AMD EPYC

TUNING GUIDE
AMD EPYC 9004

High Performance Toolchain: Compilers, Libraries & Profilers

Publication	58020
Revision	1.4
Issue Date	September, 2023

Date	Version	Changes
July, 2022	0.1	Initial NDA partner release
Sep, 2022	0.2	Misc. updates
Nov, 2022	1.0	Initial public release
Dec, 2022	1.1	Minor errata corrections
Mar, 2023	1.2	Added 97xx OPN and AMD 3D V-Cache™ technology information
Jun, 2023	1.3	Second public release
Sep, 2023	1.4	Added preliminary version 4.1 information

Audience

This guide lists high performance tools to help develop your code using high-performance math libraries and compilers, and to profile your applications to more fully leverage the performance of AMD EPYC™ 9004 processors. This guide is not all-inclusive; please see Tools and SDKs for the latest information and instructions

Author

Ashish Jha

Note: All of the settings described in this Tuning Guide apply to all AMD EPYC 9004 Series Processors of all core counts with or without AMD 3D V-Cache™ except where explicitly noted otherwise.

Table of Contents

This page intentionally left blank.

Chapter 1 Introduction

AMD continues to actively develop a high-performance toolchain of compilers, libraries, and application profilers to meet the needs of High Performance Computing (HPC) and enterprise users and developers looking to better harness the capabilities and performance of AMD EPYC™ processors. These include:

- The AMD Optimizing CPU Compiler (AOCC) includes optimizations for each AMD EPYC processor generations. The open-source GNU Compiler Collection (GNU) platform compiler also supports AMD EPYC processors and is available based on the GNU General Public License.
- AMD also offers the high-performance AMD Optimizing CPU Libraries (AOCL) math library suite, which is optimized for the AMD "Zen" architectures. Developers can leverage AOCL for high-performance applications.
- The AMD uProf tool allows developers to extract more performance by gaining deeper insights into application characteristics and identifying optimization opportunities in those applications.

AMD AOCC, AOCL and uProf are now part of AMD Zen Software Studio. The current version of these tools is 4.1, which was released on August 4, 2023 and includes support for 4th Gen AMD EPYC processors (i.e., the "Zen 4" architecture). These tools are preferred for AMD EPYC processors. You will also be able to use GCC 12.1.0 and later because it supports the AVX512 ISA that was introduced in AMD EPYC 9004 Series Processors. GCC version 12.3.0 will have full support for AMD EPYC 9004 Series Processors. GCC 13.1 is currently released and includes full support for AMD EPYC 9004 Series Processors.

Visit AMD Zen Software Studio (Zen studio) to access everything mentioned in this chapter, including AMD libraries, compilers, performance analyzer, and user guides. You need not sign in to download forms; download begins as soon as you accept the terms. If you have any issues concerning the AMD toolchain, then please either contact your local AMD Field Application Engineer or send an email to toolchainsupport@amd.com.

This page intentionally left blank.

Chapter 2

AMD EPYC™ 9004 Series Processors

AMD EPYC™ 9004 Series Processors represent the fourth generation of AMD EPYC server-class processors. This generation of AMD EPYC processors feature AMD's latest "Zen 4" based compute cores, next-generation Infinity Fabric, next-generation memory & I/O technology, and use the new SP5 socket/packaging.

2.1 General Specifications

AMD EPYC 9004 Series Processors offer a variety of configurations with varying numbers of cores, Thermal Design Points (TDPs), frequencies, cache sizes, etc. that complement AMD's existing server portfolio with further improvements to performance, power efficiency, and value. Table 1-1 lists the features common to all AMD EPYC 9004 Series Processors.

Common Features of all AMD EPYC 9004 Series Processors	
Compute cores	Zen4-based
Core process technology	5nm
Maximum cores per Core Complex (CCX)	8
Max memory per socket	6 TB
Max # of memory channels	12 DDR5
Max memory speed	4800 MT/s DDR5
Max lanes Compute eXpress Links	64 lanes CXL 1.1+
Max lanes Peripheral Component Interconnect	128 lanes PCIe® Gen 5

Table 2-1: Common features of all AMD EPYC 9004 Series Processors

2.2 Model-Specific Features

Different models of 4th Gen AMD EPYC processors have different feature sets, as shown in Table 1-2.

AMD EPYC 9004 Series Processor (Family 19h) Features by Model		
Codename	"Genoa"*	"Bergamo"*
Model #	91xx-96xx	97xx
Max number of Core Complex Dies (CCDs)	12	8
Number of Core Complexes (CCXs) per CCD	1	2
Max number of cores (threads)	96 (192)	128 (256)
Max L3 cache size (per CCX)	1,152 MB (96 MB)♦	256 MB (16 MB)
Max Processor Frequency	4.4 GHz♦♦	3.15 GHz
Includes ♦AMD 3D V-Cache (9xx4X) and ♦♦high-frequency (9xx4F) models.		
*GD-122: 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 and plans only and subject to change. "Genoa" and "Bergamo" are codenames for AMD architectures and are not product names.		

Table 2-2: AMD EPYC 9004 Series Processors features by model

2.3 Operating Systems

AMD recommends using the latest available targeted OS version and updates. Please see AMD EPYC™ Processors Minimum Operating System (OS) Versions for detailed OS version information.

2.4 Processor Layout

AMD EPYC 9004 Series Processors incorporate compute cores, memory controllers, I/O controllers, RAS (Reliability, Availability, and Serviceability), and security features into an integrated System on a Chip (SoC). The AMD EPYC 9004 Series Processor retains the proven Multi-Chip Module (MCM) Chiplet architecture of prior successful AMD EPYC processors while making further improvements to the SoC components.

The SoC includes the Core Complex Dies (CCDs), which contain Core Complexes (CCXs), which contain the "Zen 4"-based cores. The CCDs surround the central high-speed I/O Die (and interconnect via the Infinity Fabric). The following sections describe each of these components.

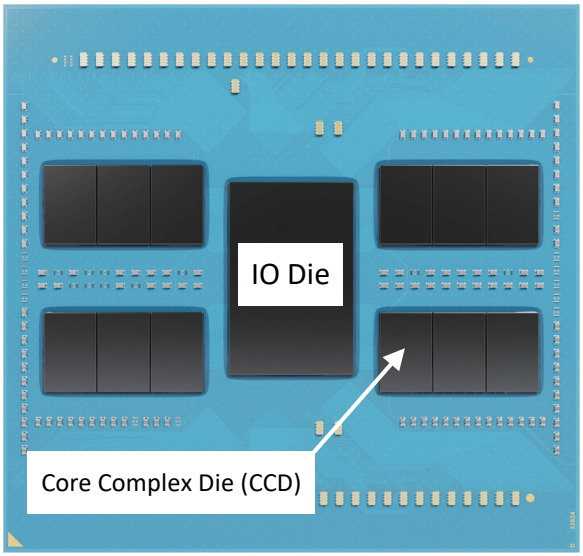

Figure 2-1: AMD EPYC 9004 configuration with 12 Core Complex Dies (CCD) surrounding a central I/O Die (IOD)

2.5 "Zen 4" Core

AMD EPYC 9004 Series Processors are based on the new "Zen 4" compute core. The "Zen 4" core is manufactured using a 5nm process and is designed to provide an Instructions per Cycle (IPC) uplift and frequency improvements over prior generation "Zen" cores. Each core has a larger L2 cache and improved cache effectiveness over the prior generation. Each "Zen 4" core includes:

- Up to 32 KB of 8-way L1 I-cache and 32 KB of 8-way of L1 D-cache
- Up to a 1 MB private unified (Instruction/Data) L2 cache.

Each core supports Simultaneous Multithreading (SMT), which allows 2 separate hardware threads to run independently, sharing the corresponding core's L2 cache.

2.6 Core Complex (CCX)

Figure 2-2 shows a Core Complex (CCX) where up to eight "Zen 4"-based cores share a L3 or Last Level Cache (LLC). Enabling Simultaneous Multithreading (SMT) allows a single CCX to support up to 16 concurrent hardware threads.

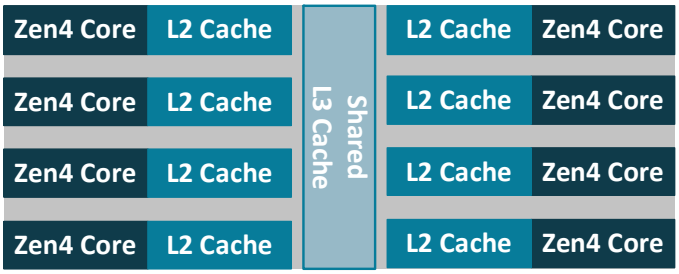

Figure 2-2: Top view of 8 compute cores sharing an L3 cache (91xx-96xx models)

2.7 Core Complex Dies (CCDs)

The Core Complex Die (CCD) in an AMD EPYC 9xx4 Series Processor may contain either one or two CCXs, depending on the processor (91xx-96xx "Genoa" vs. 97xx "Bergamo"), as shown in Figure 2-5.

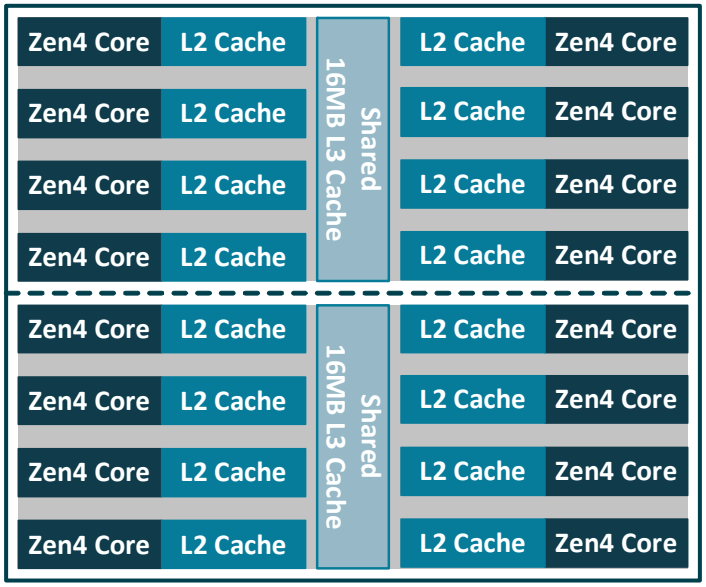

Figure 2-3: 2 CCXs in a single 4th Gen AMD EPYC 97xx CCD

Each of the Core Complex Dies (CCDs) in a 97xx model AMD EPYC 9004 Series Processor contains two CCXs (Figure 2-5):

AMD EPYC 9004 Series Processor	91xx-96xx	97xx
# of CCXs within a CCD	1	2

Table 2-3: CCXs per CCD by AMD EPYC model

You can disable cores in BIOS using one or both of the following approaches:

- Reduce the cores per L3 from 8 down to 7,6,5,4,3,2, or 1 while keeping the number of CCDs constant. This approach increases the effective cache per core ratio but reduces the number of cores sharing the cache.
- Reduce the number of active CCDs while keeping the cores per CCD constant. This approach maintains the advantages of cache sharing between the cores while maintaining the same cache per core ratio.

2.8 AMD 3D V-Cache™ Technology

AMD EPYC 9xx4X Series Processors include AMD 3D V-Cache™ die stacking technology that enables 97xx to achieve more efficient chiplet integration. AMD 3D Chiplet architecture stacks L3 cache tiles vertically to provide up to 96MB of L3 cache per die (and up to 1 GB L3 Cache per socket) while still providing socket compatibility with all AMD EPYC™ 9004 Series Processor models.

AMD EPYC 9004 Series Processors with AMD 3D V-Cache technology employ industry-leading logic stacking based on copper-to-copper hybrid bonding "bumpless" chip-on-wafer process to enable over 200X the interconnect densities of current 2D technologies (and over 15X the interconnect densities of other 3D technologies using solder bumps), which translates to lower latency, higher bandwidth, and greater power and thermal efficiencies.

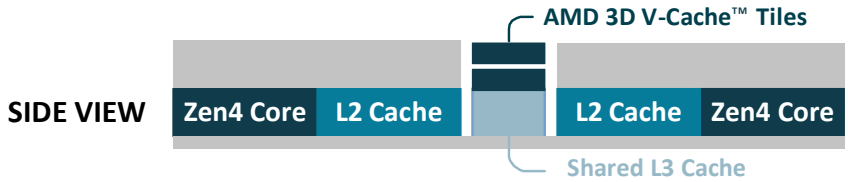

Figure 2-4: Side view of vertically-stacked central L3 SRAM tiles

AMD EPYC 9004 Series Processors	9xx4	9004X (with 3D V-Cache)
Max Shared L3 Cache per CCD	32 MB	96 MB

Table 2-4: L3 cache by processor model

Different OPNs also may have different numbers of cores within the CCX. However, for any given part, all CCXs will always contain the same number of cores.

2.9 I/O Die (Infinity Fabric™)

The CCDs connect to memory, I/O, and each other through an updated I/O Die (IOD). This central AMD Infinity Fabric™ provides the data path and control support to interconnect CCXs, memory, and I/O. Each CCD connects to the IOD via a dedicated high-speed Global Memory Interconnect (GMI) link. The IOD helps maintain cache coherency and additionally provides the interface to extend the data fabric to a potential second processor via its xGMI, or G-links. AMD EPYC 9004 Series Processors support up to 4 xGMI (or G-links) with speeds up to 32Gbps. The IOD exposes DDR5 memory channels, PCIe® Gen5, CXL 1.1+, and Infinity Fabric links.

All dies (chiplets) interconnect with each other via AMD Infinity Fabric technology. Figure 2-6 (which corresponds to Figure 2-2, above) shows the layout of a 96-core AMD EPYC 9654 processor. The AMD EPYC 9654 has 12 CCDs, with each CCD connecting to the IOD via its own GMI connection.

