AMD EPYC

TUNING GUIDE
AMD EPYC 9004

High Performance Computing (HPC)

Publication	58002
Revision	1.3
Issue Date	June, 2023

© 2023 Advanced Micro Devices, Inc. All rights reserved.

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

Trademarks

AMD, the AMD Arrow logo, AMD EPYC, 3D V-Cache, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names and links to external sites used in this publication are for identification purposes only and may be trademarks of their respective companies.

* Links to third party sites are provided for convenience and unless explicitly stated, AMD is not responsible for the contents of such linked sites and no endorsement is implied.

Date	Version	Changes
July, 2022	0.1	Initial NDA partner release
Sep, 2022	0.2	Updated BIOS information
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, & other global updates
Jun, 2023	1.3	Second public release

Audience

This document helps you tune systems powered by 4th Gen AMD EPYC™ processors for High Performance Computing (HPC) workloads but is not an all-inclusive guide. Further, some items referred to herein may have different names across various OEM systems, such as OEM-specific BIOS settings. Every HPC workload also has unique performance characteristics. This document suggests items to focus on when performing additional, application-specific tuning as a starting point for performing your own performance testing and additional tuning for your specific workload. Performing effective HPC tuning requires familiarity with server configuration. You should also:

- Have admin access to the server's management interface (BMC).
- Be familiar with the server's management interface.
- Have admin OS access and be familiar with the OS-specific configuration, monitoring, and troubleshooting tools.

Authors

Kevin Mayo, Sylvester Rajasekaran, Igor Pasichnyk, and Anre Kashyap.

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

Chapter 1 AMD EPYC™ 9004 Series Processors - 1

1.1 General Specifications ... 1
1.2 Model-Specific Features ... 1
1.3 Operating Systems ... 2
1.4 Processor Layout ... 2
1.5 "Zen 4" Core ... 2
1.6 Core Complex (CCX) ... 3
1.7 Core Complex Dies (CCDs) ... 3
1.8 AMD 3D V-Cache™ Technology ... 4
1.9 I/O Die (Infinity Fabric™) ... 5
1.10 Memory and I/O ... 6
1.11 Visualizing AMD EPYC 9004 Series Processors (Family 19h) ... 7
1.11.1 Models 91xx-96xx ("Genoa") ... 7
1.11.2 Models 97xx ("Bergamo") ... 8
1.12 NUMA Topology ... 8
1.12.1 NUMA Settings ... 8
1.13 Dual-Socket Configurations ... 10

Chapter 2 Additional Processor Information - 11

2.1 Understanding hwloc-ls and hwloc-info ... 11
2.2 Core C-States ... 13
2.3 Core P-States, Frequencies, and Boosting ... 14
2.4 CPU Governors ... 15
2.5 Useful 'cpupower' Command Examples ... 16

Chapter 3 Quick HPC Setup - 17

3.1 BIOS Settings ... 17
3.2 OS Settings ... 18
3.3 Basic System Checks ... 18
3.4 Useful Commands ... 19
3.4.1 Build CPU ID lists with 'seq' ... 19
3.4.2 Core Pinning and Memory Locality: ... 19
3.4.3 Faster 'make' with 'make -j' ... 20

Chapter 4 BIOS Settings - 21

4.1 Recommended BIOS Settings for Bare Metal Workloads ... 21
4.1.1 Determinism (Performance | Power) ... 21
4.1.2 TDP (configurable Thermal Design Point) ... 21
4.1.3 PPL (Package Power Limit) ... 21
4.1.4 Simultaneous Multi-Threading (SMT) Enabled | Disabled ... 21
4.1.5 X2APIC = Auto ... 22

4.1.6 Core Performance Boost = OFF | ON 22
4.1.7 Memory speed = AUTO 22
4.1.8 Core C-States = Enabled 22
4.1.9 Data Fabric C-States (DF C-States) Enabled | Disabled 22
4.1.10 NPS = 1 | 2 |4 22
4.1.11 CCD Control 22
4.1.12 Down-Coring / Core Control 23
4.1.13 Transparent Secure Memory Encryption (TSME) = OFF 23

Chapter 5 Operating Systems ---- 25

5.1 Linux Kernel and ISV OS Support Considerations 25
5.2 /proc and /sys 25
5.2.1 /proc/sys/vm/zone_reclaim_mode 25
5.2.2 /proc/sys/vm/drop_caches 26
5.3 Transparent Huge Pages (THP) 26
5.4 Explicit Hugepages 26
5.5 Randomize_va_space 27
5.6 NUMA Balancing 27
5.7 Spectre and Meltdown 27

Chapter 6 Linux Perf ---- 29

Chapter 7 Host System Management Port ---- 31

7.1 HSMP Driver 31

Chapter 8 Executing Applications on AMD EPYC 9004 Processors ---- 33

8.1 Characterization Strategy 33
8.2 Pinning Strategies and Hybrid Codes 34
8.3 Microbenchmark Results 35
8.3.1 Stream 35
8.4 HPL 38
8.5 DGEMM 40
8.6 HPCG 41
8.7 Mellanox Configuration 41
8.7.1 Bandwidth Test 42
8.7.2 Latency Test 43
8.8 OSU Network Test 44

Chapter 9 Additional Information ---- 47

9.1 HPL and High SIMD Frequency 47
9.2 Reference System 48
9.3 AMD Resources 49
9.4 EDA Configuration 49
9.5 Other Resources 50

**Chapter 10 Processor Identification - - -

This page intentionally left blank.

Chapter 1

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.

1.1 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 1-1: Common features of all AMD EPYC 9004 Series Processors

1.2 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)	1GB (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 1-2: AMD EPYC 9004 Series Processors features by model

1.3 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.

1.4 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.

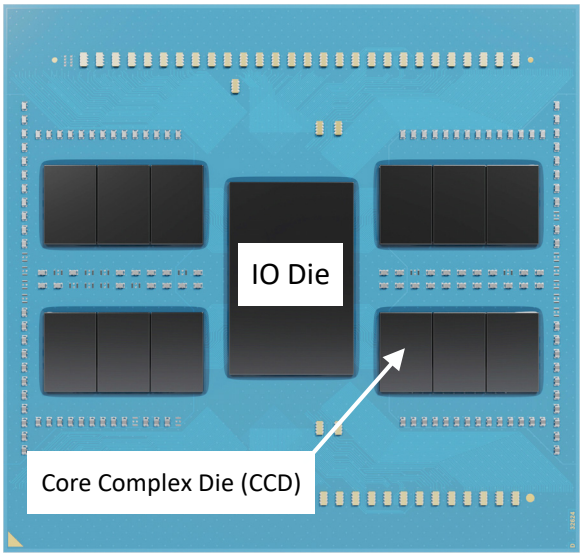

Figure 1-1: AMD EPYC 9004 configuration with 12 Core Complex Dies (CCD) surrounding a central I/O Die (IOD)

1.5 "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 1MB private unified (Instruction/Data) L2 cache. All caches use a 64B cache line size.

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

1.6 Core Complex (CCX)

Figure 1-2 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.

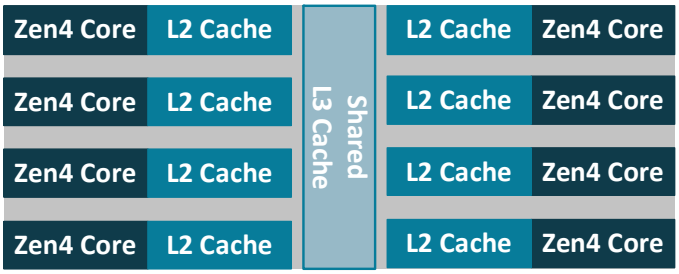

Figure 1-2: Top view of 8 compute cores sharing an L3 cache (91xx-96xx models)

1.7 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 1-5.

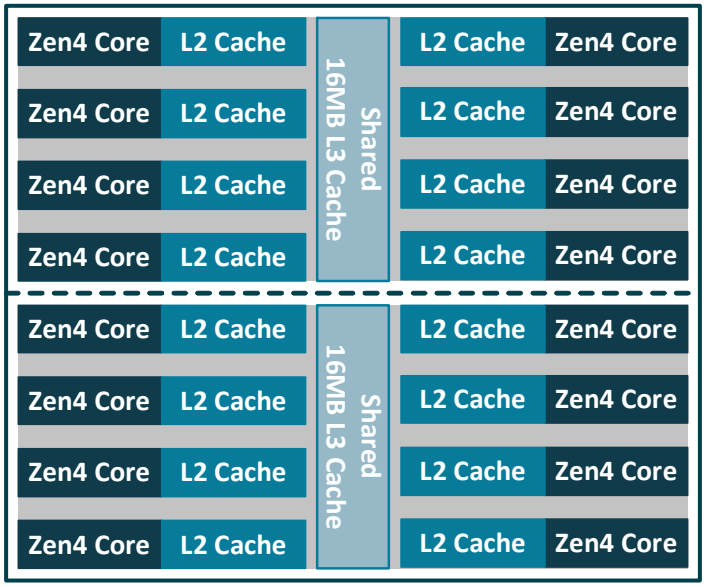

Figure 1-3: 2 CCXs in a single 4th Gen AMD EPYC 97xx CCD

Each of the Core Complex Dies (CCDs) in a 97xx model AMD EPYC 9004 Series Processor contains two CCXs (Figure 1-5):

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

Table 1-3: 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.

1.8 AMD 3D V-Cache™ Technology

AMD EPYC 9xx4X Series Processors include AMD 3D V-Cache™ die stacking technology that enables 97xxr, 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.

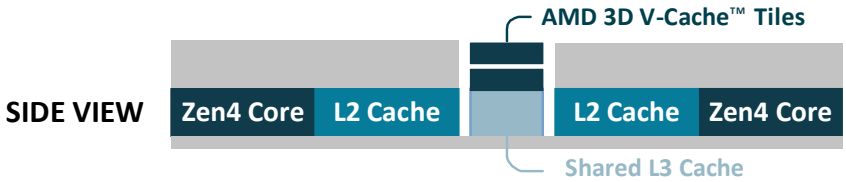

Figure 1-4: Side view of vertically-stacked central L3 SRAM tiles

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

Table 1-4: 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.

1.9 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 1-6 (which corresponds to Figure 1-2, 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.

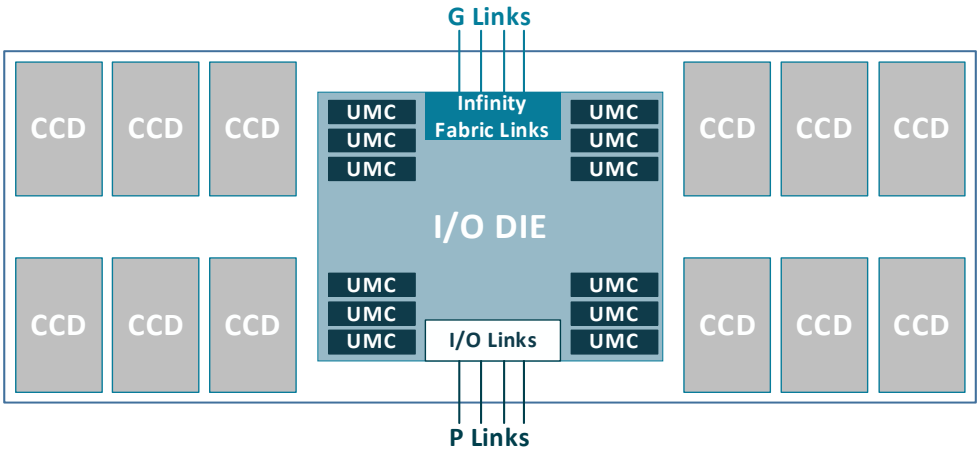

Figure 1-5: AMD EPYC 9654 processor internals interconnect via AMD Infinity Fabric (12 CCD processor shown)

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.

