TUNING GUIDE
AMD EPYC 7003

High Performance Computing (HPC)

Publication	57091
Revision	4.0
Issue Date	Mar, 2022

Date	Version	Changes
Mar, 2021	2.0	Initial public release
Sep, 2021	3.0	Added AMD μProf profiling tool, Linux perf, and HSMP driver information
Mar, 2022	4.0	Added AMD 3D V-Cache™ information

Audience

This document helps you tune systems powered by 3rd Gen AMD EPYC™ processors for High Performance Computing (HPC) workloads but is not an all-inclusive guide. Further, some items referred herein to may have different names across various OEM systems, such as OEM-specific BIOS settings. Every HPC workload has unique performance characteristics. This document suggests additional items to initially 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.
- Be familiar with the OS-specific configuration, monitoring, and troubleshooting tools.

Note: All of the settings described in this Tuning Guide apply to all AMD EPYC 7003 Series Processors with or without AMD 3D V-Cache™ except where explicitly noted otherwise.

Table of Contents

Chapter 1 AMD EPYC™ 7003 Series Processors ---- 1

1.1 Operating Systems ... 1
1.2 Microarchitecture Overview ... 2
1.2.1 "Zen 3" Core ... 2
1.2.2 Core Complex (CCX) and Core Complex Die (CCD) ... 2
1.2.3 AMD 3D V-Cache Technology ... 3
1.2.4 I/O Die (Infinity Fabric™) ... 3
1.2.5 Memory and I/O ... 4
1.2.6 NUMA Topology ... 4
1.2.7 Dual-Socket Configurations ... 6
1.2.8 Understanding hwloc-ls and hwloc-info ... 7
1.2.9 C-States ... 9
1.2.10 P-States, Frequencies, and Boosting ... 10
1.2.11 CPU Governors ... 12
1.2.12 Useful 'cpupower' Command Examples ... 12

Chapter 2 Quick HPC Setup ---- 13

2.1 BIOS Settings ... 13
2.2 OS Settings ... 14
2.3 Basic System Checks ... 14
2.3.1 Identifying a 7003X or 7003 System ... 15
2.4 Useful Commands ... 15
2.4.1 Build CPU ID lists with 'seq' ... 15
2.4.2 Core Pinning and Memory Locality: ... 15
2.4.3 Faster 'make' with 'make -j' ... 16

Chapter 3 BIOS Settings ---- 17

3.1 Recommended BIOS Settings for Bare Metal Workloads ... 17
3.1.1 Determinism (Performance | Power) ... 17
3.1.2 cTDP (configurable Thermal Design Point) ... 17
3.1.3 PPL (Package Power Limit) ... 17
3.1.4 Simultaneous Multi-Threading (SMT) Enabled | Disabled ... 17
3.1.5 X2APIC = Auto ... 18
3.1.6 Algorithmic Performance Boost Disable (APBDIS) 1 | 0 ... 18
3.1.7 Core Performance Boost = OFF | ON ... 18
3.1.8 Memory speed = AUTO ... 18
3.1.9 Core C-States = Enabled ... 18
3.1.10 Data Fabric C-States (DF C-States) Enabled | Disabled ... 18
3.1.11 NPS = 1 | 2 |4 ... 18
3.1.12 Preferred-I/O Control ... 19
3.1.13 CCD Control ... 19

3.1.14 Down-Coring / Core Control 19
3.1.15 Transparent Secure Memory Encryption (TSME) = OFF 19
3.1.16 3D V-Cache = Auto 20

Chapter 4 Operating Systems 21

4.1 Linux Kernel and ISV OS Support Considerations 21
4.2 /proc and /sys 21
4.2.1 /proc/sys/vm/zone_reclaim_mode 21
4.2.2 /proc/sys/vm/drop_caches 22
4.3 Transparent Huge Pages (THP) 22
4.4 Explicit Hugepages 22
4.5 Randomize_va_space 23
4.6 NUMA Balancing 23
4.7 Spectre and Meltdown 23

Chapter 5 Compilers, Libraries, and Profilers 25

5.1 AMD Optimizing CPU Compiler (AOCC) 25
5.1.1 AOCC Clang/Clang++ 25
5.1.2 AOCC FLang 26
5.1.3 Clang and Flang Options 26
5.2 GCC Compiler 27
5.3 AMD Optimizing CPU Libraries (AOCL) 28
5.3.1 BLIS 28
5.3.2 libFLAME 28
5.3.3 FFTW 29
5.3.4 LibM 29
5.3.5 ScaLAPACK 29
5.3.6 AMD Random Number Generator 29
5.3.7 AMD Secure RNG 29
5.3.8 AOCL-Sparse 29
5.4 AOCL Tuning Guidelines 30
5.4.1 AOCL Dynamic 30
5.4.2 BLIS DGEMM Multi-thread Tuning 30
5.4.3 Performance Suggestions for Skinny Matrices 30
5.4.4 AMD-Optimized FFTW 31
5.5 AMD μProf – Profiling Tool 32
5.5.1 System Analysis Utility (AMDuProfPcm) 33
5.5.2 Application Analysis 34
5.5.3 HPC Analysis 38
5.5.4 MPI Analysis 39
5.5.5 MPI Trace Analysis 40
5.5.6 Power Profiling 41
5.5.6.1 Profiling Effective Frequency 41
5.5.6.2 Live System-Wide Power Profile 42
5.6 Spack 43

Chapter 6 Linux Perf ---------- 45

Chapter 7 Host System Management Port ---------- 47

7.1 HSMP Driver 47

Chapter 8 Executing Applications on AMD EPYC 7003 Processors ---------- 49

8.1 Characterization Strategy 49
8.2 Pinning Strategies and Hybrid Codes 50

Chapter 9 Synthetics ---------- 53

9.1 Stream 53
9.2 HPL 55
9.3 DGEMM 57
9.4 Mellanox Configuration 58
9.4.1 Bandwidth Test 58
9.4.2 Latency Test 59
9.5 OSU Network Tests 60

Chapter 10 Additional Information ---------- 61

10.1 Reference System 61
10.2 EDA Configuration 62
10.3 AMD Resources 63
10.4 Other Resources 63

This page intentionally left blank.

Chapter 1 AMD EPYC™ 7003 Series Processors

3rd Gen AMD EPYC 7003 Series processors are socket compatible with previous 2nd Gen EPYC 7002 Series processors and may be supported in existing EPYC CPU-based system, depending on the specific platform. 3rd Gen AMD EPYC processors complement AMD's existing server portfolio by offering further improvements to performance and value in a number of configurations with varying core counts, thermal design points, frequencies, and other features. See Table 1-1.

Component	Detail
Socket	SP3
Max Number of Cores	64
Core Process Technology	7 nm
Max Memory Speed	3200 MT/s
Max Memory Channels	8 per socket
Max Memory Capacity	4TB per socket
I/O Interconnect	128 lanes (max) PCIe® Gen4

Table 1-1: General AMD EPYC 7003™ processor specification

Some AMD EPYC™ 7003 Series Processors introduce AMD's new 3D V-Cache die stacking technology that enables denser, more efficient chiplet integration. AMD 3D Chiplet architecture stacks L3 cache tiles vertically to provide 768 MB of L3 cache per socket up with to 96MB of L3 cache per CCD, while still providing socket compatibility with existing AMD EPYC 7003 Series Processors. Applications that take advantage of AMD 3D V-cache can see significant performance gain and lower overall TCO..

See *Overview of AMD EPYC™ 7003 Series Processors Microarchitecture* (available from AMD EPYC Tuning Guides) to learn more about the AMD EPYC 7003 Series Processor microarchitecture.

1.1 Operating Systems

Please see AMD EPYC™ 7003 Series Processors Minimum Operating System (OS) Versions for detailed OS requirements.

1.2 Microarchitecture Overview

The AMD EPYC 7003 Series System on Chip (SoC) Processor incorporates compute cores, memory controllers, I/O controllers, RAS, and security features. The AMD EPYC 7003 Series Processor retains the proven Multi-Chip Module (MCM) Chiplet architecture of prior successful AMD EPYC server-class processors while making further improvements and upgrading the compute units to new "Zen 3" cores. Certain AMD EPYC 7003 Series Processors introduce die stacking technology that boosts L3 cache capacity.

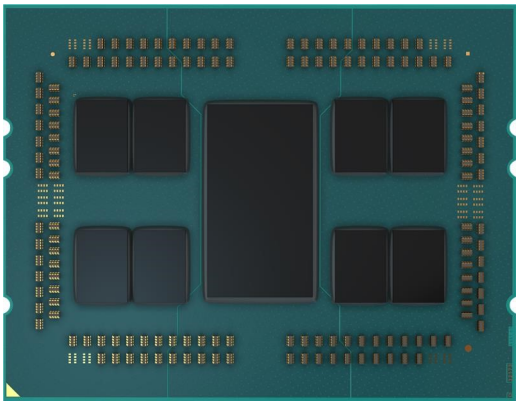


Figure 1-1: AMD EPYC 7003 configuration with 8 Core Complex Dies (CCD) and Central I/O Die (IOD)

1.2.1 "Zen 3" Core

The AMD EPYC 7003 Series Processor is based on new "Zen 3" compute cores. The "Zen 3" core is manufactured using a 7nm process and designed to provide an Instructions per Cycle (IPC) uplift over prior generation "Zen" cores. Each core supports Simultaneous Multi-threading (SMT) that allows 2 threads per core to run simultaneously when enabled. Each core includes an optimized 32KB L1 cache and private 512 KB Unified (Instruction/Data) L2 cache. All caches use a 64B cache line size.

1.2.2 Core Complex (CCX) and Core Complex Die (CCD)

Figure 1-2 shows how up to eight "Zen 3" core compute units share a L3 or Last Level Cache (LLC). This grouping is referred to as a Core Complex (CCX). With Simultaneous Multithreading (SMT) on each core, a single CCX will support up to 16 concurrent hardware threads, up to 4MB of L2 cache, and up to 32MB of base L3 cache and up to 96MB L3 on OPNs with 3D V-Cache. AMD EPYC 7003 Series Processors contain each CCX within a single die called a Core Complex Die (CCD).

Figure 1-2: Eight Compute Cores comprise a Core Complex (CCX) within a single die or CCD

1.2.3 AMD 3D V-Cache Technology

Certain 3rd Gen AMD EPYC processors introduce 3D V-Cache die stacking technology that enables denser, more efficient chiplet integration. AMD 3D Chiplet architecture stacks L3 cache tiles vertically to provide up to 96MB of L3 cache per CCD and up to 768 MB L3 Cache per socket, while still providing socket compatibility with existing AMD EPYC™ 7003 Series Processors.

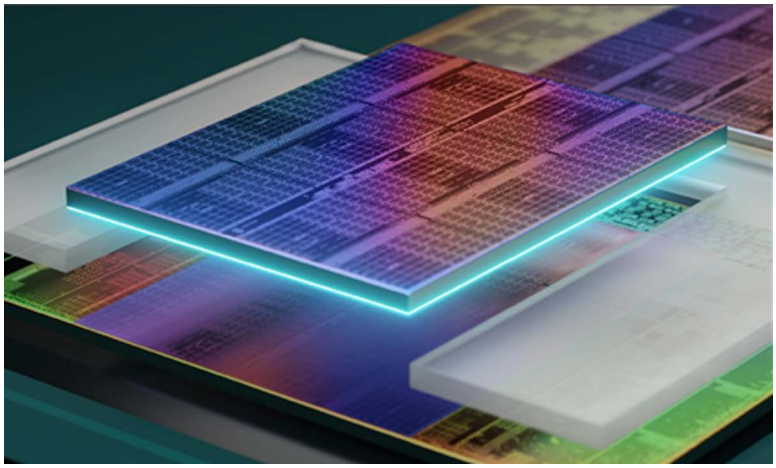


Figure 1-3: L3 Chiplet vertically stacked on the central L3 portion of base CCD.

1.2.4 I/O Die (Infinity Fabric™)

The CCDs connect to memory, I/O, and each other through the I/O Die (IOD). Each CCD connects to the IOD via a dedicated high-speed Global Memory Interconnect (GMI) link. The IOD also contains memory channels, PCIe® Gen4 lanes, and Infinity Fabric links. All dies (or chiplets) interconnect with each other via AMD's Infinity Fabric Technology. 3rd Gen AMD EPYC processors reduce memory latency by running the fabric clock (FCLK) at speeds up to 1600Mhz to couple with DDR4-3200 Memory DIMMs that also run at 1600MHz (MEMCLK).