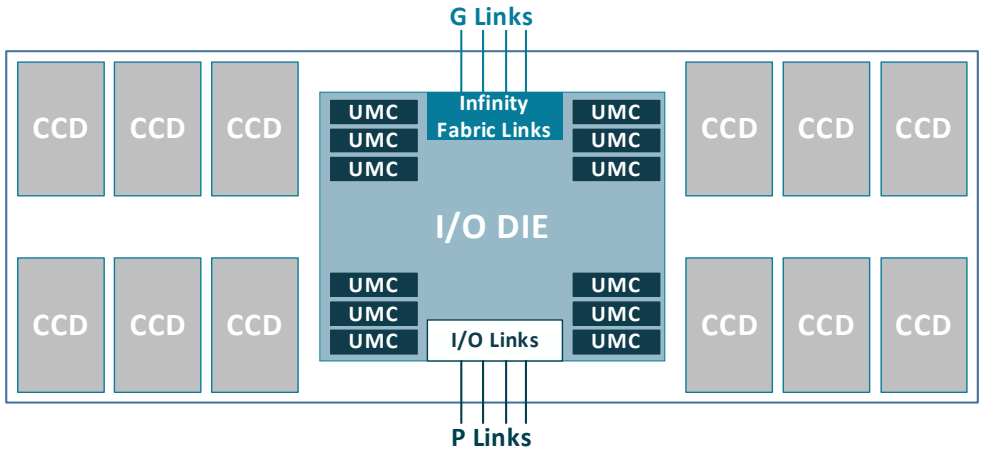

Figure 2-5: AMD EPYC 9654 processor internals interconnect via AMD Infinity Fabric (12 CCD processor shown)

AMD also provides "wide" OPNs (e.g. AMD EPYC 9334) where each CCD connects to two GMI3 interfaces, thereby allowing double the Core-to-I/O die bandwidth.

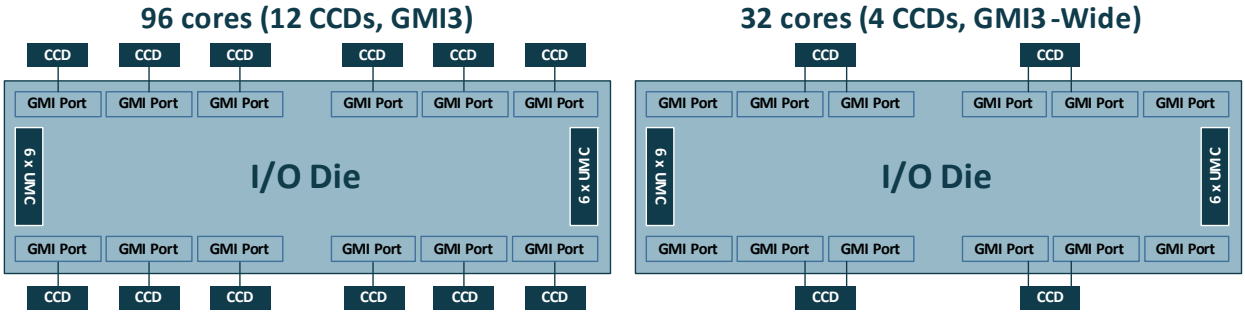

Figure 2-6: Standard vs. Wide GMI links

The IOD provides twelve Unified Memory Controllers (UMCs) that support DDR5 memory. The IOD also presents 4 'P-links' that the system OEM/designer can configure to support various I/O interfaces, such as PCIe Gen5, and/or CXL 1.1+.

2.10 Memory and I/O

Each UMC can support up to 2 DIMMs per channel (DPC) for a maximum of 24 DIMMs per socket. OEM server configurations may allow either 1 DIMM per channel or 2 DIMMs per channel. 4th Gen AMD EPYC processors can support up to 6TB of DDR5 memory per socket. Having additional and faster memory channels compared to previous generations of AMD EPYC processors provides additional memory bandwidth to feed high-core-count processors. Memory interleaving on 2, 4, 6, 8, 10, and 12 channels helps optimize for a variety of workloads and memory configurations.

Each processor may have a set of 4 P-links and 4 G-links. An OEM motherboard design can use a G-link to either connect to a second 4th Gen AMD EPYC processor or to provide additional PCIe Gen5 lanes. 4th Gen AMD EPYC processors support up to eight sets of x16-bit I/O lanes, that is, 128 lanes of high-speed PCIe Gen5 in single-socket platforms and up to 160 lanes in dual-socket platforms. Further, OEMs may either configure 32 of these 128 lanes as SATA lanes and/or configure 64 lanes as CXL 1.1+. In summary, these links can support:

- Up to 4 G-links of AMD Infinity Fabric connectivity for 2P designs.
- Up to 8 x16 bit or 128 lanes of PCIe Gen 5 connectivity to peripherals in 1P designs (and up to 160 lanes in 2-socket designs).
- Up to 64 lanes (4 P-links) that can be dedicated to Compute Express Link (CXL) 1.1+ connectivity to extended memory.
- Up to 32 I/O lanes that can be configured as SATA disk controllers.

2.11 Visualizing AMD EPYC 9004 Series Processors (Family 19h)

This section depicts AMD EPYC 9004 Series Processors that have been set up with four nodes per socket (NPS=4). Please see "NUMA Topology" on page 10 for more information about nodes.

2.11.1 Models 91xx-96xx ("Genoa")

4th Gen AMD EPYC 9004 processors with model numbers 91xx-96xx have up to 12 CCDs that each contain a single CCX, as shown below.

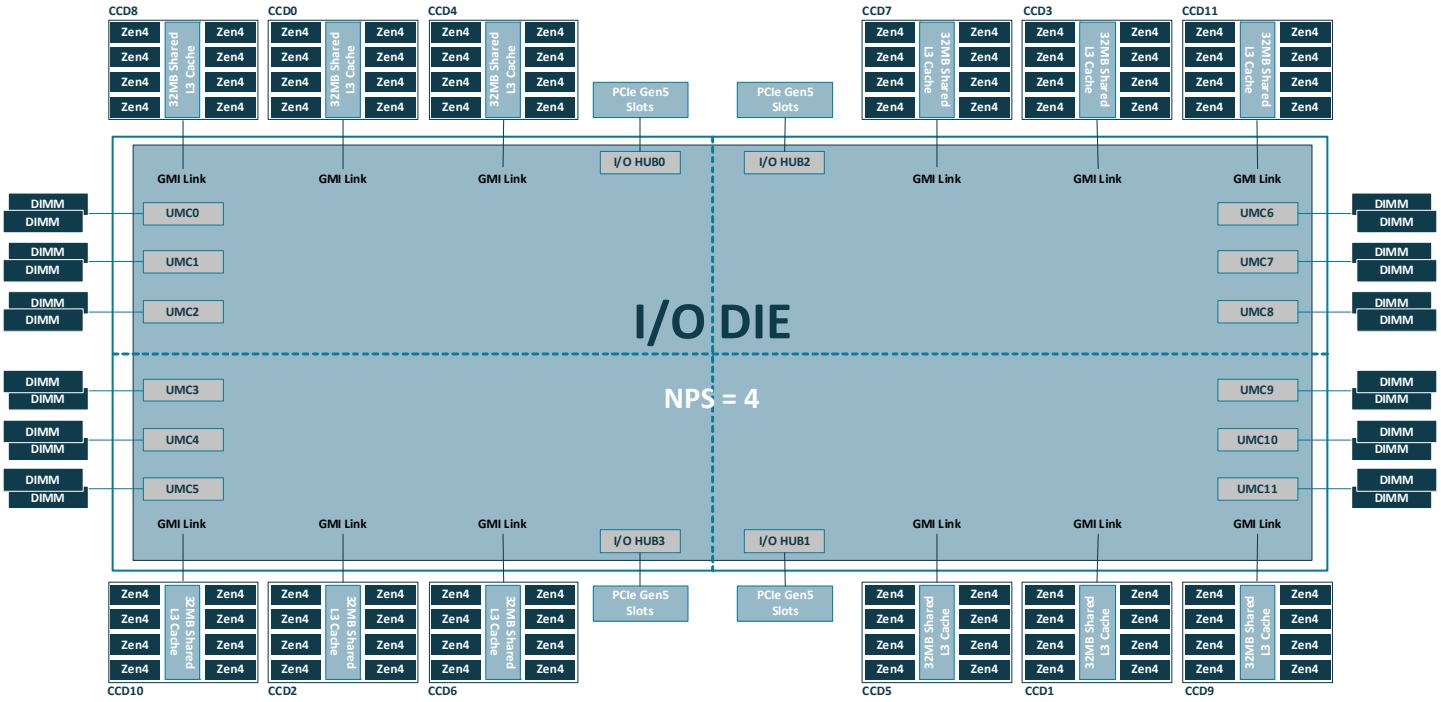

Figure 2-7: The AMD EPYC 9004 SoC consists of up to 12 CCDs and a central IOD for 91xx-96xx models, including "X" OPNs

2.11.2 Models 97xx ("Bergamo")

97xx 4th Gen AMD EPYC 9004 Series Processors with model numbers 97xx have up to 8 CCDs that each contain two CCXs, as shown below.

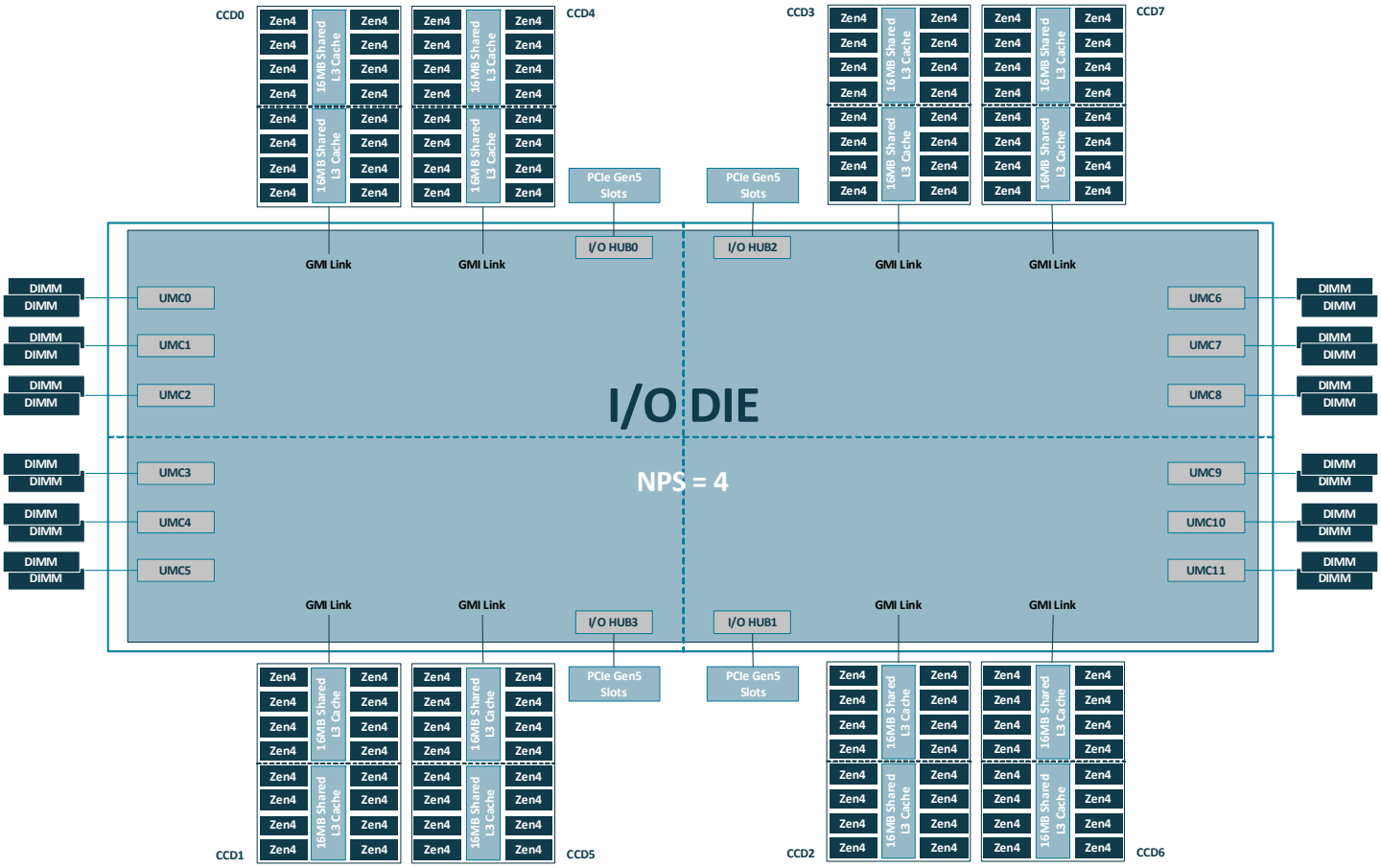

Figure 2-8: The AMD EPYC 9004 System on Chip (SoC) consists of up to 8 CCDs and a central IOD for 97xx models

2.12 NUMA Topology

AMD EPYC 9004 Series Processors use a Non-Uniform Memory Access (NUMA) architecture where different latencies may exist depending on the proximity of a processor core to memory and I/O controllers. Using resources within the same NUMA node provides uniform good performance, while using resources in differing nodes increases latencies.

2.12.1 NUMA Settings

A user can adjust the system **NUMA Nodes Per Socket** (NPS) BIOS setting to optimize this NUMA topology for their specific operating environment and workload. For example, setting NPS=4 as shown in "Memory and I/O" on page 8 divides the processor into quadrants, where each quadrant has 3 CCDs, 3 UMCs, and 1 I/O Hub. The closest processor-memory I/O distance is between the cores, memory, and I/O peripherals within the same quadrant. The furthest distance is between a core and memory controller or IO hub in cross- diagonal quadrants (or the other processor in a 2P configuration). The locality of cores, memory, and IO hub/devices in a NUMA-based system is an important factor when tuning for performance.

The NPS setting also controls the interleave pattern of the memory channels within the NUMA Node. Each memory channel within a given NUMA node is interleaved. The number of channels interleaved decreases as the NPS setting gets more granular. For example:

- A setting of NPS=4 partitions the processor into four NUMA nodes per socket with each logical quadrant configured as its own NUMA domain. Memory is interleaved across the memory channels associated with each quadrant. PCIe devices will be local to one of the four processor NUMA domains, depending on the IOD quadrant that has the corresponding PCIe root complex for that device.

- A setting of NPS=2 configures each processor into two NUMA domains that groups half of the cores and half of the memory channels into one NUMA domain, and the remaining cores and memory channels into a second NUMA domain. Memory is interleaved across the six memory channels in each NUMA domain. PCIe devices will be local to one of the two NUMA nodes depending on the half that has the PCIe root complex for that device.

- A setting of NPS=1 indicates a single NUMA node per socket. This setting configures all memory channels on the processor into a single NUMA node. All processor cores, all attached memory, and all PCIe devices connected to the SoC are in that one NUMA node. Memory is interleaved across all memory channels on the processor into a single address space.

- A setting of NPS=0 indicates a single NUMA domain of the entire system (across both sockets in a two-socket configuration). This setting configures all memory channels on the system into a single NUMA node. Memory is interleaved across all memory channels on the system into a single address space. All processor cores across all sockets, all attached memory, and all PCIe devices connected to either processor are in that single NUMA domain.

