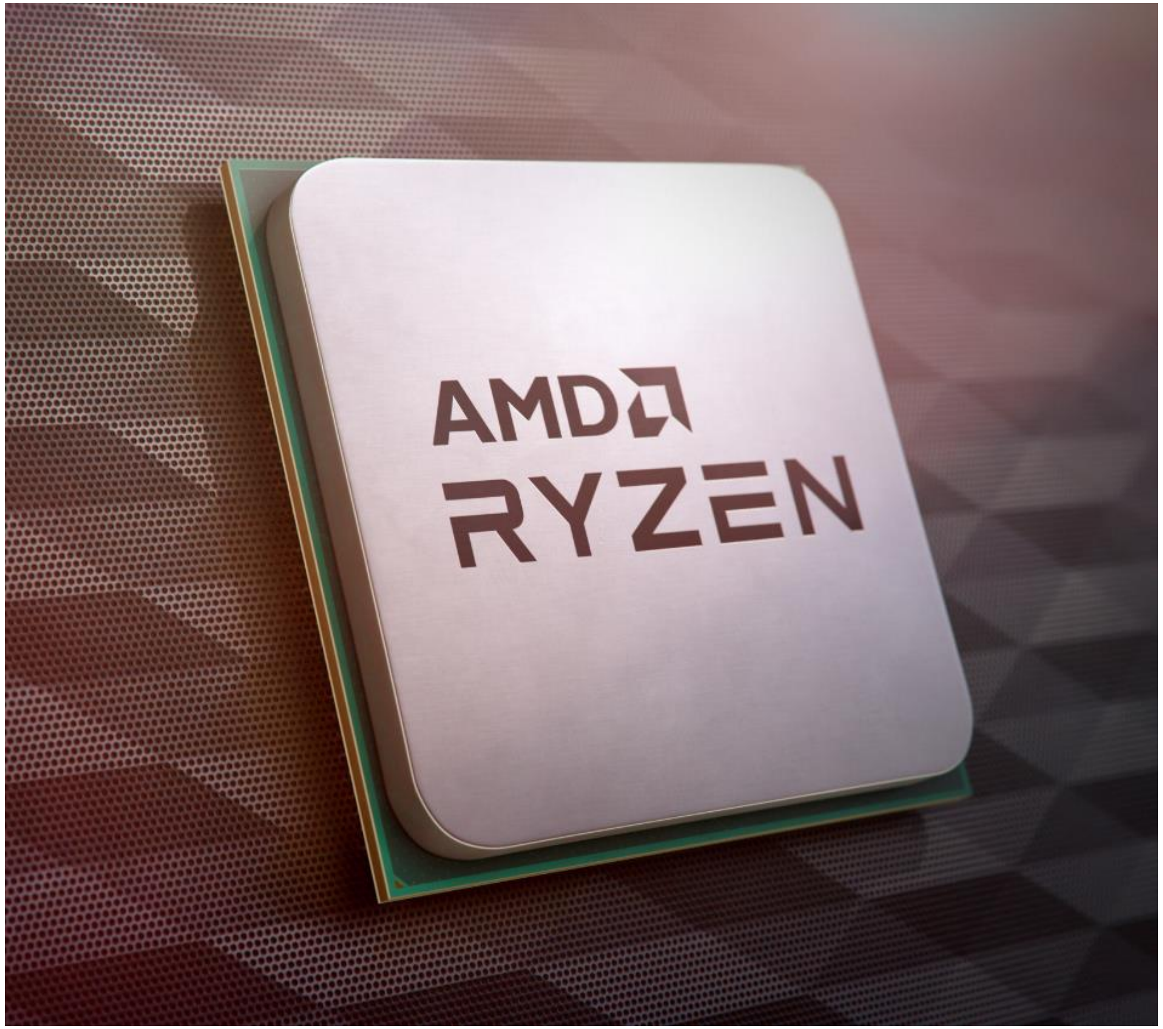
GDC

AMD GPUOpen

AGENDA

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

ABSTRACT



- Join AMD for an introduction to the AMD Ryzen™ family of processors which power today's game consoles and PCs.
- Learn about Ryzen™ products.
- Dive into instruction sets, cache hierarchies, resource sharing, and simultaneous multi-threading.
- Discover profiling tools and techniques.
- Gain insight into code optimization opportunities and lessons learned with examples including C/C++, assembly, and hardware performance-monitoring counters.

SPEAKER BIOGRAPHY

- **Ken Mitchell** is a Principal Member of Technical Staff in the AMD Game Engineering team where he focuses on helping game developers utilize AMD processors efficiently. His previous work includes automating & analyzing PC applications for performance projections of future AMD products as well as developing benchmarks. Ken studied computer science at the University of Texas at Austin.

PRODUCTS

AMD RYZEN™ 6000 SERIES MOBILE PROCESSORS

MODEL	GRAPHICS MODEL	CORES	THREADS	MAX. BOOST CLOCK	BASE CLOCK	DEFAULT TDP
AMD Ryzen™ 9 6980HX	AMD Radeon™ 680M	8	16	Up to 5.0GHz	3.3GHz	45W
AMD Ryzen™ 9 6980HS	AMD Radeon™ 680M	8	16	Up to 5.0GHz	3.3GHz	35W
AMD Ryzen™ 9 6900HX	AMD Radeon™ 680M	8	16	Up to 4.9GHz	3.3GHz	45W
AMD Ryzen™ 9 6900HS	AMD Radeon™ 680M	8	16	Up to 4.9GHz	3.3GHz	35W
AMD Ryzen™ 7 6800H	AMD Radeon™ 680M	8	16	Up to 4.7GHz	3.2GHz	45W
AMD Ryzen™ 7 6800HS	AMD Radeon™ 680M	8	16	Up to 4.7GHz	3.2GHz	35W
AMD Ryzen™ 7 6800U	AMD Radeon™ 680M	8	16	Up to 4.7GHz	2.7GHz	15-28W
AMD Ryzen™ 5 6600H	AMD Radeon™ 660M	6	12	Up to 4.5GHz	3.3GHz	45W
AMD Ryzen™ 5 6600HS	AMD Radeon™ 660M	6	12	Up to 4.5GHz	3.3GHz	35W
AMD Ryzen™ 5 6600U	AMD Radeon™ 660M	6	12	Up to 4.5GHz	2.9GHz	15-28W

AMD RYZEN™ 5000 SERIES DESKTOP PROCESSORS

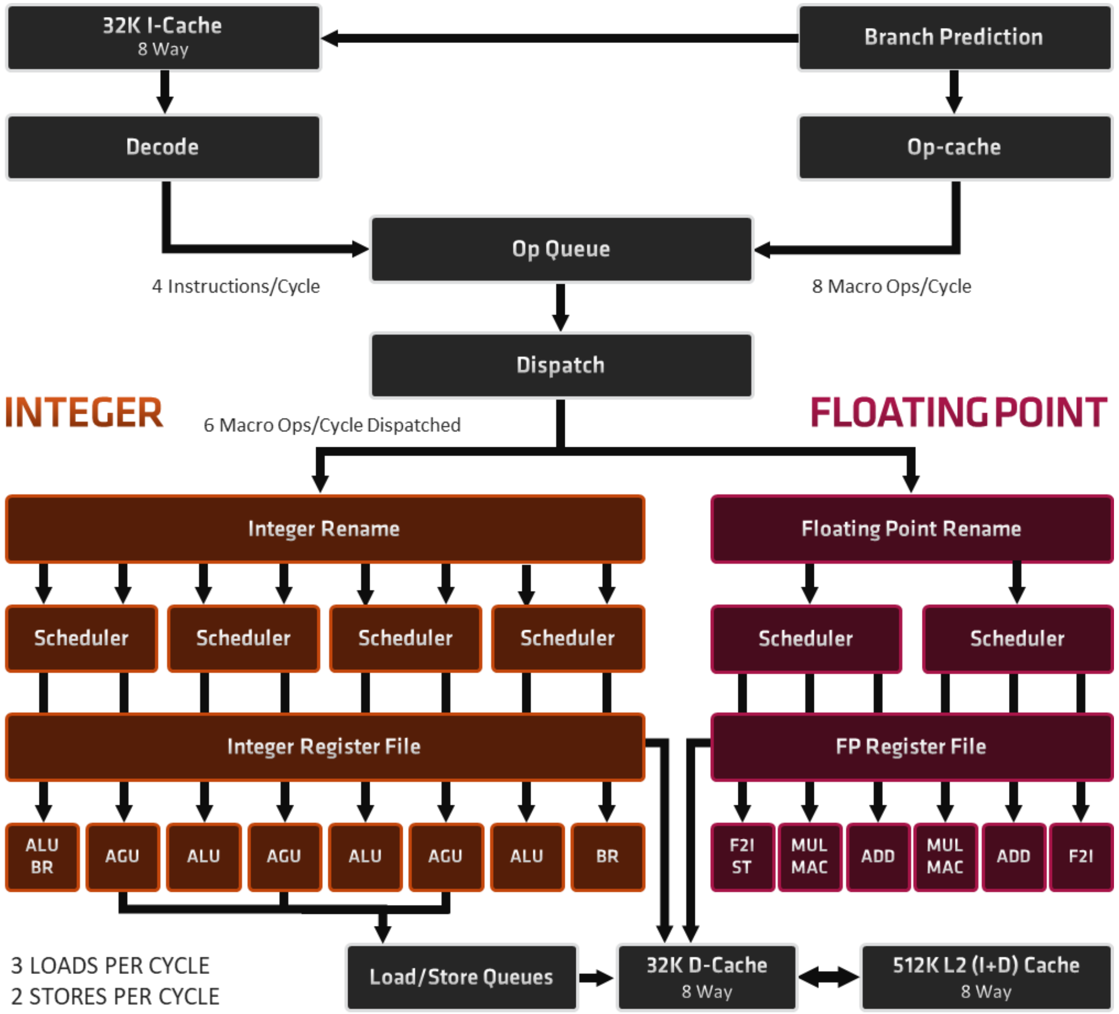
Model	Integrated Graphics	Cores	Threads	Total L3 Cache	Max Boost Clock	Base Clock	Default TDP
AMD Ryzen™ 9 5950X	-	16	32	64	Up to 4.9 GHz	3.4 GHz	105 W
AMD Ryzen™ 9 5900X	-	12	24	64	Up to 4.8 GHz	3.7 GHz	105 W
AMD Ryzen™ 7 5800X3D	-	8	16	96	Up to 4.5 GHz	3.4 GHz	105 W
AMD Ryzen™ 7 5800X	-	8	16	32	Up to 4.7 GHz	3.8 GHz	105 W
AMD Ryzen™ 5 5600X	-	6	12	32	Up to 4.6 GHz	3.7 GHz	65 W
AMD Ryzen™ 7 5700G	Radeon™	8	16	16	Up to 4.6 GHz	3.8 GHz	65 W
AMD Ryzen™ 5 5600G	Radeon™	6	12	16	Up to 4.4 GHz	3.9 GHz	65 W

AMD RYZEN™ THREADRIPPER™ PRO 5000WX SERIES PROCESSORS

Model	Integrated Graphics	Cores	Threads	Max Boost Clock	Base Clock	Default TDP
AMD Ryzen™ Threadripper™ PRO 5995WX	-	64	128	Up to 4.5 GHz	2.7 GHz	280 W
AMD Ryzen™ Threadripper™ PRO 5975WX	-	32	64	Up to 4.5 GHz	3.6 GHz	280 W
AMD Ryzen™ Threadripper™ PRO 5965WX	-	24	48	Up to 4.5 GHz	3.8 GHz	280 W
AMD Ryzen™ Threadripper™ PRO 5955WX	-	16	32	Up to 4.5 GHz	4.0 GHz	280 W
AMD Ryzen™ Threadripper™ PRO 5945WX	-	12	24	Up to 4.5 GHz	4.1 GHz	280 W

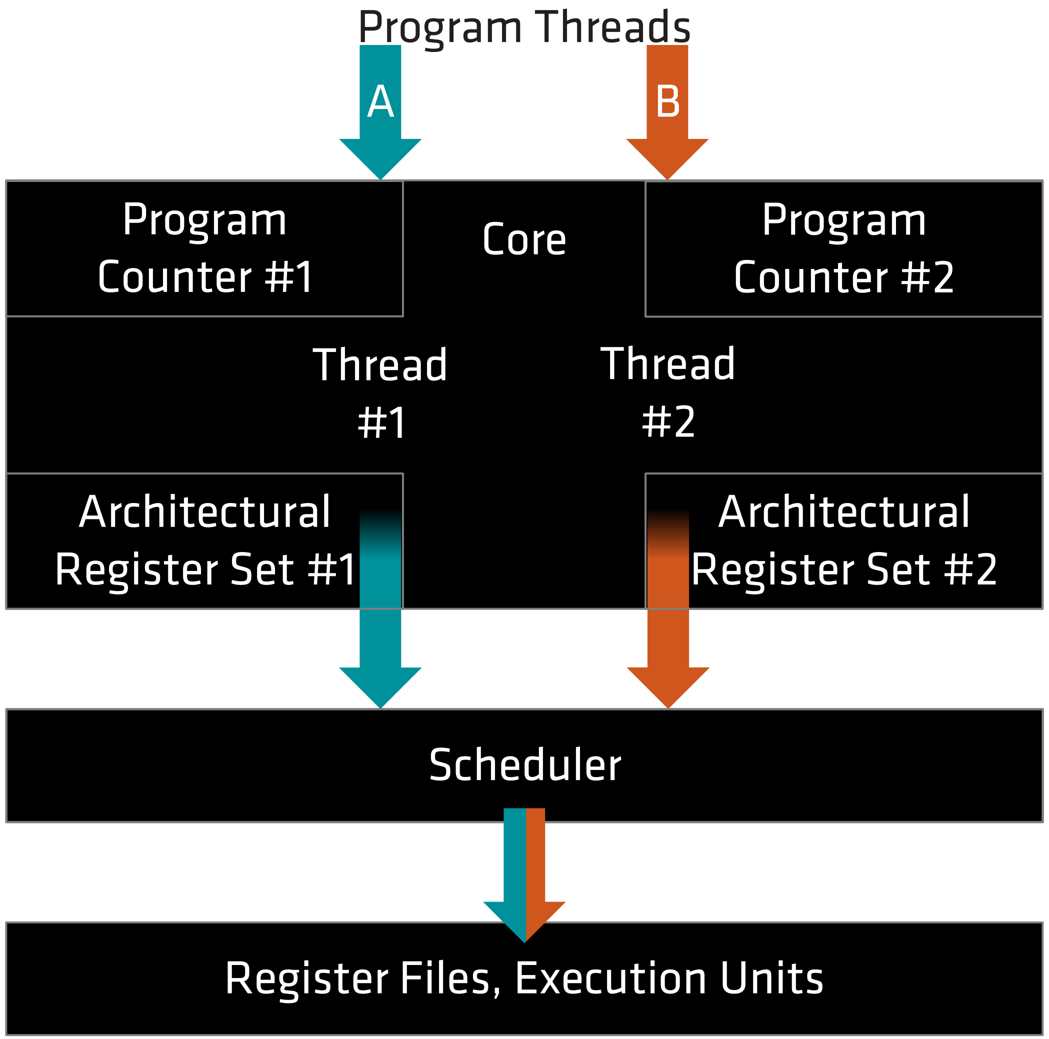
MICROARCHITECTURE

“ZEN 3”



