AMD EPYC

AMD RYZEN

AMD RADEON

AMD RYZEN™ PROCESSOR SOFTWARE OPTIMIZATION

JOHN HARTWIG & KEN MITCHELL

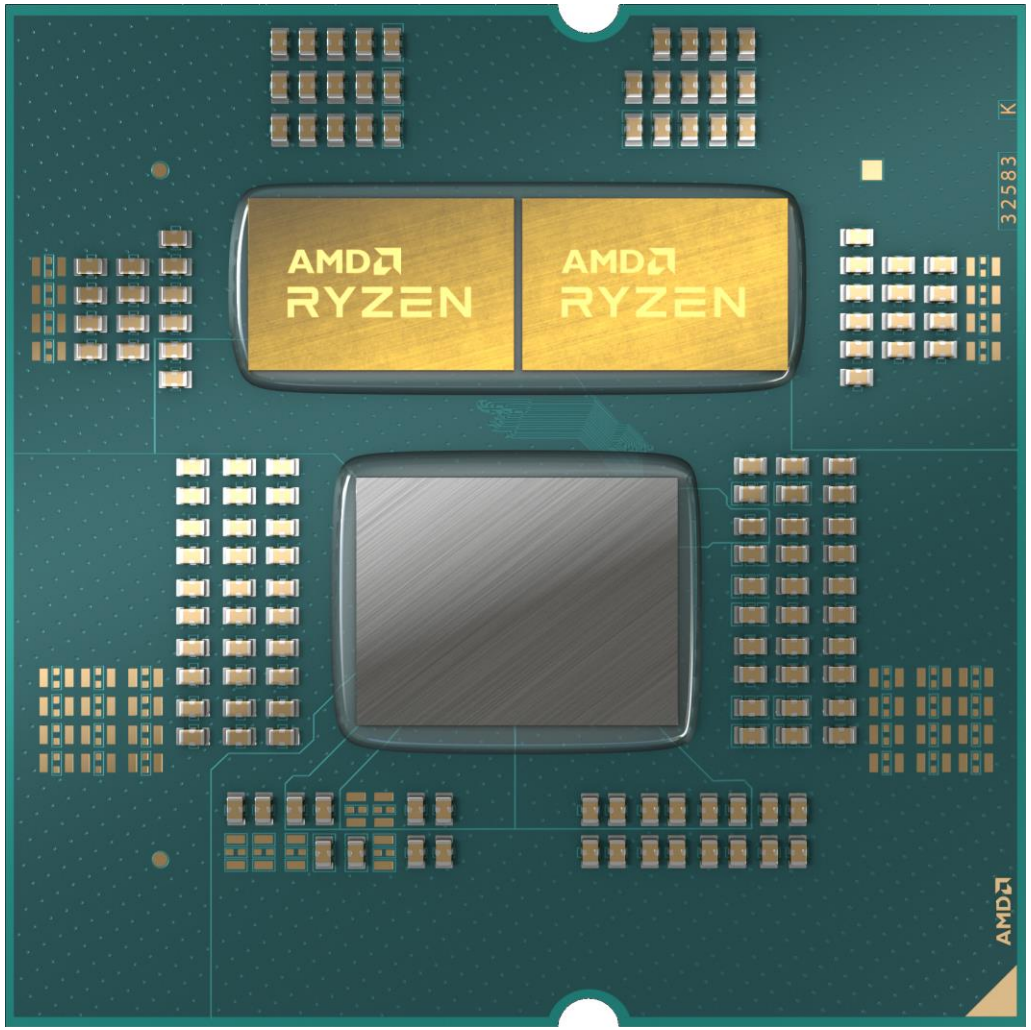
AMD GPUOpen

AMD together we advance_

AGENDA

- Abstract
- Speaker Biography
- Products
- Data Flow
- Microarchitecture
- 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

- **John Hartwig** is a Senior Member of Technical Staff and is the CPU Team Lead in the AMD Game Engineering organization. John works with game developers to optimize for AMD processors analyzing game code to provide fixes and mitigation for areas such as memory use and engine threading. John studied Game Development Programming at DePaul University in Chicago.

- John.Hartwig@amd.com

SPEAKER BIOGRAPHY

- **Ken Mitchell** is a Principal Member of Technical Staff 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

AMD RYZEN™ 7000 SERIES MOBILE PROCESSORS

Model	Graphics Model	Cores	Threads	Max Boost Clock	Base Clock	Graphics Cores	Default TDP	Code Name
AMD Ryzen™ 9 7945HX	AMD Radeon™ 610M	16	32	Up to 5.4 GHz	2.5 GHz	2	55 W	“Dragon Range”
AMD Ryzen™ 9 7845HX	AMD Radeon™ 610M	12	24	Up to 5.2 GHz	3.0 GHz	2	55 W	“Dragon Range”
AMD Ryzen™ 7 7745HX	AMD Radeon™ 610M	8	16	Up to 5.1 GHz	3.6 GHz	2	55 W	“Dragon Range”
AMD Ryzen™ 5 7645HX	AMD Radeon™ 610M	6	12	Up to 5.0 GHz	4.0 GHz	2	55 W	“Dragon Range”
AMD Ryzen™ 9 7940HS	AMD Radeon™ 780M	8	16	Up to 5.2 GHz	4.0 GHz	12	35-54 W	“Phoenix”
AMD Ryzen™ 7 7840HS	AMD Radeon™ 780M	8	16	Up to 5.1 GHz	3.8 GHz	12	35-54 W	“Phoenix”
AMD Ryzen™ 5 7640HS	AMD Radeon™ 760M	6	12	Up to 5.0 GHz	4.3 GHz	8	35-54 W	“Phoenix”
AMD Ryzen™ 7 7735HS	AMD Radeon™ 680M	8	16	Up to 4.75 GHz	3.2 GHz	12	35-54 W	“Rembrandt R”
AMD Ryzen™ 5 7535HS	AMD Radeon™ 660M	6	12	Up to 4.55 GHz	3.3 GHz	6	35-54 W	“Rembrandt R”

AMD RYZEN™ 7000 SERIES MOBILE PROCESSORS

Model	Graphics Model	Cores	Threads	Max Boost Clock	Base Clock	Graphics Cores	Default TDP	Code Name
AMD Ryzen™ 7 7736U	AMD Radeon™ 680M	8	16	Up to 4.7GHz	2.7GHz	12	15-28W	“Rembrandt R”
AMD Ryzen™ 7 7735U	AMD Radeon™ 680M	8	16	Up to 4.75GHz	2.7GHz	12	28W	“Rembrandt R”
AMD Ryzen™ 5 7535U	AMD Radeon™ 660M	6	12	Up to 4.55GHz	2.9GHz	6	28W	“Rembrandt R”
AMD Ryzen™ 3 7335U	AMD Radeon™ 660M	4	8	Up to 4.3GHz	3.0GHz	4	28W	“Rembrandt R”
AMD Ryzen™ 7 7730U	AMD Radeon™ Graphics	8	16	Up to 4.5GHz	2.0GHz	8	15W	“Barcelo R”
AMD Ryzen™ 5 7530U	AMD Radeon™ Graphics	6	12	Up to 4.5GHz	2.0GHz	7	15W	“Barcelo R”
AMD Ryzen™ 3 7330U	AMD Radeon™ Graphics	4	8	Up to 4.3GHz	2.3GHz	6	15W	“Barcelo R”
AMD Ryzen™ 5 7520U	AMD Radeon™ 610M	4	8	Up to 4.3GHz	2.8GHz	2	15W	“Mendocino”
AMD Ryzen™ 3 7320U	AMD Radeon™ 610M	4	8	Up to 4.1GHz	2.4GHz	2	15W	“Mendocino”

AMD RYZEN™ 7000 SERIES DESKTOP PROCESSORS

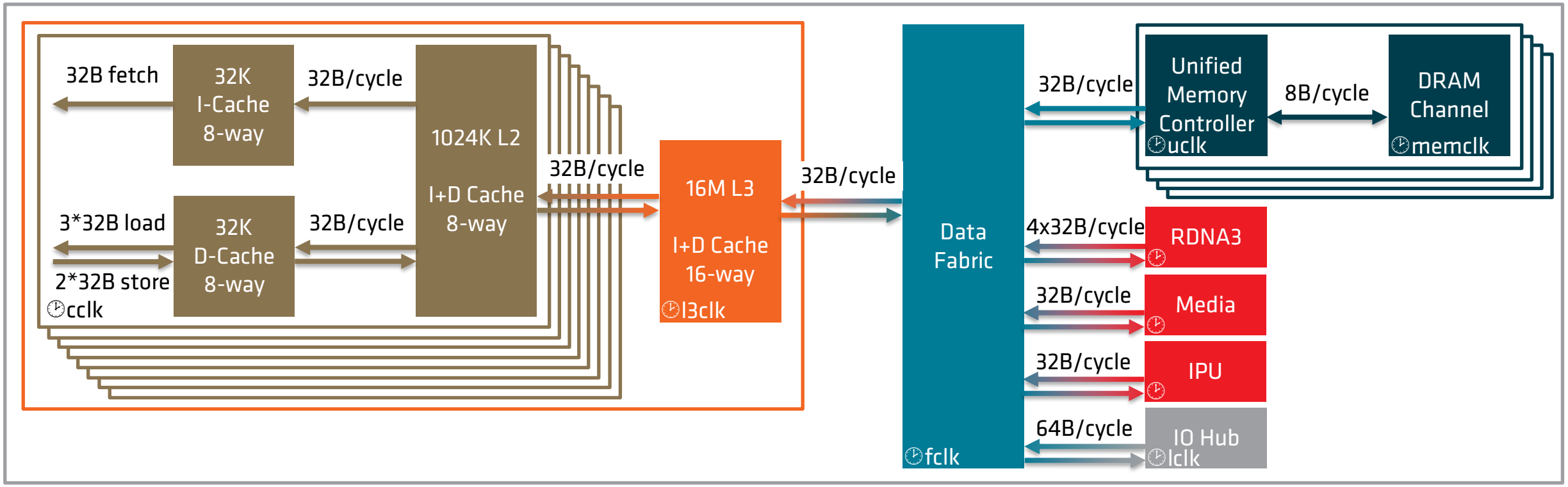
Model	Graphics Model	Cores	Threads	Total L3 Cache	Max Boost Clock	Base Clock	Graphics Cores	Default TDP	Code Name
AMD Ryzen™ 9 7950X3D	AMD Radeon™ Graphics	16	32	128	Up to 5.7 GHz	4.2 GHz	2	120 W	“Raphael”
AMD Ryzen™ 9 7950X	AMD Radeon™ Graphics	16	32	64	Up to 5.7 GHz	4.5 GHz	2	170 W	“Raphael”
AMD Ryzen™ 9 7900X3D	AMD Radeon™ Graphics	12	24	128	Up to 5.6 GHz	4.4 GHz	2	120 W	“Raphael”
AMD Ryzen™ 9 7900X	AMD Radeon™ Graphics	12	24	64	Up to 5.6 GHz	4.7 GHz	2	170 W	“Raphael”
AMD Ryzen™ 7 7800X3D	AMD Radeon™ Graphics	8	16	96	Up to 5.0 GHz		2	120 W	“Raphael”
AMD Ryzen™ 9 7900	AMD Radeon™ Graphics	12	24	64	Up to 5.4 GHz	3.7 GHz	2	65 W	“Raphael”
AMD Ryzen™ 7 7700X	AMD Radeon™ Graphics	8	16	32	Up to 5.4 GHz	4.5 GHz	2	105 W	“Raphael”
AMD Ryzen™ 7 7700	AMD Radeon™ Graphics	8	16	32	Up to 5.3 GHz	3.8 GHz	2	65 W	“Raphael”
AMD Ryzen™ 5 7600X	AMD Radeon™ Graphics	6	12	32	Up to 5.3 GHz	4.7 GHz	2	105 W	“Raphael”
AMD Ryzen™ 5 7600	AMD Radeon™ Graphics	6	12	32	Up to 5.1 GHz	3.8 GHz	2	65 W	“Raphael”