You may also be able to further improve the performance of certain environments by using the **LLC (L3 Cache) as NUMA** BIOS setting to associate workloads to compute cores that all share a single LLC. Enabling this setting equates each shared L3 or CCX to a separate NUMA node, as a unique L3 cache per CCD. A single AMD EPYC 9004 Series Processor with 12 CCDs can have up to 12 NUMA nodes when this setting is enabled.

Thus, a single EPYC 9004 Series Processor may support a variety of NUMA configurations ranging from one to twelve NUMA nodes per socket.

Note: If software needs to understand NUMA topology or core enumeration, it is imperative to use documented Operating System (OS) APIs, well-defined interfaces, and commands. Do not rely on past assumptions about settings such as APICID or CCX ordering.

2.13 Dual-Socket Configurations

AMD EPYC 9004 Series Processors support single- or dual-socket system configurations. Processors with a 'P' suffix in their name are optimized for single-socket configurations (see the "Processor Identification" chapter) only. Dual-socket configurations require both processors to be identical. You cannot use two different processor Ordering Part Numbers (OPNs) in a single dual-socket system.

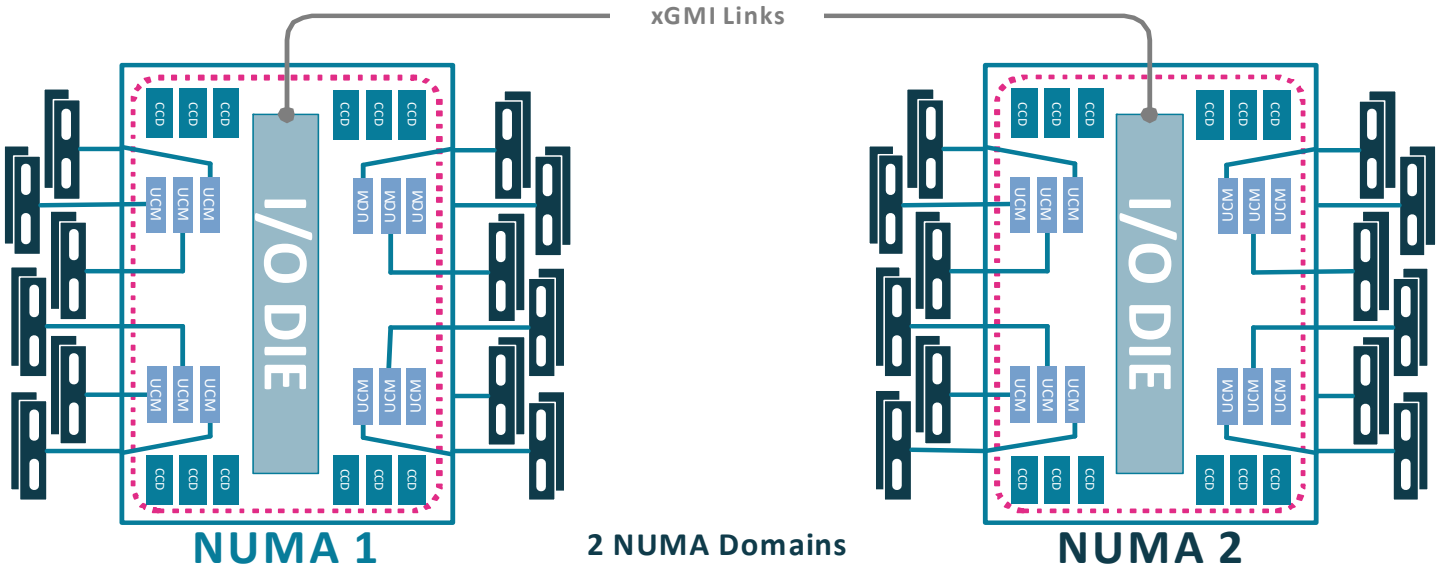

Figure 2-9: Two EPYC 9004 Processors connect through 4 xGMI links (NPS1)

In dual-socket systems, two identical EPYC 9004 series SoCs are connected via their corresponding External Global Memory Interconnect [xGMI] links. This creates a high bandwidth, low latency interconnect between the two processors. System manufacturers can elect to use either 3 or 4 of these Infinity Fabric links depending upon I/O and bandwidth system design objectives.

The Infinity Fabric links utilize the same physical connections as the PCIe lanes on the system. Each link uses up to 16 PCIe lanes. A typical dual socket system will reconfigure 64 PCIe lanes (4 links) from each socket for Infinity Fabric connections. This leaves each socket with 64 remaining PCIe lanes, meaning that the system has a total of 128 PCIe lanes. In some cases, a system designer may want to expose more PCIe lanes for the system by reducing the number of Infinity Fabric G-Links from 4 to 3. In these cases, the designer may allocate up to 160 lanes for PCIe (80 per socket) by utilizing only 48 lanes per socket for Infinity Fabric links instead of 64.

A dual-socket system has a total of 24 memory channels, or 12 per socket. Different OPNs can be configured to support a variety of NUMA domains.

Chapter 3 Compilers

AMD AOCC is the performance compiler optimized for each AMD EPYC processor generation. AOCC consists of a C/C++ compiler (`clang`) and a Fortran compiler (`flang`). The current available AOCC is version 4.0, which is optimized and targeted for the Zen 4 architecture, including latest AVX512 ISA support. You can download AOCC from https://www.amd.com/en/developer/aocc.html.

The open-source GCC platform compiler is also supported. The "Zen 4" architecture introduces AVX512 ISA, which is supported by GCC 12.1.0 and above. GCC version 12.3.0 will have full support for AMD EPYC 9004 Series Processors. GCC 13.1 is currently released, and it fully supports AMD EPYC 9004 Series processors.

3.1 AMD Optimizing CPU Compiler (AOCC)

The AOCC compiler system is a high-performance, production quality code generation tool that offers various options to developers building and optimizing C, C++ and Fortran applications targeting 32-bit and 64-bit Linux® platforms. It offers a high level of advanced optimizations, multi-threading, and processor support that includes global optimization, vectorization, inter-procedural analyses, loop transformations, and code generation. AOCC 4.1 is based on the LLVM 16.0.3 release (The LLVM Compiler Infrastructure*, May 3, 2023). It is tuned for AMD Family 17h processors and includes the following features:

- Machine-dependent optimizations for better performance on AMD EPYC 9004 Series Processors.
- Enhanced high-level optimizations for AMD EPYC 9004 Series processors.
- Improved `flang` as the default Fortran front end with added F2008 features and improvements.
- OpenMP 4.5 and OpenMP 5.0 standards for C/C++ programming.
- OpenMP 4.5 standard for Fortran programming.
- LLVM linker (`lld`) as the default linker.
- Optimized libraries including tuned for AMD AOCL-LibM (AMD Math Library) 4.1.
- Default Optimization level of `-O2`, and `-fPIC` and `-fPIE` options made default.
- Bit reproducibility support improved for C/C++ and Fortran.
- New AOCC Optimization Report (AOR) tool to display optimization details.
- Tested on RHEL 8.6 and 9.0, CentOS 8, SLES 15 SP3, and Ubuntu 22.04 LTS.

AOCC compiler binaries depend on Linux® systems having `glibc` version 2.17 or later.

This chapter includes useful compiler options you can pass to `clang` and `flang`.

3.1.1 AOCC Clang/Clang++

Clang is a C, C++, and Objective-C compiler that encompasses preprocessing, parsing, optimization, code generation, assembly, and linking. Clang supports the `-march=znver4` flag to enable best code generation and tuning for 4th Gen AMD EPYC Series processors.

3.1.2 AOCC FLang

Flang is the Fortran front-end designed for LVVM integration and suitable for interoperability with Clang/LLVM. It supports all `clang` compiler options plus a few `flang`-specific options. AMD extends the GitHub version of `flang` available from https://github.com/flang-compiler/flang*, which in turn is based on the NVIDIA/PGI commercial Fortran compiler.

3.1.3 Clang and Flang Options

Tables 3-1 and 3-2 list AOCC `clang` and `flang` compiler options, respectively.

Architecture	
Generate instructions that run on 4th Gen AMD EPYC Series CPUs.	`-march=znver4`
Generate instructions that run on 3rd Gen AMD EPYC Series CPUs.	`-march=znver4`
Generate instructions for the local machine	`-march=native`
Optimization Levels	
Disable all optimizations	`-O0`
Minimal level speed and code optimization	`-O1/ -O`
Moderate level optimization	`-O2`
Aggressive optimization	`-O3`
Maximize performance	`-Ofast`
Enable link time optimizations	`-flto`

Enable loop optimizations	`-funroll-loops`
	`-enable-licm-vrp`
	`-enable-partial-unswitch`
	`-fuse-tile-inner-loop`
	`-unroll-threshold`

Enable advanced loop optimizations	`-unroll-aggressive`

Enable function level optimizations	`-fitodcalls`
	`-function-specialize`
	`-finline-aggressive`
	`-inline-recursion=[1..4] (use with flto)`
	`-do-block-reordering={none, normal, aggressive}`

Enable advanced vectorization	`-enable-strided-vectorization` `-enable-epilog-vectorization`
Enable memory layer optimizations	`-fremap-arrays (use with flto)`
Profile guided optimizations	`-fprofile-instr-generate (1st invoc.)` `-fprofile-instr-use (2nd invocation)`
OpenMP®	`-fopening`

Table 3-1: AOCC Clang compiler options

For enabling memory stores, memory bandwidth workloads	`-fnt-store`
Enable removal of all unused array computation	`-reduce-array-computations=3`
Other Options	
Enable faster, less precise math operations (part of Ofast)	`-ffast-math -freciprocal-math`
OpenMP threads and affinity (N number of cores)	`export OMP_NUM_THREADS=N export GOMP_CPU_AFFINITY="0-{N-1}"`
Enabling vector library	`-fveclib=AMDLIBM`
Link to AMD library	`-L/libm-install-dir/lin -lalm`
For Fortran workloads	
Compile free form Fortran	`-ffree-form`

Table 3-1: AOCC Clang compiler options

3.2 GCC Compiler

GCC version 12.3.0 will have full support for the "Zen 4" architecture. GCC 13.1 is currently released, and it fully supports the "Zen 4" architecture. Please see "Introduction" on page 1 for information about features coming in future versions.

Architecture	
Generate instructions that run on 4th Gen AMD EPYC Series CPUs	`-march=znver4`
Generate instructions that run on 3rd Gen AMD EPYC Series CPUs	`-march=znver4`
Generate instructions for the local machine	`-march=native`
Generate AVX512 foundation instructions (GCC 12.1.0 and above) and its subsets	`-mavx512f along with combination of AVX512 subsets such as -mavx512vpopcntdq, -mavx512vp2intersect, -mavx512vnni, -mavx512vl, -mavx512vbmi2, -mavx512vbmi, -mavx512pf, -mavx512ifma, -mavx512er, -mavx512dq, -mavx512cd, -mavx512bw, -mavx512bitalg, -mavx512bf16, -mavx5124vnniw, -mavx5124fmaps`
Optimization Levels	
Disable all optimizations	`-O0`
Minimal level speed and code optimization	`-O1/ -O`
Moderate level optimization	`-O2`
Aggressive optimization	`-O3`
Maximize performance	`-Ofast`
Additional Optimizations	
Enable link time optimizations	`-flto`
Enable unrolling	`-funroll-all-loops`

Table 3-2: GCC compiler options

Generate memory preload instructions	`-fprefetch-loop-arrays --param prefetch-latency=300`
Profile-guided optimization	`-fprofile-generate (1st invocation)` `-fprofile-use (2nd invocation)`
OpenMP	`-fopenmp`
Other Options	
Enable generation of code that follows IEEE arithmetic	`-mieee-fp`
Enable faster, less precise math operations (part of Ofast)	`-ffast-math`
Compile free form Fortran	`-ffree-form`
OpenMP threads and affinity (N number of cores)	`export OMP_NUM_THREADS=N` `export GOMP_CPU_AFFINITY="0-{N-1}"`
Link to AMD library	`-L/libm-install-dir/lin -lalm`

Table 3-2: GCC compiler options (Continued)

Chapter 4 High Performance Math Libraries

The AMD Optimizing CPU Libraries (AMD CPU math library suite, called AOCL) are a suite of math libraries that offers optimized mathematical and scientific operations such as linear algebra with sparse and dense operations, Fast Fourier Transform, random number generator, and cryptography.

AOCL Version 4.1 is the current version at the time of publishing this Tuning Guide. Visit https://www.amd.com/en/developer/aocl.html for more details on the AOCL libraries, tar balls of FrameBuilder libraries, and instructions on how to build AOCL libraries from source.

4.1 AMD Optimizing CPU Libraries (AOCL)

AMD Optimizing CPU Libraries (AOCL) are a set of numerical libraries tuned specifically for the AMD EPYC processor family. They include a simple interface that takes advantage of the latest hardware innovations.

AOCL consists of the following libraries:

- **BLIS (BLAS Library):** Portable open-source software framework for performing high-performance Basic Linear Algebra Subprograms (BLAS) functionality.
- **libFLAME (LAPACK):** Portable library for dense matrix computations that provides the functionality present in the Linear Algebra Package (LAPACK).
- **AMD-FFTW (Fastest Fourier Transform in the West):** Comprehensive collection of fast C routines for computing the Discrete Fourier Transform (DFT) and various special cases.
- **LibM (AMD Core Math Library):** Software library containing a collection of basic math functions optimized for x86-64 processor based machines.
- **ScaLAPACK: Library of high-performance linear algebra routines for parallel distributed memory machines. It depends on** external libraries including BLAS and LAPACK for linear algebra computations.
- **AMD Random Number Generator (RNG):** Pseudo random number generator library.
- **AMD Secure RNG:** library that provides APIs to access the cryptographically secure random numbers generated by the AMD hardware random number generator.
- **AOCL-Sparse:** Library containing the basic linear algebra subroutines for sparse matrices and vectors optimized for AMD "Zen"-based processors, including EPYC, Ryzen™, and Threadripper™ PRO.
- **AOCL-Cryptography:** AMD Zen architecture optimized implementation of cryptographic functions (AES Encryption/Decryption, SHA2 Digest).

- **AOCL-LibMem:** AMD Zen architecture optimized implementation of memory/string functions.
- **AOCL enabled MUMPS library:** MUMPS (MUltifrontal Massively Parallel Solver*) is an open-source package for solving systems of linear equations of the form `Ax = b`.
- **AOCL Codec:** Supports `lz4`, `zlib`/`deflate`, `lzma`, `zstd`, `bzip2`, `snappy`, and `lz4hc`.
- **AOCL Compression:** AOCL-Compression is a software framework of lossless data compression and decompression methods tuned and optimized for AMD "Zen"-based CPUs.