- +19% IPC Improvement
  - Unified 8-Core CCD
- 32MB L3$ per CCD
- Improved Load Store Unit
- Wider FP & Int
- New Instructions
- Improved SMT fairness

SIMULTANEOUS MULTI-THREADING


Program Threads
A
B
Program Counter #1
Core
Program Counter #2
Thread #1
Thread #2
Architectural Register Set #1
Architectural Register Set #2
Scheduler
Register Files, Execution Units

- High performance cores have gaps in utilization which may be filled by additional hardware threads—this is Simultaneous Multi-Threading (SMT)

- Although each hardware thread has its own program counter and architectural register set, they share core resources

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.

CORE RESOURCE SHARING EVOLUTION

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

DESKTOP CACHE HIERARCHY EVOLUTION

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

*excluding products with AMD 3D V-Cache™ technology.

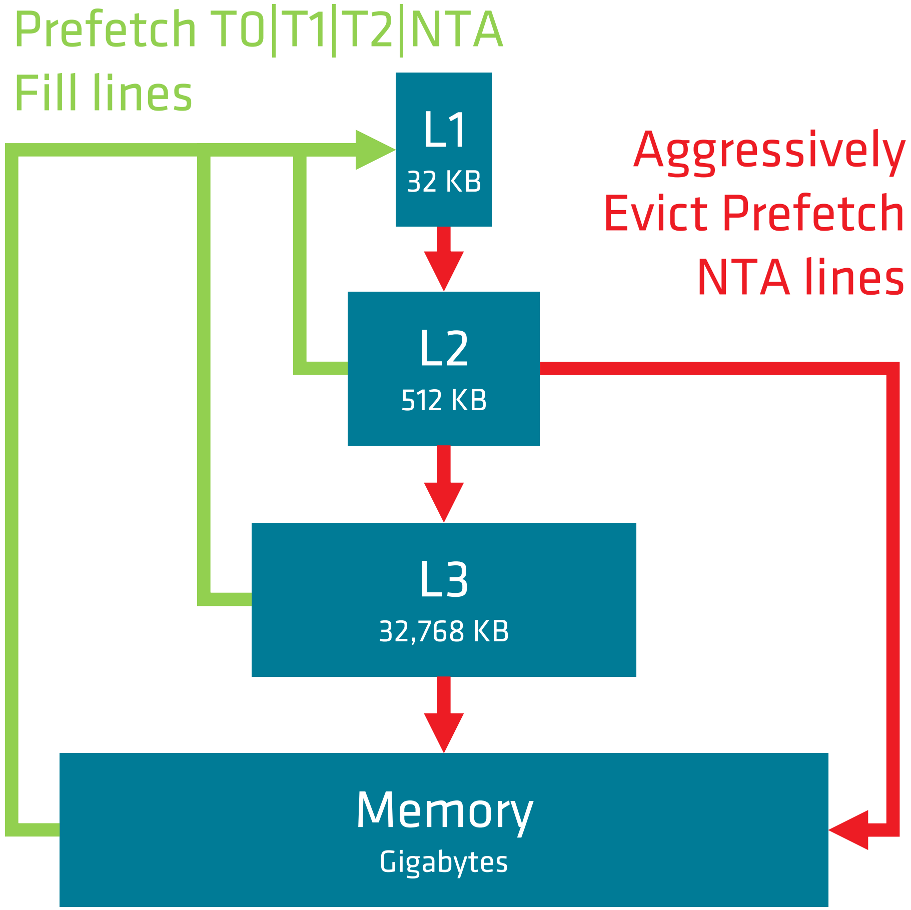
INSTRUCTION SET EVOLUTION

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

SOFTWARE PREFETCH INSTRUCTIONS

- Load a cache line from the specified memory address into the data-cache level specified by the locality reference hint T0, T1, T2, or NTA.

- Lines filled into the L2 cache with PREFETCHNTA are marked for quicker eviction from the L2, and when evicted from the L2 are not inserted into the L3.



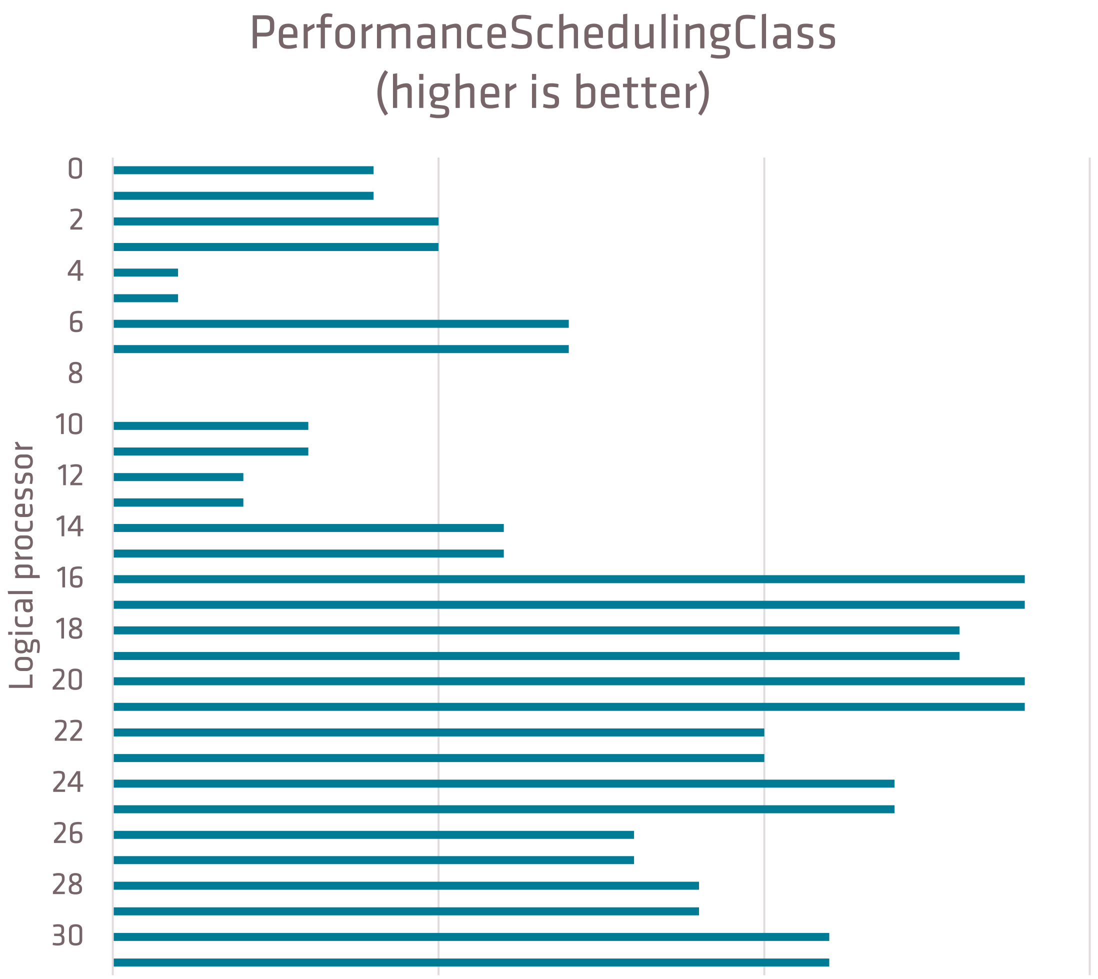
HARDWARE PREFETCHERS L1

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.

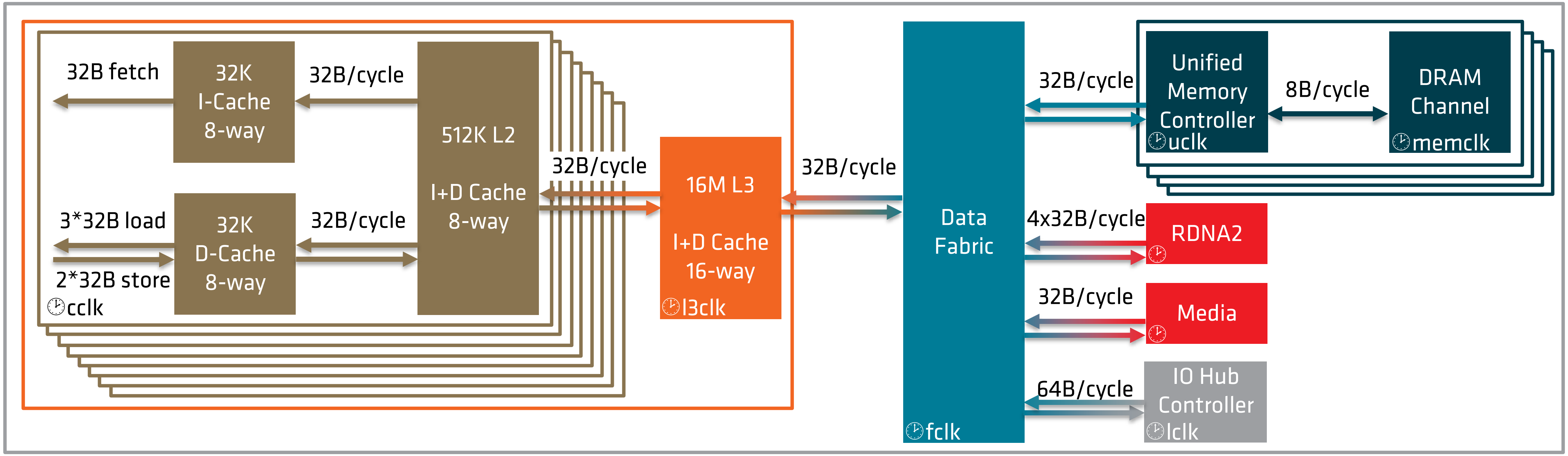
AMD PREFERRED CORE



- Some AMD products have cores which are faster than other cores.
- The system BIOS describes the CPPC Highest Performance ranking for each logical processor.
- The Windows Kernel creates a PerformanceSchedulingClass ranking based on this information and uses it during scheduling.
- Logical processor 0 and CCD0 may not be the fastest.
- Testing done by AMD performance labs February 12, 2022 on an AMD reference motherboard equipped with 16GB DDR4-3200MHz, Ryzen™ 9 5950X with Radeon™ RX 6900 XT, Win11 Pro x64 22000.493. Hypothetic example shown. Actual results may vary.

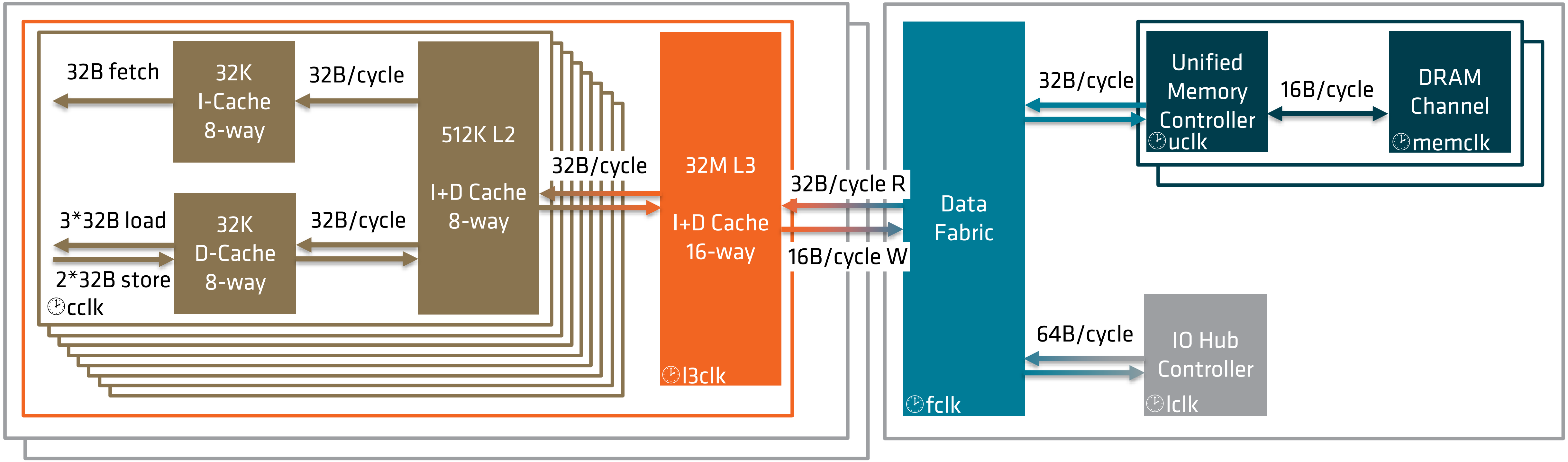
DATA FLOW

AMD RYZEN™ 7 6800U MOBILE PROCESSOR



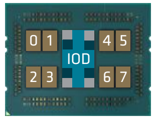
- AMD Ryzen™ 7 6800U, 15W TDP, 8 Cores, 16 Threads, up to 4.7 GHz max boost clock, 2.7 GHz base clock, integrated GPU.