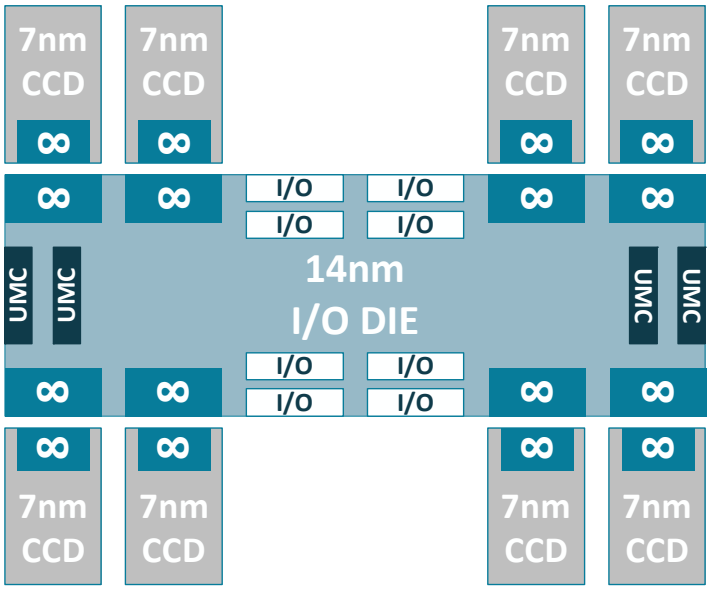


Figure 1-4: AMD EPYC 7003 Series Processor internal CCD, I/O, and DDR memory interface topology with AMD Infinity Fabric

1.2.5 Memory and I/O

3rd Gen AMD EPYC Series Processors bring additional performance capability and a new 6-way interleave mode to the memory subsystem. Each AMD EPYC 7003 Series Processor has 8 Universal Memory Controllers (UMC). Each UMC (or memory channel) can support up to 2 DIMMs per channel (DPC) for a maximum of sixteen DIMMs per socket, and a single 3rd Gen AMD EPYC processor can support 4TB of DDR4 memory. The IOD supports 4, 6, and 0 memory-channel configurations. 8 memory channels are most common and generally provide the best performance.

Each processor has eight x16-bit I/O links that provide the PCIe subsystem up to 128 lanes of high-speed PCIe Gen4 I/O for single-socket platforms.

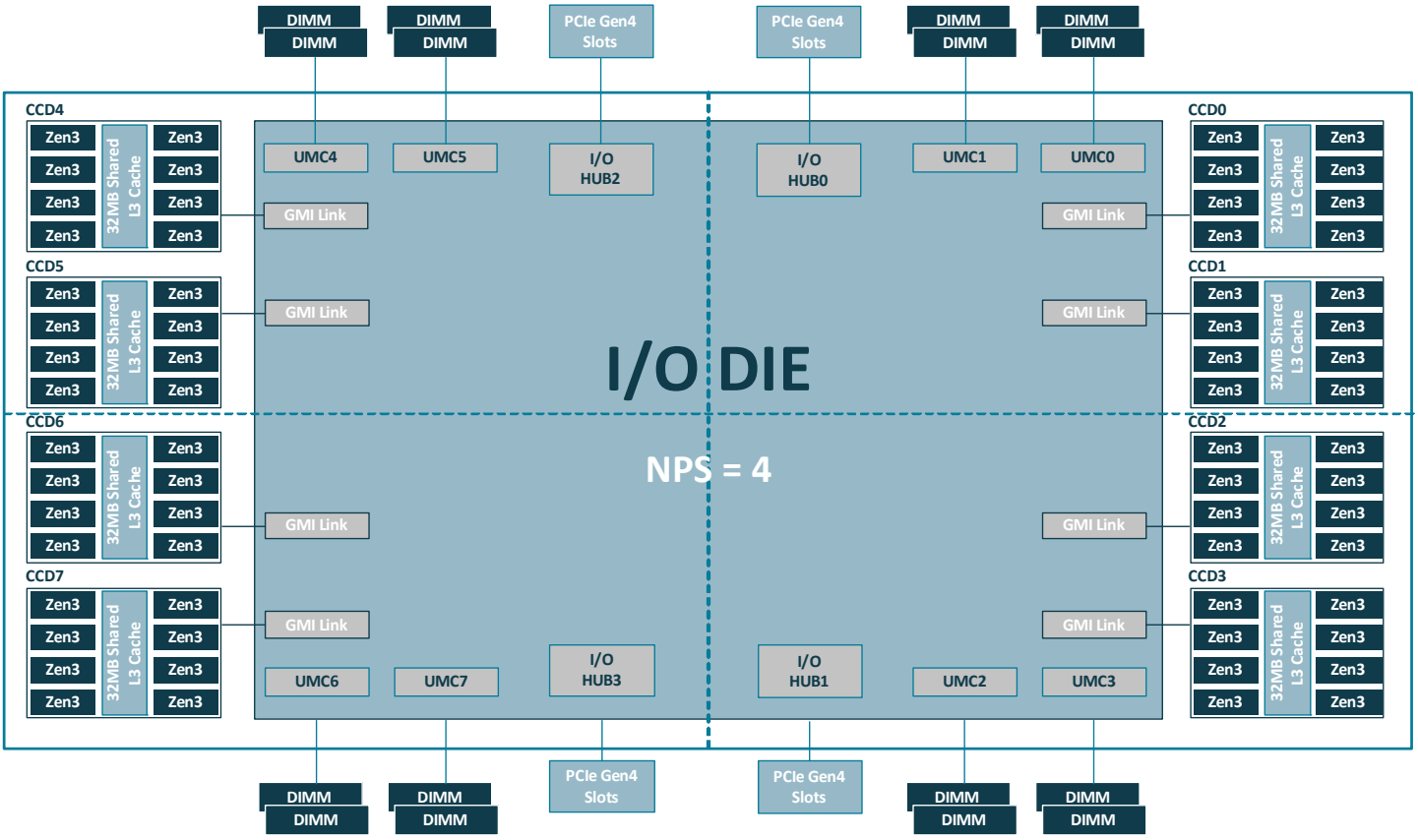


Figure 1-5: EPYC 7003 System on Chip (SoC): 8 CCDs and central IOD

1.2.6 NUMA Topology

AMD EPYC 7003 Series processors use a Non-Uniform Memory Access (NUMA) microarchitecture. The NUMA Nodes per Socket BIOS setting allows you to n optimize this NUMA topology for specific operating environments and workloads.

Setting NPS=4 as shown in Figure 1-5 divides the processor into quadrant, where each quadrant has 2 CCDs, 2 UMCs, and 1 I/O hub. The closest processor- memory-I/O distance is between the cores, memory controllers, and I/O within the same quadrant, and the farthest distance is between a core and memory controller or I/O hub in diagonal quadrants. Core, memory, and I/O locality is an important aspect of performance tuning in a NUMA-based system.

The NPS setting also controls the interleave pattern of the memory controllers because all channels within each NUMA node are interleaved. Setting more granular NPS setting decreases the number of interleaved channels. A single AMD EPYC 7003 Series Processor may support configurations ranging from a single NUMA node to 8 NUMA nodes, as follows:

- **NPS=1:** NPS=1 indicates a single NUMA node per socket (or CPU). This setting configures all memory channels on the processor into a single NUMA domain where all CPU cores and all memory and PCIe devices connected to the processor are in one NUMA domain. Memory is then interleaved across all eight memory channels on the processor.

- **NPS=2:** NPS=2 partitions the processor into two NUMA domains that group 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 four memory channels in each NUMA domain.

Note: Only NPS=1 or NPS=2 are recommended for AMD EPYC 7003 processors that have only 6 CCDs per socket.

- **NPS=4:** This is the most common setting on HPC systems. NPS=4 partitions the processor into four NUMA domains that configures each logical CPU quadrant as its own NUMA domain. Memory is interleaved across the two memory channels in each quadrant. PCIe devices will be local to one of the four NUMA domains on the processor depending on the IOD quadrant that has the PCIe root for that device.

Note: All AMD EPYC 7003 Series Processors with AMD 3D V-Cache technology include 8 CCDs.

- **NPS=0:** NPS=0 interleaves memory access all memory channels on a 2-socket system. This configuration is not recommended because it adds inter-socket latency to every memory access and should therefore be avoided.

The following additional NUMA settings are also available:

- **L3 Cache as NUMA Domain:** The **L3 Cache as NUMA** (L3CAN) BIOS setting exposes each L3 cache (1 per CCD) as its own NUMA node. For example, a single processor with 8 CCDs would have 8 NUMA nodes, 1 for each CCD. In this example, a two-socket system would have a total of 16 NUMA nodes.

- **NUMA Per Socket (NPS) and Memory Bandwidth:** The fastest memory bandwidth as measured by the STREAM memory benchmark can be achieved using NPS=4, with relatively minor differences for different NPS settings. However, real-world workloads have different memory access patterns, so your specific performance may vary.

Users and developers need to be aware of CPU-IDs within a NUMA domain and ensure they are pinning to cores appropriately, especially when not using all the cores on the socket. See "Understanding hwloc-ls and hwloc-info" on page 7.

1.2.7 Dual-Socket Configurations

AMD EPYC 7003 Series Processors support single- or dual-socket system configurations, except that processors with a 'P' suffix in their name are designed for single-socket configurations. Both processors in a dual-socket configuration must be identical. Two different processor OPNs (or *steppings*) cannot be used in the same dual-socket system.

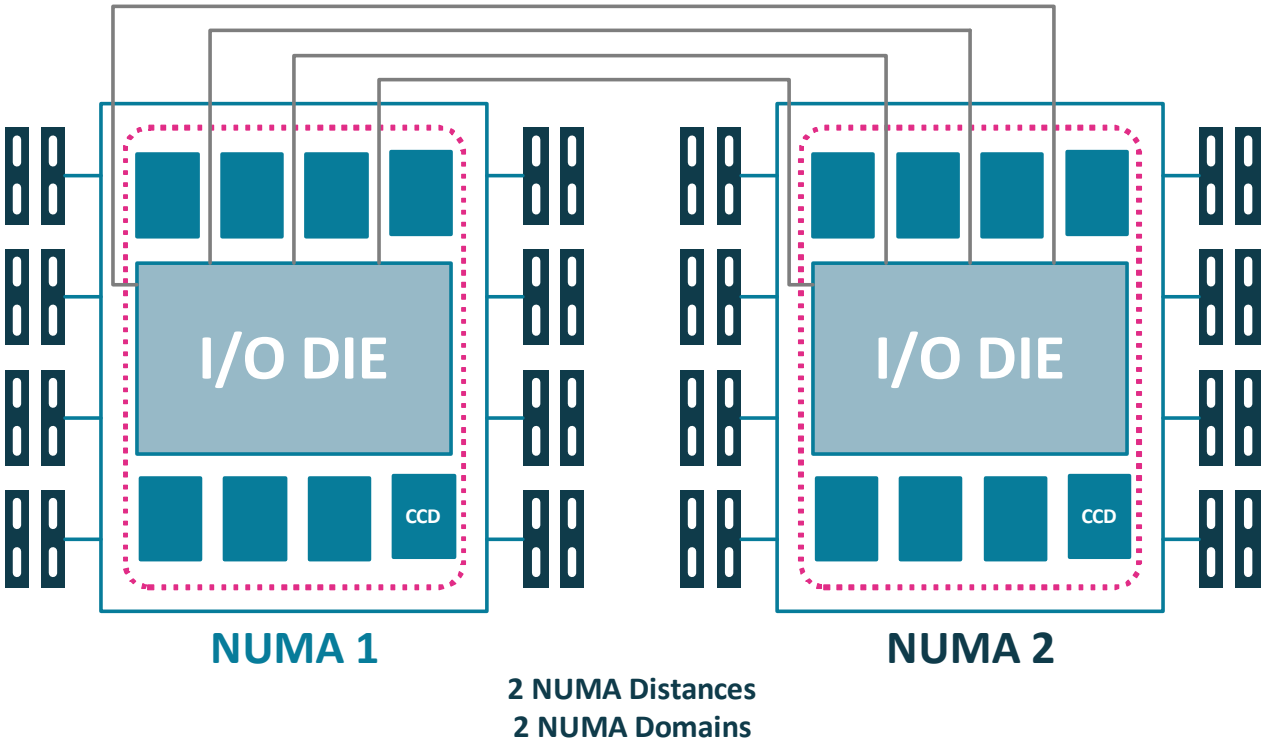


Figure 1-6: Two EPYC 7003 Processors connect through 4 xGMI links (NPS1).

