NVIDIA Grace Performance Tuning Guide
# Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>April 18. 2023</td>
<td>Initial Release</td>
</tr>
<tr>
<td>02</td>
<td>November 9, 2023</td>
<td>Version 1.0 Release</td>
</tr>
</tbody>
</table>
# Table of Contents

1. Introduction  
   1.1. NVIDIA Grace CPU Superchip and NVIDIA GH200 Grace Hopper Superchip Overview  
      1.1.1. High Performance Architecture  
      1.1.2. Alleviate Bottlenecks with NVLink-C2C Interconnect  
      1.1.3. Scale Cores and Bandwidth with NVIDIA Scalable Coherency Fabric  
      1.1.4. LPDDR5X Memory Subsystem  
      1.1.5. CPU I/O  
      1.1.6. Grace CPU Core Architecture  
      1.1.7. SIMD Vectorization  
      1.1.8. Atomic Operations  
      1.1.9. Additional Armv9 Features  
2. Understanding Your Grace Machine  
   2.1. Checking the CPUs  
   2.2. Checking the Non-Uniform Memory Access Settings  
   2.3. Checking the GPU  
   2.4. Checking the Memory  
3. Basic System Health Checks  
   3.1. STREAM  
   3.2. Fused Multiply Add  
   3.3. C2C CPU-GPU Bandwidth  
4. Power and Thermals  
   4.1. C-States  
   4.2. P-States  
   4.3. CPU Performance and Frequency Management  
   4.4. GPU and Module Power Management  
   4.5. Power and Thermal Management  
   4.6. Power Telemetry  
      4.6.1. Grace Power Telemetry  
      4.6.2. Comparing Grace and Intel® Power Telemetry  
      4.6.3. Comparing Grace and AMD Power Telemetry  
   4.7. Power Capping  
   4.8. CPU Temperature Management  
   4.9. GPU Temperatures  
5. Operating System Settings
1. Introduction

1.1. NVIDIA Grace CPU Superchip and NVIDIA GH200 Grace Hopper Superchip Overview

The Grace CPU is the first data center CPU designed by NVIDIA. The Grace CPU has 72 high-performance and power efficient Arm Neoverse V2 Cores, connected by a high-performance NVIDIA Scalable Coherency Fabric and server-class LPDDR5X memory.

The Grace CPU is found in two data center NVIDIA superchip products. The first is the NVIDIA GH200 Grace Hopper™ Superchip that pairs the power efficient, high-bandwidth NVIDIA Grace CPU with an NVIDIA H100 Tensor Core GPU to maximize the capabilities for accelerated computing and generative AI workloads. The heart of the GH200 Grace Hopper Superchip, is the NVLink-C2C that delivers up to 900 gigabytes per second (GB/s) of total bandwidth, which is 7X higher than PCIe Gen5 lanes commonly used in accelerated systems. NVLink-C2C enables the GPU to have direct access to over 600GB of memory, GH200 runs the full NVIDIA software stack and can be easily deployed in standard servers to run a variety of inference, data analytics, and other compute and memory-intensive workloads.

The second is the NVIDIA Grace CPU Superchip, with 144 cores in a no-compromise CPU for HPC, demanding cloud, and enterprise computing workloads. The Grace CPU Superchip delivers up to 1 TB/s of memory bandwidth, best-in-class data center throughput and up to 2X the performance per watt of today’s leading servers.
Figure 1-1. NVIDIA Grace CPU Superchip
Table 1-1. NVIDIA Grace CPU Superchip Specifications

<table>
<thead>
<tr>
<th>Grace CPU Superchip</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Core Architecture</strong></td>
</tr>
<tr>
<td><strong>Core Count</strong></td>
</tr>
<tr>
<td><strong>Cache</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Memory Technology</strong></td>
</tr>
<tr>
<td><strong>Raw Memory BW</strong></td>
</tr>
<tr>
<td><strong>Memory Size</strong></td>
</tr>
<tr>
<td><strong>FP64 Peak</strong></td>
</tr>
<tr>
<td><strong>PCI Express</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Power</strong></td>
</tr>
</tbody>
</table>

Table 1-2. NVIDIA GH200 Grace Hopper Superchip Specifications

<table>
<thead>
<tr>
<th>GH200 Grace Hopper Superchip</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU Core Architecture</strong></td>
</tr>
<tr>
<td><strong>CPU Core Count</strong></td>
</tr>
<tr>
<td><strong>CPU Cache</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>CPU Memory Technology</strong></td>
</tr>
<tr>
<td><strong>CPU Raw Memory BW</strong></td>
</tr>
<tr>
<td><strong>CPU Memory Size</strong></td>
</tr>
<tr>
<td><strong>GPU Multi-Processor</strong></td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
</tr>
<tr>
<td><strong>GPU Multi-Processor Count</strong></td>
</tr>
<tr>
<td><strong>GPU Memory Technology</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>GPU Memory Size</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
### 1.1.1. High Performance Architecture

The Grace CPU delivers high, single-threaded performance, high memory bandwidth, and outstanding data movement capabilities with leadership performance per watt. To enable the Grace CPU Superchip, these design goals required the development of several innovations.

The Grace Hopper CPU+GPU Superchip combines high performance of the Grace CPU with world-class GPU performance of the NVIDIA H100 GPU.

### 1.1.2. Alleviate Bottlenecks with NVLink-C2C Interconnect

To create the Grace CPU Superchip with up to 144 Arm Neoverse V2 cores and avoid bottlenecks when moving data between the chips, the NVLink Chip-2-Chip (C2C) interconnect provides a high-speed, direct connection between chips. A typical server architecture has two sockets, each composed of multiple dies and each die can represent multiple non-uniform memory (NUMA) domains. The Grace CPU Superchip uses a clean and simple memory topology. With only two NUMA nodes and the high-bandwidth NVLink-C2C, the Grace CPU Superchip helps alleviate NUMA bottlenecks for application developers and users.

Similarly, the memory of the Grace Hopper Superchip is set up as two NUMA nodes connected through the high-bandwidth NVLink-C2C, making access to both CPU and GPU memory seamless for applications developers and users.
1.1.3. Scale Cores and Bandwidth with NVIDIA Scalable Coherency Fabric

NVIDIA Scalable Coherency Fabric (SCF), shown in Figure 1-4, is a mesh fabric and distributed cache architecture that is designed by NVIDIA to scale cores and bandwidth. To keep data flowing between the CPU cores, NVLink-C2C, memory, and system IO, SCF provides over 3.2 TB/s of total bisection bandwidth.
The CPU cores and SCF cache partitions are distributed throughout the mesh, and the Cache Switch Nodes route data through the fabric and serve as interfaces between the CPU, cache memory, and system IOs. A Grace CPU Superchip has 234 MB of distributed L3 cache across the two chips.

**Figure 1-4. NVIDIA Grace CPU and the NVIDIA Scalable Coherency Fabric**

![Diagram of NVIDIA Grace CPU and the NVIDIA Scalable Coherency Fabric](image)

**Figure 1-4** shows the NVIDIA Grace CPU and the NVIDIA Scalable Coherency Fabric, which join the Neoverse V2 cores and distributed cache and system IO in a high-bandwidth mesh interconnect.

The Grace CPU supports Memory Partitioning and Monitoring (MPAM), which is the Arm® standard to partition the system cache and memory resources to provide performance isolation between jobs. By using MPAM, the NVIDIA-designed SCF Cache supports the partitioning of cache capacity, I/O, and memory bandwidth. MPAM also supports the use of MPAM performance monitor groups (PMGs) to monitor resources, such as cache storage usage and memory bandwidth usage.

### 1.1.4. LPDDR5X Memory Subsystem

The Grace CPU Superchip uses up to 960 GB of server-class LPDDR5X memory with Error Correction Code (ECC). This design strikes the optimal balance of bandwidth, energy efficiency, capacity, and cost for large-scale AI and HPC workloads.

Compared to an eight-channel DDR5 design, the Grace CPU LPDDR5X memory subsystem provides up to 53% more bandwidth at 1/8th the power per gigabyte per second while being close in cost. An HBM2e memory subsystem provides substantial
memory bandwidth and energy efficiency but at more than three times the cost-per-gigabyte and only one-eighth the maximum capacity with LPDDR5X.

The Grace CPU LPDDR5X architecture is the first data center class, resilient implementation of LPDDR technology. LPDDR5 channel sparing also restores the memory subsystem health upon reboot, which results in a low service rate due to failed memory. This allows the Grace CPU to be deployed in scenarios where serviceability is difficult and expensive.

The co-packaged memory employs a novel provisioning and error detection technique which eliminates the need to service or replace failed memory in the field, allowing the Grace CPU to be deployed in scenarios where serviceability is difficult or costly.

The lower power consumption of LPDDR5X reduces the overall system power requirements and enables more resources to be used in the CPU cores. The compact form factor enables twice the density of a typical DIMM-based design.

1.1.5.  CPU I/O

The Grace CPU Superchip supports up to 128 lanes of PCIe Gen 5 for I/O connectivity, and each PCIe Gen 5 x16 link supports up to 128 GB/s of bi-directional bandwidth and, for additional connectivity, can be bifurcated into 2x8s. Additional PCIe interfaces are provided for system management purposes. Server makers can use the standard expansion options for a variety of PCIe slot form factors with out-of-box support for NVIDIA GPUs, NVIDIA DPUs, NVIDIA ConnectX SmartNICs, E1.S, and M.2 NVMe devices, modular BMC options, and so on.

1.1.6.  Grace CPU Core Architecture

The Grace CPU Neoverse V2 core implements the Armv9.0-A architecture, which extends the architecture that was defined in the Armv8-A architectures up to Armv8.5-A. Application binaries that are built for an Armv8 architecture up to Armv8.5-A will execute on NVIDIA Grace, and this includes binaries that target CPUs like the Ampere Altra, the AWS Graviton2, and the AWS Graviton3.

Important: The NVIDIA HPC Compilers compile for fixed-length which are not binary compatible between, for example, Graviton and Grace.
1.1.7. SIMD Vectorization

The Neoverse V2 implements the following single instruction multiple data (SIMD) vector instruction sets in a 4x128-bit configuration:

- The Scalable Vector Extension version 2 (SVE2)
- Advanced SIMD (NEON)

Each of the four 128-bit functional units can retire SVE2 or NEON instructions, and this design allows more codes to take advantage of the SIMD performance.

Many applications and libraries are already taking advantage of Advanced SIMD (also known as NEON). SVE is a length-agnostic next generation SIMD instruction set architecture (orthogonal to Advanced SIMD) and provides features such as prediction, first faulting loads, gather, scatter, the ability to scale to large vector lengths without requiring recompilation, or porting to new vector lengths by hand. SVE2 provides vector length flexibility, which allows software efforts to focus on application specific optimizations.

SVE is implemented in many flagship Arm implementations, and by using length agnostic instructions for Grace CPU accrue toward portable binaries, ensures compatibility with SVE optimizations. SVE2 also extends the SVE ISA with advanced instructions that can accelerate key HPC applications like machine learning, genomics, and cryptography.

Refer to Compilers for the command-line options with popular compilers.

1.1.8. Atomic Operations

NVIDIA Grace CPU supports the Large System Extension (LSE), which was introduced in Armv8.1. LSE provides the following low-cost atomic operations, which can improve system throughput for CPU-to-CPU communication, locks, and mutexes:

- The Compare and Swap instructions, CAS, and CASP.
- Atomic memory operation instructions, LD<OP> and ST<OP>, where <OP> is ADD, CLR, EOR, SET, SMAX, SMIN, UMAX, or UMIN.
- The Swap procedure, SWP.

These instructions can operate on integer data. All compilers that support the Grace CPU automatically use these instructions in synchronization functions like GCC’s __atomic built-ins. When using LSE atomics instead of load/store exclusives, there is a huge improvement. For instance, a shared integer value can be incremented with one atomic ADD instead of the following sequence:

1. Load exclusive.
2. Add.
3. Attempt store exclusive.
4. If the operation fails, repeat the sequence.
1.1.9. Additional Armv9 Features

The Grace CPU implements multiple key features of the Armv9 portfolio that provide utilities in general purpose data center CPUs, including cryptographic acceleration, scalable profiling extension, virtualization extensions, and secure boot. In addition to standard Armv9 features, Grace also supports full-memory encryption.
2. Understanding Your Grace Machine

After you boot up your Grace machine, run `sudo ipmitool fru print` command and check the information about the NVIDIA Grace module.

Here is the sample output from a Grace CPU Superchip machine. The **FRU Device Description** is PG535, and the **Product Name** is C2.

```
FRU Device Description : PG535 (ID 192)
Board Mfg Date      : [REDACTED]
Board Mfg           : NVIDIA
Board Product       : PG535
Board Serial        : [REDACTED]
Board Part Number   : 699-2G535-0200-DV2
Product Manufacturer : NVIDIA
Product Name        : C2
Product Part Number : 900-2G535-0000-000
Product Version     : B-R00
Product Serial      : [REDACTED]
```

Here is the sample output from a Grace Hopper Superchip machine. The **FRU Device Description** is PG530, and the **Product Name** is GH200.

```
FRU Device Description : PG530 (ID 133)
Board Mfg Date      : [REDACTED]
Board Mfg           : NVIDIA
Board Product       : PG530
Board Serial        : [REDACTED]
Board Part Number   : 699-2G530-0206-QS1
Product Manufacturer : NVIDIA
Product Name        : GH200 480GB
Product Part Number : 900-2G530-0000-000
Product Version     : A-R00
Product Serial      : [REDACTED]
```
2.1. Checking the CPUs

The `lscpu` command-line utility in Linux gets CPU information about the system, fetches the CPU architecture information from the `sysfs` and `/proc/cpuinfo` files, and displays the information in a terminal.

After you boot your Grace machine, run the `lscpu` command and check the CPUs.