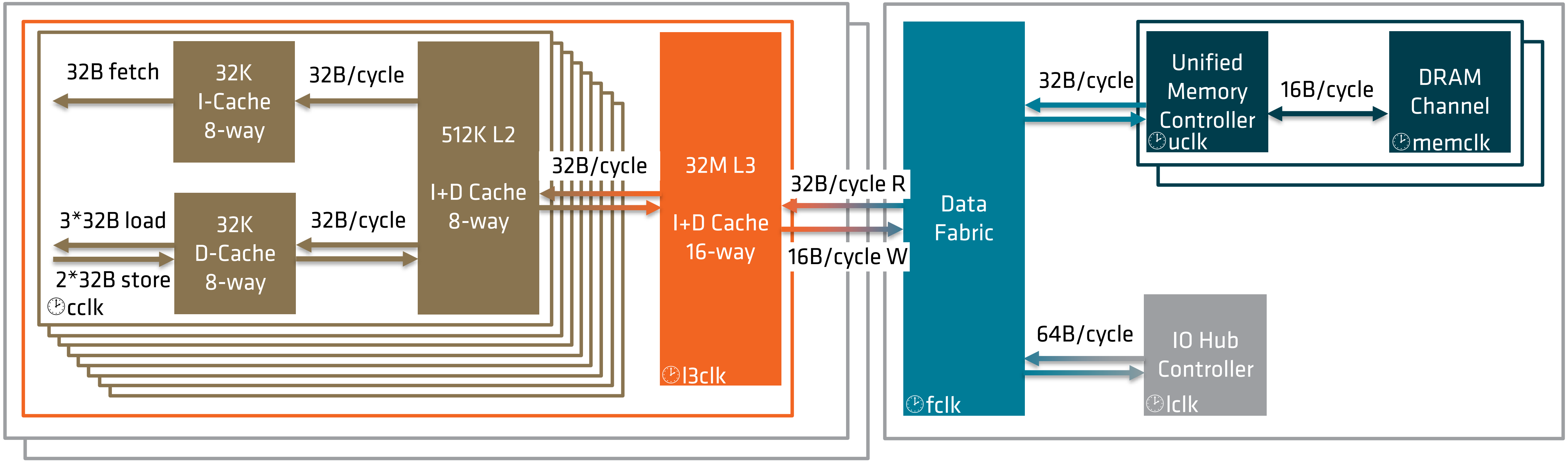
AMD RYZEN™ 9 5950X DESKTOP PROCESSOR



- AMD Ryzen™ 9 5950X, 105W TDP, 16 Cores, 32 Threads, up to 4.9 GHz max boost clock, 3.4 GHz base clock.
- Two Core Complex Die (CCD). Each CCD has one 32M L3 Cache Cluster.

AMD RYZEN™ THREADRIPPER™ PRO 5995WX PROCESSOR

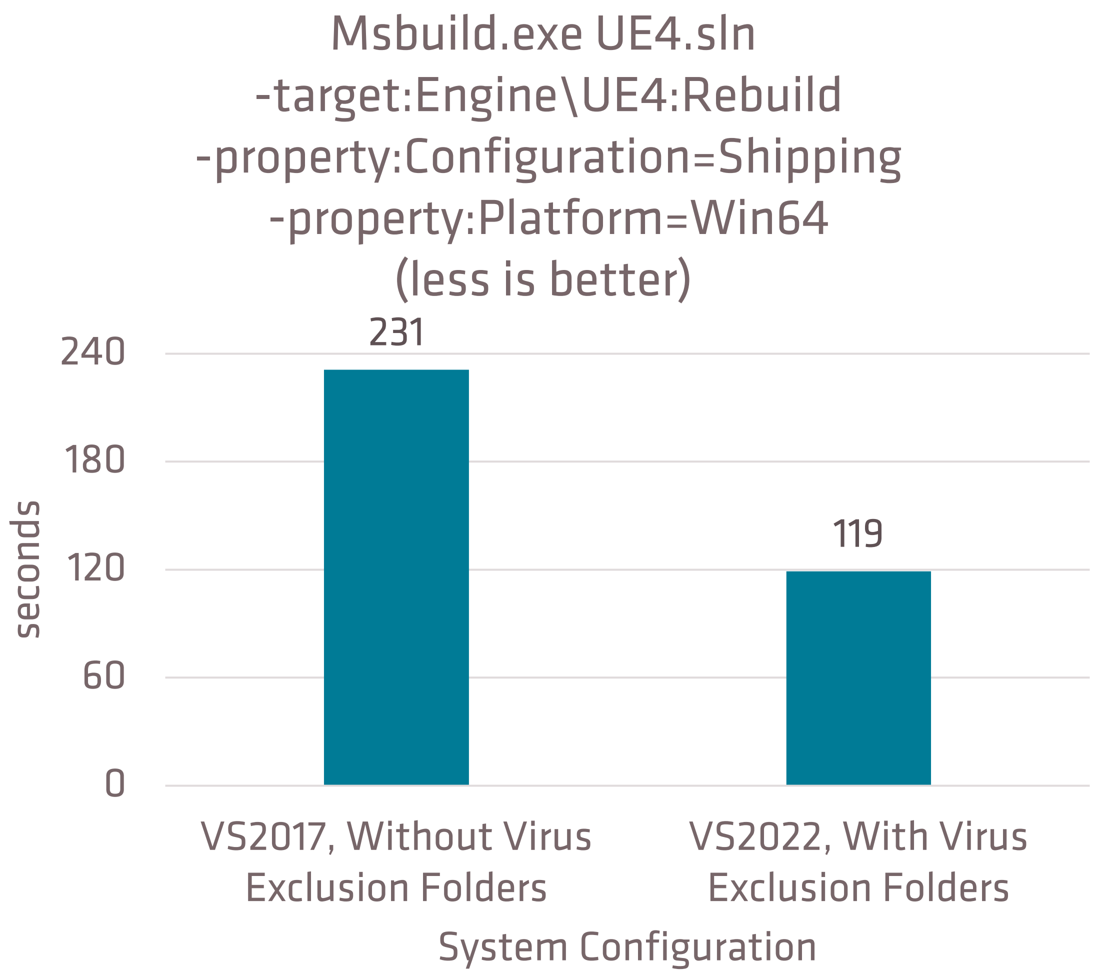




- AMD Ryzen™ Threadripper™ Pro 5995WX, 280W TDP, 64 Cores, 128 Threads, up to 4.5 GHz boost, 2.7 GHz base.
- Two CCDs per Data Fabric Quadrant shown.

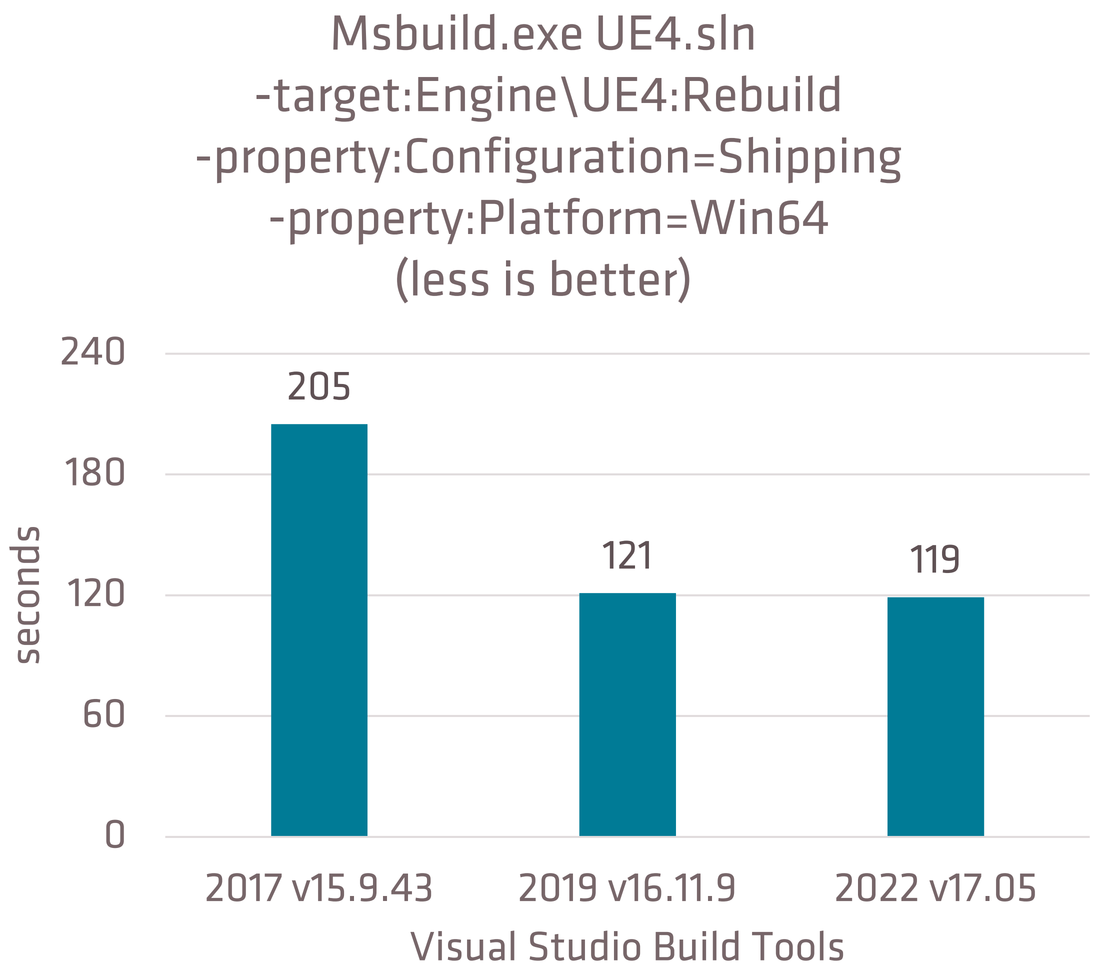
BEST PRACTICES

REDUCE BUILD TIMES



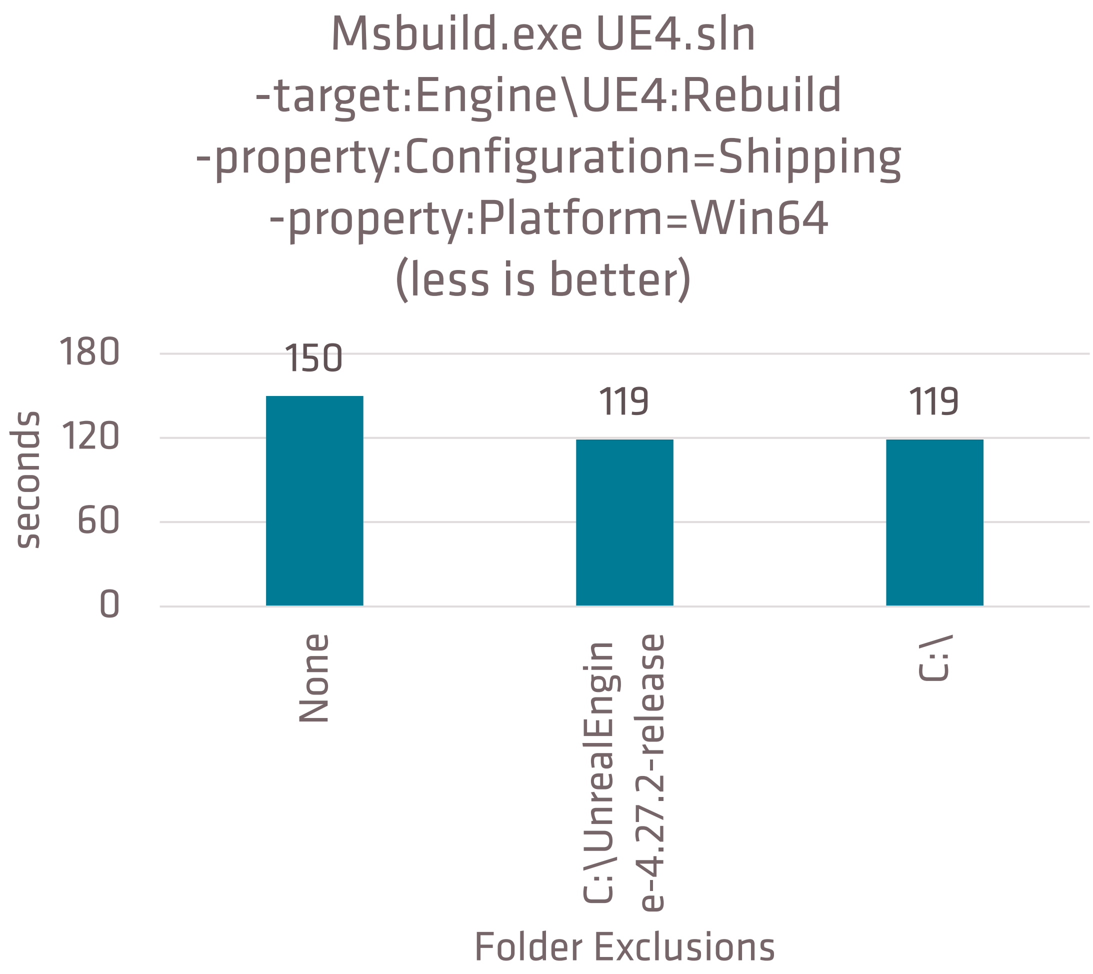
- Performance of UE4.27.2 binaries compiled with Microsoft Visual Studio.
- Testing done by AMD technology labs, February 5, 2022 on the following system. Test configuration: AMD Ryzen™ Threadripper™ PRO 5995WX, Enermax LIQTECH TR4 II series 360mm liquid cooler, 256GB (8 x 32GB 2R RDDR4-3200 at 24-22-22-52) 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 build 21H2, 1920x1080 resolution. Actual results may vary.