1.2.8 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 64 cores per socket configured with NPS=4 and SMT disabled.

```
  Group0 L#2
    NUMANode L#2 (P#2 63GB)
    L3 L#4 (32MB)
      L2 L#32 (512KB) + L1d L#32 (32KB) + L1i L#32 (32KB) + Core L#32 + PU L#32 (P#32)
      L2 L#33 (512KB) + L1d L#33 (32KB) + L1i L#33 (32KB) + Core L#33 + PU L#33 (P#33)
      L2 L#34 (512KB) + L1d L#34 (32KB) + L1i L#34 (32KB) + Core L#34 + PU L#34 (P#34)
      L2 L#35 (512KB) + L1d L#35 (32KB) + L1i L#35 (32KB) + Core L#35 + PU L#35 (P#35)
      L2 L#36 (512KB) + L1d L#36 (32KB) + L1i L#36 (32KB) + Core L#36 + PU L#36 (P#36)
      L2 L#37 (512KB) + L1d L#37 (32KB) + L1i L#37 (32KB) + Core L#37 + PU L#37 (P#37)
      L2 L#38 (512KB) + L1d L#38 (32KB) + L1i L#38 (32KB) + Core L#38 + PU L#38 (P#38)
      L2 L#39 (512KB) + L1d L#39 (32KB) + L1i L#39 (32KB) + Core L#39 + PU L#39 (P#39)
    L3 L#5 (32MB)
      L2 L#40 (512KB) + L1d L#40 (32KB) + L1i L#40 (32KB) + Core L#40 + PU L#40 (P#40)
      L2 L#41 (512KB) + L1d L#41 (32KB) + L1i L#41 (32KB) + Core L#41 + PU L#41 (P#41)
      L2 L#42 (512KB) + L1d L#42 (32KB) + L1i L#42 (32KB) + Core L#42 + PU L#42 (P#42)
      L2 L#43 (512KB) + L1d L#43 (32KB) + L1i L#43 (32KB) + Core L#43 + PU L#43 (P#43)
      L2 L#44 (512KB) + L1d L#44 (32KB) + L1i L#44 (32KB) + Core L#44 + PU L#44 (P#44)
      L2 L#45 (512KB) + L1d L#45 (32KB) + L1i L#45 (32KB) + Core L#45 + PU L#45 (P#45)
      L2 L#46 (512KB) + L1d L#46 (32KB) + L1i L#46 (32KB) + Core L#46 + PU L#46 (P#46)
      L2 L#47 (512KB) + L1d L#47 (32KB) + L1i L#47 (32KB) + Core L#47 + PU L#47 (P#47)
    HostBridge
      PCIBridge
        PCI 21:00.0 (InfiniBand)
          Net "ib0"
          OpenFabrics "mlx5_0"
        PCI 21:00.1 (Ethernet)
          Net "enp33s0f1"
          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 2 CCDs (2x 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 is from an AMD EYPC 7003 system with AMD 3D V-Cache technology, with NPS=4 and SMT=disabled. WIth AMD 3D V-Cache technology, the AMD EPYC 7003 cache to CPU topology is identical to the previous example of the AMD EYPC 7003 system, with the key difference being that hwloc-ls reports 96MB of L3 cache per CCD instead of the 32MB of L3 cache found in AMD 7003 Series Processors without AMD V-Cache technology. There are no other differences from a topology standpoint.

```
Group0 L#5
  NUMANode L#5 (P#5 126GB)
  L3 L#10 (96MB)
    L2 L#80 (512KB) + L1d L#80 (32KB) + L1i L#80 (32KB) + Core L#80 + PU L#80 (P#80)
    L2 L#81 (512KB) + L1d L#81 (32KB) + L1i L#81 (32KB) + Core L#81 + PU L#81 (P#81)
    L2 L#82 (512KB) + L1d L#82 (32KB) + L1i L#82 (32KB) + Core L#82 + PU L#82 (P#82)
    L2 L#83 (512KB) + L1d L#83 (32KB) + L1i L#83 (32KB) + Core L#83 + PU L#83 (P#83)
    L2 L#84 (512KB) + L1d L#84 (32KB) + L1i L#84 (32KB) + Core L#84 + PU L#84 (P#84)
    L2 L#85 (512KB) + L1d L#85 (32KB) + L1i L#85 (32KB) + Core L#85 + PU L#85 (P#85)
    L2 L#86 (512KB) + L1d L#86 (32KB) + L1i L#86 (32KB) + Core L#86 + PU L#86 (P#86)
    L2 L#87 (512KB) + L1d L#87 (32KB) + L1i L#87 (32KB) + Core L#87 + PU L#87 (P#87)
  L3 L#11 (96MB)
    L2 L#88 (512KB) + L1d L#88 (32KB) + L1i L#88 (32KB) + Core L#88 + PU L#88 (P#88)
    L2 L#89 (512KB) + L1d L#89 (32KB) + L1i L#89 (32KB) + Core L#89 + PU L#89 (P#89)
    L2 L#90 (512KB) + L1d L#90 (32KB) + L1i L#90 (32KB) + Core L#90 + PU L#90 (P#90)
    L2 L#91 (512KB) + L1d L#91 (32KB) + L1i L#91 (32KB) + Core L#91 + PU L#91 (P#91)
    L2 L#92 (512KB) + L1d L#92 (32KB) + L1i L#92 (32KB) + Core L#92 + PU L#92 (P#92)
    L2 L#93 (512KB) + L1d L#93 (32KB) + L1i L#93 (32KB) + Core L#93 + PU L#93 (P#93)
    L2 L#94 (512KB) + L1d L#94 (32KB) + L1i L#94 (32KB) + Core L#94 + PU L#94 (P#94)
    L2 L#95 (512KB) + L1d L#95 (32KB) + L1i L#95 (32KB) + Core L#95 + PU L#95 (P#95)
  HostBridge
    PCIBridge
      PCI c1:00.0 (InfiniBand)
        Net "ib0"
        OpenFabrics "mlx5_2"
```

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

```
Machine (1024GB total)
  Package L#0 (P#0 128GB)
    L3 L#0 (32MB)
      L2 L#0 (512KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
        PU L#0 (P#0)
        PU L#1 (P#128)
      L2 L#1 (512KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1
        PU L#2 (P#1)
        PU L#3 (P#129)
      L2 L#2 (512KB) + L1d L#2 (32KB) + L1i L#2 (32KB) + Core L#2
        PU L#4 (P#2)
        PU L#5 (P#130)
      L2 L#3 (512KB) + L1d L#3 (32KB) + L1i L#3 (32KB) + Core L#3
        PU L#6 (P#3)
        PU L#7 (P#131)
      L2 L#4 (512KB) + L1d L#4 (32KB) + L1i L#4 (32KB) + Core L#4
        PU L#8 (P#4)
        PU L#9 (P#132)
      L2 L#5 (512KB) + L1d L#5 (32KB) + L1i L#5 (32KB) + Core L#5
        PU L#10 (P#5)
        PU L#11 (P#133)
      L2 L#6 (512KB) + L1d L#6 (32KB) + L1i L#6 (32KB) + Core L#6
        PU L#12 (P#6)
        PU L#13 (P#134)
      L2 L#7 (512KB) + L1d L#7 (32KB) + L1i L#7 (32KB) + Core L#7
        PU L#14 (P#7)
```

```
        PU L#15 (P#135)
```

On a dual-socket system with 2x 64 core CPUs with SMT enabled, the first thread on each core is typically within the range 0-127, while the second thread will have CPU IDs within the range 128-255. 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 128 attached to it.

```
depth 0:          1 Machine (type #1)
 depth 1:         2 Package (type #3)
  depth 2:        8 NUMANode (type #2)
   depth 3:       16 L3Cache (type #4)
    depth 4:      128 L2Cache (type #4)
     depth 5:     128 L1dCache (type #4)
      depth 6:    128 L1iCache (type #4)
       depth 7:   128 Core (type #5)
        depth 8:         128 PU (type #6)
Special depth -3:        11 Bridge (type #9)
Special depth -4:        6 PCI Device (type #10)
Special depth -5:        9 OS Device (type #11)
```

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.

1.2.9 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
- **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:

```
# cpuppower monitor
               |Mperf               || Idle_Stats
PKG |CORE|CPU | C0   | Cx    | Freq || POLL | C1    | C2
   0|   0|   0|  0.03| 99.97|  1937||  0.00|  0.00| 99.97
   0|   0|   8|  0.22| 99.78|  1973||  0.00|  0.00| 99.80
   0|   0|  16|  0.01| 99.99|  1935||  0.00|  0.00| 100.0
   0|   0|  24|  0.00|100.00|  1703||  0.00|  0.00| 100.0
   0|   0|  64|  0.00|100.00|  1854||  0.00|  0.00| 100.0
   0|   0|  72|  0.00|100.00|  1649||  0.00|  0.00| 100.0
   0|   0|  80|  0.00|100.00|  1694||  0.00|  0.00| 100.0
   0|   0|  88|  0.00|100.00|  1712||  0.00|  0.00| 100.0
   0|   0|   1|  0.01| 99.99|  1824||  0.00|  0.00| 100.0
   0|   0|   9|  0.01| 99.99|  1720||  0.00|  0.00| 100.0
   0|   0|  17|  0.00|100.00|  1758||  0.00|  0.00| 100.0
...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`.

```
# cpuppower -c 0-11 monitor
PKG |CORE|CPU | C0    | Cx    | Freq || POLL | C1    | C2
   0|   0|   0|  0.01| 99.99|  1996||  0.00| 99.99|  0.00
   0|   0|   8|  0.20| 99.80|  1938||  0.00|  0.00| 99.75
   0|   1|   1|  0.00|100.00|  1896||  0.00| 99.96|  0.00
   0|   1|   9|  0.00|100.00|  1675||  0.00|  0.00| 99.95
   0|   2|   2|  0.00|100.00|  1856||  0.00| 99.96|  0.00
   0|   2|  10|  0.00|100.00|  1658||  0.00|  0.00| 99.97
   0|   3|   3|  0.00|100.00|  1872||  0.00| 99.96|  0.00
   0|   3|  11|  0.00|100.00|  1680||  0.00|  0.00| 99.97
   0|   4|   4|  0.00|100.00|  1690||  0.00|  0.00| 99.96
   0|   5|   5|  0.00|100.00|  1665||  0.00|  0.00| 99.96
   0|   6|   6|  0.00|100.00|  1661||  0.00|  0.00| 99.96
```

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.

1.2.10 P-States, Frequencies, and Boosting

P-States are execution power states that manage power usage while the cores are active, like how C-States manage the power levels when the cores are idle. Higher P-states are only achievable when the processor 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 its various P-States based on its utilization to balance performance and power usage. A fully-utilized core will target P0, which is its quoted base frequency. The root user can observe these P-States by executing the command `cpupower frequency-info`.