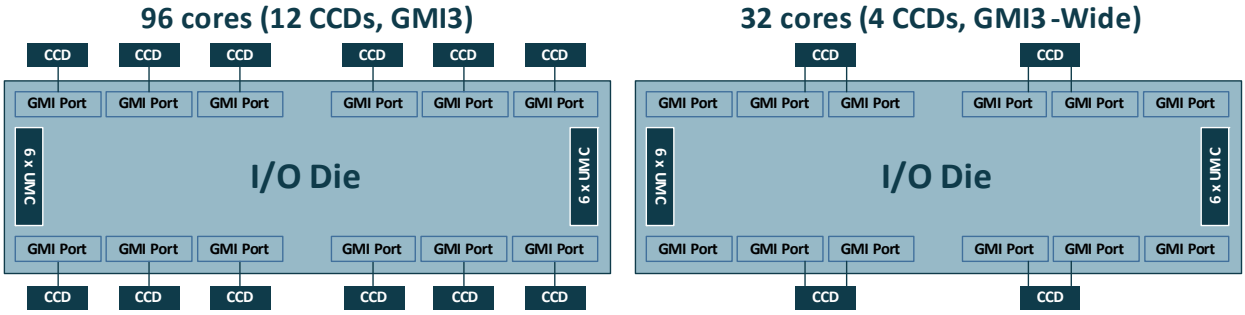

Figure 1-6: Standard vs. Wide GMI links

The IOD provides twelve unified memory controllers 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+.

1.10 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. 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.

1.11 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 8 for more information about nodes.

1.11.1 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.

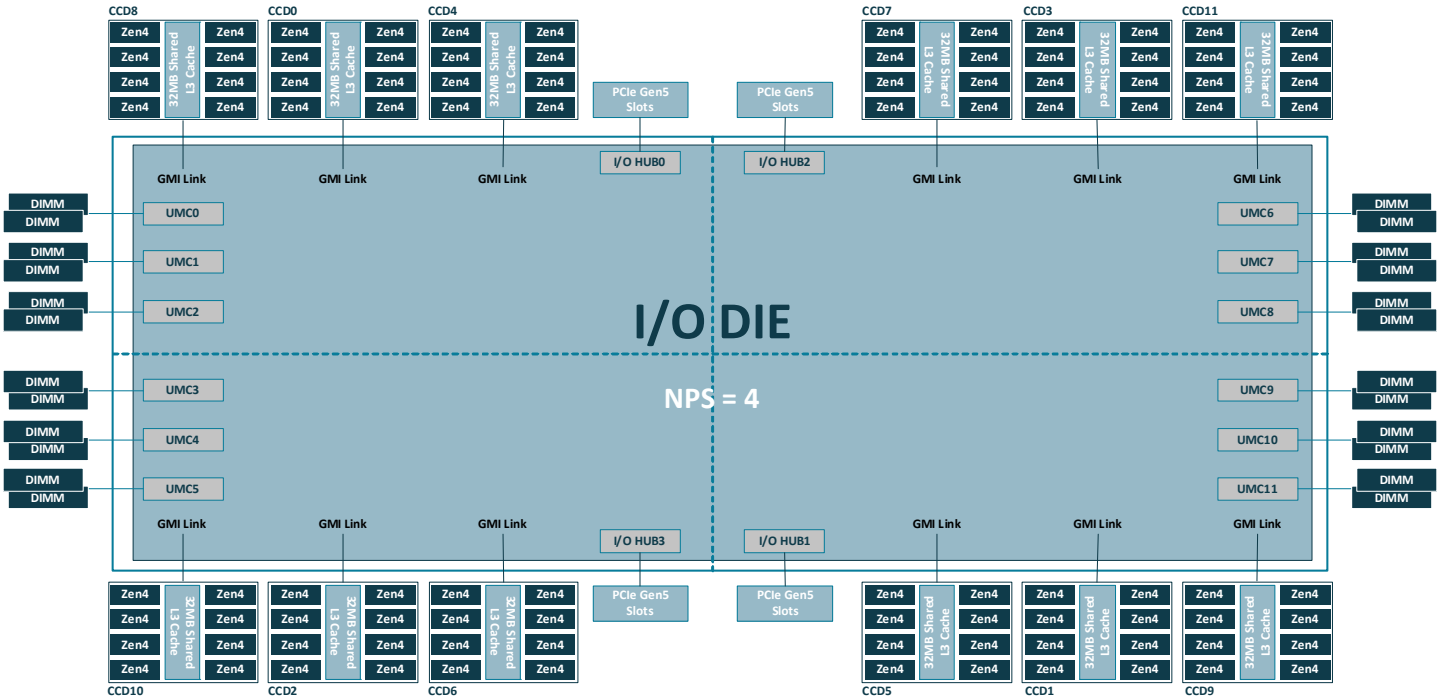

Figure 1-7: The AMD EPYC 9004 SoC consists of up to 12 CCDs and a central IOD for 91xx-96xx models, including "X" OPNs

1.11.2 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.

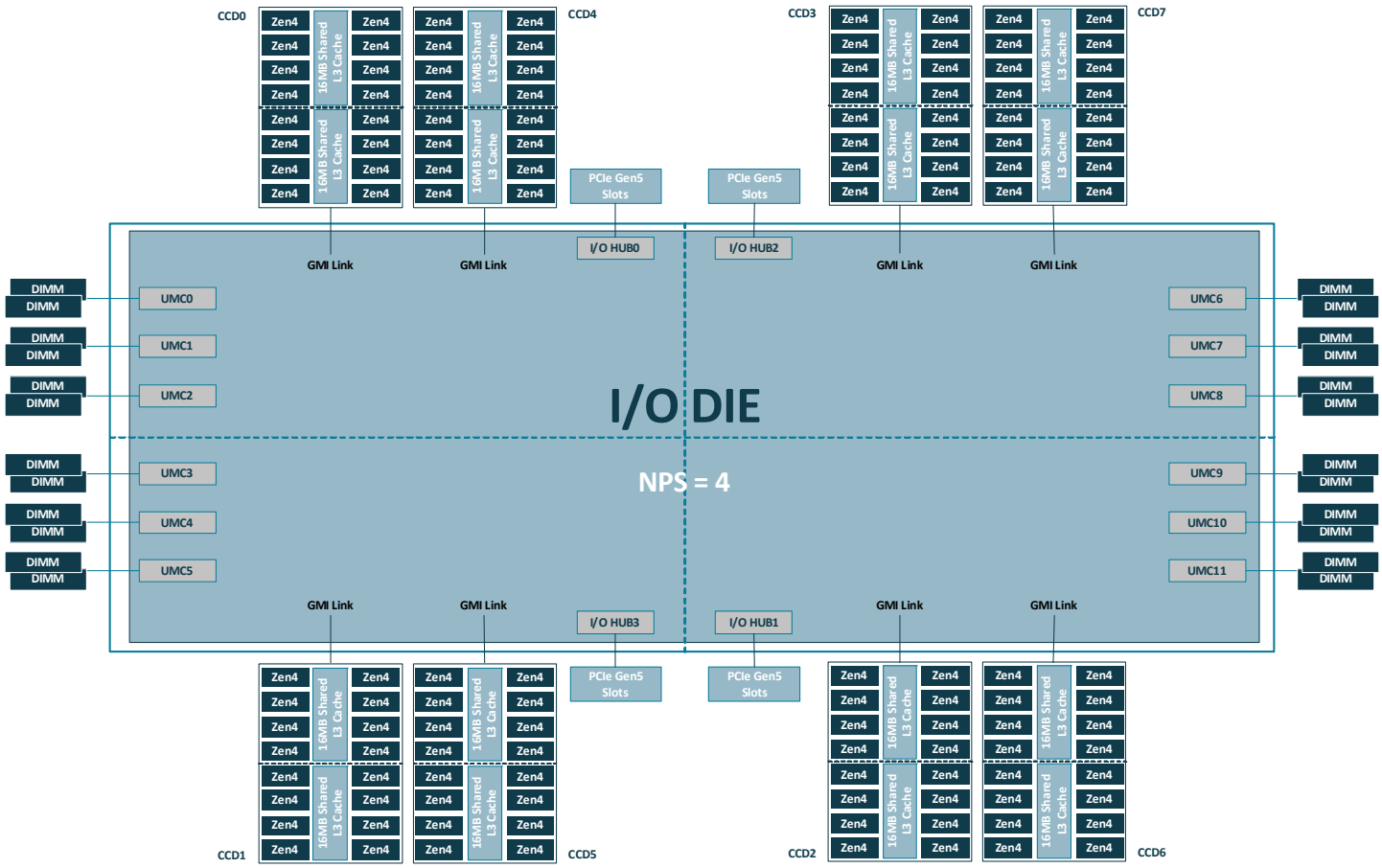

Figure 1-8: The AMD EPYC 9004 System on Chip (SoC) consists of up to 8 CCDs and a central IOD for 97xx models

1.12 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.

1.12.1 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 6 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.

1.13 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.

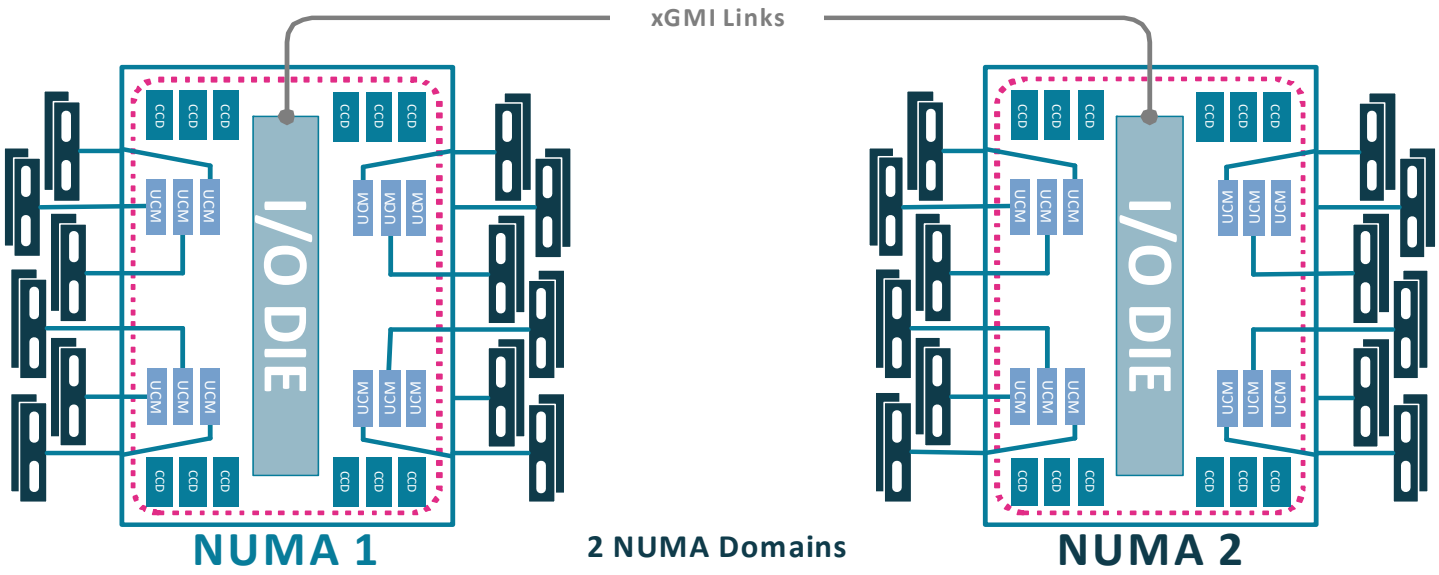

Figure 1-9: Two EPYC 9004 Processors connect through 4 xGMI links (NPS1)

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 to 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 2