USE THE LATEST COMPILER AND WINDOWS® SDK



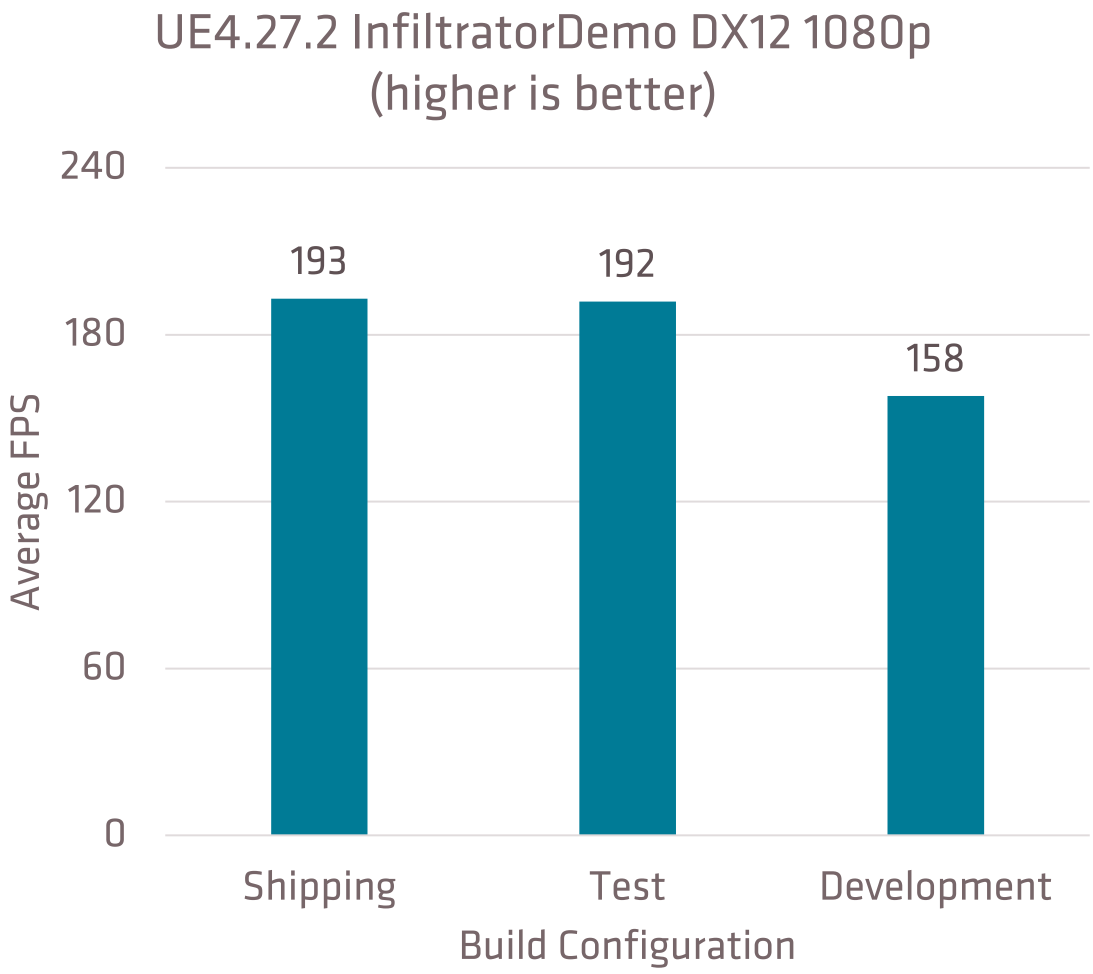
- Get the latest build and link time improvements.
- Get the latest library and runtime optimizations.
- Performance of UE4.27.2 binaries compiled with Microsoft Visual Studio.
- Testing done by AMD technology labs, February 5, 2022 on the following system. Test configuration: AMD Ryzen™ Threadripper™ 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 build 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 UE4.27.2 binaries compiled with Microsoft Visual Studio 2022 v17.0.5
- 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 build 21H2, 1920x1080 resolution. Actual results may vary.

PREFER SHIPPING CONFIGURATION BUILDS FOR CPU PROFILING



- Debug and development builds may greatly reduce performance.
  - Stats collection may cause cache pollution.
  - Logging may create serialization points.
  - Debug builds may disable multi-threading optimizations.
- While investigating open issues, developers may submit change requests which enable debug features on Test and Shipping configurations. Be sure to disable debug features before you ship!
- Performance of UE4.27.2 binaries compiled with Microsoft Visual Studio 2022 v17.0.5
- 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 build 21H2, 1920x1080 resolution. Actual results may vary.

DISABLE ANTI-TAMPER WHILE CPU PROFILING

- Build a binary similar-to Shipping configuration but without Anti-Tamper or Anti-Cheat which may prevent CPU profiling tools from properly loading symbols.

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 UE4Editor 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 the scene far exceeds its draw budget or has many duplicate objects, consider reporting the issue to its artists and profiling a different scene. Otherwise, you may risk profiling hot spots which may not be hot after the art issues are resolved.

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\DxCache"
rmdir /s /q "%LOCALAPPDATA%\AMD\GLCache"
rmdir /s /q "%LOCALAPPDATA%\AMD\VkCache"
rmdir /s /q "%ProgramData%\NVIDIA Corporation\NV_Cache"
rmdir /s /q "%ProgramFiles(x86)%\Steam\steamapps\shadercache"
```

OPTIMIZATIONS

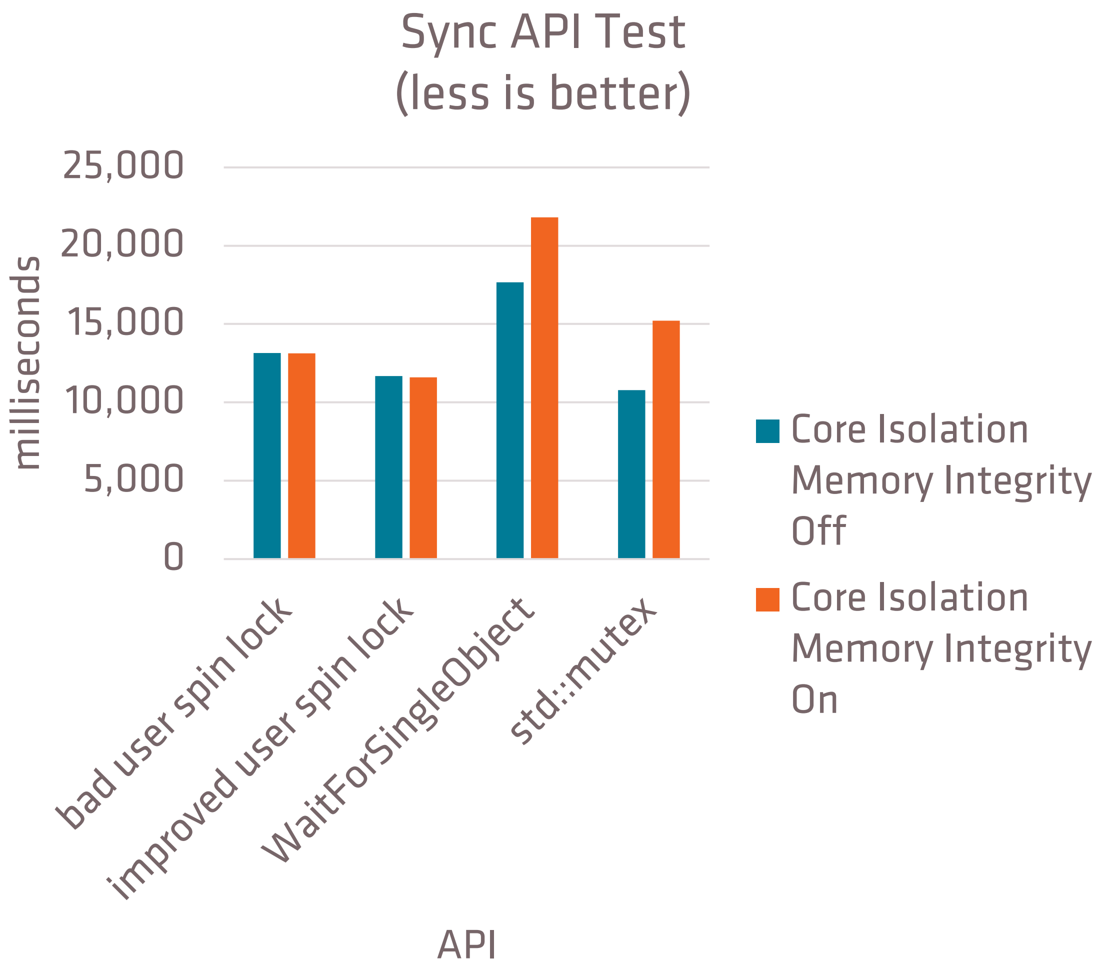
TOPICS

- Use the AMD Core Counts Sample
- Use Modern Sync APIs
- Avoid False Sharing
- Prefer data access patterns matching hardware prefetcher behaviors
- Use Software Prefetch instructions for linked data structures experiencing cache misses
- Align Memcpy source and destination pointers
- Avoid Penalties while mixing SSE and AVX instructions
- Support Hybrid Graphics
- Use Preferred Video and Audio Codecs

USE THE AMD CORE COUNTS SAMPLE

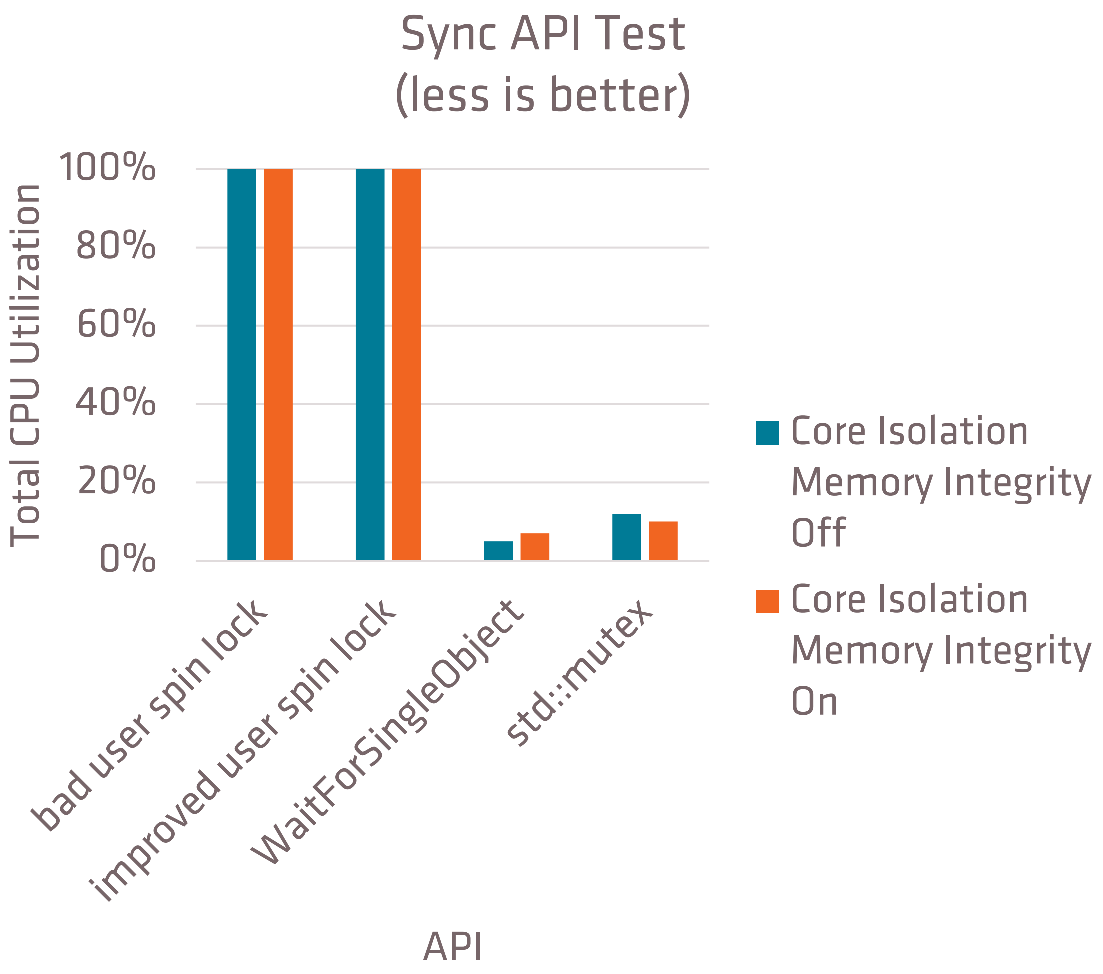
- *This advice is specific to AMD processors and is not general guidance for all processor vendors*
- Many applications show SMT benefits and use of all logical processors is recommended
- However, games often suffer from SMT and cache contention on the main or render threads during gameplay
- Creating the thread pool based on physical core count rather than logical processor count may reduce this contention
- Profile your game to determine the ideal thread count
  - Game initialization—including decompressing assets and compiling/warming shaders—may benefit from logical processors using SMT dual-thread mode
  - Game play may prefer physical core count using SMT single-thread mode
- See https://gpuopen.com/learn/cpu-core-counts/

USE MODERN SYNC APIS



- Prefer std::mutex which has good performance and low cpu utilization.
- Performance of binaries compiled with Microsoft Visual Studio 2022 v17.0.4.
- Testing done by AMD technology labs, January 3, 2022 on the following system. Test configuration: AMD Ryzen™ 5950X, NZXT Kraken X62 cooler, 16GB (2 x 8GB DDR4-3600 16-16-16-36) memory, AMD Radeon™ RX 6900 XT GPU with driver 21.11.2 (November 11, 2021), 2TB M.2 NVME SSD, AMD Reference Motherboard, Windows® 11 x64 build 21H2, 1920x1080 resolution. Actual results may vary.

USE MODERN SYNC APIS



- Prefer std::mutex which has good performance and low cpu utilization.
- Performance of binaries compiled with Microsoft Visual Studio 2022 v17.0.4.
- Testing done by AMD technology labs, January 3, 2022 on the following system. Test configuration: AMD Ryzen™ 5950X, NZXT Kraken X62 cooler, 16GB (2 x 8GB DDR4-3600 16-16-16-36) memory, AMD Radeon™ RX 6900 XT GPU with driver 21.11.2 (November 11, 2021), 2TB M.2 NVME SSD, AMD Reference Motherboard, Windows® 11 x64 build 21H2, 1920x1080 resolution. Actual results may vary.

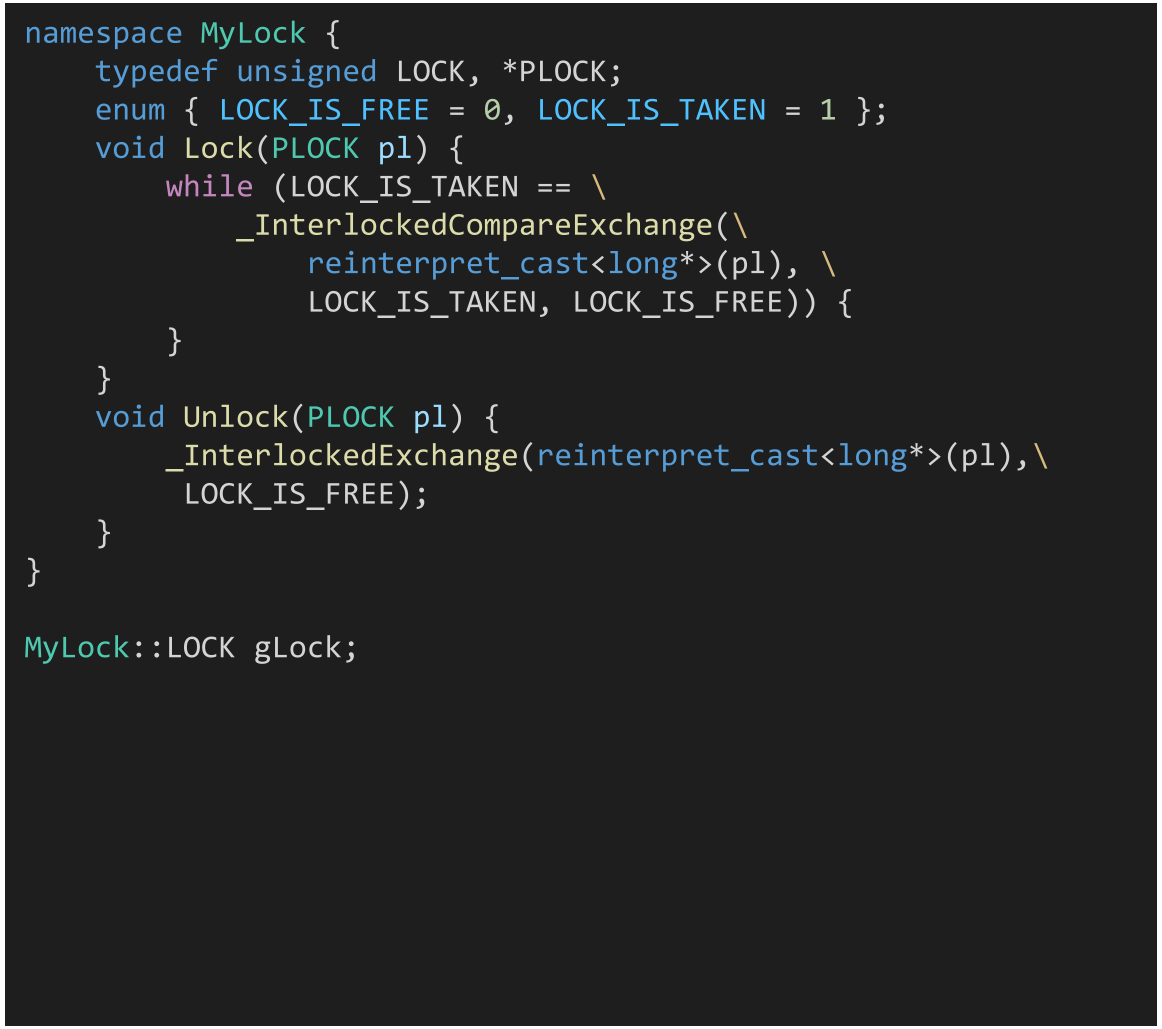
USE MODERN SYNC APIS: SHARED CODE