RHEL versions prior to RHEL v8.5 do not show P-States frequencies correctly. RHEL v8.5 kernel version 4.18.0-305.7.1.el8_4.x86_64 fixes a problem with previous versions that did not show P-States correctly. The following example shows the output for an AMD EPYC 7443 24-Core processor with the three P-States at frequencies of 1.5GHz (P2), 2.1GHz (P1) and 2.85GHz (P0):

```
analyzing CPU 0:
  driver: acpi-cpufreq
  CPUs which can 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 limts: 1.50 GHz - 2.85 GHz
  available frequency steps: 2.85 GHz, 2.10 GHz, 1.50 GHz
  available cpufreq governors: conservative ondemand userspace powersave performance
  current policy: frequency shuld be within 1.50 GHHz and 2.45 GHz.
                  The governor "performance" may decide which speed to use
                  within this range.
  current CPU frequency: 2.85 GHz (asserted by call to hardware)
  boost state support:
    Supported: yes
    Active: no
    Boost States: 0
    Total States: 3
    Pstate-P0:  2850MHz
    Pstate-P2:  2100MHz
    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 7443 CPU has a quoted base frequency of 2.85 GHz. Enabling Boost allows the AMD EPYC 7443 cores to boost up to the quoted Boost frequency of 4.0 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.

You must enable Boost in the BIOS to benefit from the higher frequencies. 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 12.

1.2.11 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 7763 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.

1.2.12 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 2 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.

2.1 BIOS Settings

- **SMT:** OFF | ON (application-dependent whether it provides advantage)
- **x2APIC:** Enabled
- **Fast Short REP MOVSB:** Disabled
- **Enhanced REP MOVSB/STOSB:** Disabled
- **REP-MOV/STOS Streaming:** Disabled
- **IOMMU:** OFF

Note: If you are using dual-socket 64-core CPUs with SMT=ON (i.e., 256 threads):

- **IOMMU: ON** (with `iommu=pt` as a recommended kernel boot parameter)
- **X2APIC:** AUTO

- **APBDIS:** 1
- **Fixed SOC P-State:** P0
- **Core Performance Boost:** ON
- **Determinism Slider:** Power
- **cTDP:** 280 (set to your CPU's max cTDP setting of your processor; for example, a 7763 is 280W)
- **PPL:** 280 (set to your CPU's max cTDP setting; for example, a 7763 is 280W)
- **NPS:** 4
- **DF C-States:** Disabled

- **Preferred-IO Control:** Manual
  - **Preferred-IO Device:** <Bus number from Infiniband card: use `lspci` >
  - **Enhanced Preferred-IO Mode:** Enabled
- **Core C-states:** Enabled
- **TSME:** OFF
- **xGMI Link Width Control:** Manual
- **xGMI Force Link Width Control:** Force
- **xGMI Max Link Width Control:** Manual
- **PCIe 10 bit tag:** Enabled
- **3D V-Cache:** Auto. This setting only takes effect in systems powered by AMD EPYC 7003 Series Processors with AMD 3D V-Cache technology.

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.

2.2 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 64-core processors, use 0-127 in the command to disable C2 for all 128 cores by executing the command `cpupower -c 0-127 idle-set -d 2`. Also, set the CPU governor to performance by executing the command `cpupower frequency-set -g performance`.

2.3 Basic System Checks

Check if SMT is enabled [Thread(s)=1 implies SMT=OFF; Threads (2)=2 implies SMT=ON]: lscpu

- Check which NUMA node your Infiniband card or other peripherals are attached to: `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 DDR4-3200 Dual Rank DIMMS with one DIMM per channel (1 core per L3 on 64 core CPU) should yield approximately 350GB/s with ICC or AOCC. Use a single core per L3 (CCD) on 64-core CPUs to get maximum STREAM results.

2.3.1 Identifying AMD 3D V-Cache technology

The 3D V-Cache on an equipped AMD EYPC 7003 system can be enabled or disabled via BIOS option. Thus, simply checking the L3 cache size will not work to determine the CPU type. Instead, execute the `lscpu` command and check the following fields:

- **CPU Family:** All EYPC 7003 Series Processors have a **CPU Family** value of 25.
- **Stepping:** An AMD EYPC 7003 Series Processor with AMD 3D V-Cache technology will have a **Stepping** value of 2. An AMD EYPC 7003 Series Processor without this feature will have a **Stepping** value of 1.

This example shows an AMD EPYC 7003 Series Processor with AMD 3D V-Cache technology:

```
CPU Family:      25
Model:           1
Model name:      AMD EPYC 7763X 64-Core Processor
Stepping:        2
```

This example shows an EMD EPYC 7003 Series Processor without AMD 3D V-Cache technology.

```
CPU Family:      25
Model:           1
Model name:      AMD EPYC 7763 64-Core Processor
Stepping:        1
```

2.4 Useful Commands

This section lists some additional useful commands.

2.4.1 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 64 core CPUs, 2 cores per L3:** `seq -s, 0 2 127`
- **System with 2x 64 core CPU, 3 cores per L3 (join 2 lists, every second core + every fourth core):**
  `seq -s , 0 2 127 | tr -d '\n';  echo ,|  tr -d '\n';  echo "$(seq -s , 1 4 127)"`

2.4.2 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=`' For example:
`numactl -C 7 --membind=0 ./mybinary`

To establish which NUMA node your core belongs to, use the output from `numactl -H`:

```
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
node 0 size: 65422 MB
node 0 free: 296 MB
node 1 cpus: 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
node 1 size: 65535 MB
node 1 free: 40 MB
node 2 cpus: 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
node 2 size: 65535 MB
node 2 free: 3602 MB
```

```
node 3 cpus: 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
node 3 size: 65523 MB
node 3 free: 41 MB
node 4 cpus: 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
node 4 size: 65535 MB
node 4 free: 321 MB
```

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

2.4.3 Faster 'make' with 'make -j'

Executing the command `./configure ; make ; make install` is a common way to build code from a source tar ball. 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 64 core system with SMT=ON you now have 256 threads and could use `./configure ; make -j 256 ; make install` to allow the compiler to leverage up to 256 threads for compilation. This will dramatically reduce build times for large code sources.

Chapter 3 BIOS Settings

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

3.1 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.

3.1.1 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.

3.1.2 cTDP (configurable Thermal Design Point)

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

3.1.3 PPL (Package Power Limit)

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

3.1.4 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.

If you are using a dual-socket system with 2x64 and SMT=ON (256 threads per system), then you must also set IOMMU=ON. AMD further recommends using **iommu=pt** as a kernel boot parameter.

3.1.5 X2APIC = Auto

This option helps the operating system deal with interrupts more efficiently in high-core-count configurations. This option must be enabled if using > 255 threads, but it is only needed when you need SMT=ON on with a 64-core CPU SKU.

3.1.6 Algorithmic Performance Boost Disable (APBDIS) 1 | 0

This setting governs AMD EPYC Infinity Fabric boost behavior. AMD recommends a fixed, non-boosted frequency for HPC workloads, which corresponds to setting the APBDIS state to 1. Some OEM BIOS display a Fixed SOC P-State parameter when APBNDIS is disabled. If this parameter appears, set it to P0, which is the highest-performance memory P-State.

- **Disable APBDIS:** 1
- **Enable APBDIS:** 0

3.1.7 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).

3.1.8 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.

3.1.9 Core C-States = Enabled

Refers to CPU C-States. Leave these enabled. If required, disable C2 via the Linux command line as root, as described in <xref>.)

3.1.10 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.

3.1.11 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.

Note: Only CPUs with either 4 or 8 CCDs can use NPS=4, otherwise this setting will default back to NPS-1. All AMD EPYC 7003 Series Processors with AMD 3D V-Cache technology include 8 CCDs.

3.1.12 Preferred-I/O Control

This needs to be enabled on systems with a single Mellanox PCI card when using a high-performance, low-latency interconnect such as Mellanox's Infiniband fabric in order to:

- Provide enhanced priority to a single PCI device [in BIOS: Preferred-IO Device]
- Increase the PCI clock 'LCLK' [in BIOS: Enhanced Preferred-IO Mode]

The OEM BIOS will ask for either the PCI device or PCI slot. Use the `lspci` command in advance to determine which PCI device the Mellanox card is hosted on. For example:

```
[user@mysystem ~l$ lscpi | grep -i Mel
c1:00.0 Infiniband controller: Mellanox Technologies MT28908 Family [ConnectX-6]
c1:00.1 Infiniband controller: Mellanox Technologies MT28908 Family [ConnectX-6]
```

3.1.13 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.

3.1.14 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.

3.1.15 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.

- **xGMI Link Width Control:** Manual
- **xGMI Force Link Width Control:** Force
- **xGMI Max Link Width Control:** Manual

Running all xGMI links at x16 also draws more power to the xGMI interface, which may slightly reduce the effective core frequency. If you are trying to fine-tune performance, then AMD recommends testing your applications in both their default configuration and with the above settings.

The following items disable some Rep-Move-Store instructions.

- **Fast Short REP MOVSB:** Disabled
- **Enhanced REP MOVSB/STOSB:** Disabled
- **REP-MOV/STOS Streaming:** Disabled

3.1.16 3D V-Cache = Auto

3D V-Cache only takes effect in systems powered by AMD EPYC 7003 Series Processors equipped with AMD 3D V-Cache technology. Setting this option to either **Auto** or **Enable** enables up to 96 MB of L3 cache per CCD compared to the 32 MB of L3 cache available to AMD EPYC 7003 Series Processors without AMD V-Cache technology. AMD recommends setting this option to **Auto** to leverage the extra cache. If your application is not cache or memory bandwidth sensitive, then AMD recommends setting this option to **Disabled**.

Chapter 4 Operating Systems

4.1 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.

See "Operating Systems" on page 1.

Note: Confirm that your planned OS supports your applications.

4.2 /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.

4.2.1 /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.

4.2.2 /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
```

4.3 Transparent Huge Pages (THP)

If transparent hugepages are 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
```

4.4 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.

4.5 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
```

4.6 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*

4.7 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 21) 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 thast 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 5 Compilers, Libraries, and Profilers

Visit AMD Developer Central to access everything mentioned in this chapter, including AMD libraries, compilers, and user guides. You need not sign in to download forms; download begins as soon as you accept the terms.

- **AOCC:** AMD Open CPU Compiler (AMD's CPU performance compiler), which consists of a C/C++ compiler (`clang`) and a Fortran compiler (`flang`). Download AOCC from https://developer.amd.com/amd-aocc/.
- **AOCL:** AMD Open CPU Libraries (AMD CPU math library suite), which consists of optimized math libraries. Download AOCL from https://developer.amd.com/amd-aocl/.

AMD continues actively developing both AOCC and AOCL to meet the needs of HPC users. AOCC Version 3.0 and AOCL version 3.0 became available when the 3rd Gen AMD EPYC Series processors launched and the latest version is AOCC 3.2 and AOCL 3.1 released Dec 10, 2022. These tools are preferred for AMD EPYC processors; however, GCC 11.0 and later also include "Zen 3" support.

5.1 AMD Optimizing CPU Compiler (AOCC)

The AOCC compiler system is a high-performance, production quality code generation tool that offers various options to developers building and optimizing C, C++, and Fortran applications targeting 32-bit and 64-bit Linux® platforms. It offers a high level of advanced optimizations, multi-threading, and processor support that includes global optimization, vectorization, inter-procedural analyses, loop transformations, and code generation. AOCC 3.2 is tuned for AMD Family 17h processors and includes the following:

- Machine-dependent optimizations for better performance on AMD EPYC 7003 Series processors.
- Enhanced high-level optimizations for AMD EPYC 7003 Series processors.
- Improved `flang` as the default Fortran front end with added F2008 features.
- Based on the LLVM 13.0 release (llvm.org*, October 4, 2021).
- Optimized libraries including AMDLibM
- LLVM linker (`lld`) as the default linker
- Tested on RHEL 8.3, SLES 12 SP5, Ubuntu 18.04 LTS, and Ubuntu 19.04.

This chapter includes useful compiler options you can pass to `clang` and `flang`, and you can also see "Stream" on page 53 and "DGEMM" on page 57.

5.1.1 AOCC Clang/Clang++

Clang is a C, C++, and Objective-C compiler that encompasses preprocessing, parsing, optimization, code generation, assembly, and linking. Clang supports the `-march=znver3` flag to enable best code generation and tuning for 3rd Gen AMD EPYC Series processors.

5.1.2 AOCC FLang

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

5.1.3 Clang and Flang Options

Table 5-1 and Table 5-2 list AOCC `clang` and `flang` compiler options, respectively. You may also see "Stream" on page 53 and "DGEMM" on page 57 for additional useful compiler options for `clang` (and therefore `flang`).