AMD RYZEN™ THREADRIPPER™ PRO 5000WX SERIES PROCESSORS

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

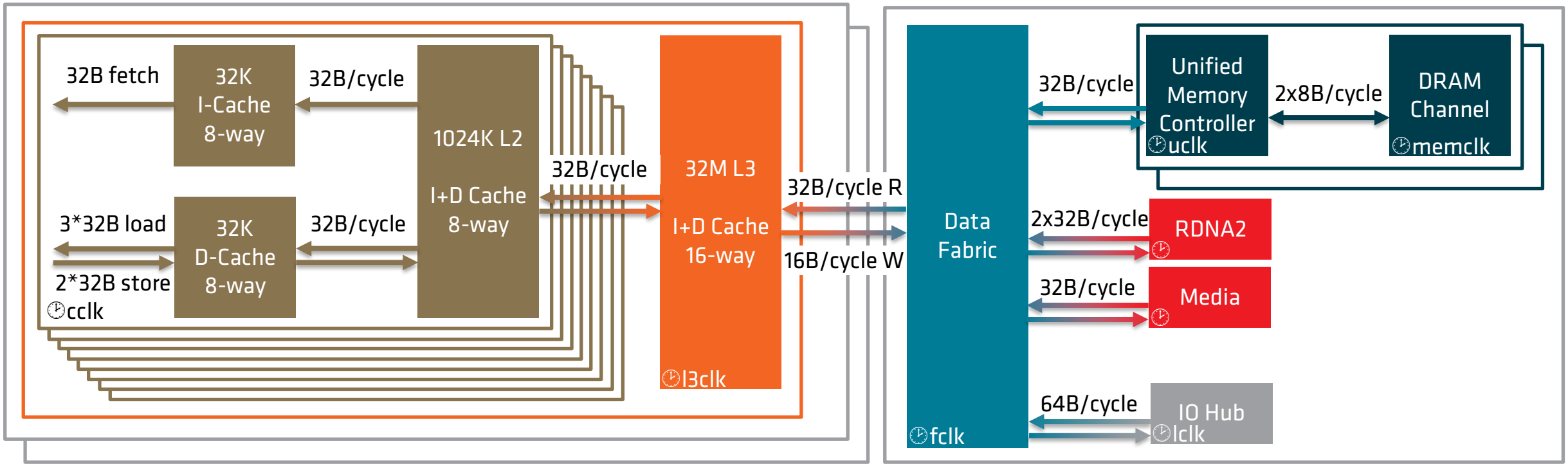
DATA FLOW

AMD RYZEN™ 9 7940HS MOBILE PROCESSOR



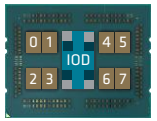
- AMD Ryzen™ 9 7940HS, 35-54W TDP, 8 cores, 16 threads, up to 5.2 GHz max boost clock, 4.0 GHz base clock with 2 channels of DDR5 memory.
- integrated RDNA3 graphics and inference processing unit.

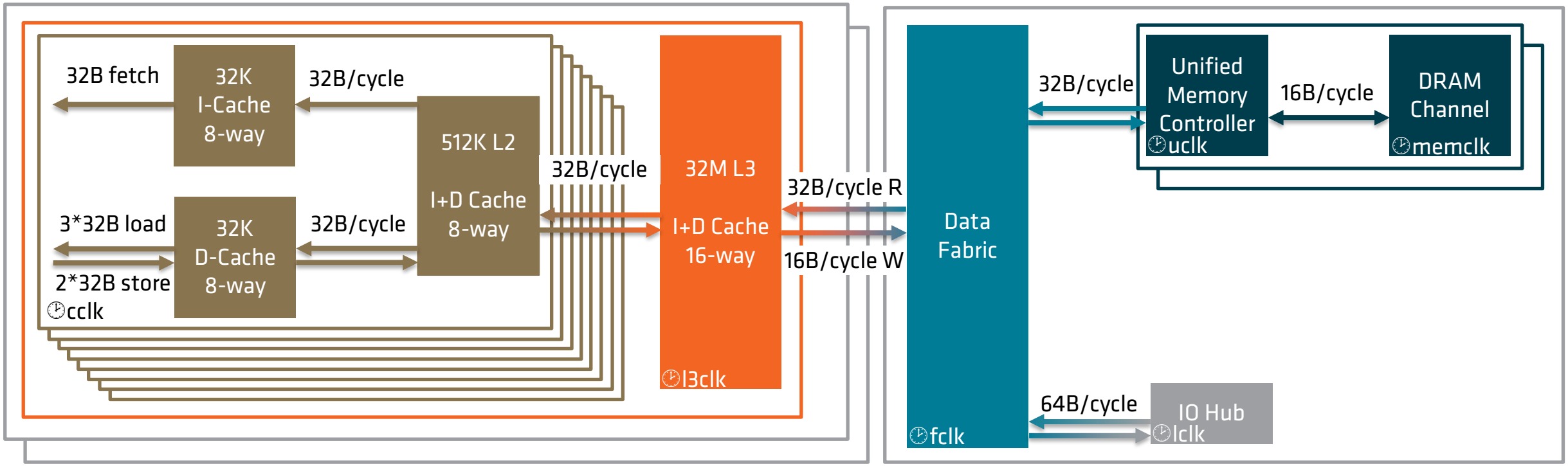
AMD RYZEN™ 9 7950X DESKTOP PROCESSOR



- 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.

AMD RYZEN™ THREADRIPPER™ PRO 5995WX PROCESSOR

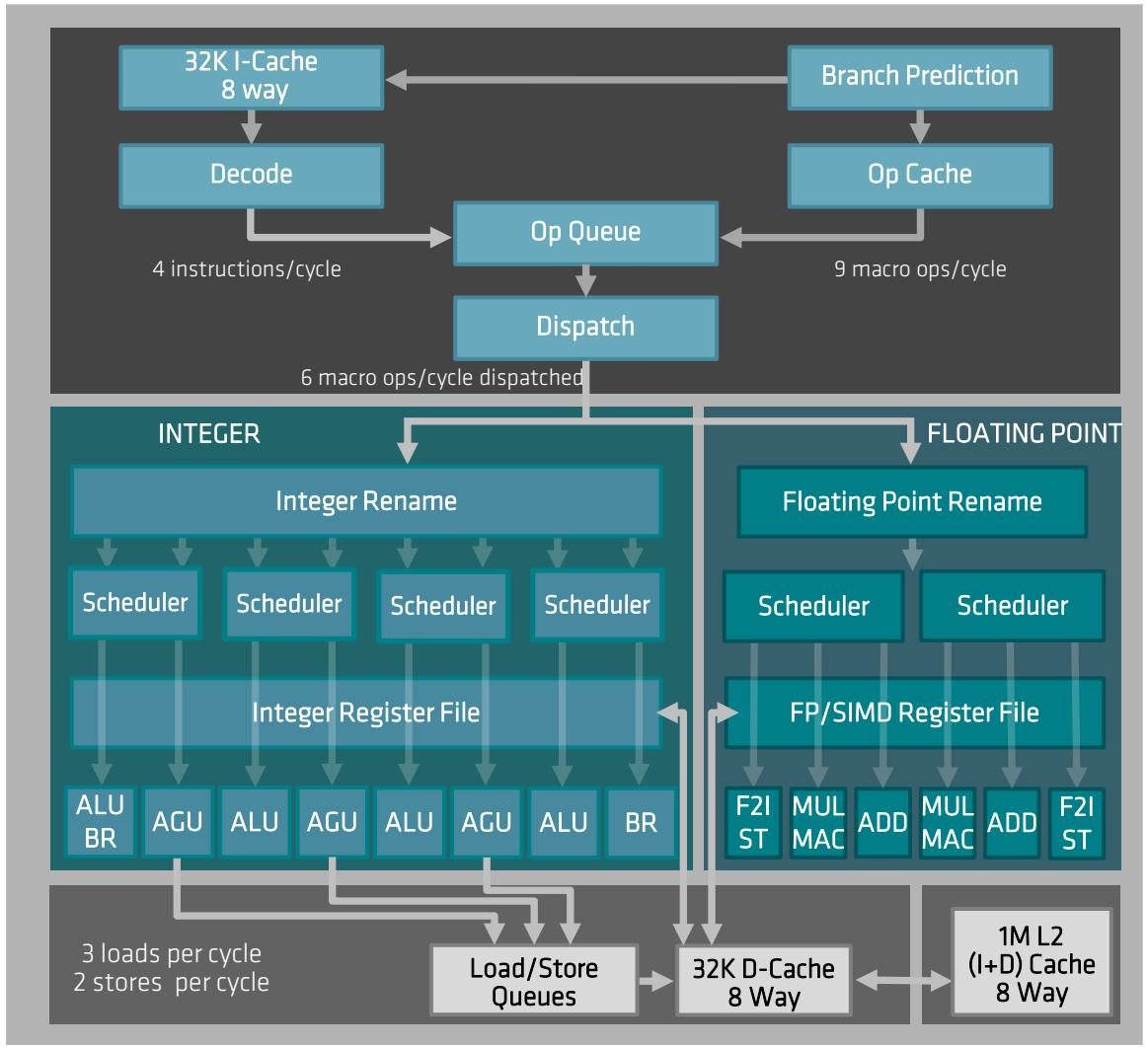




- AMD Ryzen™ Threadripper™ Pro 5995WX, 280W TDP, 64 cores, 128 threads, up to 4.5 GHz boost, 2.7 GHz base with 8 channels of DDR4 memory.
- Two CCDs per Data Fabric quadrant shown.

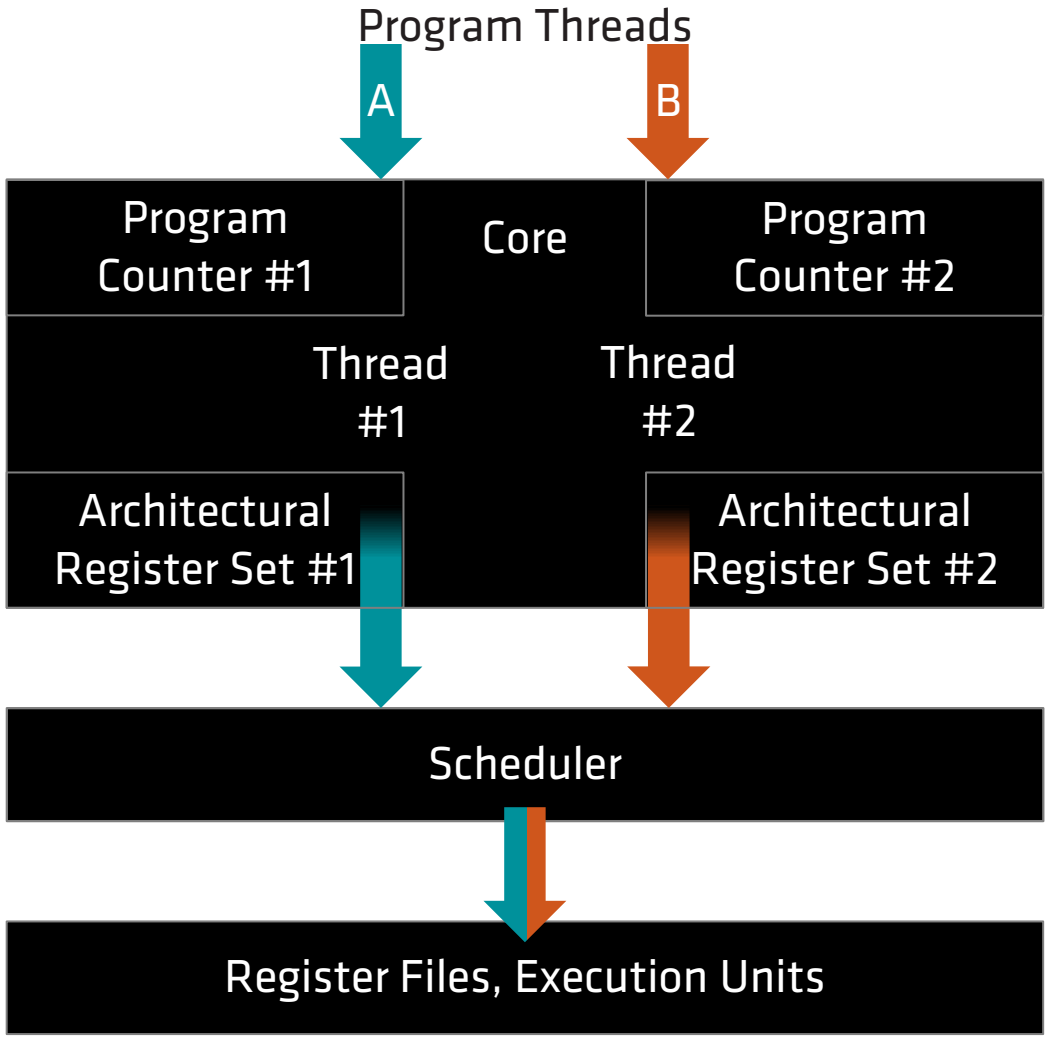
MICROARCHITECTURE

“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 at all times.
- 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.