```
#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];
```

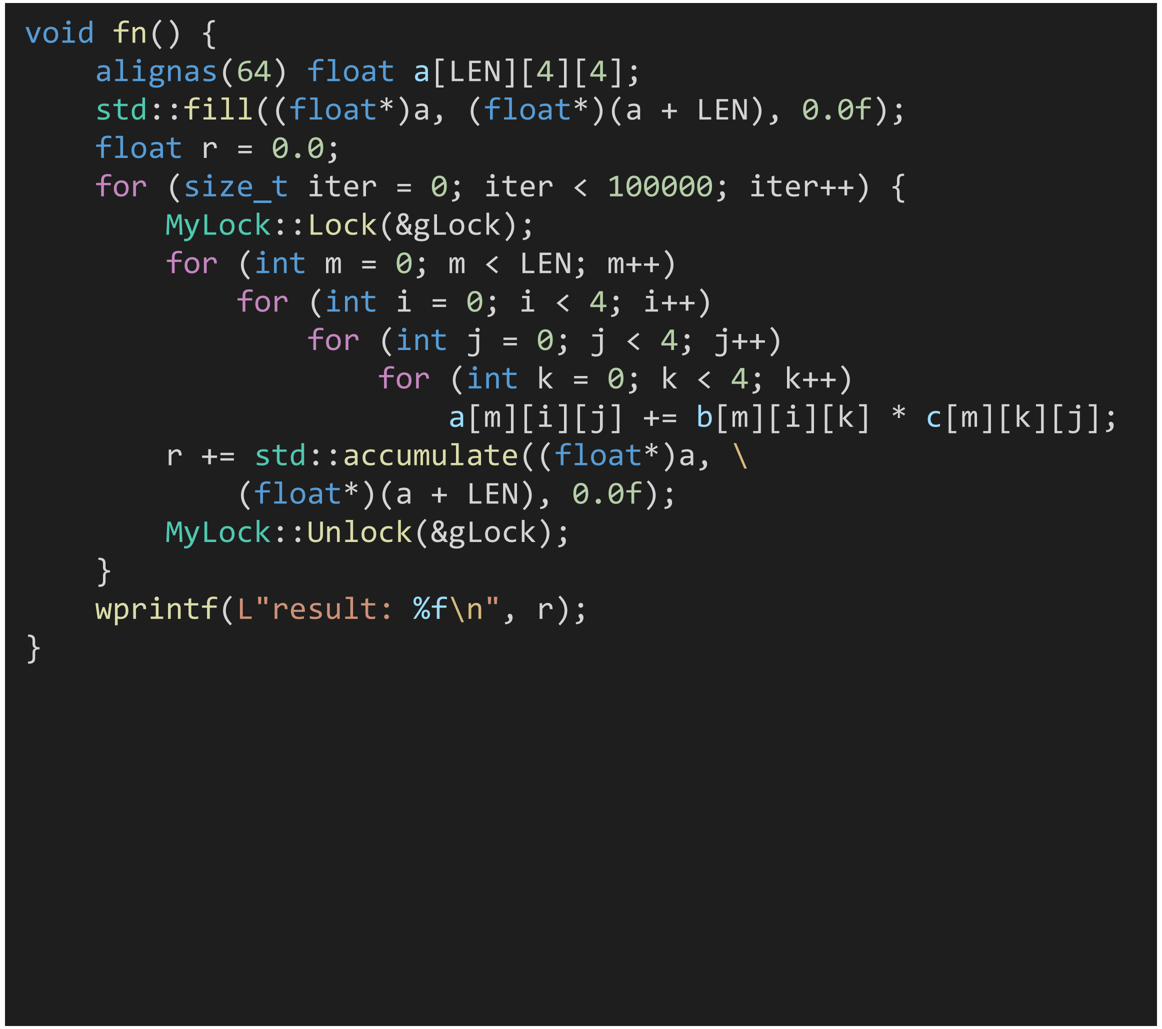
```
int main(int argc, char* argv[]) {
    using namespace std::chrono;
    float b0 = (argc > 1) ? strtof(argv[1], NULL) : 1.0f;
    float c0 = (argc > 2) ? strtof(argv[2], NULL) : 2.0f;
    std::fill((float*)b, (float*)(b + LEN), b0);
    std::fill((float*)c, (float*)(c + LEN), c0);
    int num_threads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

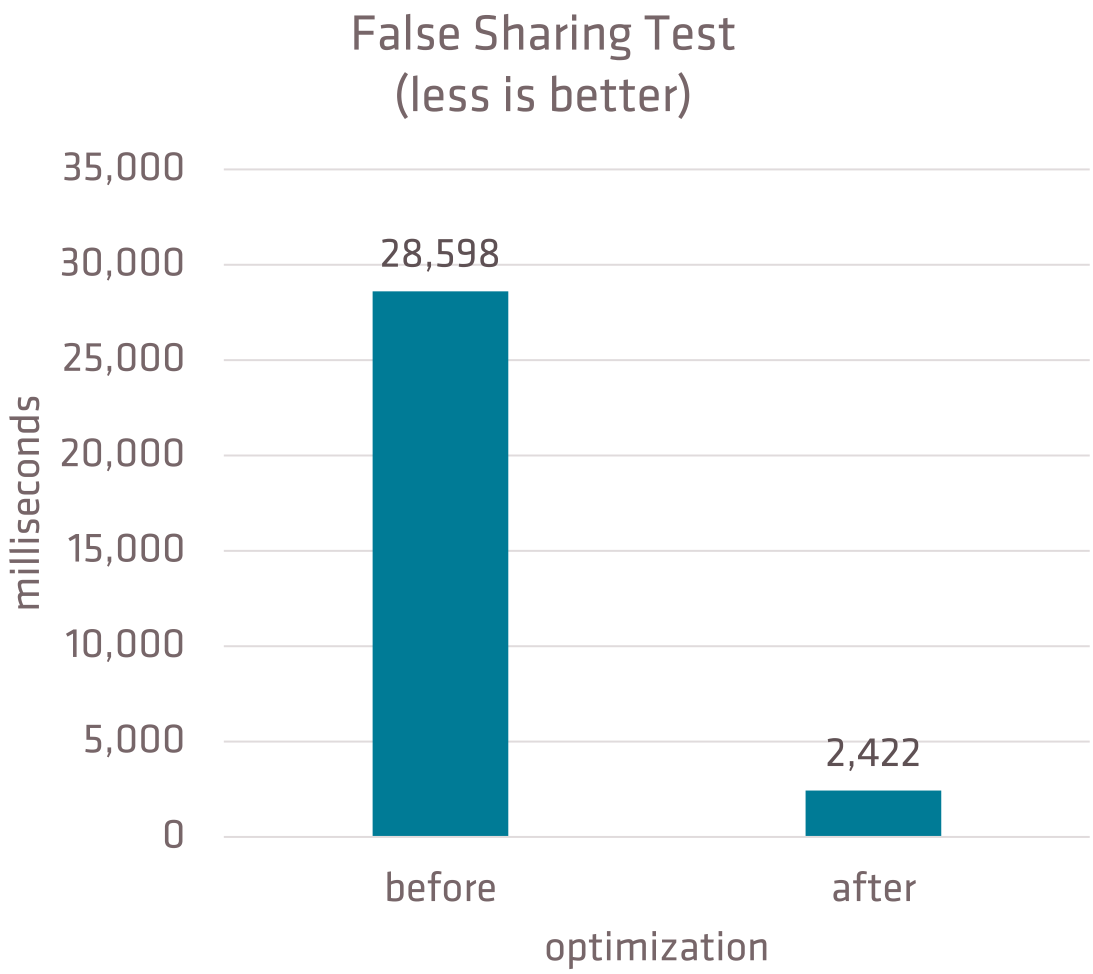
```
void fn() {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
        WaitForSingleObject(hMutex, INFINITE);
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                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;
```