If you have any issues concerning AOCL, then please either contact your local AMD Field Application Engineer or send an email to toolchainsupport@amd.com.

4.1.1 BLIS

BLIS is a portable open-source software framework for instantiating high-performance Basic Linear Algebra Subprograms (BLAS), such as dense linear algebra libraries. The framework isolates essential kernels of computation to enable optimizations of most of its commonly used and computationally intensive operations. Select kernels have been optimized for the AMD EPYC processor family. You may download the source code from https://github.com/amd/blis*. AMD offers the optimized version of BLIS that supports C, FORTRAN, and C++ template interfaces.

4.1.2 libFLAME

AOCL-libFLAME is a high performant implementation of Linear Algebra PACKage (LAPACK). LAPACK provides routines for solving systems of linear equations, least-squares problems, eigenvalue problems, singular value problems, and the associated matrix factorizations. It is extensible, easy to use, and available under an open-source license. libFLAME is a C-only implementation. Applications relying on standard Netlib LAPACK interfaces can utilize libFLAME with virtually no changes to their source code. You may download the source code from https://github.com/amd/libflame*.

4.1.3 FFTW

The AMD-optimized version of Fast Fourier Transform Algorithm (FFTW) is an open-source implementation of FFTW that offers a comprehensive collection of fast C routines for computing the Discrete Fourier Transform (DFT) and various special cases thereof that are optimized for AMD EPYC and other AMD "Zen"-based processors. It can compute transforms of real and complex valued arrays of arbitrary size and dimension. You can download the latest stable release from https://github.com/amd/amd-fftw*.

4.1.4 LibM

AMD LibM is a software library containing a collection of basic math functions optimized for x86-64 processor-based machines. It provides many routines from the list of standard C99 math functions. Applications can link into the AMD LibM library and invoke math functions instead of using the compiler's default math functions for better accuracy and performance.

4.1.5 ScaLAPACK

ScaLAPACK is a library of high-performance linear algebra routines for parallel distributed memory machines. It depends on external libraries including BLAS and LAPACK for Linear Algebra computations. AMD's optimized version of ScaLAPACK enables using the BLIS and libFLAME libraries that offer optimized dense matrix functions and solvers for the AMD EPYC family of CPUs. You can install ScaLAPACK from either source or pre-built binaries. You can clone the ScaLAPACK for AMD source from https://github.com/amd/scalapack*. You can access a pre-built AMD optimized ScaLAPACK from the AOCL master installer tar file.

4.1.6 AMD Random Number Generator

The AMD Random Number Generator (RNG) library is a pseudo-random number generator library that provides a comprehensive set of statistical distribution functions and various uniform distribution generators (base generators) including Wichmann-Hill and Mersenne Twister. The library contains six base generators and twenty-three distribution generators. In addition, you can supply a custom-built generator as the base generator for all the distribution generators.

4.1.7 AMD Secure RNG

The AMD Secure RNG is a library that provides the APIs to access the cryptographically secure random numbers generated by the AMD hardware based RNG. These are high-quality, robust random numbers designed for the cryptographic applications. The library makes use of RDRAND and RDSEED x86 instructions exposed by the AMD hardware. The applications can just link to the library and invoke a single or a stream of random numbers. The random numbers can be of 16-bit, 32-bit, 64-bit, or arbitrary size bytes.

4.1.8 AOCL-Sparse

AOCL-Sparse is a library containing basic linear algebra subroutines for the sparse matrices and vectors optimized for AMD EPYCTM and other AMD "Zen"-based processors. It is designed to be used with C and C++.

4.1.9 AOCL- Cryptography

This is a new cryptography library with AMD "Zen" support and optimizations for standard encryption algorithms, including:

- AES-CBC, CFB, GCM, and OFB modes
- Support for 128/192/256-bit keys
- SHA-2 hashing functions (224, 256, 384, and 512)
- Hardware acceleration is supported only on Intel Advanced Encryption Standard New Instructions (AES-NI) enabled platforms
- Libcrypto API support for OpenSSL

4.1.10 AOCL- LibMem

AOCL-LibMem contains AMD Zen architecture-optimized functions for memory and string functions such as `memcpy`, `mempcpy`, `memmove`, `memset`, and `memcmp`.

4.1.11 AOCL-Compression

AOCL-Compression library supports LZ4, ZLIB/DEFLATE, LZMA, ZSTD, BZIP2, Snappy, and LZ4HC based compression and decompression methods.

4.2 AOCL Tuning Guidelines

This section provides general AOCL tuning guidelines. Please see the AOCL User Guide for more detailed information.

4.2.1 AOCL Dynamic

The AOCL Dynamic feature enables AOCL BLIS to dynamically change the number of threads. This feature is enabled by default; however, it can be enabled or disabled the configuration time by executing the following commands, respectively:

```
--enable-aocl-dynamic
--disable-aocl-dynamic
```

You can also specify the preferred number of threads using the environment variables `BLIS_NUM_THREADS` or `OMP_NUM_THREADS`. If you specify both variables, then `BLIS_NUM_THREADS` takes precedence. Table 4-1 summarizes how the number of threads is determined based on the status of AOCL Dynamic and the user configuration using the variable BLIS_NUM_THREADS:

AOCL Dynamic	BLIS_NUM_THREADS	Number of threads used by BLIS
Disabled	Not set	Number of cores.
Disabled	Set	`BLIS_NUM_THREADS`
Enabled*	Not set	Number of threads determined by AOCL Dynamic
Enabled*	Set	Either the minimum of BLIS_NUM_THREADS or the number of threads determined by AOCL.
*AOCL Dynamic currently only supports the DGEMM, DGEMMT, DTRSM, and DSYRK APIs. For the other APIs, the threads selection will be the same as when AOCL Dynamic is disabled.		

Table 4-1: AOCL Dynamic

4.2.2 BLIS DGEMM Multi-thread Tuning

You can use a BLIS library on multiple platforms and applications. Multi-threading adds more configuration options at runtime by controlling the number of threads and CPU affinity settings that can be tuned to get the best performance for your requirements. Execute one of the following commands to obtain the best DGEMM multi-thread performance on 4th Gen AMD EPYC processors:

- **Thread size up to 16 (< 16):**
  `OMP_PROC_BIND=spread OMP_NUM_THREADS=<NT>./test_gemm_blis.x`
- **Thread size above 16 (>= 16):**
  `OMP_PROC_BIND=spread OMP_NUM_THREADS=<NT> numactl --interleave=all ./test_gemm_blis.x`

4.2.3 Performance Suggestions for Skinny Matrices

BLIS provides a selective packing for GEMM when one or two dimensions of a matrix is exceedingly small. This feature is only available when `sup` handling is enabled by default. For optimal performance:

```
C = beta*C + alpha*A*B
Dimension (Dim) of A - m x k Dim(B) - k x n Dim(c) - m x n
Assume row-major.
IF Dim(A) >> Dim(B)
$BLIS_PACK_A=1 ./test_gemm_blis.x - will give a better performance.
IF Dim(A) << Dim(B)
$BLIS_PACK_B=1 ./test_gemm_blis.x - will give a better performance.
```

4.2.4 AMD-Optimized FFTW

This subsection provides general tuning guidelines to optimize the performance out of AMD-optimized FFTW. lease see the AOCL User Guide for more detailed information.

- Use the configure option `--enable-amd-opt` to build the targeted library. This option enables all the improvements and optimizations meant for AMD EPYC CPUs. This is the mandatory master optimization switch that must be set for enabling any other optional configure options such as:
  - `--enable-amd-mpifft`
  - `--enable-amd-mpi-vader-limit`
  - `--enable-amd-trans`
  - `--enable-amd-fast-planner`
  - `--enable-amd-top-n-planner`
  - `--enable-amd-app-opt`
- Use the `-opatient` planner FFTW flag for best performance. For example:

  ```
  $ ./bench -opatient -s icf65536
  ```

  Where:
  - `-s` is for a speed/performance run.
  - `i` is for in-place.
  - `c` is for complex datatype.
  - `f` is for forward transform.
- When configured with `--enable-openmp` and running a multi-threaded test, set the OpenMP variables as follows:
  - `OMP_PROC_BIND=TRUE`
  - `OMP_PLACES=cores`

After setting the variables, run the test bench executable binary using `numactl` as follows:

```
numactl --interleave=0,1,2,3 ./bench -opatient -onthreads=64 -s icf65536
```

Where:

`numactl --interleave=0,1,2,3` sets the memory interleave policy on nodes 0, 1, 2, and 3.

When running a MPI FFTW test, set the appropriate MPI mapping, binding, and rank options. For example, to run 64 MPI rank FFTW on a 64-core AMD EPYCTM processor:

```
mpirun --map-by core --rank-by core --bind-to core -np 64 ./mpi-bench -opatient -s icf65536
```

4.2.5 AOCL-libFLAME Multi-Threading

AOCL-libFLAME in AOCL Versions 4.0 and later supports multi-threading using OpenMP in selected APIs. This feature is enabled by default when AOCL-libFLAME is compiled with `--enable-amd-flags` or `--enable-amd-aocc-flags`. However, you can disable multi-threading by setting `--enablemultithreading=no`.

The selected LAPACK interface APIs that support multi-threading automatically choose optimal number of threads. However, you can explicitly set the number of threads through the environment variable or OpenMP runtime APIs. In such a scenario, the number of threads is selected as follows:s:

- **User-specified threads > AOCL-libFLAME computed optimal threads:** AOCL-libFLAME computed optimal threads.
- **User specified threads < AOCL-libFLAME computed optimal threads:** User specified threads.

Chapter 5 Application Performance Analysis & Optimization

AMD µProf is a performance analysis tool for applications running on the Linux® and Windows® operating systems. It allows developers to better understand the runtime performance of their application and to identify ways to improve its performance. Users can download it for free from https://www.amd.com/en/developer/uprof.html. uProf version 4.0 is the latest available and includes full support for the AMD "Zen 4" architecture.

5.1 AMD µProf – Profiling Tool

AMD µProf offers the following functions:

- **Performance Analysis:** The CPU profile identifies application runtime performance bottlenecks.
- **System Analysis:** The Performance Counter Monitor utility monitors system performance metrics such as IPC and memory bandwidth.
- **Roofline Analysis:** AMDuProfPcm provides basic roofline modeling that relates the application performance to memory traffic and floating point computational peaks. This is a visual performance model offering insights on improving the parallel software for floating point operations. It helps to characterize an application and to identify whether a benchmark is memory or compute bound.
- **Power Profiling:** System-wide power profiles monitor system thermal and power characteristics.
- **Energy Analysis:** Power Application Analysis identifies energy hotspots in the application (Windows only).

You can use AMD µProf to:

- Analyze the performance of either one or more process(es) or the entire system.
- Track down performance bottlenecks (hotspots) in the source code.
- Identify ways to optimize the source code for better performance and power efficiency.
- Examine the behavior of kernel, driver, and system modules.
- Analyze thread concurrency.
- Observe frequency, thermal and power characteristics (power profiling).
- Observe system metrics like IPC, Core effective frequency, memory bandwidth, etc.
- Monitor the frequency, thermal, and energy metrics of various system components.

Processed profile data is stored in databases that you can use to generate reports later. Profile reports are available in comma-separated-value (CSV) format for use with spreadsheets.

μProf 4.0 was released on November 10th, 2022 with the following new features:

- **CPU Profiling GUI:** Timeline for CPU Profile events, profile duration filter in timeline, bottom-up view of `callstack` samples, thread level `callgraph` support, and thread level `flamegraph` support.
- **Holistic Analysis View:** Analyze CPU, GPU, and OS together in a holistic GUI on Linux platforms. The holistic analysis view allows you to analyze OS scheduling events, System calls, POSIXthread sync APIs, GPU activities, and MPI API event tracing.
- **OS Tracing:** Linux OS events that include thread state analysis, kernel block I/O analysis, pagefault analysis, and `memtrace` (memory alloc/dealloc) analysis.
- **MPI Tracing:** Linux HPC analysis that traces MPI applications based on MPICH and derivatives.
- **GPU Tracing:** Linux GPU trace analysis for AMD Instinct™ MI100 and MI200 accelerators.
- **GPU Profiling:** GPU kernel dispatch analysis on Linux for AMD Instinct MI100 and MI200 accelerators.
- **AMDuProfSys:** A new system analysis tool that captures system information that can help debug issues.

5.1.1 Performance Monitoring Tool (AMDuProfPcm)

The AMDuProfPcm system analysis utility helps monitor basic performance monitoring metrics. This utility periodically collects the CPU Core, L3, and DF performance events count values and reports various metrics. On Linux systems, AMDuProfPcm uses the `msr` driver and requires either root privileges or read/write permissions for `/dev/cpu/*/msr` devices. This example collects IPC and all cache levels metrics for all the cores and aggregate at system/package/CCD for a STREAM benchmark:

```
$ ./AMDuProfPcm -m ipc,l1,l2,l3 -a -A system,package,ccd -C -o /tmp/pcm-ipc-sys.csv --
<stream…>
```

Figure 5-1 shows the result of one such an analysis.