“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

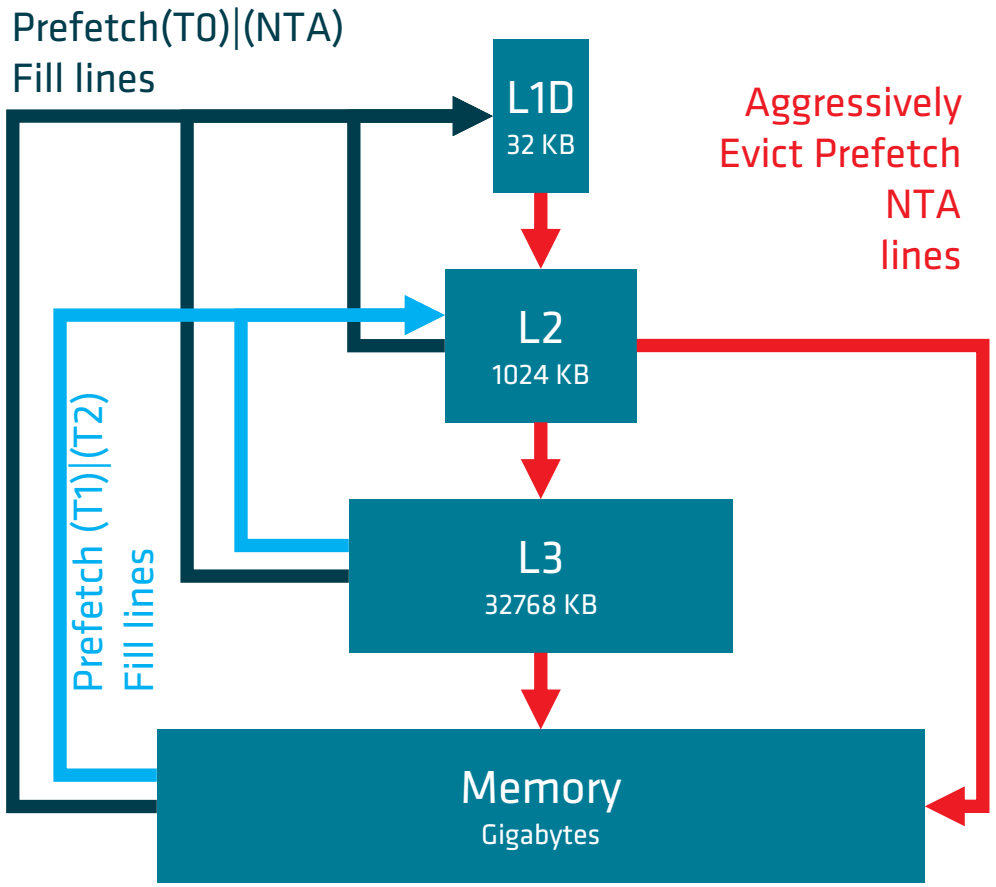
Core	AVX512*	GFNI	VAES	VPCLMUL	CLWB	ADX	CLFLUSHOPT	RDSEED	SHA	XGETBV	XSAVEC	XSAVES	AVX2	BMI2	MOVBE	RDRND	FSGSBASE	XSAVEOPT	BMI	FMA	F16C	AES	AVX	OSXSAVE	PCLMUL	SSE4.1	SSE4.2	XSAVE	SSSE3	MONITORX	CLZERO
“Zen 4”	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
“Zen 3”	0	0	1	1	1	1	1	1	1	1	1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

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	

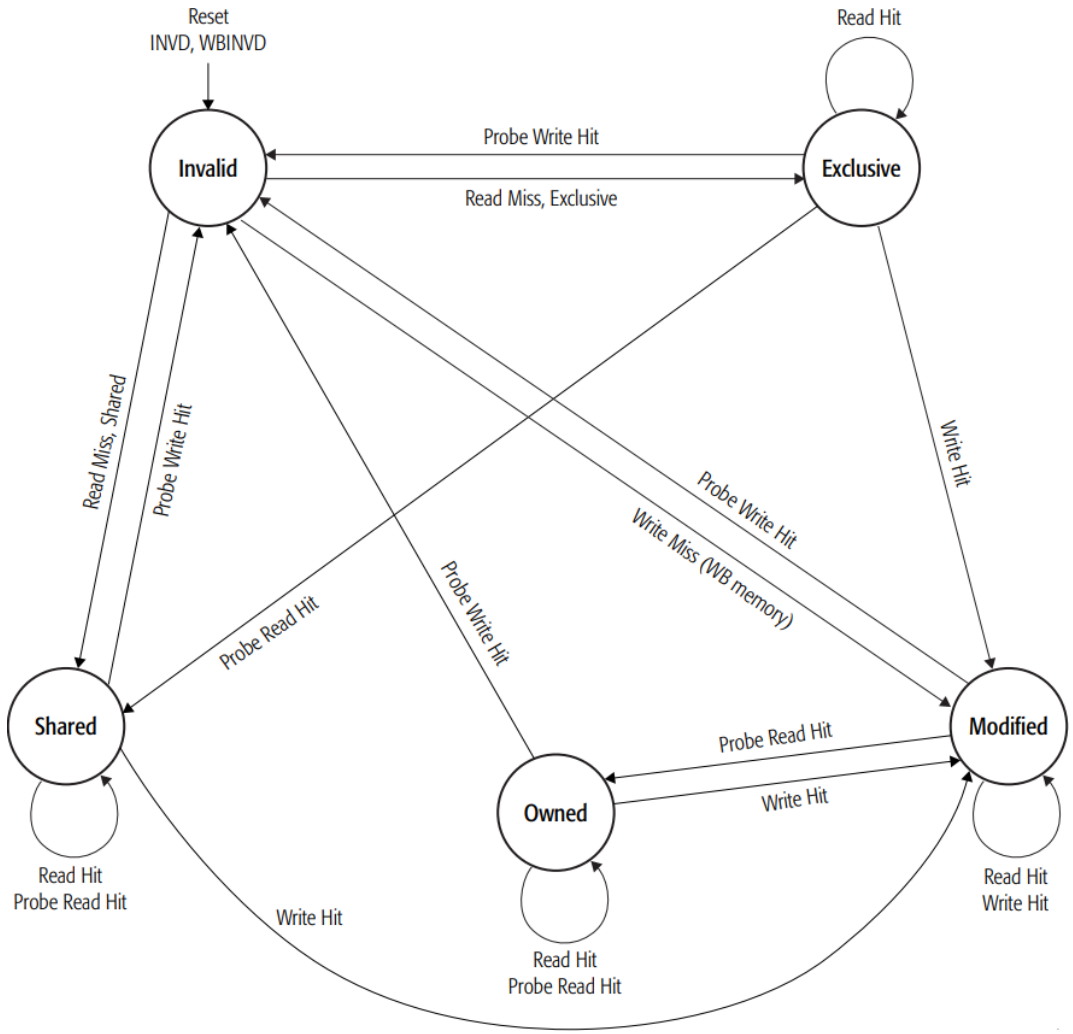
Uses memory access history of individual instructions to fetch additional lines when each access is a constant distance from the previous.

```
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
}
```

DESKTOP CACHE HIERARCHY EVOLUTION

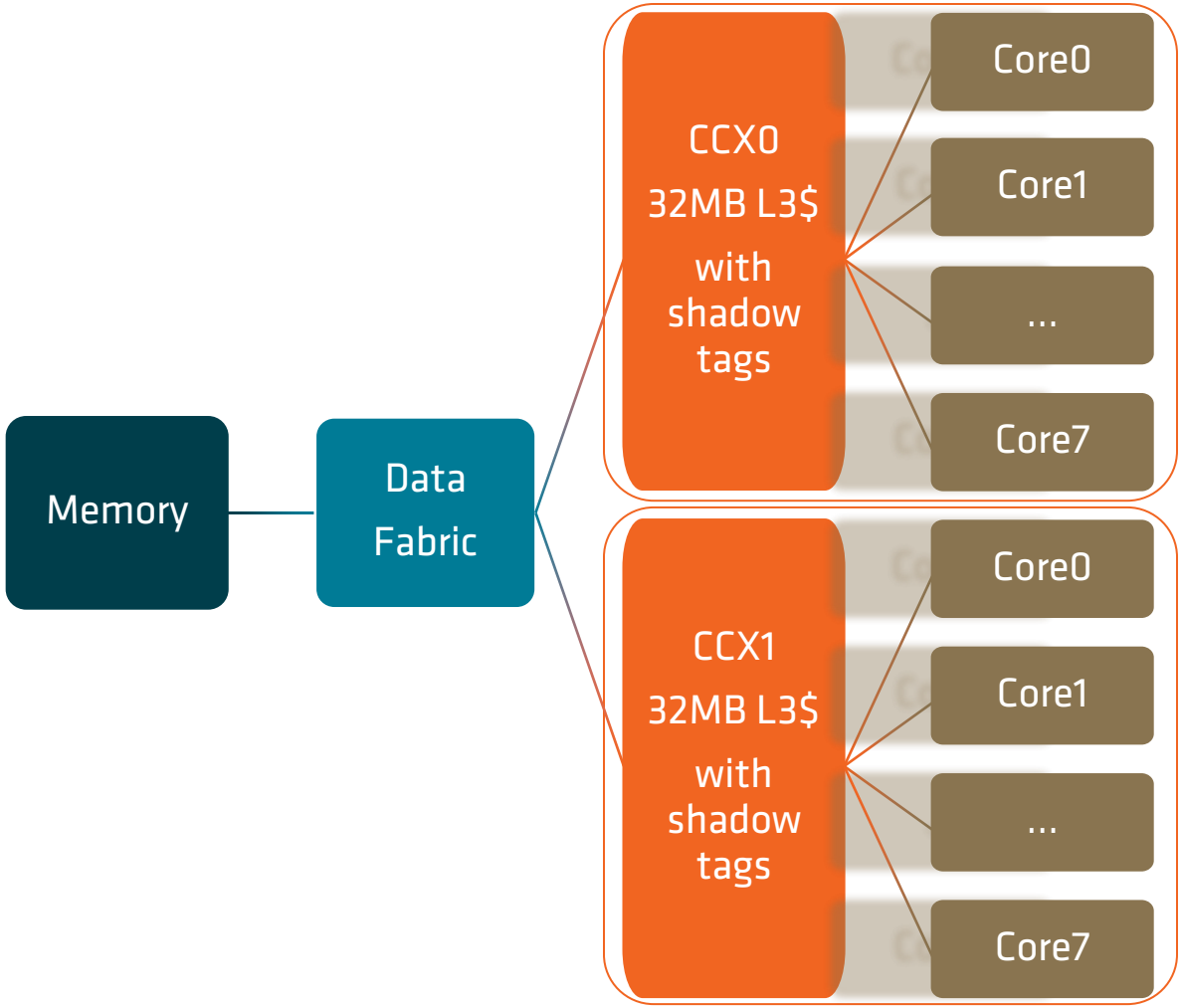
Core	uOP/Core K	L1I/Core KB	L1D/Core KB	L2/Core KB	L3/CCX MB
“Zen 4”	6.75	32	32	1024	32*
“Zen 3”	4	32	32	512	32*
“Zen 2”	4	32	32	512	16
“Zen 1”	2	64	32	512	8