Additional Processor Information

2.1 Understanding hwloc-ls and hwloc-info

This section explains how the OS sees NUMA nodes, caches, and cores. `hwloc-ls` is a very useful tool for obtaining CPU IDs when you need to be aware of what cores to pin to your job to. You can also use `numactl -H` to obtain a list of cores per NUMA domain with the respective logical weights between them. The following example shows the output of `hwloc-ls` on a dual-socket system with 96 cores per socket configured with NPS=4 and SMT disabled.

```
Group0 L#1
    NUMANode L#1 (P#1 94GB)
    L3 L#3 (32MB)
    L2 L#24 (1024KB) + L1d L#24 (32KB) + L1i L#24 (32KB) + Core L#24 + PU L#24 (P#24)
    L2 L#25 (1024KB) + L1d L#25 (32KB) + L1i L#25 (32KB) + Core L#25 + PU L#25 (P#25)
    L2 L#26 (1024KB) + L1d L#26 (32KB) + L1i L#26 (32KB) + Core L#26 + PU L#26 (P#26)
    L2 L#27 (1024KB) + L1d L#27 (32KB) + L1i L#27 (32KB) + Core L#27 + PU L#27 (P#27)
    L2 L#28 (1024KB) + L1d L#28 (32KB) + L1i L#28 (32KB) + Core L#28 + PU L#28 (P#28)
    L2 L#29 (1024KB) + L1d L#29 (32KB) + L1i L#29 (32KB) + Core L#29 + PU L#29 (P#29)
    L2 L#30 (1024KB) + L1d L#30 (32KB) + L1i L#30 (32KB) + Core L#30 + PU L#30 (P#30)
    L2 L#31 (1024KB) + L1d L#31 (32KB) + L1i L#31 (32KB) + Core L#31 + PU L#31 (P#31)
    L3 L#4 (32MB)
    L2 L#32 (1024KB) + L1d L#32 (32KB) + L1i L#32 (32KB) + Core L#32 + PU L#32 (P#32)
    L2 L#33 (1024KB) + L1d L#33 (32KB) + L1i L#33 (32KB) + Core L#33 + PU L#33 (P#33)
    L2 L#34 (1024KB) + L1d L#34 (32KB) + L1i L#34 (32KB) + Core L#34 + PU L#34 (P#34)
    L2 L#35 (1024KB) + L1d L#35 (32KB) + L1i L#35 (32KB) + Core L#35 + PU L#35 (P#35)
    L2 L#36 (1024KB) + L1d L#36 (32KB) + L1i L#36 (32KB) + Core L#36 + PU L#36 (P#36)
    L2 L#37 (1024KB) + L1d L#37 (32KB) + L1i L#37 (32KB) + Core L#37 + PU L#37 (P#37)
    L2 L#38 (1024KB) + L1d L#38 (32KB) + L1i L#38 (32KB) + Core L#38 + PU L#38 (P#38)
    L2 L#39 (1024KB) + L1d L#39 (32KB) + L1i L#39 (32KB) + Core L#39 + PU L#39 (P#39)
    L3 L#5 (32MB)
    L2 L#40 (1024KB) + L1d L#40 (32KB) + L1i L#40 (32KB) + Core L#40 + PU L#40 (P#40)
    L2 L#41 (1024KB) + L1d L#41 (32KB) + L1i L#41 (32KB) + Core L#41 + PU L#41 (P#41)
    L2 L#42 (1024KB) + L1d L#42 (32KB) + L1i L#42 (32KB) + Core L#42 + PU L#42 (P#42)
    L2 L#43 (1024KB) + L1d L#43 (32KB) + L1i L#43 (32KB) + Core L#43 + PU L#43 (P#43)
    L2 L#44 (1024KB) + L1d L#44 (32KB) + L1i L#44 (32KB) + Core L#44 + PU L#44 (P#44)
    L2 L#45 (1024KB) + L1d L#45 (32KB) + L1i L#45 (32KB) + Core L#45 + PU L#45 (P#45)
    L2 L#46 (1024KB) + L1d L#46 (32KB) + L1i L#46 (32KB) + Core L#46 + PU L#46 (P#46)
    L2 L#47 (1024KB) + L1d L#47 (32KB) + L1i L#47 (32KB) + Core L#47 + PU L#47 (P#47)
    HostBridge
    PCIBridge
        PCI 61:00.0 (InfiniBand)
        Net "ib0"
        OpenFabrics "mlx5_0"
        PCI 61:00.1 (InfiniBand)
        Net "ib1"
        OpenFabrics "mlx5_1"
```

This example illustrates several important points:

- 8 cores per L3 cache are shown grouped together along with the CPU IDs associated with each 32 MB L3 Cache (0,1,2,3). These 8-core groupings per L3 represents a single CCX/CCD.

- The 3 CCDs (3x L3 caches) appear logically grouped under each NUMA node.
- We can also see which NUMA domain the high-performance Mellanox network is connected to. The CPU IDs in that domain that are logically closest to the network interface.

The following example shows `hwloc-ls` output when SMT is enabled within the BIOS, thereby enabling a second thread on each core:

```
Machine (756GB total)
  Package L#0
    Group0 L#0
      NUMANode L#0 (P#0 94GB)
      L3 L#0 (32MB)
        L2 L#0 (1024KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
          PU L#0 (P#0)
          PU L#1 (P#192)
        L2 L#1 (1024KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1
          PU L#2 (P#1)
          PU L#3 (P#193)
        L2 L#2 (1024KB) + L1d L#2 (32KB) + L1i L#2 (32KB) + Core L#2
          PU L#4 (P#2)
          PU L#5 (P#194)
        L2 L#3 (1024KB) + L1d L#3 (32KB) + L1i L#3 (32KB) + Core L#3
          PU L#6 (P#3)
          PU L#7 (P#195)
        L2 L#4 (1024KB) + L1d L#4 (32KB) + L1i L#4 (32KB) + Core L#4
          PU L#8 (P#4)
          PU L#9 (P#196)
        L2 L#5 (1024KB) + L1d L#5 (32KB) + L1i L#5 (32KB) + Core L#5
          PU L#10 (P#5)
          PU L#11 (P#197)
        L2 L#6 (1024KB) + L1d L#6 (32KB) + L1i L#6 (32KB) + Core L#6
          PU L#12 (P#6)
          PU L#13 (P#198)
        L2 L#7 (1024KB) + L1d L#7 (32KB) + L1i L#7 (32KB) + Core L#7
          PU L#14 (P#7)
          PU L#15 (P#199)
```

On a dual-socket system with 2x 96 core CPUs with SMT enabled, the first thread on each core is within the range 0-191, while the second thread will have CPU IDs within the range 192-383. The above example shows how Core 0 has two threads associated with it:

- One with CPU ID 0 attached to it.
- One with CPU ID 192 attached to it.

Older Linux kernels may misalign or completely ignore L3. If you are installing older Linux kernels, then you will need to use either `hwloc-info` or `hwloc-ls` to ensure that the OS recognizes the correct cache hierarchy in the system.

```
depth 0:            1 Machine (type #0)
depth 1:           2 Package (type #1)
  depth 2:          8 Group0 (type #12)
   depth 3:         24 L3Cache (type #6)
    depth 4:        192 L2Cache (type #5)
     depth 5:       192 L1dCache (type #4)
      depth 6:      192 L1iCache (type #9)
       depth 7:     192 Core (type #2)
        depth 8:    384 PU (type #3)
Special depth -3:   8 NUMANode (type #13)
Special depth -4:   13 Bridge (type #14)
```

```
Special depth -5:   13 PCIDev (type #15)
Special depth -6:   6 OSDev (type #16)
Special depth -7:   28 Misc (type #17)
```

2.2 Core C-States

The CPU can idle in several Core-States or C-States:

- **C0:** Active. This is the active state while running an application.
- **C1:** Idle. AMD implementation for C1 states is based on MWAIT instruction that ensures fast, low latency transition to active state.
- **C2:** Idle and power gated. This is a deeper sleep state and will have a greater latency when moving back to the C0 state relative to when coming out of C1.

The root/sudo user can enable and disable C-States. This example shows `cpupower monitor` output with all cores generally idling in C2:

```
# cpupower monitor
                | Mperf              || Idle_Stats
 PKG|CORE| CPU| C0   | Cx   | Freq  || POLL | C1   | C2
   0|   0|   0|  0.02| 99.98|  3687||  0.00| 99.97|  0.00
   0|   1|   1|  0.01| 99.99|  3685||  0.00| 99.90|  0.00
   0|   2|   2|  0.01| 99.99|  3685||  0.00| 99.90|  0.00
   0|   3|   3|  0.01| 99.99|  3688||  0.00| 99.95|  0.00
   0|   4|   4|  0.01| 99.99|  3685||  0.00| 100.1|  0.00
   0|   5|   5|  0.01| 99.99|  3687||  0.00| 99.95|  0.00
   0|   6|   6|  0.01| 99.99|  3685||  0.00| 100.1|  0.00
   0|   7|   7|  0.01| 99.99|  3684||  0.00| 99.98|  0.00
   0|  16|  72|  0.02| 99.98|  3683||  0.00| 99.98|  0.00
   ...etc...
```

This example shows disabling C2 (-d 2) for cores 0 to 3 (-c 0-3), which prevents them from idling down into C2, keeping them in state C1 or above, using the command `cpupower -c 0-3 idle-set -d 2`.

```
# cpupower -c 0-11 monitor
                | Mperf              || Idle_Stats
PKG|CORE| CPU| C0   | Cx   | Freq  || POLL | C1   | C2
   0|   0|   0|  0.06| 99.94|  3692||  0.00| 100.2|  0.00
   0|   1|   1|  0.01| 99.99|  3682||  0.00| 100.0|  0.00
   0|   2|   2|  0.01| 99.99|  3682||  0.01| 100.1|  0.00
   0|   3|   3|  0.01| 99.99|  3686||  0.00| 100.0|  0.00
   0|   4|   4|  0.03| 99.97|  3693||  0.00|  0.01| 99.99
   0|   5|   5|  0.00|100.00|  3692||  0.00|  0.00| 100.0
   0|   6|   6|  0.00|100.00|  3693||  0.00|  0.00| 100.0
   0|   7|   7|  0.00|100.00|  3692||  0.00|  0.00| 100.0
   0|  64|   8|  0.01| 99.99|  3693||  0.00|  0.00| 100.0
   0|  65|   9|  0.00|100.00|  3692||  0.00|  0.00| 100.0
   0|  66|  10|  0.00|100.00|  3691||  0.00|  0.00| 100.0
   0|  67|  11|  0.00|100.00|  3692||  0.00|  0.00| 100.0
```

The power gated C2 idle state can be re-enabled (-e 2) on these 4 cores using the command `cpupower -c 0-3 idle-set -e 2`.

You can apply these settings to all cores by omitting the `-c` argument in the `cpupower idle-set` command. When SMT is enabled, a core cannot enter C2 if either thread is in either the C0 (active) or C1 (idle) state. Therefore, if you are disabling C2 on any logical CPU, then you should also disable C2 on the SMT sibling. For example, in a 2x32 core system, if you disable C2 on CPU 7, then you should also disable C2 on CPU 71.

Disabling C2 is important for running a high-performance, low-latency network (such as Infiniband) because the latency required in moving from the power-gated C2 state to the active C0 state can add latency to network I/O. All cores must idle in C1 (with C2 disabled) to support a high-performance network.

2.3 Core P-States, Frequencies, and Boosting

P-States are execution power states that manage power usage while the cores are active, similar to how C-States manage the power levels when the cores are idle. All P-states only make sense when the core is in the C0 "active" state.

- P-States are execution power states.
- C-States are idle power saving states.

Once active in the C0 state, a core can move between SKU-specific predefined P-States based on its utilization to balance performance and power usage. A fully utilized core will target P0 state, which corresponds to its quoted base frequency. The root user can observe these P-States by executing the command `cpupower frequency-info`.

The following example shows the output for an AMD EPYC 9654 96-Core processor with the three P-States at frequencies of 1.5GHz (P2), 2.1GHz (P1) and 2.85GHz (P0):

```
# cpupower frequency-info
analyzing CPU 0:
  driver: acpi-cpufreq
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency:  Cannot determine or is not supported.
  hardware limits: 1.50 GHz - 3.71 GHz
  available frequency steps:  2.40 GHz, 1.90 GHz, 1.50 GHz
  available cpufreq governors: conservative ondemand userspace powersave performance
schedutil
  current policy: frequency should be within 1.50 GHz and 2.40 GHz.
                  The governor "performance" may decide which speed to use
                  within this range.
  current CPU frequency: 2.40 GHz (asserted by call to hardware)
  boost state support:
    Supported: yes
    Active: yes
    Boost States: 0
    Total States: 3
    Pstate-P0:  2400MHz
    Pstate-P1:  1900MHz
    Pstate-P2:  1500MHz
```

Enabling Boost allows a core to achieve yet another, higher frequency. This frequency is quoted as the Boost frequency for a CPU. Max Boost is the max frequency that any core can boost up to, provided there is enough thermal and computational headroom in the floating-point unit.

The preceding example shows that the AMD EPYC 9654 CPU has a quoted base frequency of 2.4 GHz. Enabling Boost allows the AMD EPYC 9654 cores to boost up to the quoted Boost frequency of 3.7 GHz. If all cores within a CPU attempt to boost this high, the overall CPU will likely reach a thermal power limit and reduce the effective frequency to a range between the base and boost frequencies.

Each core can run on a frequency higher than the one defined by P0 state, in a so-called Boost regime. If the Boost is enabled in BIOS and OS, a core can run up to the maximum boost frequency defined for a specific processor. The actual boost frequency on which a core runs at any point in time depends on the power and thermal parameters of a core and entire processor and is completely under processor's control. Once enabled in BIOS, you can toggle Boost on and off from the command line as the root user. For example, on a Red Hat Linux command line:

- **Enable Boost:** `echo 1 > /sys/devices/system/cpu/cpufreq/boost`
- **Disable Boost:** `echo 0 > /sys/devices/system/cpu/cpufreq/boost`

Changing the BIOS Boost setting requires a system reboot.