CORE METRICS					
Metric	System (Aggregated)	Package (Aggregated)-0	Package (Aggregated)-1	CCD (Aggregated)-0	CCD (Aggregated)-1
Utilization (%)	9.55	9.45	9.66	10.17	9.32
Eff Freq	3262.11	3257.45	3266.7	3258.25	3257.94
IPC (Sys + User)	0.13	0.11	0.14	0.11	0.11
CPI (Sys + User)	7.97	9.11	7.09	9.31	9.43
Branch Misprediction Ratio	0	0	0	0	0
IC(32B) Fetch Miss Ratio	0.01	0.01	0.01	0.01	0.01
DC Access (pti)	617.67	647.34	593.26	644.32	644.19
L2 Access (pti)	219.23	245.05	198.91	213.89	243.45
L2 Access from IC Miss (pti)	1.67	1.5	1.81	1.38	1.52
L2 Access from DC Miss (pti)	119.28	133.53	108.07	116.33	131.41
L2 Access from HWPF (pti)	23.93	25	23.08	22.28	25.77
L2 Miss (pti)	36.63	40	34.04	36.27	40.36
L2 Miss from IC Miss (pti)	1.03	1.13	0.95	0.96	1.2
L2 Miss from DC Miss (pti)	21.51	24.51	19.21	22.21	24.23
L2 Miss from HWPF (pti)	14.09	14.36	13.88	13.1	14.94
L2 Hit (pti)	26.58	27.4	25.94	24.11	28.53
L2 Hit from IC Miss (pti)	0.78	0.29	1.15	0.19	0.19
L2 Hit from DC Miss (pti)	15.72	16.22	15.34	14.53	17.24
L2 Hit from HWPF (pti)	9.84	10.64	9.2	9.18	10.83
L3 METRICS					
Metric	System (Aggregated)	Package (Aggregated)-0	Package (Aggregated)-1	CCD (Aggregated)-0	CCD (Aggregated)-1
L3 Access	4762627165	2338595992	2424031173	292903705	291774043
L3 Miss	4667265146	2295013129	2372252017	287397063	286449222
L3 Miss %	98	98.14	97.86	98.12	98.18

Figure 5-1: Sample AMDuProfPcm analysis results

5.1.2 AMDuProfSys

AMDuProfSys is a python-based system analysis tool for AMD EPYC processors to collect `Ccore`, `L13`, `DFdf` and `umcUMC` metrics in Windows and Linux OS. On Linux, the tool uses the `perf` tool to collect the performance counters. It also uses its own driver mode to collect the profile data, with `--use-amd-driver` option,

In Linux, by default, this tool is based on the underlaying Linux `perf` utility provided by the Linux kernel, `msr` module, and basic hardware access primitives provided by the kernel. However, you can choose to use AMDuProfDriver to collect data using the `--use-amd-driver` option. The following example commands collects `core`, `l3`, and `df` metrics and report the profile data in .csv format.

```
./AMDuProfSys.py collect --config core,l3,df -a -d 5 -o /tmp/swp-perf
./AMDuProfSys.py report -f xls -i /tmp/swp-perf/swp-perf.ses
```

Here is a sample report:

	#core_config	system:0	core:0	core:1	core:2	core:3	core:4	core:5	core:6	core:7
2										
3	Avg IPC (w/ halt)	0.084966296	1.384479298	0.000795801	0.000799582	0.000793433	0.002540013	0.00078591	0.000761	0.00075228
4	Avg UPC (w/ halt)	0.083288302	1.345603494	0.001398812	0.00146172	0.001404301	0.005261642	0.001413577	0.001373	0.001334999
5	Avg CPI (w/ halt)	11.76937266	0.722293213	1256.594929	1250.65394	1260.345941	393.6986991	1272.41068	1314.063	1329.292421
6	Avg CPU (w/ halt)	12.00648805	0.743160972	714.8922229	684.1254165	712.098126	190.0547227	707.4254253	728.1228	749.0642274
7	Avg IPC (w/o halt)	1.359811329	1.56607353	0.061584906	0.070412029	0.066591815	0.073753488	0.091709197	0.091538	0.090480135
8	Avg UPC (w/o halt)	1.332956499	1.522098609	0.108250277	0.128720669	0.117861178	0.152780482	0.164952739	0.1652	0.160566416
9	Avg CPI (w/o halt)	0.735396138	0.638539622	16.23774491	14.20211884	15.01686055	13.55868083	10.90403181	10.92448	11.05214971
10	Avg CPU (w/o halt)	0.750212029	0.656987658	9.237851662	7.768760134	8.484558013	6.545338684	6.062342499	6.053259	6.227952443
11	User IPC	1.653192302	1.653192302	0	0	0	0	0	0	0
12	User UPC	1.637448251	1.637448251	0	0	0	0	0	0	0
13	System IPC	0.129833052	0.144732728	0.115645153	0.114207967	0.110436413	0.096327276	0.111616283	0.113557	0.113811032
14	System UPC	0.206609128	0.148707491	0.263256611	0.259307917	0.251190602	0.203224956	0.252226805	0.258534	0.259677001
15	User CPI	0.604890307	0.604890307	0	0	0	0	0	0	0
16	User CPU	0.610706323	0.610706323	0	0	0	0	0	0	0
17	System CPI	7.702198999	6.909287304	8.64714145	8.755956591	9.054984452	10.38127564	8.959266298	8.806135	8.786494453
18	System CPU	4.840057216	6.724610815	3.798575068	3.85641908	3.981040653	4.920655511	3.964685675	3.867956	3.850937877
19	Total clocks (M)	603572.5807	35504.26652	35504.28099	35504.28828	35504.29442	35504.2755	35504.28016	35504.28	35504.28686
20	Total inst (M)	70960.18616	70405.50107	24.455964	25.680628	24.611612	122.076596	24.553624	23.79324	23.024732
21	Total uops (M)	70954.91366	69755.75701	55.671976	58.30758	55.979776	257.54918	55.485472	54.16984	52.534392
22	Total microcode inst (M)	44.754956	1.504572	1.852424	1.936404	1.903116	17.082064	1.624272	1.534212	1.487472
23	% of all inst	0.063070517	0.002137009	7.574528651	7.540329621	7.732593867	13.99290655	6.615202709	6.448099	6.460322752
24	Total microcode uops (M)	775.908016	56.439576	34.618076	35.296604	35.02308	170.514816	32.838536	32.09868	31.72658
25	% of all uops	1.093522599	0.080910277	62.18222971	60.53518942	62.56380876	66.20670118	59.18402569	59.25563	60.392019
26	Ave Uop / Microcode-inst	17.33680659	37.51204728	18.6879872	18.22791318	18.40301905	9.982096777	20.21738724	20.92193	21.32919477

Figure 5-2: Sample AMDuProfSys report

5.1.3 Application Analysis

AMD µProf profiler uses a statistical sampling-based approach to collect profile data for application analysis using any of the following approaches:

- Timer Based Profiling (TBP) to identify the hotspots in the profiled applications.
- Event Based Profiling (EBP) sampling based on Core PMC events to identify microarchitecture-related performance issues in the profiled applications.
- Instruction based Sampling (IBS) for precise instruction-based sampling.

Numerous microarchitecture-specific events are available for monitoring. The tool groups related and interesting events to monitor into a Predefined Sampling Configuration (PSC). PSCs are a convenient way to select a useful set of sampling events for profile analysis. Execute the following command to get the list of supported PSCs that can be used with the `collect` command `--config` option by executing the following Linux command:

```
$ AMDuProfCLI info --list collect-configs
```

The following procedure describes an overall performance assessment of the STMV workload (http://www.ks.uiuc.edu/Research/namd/utilities/stmv.tar.gz*) run with the NAMD code (http://www.ks.uiuc.edu/Research/namd*). This type of analysis is specified using the `assess` PSC and represents an EBP profile type analysis. To assess the workload:

1. Build the single process version of the NAMD application with debug information using the `-g` compiler flag, because debug info is needed to correlate the samples to functions and source lines. This example uses the AOCC 3.0 compiler and AOCL 3.0 library.

2. Collect phase: Run the application on 16 cores pinned to the first NUMA node with a taskset and collect profile data by specifying the profile configuration via `–config assess`:

```
$ AMDuProfCLI collect --config assess -o /tmp/namd-assess taskset -c 0-15 <full path>/namd2
+p 16 <full path to stmv.namd>
```

   The `collect` command launches the application and generates a raw data file (`.caperf` on Linux). You an specify the name of the raw data file using the `-o` flag.

3. Translate phase: Invoke the `translate` command to generate the report from the raw profile file:

```
./AMDuProfCLI translate -i /tmp/AMDuProf-classic-EBP_Feb-25-2022_21-13-55/
```

4. Use the AMDuProf binary to launch the AMDuProf GUI, which provides a visual interface for profiling and analyzing the performance data.

5. Select **HOME>Import Session** to open the **Import Profile Session** window, and then enter the full path to the .db file in the **Profile Data File** field.

Note: If you run the profile on one system and then import the corresponding raw profile data to another system, then you may need to specify the path to binary and source files.

6. Click the **ANALYZE** button on the top menu to open the **Function HotSpots** window. Double-clicking any entry on the **Functions** table in the **Metrics** window will load the source tab for that function in the **SOURCES**.

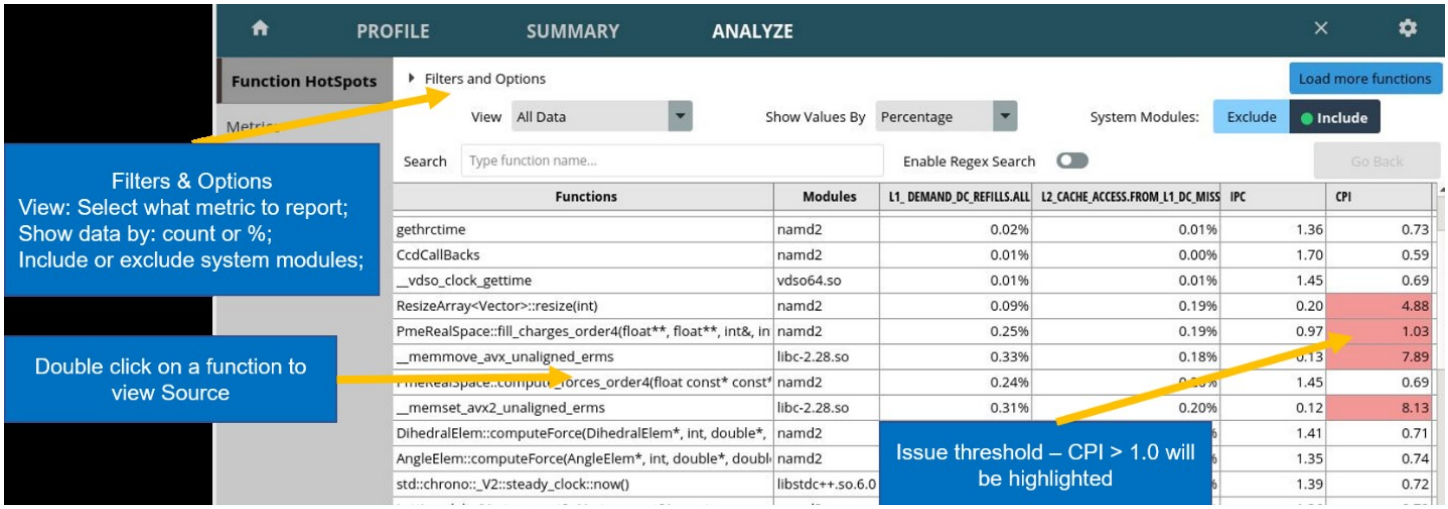

Figure 5-3: The Function HotSpots window

The following information is available for each source tab:

- The source lines of the selected function are listed, and the corresponding metrics are populated in various columns against each source line.
- The assembly instruction of the corresponding highlighted source line. The tree also shows the offset for each assembly instruction along with metrics.
- A heatmap overview of hotspots at the source level.

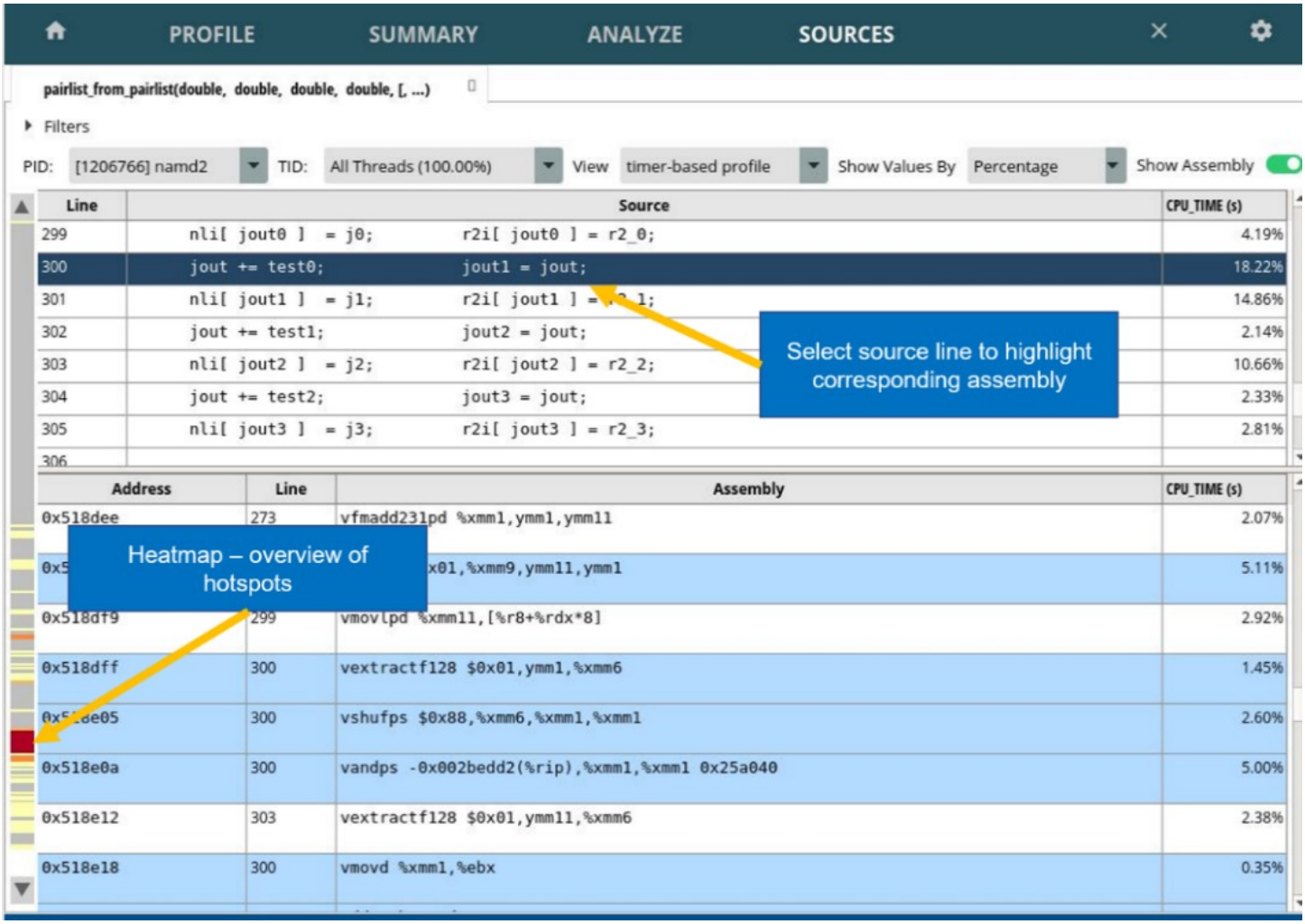

Figure 5-4: SOURCES source and assembly window

You can also collect `callstack` samples for C, C++, and Java applications with all the CPU profile types. These samples populate the **Flame Graph** window, which visually represents sampled `callstack` traces to help you quickly identify the hottest code execution paths. You can visualize the `callstack` by running an analysis with an additional `-g` flag:

```
$ AMDuProfCLI collect --config assess -g -o /tmp/namd-assess taskset -c 0-15 <full path>/
namd2 +p 16 <full path to stmv.namd>
```

Clicking **ANALYZE>Flame Graph** opens this window. The X axis shows the `callstack` profile, and the Y axis shows the stack depth. Each cell represents a stack frame. If a frame were present more often in the `callstack` samples, then the cell will be wider.