CACHE-COHERENCY PROTOCOL



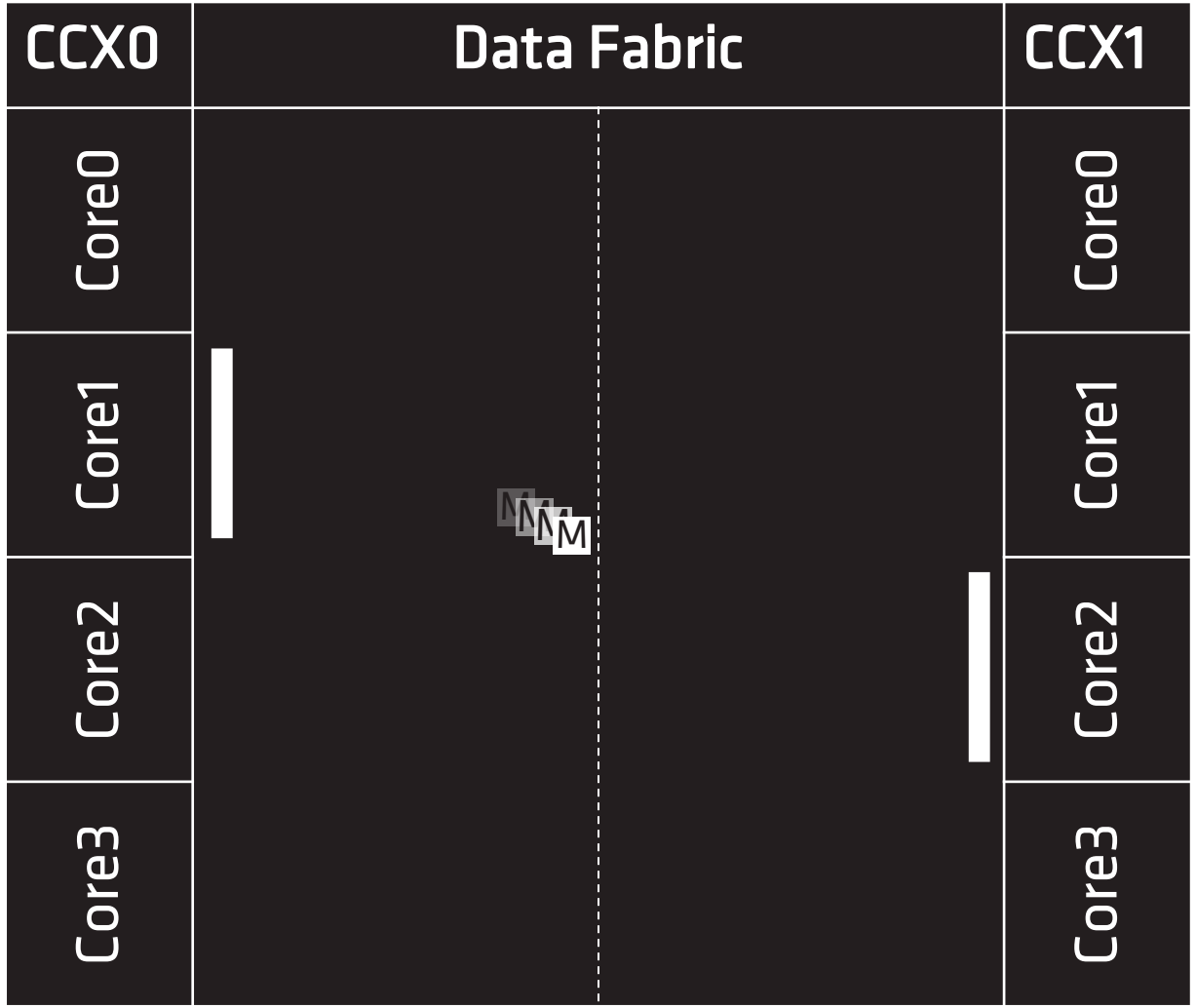
- The AMD cache-coherency protocol is MOESI (Modified, Owned, Exclusive, Shared, Invalid).
- Instruction-execution activity and external-bus transactions may change the cache's MOESI state.
- Read hits do not cause a MOESI-state change.
- **Write hits generally cause a MOESI-state change into the modified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

CACHE-TO-CACHE TRANSFERS



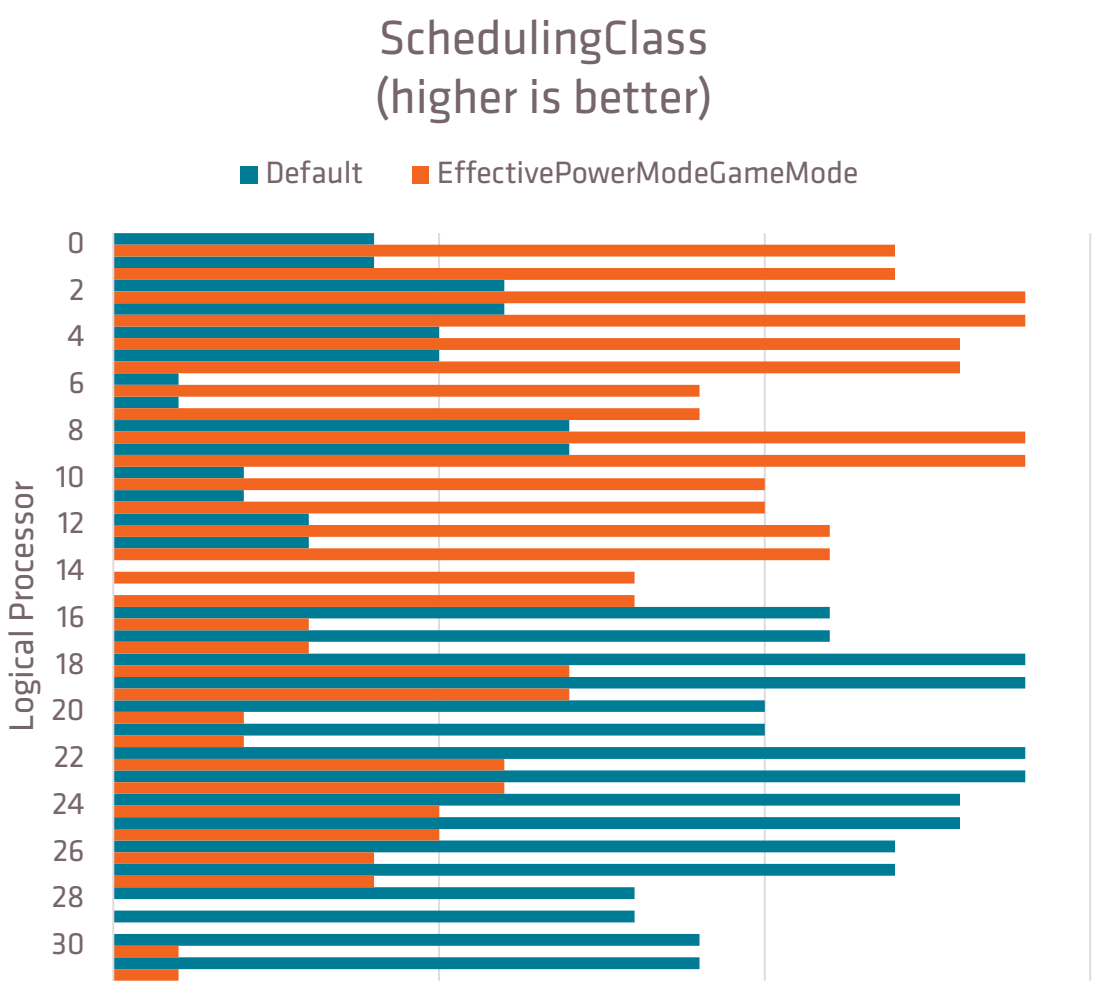
- The CCX has a L3 cache shared by up to eight cores.
- The L3 has shadow tags for each L2 in the complex.
- Shadow tags determine if a cache-to-cache transfer between cores is possible inside the CCX.
- <u>Cache-coherency probe latency responses may be slower from cores in another CCX.</u>

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.
  - Replace user spin locks with modern sync APIs.
- Use a memory allocator optimized for multi-threading.
  - Try mimalloc or jemalloc.

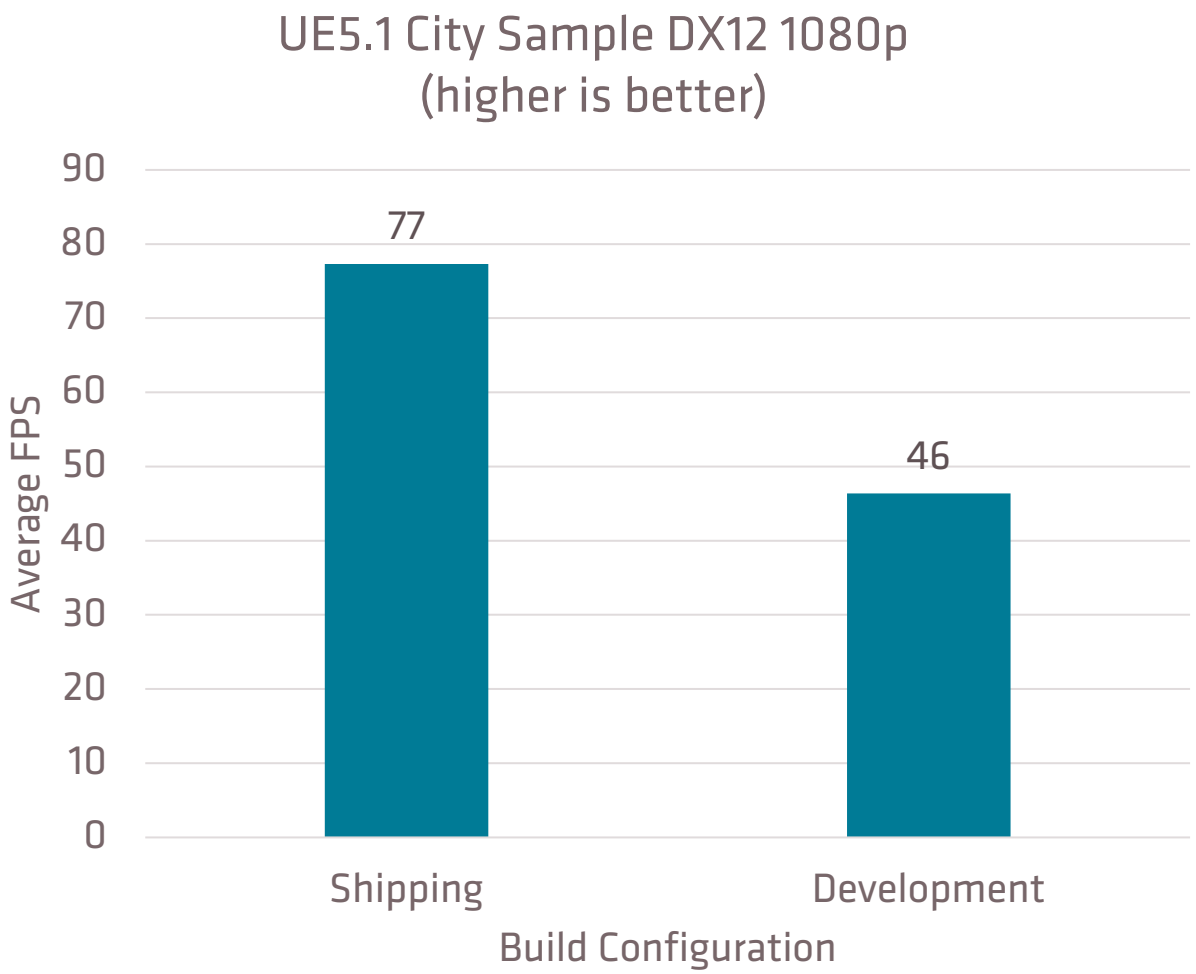
AMD "PREFERRED CORE"



- 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



- 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 UE4.5.1 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.*