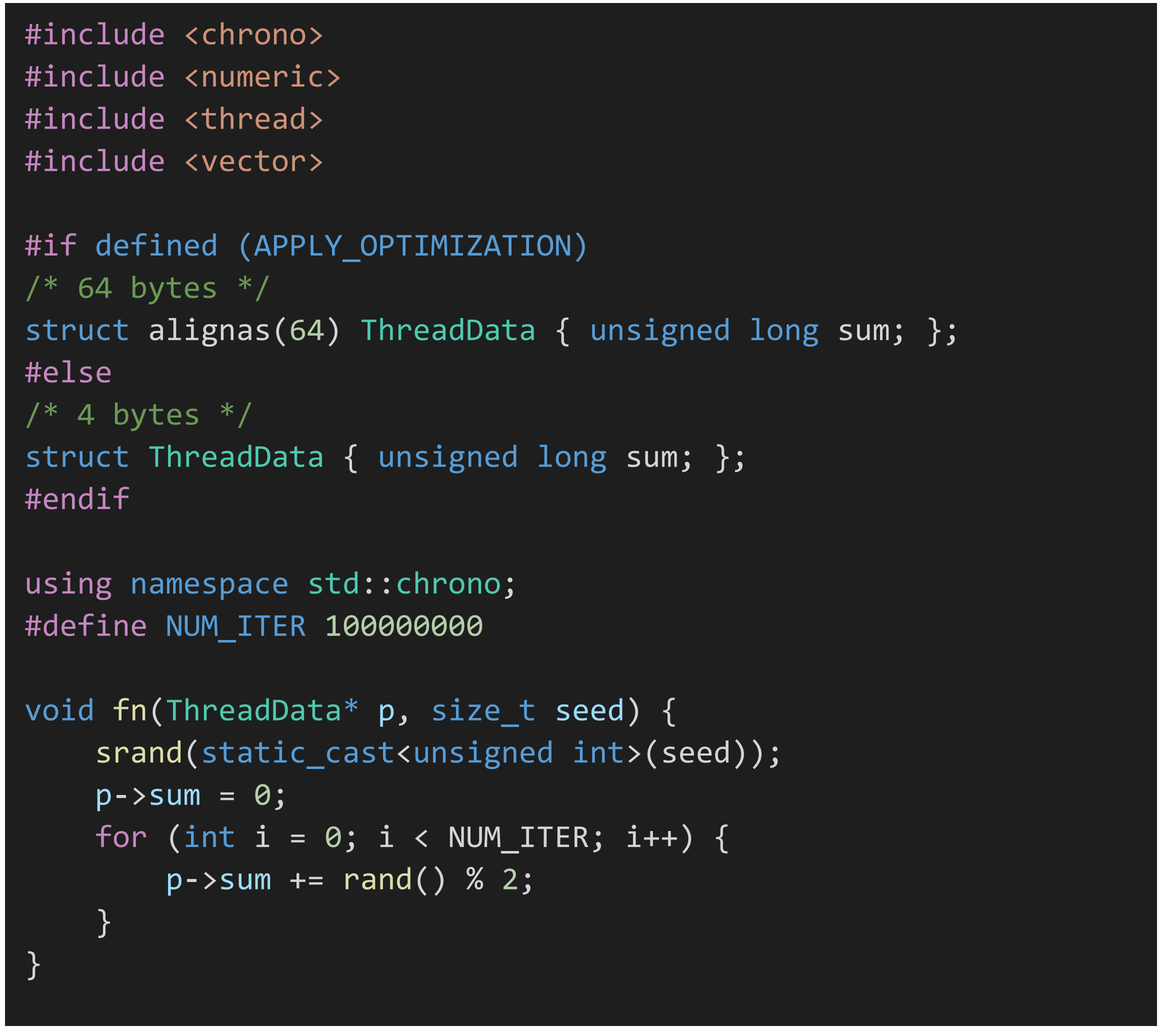
```
void fn() {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
       mutex.lock();
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j = 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);
}
```

AVOID FALSE SHARING



- Reduced execution time by 90% after optimization!
- Performance of binaries compiled with Microsoft Visual Studio 2022 v17.0.5.
- 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 build 21H2, 1920x1080 resolution. Actual results may vary.

AVOID FALSE SHARING

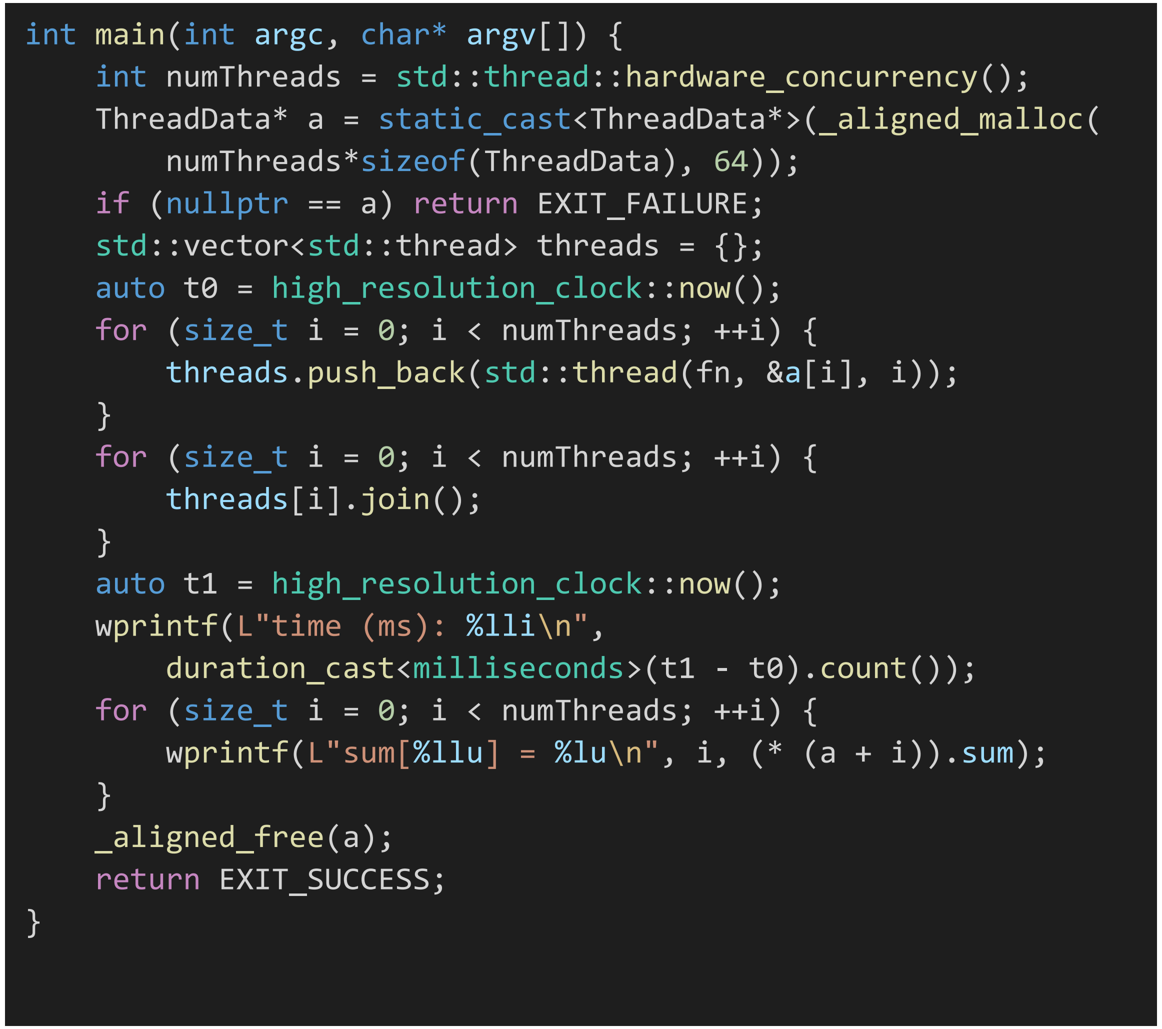


```cpp
#include <chrono>
#include <numeric>
#include <thread>
#include <vector>

#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;
    }
}
```



```cpp
int main(int argc, char* argv[]) {
    int numThreads = std::thread::hardware_concurrency();
    ThreadData* a = static_cast<ThreadData*>(_aligned_malloc(
        numThreads*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 < numThreads; ++i) {
        threads.push_back(std::thread(fn, &a[i], i));
    }
    for (size_t i = 0; i < numThreads; ++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 < numThreads; ++i) {
        wprintf(L"sum[%llu] = %lu\n", i, (* (a + i)).sum);
    }
    _aligned_free(a);
    return EXIT_SUCCESS;
}
```

PREFER DATA ACCESS PATTERNS MATCHING HARDWARE PREFETCHER BEHAVIORS

Streaming

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	840	880	8C0	900	940	980	9C0	A00	A40	A80	AC0	B00
Stream +1																									1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Stream -1	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1																				

Stride

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	840	880	8C0	900	940	980	9C0	A00	A40	A80	AC0	B00
Stride +2																									1		2		3		4		5		6		7		8		9		10		11
Stride -3	9			8			7			6			5			4			3			2			1																				

STREAMING HARDWARE PREFETCHER

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	840	880	8C0	900	940	980	9C0	A00	A40	A80	AC0	B00
Stream +1																									1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

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

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

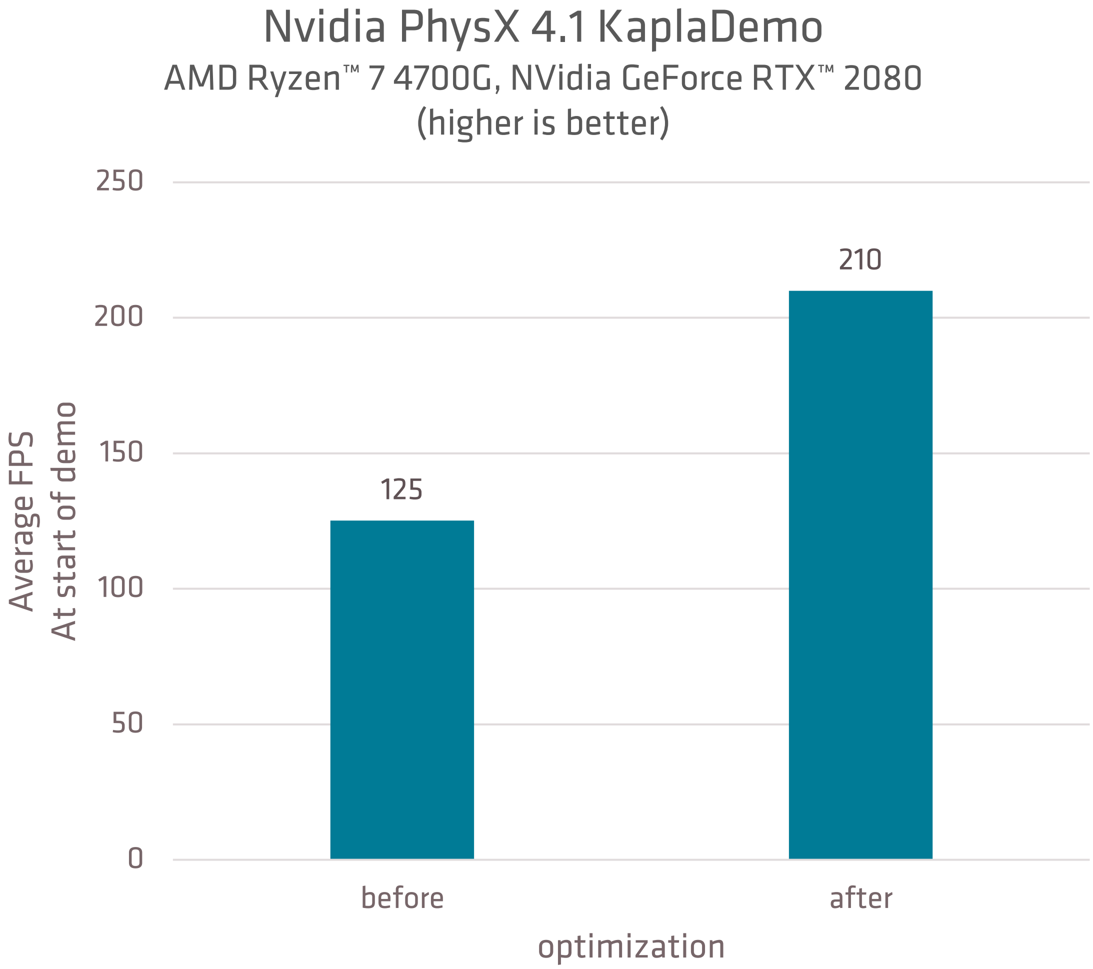
STRIDE HARDWARE PREFETCHER

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	840	880	8C0	900	940	980	9C0	A00	A40	A80	AC0	B00
Stride +5																									1					2					3					4					5
Stride +5																													1					2					3					4	

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

```
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
}
```

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…

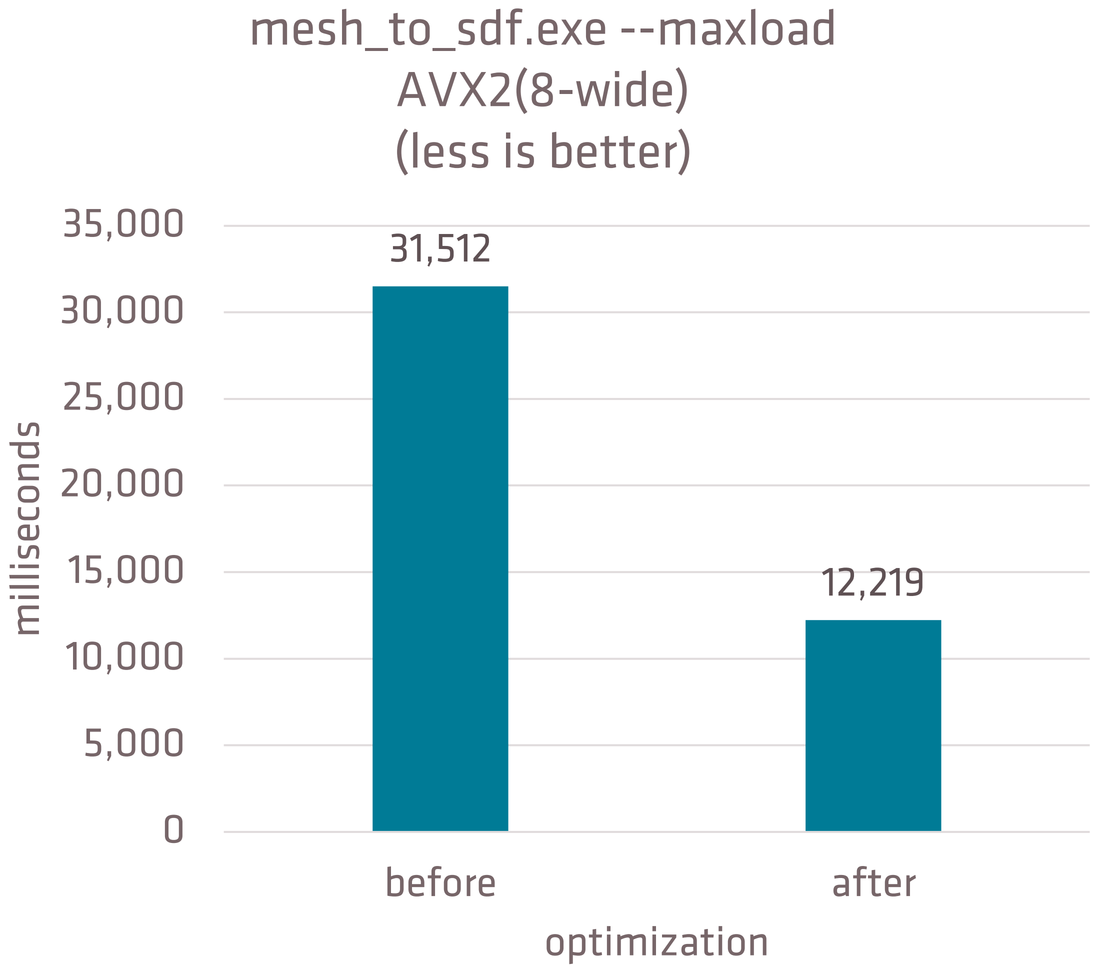
```
// Copyright (c) 2021 NVIDIA Corporation. All rights reserved
// ConvexRenderer.cpp from https://github.com/NVIDIAGameWorks/PhysX/tree/4.1/physx
void ConvexRenderer::updateTransformations()
{
 for (int i = 0; i < (int)mGroups.size(); i++) {
  ConvexGroup *g = mGroups[i];
  if (g->texCoords.empty())
   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);
 }
}
```