Here is the sample output from a Grace CPU Superchip machine:

<table>
<thead>
<tr>
<th>Architecture:</th>
<th>aarch64</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU op-mode(s):</td>
<td>64-bit</td>
</tr>
<tr>
<td>Byte Order:</td>
<td>Little Endian</td>
</tr>
<tr>
<td>CPU(s):</td>
<td>144</td>
</tr>
<tr>
<td>On-line CPU(s) list:</td>
<td>0-143</td>
</tr>
<tr>
<td>Vendor ID:</td>
<td>ARM</td>
</tr>
<tr>
<td>Model:</td>
<td>0</td>
</tr>
<tr>
<td>Thread(s) per core:</td>
<td>1</td>
</tr>
<tr>
<td>Core(s) per socket:</td>
<td>72</td>
</tr>
<tr>
<td>Socket(s):</td>
<td>2</td>
</tr>
<tr>
<td>Stepping:</td>
<td>r0p0</td>
</tr>
<tr>
<td>Frequency boost:</td>
<td>disabled</td>
</tr>
<tr>
<td>CPU max MHz:</td>
<td>3582.0000</td>
</tr>
<tr>
<td>CPU min MHz:</td>
<td>81.0000</td>
</tr>
<tr>
<td>BogoMIPS:</td>
<td>2000.00</td>
</tr>
<tr>
<td>Flags:</td>
<td>fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp a simdhp cpuid asimdrdm jscvt fcma 1rcpc dcpop sha3 sm3 sm4 asimddp sha512 sve asimdfhm dit uscat ilrcpc flagm ssbs sb pacg dcpods sve2 sveaes svepmull svebitperm svesh a3 svesm4 flagm2 print svei8mm svebf16 i8mm bf16 dgh bti</td>
</tr>
<tr>
<td>Caches (sum of all):</td>
<td></td>
</tr>
<tr>
<td>L1d:</td>
<td>9 MiB (144 instances)</td>
</tr>
<tr>
<td>L1i:</td>
<td>9 MiB (144 instances)</td>
</tr>
<tr>
<td>L2:</td>
<td>144 MiB (144 instances)</td>
</tr>
<tr>
<td>L3:</td>
<td>228 MiB (2 instances)</td>
</tr>
<tr>
<td>NUMA:</td>
<td></td>
</tr>
<tr>
<td>NUMA node(s):</td>
<td>2</td>
</tr>
<tr>
<td>NUMA node0 CPU(s):</td>
<td>0-71</td>
</tr>
<tr>
<td>NUMA node1 CPU(s):</td>
<td>72-143</td>
</tr>
<tr>
<td>Vulnerabilities:</td>
<td></td>
</tr>
<tr>
<td>Itlb multihit:</td>
<td>Not affected</td>
</tr>
<tr>
<td>L1tf:</td>
<td>Not affected</td>
</tr>
<tr>
<td>Mds:</td>
<td>Not affected</td>
</tr>
<tr>
<td>Meltdown:</td>
<td>Not affected</td>
</tr>
<tr>
<td>Mmio stale data:</td>
<td>Not affected</td>
</tr>
<tr>
<td>Retbleed:</td>
<td>Not affected</td>
</tr>
<tr>
<td>Spec store bypass:</td>
<td>Mitigation; Speculative Store Bypass disabled via prctl</td>
</tr>
<tr>
<td>Spectre v1:</td>
<td>Mitigation; __user pointer sanitization</td>
</tr>
<tr>
<td>Spectre v2:</td>
<td>Not affected</td>
</tr>
</tbody>
</table>
Srbds: Not affected
Tsx async abort: Not affected

From this output, you can see information such as the number of CPU sockets, how many cores per socket, how many hardware threads per core, and the max/min CPU frequency. You can also find the size of the L1, the L2, and the L3 caches.

Here is the sample output of a Grace Hopper Superchip system:

```
Architecture: aarch64
  CPU op-mode(s): 64-bit
  Byte Order: Little Endian
CPU(s):
  On-line CPU(s) list: 0-71
Vendor ID: ARM
  Model: 0
  Thread(s) per core: 1
  Core(s) per socket: 72
  Socket(s): 1
  Stepping: r0p0
  Frequency boost: disabled
  CPU max MHz: 3591.0000
  CPU min MHz: 81.0000
  BogoMIPS: 2000.00
  Flags:
    fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp
    asimdhp
    cpuid asimdrdm jscvt fcma lrcpc dcpop sha3 sm3 sm4
    asimddp sha512 sve asimdfhm dit uscat ilrcpc flagm ssbs sb pacac pacg
    dcpod
    p sve2 sveaes svepmull svebitperm svesha3 svesm4 flagm2
    frint s
    vei8mm svebf16 i8mm bf16 dgh bti
Caches (sum of all):
  L1d: 4.5 MiB (72 instances)
  L1i: 4.5 MiB (72 instances)
  L2: 72 MiB (72 instances)
  L3: 114 MiB (1 instance)
NUMA:
  NUMA node(s): 9
  NUMA node0 CPU(s): 0-71
  NUMA node1 CPU(s): 
  NUMA node2 CPU(s): 
  NUMA node3 CPU(s): 
  NUMA node4 CPU(s): 
  NUMA node5 CPU(s): 
  NUMA node6 CPU(s): 
  NUMA node7 CPU(s): 
  NUMA node8 CPU(s): 
Vulnerabilities:
  Itlb multihit: Not affected
```
L1tf: Not affected
Mds: Not affected
Meltdown: Not affected
Mmio stale data: Not affected
Retbleed: Not affected
Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl
Spectre v1: Mitigation; __user pointer sanitization
Spectre v2: Not affected
Srbds: Not affected
Tsx async abort: Not affected

Note: This output shows nine NUMA nodes. The first node corresponds to the Grace CPU, the second to the Hopper GPU, and the remaining seven nodes correspond to NVIDIA Multi-Instance GPU (MIG) instances.

The seven MIG instances can be ignored if MIG mode is not being used.

2.2. Checking the Non-Uniform Memory Access Settings

The lscpu output includes basic information about the Non-Uniform Memory Access (NUMA) settings on your Grace machine.

To understand more about the NUMA settings, run the numactl -H command, and here is the sample output from a Grace Superchip machine:

```
available: 2 nodes (0-1)
node 0 cpus:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
node 0 size: 245090 MB
node 0 free: 99633 MB
node 1 cpus:  72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
node 1 size: 245317 MB
node 1 free: 126895 MB
node distances:
  node 0 1
    0: 10 40
    1: 40 10
```

The output shows that there are two NUMA nodes on this machine, the number of cores on each NUMA node, and how much memory is available for each node. The output also shows the node distances between NUMA nodes, which helps the Kernel scheduler execute application threads on CPU cores that are closest to the memory resident data.
Here is the sample output from a Grace + Hopper Superchip system:

available: 9 nodes (0-8)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
node 0 size: 490310 MB
node 0 free: 166560 MB
node 1 cpus:
node 1 size: 95232 MB
node 1 free: 92094 MB
node 2 cpus:
node 2 size: 0 MB
node 2 free: 0 MB
node 3 cpus:
node 3 size: 0 MB
node 3 free: 0 MB
node 4 cpus:
node 4 size: 0 MB
node 4 free: 0 MB
node 5 cpus:
node 5 size: 0 MB
node 5 free: 0 MB
node 6 cpus:
node 6 size: 0 MB
node 6 free: 0 MB
node 7 cpus:
node 7 size: 0 MB
node 7 free: 0 MB
node 8 cpus:
node 8 size: 0 MB
node 8 free: 0 MB
node distances:
node 0 1 2 3 4 5 6 7 8
  0: 10 80 80 80 80 80 80 80
  1: 80 10 255 255 255 255 255 255
As noted in Checking the CPUs, the last seven NUMA nodes can be ignored if MIG is not used.

2.3. Checking the GPU

Running the nvidia-smi command displays the status of the GPU in the system.

Here is sample output from a Grace Hopper Superchip system:

```
+-----------------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.06       Driver Version: 535.104.06   CUDA Version: 12.2   |      |
|-----------------------------------------------------------------------------------------------+
| GPU Name   Persistence-M | Bus-Id   Disp.A | Volatile Uncorr. ECC |                  |
| Fan Temp   Perf    Pwr:Usage/Cap | Memory-Usage | GPU-Util Compute M. |                  |
|            |            |                  |                  |
|------------------------------------------+----------------+------------------|
| 0  GH200 480GB Off | 00000009:01:00.0 Off | 0 |
| N/A 29C P0 108W / 900W | 0MiB / 97871MiB | 8 | Default |
| | | Disabled |
+-----------------------------------------------------------------------------------------------+

+-----------------------------------------------------------------------------------------------+
<p>| Processes: |
| GPU GI CI PID Type Process name GPU Memory |</p>
<table>
<thead>
<tr>
<th>ID ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>No running processes found</td>
</tr>
</tbody>
</table>
+-----------------------------------------------------------------------------------------------+
2.4. Checking the Memory

One of the common ways of checking the memories on your Grace system is to run the `sudo dmidecode -t memory` command. Here is the sample output from a Grace-Grace machine:

```
# dmidecode 3.3
Getting SMBIOS data from sysfs.
SMBIOS 3.6.0 present.
# SMBIOS implementations newer than version 3.5.0 are not
# fully supported by this version of dmidecode.

Handle 0x000B, DMI type 16, 23 bytes
Physical Memory Array
  Location: System Board Or Motherboard
  Use: System Memory
  Error Correction Type: Single-bit ECC
  Maximum Capacity: 480 GB
  Error Information Handle: No Error
  Number Of Devices: 2

Handle 0x000C, DMI type 17, 92 bytes
Memory Device
  Array Handle: 0x000B
  Error Information Handle: 0x0000
  Total Width: 540 bits
  Data Width: 480 bits
  Size: 240 GB
  Form Factor: Die
  Set: None
  Locator: Not Specified
  Bank Locator: Not Specified
  Type: LPDDR5
  Type Detail: None
  Speed: Unknown
  Manufacturer: Not Specified
  Serial Number: 9223381974177924187
  Asset Tag: Not Specified
  Part Number: Not Specified
  Rank: 1
  Configured Memory Speed: Unknown
  Minimum Voltage: Unknown
  Maximum Voltage: Unknown
  Configured Voltage: Unknown
  Memory Technology: DRAM
  Memory Operating Mode Capability: None
  Firmware Version: Not Specified
  Module Manufacturer ID: Unknown
  Module Product ID: Unknown
  Memory Subsystem Controller Manufacturer ID: Unknown
```
Memory Subsystem Controller Product ID: Unknown
Non-Volatile Size: None
Volatile Size: None
Cache Size: None
Logical Size: None

Handle 0x000D, DMI type 17, 92 bytes
Memory Device
   Array Handle: 0x000B
   Error Information Handle: 0x0000
   Total Width: 540 bits
   Data Width: 480 bits
   Size: 240 GB
   Form Factor: Die
   Set: None
   Locator: Not Specified
   Bank Locator: Not Specified
   Type: LPDDR5
   Type Detail: None
   Speed: Unknown
   Manufacturer: Not Specified
   Serial Number: 9223382071351559259
   Asset Tag: Not Specified
   Part Number: Not Specified
   Rank: 1
   Configured Memory Speed: Unknown
   Minimum Voltage: Unknown
   Maximum Voltage: Unknown
   Configured Voltage: Unknown
   Memory Technology: DRAM
   Memory Operating Mode Capability: None
   Firmware Version: Not Specified
   Module Manufacturer ID: Unknown
   Module Product ID: Unknown
   Memory Subsystem Controller Manufacturer ID: Unknown
   Memory Subsystem Controller Product ID: Unknown
   Non-Volatile Size: None
   Volatile Size: None
   Cache Size: None
   Logical Size: None

You can see from the output that there are two zones of LPDDR5 memories, each with 240GB, and each zone is from one Grace chip.
3. Basic System Health Checks

To confirm that your system is healthy and is correctly configured, check the compute performance and memory bandwidth with some simple benchmarks.

3.1. STREAM

Use the STREAM benchmark to check LPDDR5X memory bandwidth. The following commands download and compile STREAM with a total memory footprint of approximately 2.7GB, which is sufficient to exceed the L3 cache without excessive runtime.

```
$ gcc -Ofast -mcpu=neoverse-v2 -fopenmp \
  -DSTREAM_ARRAY_SIZE=120000000 -DNTIMES=200 \
  -o stream_openmp.exe stream.c
```

To run STREAM, set the number of OpenMP threads (OMP_NUM_THREADS) according to the example below. Use OMP_PROC_BIND=spread to distribute the threads evenly over all available cores and maximize bandwidth.

```
$ OMP_NUM_THREADS={THREADS} OMP_PROC_BIND=spread ./stream_openmp.exe
```

System bandwidth is proportional to the memory capacity. Find your system’s memory capacity in the table below and use the given parameters to generate the expected score for STREAM TRIAD. For example, when running on a Grace-Hopper superchip with a memory capacity of 120GB, this command will score at least 450GB/s in STREAM TRIAD:

```
$ OMP_NUM_THREADS=72 OMP_PROC_BIND=spread ./stream_openmp.exe
```
Similarly, this command will score at least 900GB/s in STREAM TRIAD on a Grace CPU Superchip with a memory capacity of 240GB:

```
$ OMP_NUM_THREADS=144 OMP_PROC_BIND=spread numactl -m0,1 ./stream_openmp.exe
```

Table 3-1. Expected STREAM TRIAD Scores

<table>
<thead>
<tr>
<th>Superchip</th>
<th>Capacity (GB)</th>
<th>OMP_NUM_THREADS</th>
<th>Expected TRIAD Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grace-Hopper</td>
<td>120</td>
<td>72</td>
<td>450+</td>
</tr>
<tr>
<td>Grace-Hopper</td>
<td>480</td>
<td>72</td>
<td>340+</td>
</tr>
<tr>
<td>Grace CPU</td>
<td>240</td>
<td>144</td>
<td>900+</td>
</tr>
<tr>
<td>Grace CPU</td>
<td>480</td>
<td>144</td>
<td>900+</td>
</tr>
<tr>
<td>Grace CPU</td>
<td>960</td>
<td>144</td>
<td>680+</td>
</tr>
</tbody>
</table>

```
$ OMP_NUM_THREADS=72 OMP_PROC_BIND=spread numactl -m0,1 ./stream_openmp.exe
```

STREAM version $Revision: 5.10 $

This system uses 8 bytes per array element.

Array size = 120000000 (elements), Offset = 0 (elements)
Memory per array = 915.5 MiB (= 0.9 GiB).
Total memory required = 2746.6 MiB (= 2.7 GiB).
Each kernel will be executed 200 times.
The *best* time for each kernel (excluding the first iteration)
will be used to compute the reported bandwidth.

Number of Threads requested = 72
Number of Threads counted = 72

Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 2927 microseconds.
 (= 2927 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.

WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.

<table>
<thead>
<tr>
<th>Function</th>
<th>Best Rate MB/s</th>
<th>Avg time</th>
<th>Min time</th>
<th>Max time</th>
</tr>
</thead>
</table>

NVIDIA Grace Performance Tuning Guide
3.2. Fused Multiply Add

NVIDIA provides an open source suite of benchmarking microkernels for Arm CPUs. To allow precise counts of instructions and exercise specific functional units, these kernels are written in assembly language. To measure the peak floating point capability of a core and check the CPU clock speed, use a Fused Multiply Add (FMA) kernel.

To measure achievable peak performance of a core, the fp64_sve_pred_fmla kernel executes a known number of SVE predicated fused multiply-add operations (FMLA).

When combined with the perf tool, you can measure the performance and the core clock speed.