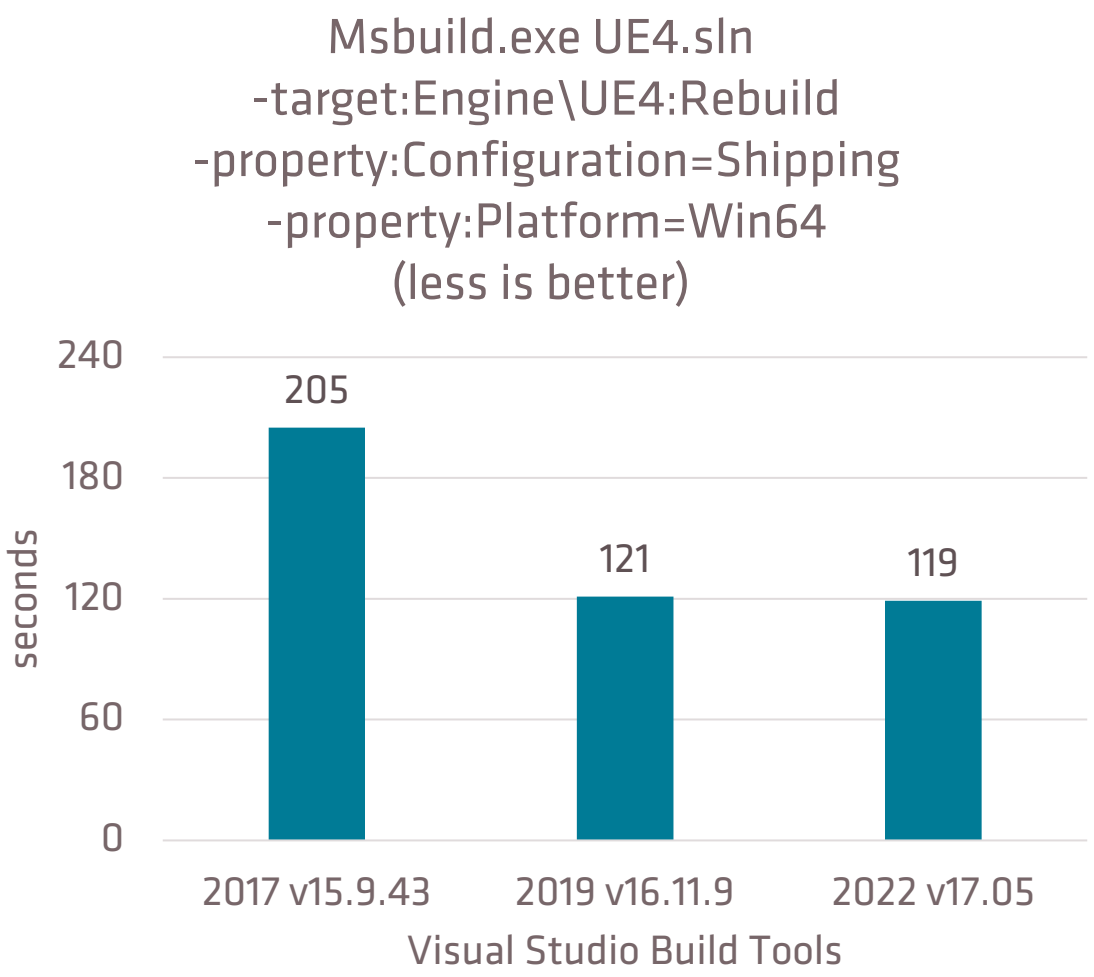
DISABLE ANTI-TAMPER WHILE CPU PROFILING

- When possible, build a binary similar-to shipping configuration but without anti-tamper or anti-cheat which may prevent CPU profiling tools from properly loading symbols.
  - A happy medium may be to leave anti-tamper on by default for test builds but to provide a launch option for easy profiling and debugging.

TEST COLD SHADER CACHE FIRST TIME USER EXPERIENCE

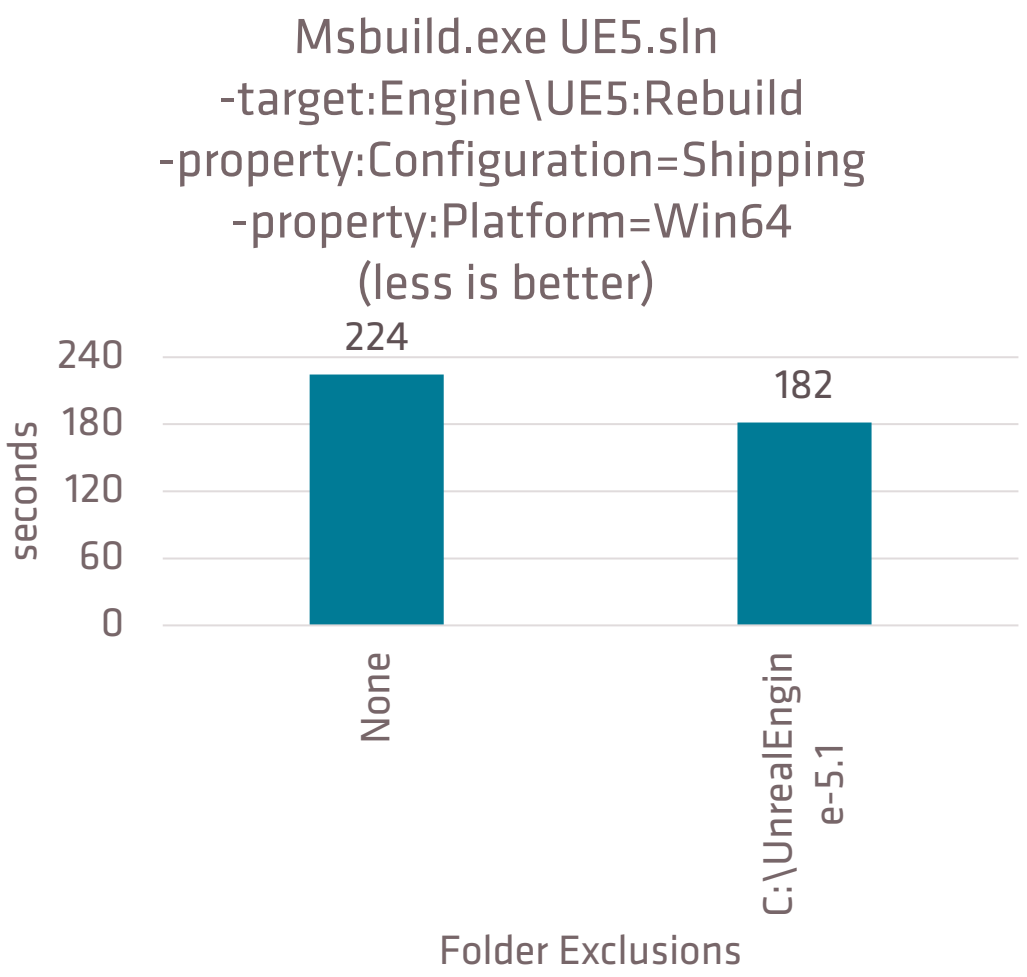
```
rem Run as administrator
rem Disable Steam shader pre-caching before running this script
rem Reboot after running this script to clear any shaders still in system memory

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"
```

USE THE LATEST COMPILER AND WINDOWS® SDK



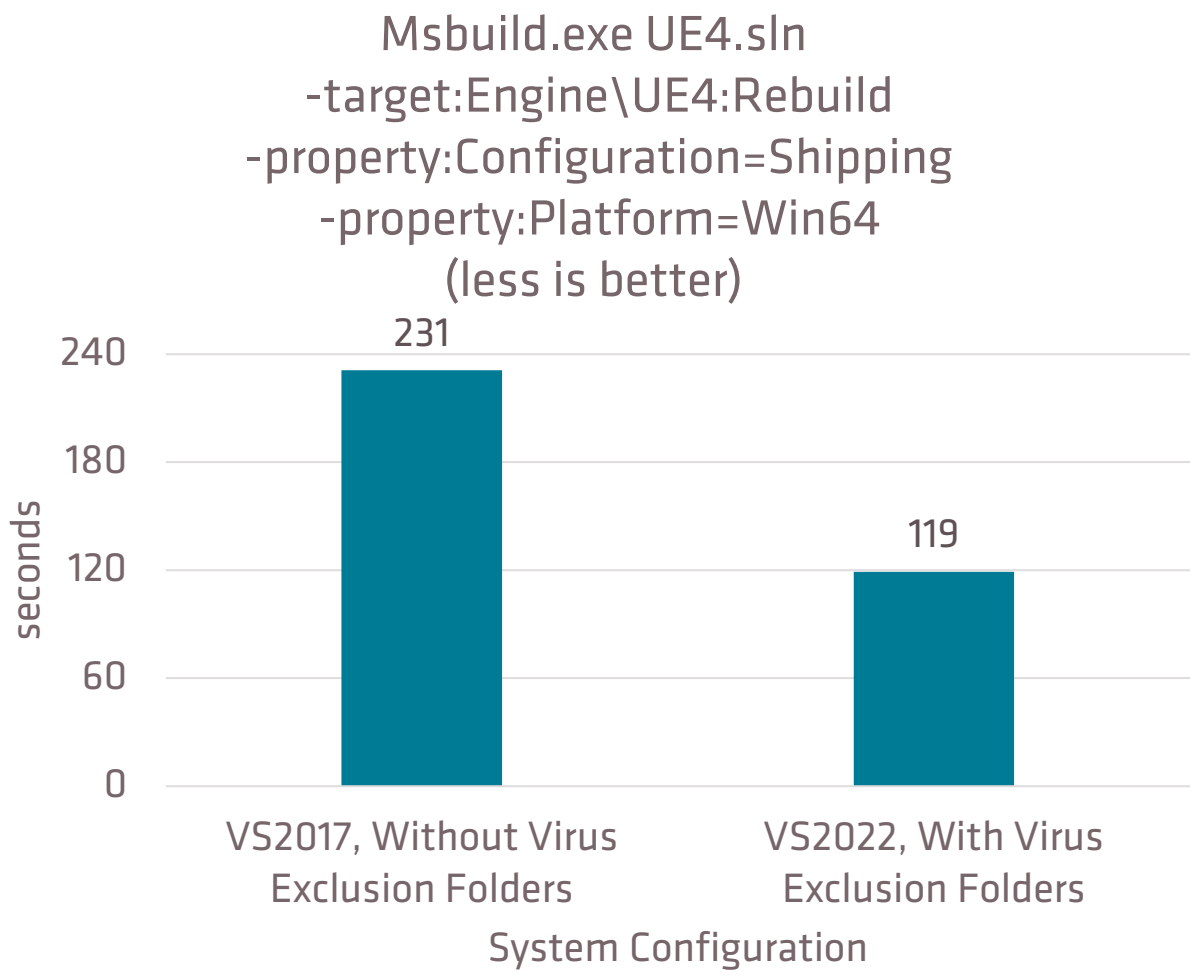
- Get the latest build and link time improvements.
- Get the latest library and runtime optimizations.
- *Performance of UE4.27.2 binaries compiled with Microsoft Visual Studio.*
- *Testing done by AMD technology labs, February 5, 2022 on the following system. Test configuration: AMD Ryzen™ Threadripper™ 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



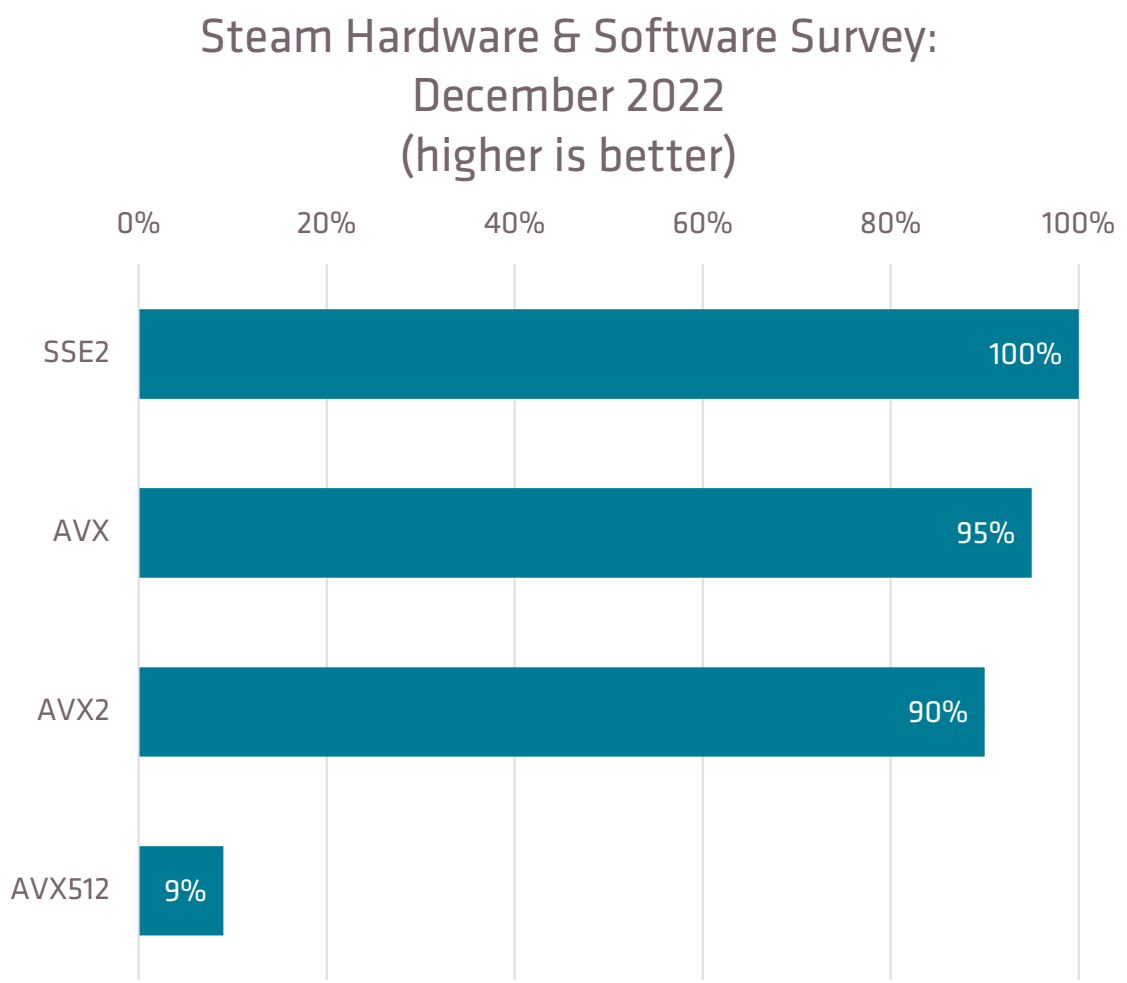
- WARNING: Not recommended for CI/CD systems. Exclusions may make your device vulnerable to threats.
- Add project folders to virus and threat protection settings exclusions for faster build times.
- Faster rebuild time after optimization!
- *Performance of UE5.1 binaries compiled with Microsoft Visual 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