Note: AMD recommends setting the CPU governor to performance for HPC workloads. You user can observe this by running either the cpupower monitor or the AMD Micro Profiler (uProf) command as the root user. See "CPU Governors" on page 15.

2.4 CPU Governors

AMD EPYC supports several CPU governors, and you can apply different governors to different cores.

- **performance:** Sets the core frequency to the highest-available frequency within P0. With Boost set to OFF, it will operate at the base frequency (such as 2.45GHz on an AMD EPYC 7443 CPU). If Boost is ON, then it will attempt to boost the frequency to the Max Boost frequency of 3.5Ghz. This still represents the P0 P-State while operating at the boosted frequencies. HPC environments typically use this governor.
- **ondemand:** Sets the core frequency depending on trailing load. This favors a rapid ramp to highest operating frequency with a subsequent slow step down to P2 when idle. This could penalize short-lived threads.
- **conservative:** Similar to `ondemand` but favors a more graceful ramp to highest frequency and a rapid return to P2 at idle.
- **powersave:** sets the lowest supported core frequency, locking it to P2.

Administrators can set the CPU governor via the `cpupower` command. For example, to set the CPU governor to Performance: `cpupower frequency-set -g performance`.

See https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt* for a more extensive discussion and explanation around CPU governors within Linux.

2.5 Useful 'cpupower' Command Examples

The `cpupower` command is a very useful utility for querying and setting a range of conditions on the CPU. Here are some examples:

- **cpupower -c 0-15 monitor:** Displays the frequencies on cores 0 to 15. Useful if a user needs to observe the changes while turning Boost ON and OFF.
- **cpupower frequency-info:** Lists the boost state, CPU governor, and other useful information about the CPU configuration.
- **cpupower frequency-set -g performance:** Changes the CPU governor to performance.
- **cpupower -c 0-15 idle-set -d 2:** Disables the C2 idle state on CPUs 0 to 15.

See https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt* for a more extensive discussion and explanation around CPU governors within Linux.

Chapter 3 Quick HPC Setup

This section quickly describes some BIOS and OS settings you can use as a baseline when tuning an HPC system. You can then perform additional tuning beyond this baseline.

Note: Note: Consider reading the rest of this document before proceeding if you are not familiar with AMD EPYC processor architecture.

3.1 BIOS Settings

- **SMT:** OFF | ON (application-dependent based on which setting provides higher performance)
- **APBDIS:** 1
- **Fixed SOC P-State:** P0
- **Core Performance Boost:** ON
- **Determinism Slider:** Power
- **cTDP:** 400 (set to the max cTDP setting for your processor; for example, a 9654 is 400W)
- **PPL:** 400 (set to the max cTDP setting for your processor; for example, a 9654 is 400W)
- **NPS:** 4
- **DF C-States:** Disabled
- **Core C-states:** Enabled
- **TSME:** OFF
- **4-link xGMI max speed:** 32Gbps
- **xGMI Link Width Control:** Manual
- **xGMI Force Link Width Control:** Force
- **xGMI Force Link Width:** 2
- **xGMI Max Link Width Control:** Manual
- **xGMI Max Link Width:** 1
- **PCIe 10 bit tag:** Enabled

- **BoostFmaxEn:** Manual
- **BoostFmax:** 3700
- **HSMP Support:** Enabled
- **Workload Profile:** HPC Optimized
- **CPU Speculative Store Modes:** More Speculative
- **DRAM Survives Warm Reset:** Enabled

Platform vendors may include Workload Profiles or System Tunings for HPC. These may disable C-states (not recommended) and/or disable Core Performance Boost (also not recommended). Use a custom setting if you cannot revert these two settings in the workload profile.

3.2 OS Settings

Disable the C2 idle state for an HPC cluster with a high-performance, low-latency interconnect such as Infiniband. You must include all cores when disabling C2. For example, for a dual-socket system with 96-core processors, use 0-191 in the command to disable C2 for all 192 cores by executing the command `cpupower -c 0-191 idle-set -d 2`. Also, set the CPU governor to performance by executing the command `cpupower frequency-set -g performance`.

3.3 Basic System Checks

- **Check if SMT is enabled:** `lscpu` (Thread(s)=1 implies SMT=OFF; Threads (2)=2 implies SMT=ON)
- **Check to which NUMA node your Infiniband card or other peripherals are attached: hwloc-ls**
- **Check if boost is ON (1) or OFF (0):** `cat /sys/devices/system/cpu/cpufreq/boost`
- **Check CPU governor and other useful settings:** `cpupower frequency-info`
- **Visually check which cores/threads are busy:** `htop`
- **Check core frequencies and idle states:** `cpupower monitor`

Run the STREAM memory benchmark. A dual-socket system with DDR5-4800 Dual Rank DIMMS with one DIMM per channel (8 cores per L3 on 96 core CPU) should yield approximately 770GB/s with AOCC or Intel compiler. Use four cores per L3 (CCD) on 96-core CPUs to get maximum STREAM results.

3.4 Useful Commands

This section lists some additional useful commands.

3.4.1 Build CPU ID lists with 'seq'

You may sometimes need to build comma-delimited lists of CPU IDs. Use the Linux `seq` command.

- **System with 2x 96 core CPUs, 2 cores per L3:**
  `seq -s, 0 4 191`
- **System with 2x 96 core CPU, 3 cores per L3 (join 2 lists, every fourth core + every eight core):**
  `seq -s , 0 4 191 | tr -d '\n'; echo ,| tr -d '\n'; echo "$(seq -s , 2 8 191)"`

3.4.2 Core Pinning and Memory Locality:

You can pin your binary to run on a specific core, such as `core 7: numactl -C 7 ./mybinary`

This places the thread but still allows the kernel to freely choose which memory slots to use. To ensure the memory is logically closest to the core, add the argument `--membind=node`. It will allocate memory only on the specified nodes. For example:

```
numactl -C 7 --membind=0 ./mybinary

To establish which NUMA node your core belongs to, use the output from numactl -H:
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
node 0 size: 96486 MB
node 0 free: 95793 MB
node 1 cpus: 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
node 1 size: 96754 MB
node 1 free: 96224 MB
node 2 cpus: 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
node 2 size: 96754 MB
node 2 free: 95686 MB
node 3 cpus: 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
node 3 size: 96754 MB
node 3 free: 95283 MB
node 4 cpus: 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116
117 118 119
node 4 size: 96716 MB
node 4 free: 95695 MB
node 5 cpus: 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143
node 5 size: 96700 MB
node 5 free: 95730 MB
node 6 cpus: 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163
164 165 166 167
node 6 size: 96754 MB
node 6 free: 95959 MB
node 7 cpus: 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
188 189 190 191
node 7 size: 96754 MB
node 7 free: 96220 MB
```

In this example, the core with CPU ID 7 is in NUMA domain 0.

3.4.3 Faster 'make' with 'make -j'

Executing the command `./configure ; make ; make install` is a common way to build code from a source. The `make` step will implicitly use a single core for the compilation. AMD EPYC processors allow you to significantly speed up compilation by passing the `-j` flag with a number representing the maximum number of threads you want the compiler to use. For example, on a dual socket 2x 96 core system with SMT=ON you now have 384 threads and could use `./configure ; make -j 384 ; make install` to allow the compiler to leverage up to 384 threads for compilation. This will dramatically reduce build times for large code sources.

Chapter 4 BIOS Settings

This section describes common BIOS settings within a High-Performance Computing environment.

4.1 Recommended BIOS Settings for Bare Metal Workloads

These guidelines will help get you started with the tuning. Make sure to evaluate your needs and apply the appropriate settings that are best for your requirements. Check your server manufacturer's guide if you cannot find the exact options shown in this section.

4.1.1 Determinism (Performance | Power)

Power and Performance determinism modes determine which policy the System Management Unit (SMU) use to determine the effective frequency of each core. The SME tracks real-time power, current draw, and temperatures across all parts of the processor and has knowledge of the cTDP power budget.

- **Performance:** Reduces CPU-to-CPU performance variability within a cluster. It provides repeatable performance for identical SKUs across your cluster by using a baseline reference setting that is common to all CPUs of that model number.
- **Power:** Takes advantage of yield variability and can provide improved runtime performance. It allows each processor to draw power up to its individual capability while keeping the part from exceeding its cTDP power limit. You must set this option to **Power** in order to set the cTDP to its maximum value.

4.1.2 TDP (configurable Thermal Design Point)

Each CPU has a minimum and maximum TDP threshold. Set TDP=PPL.

4.1.3 PPL (Package Power Limit)

Every CPU has a maximum PPL limit. Ensure PPL=TDP. You can set PPL lower than the TDP minimum.

4.1.4 Simultaneous Multi-Threading (SMT) Enabled | Disabled

Enabling SMT makes each core exhibit two threads. You can use the command `hwloc-ls` to obtain a list of CPUs IDs for each thread. Choose whether or not to enable this option on a per-application basis. Many HPC workloads typically disable SMT. If you are not in a compute-bound scenario, then you may see some benefit from enabling SMT. If your code is not licensed per core or if there is no other monetary impact for enabling SMT, then you may want to experiment to see whether it helps your workload.

- **Enabled:** Allows 1 core to execute 2 threads.
- **Disabled:** Allows 1 core to execute 1 thread.

AMD further recommends using `iommu=pt` as a kernel boot parameter.

4.1.5 X2APIC = Auto

This option helps the operating system deal with interrupts more efficiently in high-core-count configurations.

4.1.6 Core Performance Boost = OFF | ON

Turns the boost function ON or OFF on all cores. This can also be toggled on or off via the Linux command line as root. This requires Core Performance Boost to already be enabled/on in the BIOS at boot up. If it set to OFF in BIOS, it cannot be toggled on the Linux command line).

4.1.7 Memory speed = AUTO

AUTO allows the system to automatically train to the correct speed setting for a given DIMM population and memory rank. You can clock this down to save power and increase boost headroom for applications that are not sensitive to memory speed.

4.1.8 Core C-States = Enabled

Refers to CPU C-States. Leave these enabled. If required, disable C2 via the Linux command line as root.

4.1.9 Data Fabric C-States (DF C-States) Enabled | Disabled

The Infinity fabric can enter lower C-States to save power during long idle periods. Disable this feature for HPC workloads to eliminate latency caused by coming out of the lower C-State. This setting does not matter if core C2 is disabled in the OS.

4.1.10 NPS = 1 | 2 |4

Sets the NUMA domains Per Socket by interleaving pairs of memory channels. Many HPC applications allow pinning ranks and memory to cores and NUMA nodes. The typical recommendation in this case is to use the NPS=4 option. If your workload is not very well NUMA aware or suffers when NUMA complexity increases, then you can experiment with NPS=1.

4.1.11 CCD Control

This option allows you to disable cores by modifying the number of active CCDs in the processor. You can use this together with Core Control to change the effective part layout. See below.

4.1.12 Down-Coring / Core Control

This option allows you to disable cores by modifying the number of active cores in each CCD. The options appear as (x), where x is the number of active cores per CCD. For example, setting this to (6) means there are 6 active cores per CCD. If the part has 8 CCDs, then you have a total of 48 cores, because 6 cores per CCD * 8 CCDs = 48 total cores.

Note: You will typically use these options to simulate the behavior of different part configurations with your workload. This experiment can help you to identify the best part configuration for your workload and needs.

4.1.13 Transparent Secure Memory Encryption (TSME) = OFF

Secure Memory Encryption encrypts all memory. The following settings disable Dynamic Link Width Management (DLWM), which forces the xGMI links to always run at x16 lanes. This maximizes the bandwidth and minimizes the latency for traffic between the sockets.

This page intentionally left blank.

Chapter 5 Operating Systems

5.1 Linux Kernel and ISV OS Support Considerations

There have been several AMD EPYC-related patches. You may either:

- Apply these patches manually.
- Deploy an OS with a suitably up-to-date kernel to avoid manual patching.
- Apply all updates to your supported OS through the package manager.

Please see AMD EPYC™ Processors Minimum Operating System (OS) Versions for detailed OS version information.

Note: Confirm that your planned OS supports your applications.

5.2 /proc and /sys

There are several ways to consume memory over the uptime of a system. This section summarizes some of the areas that affect performance and enable memory 'clean up' in NUMA systems. See https://www.kernel.org/doc/Documentation/sysctl/vm.txt* for a detailed discussion of the Linux Virtual Memory system.