```bash
$ git clone https://github.com/NVIDIA/arm-kernels.git
$ cd arm-kernels
$ make
$ perf stat ./arithmetic/fp64_sve_pred_fmla.x
```

The benchmark score is reported in giga-operations per second (Gop/sec) near the top of the benchmark output. Grace can perform 16 FP64 FMA operations per cycle, so a Grace CPU with a nominal CPU frequency of 3.3GHz should report between 52 and 53 Gop/sec. The CPU frequency is reported in the perf output on the `cycles` line and after the `#` symbol.

Here is an example of the fp64_sve_pred_fmla.x execution output:

```bash
$ perf stat ./arithmetic/fp64_sve_pred_fmla.x
4( 16(SVE_FMLA_64b) );
Iterations;10000000
Total Inst;6400000000
Total Ops;25600000000
Inst/Iter;64
Ops/Iter;256
Seconds;0.481267
G0ps/sec;53.1929
```

Performance counter stats for `./arithmetic/fp64_sve_pred_fmla.x`:

```
 482.25 msec task-clock      # 0.996 CPUs utilized
     0 context-switches      # 0.000 /sec
     0 cpu-migrations       # 0.000 /sec
```
3.3. C2C CPU-GPU Bandwidth

NVIDIA provides an open-source benchmark, similar to STREAM, that is designed to test the bandwidth between various memory units on the system. This can be used to test the bandwidth provided by NVLink C2C between the CPU and GPU of a Grace Hopper Superchip.

Download, build, and run nvbandwidth:

```bash
git clone https://github.com/NVIDIA/nvbandwidth.git
cd nvbandwidth
# may need to update version of CUDA
docker run -it --rm --gpus all -v $(pwd):/nvbandwidth nvidia/cuda:12.2.0-devel-ubuntu22.04

# within docker
cd /nvbandwidth
apt update
apt install libboost-program-options-dev
./debian_install.sh
./nvbandwidth -t 0

# next test
./nvbandwidth -t 1

# all tests can be listed with ./nvbandwidth -l
```
Here is the output from the previous two commands on a sample system:

```
# ./nvbandwidth -t 0
nvbandwidth Version: v0.2
Built from Git version:

NOTE: This tool reports current measured bandwidth on your system. Additional system-specific tuning may be required to achieve maximal peak bandwidth.

CUDA Runtime Version: 12020
CUDA Driver Version: 12020
Driver Version: 535.82

Device 0: GH200 120GB

Running host_to_device_memcpy_ce.
memcpy CE CPU(row) -> GPU(column) bandwidth (GB/s)
   0
  0   416.34

SUM host_to_device_memcpy_ce 416.34
```

```
# ./nvbandwidth -t 1
nvbandwidth Version: v0.2
Built from Git version:

NOTE: This tool reports current measured bandwidth on your system. Additional system-specific tuning may be required to achieve maximal peak bandwidth.
```
CUDA Runtime Version: 12020
CUDA Driver Version: 12020
Driver Version: 535.82

Device 0: GH200 120GB

Running device_to_host_memcpy_ce.
memcpy CE CPU(row) <- GPU(column) bandwidth (GB/s)

0
0 295.47

SUM device_to_host_memcpy_ce 295.47

For memory copies that use CUDA copy engines (CEs), you should expect similar numbers as shown in the output for systems with 120GB or 240GB of LPDDR5 memory. Systems with 480GB LPDDR5 memory might have a lower bandwidth for host-to-device copies (compared to the first test output shown above). On a healthy system, this bandwidth should be approximately 350-360 GB/s.

Systems with 480GB LPDDR5 should have similar device-to-host bandwidth as shown above in the second test, except Grace-Hopper x4 systems, where this bandwidth should be approximately 170 GB/s due to more CEs being reserved for saturating NVLink bandwidth between GPUs.

To run bandwidth tests using the GPU’s streaming micro-processors (SMs), run the ./nvbandwidth -l command for the exact test numbers. The achieved bandwidth should be at least as large as the outputs shown by CE-based tests.
4. Power and Thermals

This chapter provides information about CPU power and thermal management settings.

4.1. C-States

C-States refer to idle CPU power states, and Grace includes the following C-states:

- **C0**: active/run state.
  
  This is the state of the CPU core while active.

- **C1**: clock gated state.
  
  This state is entered when WFI/WFE instructions are executed by the CPU core. The latency to enter/exit this state is negligible.

The LPI table in Advanced Configuration and Power Interface (ACPI) provides information about the C-states to any CPU idle governors such as the cpu idle framework in Linux.

For systems that have the cpu idle governors enabled, the number of times the C1 state is entered through the idle framework can be read by running the following command:

```
$ cat /sys/devices/system/cpu/cpu<n>/cpuidle/state1/usage
```

For systems that do not use cpu idle governors, the cpu cores can still enter clock gated state when WFI/WFE instructions are executed, but no stats will be available.

4.2. P-States

P-States refers to performance states, and Grace does not offer explicit P-states. Instead, Grace exposes the maximum and minimum performance capabilities through ACPI's CPPC mechanism. CPPC offers users and operating systems the ability to request any performance in the allowed bounds rather than discrete P-State. Refer to [CPU Performance and Frequency Management](#) for more information.
4.3. CPU Performance and Frequency Management

Each CPU core can operate at its own independent frequency, and the frequency is determined by the frequency policy governors that were used. Linux provides the following frequency governors:

- **Performance governor**: Geared towards getting the maximum performance and sets the performance/frequency request of the CPU cores to the maximum possible value.

  The request is not based on activity and kept fixed at highest value.

- **Userspace governor**: Bypasses a kernel governor and provides control to the userspace application for frequency control.

  To set the frequency of the cores, a hypervisor, or a higher level software entity, can take input from an application. The kernel does not modify the frequency based on other information but will honor frequency caps based on thermals.

- **Schedutil governor**: Incorporates information from the scheduler, which are the threads that are currently scheduled on cores, the activity on the core, load estimation, and so on, to determine the optimal frequency for the core.

  The goal of this governor is to provide best performance while saving power by matching the frequency based on scheduler visible workloads.

- **Ondemand governor**: Adjusts the frequency based on the trailing load of the CPU core.

  This governor predicts the future load and ramps frequency accordingly.


The default frequency governor in Grace is the performance governor, which sets the frequency to the maximum value for that core. The maximum frequency usually corresponds to the maximum possible performance and is higher than the frequency at which nominal (sustained) performance can be achieved. When running at the maximum frequency violates thermals, the thermal management solution throttles frequency. Refer to [Power and Thermal Management](https://www.nvidia.com) for more information.

Here are the CPU frequency settings:

- Software frequency request (kHz) for the CPU core `<n>`.
  ```
  $ cat /sys/devices/system/cpu/cpu<n>/cpufreq/scaling_cur_freq
  ```

- Measured frequency (kHz).
  ```
  $ cat /sys/devices/system/cpu/cpu<n>/cpufreq/cpuinfo_cur_freq
  ```
Note: This makes use of AMU (actmon), which is provided by ARM, where the source and reference clocks are measured, and where the ratio is used to compute the actual frequency. With the default measuring window used in Linux, there might be up to a 3% error in the frequency read. To increase accuracy, the measurement window should be increased in upstream Linux code.

4.4. GPU and Module Power Management

GPU provides power capping at the following scopes:

- Limit power consumption of the Grace + Hopper superchip (Module) and keep it within the provided power limit.
- Limit power consumption of the GPU and keep it within the provided power limit.

**Figure 4-1: GPU and Module Power Management**

This is done by Automatic power steering in the GPU because the GPU monitors power telemetry for Grace, Module, and the GPU. Power capping at the Module scope works on monitoring the consumed Grace power, removing that from the Module power limit, and giving the rest to the GPU.

The GPU can work within the new power limit or can stick to the limit that was explicitly set for the GPU, where the lower of the two limits is respected. This leads to efficiently balancing power between Grace and GPU to improve overall app perf by opportunistically boosting the GoPU power budget. The GPU achieves power capping by using DVFS.
Here is the output from the NVSMI log:

```
nvidia@localhost:~$ nvidia-smi -q -d POWER

==============NVSMI LOG==============
Timestamp : Fri Oct 6 22:46:55 2023
Driver Version : 535.122
CUDA Version : 12.2

Attached GPUs : 1
GPU 00000009:01:00.0
  GPU Power Readings
    Power Draw : 77.61 W
    Current Power Limit : 900.00 W
    Requested Power Limit : 900.00 W
    Default Power Limit : 900.00 W
    Min Power Limit : 100.00 W
    Max Power Limit : 900.00 W

  Power Samples
    Duration : 2.36 sec
    Number of Samples : 119
    Max : 78.26 W
    Min : 76.65 W
    Avg : 77.48 W

  Module Power Readings
    Power Draw : 147.49 W
    Current Power Limit : 1000.00 W
    Requested Power Limit : 1000.00 W
    Default Power Limit : 1000.00 W
    Min Power Limit : 200.00 W
    Max Power Limit : 1000.00 W
```
4.5. Power and Thermal Management

Grace provides the following types of thermal management types:

- Limit power consumption and keep it within the provided power limit.
- Thermal sensor (Tj)-based management.

4.6. Power Telemetry

This section provides information about power telemetries for Grace, and guidance for comparing Grace power telemetry to Intel and AMD power telemetry. This can be useful when making comparisons in power efficiency to other CPU architectures.

4.6.1. Grace Power Telemetry

Grace exposes power telemetry through hwmon, which uses the ACPI power meter interface. You can read the power telemetry information in one of the following ways:

- To display the name of the power meter.
  
  This gives information about which power is being reported on hwmon node X.
  
  `cat /sys/class/hwmon/hwmonX/device/power1_oem_info`

- To display power consumption, which is average power over past 50ms interval by default, on hwmon node X:
  
  `cat /sys/class/hwmon/hwmonX/device/power1_average`

- To display the power stats interval in milliseconds, on hwmon node X:
  
  `cat /sys/class/hwmon/hwmonX/device/power1_average_interval`

- To change the power stats interval in milliseconds, on hwmon node X (default is 50):
  
  `echo <value> | sudo tee /sys/class/hwmon/hwmonX/device/power1_average_interval`

Table 4-2 provides information about the available power telemetry.

<table>
<thead>
<tr>
<th>System</th>
<th>Telemetry</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grace Superchip</td>
<td>Grace Power Socket 0</td>
<td>Total power of the socket 0, including DRAM power and regulator loss.</td>
</tr>
<tr>
<td>System</td>
<td>Telemetry</td>
<td>Details</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>CPU Power Socket 0</td>
<td></td>
<td>CPU rail power for socket 0.</td>
</tr>
<tr>
<td>SysIO Power Socket 0</td>
<td></td>
<td>SOC rail power.</td>
</tr>
<tr>
<td>Grace Power Socket 1</td>
<td></td>
<td>Total power of the socket 1, including DRAM power and regulator loss.</td>
</tr>
<tr>
<td>CPU Power Socket 1</td>
<td></td>
<td>CPU rail power for socket 1.</td>
</tr>
<tr>
<td>SysIO Power Socket 1</td>
<td></td>
<td>SOC rail power.</td>
</tr>
<tr>
<td>Grace Hopper Superchip</td>
<td>Module Power Socket 0</td>
<td>Total power of the CG1 module, including DRAM power and regulator loss. This also includes GPU and GPU HBM Power.</td>
</tr>
<tr>
<td>Grace Power Socket 0</td>
<td></td>
<td>Power of Grace socket.</td>
</tr>
<tr>
<td>CPU Power Socket 0</td>
<td></td>
<td>CPU rail power.</td>
</tr>
<tr>
<td>SysIO Power Socket 0</td>
<td></td>
<td>SOC rail power.</td>
</tr>
</tbody>
</table>

**Figure 4-2. Grace Power Telemetry Sensors**

Grace Hopper Superchip Telemetry

![Grace Power Telemetry Diagram](image)

**Legend**

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Total module power</td>
</tr>
<tr>
<td>1</td>
<td>Grace Power, including DRAM and power for all regulators</td>
</tr>
<tr>
<td>2</td>
<td>CPU Power, including regulator power</td>
</tr>
<tr>
<td>3</td>
<td>SysIO Power, including regulator power</td>
</tr>
</tbody>
</table>
As noted in Table 4-2, the total power reported for the socket includes CPU power, DRAM power, and regulator loss. Similarly, power is reported for the CPU cores and includes regulator losses.

Regulator loss accounts for 15% of the TDP power limit.

DRAM power can be estimated based on total traffic using the formula in Table 4-3.

**Table 4-3. Estimating DRAM Power**

<table>
<thead>
<tr>
<th>Config</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>128GB, 4266MHz</td>
<td><code>0.00000136</code></td>
<td><code>28.9</code></td>
<td><code>2334</code></td>
</tr>
<tr>
<td>128GB, 3200MHz</td>
<td><code>0.00000175</code></td>
<td><code>28.3</code></td>
<td><code>2043</code></td>
</tr>
</tbody>
</table>
### 4.6.2. Comparing Grace and Intel® Power Telemetry

Intel® CPUs expose power telemetry through the Intel® Performance Counter Monitor (Intel® PCM) APIs. When the power consumption of Grace CPUs is compared to Intel® CPUs, these APIs can be used to gather comparable metrics.

The PCM APIs are available on GitHub at [https://github.com/intel/pcm](https://github.com/intel/pcm).

Refer to the following documentation for more information:

- [Building from the source](https://github.com/intel/pcm#building-pcm-tools)
- [Installing precompiled binaries](https://github.com/intel/pcm#downloading-pre-compiled-pcm-tools)

PCM’s pcm-power utility can be run to collect performance metrics for a number of intervals and the duration per interval. For example, to capture one minute of samples at one-second intervals, run the following command:

```
sudo pcm-power 1.00 -i=60 -silent
```

For each interval, pcm-power prints the power consumption for each socket (S0, S1) including CPU power consumption and and DRAM power consumption:

```
$ sudo pcm-power -silent 1.0 -i=60 | grep '^S.; Consumed'
S0; Consumed energy units: 3563683; Consumed Joules: 217.51; Watts: 217.51
S0; Consumed DRAM energy units: 533250; Consumed DRAM Joules: 32.55; DRAM Watts: 32.55
S1; Consumed energy units: 3350361; Consumed Joules: 204.49; Watts: 204.49
S1; Consumed DRAM energy units: 597938; Consumed DRAM Joules: 36.50; DRAM Watts: 36.50
```

The DRAM bandwidth can be determined using the PMU metrics described in [Grace CPU Performance Metrics](#).
As illustrated in Figure 4-4, the power consumption per socket does not include regulator losses, and so is not directly comparable to the CPU Power Socket 0 and CPU Power Socket 1 telemetry as described in Table 4-2. To compare Intel CPU power consumption to Grace CPU power, remove the Grace regulator losses.

For more information about Power consumption metrics that are available through the Linux powercap kernel interface in sysfs, go to Power Capping Framework.

To measure power consumption for cores only, excluding regulator losses or DRAM power consumption, the metrics per CPU are available at:

For CPU 0:
```
cat /sys/class/powercap/intel-rapl/intel-rapl:0/intel-rapl:0:0/energy_uj
```

For CPU 1:
```
cat /sys/class/powercap/intel-rapl/intel-rapl:1/intel-rapl:1:0/energy_uj
```
These counters provide a running total of the microjoules consumed for each CPU. Measurements from this interface are comparable to the CPU Power Socket 0 and CPU Power Socket 1 telemetry as described in Table 4-2.

4.6.3. Comparing Grace and AMD Power Telemetry

AMD’s AMD μProf package includes utilities that provide power telemetry. When you compare the power consumption of Grace CPUs to AMD CPUs, these APIs can be used to gather comparable metrics.

To download and install the AMD μProf, go to https://www.amd.com/en/developer/uprof.html.

Refer to the AMD μProf User Guide for platform-specific information about the available metrics.

To capture measurements of power consumption per socket for 60 seconds with measurements at one-second intervals, run the following command:

```
AMDuProfCLI-bin timechart --event socket=0-1,power --interval 1000 --duration 60 -o powerOutput
```

The resulting output file, in a CSV format, will be reported in the command output, for example:

```
Live Profile Output file : /home/nvex/powerOutput/AMDuProf-SWP-Timechart_Aug-05-2023_00-03-29/timechart.csv
```

It contains CSV-formatted power measurements per interval, with one column per socket, for example:

```
RecordId, Timestamp, socket0-package-power, socket1-package-power
1,0:3:30:462, 95.56, 91.05
2,0:3:31:462, 95.09, 90.63
3,0:3:32:462, 95.17, 90.23
4,0:3:33:462, 95.70, 90.70
```
Guidance for measuring power usage for the DRAM with AMD processors depends on platform implementation details. Contact your platform vendor for guidance about measuring power usage for comparison to the measurements LPDDR5x power readings with Grace.

AMD μProf also allows per-core power utilization measurements to be captured. For example, on a 64 core AMD processor:

```
AMDuProfCLI-bin timechart --event core=0-63,power --interval 1000 --duration 60 -o power0output
```
The resulting file contains CSV-formatted power measurements per interval, with one column per core. These cores are summed to get the total power output across all cores to determine total power consumption by the CPU. As illustrated in Figure 4-5, this measurement does not include regulator losses. When regulator losses are removed from the Grace CPU measurement the total is comparable to the Grace CPU rail power CPU Power Socket 0 or CPU Power Socket 1 telemetries as described in Table 4-2.

4.7. Power Capping

Power capping limits average power consumption and is usually set based on the thermal power dissipation capability of the system. Grace throttles power when average power exceeds this limit. Users can reduce the power limit lower than the default value that was set in the BIOS. This setting is exposed through the Hwmon nodes and can be applied to total socket power. Power capping can be applied only at the socket level and not at the vdd_cpu or vdd_soc power levels.

To set power limit, run the following command:

```bash
echo <power value in micro Watts> > /sys/class/hwmon/hwmonX/device/power1_cap
```

For a Grace Hopper Superchip system, the capping power of the Grace CPU allows the Hopper GPU to draw more power, which can improve performance of GPU-heavy applications.

Power capping of the GPU can be applied according to GPU and Module Power Management.

4.8. CPU Temperature Management

ACPI thermal management (Tj) uses telemetry from temperature sensors to ensure that no local hotspots exceed the operating temperature. Power capping ensures that the average power of the socket/module is at or below the thermal capacity of the system. However, this does not account for asymmetric power distribution based on workload distribution across the cores.

ACPI tables provide passive and critical temperature limits, and the thermal governor tries to throttle CPUs to maintain temperature at or below the passive temperature limits. If the temperature exceeds the critical temperature limit, a shutdown is initiated.

To read the critical and passive trip points used for ACPI software throttling:

```bash
cat /sys/class/thermal/thermal_zone*/trip_point_0_type
cat /sys/class/thermal/thermal_zone*/trip_point_0_temp
cat /sys/class/thermal/thermal_zone*/trip_point_1_type
cat /sys/class/thermal/thermal_zone*/trip_point_1_temp
```
To modify these values, update the ACPI table.

| Caution: We recommend that you do not change the default values. If the passive trip point is lowered, throttling might occur more often, which affects the performance. If the passive trip point is increased, the software might not always settle at the temperature, which leads to more aggressive hardware throttling, and can reduce performance. |

4.9. GPU Temperatures

For GPU temperature using nvidia-smi, to get the temperature output, run the nvidia-smi -q -d TEMPERATURE command.

This step gets the current temperature and the temperature-related limits.
5. Operating System Settings

This chapter provides information about the operating system settings.

5.1. Page Size

Grace supports 64K and 4K Linux kernel page sizes. To configure your Linux kernel with the page size that suits your business needs, change the following `config` settings during the kernel compilation:

- 4K page size: `CONFIG_ARM64_4K_PAGES=y`
- 64K page size: `CONFIG_ARM64_64K_PAGES=y`

The 64K page size can benefit the applications that allocate a large amount of memory because there will be fewer page faults, better TLB hits, and efficiency.

**Note:** The recommended default value for the page size is 64K.

5.2. Huge Pages

Huge pages might be beneficial to applications that allocate large chunks of memories, and the main benefit is fewer TLB misses.

You can use huge pages on Grace systems in the following ways:

- Transparent Huge Pages (THP)
  - Transparent to the application.
  - Mostly automatic with a few available kernel tuning parameters.
  - When using the recommended 64 KB page size, THP pages are currently too large for practical use in most applications (refer to [Transparent Huge Pages](#) for more information).
● Hugetlbfs
  ○ Does not suffer from fragmentation concerns or from allocation latency because the huge pages are pre-allocated and indivisible.
  ○ Requires application modification.
  ○ Requires sysadmin setup.

5.2.1.  Transparent Huge Pages

THP is completely transparent to applications, and applications can get the benefit of huge pages without changing their source code (refer to Transparent Hugepage Support for more information). As of kernel version 6.5, only 512MB THP pages are supported when a 64KB system page size is configured. The 512MB THP is too large to be practical for most applications.

Refer to Transparent Hugepage Support for more information about THP.

Note: The default huge page size is related to the kernel page size (refer to HugeTLBpage on ARM64 for more information).

5.2.2.  Proactive Compaction

Proactive Compaction reduces allocation latency of huge pages by preemptively performing the work in the background. Proactive compaction does not change the probability of obtaining a huge page, but it changes how fast you can get one.

Without compaction, the kernel will return huge pages until it runs out of them. The application will then experience a perf cliff because the kernel is going to defragment the memory, and Proactive compaction smooths this out this process.

With compaction, when the applications start hitting a threshold of memory fragmentation, the kernel begins to defragment the memory pages in the background with anticipation of avoiding running out of huge pages and hitting a performance cliff.

The proactive compaction exposes a tunable, /proc/sys/vm/compaction_proactiveness, which accepts values in the [0, 100] range, and a default value of 20. This tunable determines how aggressively the kernel should compact memory in the background and setting an aggressive value can lead to increased address translation latency. The default value of 20 is reasonable and should only be changed based on perf data.

To limit the overhead of proactive compaction, you can use the on-demand compaction method, which is available only after CONFIG_COMPACTION is set. When 1 is written to the /proc/sys/vm/compact_memory file, all zones are compacted, and free memory is available in contiguous blocks where possible. This can be important, for example, when allocating
huge pages, because it will also directly compact memory as required. Refer to Documentation for /proc/sys/vm/ for more information.

5.2.3. Hugetlbfs

By using hugetlbfs, pools of hugetlb pages can be preallocated, and applications can use the huge pages in these pools. However, this requires changes in applications.

You can specify the minimum number of huge pages that are reserved by the system and how big the pool can grow. When huge pages are reserved, they cannot be used for other allocations, so the normal use of malloc might trigger an Out-Of-Memory, and the kernel might kill some processes. The allocation might also fail at runtime because of memory fragmentation, so these pages can also be reserved when you boot.

The benefit of reserving a pool of huge pages at boot time is that at boot time, the memory is not fragmented, so there is a greater chance that the requested number of huge pages can be assembled.

Refer to HugeTLB Pages for more information.

5.3. Configuring Linux Perf

Refer to Configuring Perf for more information.

5.4. Performance Governor

You can set the CPU governor using the cpupower command. For example, to set the CPU governor to Performance, run the following command:

```
sudo cpupower frequency-set -g performance
```

5.5. Init on Alloc

The CONFIG_INIT_ON_ALLOC_DEFAULT_ON kernel configuration option controls whether the kernel will fill newly allocated pages and heap objects with zeroes by default. You can overwrite this setting with the init_on_alloc=[0|1] kernel parameter.

On coherent systems, such as Grace Hopper, where GPU memory is exposed as system memory, this can cause heavy performance impacts to cudaMemcpy() operations.

Note: The recommended default value on GH is the init_on_alloc=0 parameter.
Not all distros will set the CONFIG_INIT_ON_ALLOC_DEFAULT_ON config on their kernels. For example, the SUSE and RHEL kernels do not currently set this option, but the Ubuntu -generic kernel does set this option.

The current value of the init_on_alloc kernel configuration option on a system might be printed as follows:

cat /proc/cmdline | grep init_on_alloc

which should provide output like the following:

```
BOOT_IMAGE=/boot/vmlinuz-6.2.0-1010-nvidia-64k
root=UUID=7123054d-9b18-4c3d-8844-c538c751b59a ro rd.driver.blacklist=nouveau
nouveau.modeset=0 earlycon module.blacklist=nouveau acpi.power_meter.force_cap_on=y
numa_balancing=disable init_on_alloc=0 preempt=none
```

### 5.6. Input-Output Memory Management Unit Passthrough

The Input-Output Memory Management Unit (IOMMU) is a hardware component that performs address translation from I/O device virtual addresses (also called I/O virtual address (IOVA)) to physical addresses. Different platforms have different IOMMUs, such as the Intel IOMMU graphics address remapping table (GART) that is used by PCI Express graphics cards, and System Memory Management Unit (SMMU) that is used by the ARM platform.

Linux provides the iommu.passthrough mode, and you can configure the DMA to use (or not use) the IOMMU to access the memory for addressing. This release requires that SMMU passthrough NOT be enabled. Future kernel releases will change that guidance, but for now we cannot run CUDA programs with SMMU in passthrough mode.

Setting iommu.passthrough to 1 on the kernel command line bypasses the IOMMU translation for DMA and setting it to 0 uses IOMMU translation for DMA. This value needs to be set at deployment (in the kernel configuration) or by editing the appropriate grub configuration files. For the changes to take effect, you need to reboot the system.

To add kernel parameters, complete the steps for your distro:

**Ubuntu**

1. Create the `/etc/default/grub.d/iommu_passthrough.cfg` file with the following contents:

   ```
   GRUB_CMDLINE_LINUX="$GRUB_CMDLINE_LINUX iommu.passthrough=0"
   ```

2. Run the following commands:

   ```
   sudo update-grub
   ```
RedHat
1. Run the following commands:

```
sudo grubby --update-kernel=ALL --args="iommu.passthrough=0"
sudo reboot
```

SUSE
1. Edit the `/etc/default/grub` file.
2. On the line that contains the `GRUB_CMDLINE_LINUX` string, append the `iommu.passthrough=0` parameter, and run the following commands:

```
sudo update-bootloader --refresh
sudo reboot
```

5.7. Automatic NUMA Scheduling and Balancing

When using a Grace Hopper system, we recommend that you **do not** use Automatic NUMA Scheduling and Balancing (AutoNUMA) features of the Linux kernel. This is because of the additional page-faults that are introduced by AutoNUMA, which can significantly hurt GPU-heavy application performance.

- To see the status of AutoNUMA, use `cat /proc/sys/kernel/numa_balancing`.
- If the output is 1, AutoNUMA is **enabled**, if it is 0, it is **disabled**.
- To disable AutoNUMA in a session, use `echo 0 > /proc/sys/kernel/numa_balancing`.
- To disable AutoNUMA permanently, use `echo "kernel.numa_balancing = 0" >> /etc/sysctl.conf`.

5.8. Swap File Size

This section applies **only** to Grace Hopper systems.

If an application allocates a large enough fraction of CPU memory, the kernel might decide to migrate some pages, possibly from third-party applications, from CPU memory to GPU memory. Currently, this memory can only be reclaimed through a swap file. We recommend that you have a large enough swap file for these scenarios.
Note: On a Grace Hopper system, we recommend using a swap file of at least \( \frac{1}{4} \) to \( \frac{1}{2} \) the aggregate GPU memory size in the system.
6. Optimizing IO Performance

6.1. Networking

We recommend that you download the latest driver and firmware for your network adapter. Before making any changes, contact your network adapter’s vendor for information about whether the tuning options in this guide are applicable.

6.1.1. NUMA Node

Always ensure that you use local CPU and memory that are in the same NUMA domain as your network adapter.

To check your network adapter’s NUMA domain, run following commands:

```
cat /sys/class/net/<ethernet interface>/device/numa_node

cat /sys/class/net/<ethernet interface>/device/local_cpulist
```

6.1.2. IRQ Balance

The operating system typically distributes the interrupts among all CPU cores in a multi-processor system, but this can cause delayed interrupt processing.

To disable this on Linux, run the following command:

```
sudo systemctl disable irqbalance
```

6.1.3. Configuring Interrupt Handling

A channel in a network adapter is an IRQ and a set of queues that can trigger that IRQ. Typically, you do not want more interrupt queues than the number of cores in the system, so control the number of interrupt queues in a NUMA domain.

To set the number of channels:

**Before you begin,** stop the `irqbalance` service.

1. Check the current settings with the following command:

   `ethtool -l <adapter>`

2. It tells you the current setting of various queue types.
3. Set the number of channels, for example:
   
   ```bash
   sudo ethtool -L <adapter> combined 16 tx 0 rx 0
   ```

4. To receive and to transmit (combined), set the receive queue (rx), the transmit queue (tx), or a combined queue of both types.

5. Contact your vendor for information.

   For NVIDIA Mellanox network adapters, to set the appropriate interrupt handling masks, invoke the following script:
   
   ```bash
   sudo set_irq_affinity.sh <adapter>
   ```

   This script comes with a MOFED installation.

   **6.1.4. TX/RX Queue Size**

   The NIC’s queue size dictates how many ring buffers are allocated for DMA transfer. To help prevent package drops, we recommend that you set the size to the maximum allowed value. You can also set it to a value that works best for your use case.

   To query the current setting of the queue size:
   
   ```bash
   ethtool -g enp1s0
   ```

   **Ring parameters for ibp1s0:**
   
   Pre-set maximums:
   
   - **RX:** 8192
   - **RX Mini:** n/a
   - **RX Jumbo:** n/a
   - **TX:** 8192
   
   Current hardware settings:
   
   - **RX:** 512
   - **RX Mini:** n/a
   - **RX Jumbo:** n/a
   - **TX:** 1024

   To set the queue size of a NIC:
   
   ```bash
   sudo ethtool -G <adapter> rx <value> tx <value>
   ```

   **6.1.5. Large Receive Offload**

   Depending on your use case, you can optimize for max throughput or best latency, but rarely both. Enabling Large Receive Offload (LRO) is a typical setting to optimize for maximum network throughput, but it might negatively affect the network latency. Contact your network adapter vendors for more information about whether LRO is supported and the best practices for usage.

   To enable/disable LRO:
   
   ```bash
   sudo ethtool lro <on|off>
   ```
6.1.6. **MTU**

We recommend that you set the network adapter’s MTU to jumbo frame (9000) when you bring up the network interface:

```
sudo ifconfig <adapter> <IP_address> netmask <network_mask> mtu 9000 up
```

To check the current settings, here is a sample command you can run:

```
ifconfig <adapter> | grep mtu
```

6.1.7. **MAX_ACC_OUT_READ**

This setting is NVIDIA Mellanox-specific, and here are recommended values for the following NICs:

- ConnextX-6: 44
- ConnectX-7: 128

To check the current settings:

```
sudo mlxconfig -d <dev> query | grep MAX_ACC_OUT_READ
```

To set this setting to the recommended value:

1. Run the following commands:
   ```
   sudo mlxconfig -d <dev> set ADVANCED_PCI_SETTINGS=1
   sudo mlxconfig -d <dev> set MAX_ACC_OUT_READ=<value>
   ```

2. For this setting to take effect, reboot the system.

6.1.8. **PCIe Max Read Request**

This setting is also NVIDIA Mellanox specific and can be applied to other network adapters.

⚠️ **Note:** Ensure that you set the MRRS to an appropriate value as recommended by your vendor.

Here is an example that shows you how to set the MRRS of an NVIDIA Mellanox NIC to 4096:

```
sudo setpci -v -d <dev> cap_exp+8.w=5000:7000
```

This setting does not persist after the system reboots.
6.1.9. Relaxed Ordering

Setting the PCIe ordering to relaxed for the network adapter sometimes results in better performance. There are different ways to enable relaxed ordering on the network adapter. Contact your vendor for more information.

Here is a sample command to check relaxed ordering on NVIDIA Mellanox NICs. For this command to work, set ADVANCED_PCI_SETTINGS to True (refer to MAX_ACK_OUT_READ for more information).

```
sudo mlxconfig -d <dev> query | grep PCI_WR_ORDERING
PCI_WR_ORDERING per_mkey(0)
```

A value of 0 means that this setting is disabled.

1. To enable relaxed ordering:
   ```
sudo mlxconfig -d <dev> set PCI_WR_ORDERING=1
   ```
2. Reboot the system.

6.1.10. 10b PCIe tags

Ideally, the PCIe endpoint should use 10b PCIe tags to ensure that it can issue a large number of read requests to hide high read latencies when the system is busy. Contact your endpoint’s vendor for more information.

Here is an example for ConnectX-7:

```
setpci -s <bus> -v cap_exp+28.w 1000
```

If bit 12 is 1, then 10b tags are enabled.
If not, set bit 12. The drivers for IB should be unloaded first. Example:

```
systemctl stop openibd
setpci -s <bus> -v cap_exp+28.w 0040
setpci -s <bus> -v cap_exp+28.w=1040:1040
systemctl start openibd
```

6.2. Storage/Filesystem

This section provides information about performance tunings that are related to storage and the filesystem.

6.2.1. Drop Page Cache

When files are read from storage into memories on a Linux system, they are cached in unused memory areas called page cache. There are times you might want to drop the
page cache. For example, if you want to benchmark the storage subsystem, you might need to drop the page cache before benchmarking to see true storage performance.

To drop page cache, run the following command:

echo 3 | sudo tee /proc/sys/vm/drop_caches

To compare how much memory area has been released from dropping the page cache, compare the Cached: line in the output of following command before and after invoking the previous command:

cat /proc/meminfo | grep Cached
7. Measuring Workload Performance with Hardware Performance Counters

Many software performance analysis tools rely on event counts from hardware performance monitoring units (PMUs) to characterize workload performance. This chapter provides information about how data from PMUs can be gathered and combined to form metrics for performance optimization. For simplicity, the Linux perf tool is used, but the same metrics can be used in any tool that gathers hardware performance events from PMUs, for example, NVIDIA NSIGHT Systems.

7.1. Introduction to Linux perf

The Linux perf tool is a widely available, open-source tool that is used to collect application-level and system-level performance data. perf can monitor a rich set of software and hardware performance events from different sources and, in many cases, does not require administrator privileges (for example, root).

Installing perf depends on the distribution:

- Ubuntu: `apt install linux-tools-$\{uname -r\}`
- Red Hat: `dnf install perf-$\{uname -r\}`
- SLES: `zypper install perf`

perf gathers data from PMUs by using the `perf_event` system that is provided by the Linux kernel. Data can be gathered for the lifetime of a process or for a specific period. perf supports the following measurement types:

- **Performance summary** (`perf stat`): perf collects the total PMU event counts for a workload and provides a high-level summary of the basic performance characteristics.
  
  This is a good approach for a first-pass performance analysis.

- **Event-based sampling** (`perf record`): perf periodically gathers PMU event counters and relates this data to source code locations.

  The event counts are gathered when a specially configured event counter overflows, which causes an interrupt that contains the instruction pointer addresses and register information. perf uses this information to build stack traces and function-level annotations to characterize the performance of functions of interest on specific call paths. At a high level, this is a good approach for detailed performance characterization after an initial workload characterization. After
gathering data with the `perf record` command, to analyze the data, use the `perf report` or `perf annotate` commands.

7.2. Configuring Perf

By default, unprivileged users can only gather information about context-switched events, which includes most of the predefined CPU core events, such as cycle counts, instruction counts, and some software events. To give unprivileged access to all PMUs and global measurements, the following system settings need to be configured:

![Note: To configure these settings, you must have root access.](image)

- `perf_event_paranoid`: This setting controls privilege checks in the kernel.
  
  Setting this to -1 or 0 allows non-root users to perform per-process and system-wide performance monitoring (refer to Unprivileged users for more information).

- `kptr_restrict`: This setting affects how kernel addresses are exposed.
  
  Setting it to 0 assists in kernel symbol resolution.

For example:

```bash
$ echo -1 | sudo tee /proc/sys/kernel/perf_event_paranoid
$ echo 0 | sudo tee /proc/sys/kernel/kptr_restrict
```

To make these settings reboot persistent, follow your Linux distribution's instructions for configuring system parameters. Typically, you need to edit `/etc/sysctl.conf` or create a file in the `/etc/sysctl.d/` folder that contains the following lines:

```bash
kernel.perf_event_paranoid=-1
kernel.kptr_restrict=0
```

⚠️ **Warning:** There are security implications for configuring these settings as shown in the example above. You **must** read and understand the relevant Linux kernel documentation and consult your system administrator.
7.3. Gathering Hardware Performance Data with Perf

To generate a high-level report of event counts, run the `perf stat` command. For example, to count cache miss events, CPU cycles, and CPU instructions for ten seconds:

```bash
$ perf stat -a -e cache-misses,cycles,instructions sleep 10
```

You can also gather information for a specific process:

```bash
$ perf stat -e cycles,stalled-cycles-backend ./stream.exe
```

This counts total CPU cycles and cycles where the CPU is stalled on the frontend or backend while stream.exe is executing.

The `-e` flag accepts a comma-separated list of performance events, which can be predefined, or raw, events. To see the predefined events, type `perf list`. Raw events are specified as `rxxxx` where `xxxx` is a hexadecimal event number. Refer to the Arm Neoverse V2 Core Technical Reference Manual for more information about event numbers.

For a more detailed analysis, and gather in event-based sampling mode, run the `perf record` command:

```bash
$ perf record -e cycles,instructions,dTLB-loads,dTLB-load-misses ./xhpl.exe
$ perf report
$ perf annotate
```

Additional information about basic perf usage is available in the perf man pages.

7.4. Grace CPU Performance Metrics

This section provides formulas for useful performance metrics, which are functions of a hardware event counts that more fully express the performance characteristics of the system. For example, a simple count of instructions is less meaningful than the ratio of instructions-per-cycle, which characterizes the processor’s usage. These metrics can be used with any tool that gathers hardware performance event data from the Grace PMUs.

The counters are provided by name, instead of event number because most performance analysis tools provide names for common events. If your tool does not have a named counter for one of the following events, use the translation tables in the Arm Neoverse V2 Core Technical Reference Manual to convert the following event names to raw event numbers. For example, FP_SCALE_OPS_SPEC has event number 0x80C0 and FP_FIXED_OPS_SPEC has event number 0x80C1, so data for the FLOPS computational intensity metric can be gathered using perf by measuring raw events 0x80C0 and 0x80C1:

```bash
perf record -e r80c0 -e r80c1 ./a.out
```
7.4.1. Cycle and Instruction Accounting

- **IPC**: Instructions retired per cycle.
  \[ \text{INST\_RETIRED/\text{CPU\_CYCLES}} \]

- **Retiring**: Percentage of total slots that are retired operations and indicates efficient CPU usage.
  \[ 100 \times (\text{OP\_RETIRED/\text{OP\_SPEC}}) \times (1 - (\text{STALL\_ SLOT/}(\text{CPU\_CYCLES} \times \text{8}))) \]

- **Backend Stalls**: Fraction of total cycles that were stalled because of resource constraints in the processor backend.
  \[ \text{STALL\_ BACKEND/\text{CPU\_CYCLES}} \]

- **Frontend Stalls**: Fraction of total cycles that were stalled because of resource constraints in the processor frontend.
  \[ \text{STALL\_ FRONTEND/\text{CPU\_CYCLES}} \]

- **Backend Bound**: Fraction of total slots that were stalled because of resource constraints in the processor backend.
  \[ \text{STALL\_ SLOT\_ BACKEND/}(\text{CPU\_ CYCLES} \times \text{8}) \]

- **Frontend Bound**: Fraction of total slots that were stalled due to resource constraints in the processor frontend.
  \[ ((\text{STALL\_ SLOT\_ FRONTEND}/(\text{CPU\_ CYCLES} \times \text{8}))) - ((\text{BR\_MIS\_PRED} \times \text{4})/\text{CPU\_ CYCLES})) \]

7.4.2. Computational Intensity

- **SVE FLOPS**: Floating point operations per second in any precision performed by the SVE instructions.
  
  Fused instructions count as two operations, for example, a fused multiply-add instruction increases the count by twice the number of active SVE vector lanes. These operations do not count as floating point operations that are performed by scalar or NEON instructions.
  \[ \text{FP\_SCALE\_OPS\_SPEC/TIME} \]

- **Non-SVE FLOPS**: Floating point operations per second in any precision performed by an instruction that is not an SVE instruction.
  
  Fused instructions count as two operations, for example, a scalar fused multiply-add instruction increases the count by two, and a fused multiply-add NEON instruction increases the count by twice the number of vector lanes. These operations do not count as floating point operations performed by SVE instructions.
  \[ \text{FP\_FIXED\_OPS\_SPEC/TIME} \]
- **FLOPS**: Floating point operations per second in any precision performed by any instruction.

  Fused instructions count as two operations.

  \[(\text{FP\_SCALE\_OPS\_SPEC} + \text{FP\_FIXED\_OPS\_SPEC}) / \text{TIME}\]

### 7.4.3. Operation Mix

- **Load Percentage**: Fraction of total instructions that were speculatively executed because of the load instructions.

  \[
  \frac{\text{LD\_SPEC}}{\text{INST\_SPEC}}
  \]

- **Store Percentage**: Fraction of total instructions speculatively executed because of the store instructions.

  \[
  \frac{\text{ST\_SPEC}}{\text{INST\_SPEC}}
  \]

- **Branch Percentage**: Fraction of total instructions that were speculatively executed because of the branch instructions.

  \[
  \frac{\text{(BR\_IMMED\_SPEC} + \text{BR\_INDIRECT\_SPEC})}{\text{INST\_SPEC}}
  \]

- **Scalar Integer Percentage**: Fraction of total instructions that were speculatively executed because of the scalar integer instructions.

  \[
  \frac{\text{DP\_SPEC}}{\text{INST\_SPEC}}
  \]

  **Note:** The DP in DP\_SPEC stands for Data Processing.

- **Scalar Floating Point Percentage**: Fraction of total instructions speculatively executed because of the scalar floating point instructions.

  \[
  \frac{\text{VFP\_SPEC}}{\text{INST\_SPEC}}
  \]

- **Synchronization Percentage**: Fraction of total instructions that were speculatively executed because of the synchronization instructions.

  \[
  \frac{\text{(ISB\_SPEC} + \text{DSB\_SPEC} + \text{DMB\_SPEC})}{\text{INST\_SPEC}}
  \]

- **Crypto Percentage**: Fraction of total instructions that were speculatively executed because of the crypto instructions.

  \[
  \frac{\text{CRYPTO\_SPEC}}{\text{INST\_SPEC}}
  \]

- **SVE SIMD Percentage**: Fraction of total instructions that were speculatively executed because of the integer or floating point SVE SIMD instructions.

  \[
  \frac{\text{SVE\_INST\_SPEC}}{\text{INST\_SPEC}}
  \]
- **NEON SIMD Percentage**: Fraction of total instructions that were speculatively executed because of the integer or floating point NEON SIMD instructions.
  
  \[\frac{ASE\_INST\_SPEC}{INST\_SPEC}\]

- **SIMD Percentage**: Fraction of total instructions that were speculatively executed because of the integer or floating point vector/SIMD instructions.
  
  \[\frac{(SVE\_INST\_SPEC + ASE\_INST\_SPEC)}{INST\_SPEC}\]

- **FP16 Percentage**: Fraction of total instructions that were speculatively executed because of the half-precision floating point instructions.
  
  Includes scalar, fused, and SIMD instructions and cannot be used to measure computational intensity.
  
  \[\frac{FP\_HP\_SPEC}{INST\_SPEC}\]

- **FP32 Percentage**: Fraction of total instructions that were speculatively executed because of the single-precision floating point instructions.
  
  Includes scalar, fused, and SIMD instructions and cannot be used to measure computational intensity.
  
  \[\frac{FP\_SP\_SPEC}{INST\_SPEC}\]

- **FP64 Percentage**: Fraction of total instructions that were speculatively executed because of the double-precision floating point instructions.
  
  Includes scalar, fused, and SIMD instructions and cannot be used to measure computational intensity.
  
  \[\frac{FP\_DP\_SPEC}{INST\_SPEC}\]

7.4.4. **SVE Predication**

- **Full SVE Instructions**: Fraction of total instructions that were speculatively executed because of the SVE SIMD instructions with all active predicates.
  
  \[\frac{SVE\_PRED\_FULL\_SPEC}{INST\_SPEC}\]

- **Partial SVE Instructions**: Fraction of total instructions that were speculatively executed because of the SVE SIMD instructions in which at least one element is FALSE.
  
  \[\frac{SVE\_PRED\_PARTIAL\_SPEC}{INST\_SPEC}\]

- **Empty SVE Instructions**: Fraction of total instructions that were speculatively executed because of the SVE SIMD instructions with no active predicate.
  
  \[\frac{SVE\_PRED\_EMPTY\_SPEC}{INST\_SPEC}\]
7.4.5. Cache Effectiveness

- **L1 Data Cache Misses**: Fraction of total level 1 data cache read or write accesses that miss.
  
  \[ \text{L1D\_CACHE includes reads and writes and is the sum of L1D\_CACHE\_RD and L1D\_CACHE\_WR.} \]

- **L1D_CACHE_REFILL/L1D_CACHE**

- **L1 Data Cache Miss Rate**: Count of level 1 data cache read or write accesses that miss per kilo-instructions executed.
  
  \[ \text{L1D\_CACHE\_REFILL/(INST\_RETIRED \times 1000)} \]

- **L1 Instruction Cache Misses**: Fraction of total level 1 instruction cache accesses that miss.
  
  \[ \text{L1I\_CACHE does not measure cache maintenance instructions or non-cacheable accesses.} \]

- **L1I_CACHE\_REFILL/L1I_CACHE**

- **L1 Instruction Cache Miss Rate**: Count of level 1 instruction cache accesses missed per kilo-instructions executed.
  
  \[ \text{L1I\_CACHE\_REFILL/(INST\_RETIRED \times 1000)} \]

- **L2 Cache Misses**: Fraction of total level 2 cache read or write accesses that miss.
  
  \[ \text{L2D\_CACHE does not count cache maintenance operations or snoops from outside the core.} \]

- **L2D_CACHE\_REFILL/L2D_CACHE**

- **L2 Cache Miss Rate**: Count of level 2 cache read or write accesses that miss per kilo-instructions executed.
  
  \[ \text{L2D\_CACHE\_REFILL/(INST\_RETIRED \times 1000)} \]

- **L3 Cache Read Hits**: Fraction of L3 cache read accesses that hit.
  
  \[ \text{(LL\_CACHE\_RD \minus LL\_CACHE\_MISS\_RD)/LL\_CACHE\_RD} \]

- **L3 Cache Read Misses**: Fraction of L3 cache read accesses that miss.
  
  \[ \text{LL\_CACHE\_MISS\_RD/LL\_CACHE\_RD} \]

- **L3 Cache Read Miss Rate**: Count of L3 cache read accesses missed per kilo-instructions executed.
  
  \[ \text{LL\_CACHE\_MISS\_RD/(INST\_RETIRED \times 1000)} \]

7.4.6. TLB Effectiveness

- **L1 Data TLB Misses**: Fraction of total level 1 data TLB accesses that miss.
  
  \[ \text{TLB maintenance instructions are not counted.} \]

- **L1D_TLB\_REFILL/L1D_TLB**
• **L1 Data TLB Miss Rate**: Count of level 1 data TLB accesses that miss per kilo-instructions executed.

\[
\text{L1D_TLB_REFILL}/(\text{INST_RETIRED} \times 1000)
\]

• **L1 Instruction TLB Misses**: Fraction of total level 1 instruction TLB accesses that miss.

TLB maintenance instructions are not counted.

\[
\text{L1I_TLB_REFILL}/\text{L1I_TLB}
\]

• **L1 Instruction TLB Miss Rate**: Count of level 1 instruction TLB accesses that miss per kilo-instructions executed.

\[
\text{L1I_TLB_REFILL}/(\text{INST_RETIRED} \times 1000)
\]

7.4.7. **Branching**

• **Branch Mispredictions**: Fraction of architecturally executed branches that were mispredicted.

\[
\text{BR_MIS_PRED_RETIRED}/\text{BR_RETIRED}
\]

• **Branch Misprediction Rate**: Count of branches that were mispredicted for each kilo-instructions that were executed.

\[
\text{BR_MIS_PRED_RETIRED}/(\text{INST_RETIRED} \times 1000)
\]

7.4.8. **Grace Uncore PMU Units**

Grace includes the following uncore PMUs that are registered by the PMU driver with the following naming conventions:

<table>
<thead>
<tr>
<th>System</th>
<th>Uncore PMU name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalable Coherency Fabric</td>
<td>nvidia_scf_pmu_&lt;socket-id&gt;</td>
</tr>
<tr>
<td>NVLINK-C2C0</td>
<td>nvidia_nvlink_c2c0_pmu_&lt;socket-id&gt;</td>
</tr>
<tr>
<td>NVLINK-C2C1 (Grace-Hopper Only)</td>
<td>nvidia_nvlink_c2c1_pmu_&lt;socket-id&gt;</td>
</tr>
<tr>
<td>PCIe</td>
<td>nvidia_pcie_pmu_&lt;socket-id&gt;</td>
</tr>
<tr>
<td>CNVLINK (Grace-Hopper Only)</td>
<td>nvidia_cnvlink_pmu_&lt;socket-id&gt;</td>
</tr>
</tbody>
</table>

The traffic pattern determines which PMU is used for measuring the different access types.

**Table 7-2** provides information about PMU accounting for access patterns on the NVIDIA Grace CPU Superchip.
Table 7-2. PMU Accounting for Access Patterns on the Grace Superchip

<table>
<thead>
<tr>
<th>Destination</th>
<th>Source - Socket A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Socket A PCI R/W</td>
</tr>
<tr>
<td></td>
<td>CPU</td>
</tr>
<tr>
<td></td>
<td>Socket B PCIe strongly ordered (SO) writes</td>
</tr>
<tr>
<td></td>
<td>Socket B PCIe reads or relaxed order (RO) writes</td>
</tr>
<tr>
<td>Local memory</td>
<td>PCIe PMU</td>
</tr>
<tr>
<td>Remote memory over NVLink-C2C</td>
<td>PCIe PMU</td>
</tr>
</tbody>
</table>

Table 7-3 provides information about PMU accounting for access patterns on the NVIDIA Grace Hopper Superchip.

Table 7-3. PMU Accounting for Access Patterns on the Grace Hopper Superchip

<table>
<thead>
<tr>
<th>Destination</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Socket A PCI R/W</td>
</tr>
<tr>
<td></td>
<td>GPU ATS Translated Accesses</td>
</tr>
<tr>
<td></td>
<td>GPU Accesses not translated by ATS</td>
</tr>
<tr>
<td></td>
<td>Local CPU</td>
</tr>
<tr>
<td>Local CPU memory</td>
<td>PCIe PMU</td>
</tr>
<tr>
<td>Local GPU memory</td>
<td>PCIe PMU</td>
</tr>
</tbody>
</table>

Uncore PMU events are not attributable to a core, and perf must be run in system-wide mode, as opposed to per-thread mode. If the measurement requires multiple events to be measured, perf tools support event grouping from the same PMU.

For example, to monitor SCF CYCLES, CMEM WB ACCESS and CMEM WR ACCESS events from the SCF PMU for socket 0:

```bash
$ perf stat -a -e
duration_time,'{nvidia_scf_pmu_0/cycles/,nvidia_scf_pmu_0/cmem_wb_access/,nvidia_scf_pmu_0/cmem_wr_access/}' cmem_write_test
Performance counter stats for 'system wide':