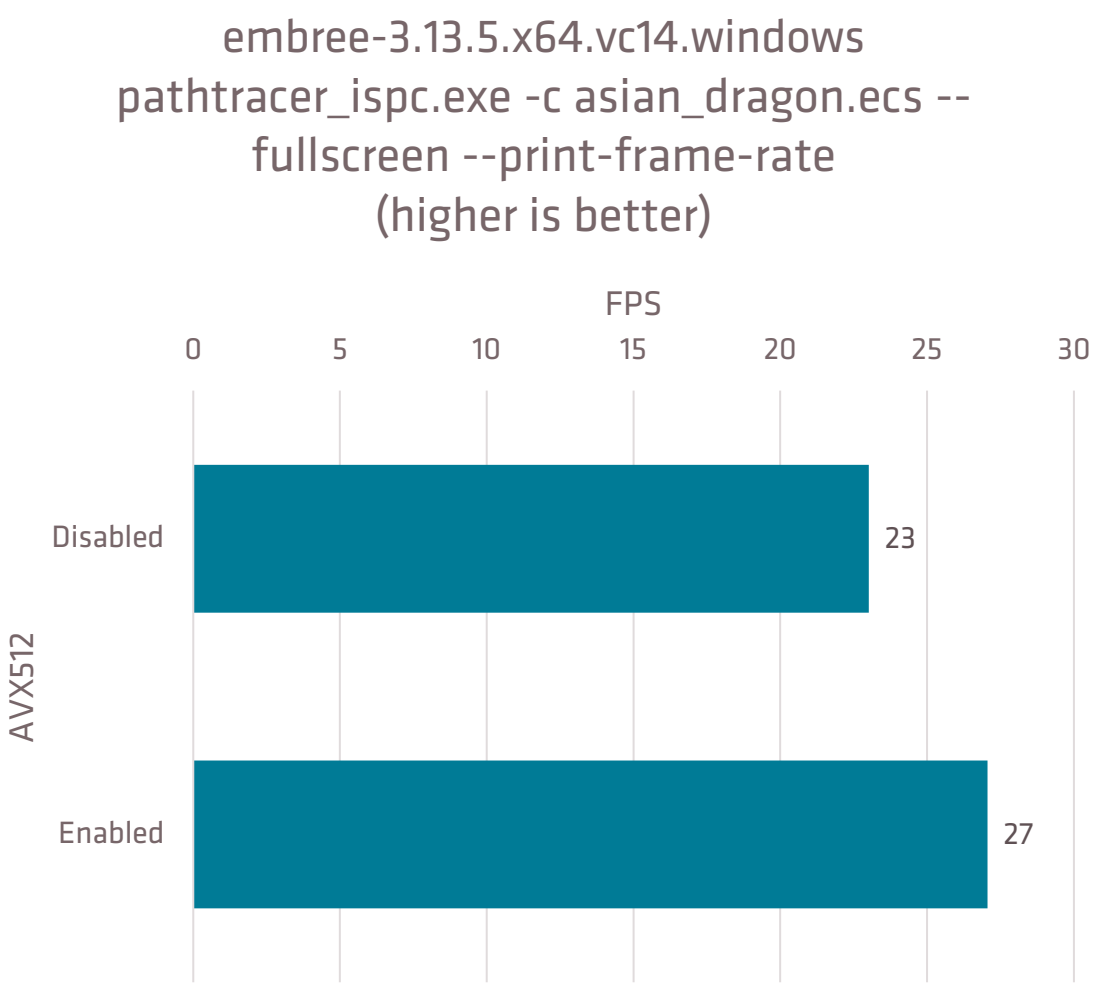
- *Performance of UE4.27.2 binaries compiled with Microsoft Visual Studio.*
- *Testing done by AMD technology labs, February 5, 2022 on the following system. Test configuration: AMD Ryzen™ Threadripper™ PRO 5995WX, Enermax LIQTECH TR4 II series 360mm liquid cooler, 256GB (8 x 32GB 2R RDDR4-3200 at 24-22-22-52)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



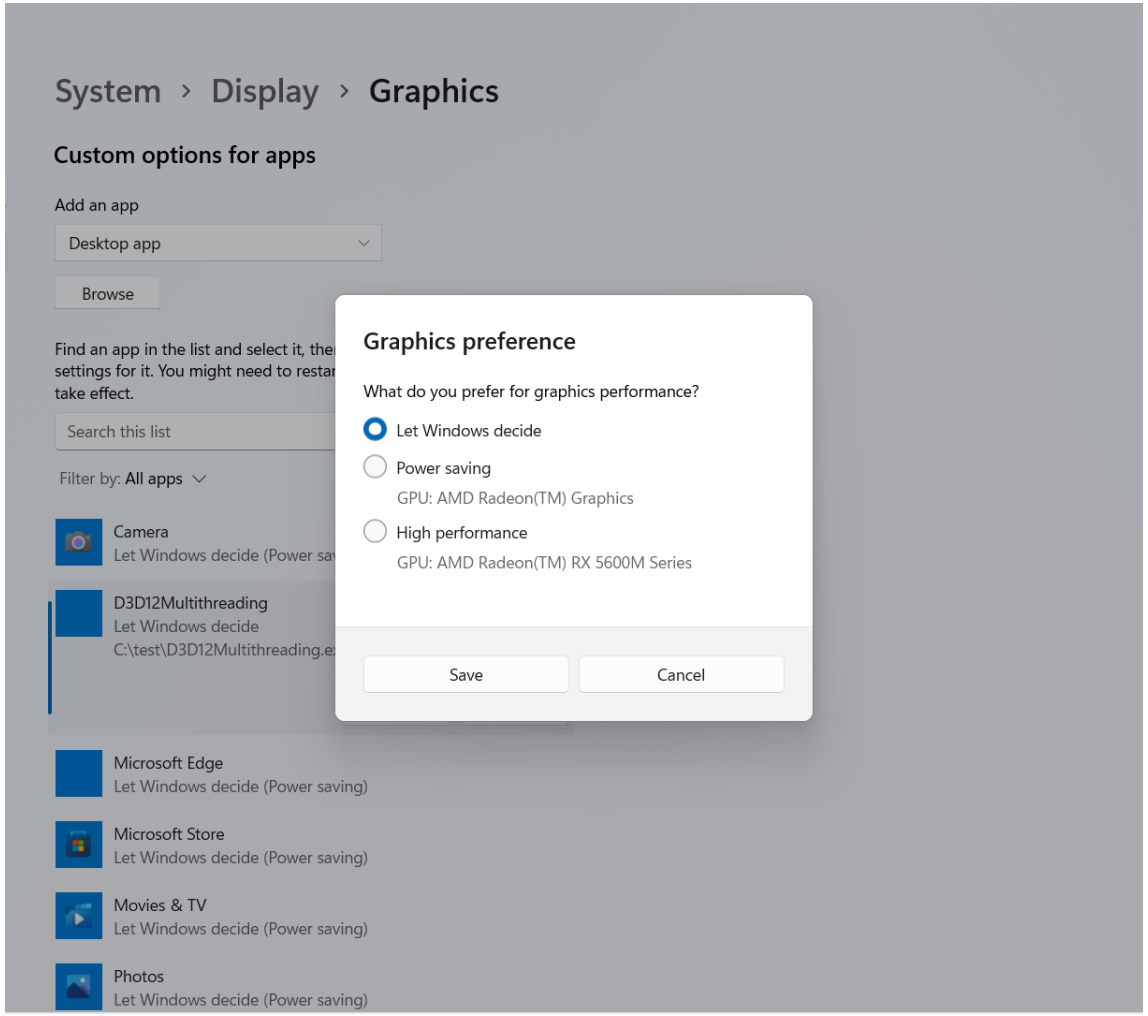
- Development tools may benefit from AVX512.
- Examples:
  - Light Baking.
  - Texture Compression.
  - Mesh to Signed Distance Fields.

- *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 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.

SUPPORT HYBRID GRAPHICS



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

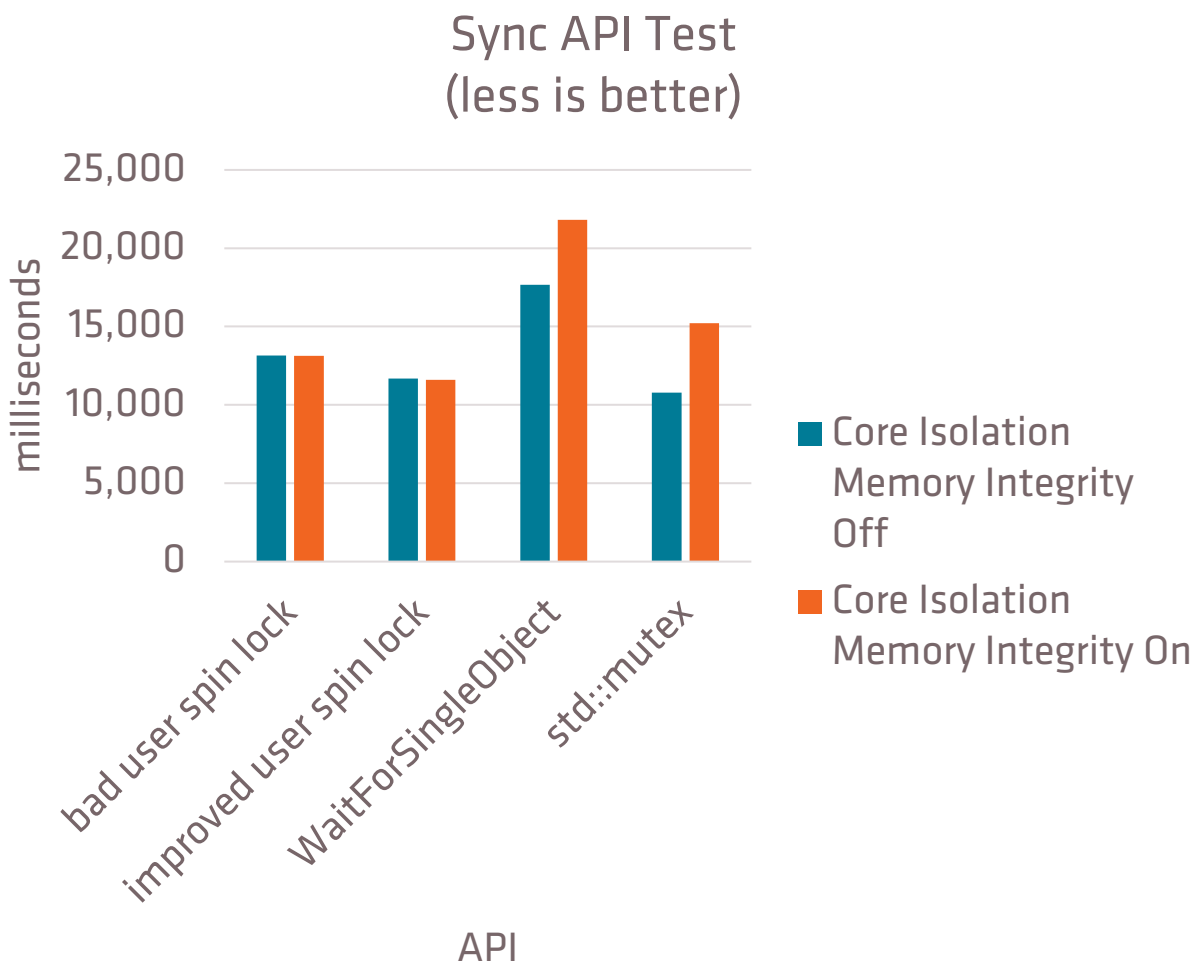
- *Testing done by AMD performance labs January 24, 2022 on a Dell G5 15 SE laptop equipped with, 16GB DDR4-3200MHz, Ryzen™ 9 4900H with Radeon™ RX 5600M, Win11 Pro x64 22000.434.*