5.2.1 /proc/sys/vm/zone_reclaim_mode

From www.kernel.org*: `zone_reclaim_mode` *allows someone to set aggressive approaches to reclaim memory when a zone runs out of memory. If it is set to zero, then no zone reclaim occurs. Allocations will be satisfied from other zones / nodes in the system*.

The following three bits control kernel behavior:

- **1:** Zone reclaim on
- **2:** Zone reclaim writes dirty pages out
- **4:** Zone reclaim swaps pages

These can be logically OR'ed; that is, a setting of 3 (1+2) is permitted.

Zone reclaim is usually turned off for workloads that benefit from having file system data cached. Balanced jobs where either job size exceeds the memory of a NUMA node and/or your job is multi-cored and extends outside the NUMA node probably benefit from enabling this feature. See https://www.kernel.org//Documentation/sysctl/vm.txt*.

All 2P AMD EPYC systems implement Zone Reclaim with respect to the remote socket. AMD recommends setting this to either 1 or 3.

5.2.2 /proc/sys/vm/drop_caches

After running applications in Linux, you may find that your available memory reduces while your buffer memory increases, despite not running any applications. For example, the `numactl -H` command will show which NUMA node(s) the memory is buffered with (possibly all). Linux users with root or sudo permissions have three ways to clean the caches and return buffered or cached memory to 'free':

- `echo 1 > /proc/sys/vm/drop_caches [frees page-cache]`
- `echo 2 > /proc/sys/vm/drop_caches [frees slab objects e.g., dentries, inodes]`
- `echo 3 > /proc/sys/vm/drop_caches [cleans page-cache and slab objects]`

HPC systems often benefit from cleaning up buffered and cached memory after a job has finished and before assigning the same node to the next user. For example, SLURM can accomplish this using an epilog script.

AMD recommends disabling swap. If you need to use swap, then you need to increase the node memory capacity. Ensure that all nodes have sufficient memory to handle your workloads. Disabling swap without sufficient memory can have undesired effects.

```
swapoff -a
```

5.3 Transparent Huge Pages (THP)

If transparent hugepages are not required, then you can disable them by executing the following commands:

```
echo 'never' > /sys/kernel/mm/transparent_hugepage/enabled
echo 'never' > /sys/kernel/mm/transparent_hugepage/defrag
```

5.4 Explicit Hugepages

You can disable transparent hugepages if your application supports explicit hugepages. Follow the application's guidance for allocating explicit hugepages at boot time. For example, HPL performance improves slightly by disabling THP and enabling hugepages. To reserve 240GB with 2m hugepages:

```
echo 3 > /proc/sys/vm/drop_caches
echo 1 > /proc/sys/vm/compact_memory
$number_huge_2m_pages = 240 * 1024 / 2
echo $number_huge_2m_pages > /proc/sys/vm/nr_hugepages
```

where `$number_huge_2m_pages` = 240 * 1024 / 2.

Execute `mybinary.bin` using `hugectl` with the reserved hugepages in place:

```
hugectl --force-preload -heap mybinary.bin
```

One advantage of this method over THPS is that the code will warn you if it runs out of hugepages or cannot allocate the hugepages the binary requires.

5.5 Randomize_va_space

Address space randomization is a security feature that guards against security hacks. To disable this feature:

```
echo 0 > /proc/sys/kernel/randomize_va_space
```

5.6 NUMA Balancing

The NUMA balancing feature that allows the OS to scan memory and attempt to migrate to a DIMM that is logically closer to the cores accessing it. This causes an overhead because the OS is second-guessing your NUMA allocations but may be useful if the NUMA locality access is very poor. Most HPC applications can therefore benefit from disabling NUMA balancing. For example, the STREAM memory bandwidth benchmark runs a slower with NUMA balancing enabled. To disable:

```
echo 0 > /proc/sys/kernel/numa_balancing
```

For more information:

- https://documentation.suse.com/sles/15-SP1/html/SLES-all/cha-tuning-numactl.html*
- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_tuning_and_optimization_guide/sect-virtualization_tuning_optimization_guide-numa-auto_numa_balancing*

5.7 Spectre and Meltdown

In 2018, Google Project Zero (GPZ) announced several vulnerabilities concerning speculative execution that take three variants. Variant-3 (called *Meltdown*) does not affect AMD EPYC CPUs, but Variant-1 and Variant-2 (*Spectre*) do affect AMD EPYC CPUs. Please visit https://www.amd.com/en/corporate/speculative-execution-previous-updates#paragraph-337801 for a more complete discussion by the AMD Chief Technology Officer about these vulnerabilities.

Newer kernels (see "Linux Kernel and ISV OS Support Considerations" on page 25) automatically apply patches to help protect against these vulnerabilities at the expense of a small performance of a few percent. Some customers are electing to keep their 'edge nodes' or 'head' nodes (nodes that connect from their organization to the outside) patched against these vulnerabilities while disabling these patches for compute nodes that are logically isolated inside their organization. See https://access.redhat.com/articles/3311301*. In general, a root user can disable these patches by setting the single-entry value to '0' in two files:

```
echo 0 > /sys/kernel/debug/x86/retp_enabled
echo 0 > /sys/kernel/debug/x86/ibpb_enabled
```

You can then view the two files to verify that the Spectre patches are now disabled:

```
cat /sys/kernel/debug/x86/retp_enabled
1
cat /sys/kernel/debug/x86/ibpb_enabled
1
```

This page intentionally left blank.

Chapter 6 Linux Perf

Performance Counters for Linux is a subsystem that offers unified handling and tooling performance analysis. The `perf` tool is the most common way to utilize this subsystem. Non-root users can normally only access context-switched events, which are typically predefined CPU Performance Monitoring Unit (PMU) events that include cycles and instructions and some software events. Only root users can normally access other PMUs and global measurements. You can override this by setting the `kernel.perf_event_paranoid sysctl` to -1 using either:

- `sudo echo -1 > /proc/sys/kernel/perf_event_paranoid`
- `sudo sysctl -w kernel.perf_event_paranoid=-1`

The perf list shows two generic L2 cache events:

- `cache-misses  [Hardware event]`
- `cache-references  [Hardware event]`

For example, execute the following command to measure L2 cache events on your system for 2 seconds:

```
perf stat -a -e cache-misses,cache-references sleep 2'
```

Raw event specification also can be used. For example:

- `cpu/event=0x60,umask=0xff/` for cache-references.
- `cpu/event=0x64,umask=0x09/` for cache-misses.

See AMD Developer Central for a full list of available hardware events in the *Processor Programming Reference (PPR) for Family 19h Models ooh-oFh*. The PPR at the time this document was published contains a table called *Guidance for Common Performance Statistics with Complex Event Selects* that describes raw events for L1 Data Cache (DC) Fills.

Note: When searching the PPR, add the prefix `PMCx` *to the three-hexadecimal-digit number.*

You can obtain information about DC fills from within the same CCX, from external CCX cache, and from a remote node. Doing this requires you to copy the raw hex values and assign them using `config=0x<rawvalue>`. For example, to get L1 DC fills within the same CCX together with DC fills from external CCX:

```
perf stat -e cpu/config=0x430344/,cpu/config=0x431444/ perf bench mem memcpy'
```

This example uses the `memcpy` memory system `perf` benchmark.

The *Guidance for Common Performance Statistics with Complex Event Selects* table also contains raw events for L3 cache accesses and L3 misses. You can obtain the average L3 cache read miss latency (in core clocks) using the `amd_l3` PMU subsystem with the following formula:

Average L3 Cache Read Miss Latency= (L3Event[0x0300C00000400090] * 16)/(L3Event[0x0300C00000401F9a])

The `perf` bench memory benchmark uses the following command:

```
$ perf stat -e amd_l3/config=0x0300C00000400090/,amd_l3/config=0x0300C00000401F9a/ perf
bench mem memcpy
Performance counter stats for 'system wide':
        13,584,923      amd_l3/config=0x0300C00000400090/
           441,401      amd_l3/config=0x0300C00000401F9a/
```

In this example, the average L3 cache miss latency is 492 core cycles.

For the Data Fabric (DF), you can currently only use the raw event specification (DFEvent). You must monitor the following DFEvents mentioned in the PPR to compute memory bandwidth:

```
DfEvent[0x0000000000403807]
DfEvent[0x0000000000403847]
DfEvent[0x0000000000403887]
DfEvent[0x00000000004038C7]
DfEvent[0x0000000100403807]
DfEvent[0x0000000100403847]
DfEvent[0x0000000100403887]
DfEvent[0x00000001004038C7]
```

The kernel driver uses four DF counters. Only specify four at one time to ensure that `perf` does not multiplex them. You can get the memory bandwidth from the STREAM benchmark workload by using the formula from the PPR for the combined DRAM bytes of all the channels. Run the STREAM benchmark and `perf` command simultaneously on the same node to get a rough idea of bandwidth.

The peak value of the DFEvent counter equals 716,685,795. Multiplying this by 8*64B gives approximately 367 GB/s, which is a reasonable estimate for an AMD EPYC 7763 processor with 1 thread per L3.

Chapter 7 Host System Management Port

The Host System Management Port (HSMP) is an interface that grants OS-level software access to system management functions via a set of mailbox registers. The power management firmware in the SMU implements HSMP interface functionality. See AMD Developer Central for full HSMP documentation in the Processor Programming Reference (PPR) for Family 19h.

Some of the things that HSMP lets you change on-the-fly include:

- Package Power Limit
- Maximum boost clock (upper CCLK limit) for either all cores or a subset of cores.
- NBIO LCLK DPM levels (or fixed DPM level to highest power / lowest latency state)
- Data Fabric P-state levels (or fixed P-State)
- xGMI Dynamic Link Width Management (DLWM) levels (or fixed link width)

7.1 HSMP Driver

The `libhsmp.so` userspace library provides user-level access to the HSMP mailboxes. The AMD HSMP kernel module source code (`amd_hsmp`) is available from https://github.com/amd/amd_hsmp and includes documentation.

The AMD HSMP driver is now part of the Linux kernel upstream starting in v5.18-rc1. Review kernel.org for updated versions of this driver.

The AMD EPYC™ System Management Interface (E-SMI) In-band library provides C APIs for the user space application on top of this module.

The following repositories contain the installation instructions and command details to build and run the E-SMI command line tool:

- https://github.com/amd/esmi_ib_library
- https://github.com/amd/amd_energy

Here is an example of the E-SMI command to set power limit for a given socket in mWatts:
`./e_smi_tool --setpowerlimit [SOCKET] [POWER] (05h)`

This page intentionally left blank.

Chapter 8 Executing Applications on AMD EPYC 9004 Processors

8.1 Characterization Strategy

You may need to consider several settings when tuning an HPC server to determine the optimal configuration for your application. For example, Table 8-1 shows a matrix of SMT ON/OFF, Boost ON/OFF, across 8 different CPU core counts/configurations. This type of table can help guide your investigation and test plan.

Cores/L3	Cores/ Socket	SMT=OFF		SMT=ON	
		Boost=OFF	Boost=ON	Boost=OFF	Boost=ON
1	8				
2	16				
…	…				
8	96				

Table 8-1: Sample settings comparison table

This specific example helps you understand:

- The benefits of SMT and boost.
- The benefits of increasing core count per L3.