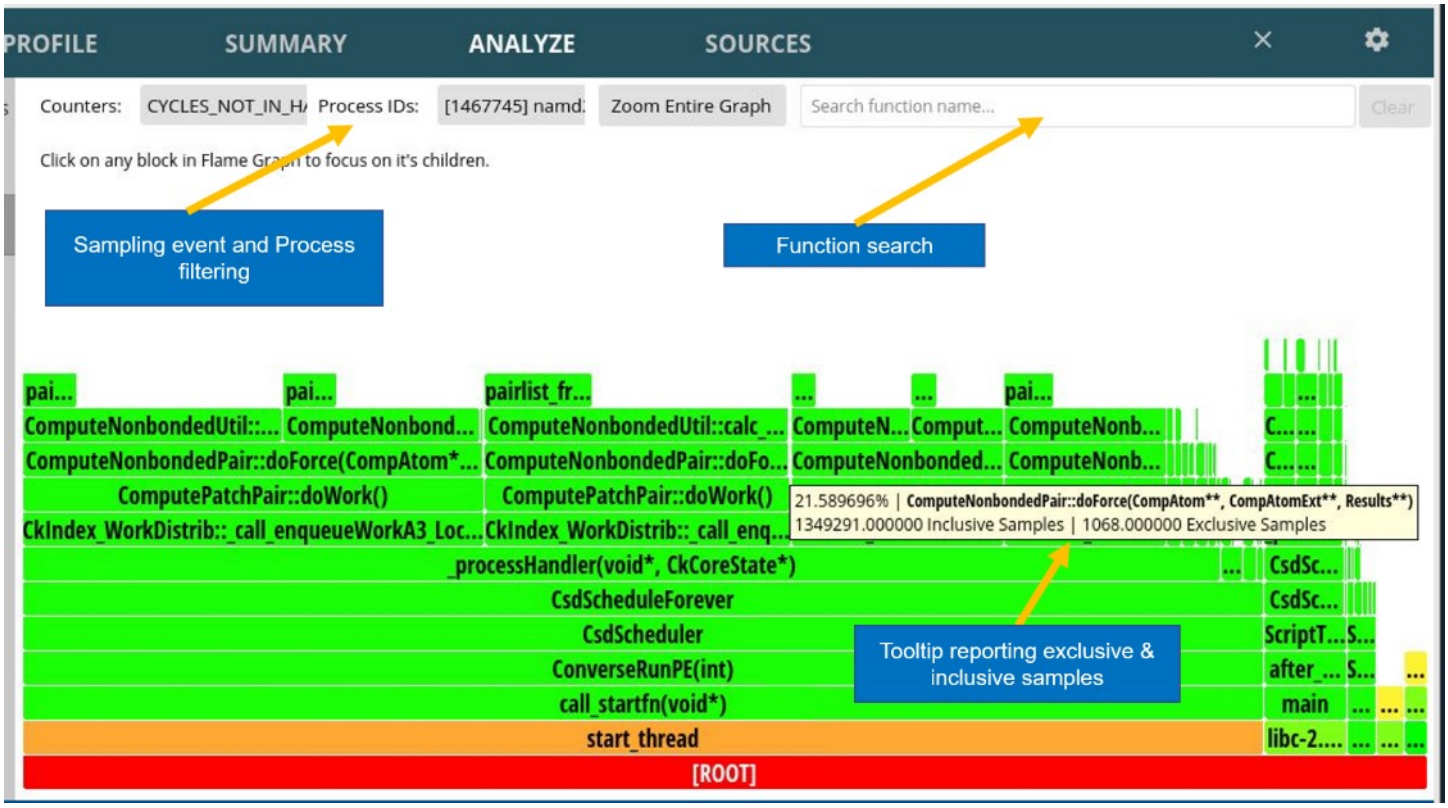

Figure 5-5: The Flame Graph window

5.1.4 HPC Analysis

To enable OpenMP profiling and analyze tracing data:

1. Select the profile target and profile type.
2. Click the **Advanced Options** button to turn on the **Enable OpenMP Tracing** option in the **Enable OpenMP Tracing** pane.
3. Wait for the profile to complete.
4. Navigate to the HPC page to analyze the OpenMP tracing data. Figure 5-5 shows the Regions Detailed Analysis page with the thread activity in a parallel region.

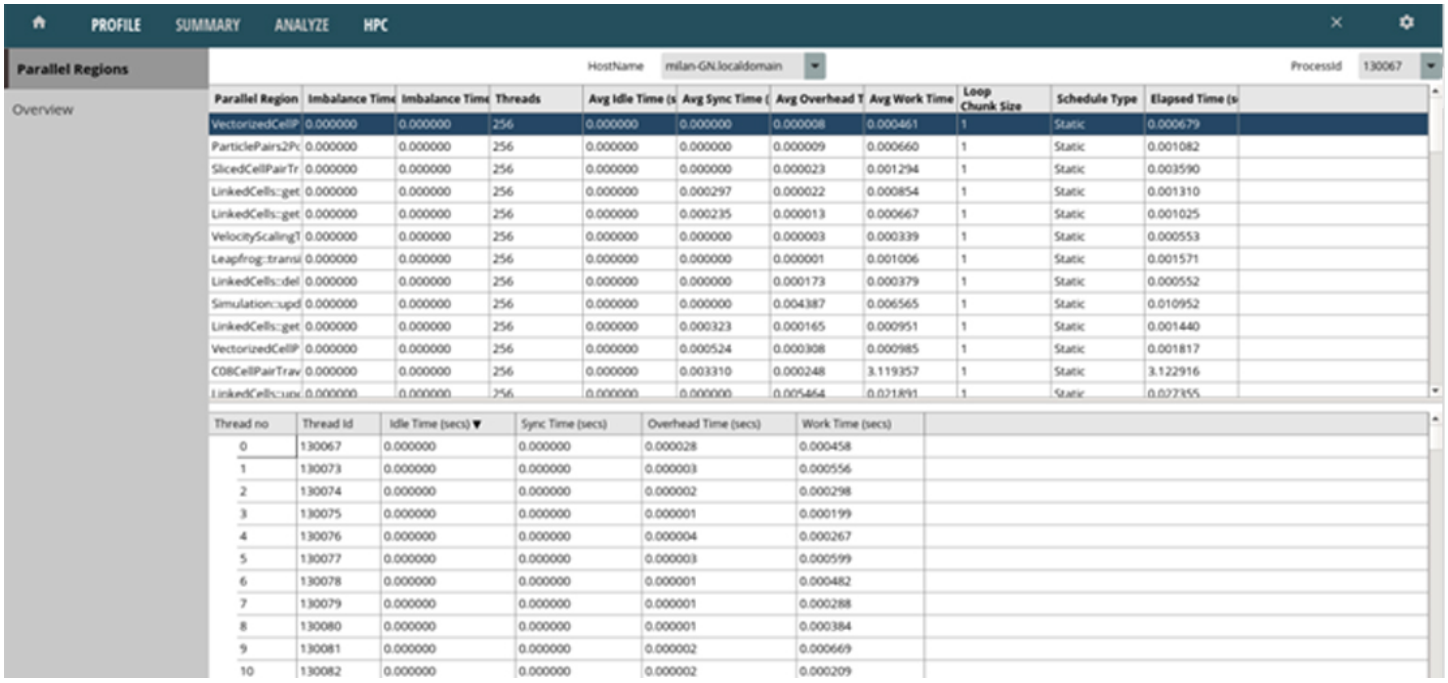

Figure 5-6: OpenMP Parallal Regions Detailed Analysis page with thread activity

5.1.5 MPI Analysis

AMD μProf includes the following MPI-specific profiling options:

- `--mpi` to specify MPI application profiling.
- `-o <output dir>` specifies the path where profiles will be saved. Each MPI process creates a corresponding raw profile file.

If you launch an MPI application on multiple nodes, then AMDuProfCLI will profile all MPI rank processes running on all nodes, and you can analyze data for processes run on one, many, or all nodes. Here is a sample MPI analysis that sends a trivial message around in a ring (https://github.com/open-mpi/ompi/blob/master/examples/ring_c.c*). To profile the MPI application on 4 ranks on a single node and analyze the data:

1. Collect the profile data:

```
$ mpirun -np 4 AMDuProfCLI collect --config tbp --mpi -o /tmp/uprof-mpi-ring-c ./ring-c
```

2. Process the profile data using the `translate` command to generate the profile DB. For example, to generate a report for all the MPI processes that ran on the host in which the MPI launcher was launched, use the new `--input-dir` option:

```
$ AMDuProfCLI translate -i /tmp/uprof-mpi-ring-c
```

3. Import the profile DB to the GUI. After importing, profile data for all profiled ranks will be available for analysis.

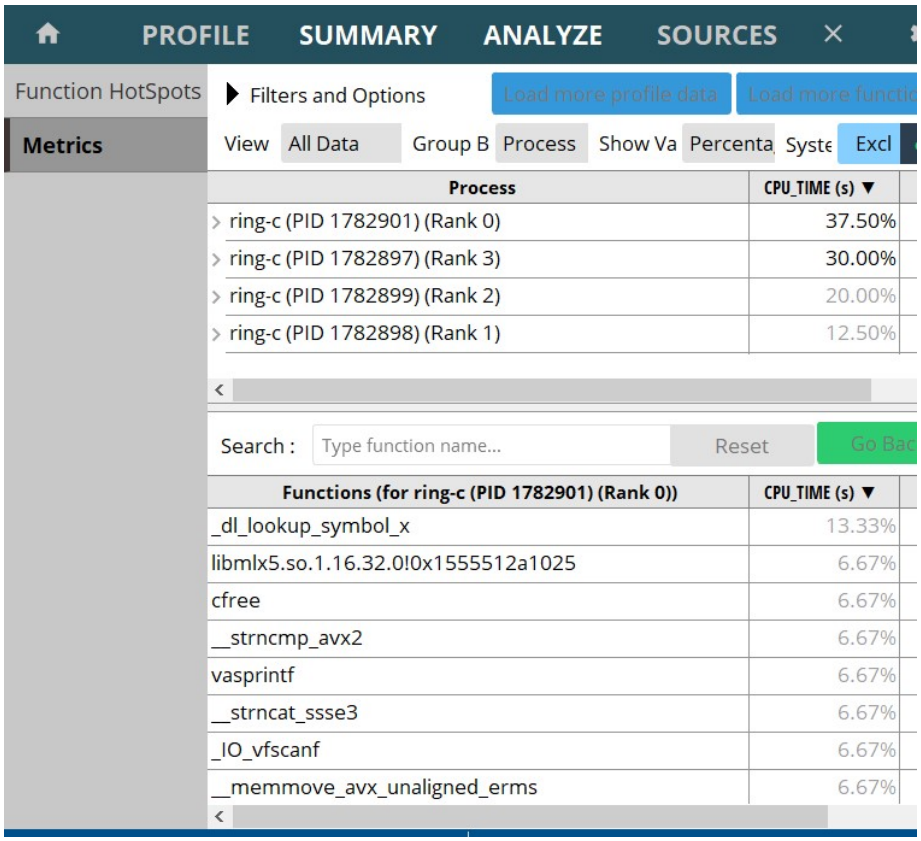

Figure 5-7: Imported MPI process data

5.1.6 MPI Trace Analysis

You can use MPI trace analysis to analyze, compute, and pass load imbalance messages among the ranks of a MPI application running on a cluster. It supports MPICH and derivative implementations. The supported thread models are:

- `FUNNLED`
- `SERIALIZED`

The profile reports are generated for Point-to-Point and Collective API activity summary. The support Matrix is:

- **MPI Spec versions:** MPI-3.0
- **Implementations:** OpenMPI, MPICH, and derivatives
- **Languages:** C, C++, and Fortran

The `AMDuProf` CLI supports the following 2 modes for MPI tracing:

- **LWT:** Light-weight tracing is useful for quick analysis of an application. The report gets generated in CSV format on-the-fly during collection stage.
- **FULL:** Full tracing is useful for in-depth analysis. This mode requires post-processing for report generation in CSV format. This mode traces more APIs than LWT tracing and is helpful for in-depth analysis of MPI application activity. The report file for the full tracing includes multiple tables to represent various details:
  - Communicator summary
  - Rank summary
  - P2P API summary
  - Communication matrix
  - Collective API summary

AMD uProf supports tracing Open MPI, MPICH, and the derivatives:

- `--trace mpi=mpich` for MPICH and derivatives (default option)
- `--trace mpi=openmpi` for Open MPI

Ensure that the correct option (`mpich` or `openmpi`) is passed depending on the MPI implementation used for compiling the MPI application.

The list of supported MPI APIs includes:

MPI_Pcontrol, MPI_Mrecv, MPI_Reduce, MPI_Iallreduce, MPI_Cancel, MPI_Imrecv, MPI_Allreduce, MPI_Ialltoall, MPI_Probe, MPI_Send, MPI_Alltoall, MPI_Ialltoallv, MPI_Iprobe, MPI_Bsend, MPI_Alltoallv, MPI_Ialltoallw, MPI_Mprobe, MPI_Ssend, MPI_Alltoallw, MPI_Ineighbor_Alltoall, MPI_Improbe, MPI_Rsend, MPI_Neighbor_Alltoal l, MPI_Ineighbor_Alltoallw, MPI_Start, MPI_Bsend_init, MPI_Neighbor_Alltoal lw, MPI_Ineighbor_Alltoallv, MPI_Startall, MPI_Ssend_init, MPI_Neighbor_Alltoal lv, MPI_Ibarrier, MPI_Test, MPI_Rsend_init, MPI_Bcast, MPI_Ibcast, MPI_Testall, MPI_Send_init, MPI_Scan, MPI_Comm_create, MPI_Testany, MPI_Ibsend, MPI_Reduce_Scatter, MPI_Comm_dup, MPI_Testsome, MPI_Issend, MPI_Ireduce_Scatter, MPI_Comm_dup_with_info, MPI_Wait MPI_Irsend, MPI_Iscan MPI_Comm_split, MPI_Waitall, MPI_Isend, MPI_Iscatter, MPI_Comm_split_type, MPI_Waitany, MPI_Scatter, MPI_Iscatterv, MPI_Intercomm_create, MPI_Waitsome, MPI_Scatterv, MPI_Igather, MPI_Intercomm_merge, MPI_Barrier, MPI_Gather, MPI_Igatherv, MPI_Cart_create, MPI_Recv, MPI_Gatherv, MPI_Iallgather, MPI_Cart_sub, MPI_Irecv, MPI_Allgather, MPI_Iallgatherv, MPI_Graph_create, MPI_Sendrecv, MPI_Allgatherv, MPI_INeighbor_Allgat her, MPI_Dist_graph_create, MPI_Sendrecv_repl ace, MPI_Neighbor_Allgat her, MPI_Ineighbor_Allgat herv, MPI_Dist_graph_create_adjacent, MPI_Recv_Init, MPI_Neighbor_Allgat herv, MPI_Ireduce

5.1.7 Roofline Analysis

AMDuProfPcm provides basic roofline modeling that relates the application performance to memory traffic and floating-point computational peaks. This is a visual performance model offering insights on improving the parallel software for floating point operations. This helps characterize an application and identify whether a benchmark is memory or compute bound. The HPC community commonly uses this to characterize both applications/kernels and modern architectures, such as CPUs and GPUs

The Roofline model combines arithmetic intensity, memory performance, and floating-point performance into a two-dimensional graph using bound and bottleneck analysis.