USE PREFERRED VIDEO AND AUDIO CODECS

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

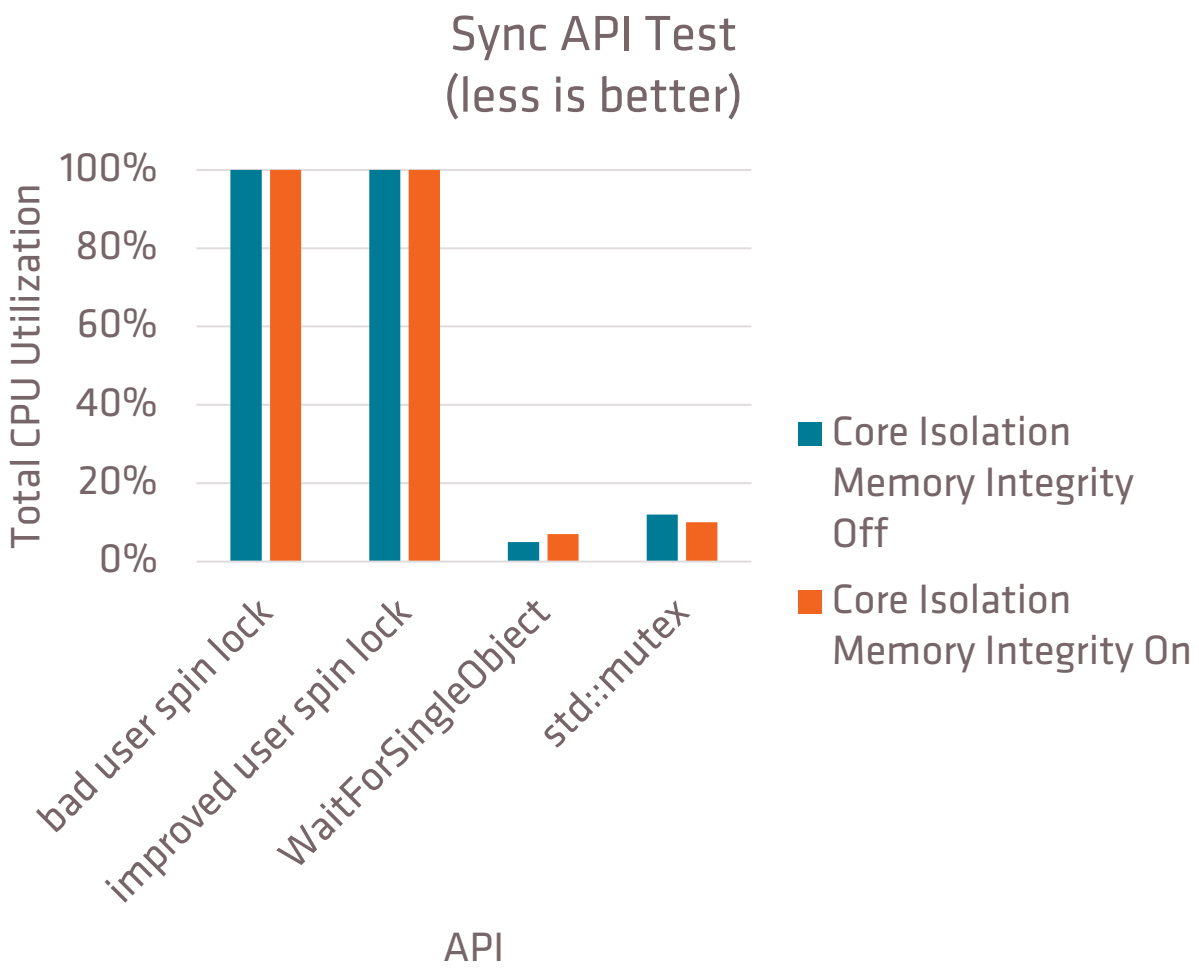
OPTIMIZATIONS

USE MODERN SYNC APIS



- Prefer std::mutex which has good performance and low cpu utilization.
- Legacy APIs like WaitForSingle rely on heavy kernel mode calls and for compatibility reasons will not compile into vendor optimized instructions like the AMD low overhead mwaitx instruction.
- *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 version 21H2, 1920x1080 resolution. Actual results may vary.*

USE MODERN SYNC APIS



- Prefer std::mutex which has good performance and low cpu utilization.
- Spin locks burn cycles and drain laptop batteries.
- *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 version 21H2, 1920x1080 resolution. Actual results may vary.*

USE MODERN SYNC APIS – THEY'RE MODERN FOR A REASON

- **The Good:**
  - AcquireSRWLockExclusive
  - AcquireSRWLockShared
  - SleepConditionVariableSRW
  - SleepConditionVariableCS
  - EnterCriticalSection
    - calls EnterCriticalSectionContended
  - std::mutex
    - calls AcquireSRWLockExclusive;
  - std::shared_mutex
    - calls AcquireSRWLockShared;
- These compile into mwaitx and avoid costly syscall instructions ☺

- **The Bad:**
- Avoid costly syscall instructions in:
  - NtWaitForSingleObject
  - NtWaitForMultipleObjects
  - WakeAllConditionVariable
    - calls NtAlertThreadByThreadId
  - NtReleaseSemaphore
    - calls NtAlertThreadByThreadId
- Some of these have been around a long time.

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: 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;
```

- Make the most of your pause duration.
  - Try aligning your pause count to the latency of the pause instruction on your target hardware.
- If at first you don't succeed, call it a day and put the loop to sleep.
  - Try a set number of spin/pause cycles or even an exponential backoff algorithm.
  - If you still don't have your resource, and you're still waiting, then this might not be the quick low overhead lock you thought it was. Put the thread to sleep and have it signal wake.

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;
```