Architecture	
Generate instructions that run on 3rd Gen AMD EPYC Series CPUs	`-march=znver3`
Generate instructions for the local machine	`-march=native`
Optimization Levels	
Disable all optimizations	`-O0`
Minimal level speed and code optimization	`-O1/ -O`
Moderate level optimization	`-O2`
Aggressive optimization	`-O3`
Maximize performance	`-Ofast`
Enable link time optimizations	`-flto`

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

Enable advanced loop optimizations	`-unroll-aggressive`

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

Enable advanced vectorization	`-enable-strided-vectorization` `-enable-epilog-vectorization`
Enable memory layer optimizations	`-fremap-arrays (use with flto)`
Profile guided optimizations	`-fprofile-instr-generate (1st invoc.)` `-fprofile-instr-use (2nd invocation)`
OpenMP®	`-fopening`
For enabling memory stores, memory bandwidth workoads	`-fnt-store`
Enable removal of all unused array computation	`-reduce-array-computations=3`
Other Options	
Enable faster, less precise math operations (part of Ofast)	`-ffast-math` `-freciprocal-math`

Table 5-1: AOCC Clang compiler options

OpenMP threads and affinity (N number of cores)	`export OMP_NUM_THREADS=N` `export GOMP_CPU_AFFINITY="0-{N-1}"`
Enabling vector library	`-fveclib=AMDLIBM`
Link to AMD library	`-L/libm-install-dir/lin -lalm`
For Fortran workloads	
Compile free form Fortran	`-ffree-form`

Table 5-1: AOCC Clang compiler options (Continued)

5.2 GCC Compiler

GCC is the default compiler that ships with RHEL 8.3. GCC versions 11 and later support the "Zen 3" architecture.

Architecture	
Generate instructions that run on 3rd Gen AMD EPYC Series CPUs	`-march=znver3`
Generate instructions for the local machine	`-march=native`
Optimization Levels	
Disable all optimizations	`-O0`
Minimal level speed and code optimization	`-O1/ -O`
Moderate level optimization	`-O2`
Aggressive optimization	`-O3`
Maximize performance	`-Ofast`
Additional Optimizations	
Enable link time optimizations	`-flto`
Enable unrolling	`-funroll-all-loops`
Generate memory preload instructions	`-fprefetch-loop-arrays --param prefetch-latency=300`
Profile-guided optimization	`-fprofile-generate (1st invocation)` `-fprofile-use (2nd invocation)`
OpenMP	`-fopenmp`
Other Options	
Enable generation of code that follows IEEE arithmetic	`-mieee-fp`
Enable faster, less precise math operations (part of Ofast)	`-ffast-math`
Compile free form Fortran	`-ffree-form`
OpenMP threads and affinity (N number of cores)	`export OMP_NUM_THREADS=N` `export GOMP_CPU_AFFINITY="0-{N-1}"`
Link to AMD library	`-L/libm-install-dir/lin -lalm`

Table 5-2: GCC compiler options

5.3 AMD Optimizing CPU Libraries (AOCL)

AMD Optimizing CPU Libraries (AOCL) are a set of numerical libraries tuned specifically for the AMD EPYC processor family. They include a simple interface that takes advantage of the latest hardware innovations. THe latest release at the time this Tuning Guide was published is 3.1, which as releast on December 21st, 2021. Visit https://developer.amd.com/amd-aocl/ for more details on the AOCL libraries, tar balls of prebuilt libraries, and instructions on how to build AOCL libraries from source. You can also install these libraries using Spack, as described in "Spack" on page 43.

AOCL consists of the following libraries:

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

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

5.3.1 BLIS

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

5.3.2 libFLAME

libFLAME is a portable library for dense matrix computations that provides the complete functionality present in Linear Algebra Package (LAPACK). It includes the FLAPACK compatibility layer, which includes the complete LAPACK implementation. The library provides scientific and numerical computing communities with a modern, high-performance dense linear algebra library. It is extensible, easy to use, and available under an open-source license. libFLAME is a C-only implementation and does not depend on any external FORTRAN libraries, including LAPACK. The

optional lapack2flame backward compatibility layer maps LAPACK routine invocations to their corresponding native C implementations in libFLAME. This allows the legacy applications to take advantage of libFLAME with virtually no changes to their source code. You may download the source code from https://github.com/amd/libflame*.

5.3.3 FFTW

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

5.3.4 LibM

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

5.3.5 ScaLAPACK

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

5.3.6 AMD Random Number Generator

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

5.3.7 AMD Secure RNG

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

5.3.8 AOCL-Sparse

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

5.4 AOCL Tuning Guidelines

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

5.4.1 AOCL Dynamic

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

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

You can also specify the preferred number of threads using the environment variables `BLIS_NUM_THREADS` or `OMP_NUM_THREADS`. If you specify both variables, then `BLIS_NUM_THREADS` takes precedence. Table 5-3 summarizes how the number of threads is determined based on the status of AOCL Dynamic and the user configuration using the variable BLIS_NUM_THREADS:

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

Table 5-3: AOCL Dynamic

5.4.2 BLIS DGEMM Multi-thread Tuning

You can use a BLIS library on multiple platforms and applications. Multi-threading adds more configuration options at runtime by controlling the number of threads and CPU affinity settings that can be tuned to get the best performance for your requirements. Execute one of the following commands to obtain the best DGEMM multi-thread performance on 3rd Gen AMD EPYC processors (both with and without AMD 3D V-Cache technology):

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

5.4.3 Performance Suggestions for Skinny Matrices

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

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

```
$BLIS_PACK_B=1 ./test_gemm_blis.x – will give a better performance.
```

5.4.4 AMD-Optimized FFTW

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

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

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

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

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

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

  Where:

  `numactl --interleave=0,1,2,3` sets the memory interleave policy on nodes 0, 1, 2, and 3.
- When running a MPI FFTW test, set the appropriate MPI mapping, binding, and rank options. For example, to run 64 MPI rank FFTW on a 64-core AMD EPYCTM processor:
  ```
  mpirun --map-by core --rank-by core --bind-to core -np 64 ./mpi-bench -opatient -s icf65536
  ```

5.5 AMD µProf – Profiling Tool

AMD µProf is a performance analysis tool for applications running on the Linux and Windows operating systems. It allows developers to better understand the runtime performance of their application and to identify ways to improve its performance. Users can download it for free from https://developer.amd.com/amd-uprof/. AMD EPYC 7003 processors require µProf v3.4 or later. AMD µProf offers:

- **Performance Analysis:** The CPU profile identifies application runtime performance bottlenecks.
- **System Analysis:** The Performance Counter Monitor utility monitors system performance metrics such as IPC and memory bandwidth.
- **Power Profiling:** System-wide power profiles monitor system thermal and power characteristics.
- **Energy Analysis:** Power Application Analysis identifies energy hotspots in the application (Windows only).

You can use AMD µProf to:

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

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

µProf 3.5 was released on January 17th, 2022 with the following new features:

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

- **AMDuProfSys:** A new system analysis tool that captures system information that can help debug issues.

5.5.1 System Analysis Utility (AMDuProfPcm)

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

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

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

CORE METRICS					
Metric	System (Aggregated)	Package (Aggregated)-0	Package (Aggregated)-1	CCD (Aggregated)-0	CCD (Aggregated)-1
Utilization (%)	9.55	9.45	9.66	10.17	9.32
Eff Freq	3262.11	3257.45	3266.7	3258.25	3257.94
IPC (Sys + User)	0.13	0.11	0.14	0.11	0.11
CPI (Sys + User)	7.97	9.11	7.09	9.31	9.43
Branch Misprediction Ratio	0	0	0	0	0
IC(32B) Fetch Miss Ratio	0.01	0.01	0.01	0.01	0.01
DC Access (pti)	617.67	647.34	593.26	644.32	644.19
L2 Access (pti)	219.23	245.05	198.91	213.89	243.45
L2 Access from IC Miss (pti)	1.67	1.5	1.81	1.38	1.52
L2 Access from DC Miss (pti)	119.28	133.53	108.07	116.33	131.41
L2 Access from HWPF (pti)	23.93	25	23.08	22.28	25.77
L2 Miss (pti)	36.63	40	34.04	36.27	40.36
L2 Miss from IC Miss (pti)	1.03	1.13	0.95	0.96	1.2
L2 Miss from DC Miss (pti)	21.51	24.51	19.21	22.21	24.23
L2 Miss from HWPF (pti)	14.09	14.36	13.88	13.1	14.94
L2 Hit (pti)	26.58	27.4	25.94	24.11	28.53
L2 Hit from IC Miss (pti)	0.78	0.29	1.15	0.19	0.19
L2 Hit from DC Miss (pti)	15.72	16.22	15.34	14.53	17.24
L2 Hit from HWPF (pti)	9.84	10.64	9.2	9.18	10.83
L3 METRICS					
Metric	System (Aggregated)	Package (Aggregated)-0	Package (Aggregated)-1	CCD (Aggregated)-0	CCD (Aggregated)-1
L3 Access	4762627165	2338595992	2424031173	292903705	291774043
L3 Miss	4667265146	2295013129	2372252017	287397063	286449222
L3 Miss %	98	98.14	97.86	98.12	98.18
Ave L3 Miss Latency	942.58	951.81	933.64	960.34	964.23

Figure 5-1: Sample AMDuProfPcm analysis results

5.5.2 Application Analysis

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

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

The minimal RHEL8 Linux kernel version with full support for 3rd Gen AMD EPYC Series processor core PMC events is `kernel-4.18.0-240.el8`.

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

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

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

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

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

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

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

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

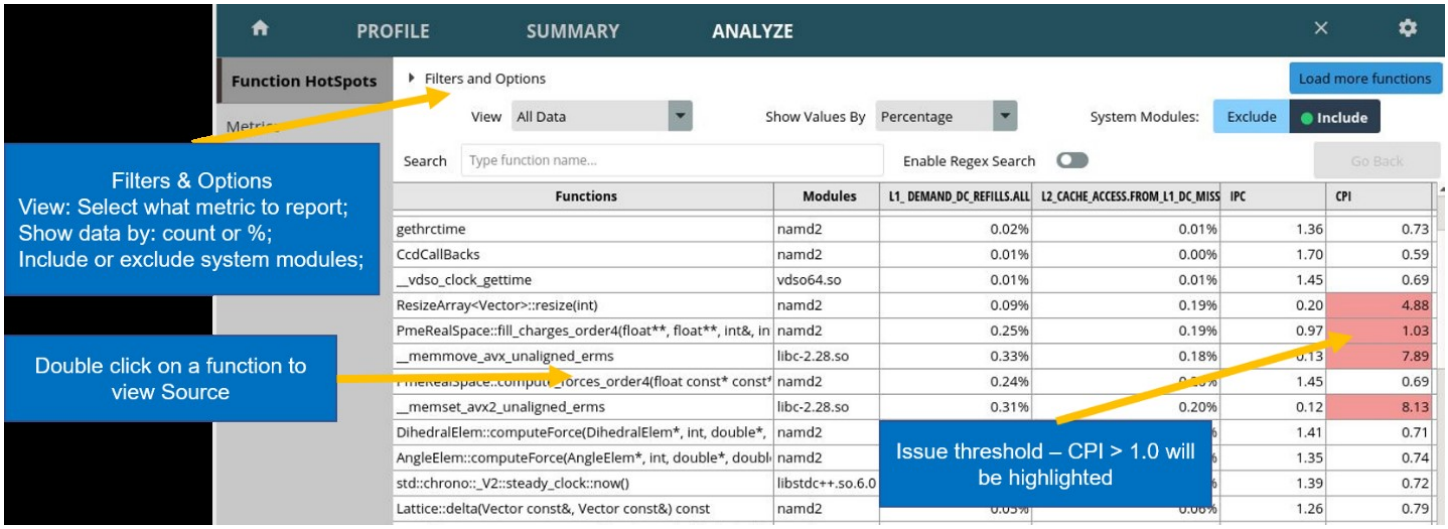
```
./AMDuProfCLI translate -i /tmp/AMDuProf-classic-EBP_Feb-25-2022_21-13-55/
```

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

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

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

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



Functions	Modules	L1_DEMAND_DC_REFILLS.ALL	L2_CACHE_ACCESS.FROM_L1_DC_MISS	IPC	CPI
gethrctime	namd2	0.02%	0.01%	1.36	0.73
CcdCallBacks	namd2	0.01%	0.00%	1.70	0.59
__vdso_clock_gettime	vdso64.so	0.01%	0.01%	1.45	0.69
ResizeArray<Vector>::resize(int)	namd2	0.09%	0.19%	0.20	4.88
PmeRealSpace::fill_charges_order4(float**, float**, int&, in	namd2	0.25%	0.19%	0.97	1.03
__memmove_avx_unaligned_erms	libc-2.28.so	0.33%	0.18%	0.13	7.89
PmeRealSpace::compute_forces_order4(float const* const*	namd2	0.24%		1.45	0.69
__memset_avx2_unaligned_erms	libc-2.28.so	0.31%	0.20%	0.12	8.13
DihedralElem::computeForce(DihedralElem*, int, double*,	namd2			1.41	0.71
AngleElem::computeForce(AngleElem*, int, double*, doubl	namd2			1.35	0.74
std::chrono::_V2::steady_clock::now()	libstdc++.so.6.0			1.39	0.72
Lattice::delta(Vector const&, Vector const&) const	namd2			1.26	0.79