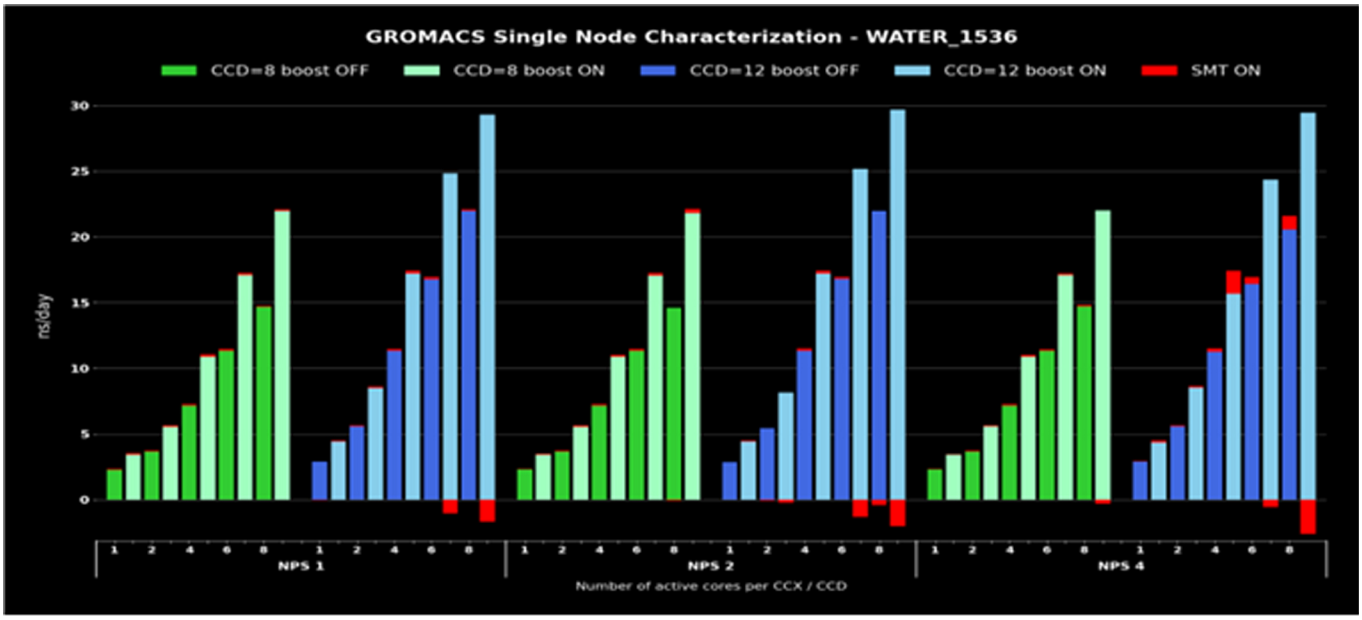

Figure 8-1: Sample GROMACS performance characterization

You could, for example, notice performance dropping beyond 5 cores, which can help you determine the most appropriate setting for executing your application. Information like this can even affect an entire procurement.

You could, for example, notice performance dropping beyond 5 cores, which can help you determine the most appropriate setting for executing your application. Information like this can even affect an entire procurement.

Figure 8-1 shows an example of characterizing the GROMACS molecular dynamics application using the water_1536 test case (higher is better) where we test 1/2/4/6/8 cores per L3 for each NPS setting.

The GROMACS characterization study used a server with 2x 96-core AMD EPYC 9654 CPUs with 1,2,..,8 cores per L3/CCD. The AMD EPYC 9654 CPU provides the flexibility to test NPS=1/2/4 and reduce CCD count from 12 to 8 CCDs. You can now determine whether a 96 or 64 core CPU is the most appropriate for maximum performance.

Figure 8-1 shows two bars for each core count per CCD, that is, 1 core per CCD:

- **BOOST=OFF:** Dark blue/green
- **BOOST=ON:** Corresponding pale blue/green

The red portion at the top of the bar shows the incremental benefit of SMT=ON, whereas the portion under the horizontal axes shows detrimental effect of using SMT threads in a particular setting. This example achieves best performance with NPS=4, 12x CCDs, 8 cores per L3 (96 core CPU), Boost=ON, and SMT=ON..

8.2 Pinning Strategies and Hybrid Codes

The most reliable way to pin cores is generally via an appfile. Performance may drop slightly when relying on the OpenMPI interface for CPU ID (`-cpu-list`). Explicit pinning remains advisable when using all the cores/threads within the socket to ensure minimum execution time. Hybrid MPI + OpenMP codes normally offer three strategies that you can adopt when executing code:

- **All MPI ranks:** Per available hardware thread or core.
- **Map by L3 cache:** 1 or more MPI ranks per L3 with the rest of the cores running OpenMP threads. AMD EPYC 9004 CPUs have up to 8 cores per L3 cache, which means that you could try following combinations of the MPI ranks and OpenMP threads per L3 cache:
  - **With SMT disabled:** 8 hardware threads available per L3 cache): 1 rank and 8 threads, 2 ranks and 4 threads and 4 ranks and two threads.
  - **With SMT enabled:** 16 hardware threads per L3 cache): 1 ranks and 16 threads, 2 ranks and 8 threads, 4 ranks and 4 threads and 8 ranks with two threads per rank.
- **Map by core:** if SMT=ON, then you can choose 1 MPI rank per core + OMP_NUM_THREADS=2.

For example, to execute `mybinary` using OpenMPI with 2x AMD EPYC 9654 (i.e. 2x 96-core CPUs) with:

- SMT=OFF
- 1 MPI rank per L3, no OpenMP threads

```
export CPULIST=$(seq -s, 0 8 191)
mpirun --bind-to none -cpu-list $CPULIST --app myappfile-1rankperL3.txt
```

The accompanying appfile `myappfile-1rankperL3.txt` is therefore:

```
-np     1     numact l     --physcpubind=0 mybinary -parallel
-np     1     numact l     --physcpubind=8 mybinary -parallel
-np     1     numact l     --physcpubind=16 mybinary -parallel
-np     1     numact l     --physcpubind=24 mybinary -parallel
-np     1     numact l     --physcpubind=32 mybinary -parallel
-np     1     numact l     --physcpubind=40 mybinary -parallel
-np     1     numact l     --physcpubind=48 mybinary -parallel
-np     1     numact l     --physcpubind=56 mybinary -parallel
-np     1     numact l     --physcpubind=64 mybinary -parallel
-np     1     numact l     --physcpubind=72 mybinary -parallel
-np     1     numact l     --physcpubind=80 mybinary -parallel
-np     1     numact l     --physcpubind=88 mybinary -parallel
-np     1     numact l     --physcpubind=96 mybinary -parallel
-np     1     numact l     --physcpubind=104 mybinary -parallel
-np     1     numact l     --physcpubind=112 mybinary -parallel
-np     1     numact l     --physcpubind=120 mybinary -parallel
-np     1     numact l     --physcpubind=128 mybinary -parallel
-np     1     numact l     --physcpubind=136 mybinary -parallel
-np     1     numact l     --physcpubind=144 mybinary -parallel
-np     1     numact l     --physcpubind=152 mybinary -parallel
-np     1     numact l     --physcpubind=160 mybinary -parallel
-np     1     numact l     --physcpubind=168 mybinary -parallel
-np     1     numact l     --physcpubind=176 mybinary -parallel
-np     1     numact l     --physcpubind=184 mybinary -parallel
```

8.3 Microbenchmark Results

This workload results chapter contains the outcome of different benchmarks measuring the performance of AMD EPYC 9004 systems. These workloads exercise different capabilities of the system and the results show how well the system performed. The standard specifications of the CPU models presented here do not vary. However, run-to-run, sample-to-sample, and system-to-system variations can accumulate to impact measured performance from the data reported in this document. The results of tests run against key enterprise/server benchmarks are provided as a reference to help you assess performance across different workloads. This chapter identifies the performance results that AMD obtained during testing and that you may be able to use for comparison to determine acceptable results. Compare your averaged results with those shown. If the results are significantly outside of the min/max range, then further investigation is needed to determine why performance is not as expected. This could be a configuration, design, or other type of defect.

8.3.1 Stream

STREAM tests the maximum memory bandwidth of either a core or an entire CPU. You can download an optimized binary https://www.amd.com/en/developer/zen-software-studio/applications/pre-built-applications.html. Alternatively, you can build the STREAM benchmark using Spack:

1. Build STREAM using Spack:
   ```
   spack install stream %aocc@4.0.0 +openmp cflags="-O3 -mcmodel=large -
   DSTREAM_TYPE=double - mavx2 -DSTREAM_ARRAY_SIZE=430080000 -DNTIMES=10 -ffp-
   contract=fast -fnt-store -mprefer-vector-width=256 -mllvm -force-vector-interleave=2 "
   ```
2. Load the STREAM build:
   ```
   spack load stream
   ```
3. Verify that you have root/sudo permissions.

4. Setup your environment for STREAM:
```
#!/bin/bash
echo 0 > /proc/sys/kernel/randomize_va_space echo 0 > /proc/sys/vm/nr_hugepages
echo 0 > /proc/sys/kernel/numa_balancing
echo 'always' > /sys/kernel/mm/transparent_hugepage/enabled
echo 'always' > /sys/kernel/mm/transparent_hugepage/defrag
```

5. Run STREAM (e.g., on a 2P AMD EPYC 9654 node):
```
#!/bin/bash
export GOMP_CPU_AFFINITY=0-191:2
export OMP_NUM_THREADS=48 stream_c.exe
```

Be sure to adjust the `STREAM_ARRAY_SIZE` parameter according to the amount of L3 cache per core in order to reach the optimal STREAM score. The following values deliver optimal results:

SKU	# of CCDs	STREAM_ARRAY_SIZE	Memory per array
9654	8	286720000	2.1 GiB
9654	12	430080000	3.2 GiB
9754	8	286720000	2.1 GiB
9684X	8	860160000	6.4 GiB
9684X	12	1290240000	9.6 GiB

Table 8-2: Optimal STREAM values by AMD EPYC model

The following tables compare STREAM Triad results (MB/s) on a system with 2x 9654 and 1 DPC (DIMM Per Channel), with DDR5-4800 Dual-Rank DIMMs. The tables also show the effect of changing NUMA domains per socket from 4 to 2 to 1 (NPS), as well as adjusting the number of active cores per L3.

CCD=12 NPS=4		**SMT=OFF**			**SMT=ON**		
		Threads	Boost=OFF	Boost=ON	Threads	Boost=OFF	Boost=ON
Cores/L3	1	24	723532	739500	24	719755	743973
	2	48	764642	770343	48	746783	749148
	4	96	760919	771153	96	756082	760759
	6	144	760613	774746	144	745639	750421
	8	192	750525	766863	192	660123	655651

Table 8-3: AMD EPYC 9654 SMT comparison, NPS=4

CCD=12 NPS=2		**SMT=OFF**			**SMT=ON**		
		Threads	Boost=OFF	Boost=ON	Threads	Boost=OFF	Boost=ON
Cores/L3	1	24	703990	716053	24	695083	715213
	2	48	735180	740347	48	720450	721410
	4	96	732208	743602	96	733610	737283
	6	144	739500	751450	144	733188	737484
	8	192	731639	746076	192	668127	653251

Table 8-4: AMD EPYC 9654 SMT comparison, NPS=2

CCD=12 NPS=1		**SMT=OFF**			**SMT=ON**		
		Threads	Boost=OFF	Boost=ON	Threads	Boost=OFF	Boost=ON
Cores/L3	1	24	708727	719959	24	684977	701367
	2	48	723121	726995	48	704185	707268
	4	96	719396	729003	96	718191	720510
	6	144	727300	737220	144	717536	723786
	8	192	733088	726641	192	650205	641743

Table 8-5: AMD EPYC 9654 SMT comparison, NPS=1

Table 8-6 shows how performance changes when CCDs are reduced from CCD=12 to CCD=8 per socket of an AMD EPYC 9654 running STREAM with the same array size of 430080000:

CCD=8 NPS=4		**SMT=OFF**			**SMT=ON**		
		Threads	Boost=OFF	Boost=ON	Threads	Boost=OFF	Boost=ON
Cores/L3	1	16	629308	651350	16	620230	663539
	2	32	759358	771730	32	736280	751946
	4	64	781905	784341	64	764764	762729
	6	96	776380	779301	96	759531	767203
	8	128	748345	765102	128	738970	666440

Table 8-6: AMD EPYC 9654 STREAM performance changes while changing CCD=8 to CCD=4 per socket

Tables 8-7 and 8-8 show STREAM performance for a 2P AMD EPYC 9654 and a 2P AMD EPYC 9684X system with DDR5-4800 Dual-Rank DIMMS:

CCD=8 NPS=4		**SMT=OFF**			**SMT=ON**		
		Threads	Boost=OFF	Boost=ON	Threads	Boost=OFF	Boost=ON
Cores/L3	1	16	755515	755930	16	744792	759611
	2	32	779132	782937	32	762642	764500
	4	64	781686	787422	64	767489	771799
	6	96	775532	780101	96	760792	760291
	8	128	765189	774657	128	751561	747647

Table 8-7: AMD EPYC 9754 STREAM performance for CCD8, NPS4

CCD=12 NPS=4		SMT=OFF			SMT=ON		
		Threads	Boost=OFF	Boost=ON	Threads	Boost=OFF	Boost=ON
Cores/L3	1	24	712529	717866	24	717715	732778
	2	48	750846	755924	48	733838	736001
	4	96	752948	756739	96	736614	740828
	6	144	751267	755686	144	734660	737212
	8	192	751780	753407	192	732621	734918

Table 8-8: AMD EPYC 9684X STREAM performance for CCD12, NPS4

The following table compares STREAM results for both a 2 x AMD EPYC 9654 96c SKU and a AMD EPYC 9754 128c SKU node with NPS=4, SMT=off, Boost=ON with single vs. dual rank DIMMs.