The tool monitors the memory traffic and floating-point operations when the profiled application is running. It also computes the Arithmetic Intensity, that is, the "operations per byte of DRAM traffic [FLOPS/BYTE]." The roofline graph is represented on a log-log scale, where the Y-axis is attainable floating-point performance, and the X-axis is arithmetical intensity. Horizontal lines show peak floating-point (FP) performance for a given architecture. On a log-log scale, the system's peak memory performance is the straight line with unit slope, where intercept is given by the logarithm of maximal value of the memory bandwidth. The peak memory performance and the peak FP performance intersect at the point of computational performance and peak memory bandwidth. Note that these limits are created once per system under consideration.

For a given application, the roofline tool calculates both arithmetic intensity and throughput and plots corresponding dot on the graph. Depending on the relative position of the dot with respect to diagonal and horizontal lines, one can judge whether the application is memory bound, i.e., below the peak memory performance line, or compute-bound, i.e., below the peak FP performance.

An inflection point where the diagonal and horizontal lines are intersect offers an insight into the overall system performance. The x-coordinate of the inflection point is the minimum arithmetic intensity required to achieve maximum performance. If the inflection point is far to the right, then only applications with very high arithmetic intensity can achieve the maximum performance of that system. On contrary If it is far to the left, then almost any workload can potentially hit the maximum performance.

To generate the roofline chart of an application:

1. Collect the profile data using AMDuProfPcm:
   ```
   $ AMDuProfPcm roofline -X -o /tmp/myapp-roofline.csv -- /tmp/myapp.exe
   ```
   On AMD "Zen4" 9xx4 Series processors, if the Linux kernel doesn't support accessing Data Fabric (DF) counters, then use the following command with root privilege:
   ```
   $ AMDuProfPcm roofline -o /tmp/myapp-roofline.csv -- /tmp/myapp.exe
   ```
2. Execute the following command to generate the roofline chart:
   ```
   $ AMDuProfModelling.py -i /tmp/myapp-roofline.csv -o /tmp/ --memspeed 4800 -a myapp
   ```

The roofline chart is saved in the file `/tmp/AMDuProf_roofline-2023-01-23-19h24m46s`.

The following plot hows a roofline for the MotorBike-42Mcells OpenFOAM workload run on a 2P AMD EPYC 9456 system. It shows peak FP values for both single (SP) double precision (DP).

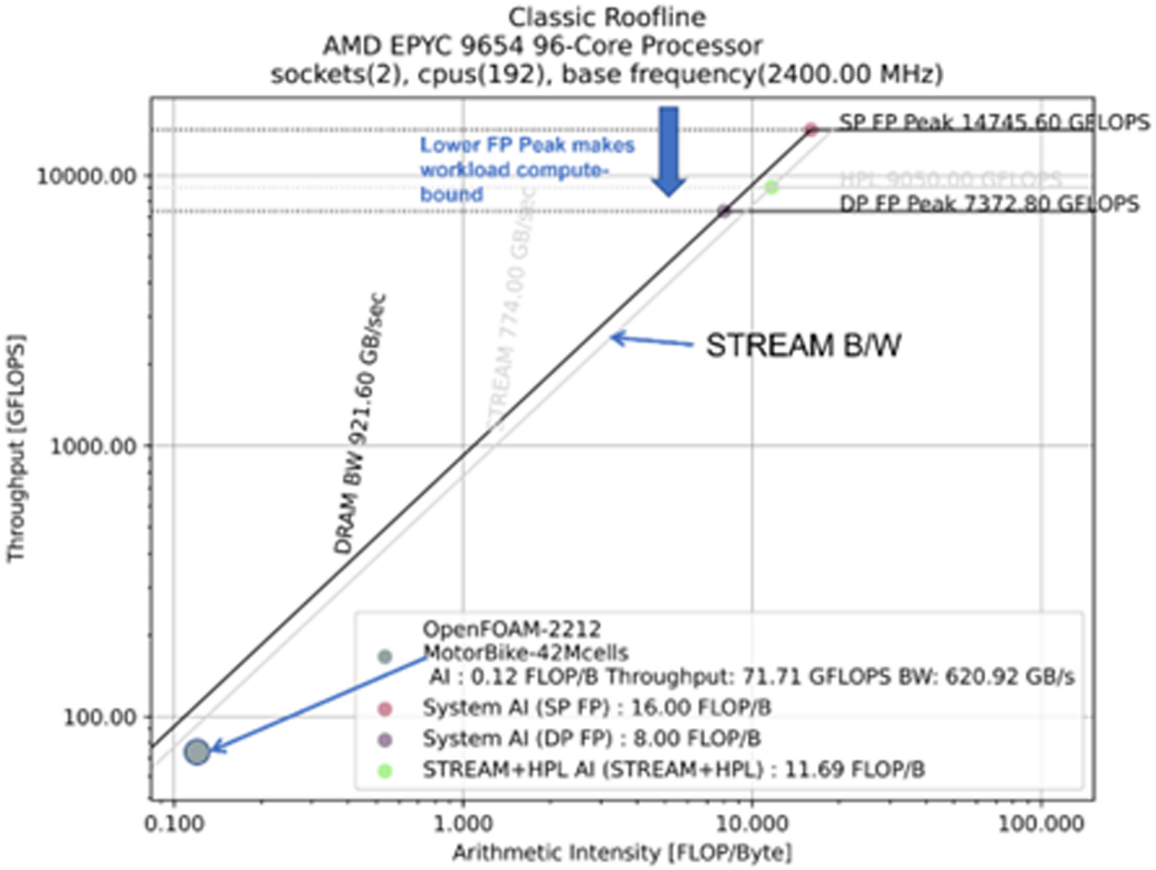

Figure 5-8: Sample roofline plot

The bandwidth and throughput values labeled "DRAM BW 921.60 GB/sec" are computed using the DIMM memory speed as reported by the `sudo dmidecode -type 17` or `lshw` commands. AMDuProf has an option to provide STREAM and DGEMM scores to the `AMDuProdModelling.py` script, which are used to plot the achievable roof. Running the STREAM and HPL on a node, one can input the scores to the roofline tool using the `–stream` and `-hpl` command line options. For a 2P AMD EPYC 9654 system, it gives 774 GB/sec for STREAM and 11.69 FLOP/B for STREAM+HPL values shown on the plot.

The gray dot on the plot shows the overall performance of OpenFOAM MotorBike-42Mcells workloads. The value is directly below the bandwidth roofline and shows that the OpenFOAM is memory-bound in this case.

5.1.8 Power Profiling

AMD uProf provides various counters for monitoring power and thermal characteristics. These counters are collected from various resources like RAPL, SMU, and MSRs. The following counter categories are supported for 3rd Gen AMD EPYC processors:

- **Power:** Average power for the sampling period, reported in watts. Viable for core and package.
- **Frequency:** Core Effective Frequency for the sampling period, reported in MHz.
- **P-State:** CPU Core P-State at the time when sampling was performed.
- **Temperature:** Average package temperature for the sampling period, reported in Celsius.

Use the AMDuProfCLI `timechart` command to collect system metrics and write them into either a text file or comma-separated-value (CSV) file.

5.1.8.1 Profiling Effective Frequency

To collect power profile counter values:

1. Get the list of supported counter categories by running the AMDuProfCLI `timechart` command with the `--list` option.
2. Collect and the report the required counters using the AMDuProfCLI `timechart` command by specifying the interesting counters using the `-e` or `--event` option.

For example, to collect frequency counter values from core 0 running a STREAM workload:

```
AMDuProfCLI timechart --event core=0,frequency -o ./eff_freq_core0  <STREAM binary>
```

The resulting `eff_freq_core0.csv` file has different sections from which you can obtain frequency counter timestamps, such as via the `sed` command.

5.1.8.2 Live System-Wide Power Profile

This profile type performs the power analysis and plots the metrics in a live timeline graph and/or saves them to a database. This profile is available through the GUI. To configure and start the profile:

1. In the **PROFILE** page, select the appropriate profile target by clicking the **Next** button.
2. In the **Select Profile Type** fragment, select **System-wide Power Profile (Live)**, and then enable the desired counters from the list.
3. Click the **Start Profile** button to start collecting profile data. This profile type reports profile data as line graphs in the **TIMECHART** page for further analysis.

Figure 5-9: Thermal characteristic - Timeline

Chapter 6 uProf Quick Reference Guide

6.1 CPU Profiling

(User Guide § 6.4, 6.5, 6.6)

Profile applications to analyze performance bottlenecks. In-depth analysis at all levels of application: process, thread, module, function, source line, instruction. Supports call-stack sampling, IMIX reporting. On FreeBSD, only EBP profiling supported.

Profile Types:

- **TBP:** Software timer-based sampling
- **EBP:** CPU event-based sampling
- **IBS:** CPU IBS MSR based sampling
- **Custom:** Choose events from TBP, EBP, IBS to create specialized custom type.
- **Profile Scopes:** Per-process, System-wide
- **OS:** Windows, Linux, FreeBSD
- **Languages:** C, C++, Fortran, Java, .NET, Assembly
- **CLI option syntax:**
```
collect --config <type>
collect -e <event-specifier>
report [--view-config <type>]
```
- **List of supported Predefined Configs** (See "Appendix D: AMDuProf Predefined Configs" on page 44):
```
$ AMDuProfCLI info --list collect-configs
```
- **List of supported Predefined Events:**
```
$ AMDuProfCLI info --list predefined-events
```
- **List of supported PMU (EBP) Events:**
```
$ AMDuProfCLI info --list pmu-events
```
- **List of supported View Configs:**
```
$ AMDuProfCLI info --list view-configs
```
- **Collect command example:**
```
$ AMDuProfCLI collect --config assess -o /tmp/cpuprof-assess ./classic-app
```

- **Report command example:**
  `$ AMDuProfCLI report -i /tmp/cpuprof-assess/AMDuProf-classic-TBP_Dec-09-2021_12-19-27`

6.2 MPI Profiling (Linux only)

(User Guide § 8.2)

Profile MPI applications using HW perf-counters to analyze the hotspots. This is same as performing CPU profiling on a cluster.

- **Supported MPI Spec:** MPI-3.0 or later
- **Languages:** C, C++, Fortran
- **Implementations:** MPICH and derivatives (Ex. Intel MPI), Open MPI

Example 1:

```
$ mpirun -np 4 ./AMDuProfCLI collect --config tbp --mpi -o /tmp/myapp-perf ./myapp.exe
```

```
$ AMDuProfCLI report -i /tmp/myapp-perf/AMDuProf-myapp_MPI --host all
```

Example 2:

```
$ mpirun -np 32 --host $HOSTNAME1:16,$HOSTNAME2:16 AMDuProfCLI collect --config assess --
mpi -o /tmp/multihost ./myapp.exe
```

```
$ AMDuProfCLI report -i /tmp/multihost/AMduProf_myapp-assess_MPI    --host all --detail
```

6.3 Power Profiling

(User Guide § 6.7)

Profile the system for power, thermal, and frequency characteristics. Report generated on-the-fly by the `collect` command.

- **OS:** Windows, Linux
- **CLI option syntax:**
  `timechart -e <counter-specifier>`
- **Supported Counter Categories:**
  `$ AMDuProfCLI timechart --list`
- **Collect command example:**
  `$ AMDuProfCLI timechart -e frequency -t 500  -d 10 -o /tmp/PowerOutput/`

6.4 MPI Tracing (Linux only)

(User Guide § 8.12)

Trace MPI applications to analyze compute and message passing load imbalance among the ranks on a cluster. Supports MPICH and derivative implementations. Supports FUNNLED and SERIALIZED thread models. Generate report for point-to-point and collective API activity summary.

- **Supported MPI Spec:** MPI-3.0 or later
- **Languages:** C, C++, Fortran
- Implementations: MPICH and derivatives (such as ParaStation MPI, Intel MPI), Open MPI.
- **Modes:** (Default mode is FULL)
  - **LWT:** Light-weight tracing for quick analysis, report generated on-the-fly during collection stage.
  - **FULL:** Complete tracing for in-depth analysis, requires report generation for analysis.
- **CLI option syntax:**

```
collect --trace mpi=mpich,lwt    #LWT of MPICH
collect --trace mpi=mpich,full   #FULL of MPICH
collect --trace mpi=openmpi,lwt  #LWT of OpenMPI
collect --trace mpi=openmpi,full #FULL of OpenMPI
collect --trace mpi              #FULL of MPICH
```

- **Collect command example:**

```
$ mpirun -np 4 ./AMDuProfCLI collect --trace mpi=mpich,full -o /home/myhome/ ./
my_mpi_app
```

- **Report command example:**

```
$ AMDuProfCLI report -i /home/myhome/AMDuProf-classic-HPC_Dec-09-2021_12-19-27
```

6.5 OpenMP Tracing (Linux only)

(User Guide § 8.1)

Trace OpenMP applications to analyze load imbalance among the threads in a parallel region.

- **Supported OpenMP Spec:** v5.0 or later
- **Compiler:** LLVM 8.0 or later, AOCC 2.1 or later, ICC 19.1 or later
- **Languages:** C, C++, Fortran
- **OS:** Ubuntu 18.04 or later, RHEL/CentOS 8.x
- **CLI option syntax:**

```
collect --trace openmp --config tbp
```

- **Collect command example:**

```
$ AMDuProfCLI collect --trace openmp --config tbp -o /home/myhome/ ~/classic-app
```

- **Report command example:**
  ```
  $ AMDuProfCLI report -i /home/myhome/AMDuProf-classic-TBP_Dec-09-2021_12-19-27
  ```

6.6 HPC Hybrid Tracing

Trace the hybrid programs (OpenMP + MPI) using uProf to analyze the load imbalance. Trace OpenMP as well as MPI at the same time.