Figure 5-2: The Function HotSpots window

The following information is available for each source tab:

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

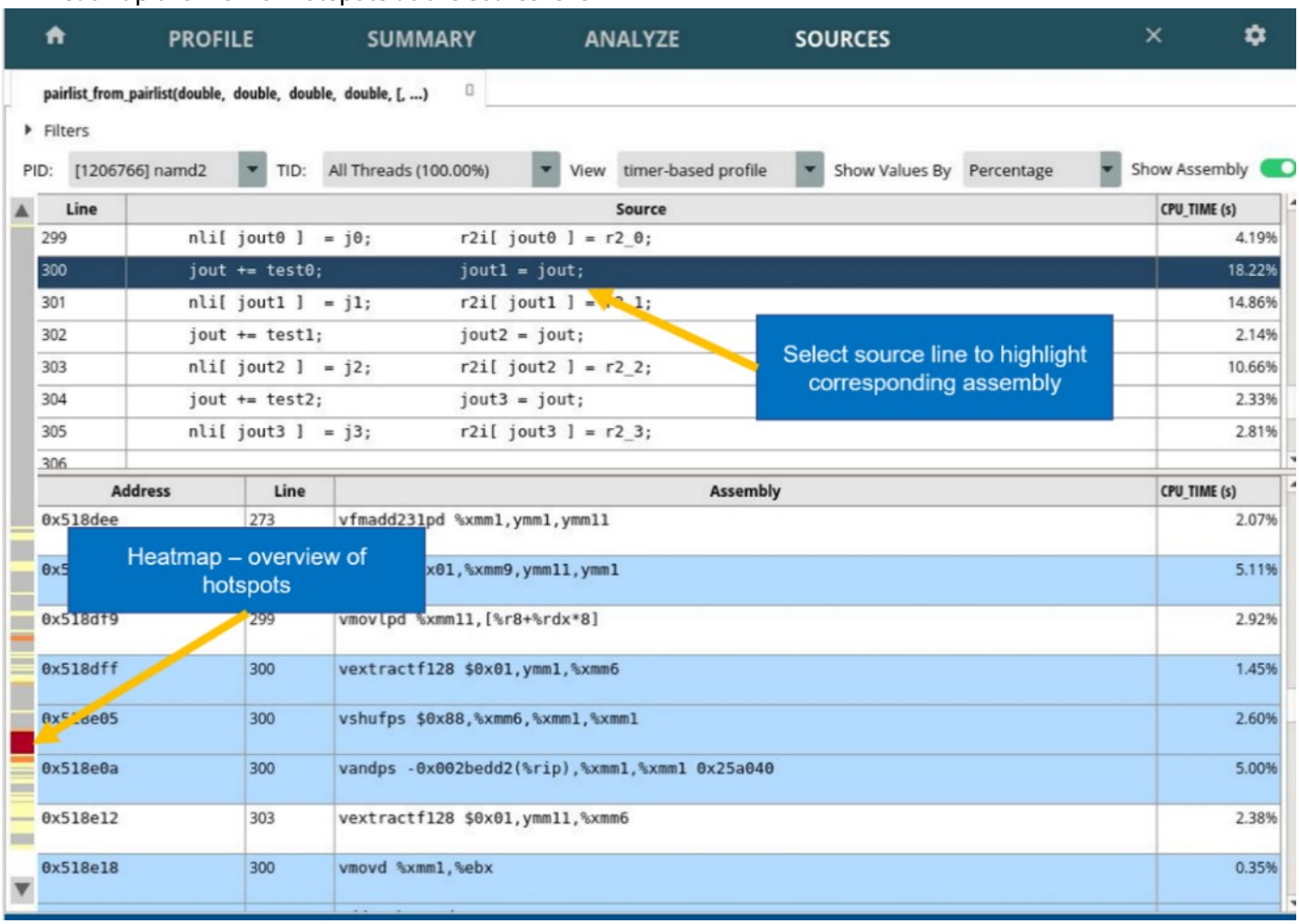


Figure 5-3: SOURCES source and assembly window

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

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

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

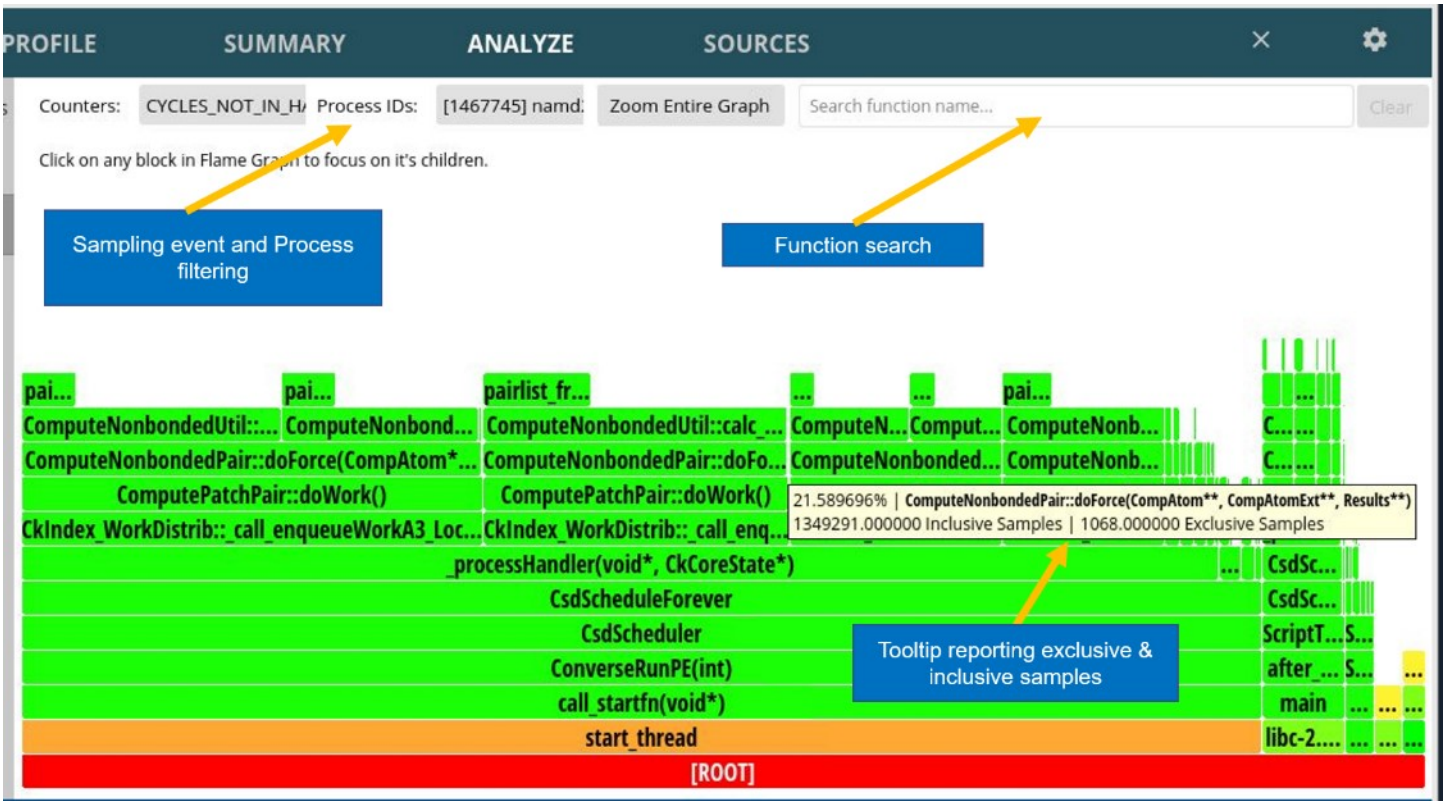
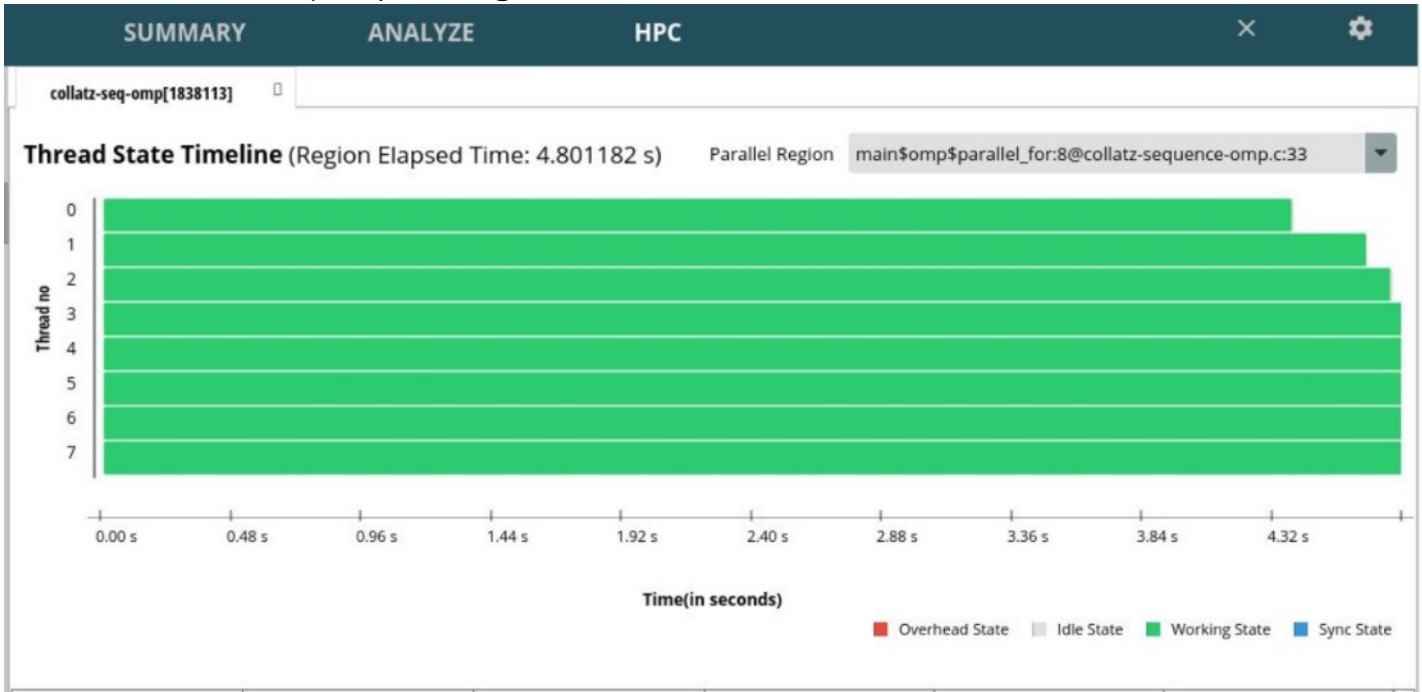


Figure 5-4: The Flame Graph window

5.5.3 HPC Analysis

To enable OpenMP profiling and analyze tracing data:

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



Thread no	Thread Id	Idle Time (s)	Sync Time (s)	Overhead Time (s)	Work Time (s)
0	1838113	0.00	0.00	0.00	4.21
1	1838117	0.00	0.00	0.00	4.48
2	1838118	0.00	0.00	0.00	4.56
3	1838119	0.00	0.00	0.00	4.64
4	1838120	0.00	0.00	0.00	4.67
5	1838121	0.00	0.00	0.00	4.72
6	1838122	0.00	0.00	0.00	4.72
7	1838123	0.00	0.00	0.00	4.80

Figure 5-5: Regions Detailed Analysis page with thread activity

5.5.4 MPI Analysis

AMD µProf includes the following MPI-specific profiling options:

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

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

1. Collect the profile data:

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

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

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

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

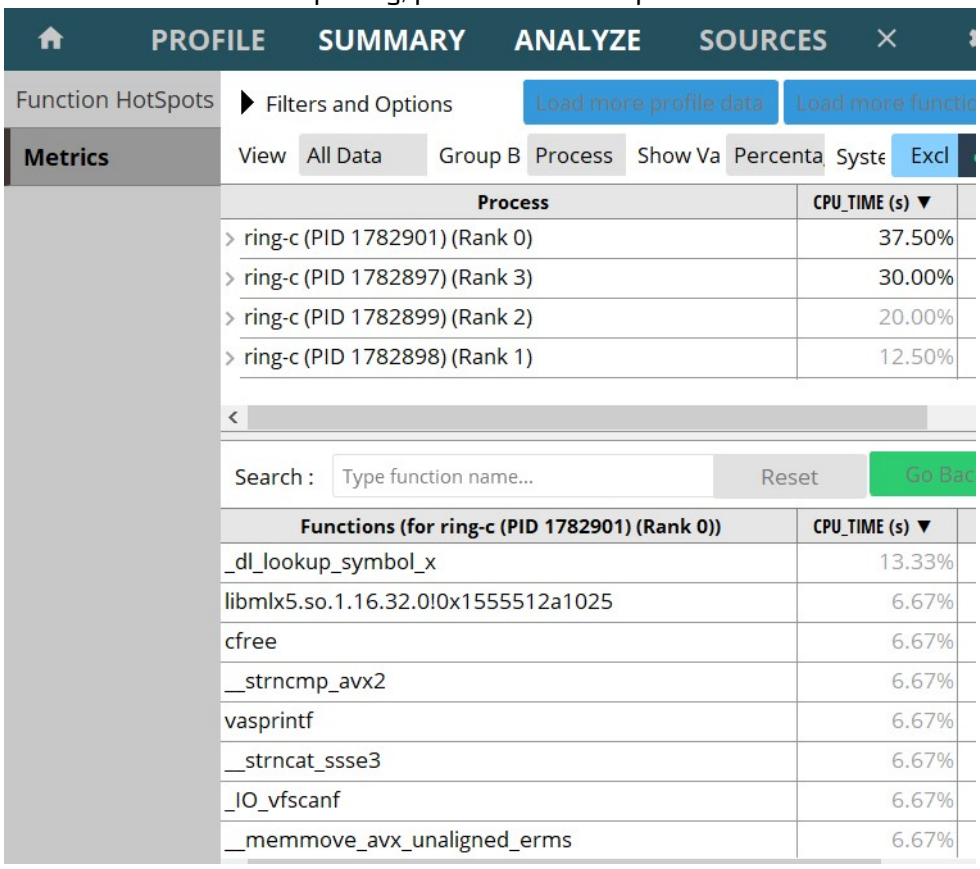


Figure 5-6: Imported MPI process data

5.5.5 MPI Trace Analysis

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

- `FUNNLED`
- `SERIALIZED`

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

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

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

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

The list of supported MPI APIs includes:

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

5.5.6 Power Profiling

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

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

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

5.5.6.1 Profiling Effective Frequency

To collect power profile counter values:

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

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

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

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

5.5.6.2 Live System-Wide Power Profile

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

1. In the **PROFILE** page, select the appropriate profile target by clicking the **Next** button.

2. In the **Select Profile Type** fragment, select **System-wide Power Profile (Live)**, and then enable the desired counters from the list.

3. Click the **Start Profile** button to start collecting profile data. This profile type reports profile data as line graphs in the **TIMECHART** page for further analysis.

Figure 5-7: Thermal characteristic - Timeline

5.6 Spack

Spack is a package manager that simplifies installing various HPC codes and microbenchmarks. The following procedure walks you through installing Spack and downloading the AMD AOCC compiler through Spack. See "Synthetics" on page 53 for examples of using Spack to install the STREAM and HPL benchmarks. To set up your Spack environment:

1. Clone the public Spack repo.

```
git clone https://github.com/spack/spack.git
```

2. Set `SPACK_ROOT` to the root of the cloned repo:

```
export SPACK_ROOT=<insert the root of your local Spack repo>
source ${SPACK_ROOT}/share/spack/setup-env.sh
```

3. Install `gcc`.

```
spack install gcc@10.2.0
```

4. Install `aocc`. A license prompt appears in vi; close this to proceed with the installation.

```
spack install -v aocc@3.0.0 +license-agreed
```

5. Make Spack aware of the new compilers:

```
spack cd -i aocc@3.0.0
spack compiler add $PWD
spack cd -i gcc@10.2.0
spack copiler add $PWD
```

6. Ensure that Spack is aware of the new compilers by executing the command `spack compilers`:

```
==> Available compilers
-- aocc rhel8-x86_64 ---------------------------------------------------------
aocc@3,0.0  aocc@2.3.0   aocc@2.2.0

-- clang rhel8-x86_64 --------------------------------------------------------
clang@10.0.1

-- gcc rhel8-x86_64 ----------------------------------------------------------
gcc@11.0.0  gcc@10.2.0  gcc@8.3.1
```

You can also find the list of AMD math libraries using Spack (bold text):

```
[jason@milan ~]$ spack list amd
==> 8 packages.
amdblis  amdfftw  amdlibflame  amdlibm  amdscalapack  bamdst  llvm-amdgpu  namd
```

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.

AMD also released a public userspace library for Linux systems that allows customers to write their own userspace code to implement specific HSMP functions. That library is available from https://github.com/AMDESE/libhsmp. This source code contains an example that invokes the API exported by `libhsmp`.

This page intentionally left blank.

Chapter 8

Executing Applications on AMD EPYC 7003 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	64				

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.

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_3072` test case (higher is better) where we test 1/2/4/8 cores per L3 for each NPS setting.

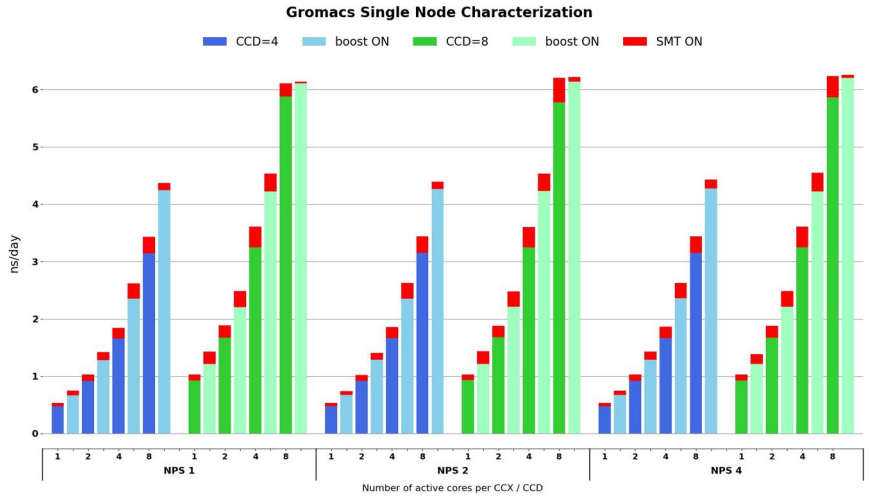


Figure 8-1: Sample GROMACS performance characterization

The GROMACS characterization study used a server with 2x 64-core AMD EPYC 7763 CPUs with 1,2,..,8 cores per L3/CCD. The AMD EPYC 7763 CPU provides the flexibility to test NPS=1/2/4 and reduce CCD count from 8 to 4 CCDs. You can now determine whether a 64, 48, or 32 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):

- The left bar reflects BOOST=OFF.
- The right bar reflects BOOST=ON. The red portion at the top of the bar shows the incremental benefit of SMT=ON.

This example achieves best performance with NPS=4, 8x CCDs, 8 cores per L3 (64 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 inteface for CPU ID (`-cpu-list`). Explicit pinning remains advisable when using all of the cores/threads within the socket to ensure minimum execution time. Hybrid MPI + OpenMP codes normally offer three strategies that y can adopt when executing code:

- All MPI ranks per available thread.
- Map by L3 (1 MPI rank per L3).
  - If SMT=OFF, then OMP_NUM_THREADS is normally set to the number of cores residing on that L3 cache. A 64-core CPU uses OMP_NUM_THREADS=8.
  - If SMT=ON, then OMP_NUM_THREADS is normally set to the number of threads residing on that L3 cache. A 64-core CPU uses OMP_NUM_THREADS=16.
- 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 7763 (i.e. 2x 64-core CPUs) with:

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

```
export CPULIST=$(seq -s , 0 8 127)
mpirun --bind-to none -cpu-list $CPULIST -x UCX_NET_DEVICES=mlx5_2:1 \\
--app myappfile-1rankperL3.txt
```

The Infiniband card in this example is a dual port device represented as `mlx5_2` and this example uses port 1, hence `UCX_NET_DEVICES=mlx5_2:1`. The accompanying appfile `myappfile-1rankperL3.txt` is therefore:

```
-np 1 numactl --physcpubind=0 mybinary -parallel
-np 1 numactl --physcpubind=8 mybinary -parallel
-np 1 numactl --physcpubind=16 mybinary -parallel
-np 1 numactl --physcpubind=24 mybinary -parallel
-np 1 numactl --physcpubind=32 mybinary -parallel
-np 1 numactl --physcpubind=40 mybinary -parallel
-np 1 numactl --physcpubind=48 mybinary -parallel
-np 1 numactl --physcpubind=56 mybinary -parallel
-np 1 numactl --physcpubind=64 mybinary -parallel
-np 1 numactl --physcpubind=72 mybinary -parallel
-np 1 numactl --physcpubind=80 mybinary -parallel
-np 1 numactl --physcpubind=88 mybinary -parallel
-np 1 numactl --physcpubind=96 mybinary -parallel
-np 1 numactl --physcpubind=104 mybinary -parallel
-np 1 numactl --physcpubind=112 mybinary -parallel
-np 1 numactl --physcpubind=120 mybinary -parallel
```

This page intentionally left blank.

Chapter 9 Synthetics

9.1 Stream

STREAM tests the maximum memory bandwidth of either a core or an entire CPU. To build the STREAM benchmark using Spack:

1. Build STREAM using Spack:

```
spack install stream %aocc@3.0.0 +openmp cflags="-O3 -mcmodel=large -DSTREAM_TYPE=double -
mavx2 -DSTREAM_ARRAY_SIZE=2500000000 -DNTIMES=10 -ffp-contract=fast -fnt-store"
```

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

5. Run STREAM

```
#!/bin/bash
export GOMP_CPU_AFFINITY=0-127:8
export OMP_NUM_THREADS=16
stream_c.exe
```

The following tables compare STREAM Triad results (MB/s) on a system with 2x 7763 and 1 DPC (DIMM Per Channel), with DDR4-3200 Dual-Rank DIMMS. They also check 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=8 NPS=4		**SMT=OFF**			**SMT=ON**		
		Threads	Boost=OFF	Boost=ON	Threads	Boost=OFF	Boost=ON
Cores/L3	1	16	382279	382898	32	368351	368366
	2	32	369900	370501	64	353369	357269
	4	64	357984	357437	128	341277	340916
	6	96	351926	351950	192	330823	332235
	8	128	344876	342379	256	299722	309224

Table 9-1: SMT comparison, NPS=4

CCD=8 NPS=2		**SMT=OFF**			**SMT=ON**		
		Threads	Boost=OFF	Boost=ON	Threads	Boost=OFF	Boost=ON
Cores/L3	1	16	354224	354814	32	342912	342379
	2	32	343740	343565	64	326477	332935
	4	64	336843	335518	128	316929	313345
	6	96	330985	324872	192	313071	311893
	8	128	323096	320987	256	282781	290895

Table 9-2: SMT comparison, NPS=2

CCD=8 NPS=1		**SMT=OFF**			**SMT=ON**		
		Threads	Boost=OFF	Boost=ON	Threads	Boost=OFF	Boost=ON
Cores/L3	1	16	324173	325631	32	321793	321962
	2	32	323916	323745	64	307692	311813
	4	64	315042	314773	128	303017	299224
	6	96	310682	311436	192	299270	299767
	8	128	302644	301458	256	282406	272847

Table 9-3: SMT comparison, NPS=1

Table 9-4 shows how performance changes when CCDs are reduced from CCD=8 to CCD=4 per socket:

CCD=4 NPS=4		**SMT=OFF**			**SMT=ON**		
		Threads	Boost=OFF	Boost=ON	Threads	Boost=OFF	Boost=ON
Cores/L3	1	16	334096	342319	32	330412	336069
	2	32	357770	359066	64	341562	340614
	4	64	336162	336473	128	320185	320301
	6	96	326688	326916	192	315644	315675
	8	128	319967	318126	256	308805	396941

Table 9-4: Performance changes when changing CCD=8 to CCD=4 per socket

9.2 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.