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.
- 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.
  - The processor uses linear address bits 0 thru 11 to determine Store-To-Load-Forward eligibility.
  - 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.
- Alignas(4096) may benefit probe filtering on AMD Threadripper™ and EPYC™ processors.
- Aligning to the bit_floor may provide a good balance of cache hits and alignment:
  - std::clamp(std::bit_floor(count), 4, 4096);

AVOID PENALTIES WHILE MIXING SSE AND AVX INSTRUCTIONS



- 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 60% after VZeroUpper optimization.
- Performance of binaries compiled with Microsoft Visual Studio 2022 v17.0.5.
- 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 build 21H2, 1920x1080 resolution. Actual results may vary.

AVOID PENALTY FOR MIXING SSE AND AVX INSTRUCTIONS

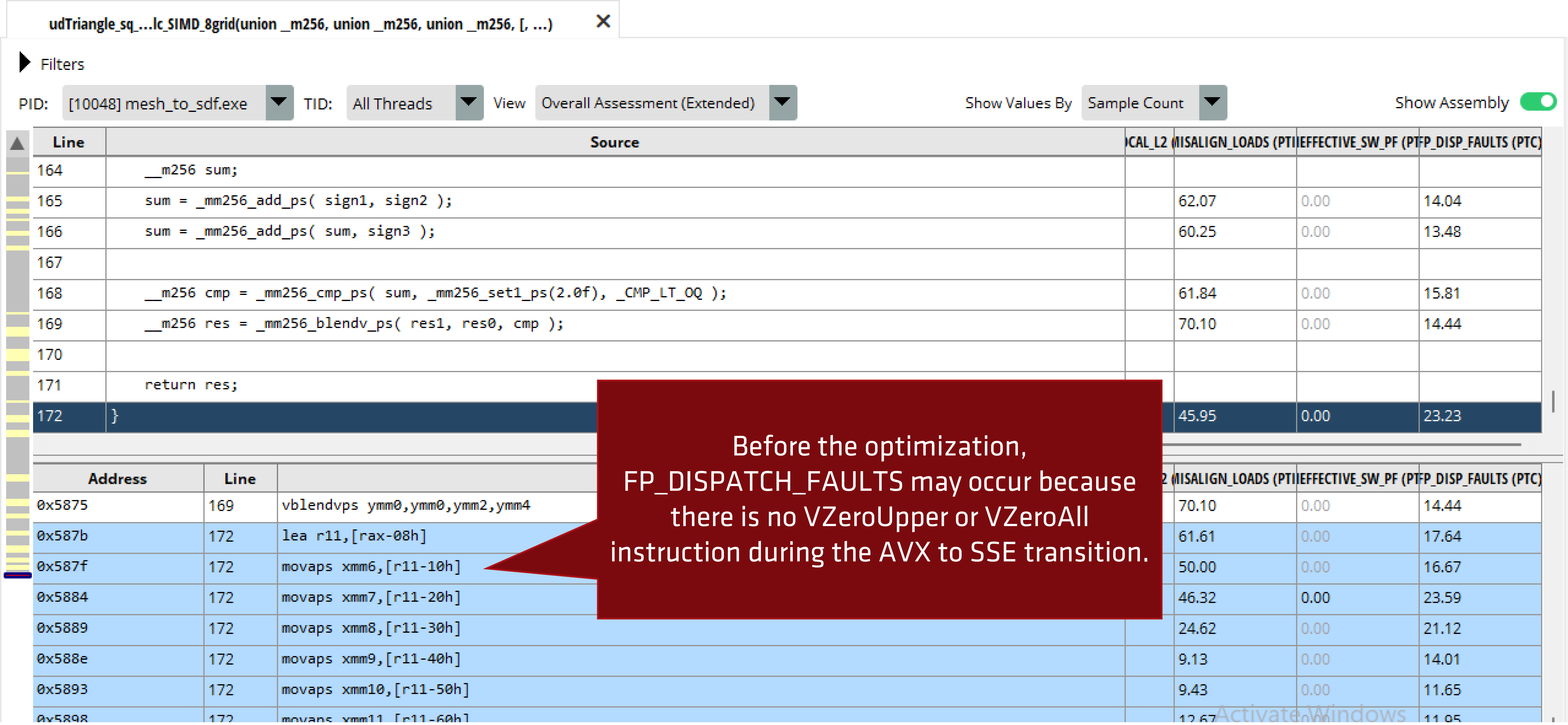
- Use PMCx00E Floating Point Dispatch Faults > 0 to find code which may be missing VZeroUpper or VZeroAll instructions during AVX to SSE and SSE to AVX transitions.
- Optimization 1:
  - Use the /arch:AVX compiler flag.
  - AVX is supported by 94% of users according to the January 2022 Steam Hardware & 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 );
}
```

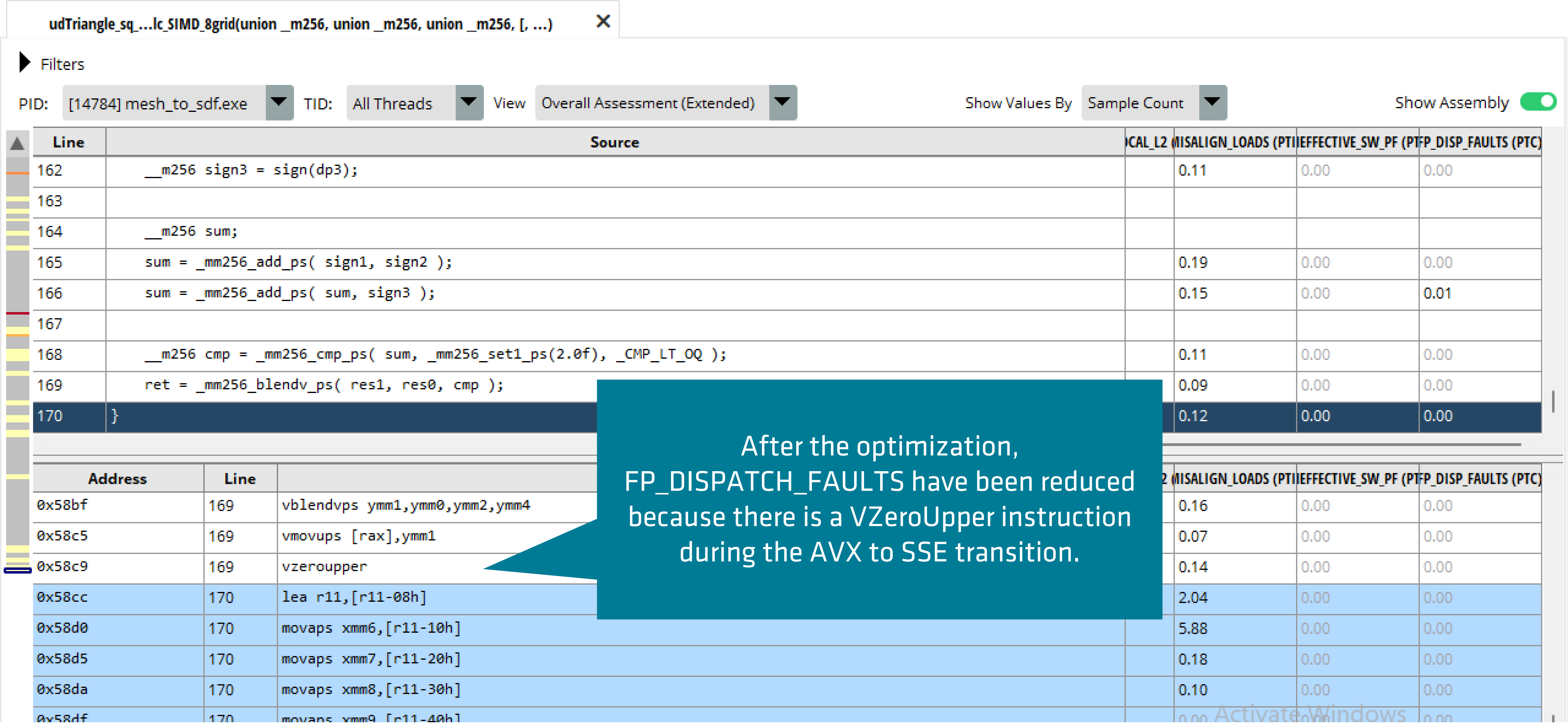

udTriangle_sq_...lc_SIMD_8grid(union _m256, union _m256, union _m256, [, ...)
✕
Filters

PID: [10048] mesh_to_sdf.exe TID: All Threads View Overall Assessment (Extended) Show Values By Sample Count Show Assembly

Line	Source	)CAL_L2	MISALIGN_LOADS (PTI	EFFECTIVE_SW_PF (PT	FP_DISP_FAULTS (PTC)
164	`__m256 sum;`				
165	`sum = _mm256_add_ps( sign1, sign2 );`		62.07	0.00	14.04
166	`sum = _mm256_add_ps( sum, sign3 );`		60.25	0.00	13.48
167					
168	`__m256 cmp = _mm256_cmp_ps( sum, _mm256_set1_ps(2.0f), _CMP_LT_OQ );`		61.84	0.00	15.81
169	`__m256 res = _mm256_blendv_ps( res1, res0, cmp );`		70.10	0.00	14.44
170					
171	`return res;`				
172	`}`		45.95	0.00	23.23

Address	Line		2	MISALIGN_LOADS (PTI	EFFECTIVE_SW_PF (PT	FP_DISP_FAULTS (PTC)
0x5875	169	`vblendvps ymm0,ymm0,ymm2,ymm4`		70.10	0.00	14.44
0x587b	172	`lea r11,[rax-08h]`		61.61	0.00	17.64
0x587f	172	`movaps xmm6,[r11-10h]`		50.00	0.00	16.67
0x5884	172	`movaps xmm7,[r11-20h]`		46.32	0.00	23.59
0x5889	172	`movaps xmm8,[r11-30h]`		24.62	0.00	21.12
0x588e	172	`movaps xmm9,[r11-40h]`		9.13	0.00	14.01
0x5893	172	`movaps xmm10,[r11-50h]`		9.43	0.00	11.65
0x5898	172	`movaps xmm11,[r11-60h]`		12.67		11.95

Before the optimization, FP_DISPATCH_FAULTS may occur because there is no VZeroUpper or VZeroAll instruction during the AVX to SSE transition.



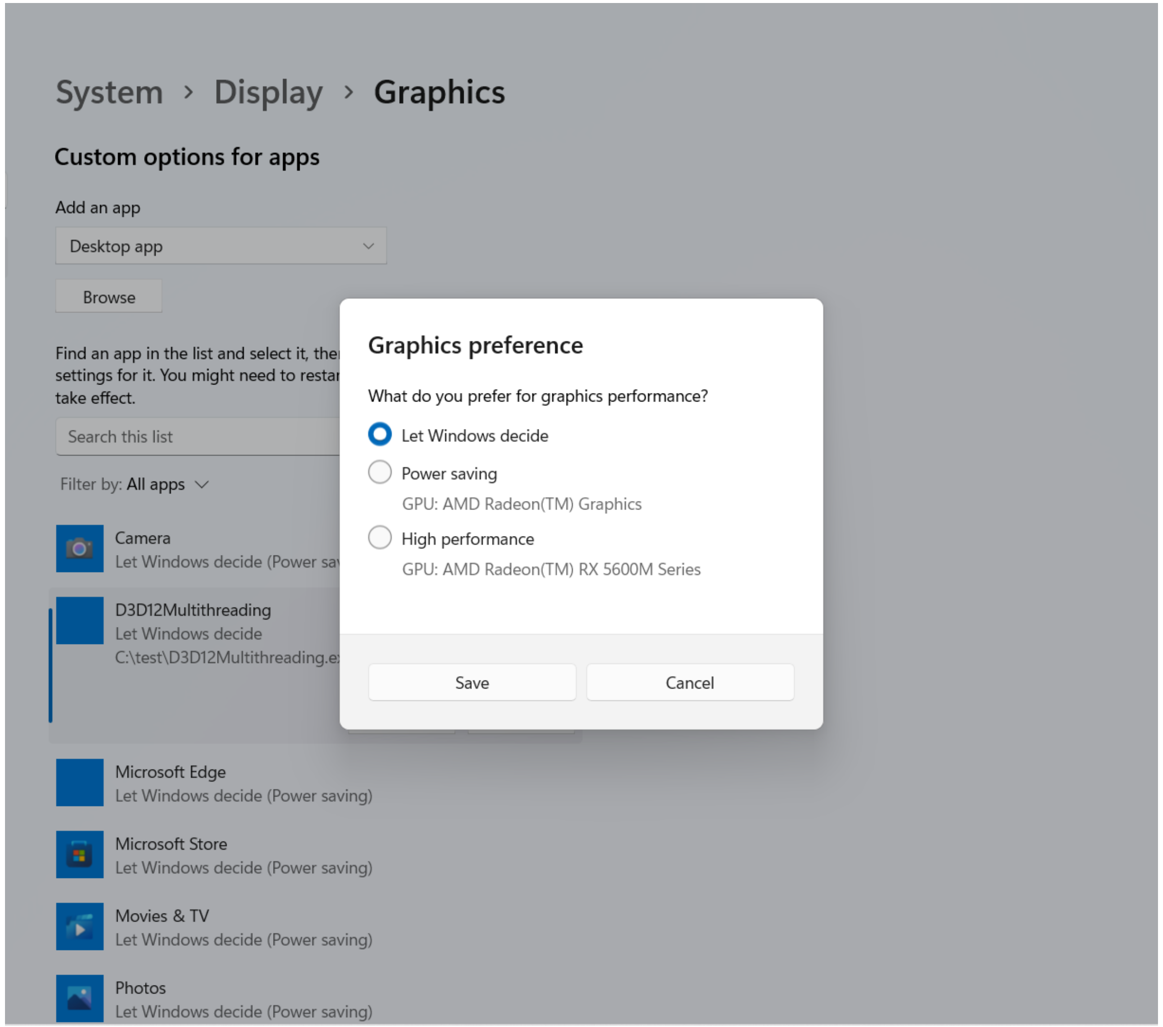
udTriangle_sq_...lc_SIMD_8grid(union _m256, union _m256, union _m256, [, ...)
Filters
PID: [14784] mesh_to_sdf.exe
TID: All Threads
View Overall Assessment (Extended)
Show Values By Sample Count
Show Assembly

Line	Source	CAL_L2	MISALIGN_LOADS (PTI	EFFECTIVE_SW_PF (PT	FP_DISP_FAULTS (PTC)
162	`__m256 sign3 = sign(dp3);`		0.11	0.00	0.00
163					
164	`__m256 sum;`				
165	`sum = _mm256_add_ps( sign1, sign2 );`		0.19	0.00	0.00
166	`sum = _mm256_add_ps( sum, sign3 );`		0.15	0.00	0.01
167					
168	`__m256 cmp = _mm256_cmp_ps( sum, _mm256_set1_ps(2.0f), _CMP_LT_OQ );`		0.11	0.00	0.00