- **Collect command example:**
  ```
  $ mpirun -np 4 ./AMDuProfCLI collect --mpi --trace mpi=mpich,full --trace openmp --config tbp -o /home/myhome/ ./my_hybrid_app
  ```

- **Report command example:**
  ```
  $ AMDuProfCLI report -i /home/myhome/AMDuProf-my-hybrid-app_MPI     --host all
  ```

6.7 OS Tracing (Linux only)

(User Guide § 8.6, 8.7, 8.8)

Monitors the operating system during the application runtime. Supported on Linux kernel 4.7 or later. Needs root access. Depends on BCC installation.

- **Prerequisites:** See "Appendix A: Installing BCC and eBPF" on page 43.

- **Supported Events:** schedule, `diskio`, `syscall`, `pagefault`, `memtrace`, `pthread`

- **CLI option syntax:**
  ```
  collect --trace os[=schedule,diskio,syscall,pagefault,memtrace,pthread]
  ```

- **Collect command example:**
  ```
  $ AMDuProfCLI collect --trace os -o /home/myhome/ ~/classic-app
  ```

- **Report command example:**
  ```
  $ AMDuProfCLI report -i /home/myhome/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27
  ```

6.8 AMDuProfPcm (System Analysis)

(User Guide § 3)

Monitor the basic performance metrics. It periodically collects the CPU Core, L3, and DF perf-counters and reports various metrics.

- **OS:** Windows, Linux, FreeBSD

- **Going root-less (on Linux):** Use the pre-release option `-X` to monitor the metrics without having a dependency on the `msr` module and root access.

- **Prerequisites (on Linux):** Uses the `msr` driver to collect the data. See "Appendix B: AMDuProf Linux Prerequisites" on page 43.

- **Prerequisites (on FreeBSD):** Uses the `cpuctl` module and requires either root privileges or read write permissions for `/dev/cpuctl*` devices.

- **CLI option syntax:**
  `AMDuProfPcm -m <METRIC> -o <FILE-PATH>`
- **Supported core PMC events:**
  `# AMDuProfPcm -l`
- **Help & list of supported metrics:**
  `# AMDuProfPcm -h`
- **Collect command example:**
  ```
  $ AMDuProfPcm -m ipc,l2,l3 -c core=0 -d 60 -o /tmp/pcmdata.csv
  $ AMDuProfPcm -m memory -c core=0 -o /tmp/pcmdata.csv -- ./myapp.exe
  ```

6.9 AMDuProfSys (System Analysis)

(User Guide § 4)

Python-based system analysis tool. Collects data from CPU Core, L3, DF, and UMC perf-counters. This tool is useful for collecting the hardware events and evaluate the simple counter values or complex recipe using collected raw events. On Linux, use the `--use-amd-driver` option to collect data using AMDuProf driver instead of Linux PERF.

- **OS:** Linux, Windows (only Core PMC)
- **Prerequisites (on Linux):** Uses the Linux `perf` subsystem, `msr` module, and basic hardware access primitives. See "Appendix C: AMDuProfSys Linux Prerequisites" on page 43.
- **Prerequisites (on Windows):** Uses the uProf driver to collect the Core PMC event counters. The AMDuProf supplied driver must be installed; the Windows installer takes care of the driver installation.
- **CLI option syntax:**
  ```
  AMDuProfSys.py collect --config <CONFIG>
  AMDuProfSys.py report -i <SESSION-DIR>
  ```
- **Help command:**
  `$ AMDuProfSys.py -h`
- **Collect command example:**
  `$ AMDuProfSys.py collect --config core -C 0 -o output taskset -c 0 <application>`
- **Report command example:**
  `$ AMDuProfSys.py report -i output_core.ses`

6.10 Pre-release Features of AMDuProfPcm on Linux

6.10.1 Roofline Model Analysis

(User Guide § 3.5.2)

Roofline model relates the application performance to memory bandwidth and floating-point computational peaks. This model offers insights on improving the parallel software for floating point operations. This helps to characterize whether an application is memory or compute bound.

To generate the roofline chart:

1. Collect the data for roofline modeling
   `$ AMDuProfPcm roofline -X -o /tmp/myapp-roofline.csv -- /tmp/myapp.exe`

2. Generate the roofline chart in PDF format
   `$ AMDuProfModelling.py -i /tmp/myapp-roofline.csv -o /tmp/ --memspeed 3200 -a myapp`

6.10.2 Pipeline Utilization (Top-Down Analysis)

(User Guide § 3.5.3)

Supported only on Zen4 CPUs. This feature provides top-down metrics to analyze the bottlenecks in the CPU pipeline. Use the option -m topdown to monitor and report the level-1 and level-2 top-down metrics.

Examples:

- Run the following command to collect the top-down metrics:
  `$ sudo AMDuProfPCm -m topdown -c core=0 -A system -o /tmp/myapp-td.csv -- /usr/bin/taskset -c 0 myapp.exe`

- Otherwise, use the pre-release option `-X` that does not require root access:
  `$ AMDuProfPCm -X -m topdown -A system -o /tmp/myapp-td.csv -- /usr/bin/taskset -c 0 myapp.exe`

6.11 Help Commands

Looking for more options? Use the following commands from the terminal to explore more.

- **All commands help:** `$ AMDuProfCLI -h`
- **Collect command help:** `$ AMDuProfCLI collect -h`
- **Report command help:** `$ AMDuProfCLI report -h`
- **Timechart command help:** `$ AMDuProfCLI timechart -h`
- **Info command help:** `$ AMDuProfCLI info -h`

6.12 Appendices

6.12.1 Appendix A: Installing BCC and eBPF

(User Guide § 1.3.2)

Follow the steps listed at https://github.com/iovisor/bcc/blob/master/INSTALL.md* to install the BCC.

After installing BCC, execute the following commands to validate the BCC installation:

```
$ cd AMDuProf_Linux_x64_4.0.x/bin/
$ sudo ./AMDuProfVerifyBpfInstallation.sh
```

If you install uProf using RPM/DEB installer, then the installer will run the script and provide the info about BCC installation and BPF support on the host.

6.12.2 Appendix B: AMDuProf Linux Prerequisites

(User Guide § 3.1.1.1)

Root permission is required for the following steps.

1. Using the msr driver requires root privileges or read write permissions for `/dev/cpu/*/msr` devices.
2. The NMI watchdog must be disabled.
   `$ sudo echo 0 > /proc/sys/kernel/nmi_watchdog`
3. Set `/proc/sys/kernel/perf_event_paranoid` to -1.
4. Load the msr driver.
   `$ modprobe msr`

6.12.3 Appendix C: AMDuProfSys Linux Prerequisites

(User Guide § 4.1)

Root permission the required for the following steps.

1. If you download a tar ball, then you will need to manually install the driver:
   `$ sudo ./AMDPowerProfilerDriver.sh install`
2. If `perf` is not installed already, then you can install it by executing the following command (on Ubuntu):
   `$ sudo apt-get install linux-tools-common linux-tools-generic linux-tools-`uname -r``
3. AMDuProfSys uses the msr driver for UMC counter access and requires either root privileges or read/write permissions for `/dev/cpu/*/msr` devices.
4. The tool requires Python 3.x. The following Python modules need to be installed:
   - **tqdm:** Execute `pip3 install tqdm`.
   - **xlsxwriter:** Execute `pip3 install XlsxWriter`.
   - **yaml:** Execute either `apt-get install python-yaml` or `pip3 install pyyaml`.

- **yamlordereddictloader:** Execute `pip3 install yamlordereddictloader`.

6.12.4 Appendix D: AMDuProf Predefined Configs

(User Guide § 2.2)

The Predefined Configuration provides a convenient way to select a useful set of sampling events for profile analysis. The following table lists all such configurations:

Config	Type	Description
tbp	TBP	(Time Based Profiling) Identify where the programs are consuming time.
assess	EBP	(Assess Performance) Provides an overall assessment of the performance.
assess_ext	EBP	(Assess Performance Extended) Provides an overall assessment of the performance with additional metrics.
data_access	EBP	(Investigate Data Access) Find data access operations with poor L1 data cache locality and poor DTLB behavior.
inst_access	EBP	(Investigate Instruction Access) Find instruction fetches with poor L1 instruction cache locality and poor ITLB behavior.
branch	EBP	(Investigate Branching) Find poorly predicted branches and near returns.
cpi	EBP	(Investigate CPI) Analyze the CPI and IPC metrics of the running application or the entire system.
ibs	IBS	`(Instruction Based Sampling) Collect the precise sample data using IBS Fetch and IBS Op.`
memory	IBS	(Cache Analysis) Identify the false cache-line sharing issues.
energy	Energy	(Energy Analysis) Identify where the program is consuming more energy.

Chapter 7 Resources

7.1 AMD Resources

- AMD EPYC Solution Briefs and White papers (includes 7001, 7002, 7003, and 9004 series documents)
- AMD EPYC Tuning Guides (includes 7001, 7002, 7003, and 9004 series documents)
- Reference for all developers on AMD platforms. Includes access to additional tuning guides, developer guides, manuals, ISA documents, and specs.
- AMD Processors documentation and guides
- AMD Zen Software Studio
- AMD AOCC: AMD Optimizing CPU Compiler
- AMD AOCL: AMD Optimizing CPU Libraries
- AMD μProf: Performance Analysis Tool
- Community Forum on AMD to discuss technical issues and contact a Server Guru
- Processor Programming Reference (PPR) for AMD Family 19h Models 00h-0Fh Processors, Revision B1 Processors

7.1.1 Other Resources

- Linux virtual memory*
- Linux kernel and CPU governors documentation*
- Spectre and Meltdown*
- AMD Statement on Spectre and Meltdown

This page intentionally left blank.

Chapter 8

Processor Identification

Figure 8-1 shows the processor naming convention for AMD EPYC 9004 Series Processors and how to use this convention to identify particular processors models:

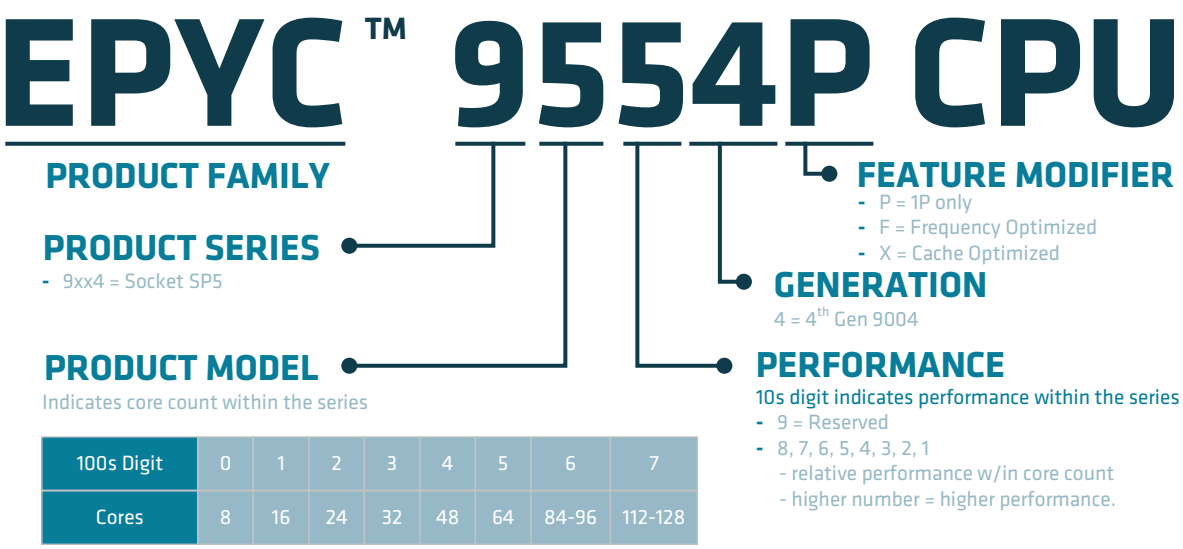

Figure 8-1: AMD EPYC SoC naming convention

8.1 CPUID Instruction

Software uses the `CPUID` instruction (`Fn0000_0001_EAX`) to identify the processor and will return the following values:

- **Family:** 19h identifies the "Zen 4" architecture
- **Model:** Varies with product. For example, EPYC Family 19h, Model 10h corresponds to an "A" part "Zen 4" CPU.
  - **91xx-96xx (including "X" OPNs):** Family 19h, Model 10-1F
  - **97xx:** Family 19h, Model A0-AF
- **Stepping:** May be used to further identify minor design changes

For example, `CPUID` values for Family, Model, and Stepping (decimal) of 25, 17, 1 correspond to a "B1" part "Zen 4" CPU.

8.2 New Software-Visible Features

AMD EPYC 9004 Series Processors introduce several new features that enhance performance, ISA updates, provide additional security features, and improve system reliability and availability. Some of the new features include:

- 5-level Paging
- AVX-512 instructions on a 256-byte datapath, including BFLOAT16 and VNNI support.
- Fast Short Rep STOSB and Rep CMPSB

Not all operating systems or hypervisors support all features. Please refer to your OS or hypervisor documentation for specific releases to identify support for these features.

Please also see the latest version of the *AMD64 Architecture Programmer's Manuals* or *Processor Programming Reference (PPR) for AMD Family 19h*.

8.2.1 AVX-512

AVX-512 is a set of individual instructions supporting 512-bit register-width data (i.e., single instruction, multiple data [SIMD]) operations. AMD EPYC 9004 Series Processors implement AVX 512 by "double-pumping" 256-bit-wide registers. AMD's AVX-512 design uses the same 256-bit data path that exists throughout the Zen4 core and enables the two parts to execute on sequential clock cycles. This means that running AVX-512 instructions on AMD EPYC 9004 Series will cause neither drops on effective frequencies nor increased power consumption. On the contrary, many workloads run more energy-efficiently on AVX-512 than on AVX-256P.

Other AVX-512 support includes:

- Vectorized Neural Network Instruction (VNNI) instructions that are used in deep learning models and accelerate neural network inferences by providing hardware support for convolution operations.
- Brain Floating Point 16-bit (BFLOAT16) numeric format. This format is used in Machine Learning applications that require high performance but must also conserve memory and bandwidth. BFLOAT16 support doubles the number of SIMD operands over 32-bit single precision FP, allowing twice the amount of data to be processed using the same memory bandwidth. BFLOAT16 values mantissa dynamic range at the expense of one radix point.