1. Build HPL using Spack:

```
spack -d install -v hpl %aocc@3.1.0 +openmp target=zen3 cflags="-O3" ^amdblis@3.0
threads=openmp ^openmpi@4.0.5 fabrics="knem" ^knem%gcc@9.2.0 target=zen2
```

HPL requires tuning for both each processor type/configuration and memory configuration. The following example assumes a dual-socket system with 64-core EPYC 7763 processors and 512GB of memory, configured with SMT=OFf, cTPD=PPL=280W, and memory speed = 3200 MHz.

2. Create the following scripts to run HPL:

   - `run_hpl_ccx.sh`

```
#! /bin/bash
source ${SPACK_ROOT}/share/spack/setup-env.sh
# To load the Openmpi modules (To load specific OpenMPI use the # value.  spack load
openmpi /version_hash)
spack load openmpi

spack load hpl %aocc@3.0.0 # To load HPL

ldd xhpl
which mpicc
# Run the appfile which specifies 16 processes, each with its own CPU binding for OpenMP
# Verify the knem module is loaded
lsmod | grep -q knem
mpi_options="--mca mpi_leave_pinned 1 --bind-to none --report-bindings --mca btl
self,vader --map-by ppr:2:l3cache -x OMP_NUM_THREADS=4 -x OMP_PROC_BIND=TRUE -x
OMP_PLACES=cores"
mpirun $mpi_options -app ./appfile_ccx
```

   - `xhphl_ccx.sh`

```
#! /bin/bash
#
# Bind memory to node $1 and four child threads to CPUs specified in $2
#
# Kernel parallelization is performed at the 2nd innermost loop (IC)
export LD_LIBRARY_PATH=$BLISROOT/lib:$LD_LIBRARY_PATH

export OMP_NUM_THREADS=$3
export GOMP_CPU_AFFINITY="$2"
export OMP_PROC_BIND=TRUE
# BLIS_JC_NT=1 (No outer loop parallelization):
export BLIS_JC_NT=1
# BLIS_IC_NT= #cores/ccx (# of 2nd level threads - one per core in the shared L3 cache
domain):
export BLIS_IC_NT=$OMP_NUM_THREADS
# BLIS_JR_NT=1 (No 4th level threads):
export BLIS_JR_NT=1
# BLIS_IR_NT=1 (No 5th level threads):
export BLIS_IR_NT=1
numactl --membind=$1 ./xhpl
```

- `appfile_ccx`. The following sample is for a dual-socket 64-core server with SMT=OFF. It configures HPL to run in a hybrid MPI + OpenMP configuration with 4 cores/MPI rank.

```
-np 1 ./xhpl_ccx.sh 0 0-3 4
-np 1 ./xhpl_ccx.sh 0 4-7 4
-np 1 ./xhpl_ccx.sh 0 8-11 4
-np 1 ./xhpl_ccx.sh 0 12-15 4
-np 1 ./xhpl_ccx.sh 1 16-19 4
-np 1 ./xhpl_ccx.sh 1 20-23 4
-np 1 ./xhpl_ccx.sh 1 24-27 4
-np 1 ./xhpl_ccx.sh 1 28-31 4
-np 1 ./xhpl_ccx.sh 2 32-35 4
-np 1 ./xhpl_ccx.sh 2 36-39 4
-np 1 ./xhpl_ccx.sh 2 40-43 4
-np 1 ./xhpl_ccx.sh 2 44-47 4
-np 1 ./xhpl_ccx.sh 3 48-51 4
-np 1 ./xhpl_ccx.sh 3 52-55 4
-np 1 ./xhpl_ccx.sh 3 56-59 4
-np 1 ./xhpl_ccx.sh 3 60-63 4
-np 1 ./xhpl_ccx.sh 4 64-67 4
-np 1 ./xhpl_ccx.sh 4 68-71 4
-np 1 ./xhpl_ccx.sh 4 72-75 4
-np 1 ./xhpl_ccx.sh 4 76-79 4
-np 1 ./xhpl_ccx.sh 5 80-83 4
-np 1 ./xhpl_ccx.sh 5 84-87 4
-np 1 ./xhpl_ccx.sh 5 88-91 4
-np 1 ./xhpl_ccx.sh 5 92-95 4
-np 1 ./xhpl_ccx.sh 6 96-99 4
-np 1 ./xhpl_ccx.sh 6 100-103 4
-np 1 ./xhpl_ccx.sh 6 104-107 4
-np 1 ./xhpl_ccx.sh 6 108-111 4
-np 1 ./xhpl_ccx.sh 7 112-115 4
-np 1 ./xhpl_ccx.sh 7 116-119 4
-np 1 ./xhpl_ccx.sh 7 120-123 4
-np 1 ./xhpl_ccx.sh 7 124-127 4
```

- `HPL.dat`. The following sample provided is for a server with 512GB of memory. You must adjust this to suit you needs.

```
HPLinpack 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)
241024       Ns
1            # of NBs
224          # of problems sizes (N)
0            MAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
4            Ps
8            Qs
16.0         threshold
1            # of panel fact<
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            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)
```

3. Execute the command `./run_hpl_ccx.sh`.

   Using this process with AOCC v3.1 and AOCL v3.0 on 2-socket test servers results in 3.9 TFLOPS using AMD EPYC 7763 procesors (2x 64 cores).

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R2      241024   224     4     8            2349.63             3.9728e+03
HPL_pdgesv() start time Fri Jul  9 04:39:25 2021
```

9.3 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 is available from http://portal.nersc.gov/project/m888/apex/*. BLIS Multi-Threaded (MT) libraries are also required and are available from https://developer.amd.com/amd-aocl/blas-library/.

Makefile:

```
CFLAGS= -Ofast -fopenmp -lm -D USE_CBLAS -mavx2 -funroll-loops -lomp \\
        -ffp-contract=fast -mtune=znver3 -march=znver3
LIBS=/home/software/aocl/aocc/3.0-6/lib/libblis-mt.a
INC=-I/home/software/aocl/aocc/3.0-6/include
```

When compiled with AOCC, use the following flags if you want to run across all 64 cores on socket-0 in a dual-socket AMD EPYC 7763 system:

```
OMP_NUM_THREADS=64 GOMP_CPU_AFFINITY=0-63  \\
numactl --interleave=all ./mt-dgemm.aocc 14000
```

You can derive other combinations (per L3, per CCD) by carefully choosing which cores to pin to.

The result on one AMD EPYC 7763 socket is: `GFLOP/s rate: 2080.58 GF/s`

9.4 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 3rd 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.
- Enable Preferred-IO Mode in BIOS, as described in "Preferred-I/O Control" on page 19.
- The system manager can perform several tests once PFED is installed.

9.4.1 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:** `numactl -C 20 ib_write_bw -a --report_gbits`
- **On node-2:** `numactl -C 15 ib_write_bw -a --report_gbits -d mlx5_0 node-1`

This example:

- Tests the connection from `node 1` to `node 2`.
- Uses `numactl` to ensure you selected 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 descrbied in "OSU Network Tests" on page 60. The `hwloc-ls` command allows you to select the cores to choose for this purpose. See "Understanding hwloc-ls and hwloc-info" on page 7.
- Ensures the cores are idle in the C1 State, as described in <xref to discussion above>.
- 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     #iterations    BW peak[Gb/sec]    BW average[Gb/sec]   MsgRate[Mpps]
 2           5000            0.067164           0.066476            4.154756
 4           5000              0.13               0.13              4.187184
 8           5000              0.27               0.27              4.187324
 16          5000              0.54               0.54              4.197941
 32          5000              1.08               1.07              4.178680
 64          5000              2.15               2.14              4.176621
 128         5000              4.31               4.27              4.169307
 256         5000              8.61               8.57              4.182211
 512         5000             17.07              17.02              4.156448
 1024        5000             34.07              33.98              4.148378
 2048        5000             67.39              67.28              4.106271
 4096        5000            132.84             132.15              4.032974
 8192        5000            186.65             186.43              2.844643
 16384       5000            191.88             191.75              1.462959
 32768       5000            196.80             196.78              0.750658
 65536       5000            196.44             196.44              0.374673
 131072      5000            197.07             197.06              0.187927
 262144      5000            197.13             197.13              0.093999
 524288      5000            197.14             197.14              0.047001
 1048576     5000            197.13             197.13              0.023500
 2097152     5000            197.15             197.14              0.011751
 4194304     5000            197.11             197.10              0.005874
 8388608     5000            194.47             194.40              0.002897
```

This example shows a representative profile across the different packet sizes. Very small packet sizes do not achieve maximum bandwidth, but the maximum bandwidth of 200Gbps is achieved and maintained as the packet sizes increase. Achieving these results require 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.

9.4.2 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 58. You should expect to see latency of about 1 microsecond on a Mellanox HDR200 network with AMD EPYC 7003 Series processors. AMD measured 0.99 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.

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 --enable-mpi1-compatibility --with-hcoll=/opt/mellanox/hcoll \
--with-knem=/opt/knem-1.1.3.90mlnx1 --with-ucx=/opt/ucx/1.5.1 \
--with-xpmem=/opt/xpmem/2.6.5
```

9.5 OSU Network Tests

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 58 for the minimum correct versions of OFED and firmware when using ConnectX-6 cards.

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 one 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 t obtain the min, max, and average latency numbers. These tests use blocking versions of `MPI_Send` and `MPI_Recv`:

```
mpirun --allow-run-as-root -rf rankfile --report-bindings -x UCX_NET_DEVICES=mlx5_2:1
osu_latency
```

A rankfile was used to pin the thread on each system to a specific core. Isolate the test to testing the shortest latency between the systems by choosing 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	1.01
1	1
2	1.01
4	1.01
8	1
16	1.02
32	1.13
64	1.1
128	1.22
256	1.54
512	1.57
1024	1.7
2048	2.06
4096	2.58
8192	3.12
16384	4.28
32768	5.91
65536	7.9
131072	12.5
262144	15.52
524288	26.11
1048576	47.89
2097152	90.78
4194304	176.14

Table 9-5: Representative osu_latency test results

Chapter 10 Additional Information

10.1 Reference System

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

Setting	Value
BIOS	GIGABYTE Version M04 Release Date 05/17/2021
CPU	AMD EPYC 7763
Base Clock	2.45 GHz
NUMA	4
System Profile	Power
SMT	Off
Preferred IO	Off
Boost	On
C2	Disabled
CPU Governor	Performance
OS	Red Hat Enterprise Linux 8.3 (Ootpa)
Kernel	4.18.0-240.el8.x86_64
HCA	Mellanox ConnectX-6
Speed	HDR200
OFED	MLNX_OFED_LINUX-5.2-2.2.3.0
HCA Firmware	20.29.2002

Table 10-1: Reference system configuration

10.2 EDA Configuration

Registry Transfer Level simulations comprise more than half of the compute in most digital design companies. The tuning recommendations in this section are thus primarily geared towards this workload:

- **RAM:** Use 3200 MHz DIMMs to couple the Infinity Fabric Clock (1600 MHz)
- **BIOS:** Use the following BIOS settings:
  - **SMT:** Disabled. EDA applications not sufficiently threaded. Disabling SMT obtains maximum performance from every thread.
  - **Power:** Set **Power Determinism** mode to allow energy-efficient samples to run at their maximum possible speeds within TDP.
  - **NPS:** Set NPS=4 for the highest memory bandwidth and lowest latency. However, if you 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 settings:
  - **tuned-adm profile:** Use throughput-performance.
  - **Linux kernel:** Use a kernel with an AMD EPYC scheduling patch (e.g. kernel 4.15 or later, RHEL 7.7, or other OS with back-ported patches). https://git.kernel.org/pub/scm/linux/kernel/git/next/linux-next.git/commit/?id=051f3ca02e46432c0965e8948f00c07d8a2f09c0.

10.3 AMD Resources

- AMD EPYC Solution Briefs and Whitepapers (includes 7001, 7002, and 7003 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

10.4 Other Resources

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

This page intentionally left blank.