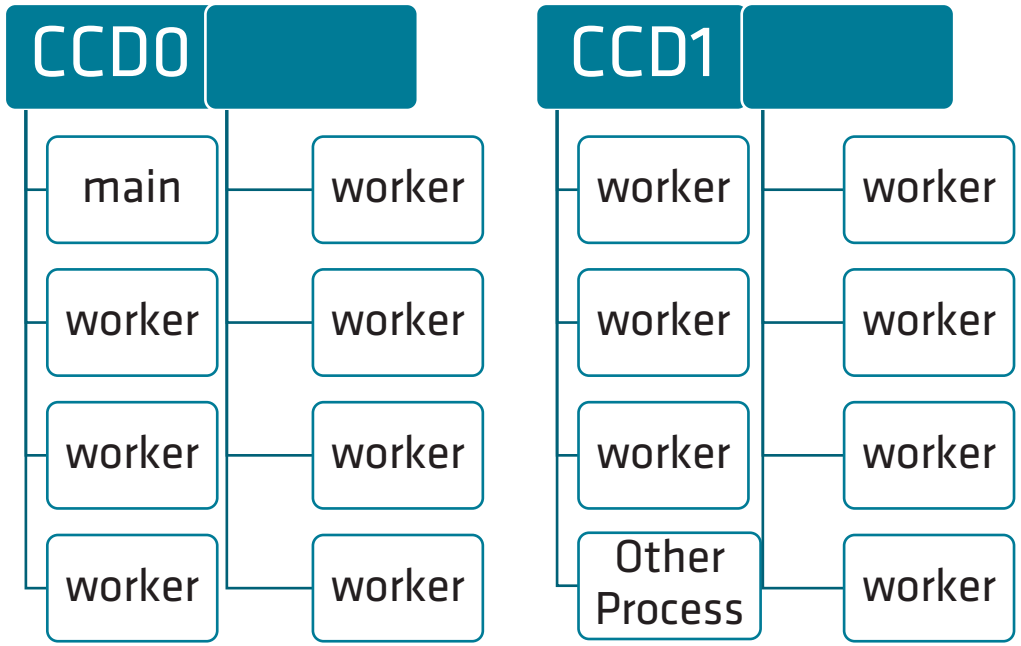
```
void fn() {
    alignas(64) float a[LEN][4][4];
    std::fill((float*)a, (float*)(a + LEN), 0.0f);
    float r = 0.0;
    for (size_t iter = 0; iter < 100000; iter++) {
       mutex.lock();
        for (int m = 0; m < LEN; m++)
            for (int i = 0; i < 4; i++)
                for (int j =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

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);

THREADING

WRITE CODE THAT SCALES WITH CORES



physical cores-1 is often a good place to start

- Under threading is bad.
  - Unlike consoles, PCs don't have a single config.
  - Avoid hard coding thread pool size on PC.
- Over-threading is bad.
  - May cause thread migration and lock contention.
- In a perfect world you can scale to all logical processors.
  - PSO compilation scales nicely.
- Games may perform better using physical cores.
  - May reduce SMT and cache contention.
  - In the real work you often need to scale real-time code to physical cores.
- See https://gpuopen.com/learn/cpu-core-counts/

WATCH OUT FOR AFFINITY MASKS

Main affinity=none

CPU0	main	other process	main	idle
CPU1	idle	main	idle	idle

Main affinity=1

CPU0	main	other process	main	main
CPU1	idle	idle	idle	idle

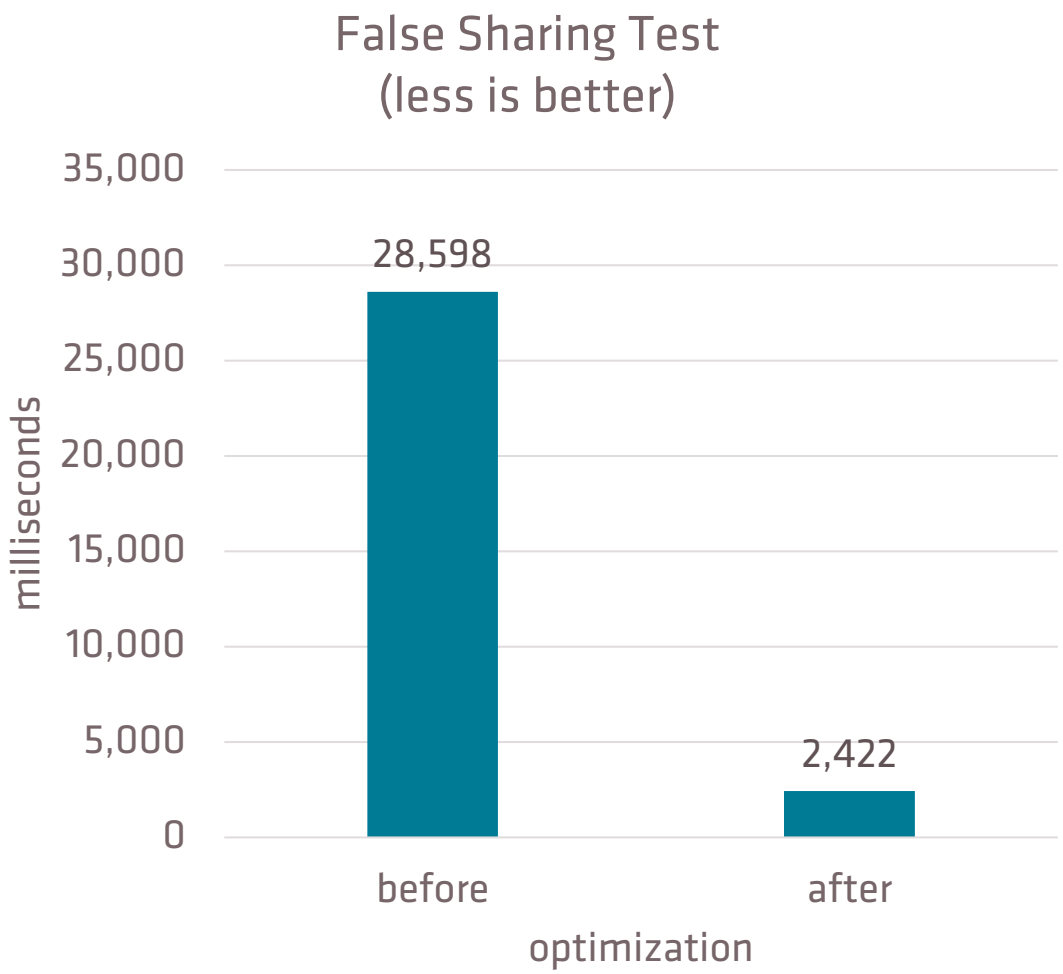
- These generally interfere with OS scheduling and power management.
- If you lock a single high priority thread to a core and that thread stalls, your core stalls.
- It's better to let OS float low priority work across cores when idles are found.
  - This way critical threads can pre-empt as needed.
- Prefer priority over affinity masks when possible.

PRIORITY IS BETTER BUT NOT A PERFECT SCIENCE

- Priority allows work to float across cores.
- Don't starve the OS.
  - Greedy work can starve critical OS functions.
  - Most game systems aren't important enough to run the highest available priority levels.
- Watch out for Priority Boosts!
  - Sometimes this can get called unexpectedly when waiting on critical sections.
  - Boost will increase a threads priority by 1 temporarily.
  - Stagger your priority ranges at least 2 apart.
  - Small ranges mean a boosted thread can fully switch priority class from low to med or high to crit.
    - This is typically NOT expected behavior.
  - Routinely boosted threads can unintentionally become effectively permanently boosted.
- Try disabling priority boost if you suspect this is the source of your apps issue.
  - If performance improved, you may have some inappropriate boosts or an improper range to address.

DATA ACCESS

AVOID FALSE SHARING



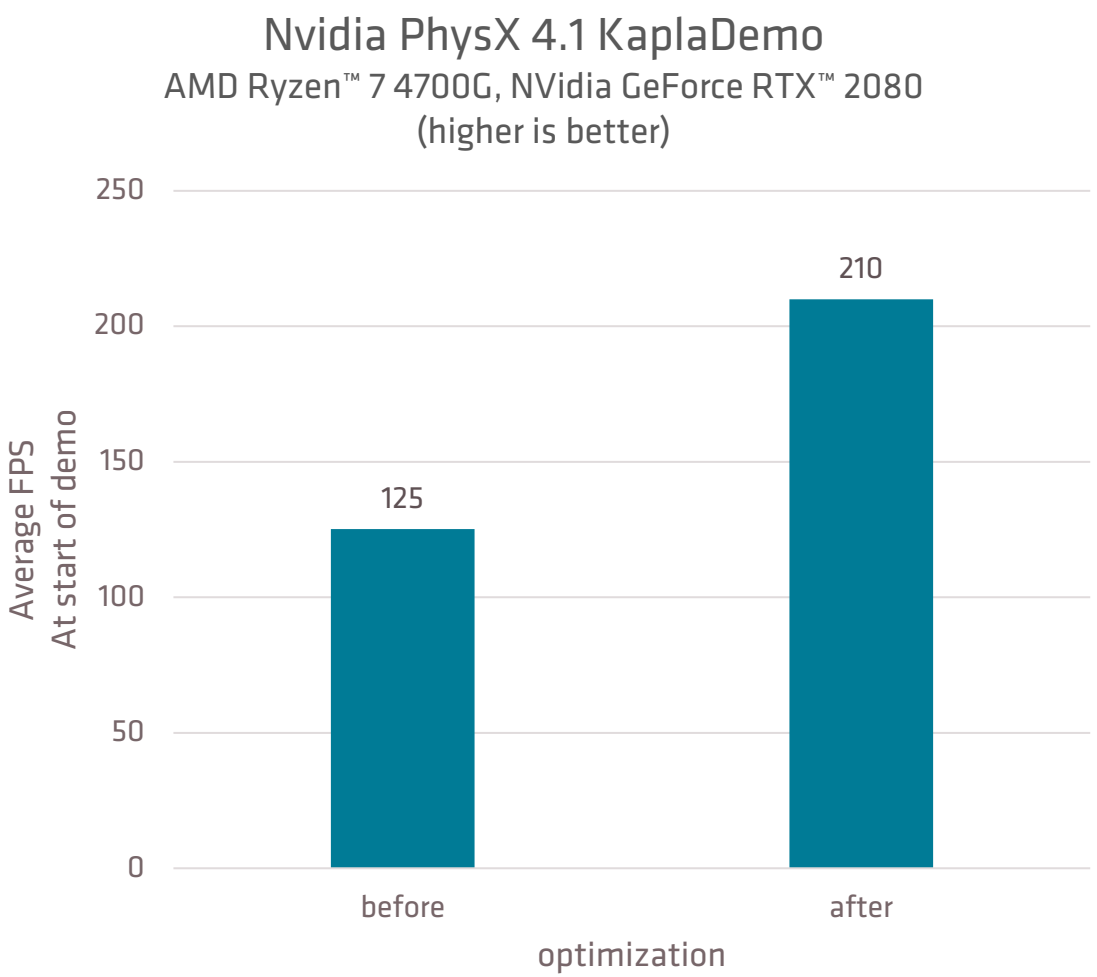
- True sharing:
  - Examples: shared_ptr, ref count, globals.
- False sharing:
  - Examples: Two locks share one cache line.
  - Common with tightly packed arrays.
- If you suspect false sharing, try alignas(64).
- *Performance of binaries compiled with Microsoft Visual Studio 2022 v17.0.5.*
- *Testing 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 version 21H2, 1920x1080 resolution. Actual results may vary.*

AVOID FALSE SHARING

```
#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;
    }
}
```

```
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;
}
```

USE SOFTWARE PREFETCH INSTRUCTIONS FOR LINKED DATA



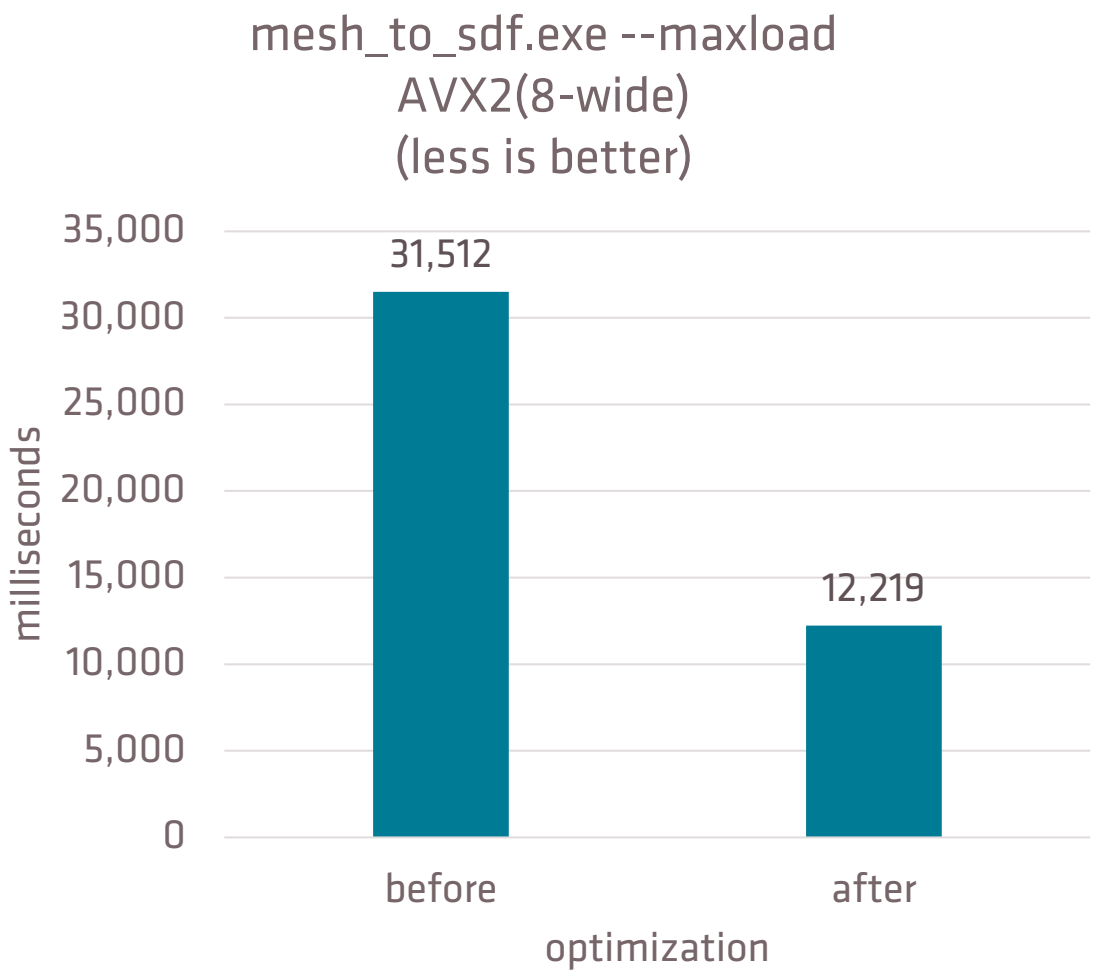
- Over 60% faster after optimization!
- *Performance of binaries compiled with Microsoft Visual Studio 2019 v16.8.3.*
- *Testing done by AMD technology labs, January 4, 2021 on the following system. Test configuration: AMD Ryzen™ 7 4700G, AMD Wraith Spire Cooler, 16GB (2 x 8GB DDR4-3200 at 22-22-22-52) 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 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 version 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 95% of users according to the December 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	0.00	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

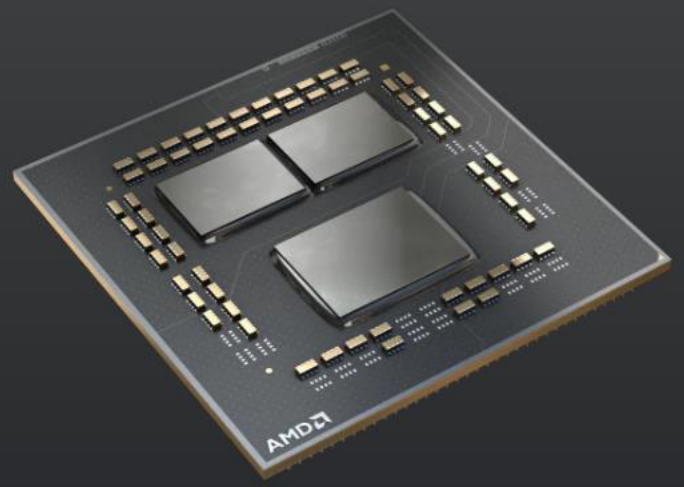
Line	Source	CAL_L2	MISALIGN_LOADS (PTI	EFFECTIVE_SW_PF (PT	FP_DISP_FAULTS (PTC)
162	`__m256 sign3 = sign(dp3);`		0.11	0.00	0.00
163					
164	`__m256 sum;`				
165	`sum = _mm256_add_ps( sign1, sign2 );`		0.19	0.00	0.00
166	`sum = _mm256_add_ps( sum, sign3 );`		0.15	0.00	0.01
167					
168	`__m256 cmp = _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		2 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	0.00

After the optimization, FP_DISPATCH_FAULTS 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.

SOFTWARE OPTIMIZATION GUIDES AT DEVELOPER.AMD.COM

"ZEN 4"

Software Optimization Guide for the AMD Zen4 Microarchitecture

- Software Optimization Guide for the AMD Zen4 Microarchitecture

"ZEN 3"

Software Optimization Guide for AMD EPYC™ 7003 Processors

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

"ZEN 2"

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

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

John.Hartwig@amd.com

Kenneth.Mitchell@amd.com

Design faster. Render faster. Iterate faster.

DISCLAIMER AND NOTICES

DISCLAIMER AND NOTICES

- Code sample on slide 64 is modified.
- Copyright (c) 2023 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 2023 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 – 2023, 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

AMD GPUOpen

AMD EPYC

AMD RYZEN

AMD RADEON