169	`ret = _mm256_blendv_ps( res1, res0, cmp );`		0.09	0.00	0.00
170	`}`		0.12	0.00	0.00

Address	Line		MISALIGN_LOADS (PTI	EFFECTIVE_SW_PF (PT	FP_DISP_FAULTS (PTC)
0x58bf	169	`vblendvps ymm1,ymm0,ymm2,ymm4`	0.16	0.00	0.00
0x58c5	169	`vmovups [rax],ymm1`	0.07	0.00	0.00
0x58c9	169	`vzeroupper`	0.14	0.00	0.00
0x58cc	170	`lea r11,[r11-08h]`	2.04	0.00	0.00
0x58d0	170	`movaps xmm6,[r11-10h]`	5.88	0.00	0.00
0x58d5	170	`movaps xmm7,[r11-20h]`	0.18	0.00	0.00
0x58da	170	`movaps xmm8,[r11-30h]`	0.10	0.00	0.00
0x58df	170	`movaps xmm9,[r11-40h]`	0.00		0.00

After the optimization, FP_DISPATCH_FAULTS have been reduced because there is a VZeroUpper instruction during the AVX to SSE transition.

SUPPORT HYBRID GRAPHICS



System › Display › Graphics

Custom options for apps

Add an app

Desktop app

Find an app in the list and select it, the
settings for it. You might need to resta
take effect.

Filter by: All apps

Camera
Let Windows decide (Power sa

D3D12Multithreading
Let Windows decide
C:\test\D3D12Multithreading.e

Microsoft Edge
Let Windows decide (Power saving)

Microsoft Store
Let Windows decide (Power saving)

Movies & TV
Let Windows decide (Power saving)

Photos
Let Windows decide (Power saving)

Graphics preference

What do you prefer for graphics performance?

● Let Windows decide

○ Power saving
GPU: AMD Radeon(TM) Graphics

○ High performance
GPU: AMD Radeon(TM) RX 5600M Series

- 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.

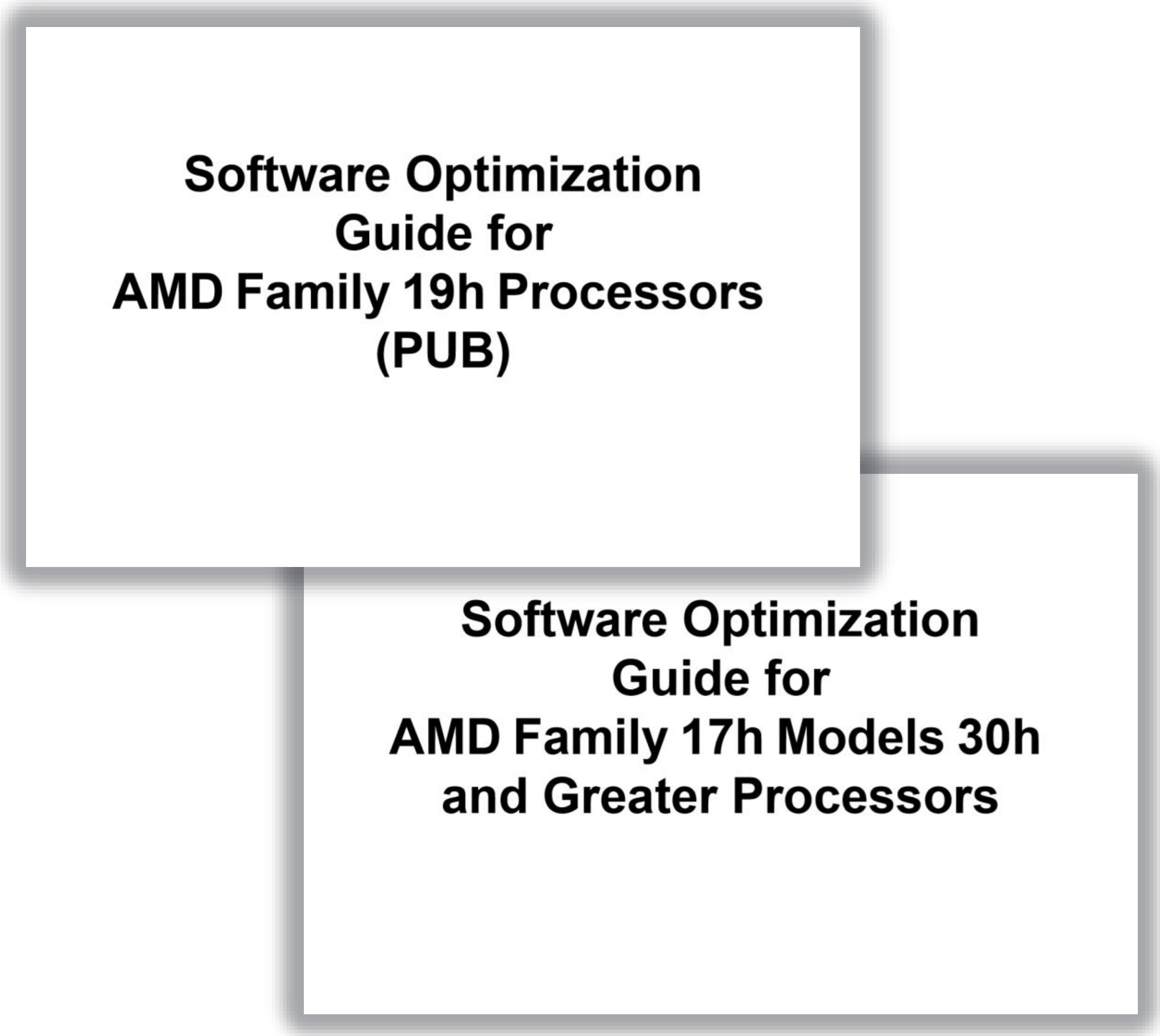
USE PREFERRED VIDEO AND AUDIO CODECS

- Prefer H264 video and AAC audio codecs as recommended by the Unreal Engine Electra Plugin.

- Hardware accelerated codecs may increase hours of battery life and reduce CPU work.

- Radeon™ RX 6500 XT and Radeon™ RX 6400 Supported Rendering Format:
  - 4K H264 Decode Yes.
  - WMV3 Decode No.
  - See amd.com for more.

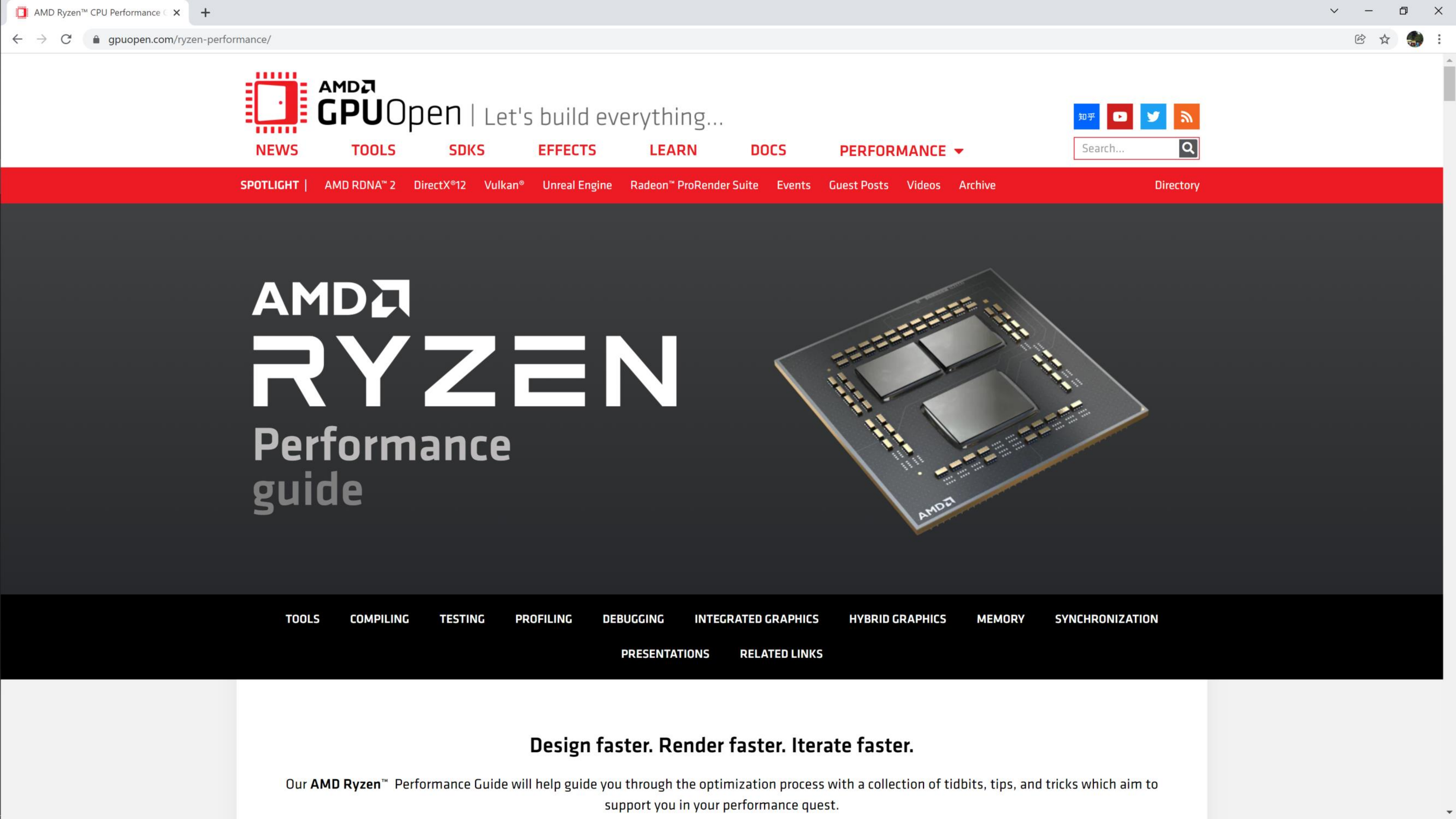
SOFTWARE OPTIMIZATION GUIDES



Software Optimization Guide for AMD Family 19h Processors (PUB)

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

- AMD Family 19h is “Zen 3”
- AMD Family 17h Models 30h is “Zen 2”
- See https://developer.amd.com/resources/developer-guides-manuals/



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.

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 48 is modified.

DISCLAIMER AND NOTICES

- Claim “Zen 3” +19% IPC uplift
  - Testing by AMD performance labs as of 09/01/2020. IPC evaluated with a selection of 25 workloads running at a locked 4GHz frequency on 8-core "Zen 2" Ryzen™ 7 3800XT and "Zen 3" Ryzen™ 7 5800X desktop processors configured with Windows® 10, NVIDIA GeForce RTX 2080 Ti (451.77), Samsung 860 Pro SSD, and 2x8GB DDR4-3600. Results may vary. R5K-003
- 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 herein are plans only and subject to change. "Zen 2" and "Zen 3" are codenames for AMD architectures, and are not product names. GD-122
- Engineering projections are not a guarantee of final performance. Performance projections by AMD engineering staff based on expected Ryzen™ Threadripper™ Pro 5000 WX series processors vs Ryzen™ Threadripper™ Pro 3000 WX series processors. Specific projections are based on reference design platforms and are subject to change when final products are released in market.

AMD