STREAM - 3.2 GiB (12 CCD) and 2.1 GiB (8 CCD) array size (MB/s)								
Cores per L3	AMD EPYC 9654					AMD EPYC 9764		
	Threads (8/12 CCD)	Single Rank DIMMs		Dual Rank DIMMs		Threads	Single Rank DIMMs	Dual Rank DIMMs
		8 CCDs	12 CCDs	8 CCDs	12 CCDs		8 CCDs	8 CCDs
1	16/24	655393	720198	656994	742480	16	746680	755930
2	32/48	728745	728071	772818	772308	32	756604	782937
4	64/96	726434	722928	787566	771964	64	750661	787422
6	96/144	722723	717750	784398	774630	96	744331	780101
8	128/192	712846	711031	778488	766348	128	738012	774657

Table 8-9: Comparing single- vs. dual-rank DIMMs for AMD EPYC 9654 and AMD EPYC 9754

8.4 HPL

HPL is a benchmark that stresses the CPU cores by solving a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers. AMD Zen Software Studio offers a prebuilt optimized HPL version based on HPL version 2.3 of the software developed at the University of Tennessee, Knoxville, Innovating Computing Laboratory. The optimized version is built with the AOCC 4.0 compiler using a customized version of AMD's AOCL 4.0 libraries. AMD-optimized HPL is offered as a static binary executable that can be downloaded from https://www.amd.com/en/developer/zen-software-studio/applications/pre-built-applications/zen-hpl-eula.html?filename=amd-zen-hpl-2022_11.tar.gz.

This binary was built with AVX512 support and will only run properly on systems that support AVX512 instructions.

HPL requires tuning for each processor type/configuration and memory configuration. The following example assumes a dual-socket system with 96-core EPYC 9654 processors and 768GB of memory, configured with SMT=OFF, cTDP=PPL=400W, and memory speed = 4800 MHz.

To run the optimized HPL binary:

1. Create the following scripts to run HPL:

   - `run_hpl_opt.sh`

```
#! /bin/bash
module load hpcx/aocc40/2.13.0
export OMPI_MCA_pml=ucx
export OMPI_MCA_osc=ucx
export OMPI_MCA_spml=ucx
export OMPI_MCA_btl=^vader,tcp,openib,uct
export OMPI_MCA_coll_hcoll_enable=1
export OMPI_MCA_hwloc_base_binding_policy=none
 ldd xhpl
which mpicc
NT=96
NR=2
MAP_BY=socket
mpirun --map-by ${MAP_BY}:PE=$NT -np $NR --bind-to core -x OMP_NUM_THREADS=$NT -x
OMP_PROC_BIND=close -x OMP_PLACES=cores ./bindmem.sh ./xhpl
```

   - `HPL.dat`. The following sample provided is for a server with 768GB of memory. You must adjust this to suit your needs.

2. Execute the command `./run_hpl_opt.sh`.

```
PLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6           device out (6=stdout,7=stderr,file)
1           # of problems sizes (N)
197376 356352 165888 144384 127600         Ns
1           # of NBs
384      #  NBs
0           PMAP process mapping (0=Row-,1=Column-major)
1           # of process grids (P x Q)
1            Ps
2            Qs
16.0        threshold
1           # of panel fact
1           PFACTs (0=left, 1=Crout, 2=Right)
1           # of recursive stopping criterium
48           NBMINs (>= 1)
1           # of panels in recursion
8           NDIVs
1           # of recursive panel fact.
2           RFACTs (0=left, 1=Crout, 2=Right)
1           # of broadcast
7           BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM,6=Psh,7=Psh2)
1           # of lookahead depth
0           DEPTHs (>=0)
2           SWAP (0=bin-exch,1=long,2=mix)
64          swapping threshold
0           L1 in (0=transposed,1=no-transposed) form
0           U  in (0=transposed,1=no-transposed) form
1           Equilibration (0=no,1=yes)
8           memory alignment in double (> 0)
```

Using this process with AMD optimized HPL binary on 2-socket test servers results in 8.86 TFLOPS using AMD EPYC 9654 processors (2x 96 cores).

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR0XR8C48      197376   384     1     2             578.83             8.8562e+03
HPL_pdgesv() start time Wed Feb  8 20:07:54 2023
```

The following table summarizes results for 2P AMD EPYC 9654, 9684X, and 9754 nodes:

SKU	Cores	GFLOP/s
9654	192	8856
9754	256	10134
9684X	192	8620

Table 8-10: Summarized HPL results for various AMD EPYC SKUs

8.5 DGEMM

The DGEMM benchmark stresses the system cores and can be used to calculate the FLOPS of a core, CCX, NUMA node, or socket. DGEMM measures the floating point rate of execution of double precision real matrix-matrix multiplication. DGEMM can be run as part of the HPC Challenge (HPCC) benchmark suite, which measures a range memory access pattern. HPCC can be built via spack.

```
spack install hpcc +openmp %aocc@4.0.0 cflags="-O3” ^amdblis@4.0.0 threads=openmp
^openmpi@4.1.1
```

To run across all 192 cores of a dual-socket AMD EPYC 9654 system:

```
mpirun --map-by socket:PE=96 -np 2 --bind-to core -x OMP_NUM_THREADS=96 -x
OMP_PROC_BIND=close -x OMP_PLACES=cores ./dgemm 16100 30
```

You can derive other combinations (per L3, per CCD) by carefully choosing which cores to pin to. The result on 2-socket AMD EPYC 9654 node is: `GFLOP/s rate: 8595 GF/s`.

The following table compares DGEMM results for 2P AMD EPYC 9654, 9754, and 9684X nodes:

SKU	Cores	GFLOP/s
9654	192	8658
9754	256	10730
9684X	192	8525

Table 8-11: Summarized DGEMM results for various AMD EPYC SKUs

8.6 HPCG

The High-Performance Conjugate Gradient (HPCG) benchmark is designed to be the successor of the HPL benchmark. The motivation to create a new benchmark was to produce a memory performance metric of HPC systems that better reflects the performance of common application workloads on a modern architecture. HPC workloads are no longer limited by CPU performance alone but may also be sensitive to memory performance.

As with the HPL binary, AMD Zen Software Studio offers a pre-built optimized HPCG version with AVX512 support and with the AOCC 4.0 compiler using a customized version of AMD's AOCL 4.0 libraries. AMD-optimized HPCG is offered as a static binary executable and can downloaded from https://www.amd.com/en/developer/zen-software-studio/applications/pre-built-applications/zen-hpcg-eula.html?filename=amd-zen-hpcg-2022_11.tar.gz.

For example, you can run the optimized HPCG binary on a 2P AMD EPYC 9654 node using the following script:

```
#! /bin/bash
module load hpcx/aocc40/2.13.0
export OMPI_MCA_pml=ucx
export OMPI_MCA_osc=ucx
export OMPI_MCA_spml=ucx
export OMPI_MCA_btl=^vader,tcp,openib,uct
export OMPI_MCA_coll_hcoll_enable=1
export OMPI_MCA_hwloc_base_binding_policy=none
 ldd xhcg
which mpicc
NT=96
NR=2
mpirun -np $NT --bind-to core --map-by ppr:4:l3cache:pe=$NR -x OMP_NUM_THREADS=$NR -x
OMP_PROC_BIND=true -x OMP_PLACES=cores ./xhpcg --nx=192 --ny=192 --nz=192 --rt=60
```

The following table summarizes HPCG results for 2P AMD EPYC 9654, 9754, and 9684X nodes using both single and dual rank DIMMs:

SKU	HPCG (FLOP/s)	
	Single Rank	Dual Rank
9654	129.2	135.0
9754	132.0	137.9
9684X	-	137.9

Table 8-12: Summarized HPCG results for various AMD EPYC SKUs

8.7 Mellanox Configuration

Mellanox uses the OFED middleware stack to operate their fabric. There is a community OFED version available from openfabrics.org*; however, AMD recommends using the OFED version that Mellanox bundles and provides free from their website.

You must use the latest OFED on clusters-based 4th Gen AMD EPYC Series processors because earlier OFED versions will not yield correct bandwidth profiles. Please also use the latest firmware on your ConnectX-6 cards and optimized xGMI and PCIe BIOS options.

8.7.1 Bandwidth Test

This example shows how to ensure that the correct bandwidth profile exists between any 2 compute nodes, such as `node-1` and `node-2`.

- **On node 1:** `taskset -c 24 ib_write_bw -d mlx5_1 -F -a`
- **On node-2:** `taskset -c 24 ib_write_bw -d mlx5_1 -F -a node-1`

This example:

- Tests the connection from `node 1` to `node 2`.
- Uses `taskset` or `numactl` to ensure you select a core that is logically closest to the Mellanox Host Channel Adapter (HCA) PCI card, or that are local to the PCI slot hosting the ConnectX-6 card, as described in "OSU Network Test" on page 44. The `hwloc-ls` command allows you to select the cores to choose for this purpose. See "Understanding hwloc-ls and hwloc-info" on page 11.
- Ensures the cores are idle in the C1 State.
- Explicitly states which port to use on the InfiniBand card via the `-d` flag.

The following results are for an HDR, 200 Gbps Mellanox InfiniBand network:

#bytes	#iteration	BW peak[MB/sec]	BW average[MB/sec]	MsgRate[Mpps]
2	5000	13.08	12.93	6.779569
4	5000	26.31	26.09	6.839758
8	5000	52.47	52.24	6.846876
16	5000	105.23	105.09	6.887147
32	5000	209.86	208.75	6.84032
64	5000	420.93	418.34	6.85401
128	5000	839.45	834.89	6.839384
256	5000	1678.9	1676.1	6.865303
512	5000	3338.68	3323.18	6.805869
1024	5000	6658.38	6620.31	6.779201
2048	5000	13204.23	13140.5	6.727936
4096	5000	20649.78	20614.98	5.277435
8192	5000	22727.27	22699.27	2.905507
16384	5000	23291.93	23287.94	1.490428
32768	5000	23452.16	23450.33	0.75041
65536	5000	23400.94	23397.69	0.374363
131072	5000	23485.2	23484.78	0.187878
262144	5000	23490.72	23490.35	0.093961
524288	5000	23487.5	23487.06	0.046974
1048576	5000	23487.96	23487.67	0.023488

Table 9: Sample Mellanox Infiniband performance

2097152	5000	23488.54	23488.38	0.011744
4194304	5000	23490.95	23490.33	0.005873
8388608	5000	23491.07	23490.67	0.002936

Table 9: Sample Mellanox Infiniband performance (Continued)

This example shows a representative profile across the different packet sizes. Very small packet sizes do not utilize available bandwidth fully, but the maximum bandwidth of 200Gbps is achieved and maintained as the packet sizes increase. Achieving these results requires the cores to be in the C1 idle state. Cores idling in the C2 state would exhibit a possible brief bandwidth maximum in the middle of the packet band, after which bandwidth would decrease with increasing packet size. Larger packet sizes give the cores time to drop into C2, which causes latency as the cores move back into C1.

8.7.2 Latency Test

You can test network latency with `ib_write_lat` in the same way **ib_write_bw** was used in "Bandwidth Test" on page 42. You should expect to see latency of about 1 microsecond on a Mellanox HDR200 network with AMD EPYC 9004 Series processors. AMD measured 1.02 microseconds with a 2-byte packet with Boost=ON. These are very useful tests because they demonstrate that there are no broken system components (cables, cards, PCI slots etc.) and that the system is configured correctly. System managers should also monitor `/var/log/messages` for any Mellanox-related warnings/errors.

#bytes	#itera-tions	t_min [usec]	t_max [usec]	t_typical [usec]	t_avg [usec]	t_stdev [usec]	99%	percentile [usec]
2	1000	0.88	2.18	0.97	0.97	0.04	1.06	2.18
4	1000	0.87	2.75	0.96	0.96	0.06	1.07	2.75
8	1000	0.88	2.46	0.97	0.97	0.06	1.05	2.46
16	1000	0.88	2.68	0.97	0.97	0.07	1.07	2.68
32	1000	0.94	2.82	0.98	0.99	0.04	1.06	2.82
64	1000	0.96	2.33	1.04	1.05	0.06	1.20	2.33
128	1000	0.99	2.33	1.08	1.09	0.07	1.23	2.33
256	1000	1.49	3.75	1.58	1.59	0.10	1.71	3.75
512	1000	1.54	3.68	1.65	1.66	0.07	1.84	3.68
1024	1000	1.62	3.44	1.71	1.71	0.06	1.81	3.44
2048	1000	1.78	2.40	1.93	1.92	0.06	2.04	2.40
4096	1000	2.10	3.53	2.19	2.20	0.08	2.31	3.53
8192	1000	2.57	3.42	2.67	2.68	0.06	2.80	3.42
16384	1000	3.10	4.41	3.23	3.24	0.07	3.36	4.41
32768	1000	4.00	5.00	4.21	4.21	0.08	4.39	5.00
65536	1000	5.35	6.45	5.50	5.50	0.06	5.65	6.45
131072	1000	7.99	9.03	8.15	8.15	0.07	8.31	9.03
262144	1000	13.29	15.21	13.46	13.46	0.09	13.61	15.21
524288	1000	23.90	24.58	24.09	24.09	0.07	24.24	24.58

Table 8-1: Sample latency test results

1048576	1000	45.12	46.97	45.62	45.59	0.14	45.84	46.97
2097152	1000	87.76	89.33	88.03	88.04	0.07	88.18	89.33
4194304	1000	173.37	175.47	173.51	173.52	0.11	173.72	175.47
8388608	1000	343.66	344.82	343.97	343.97	0.10	344.25	344.82

Table 8-1: Sample latency test results (Continued)

Mellanox also provides a pre-compiled version of OpenMPI as part of their OFED bundle. If you want to use an alternative MPI library, then AMD recommends reading how Mellanox builds OpenMPI in their latest release documentation. Pass the following flags to `./configure` if your MPI library supports them:

```
./configure --with-ucc=/home/software/ucc/1.2.0 --with-ucx=/home/software/ucx/1.14.0_mt --
enable-mpi1-compatibility --with-hcoll=/opt/mellanox/hcoll --with-knem=/opt/knem-
1.1.4.90mlnx1 --enable-mca-no-build=btl-uct --with-xpmem=/home/software/xpmem/2.3.0 --
with-hwloc=/home/software/hwloc/2.3.0 --with-pmix --with-slurm
```

8.8 OSU Network Test

The Ohio State University (OSU) Micro Benchmarks are a set of MPI, SHMEM and UPC tests that measure the performance of a wide range of parallel message exchange operations. The benchmark is available from: http://mvapich.cse.ohio-state.edu/benchmarks/*. See "Mellanox Configuration" on page 41 for the minimum correct versions of OFED and firmware when using ConnectX-6 cards (HDR 200gbs).

The `osu_latency` test measures the latency between 2 nodes at various packet sizes. The testing shown here is running on one core per node, where one node (core) is the sender and the other node (core) is the receiver. The sender sends a message with a certain data size to the receiver and waits for a reply from the receiver. The receiver receives the message from the sender and sends back a reply with the same data size. Many iterations of this ping-pong test are carried out to obtain the min, max, and average latency numbers. These tests use blocking versions of `MPI_Send` and `MPI_Recv`:

```
mpirun -x LD_LIBRARY_PATH  --allow-run-as-root -H node-1, node-2 -x
UCX_NET_DEVICES=mlx5_0:1 -x UCX_TLS=self,sm,rc_x -map-by ppr:1:node -n 2 -npernode 1
taskset -c 64 osu_latency
```

A rankfile was used to pin the thread on each system to a specific core. To test the shortest latency between the systems choose the CPU IDs listed in the rankfile such that they are in the same NUMA domain as the ConnectX-6 cards. Table 9-5 shows representative `osu_latency` test results:

Message Size	Latency (us)
0	0.96
1	0.97
2	0.97
4	0.97
8	0.98
16	0.98
32	1.09
64	1.17

Table 8-2: Representative osu_latency test results

128	1.19
256	1.48
512	1.47
1024	1.64
2048	2.14
4096	2.66
8192	3.59
16384	4.91
32768	7.03
65536	8.77
131072	13.6
262144	15.05
524288	26.05
1048576	47.37
2097152	89.92
4194304	175.51

Table 8-2: Representative osu_latency test results (Continued)

This page intentionally left blank.

Chapter 9 Additional Information

9.1 HPL and High SIMD Frequency

AMD launched 4th Gen AMD EPYC processors in November 2022, and our customers and partners have recognized its award-winning performance within and beyond the HPC domain. The High Performance Linpack benchmark (HPL) is a commonly used measure of HPC performance. HPL measures the theoretical peak FLOPS 'Rpeak' versus the observed FLOPS 'Rmax'. The HPL efficiency is thus:

```
efficiency=Rmax/Rpeak
```

where:

```
Rpeak = base frequency   x   number of cores    x   FLOPS per cycle per core
```

The HPC market efficiency has observed for some time now as typically never reaching 100%. However, 4th Gen AMD EPYC processors are now seeing efficiency > 100% for several reasons, including:

- Rmax (the observed performance) depends on the CPU SIMD frequency, whereas the Rpeak as shown above is based on the base frequency. These frequencies are separate and distinguishable from each other. What identifies them:
  - **Base Frequency:** Minimum average integer frequency at a particular Thermal Design Point (TDP). It must assume all the IO links into the CPU (PCIe lanes plus AMD Infinity Fabric) are fully loaded
  - **SIMD frequency:** Frequency at which the CPU operates when executing a FLOPS orientated kernel such as DGEMM/HPL, etc. Consider it the natural operating speed of the CPU for such a workload. Also, in a HPC setting many, of the IO links are often unused.
- AMD defines a TDP and a configurable TDP 'cTDP' for each CPU. For example, the TDP for the 96-core AMD EPYC 9654 is 360W, but the cTDP is 320-400W. Users will often tune the cTDP to its maximum 400W in order to achieve the highest performance. In conjunction with setting BOOST=ON and DETERMISM=POWER in the BIOS, this in turn allows the System Management Unit (SMU) to use the extra power budget. In turn, this can translate into a higher effective SIMD frequency depending on the workload (applies for HPL and DGEMM), i.e. higher than base frequency. The way in which EPYC manages power in this way (redeploying power and boosting the frequency should thermal and electrical design limits allow) is a key differentiator for EPYC CPUs.
- 4th Gen AMD EPYC processors support AVX512 instructions but the data path is 256bits wide rather than a native 512bits. Thus, loading an AVX512 vector takes 2 CPU cycles (it is 'double pumped'). In taking this approach:
  - The CPU saves power, which in turn can be reassigned by the SMU and increase the operating frequency.
  - The CPU leverages 2 previous generations of AMD EPYC CPUs where we have optimized power over a 256bit data path.

- Finally, there are generational differences when comparing 4th Gen AMD EPYC processors (SP5) to previous EPYC generations (SP3):
  - 4th gen AMD EPYC processors support DDR5 and PCIe Gen5. These consume power and suppress the base frequency when in definition.
  - 4th Gen AMD EPYC (SP5) processors use a 5nm process, which is smaller than the 7nm process used for 2nd and 3rd Gen AMD EPYC processors. This delivers significant advantages and power gains and enables a dramatically increased core count, plus extra thermal and electrical headroom that the SMU can reuse and reassign to boost the frequency further.

In general, 4th Gen AMD EPYC processors include a number of design changes and improvements that come together with the net effect of providing a SIMD frequency gain compared to previous AMD EPYC generations. AMD therefore strongly advises that customers considering HPL in their next procurement use the expected observable performance (Rmax) in their considerations and considering the theoretical peak (Rpeak) because this can be misleading on expected delivered performance.

9.2 Reference System

The following system configuration was used for the testing examples included in this document:

Setting	Value
Platform	AMD reference system 'Quartz', water-cooled
BIOS	Agesa 1007
CPU	AMD EPYC 9654, AMD EPYC 9684X, AMD EPYC 9754
Base Clock	2.4 GHz (9654), 2.25 GHz (9764), 2.55 GHz (9684X)
NUMA	4
System Profile	Power
SMT	Off
Boost	On
C2	Disabled
CPU Governor	Performance
OS	Rocky Linux 8.7
Kernel	4.18.0-425.13.1.el8_7.x86
HCA	Mellanox ConnectX-6
Speed	HDR200
OFED	MLNX_OFED_LINUX-5.90
HCA Firmware	20.32.1010

Table 9-1: Reference system configuration

9.3 AMD Resources

- AMD EPYC Solution Briefs and Whitepapers (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 Optimizing CPU Compiler
- AMD Optimizing CPU Libraries
- AMD uProf code profiler
- 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
- *High Performance Toolchain: Compilers, Libraries & Profilers Tuning Guide for AMD EPYC™ 9004 Series Processors*, available from AMD EPYC™ Server Performance Tuning Guides.

9.4 EDA Configuration

If you are running EDA workloads, then use the following BIOS and OS settings for testing:

- **BIOS:**
  - **SMT:** Disabled. EDA applications are not sufficiently threaded. Disabling SMT obtains maximum performance from every thread.
  - **Determinism:** Set Power Determinism mode to allow energy-efficient samples to run at their maximum possible speeds within TDP.
  - **NPS (NUMA Per Socket):** Set either NPS=2 or NPS=4 for the highest memory bandwidth and lowest latency.

Note: AMD EPYC 9004 Series Processors without AMD 3D V-Cache technology perform better with NPS=2 versus NPS=4 based on AMD testing. However, if you are running a mix of workloads in addition to RTL simulations, then use NPS=1 for consistent performance.

  - **Infinity Fabric:** P States=Disabled.
- **OS:** Use the following OS and TuneD settings:
  - **Minimum OS version:** Your OS must meet the minimum requirements listed at https://www.amd.com/en/processors/epyc-minimum-operating-system.
  - **TuneD:** Install the OS, then set the TuneD profile to `throughput-performance`:
    `# tuned-adm profile: Use throughput-performance`

9.5 Other Resources

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

Chapter 10 Processor Identification

Figure 10-1 shows the processor naming convention for AMD EPYC 9004 Series Processors and how to use this convention to identify particular processors models:

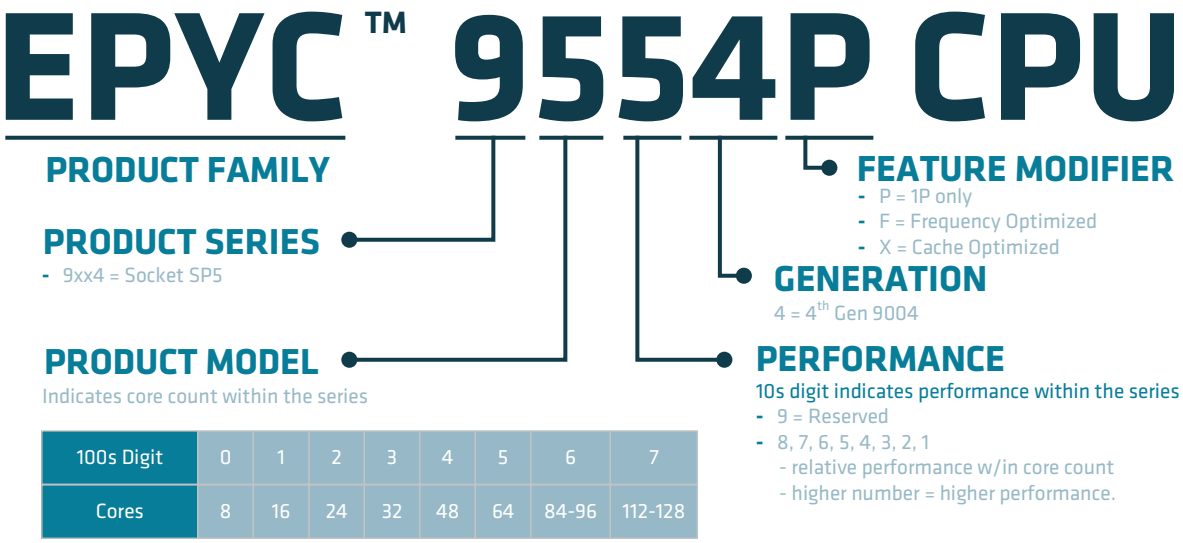

Figure 10-1: AMD EPYC SoC naming convention

10.1 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 Model 10h corresponds to an "A" part "Zen 4" CPU.
  - **91xx-96xx (including "X" OPNs):** Family 19h 10-1F
  - **97xx:** Family 19h 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.

10.2 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*.

10.2.1 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.