  168225760 ns  duration_time
  10515321      nvidia_scf_pmu_0/cycles/
  191567        nvidia_scf_pmu_0/cmem_wb_access/
```
7.4.9. Scalable Coherency Fabric PMU Accounting

**Note:** The bandwidth metrics in this section are in MBs per second.

This section provides additional formulas for useful performance metrics based on events provided by the Scalable Coherency Fabric (SCF) PMU.

- **Duration**: Duration in nanoseconds
  
  \[DURATION\_TIME\]

- **Cycles**: SCF cycle count
  
  \[CYCLES\]

- **SCF local CPU memory write bandwidth**: Write bandwidth from SCF to local CPU memory
  
  \[CMEM\_WR\_TOTAL\_BYTES*1000/DURATION\_TIME\]

- **SCF local CPU memory read bandwidth**: Read bandwidth from SCF to local CPU memory.
  
  Total "beats" are measured with each beat reading 32 bytes.
  
  \[CMEM\_RD\_DATA*32*1000/DURATION\_TIME\]

- **SCF local GPU memory write bandwidth**: Write bandwidth from SCF to local GPU memory
  
  \[GMEM\_WR\_TOTAL\_BYTES*1000/DURATION\_TIME\]

- **SCF local GPU memory read bandwidth**: Read bandwidth from SCF to local GPU memory.
  
  Total "beats" are measured with each beat reading 32 bytes.
  
  \[GMEM\_RD\_DATA*32*1000/DURATION\_TIME\]

- **SCF remote memory write bandwidth**: Write bandwidth from SCF to remote socket memory
  
  \[REMOTE\_OCKET\_WR\_TOTAL\_BYTES*1000/DURATION\_TIME\]

- **SCF remote memory read bandwidth**: Read bandwidth from SCF to remote socket memory.
  
  Total “beats” are measured with each beat reading 32 bytes.
  
  \[REMOTE\_OCKET\_RD\_DATA*32*1000/DURATION\_TIME\]
- **SCF local CPU memory write utilization percentage**: Percent utilization from SCF to local CPU memory for writes.

A total of CPU memory writeback (CMEM_WB_ACCESS) plus write-unique and non-coherent write requests (CMEM_WR_ACCESS) divided by the maximum writes (16*CYCLES).

\[
\left( \frac{\text{CMEM}_\text{WB}\_\text{ACCESS} + \text{CMEM}_\text{WR}\_\text{ACCESS}}{16\times\text{CYCLES}} \right) \times 100.0
\]

- **SCF local GPU memory and remote memory write utilization percentage**: Percent utilization from SCF to local GPU memory and remote socket memory for writes.

A total of local GPU memory and remote socket memory writeback (GMEM_WB_ACCESS + SOCKET_(0,1)_WB_ACCESS) plus write-unique and non-coherent write requests (GMEM_WR_ACCESS + SOCKET_(0,1)_WR_ACCESS) divided by the maximum writes (16*CYCLES).

Socket 0 access to local GPU memory and socket 1 memory:

\[
\left( \frac{\text{GMEM}_\text{WB}\_\text{ACCESS} + \text{GMEM}_\text{WR}\_\text{ACCESS} + \text{SOCKET}_1\_\text{WR}\_\text{ACCESS}}{16\times\text{CYCLES}} \right) \times 100.0
\]

Socket 1 access to the local GPU and socket 0 memory:

\[
\left( \frac{\text{GMEM}_\text{WB}\_\text{ACCESS} + \text{GMEM}_\text{WR}\_\text{ACCESS} + \text{SOCKET}_0\_\text{WR}\_\text{ACCESS}}{16\times\text{CYCLES}} \right) \times 100.0
\]

- **SCF local CPU memory read utilization percentage**: Percent utilization from SCF to local CPU memory for reads.

Total local CPU memory reads (CMEM_RD_ACCESS) divided by the maximum reads (16 * CYCLES).

\[
\left( \frac{\text{CMEM}_\text{RD}\_\text{ACCESS}}{16\times\text{CYCLES}} \right) \times 100.0
\]

- **SCF local GPU memory and remote memory read utilization percentage**: Percent utilization from SCF to local GPU memory and remote socket memory for reads.

A total of local GPU memory and remote socket memory reads (GMEM_RD_ACCESS + SOCKET_(0,1)_RD_ACCESS) divided by the maximum reads (16 * CYCLES).

Socket 0 access to local GPU memory and socket 1 memory:

\[
\left( \frac{\text{GMEM}_\text{RD}\_\text{ACCESS} + \text{SOCKET}_1\_\text{RD}\_\text{ACCESS}}{16\times\text{CYCLES}} \right) \times 100.0
\]

Socket 1 access to the local GPU memory and socket 0 memory:

\[
\left( \frac{\text{GMEM}_\text{RD}\_\text{ACCESS} + \text{SOCKET}_0\_\text{RD}\_\text{ACCESS}}{16\times\text{CYCLES}} \right) \times 100.0
\]

- **SCF Frequency**: Frequency of SCF cycles in GHz

\[
\frac{\text{CYCLES}}{\text{DURATION}}
\]

- **SCF local CPU memory read latency**: Latency of SCF reads to local CPU memory, in nanoseconds.

\[
\frac{\text{CMEM}_\text{RD}\_\text{OUTSTANDING}\_\text{ACCESS}}{\text{CYCLES}\_\text{DURATION}}
\]

Average cycles per SCF local read request: (CMEM_RD_OUTSTANDING / CMEM_READ_ACCESS) divided by SCF frequency (CYCLES/DURATION) to determine average nanoseconds per local read.
- **SCF local GPU memory read latency:** Latency of SCF reads to local GPU memory, in nanoseconds.
  \[
  \frac{(\text{GMEM_RD_OUTSTANDING} / \text{GMEM_RD_ACCESS})}{(\text{CYCLES/DURATION})}
  \]
  Average cycles per SCF local read request: \((\text{GMEM_RD_OUTSTANDING} / \text{GMEM_RD_ACCESS})\) divided by SCF frequency \((\text{CYCLES/DURATION})\) to determine average nanoseconds per local read.

- **SCF Remote memory read latency:** Latency of SCF reads to remote memory in nanoseconds.
  \[
  \frac{(\text{SOCKET}_{0,1}\text{RD_OUTSTANDING} / \text{SOCKET}_{0,1}\text{RD_ACCESS})}{(\text{SCF}\text{FREQUENCY})}
  \]
  Average cycles per SCF remote read request: \((\text{SOCKET}_{0,1}\text{RD_OUTSTANDING} / \text{SOCKET}_{0,1}\text{RD_ACCESS})\) divided by SCF frequency \((\text{CYCLES/DURATION})\) to determine average nanoseconds per remote socket read.

  Socket 0 access to socket 1 memory:
  \[
  \frac{(\text{SOCKET}_1\text{RD_OUTSTANDING} / \text{SOCKET}_1\text{RD_ACCESS})}{(\text{CYCLES/DURATION})}
  \]

  Socket 1 access to socket 0 memory:
  \[
  \frac{(\text{SOCKET}_0\text{RD_OUTSTANDING} / \text{SOCKET}_0\text{RD_ACCESS})}{(\text{CYCLES/DURATION})}
  \]

7.4.10. PCIe PMU Accounting

**Note:** The bandwidth metrics in this section are in GBps per second.

The PCIe PMU requires an event filter to be specified when counting events, and the Grace CPU has 10 PCIe root ports per socket.

The `root_port` bitmap parameter can be passed to select the port(s) to monitor. For example, the `root_port=0xF` parameter corresponds to root ports 0 through 3.

**Example:** To count the `rd_bytes_loc` event from PCIe root port 0 and 1 of socket 0:

```bash
$ perf stat -a -e nvidia_pcie_pmu_0/rd_bytes_loc,root_port=0x3/
```

**Example:** To count the `rd_bytes_loc` event from PCIe root port 0 and 1 in socket 1 and measure the test duration (in nanoseconds):

```bash
$ perf stat -a -e duration_time,"\{nvidia_pcie_pmu_1/rd_bytes_loc,root_port=0x3/\}"
```

The rest of this section provides additional formulas for useful performance metrics based on events provided by the PCIe PMU.

- **PCIe read bandwidth:** PCI Express read bandwidth, in GBps
  Number of bytes read from local memory and remote memory \((\text{RDBYTES_LOC, RDBYTES_REM})\) divided by the duration in nanoseconds.
  \[
  \frac{(\text{RDBYTES_LOC} + \text{RDBYTES_REM})}{(\text{DURATION_TIME})}
  \]
- **PCIe write bandwidth**: PCI Express write bandwidth, in GBps
  Number of bytes written to local or remote memory (WR_BYTES_LOC, WR_BYTES_REM), divided by the duration in nanoseconds.
  \[
  \frac{(WR_{BYTES_{LOC}} + WR_{BYTES_{REM}})}{DURATION_{TIME}}
  \]

- **PCIe bidirectional bandwidth**: PCI Express read and write bandwidth, in GBps
  Number of bytes read or written from local or remote memories, divided by the duration in nanoseconds.
  \[
  \frac{(RD_{BYTES_{LOC}} + RD_{BYTES_{REM}} + WR_{BYTES_{LOC}} + WR_{BYTES_{REM}})}{DURATION_{TIME}}
  \]

- **PCIe read utilization percentage**: Percent utilization of PCI Express for reads
  Number of read requests to local and remote memories (RD_REQ_LOC, RD_REQ_REM) divided by the maximum number of read requests per cycle (1 per port, total 10) times the number of cycles.
  \[
  \left(\frac{(RD_{REQ_{LOC}} + RD_{REQ_{REM}})}{(10 \times CYCLES)}\right) \times 100.0
  \]

- **PCIe write utilization percentage**: Percent utilization of PCI Express for writes
  Number of write requests to local and remote memories (WR_REQ_LOC, WR_REQ_REM) divided by the maximum number of write requests per cycle (1 per port, total 10) times the number of cycles.
  \[
  \left(\frac{(WR_{REQ_{LOC}} + WR_{REQ_{REM}})}{(10 \times CYCLES)}\right) \times 100.0
  \]

- **PCIe Frequency**: Frequency of PCIe cycles in GHz
  \[
  \frac{CYCLES}{DURATION}
  \]

- **PCIE local memory read latency**: Latency of PCIe reads to local memory, in nanoseconds.
  \[
  \frac{RD_{CUM\_OUTS_{LOC}}}{RD_{REQ_{LOC}}} \div (CYCLES/DURATION)
  \]
  Average cycles per PCIe local read request: \(\frac{RD_{CUM\_OUTS_{LOC}}}{RD_{REQ_{LOC}}}\)
  divided by PCIe frequency \((CYCLES/DURATION)\) to determine average nanoseconds per local read.

- **PCIE remote memory read latency**: Latency of PCIe reads to remote CPU memory, in nanoseconds.
  \[
  \frac{RD_{CUM\_OUTS_{REM}}}{RD_{REQ_{REM}}} \div (CYCLES/DURATION)
  \]
  Average cycles per PCIe remote read request: \(\frac{RD_{CUM\_OUTS_{REM}}}{RD_{REQ_{REM}}}\)
  divided by PCIe frequency \((CYCLES/DURATION)\) to determine average nanoseconds per remote read.
7.4.11. NVLink C2C Accounting

**Note:** The bandwidth metrics in this section are in GBps per second.

The NVLink C2C PMU provides performance metrics for the CPU or GPU memory accesses via the NVLink Chip-2-Chip (C2C) interconnect.

Two C2C PMUs, NVLINK-C2C0, and NVLINK-C2C1 are available and cover different types of traffic. Refer to Table 7-2 and Table 7-3 for more information about the traffic patterns covered by each PMU. The NVLINK-C2C1 PMU is unused on the Grace Superchip.

This section provides additional formulas for useful performance metrics based on events provided by the NVLINK C2C PMU.

- **NVLink C2C read bandwidth:** NVLink C2C read bandwidth, in GBps
  Number of bytes read from local memory or remote memory (RD_BYTES_LOC, RD_BYTES_REM) divided by the duration in nanoseconds.
  \[
  \frac{(RD_{BYTES\_LOC} + RD_{BYTES\_REM})}{DURATION\_TIME}
  \]

- **NVLink C2C write bandwidth:** NVLink C2C write bandwidth, in GBps
  Number of bytes written to local or remote memory (WR_BYTES_LOC, WR_BYTES_REM), divided by the duration in nanoseconds.
  \[
  \frac{(WR_{BYTES\_LOC} + WR_{BYTES\_REM})}{DURATION\_TIME}
  \]

- **NVLink C2C bidirectional bandwidth:** NVLink C2C read and write bandwidth, in GBps
  Number of bytes read or written from local or remote memories, divided by the duration in nanoseconds.
  \[
  \frac{(RD_{BYTES\_LOC} + RD_{BYTES\_REM} + WR_{BYTES\_LOC} + WR_{BYTES\_REM})}{DURATION\_TIME}
  \]

- **NVLink C2C read utilization percentage:** NVLink C2C utilization for reads
  Number of read requests to local and remote memories (RD_REQ_LOC, RD_REQ_REM) divided by the maximum number of read requests per cycle (10) times the number of cycles.
  \[
  \frac{(RD_{REQ\_LOC} + RD_{REQ\_REM})}{(10 \times CYCLES)} \times 100.0
  \]

- **NVLink C2C write utilization percentage:** NVLink C2C utilization for writes
  Number of write requests to local and remote memories (WR_REQ_LOC, WR_REQ_REM) divided by the maximum number of write requests per cycle (10) times the number of cycles.
  \[
  \frac{(WR_{REQ\_LOC} + WR_{REQ\_REM})}{(10 \times CYCLES)} \times 100.0
  \]

- **NVLink C2C Frequency:** Frequency of NVLink C2C cycles in GHz
  \[
  \frac{CYCLES}{DURATION}
  \]
• **NVLink C2C local memory read latency**: Latency of NVLink C2C reads to local memory, in nanoseconds.

\[
\text{(RD\_CUM\_OUTS\_LOC / RD\_REQ\_LOC) / (CYCLES/DURATION)}
\]

Average cycles per C2C local read request (RD\_CUM\_OUTS\_LOC / RD\_REQ\_LOC) divided by C2C frequency (CYCLES/DURATION) to determine average nanoseconds per local read.

• **NVLink C2C remote memory read latency**: Latency of NVLink C2C reads to remote CPU memory, in nanoseconds.

\[
\text{(RD\_CUM\_OUTS\_REM / RD\_REQ\_REM) / (CYCLES/DURATION)}
\]

Average cycles per C2C remote read request (RD\_CUM\_OUTS\_REM / RD\_REQ\_REM) divided by PCIe frequency (CYCLES/DURATION) to determine average nanoseconds per remote read.

### 7.4.12. Profiling CPU Behavior with Nsight Systems

The Nsight Systems tool (also referred to as nsys) profiles the systems compute units including the CPUs and GPUs (refer to [Nsight Systems | NVIDIA Developer](https://developer.nvidia.com/nsight) for more information). The tool can trace more than 25 APIs including CUDA APIs, sample CPU instruction pointers/backtraces, sample both CPU and SoC event counts, and sample GPU hardware events to provide a system-wide view of a workload’s behavior.

nsys can sample the CPU and SoC events described above and graph their rate on the nsys UI timeline. If CPU IP/backtrace data is gathered concurrently, users can determine when CPU and SoC events are extremely active (or inactive) and correlate that information with the IP/backtrace data to determine which workload aspect was actively running at that time.

**Figure 7-1** shows a sample example nsys timeline. In this case, 6 CPU events were sampled including the CPU Cycles, FP\_DP\_SPEC, FP\_FIXED\_OPS\_SPEC, FP\_SCALE\_OPS\_SPEC, FP\_SP\_SPEC, and OP\_RETIRED events. CPU IP/backtrace samples were also collected. The orange-yellow tick marks under a thread’s timeline represent individual IP/backtrace samples. The following timeline shows the following tooltips:

- For a IP/backtrace sample (red circle) on thread 194865 that is running on CPU 72.
- For the rate of the FP\_FIXED\_OPS\_SPEC event that is on CPU 72 at the same time.

Only one tooltip can be shown at a time in the tool (refer to **Table 7-1** for more information).

The IP/backtrace tooltip shows that the clover\_leaf process is currently executing the _nv\_advec\_cell\_kernel\_module\_advec\_cell\_kernel\_F1L81\_1_ function and the path taken to that function. The FP\_FIXED\_OPS\_SPEC tooltip shows that the rate of the FP\_FIXED\_OPS\_SPEC event was 1310.26 floating point operations/us at that time.
Use the \texttt{--cpu-core-events} and \texttt{--sample} \texttt{nsys} CLI switches to collect the above data. Also, see the \texttt{nsys} CLI switches \texttt{--cpu-socket-events} and \texttt{--cpuctxsw} for profiling CPU and/or SoC performance issues. For more information, run \texttt{nsys profile --help}. 
8. Compilers

Many commercial and open-source compilers fully support NVIDIA Grace. This section provides information about the available compilers, the recommended versions, and the recommended command-line options.

8.1. NVIDIA HPC Compilers

The NVIDIA HPC SDK includes proven compilers, libraries, and software tools. The HPC SDK compilers (NVHPC) enable cross-platform C, C++, and Fortran programming for NVIDIA GPUs and multicore Arm, OpenPOWER, or x86-64 CPUs. The compilers are ideal for HPC modeling and simulation applications that are written in C, C++, or Fortran with OpenMP, OpenACC, and NVIDIA CUDA®.

When building natively on Grace, NVHPC version 23.3 or later automatically optimizes for Grace without additional command-line options. To verify, pass the --version command-line option and look for -tp neoverse-v2 in the output:

```
nvidia@localhost:~$ nvc --version
nvc 23.3-0 linuxarm64 target on aarch64 Linux -tp neoverse-v2
```

The optimization and floating-point control flags for NVHPC are the same on NVIDIA Grace as on other CPUs. Refer to the NVIDIA HPC Compilers User’s Guide for more information.

**Note:** NVIDIA provides the BLAS, LAPACK, and FFT math libraries that are optimized for Grace, and we strongly recommend that you use them.
8.2. GNU Toolchain

When using the GNU toolchain, we recommend GCC version 12.3 or later. When possible, always use the latest version of GCC. GCC version 7 can be used on NVIDIA Grace, but because these older compilers target earlier Armv8-A architecture variants, the performance will be suboptimal. The latest and greatest binary versions of GNU toolchain can be found from your GNU/Linux distribution or downloaded using Spack.

Even with the latest version of GCC, unless your toolchain has been configured and built for Grace, additional command-line options are necessary to generate optimal code for NVIDIA Grace. If no additional flags are provided, GCC will generate code targeting a generic Armv8-A CPU. The recommended flags are provided in Table 8-1.

![Note: More aggressive optimizations will trade floating point accuracy for performance.]

Table 8-1. Organization Levels and Flags

<table>
<thead>
<tr>
<th>Optimization Level</th>
<th>Flags</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggressive</td>
<td>-Ofast, -mcpu=neoverse-v2</td>
<td>Enable fast math optimizations</td>
</tr>
<tr>
<td>Moderate</td>
<td>-O3, -mcpu=neoverse-v2</td>
<td>Recommended in most cases</td>
</tr>
</tbody>
</table>

The -mcpu=neoverse-v2 flag is used in all cases. We recommend that you use the -mcpu flag instead of the -march and -mtune flags because this flag will select the CPU that you were targeting for convenience instead of specifying the architecture with the required extensions using the -march option and then specifying the -mtune option. Refer to https://gcc.gnu.org/onlinedocs/gcc/AArch64-Options.html#aarch64-feature-modifiers for more information about the instruction set features that can be turned on and off on a per-feature basis.

The __sync built-ins in GNU C or GNU C++ are precursors to modern atomic extensions that are used in the C11 / C++11 standards. These built-ins are now considered legacy, and users should port the atomic extensions in C11 / C++11. This is good advice for any platform, but it is particularly relevant for CPUs implementing the AArch64 architecture because the legacy __sync built-ins tend to enforce more strict orderings than are necessary. Refer to https://gcc.gnu.org/onlinedocs/gcc/_005f_005fatomic-Builtins.html for more information.

The C standard does not specify the signedness of the char type. On x86, a char is assumed to be signed by default, and on Arm, char is assumed to be unsigned. This
difference can be addressed by using the standard int types that specify signedness when the sign of a number is important (for example, uint8_t and int8_t) or by compiling with the -fsigned-char flag to set the signedness of char at compile time.

Refer to https://gcc.gnu.org/onlinedocs/gcc-13.1.0/gcc/AArch64-Options.html for more information about the command-line options that are required for the AArch64 target in GCC.

8.3. LLVM Clang and Flang Compilers

When you use LLVM, we recommend LLVM version 16 or later. LLVM compilers support Arm64 CPUs but mainly for C and C++ (the clang and clang++ commands). LLVM’s Fortran compiler (flang) is not yet widely used and is still maturing. Like the GNU compilers, Clang prioritizes portability over performance and additional flags must be added to enable optimization.

NVIDIA provides builds of LLVM Clang at developer.nvidia.com/grace/clang that are specially packaged for the Grace CPU. These builds are mainline Clang that are configured to support Grace, so the builds can be used as a drop-in replacement for Clang in your current workflows.

Table 8-2. Optimization Levels and Flags

<table>
<thead>
<tr>
<th>Optimization Level</th>
<th>Flags</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggressive</td>
<td>-Ofast</td>
<td>Enable fast math optimizations</td>
</tr>
<tr>
<td></td>
<td>-mcpu=neoverse-v2</td>
<td></td>
</tr>
<tr>
<td>Moderate</td>
<td>-O3</td>
<td>Recommended in most cases</td>
</tr>
<tr>
<td></td>
<td>-mcpu=neoverse-v2</td>
<td></td>
</tr>
<tr>
<td>Conservative</td>
<td>-O3 -ffp-contract=off</td>
<td>Disable fused math operations</td>
</tr>
<tr>
<td></td>
<td>-mcpu=neoverse-v2</td>
<td></td>
</tr>
</tbody>
</table>

The -mcpu=neoverse-v2 flag is used in all cases, and we recommend using the -mcpu flag instead of the -march and -mtune flags.

8.4. Arm Compiler for Linux and Other Commercial Compilers

The Arm Compiler for Linux (ACfL) is a commercially supported, closed-source compiler provided by Arm. It is free and is bundled with the optimized BLAS, LAPACK, and FFT libraries. Arm supports NVIDIA Grace in ACfL through support for the Neoverse-V2 CPU
microarchitecture. Refer to Arm Compiler for Linux for more information about compiler options, flags, and support.

System vendors, such as HPE/Cray and Fujitsu, also provide compilers that target their own Arm-based products. The code generated by these vendor compilers tends to be highly tuned for the target platform, which makes them a good choice in performance-critical situations. Contact the system vendor for information and support.

8.5. Arm Architecture Feature Support

Like other major CPU architectures, there are instructions that cannot be reached directly by translating the C / C++ language. These instructions are usually supported by the weight of compiler intrinsics and a set of feature macros that can be used in applications to test for the specific architecture extension support at compile time. The common set of intrinsics, additional data types, architectural feature macros among others for the Arm architecture for compilers are defined by the Arm C / C++ Language Extensions (ACLE). Refer to https://github.com/ARM-software/acle for more information.

A discussion on ACLE and intrinsics programming is out of scope for this document and the users are advised to refer to it and their compiler documentation to check for the level of compliance with the same.

NVIDIA Grace implements the Armv9-A architecture and several of the Armv9-A architectural extensions. To see which Arm architectural features are enabled at compile time, run the following command:

```
gcc -dM -E -mcpu=neoverse-v2 - < /dev/null | grep ARM_FEATURE
```

Here is an example of the output with GCC 12 on NVIDIA Grace:

```
#define __ARM_FEATURE_AES 1
#define __ARM_FEATURE_ATOMIC 1
#define __ARM_FEATURE_BF16_SCALAR_ARITHMETIC 1
#define __ARM_FEATURE_BF16_VECTOR_ARITHMETIC 1
#define __ARM_FEATURE_CLZ 1
#define __ARM_FEATURE_COMPLEX 1
#define __ARM_FEATURE_CRC32 1
#define __ARM_FEATURE_CRYPTO 1
#define __ARM_FEATURE_FMA 1
#define __ARM_FEATURE_FP16 FML 1
#define __ARM_FEATURE_FP16_SCALAR_ARITHMETIC 1
#define __ARM_FEATURE_FP16_VECTOR_ARITHMETIC 1
#define __ARM_FEATURE_FRINT 1
#define __ARM_FEATURE_IDIV 1
#define __ARM_FEATURE_IDIV 1
#define __ARM_FEATURE_JCVT 1
#define __ARM_FEATURE_MATMUL_INT8 1
#define __ARM_FEATURE_NUMERIC_MAXMIN 1
#define __ARM_FEATURE_QRDMX 1
```

NVIDIA Grace Performance Tuning Guide
Refer to the output of the following command for more information about target specific flags on arm64:

gcc -Q --help=target

Here is the sample output:

```bash
nvidia@localhost:~$ gcc -Q --help=target
The following options are target specific:
  -mabi= lp64
  -march= armv8-a
  -mbig-endian [disabled]
  -mbionic [disabled]
  -mbranch-protection=
  -mcmodel= small
  -mcpu= generic
  -mfixed-cortex-a53-835769 [enabled]
  -mfixed-cortex-a53-843419 [enabled]
  -mgeneral-regs-only [disabled]
  -mglIBC [enabled]
  -mharden-sls=
  -mlittle-endian [enabled]
  -mlow-precision-div [disabled]
  -mlow-precision-recip-sqrt [disabled]
  -mlow-precision-sqrt [disabled]
  -mmusl [disabled]
  -momit-leaf-frame-pointer [enabled]
  -moutline-atomics [enabled]
  -mpc-relative-literal-loads [enabled]
  -msign-return-address= none
  -mstack-protector-guard-offset=
  -mstack-protector-guard-reg=
  -mstack-protector-guard= global
  -mstrict-align [disabled]
```
-msve-vector-bits=<number>  scalable
-mtls-dialect=  desc
-mtls-size=  24
-mtrack-speculation  [disabled]
-mtune=  generic
-muclibc  [disabled]
-mverbose-cost-dump  [disabled]

Known AArch64 ABIs (for use with the -mabi= option):
  ilp32  lp64

Supported AArch64 return address signing scope (for use with
-msign-return-address= option):
  all  non-leaf  none

The code model option names for -mcmodel:
  large  small  tiny

Valid arguments to -mstack-protector-guard=:
  global  sysreg

The possible SVE vector lengths:
  1024  128  2048  256  512  scalable

The possible TLS dialects:
  desc  trad

## 8.6. Using Code Locality to Improve Performance

Improving executable code locality can increase efficiency on Grace, which benefits
the instruction cache hit rate, the iTLB hit rate, and branch prediction. Executables
and large shared objects with code spread over a wide virtual address range are
likely to see performance improvements by grouping frequently called functions
into as few naturally aligned 2MB virtual address ranges as possible. The perf
record and perf script commands can help determine the observed program
counter addresses over a span of time. To determine whether a given application
might be a candidate for this optimization, we recommend that you count the
number of observed address ranges in the perf output.

For large applications and/or libraries that are confirmed to access more than 30
such ranges in quick succession, this form of optimization might yield speedups of
as much as 50%. To achieve this, there are several ways to rearrange the linked
binary/binaries to group frequently called functions or group functions with the
other functions that they typically call. For example, some forms of automated
Profile-Guided Optimization (PGO) might be beneficial in this scenario. The `perf record/perf script` output can also be used to capture the names of the most frequently called functions. By compiling with `-ffunction-sections`, the frequency-sorted list of observed function names can be used to produce a linker script that groups the "hot" functions nearby in memory, which achieves the same goal.

Optimizations to decrease code size generally might be beneficial because smaller code naturally spans fewer 2MB ranges. For example, if you are using `gcc -O3`, consider using `-fno-ipa-cp-clone`. 
9. Performance Tuning for Grace Hopper-Based Applications

9.1. Overview of GH200 Grace Hopper Superchip

NVIDIA GH200 Grace Hopper Superchip main architectural features are its NVLink-C2C interconnect, which provides memory coherency within the superchip enabling access to large CPU Memory and NVLink Switch System which extends GPU accessible memory beyond a single superchip. NVLink Switch System and NVLink C2C are high-bandwidth and low-latency interconnects, providing up to 900 GB/s total bandwidth when accessing up to 144 TB of memory. NVIDIA Grace Hopper also provides a 1:1 GPU-to-CPU ratio, making the platform excel at GPU-heavy, CPU-heavy, and truly heterogeneous workloads that intensely engage both the GPU and the CPU.

This section provides information about how these unique hardware features can be tuned for applications, enable new applications, and improve developer productivity when writing new applications or incrementally accelerating existing applications on the GPU.

We highlight major memory and NUMA optimizations that help tune the performance at system level for Grace Hopper.

For example, the NVIDIA Hopper GPU Architecture accelerates dynamic programming by using DPX Instructions. These instructions benefit applications in industries including healthcare, robotics, quantum computing, and data science. Tuning application performance for the Hopper GPU architecture is described in the NVIDIA Hopper Tuning Guide.

9.2. Memory Allocators

This section provides information about memory allocators that are available in CUDA and generic Linux system allocators.
9.2.1.  Grace Hopper: A Unified Memory System

For a brief overview of the IOMMU/SMMU systems in a Grace Hopper system, refer to Input-Output Memory Management Unit Passthrough. Refer to Page Size for information about physical page sizes.

On a Linux system, a system of caches, called translation-lookaside buffers (TLBs), is used to cache translation from virtual addresses to physical addresses, and these pages are used to stage the translation.

On a Grace Hopper system, the memory space shares a common page table for CPU and GPU pages. Since the physical address spaces are separate, ARM’s SMMU and the GPU’s GMMU and the respective TLBs have to interact accordingly.

Due to the high data transfer bandwidth across the NVLink C2C, Grace Hopper uses a specific unit to improve performance of address translations referred to as the Address Translation Service Translation Buffer Unit (ATS-TBU). This translation mechanism will be referred to as ATS. For an application, memory has to be allocated using the right mechanisms to use ATS for transfers across the C2C (refer to Overview of Allocators for more information about ATS support through various allocators).

Additionally, Grace Hopper supports coherency between the CPU and GPU memory spaces. This allows for an easier programming model across the system, but it can come at a performance cost in some scenarios (refer to NVIDIA NVLink C2C System for more information).

9.2.2.  Overview of Allocators

Grace Hopper supports many allocators, and each allocator has different features.

Table 9-1 attempts to provide a comprehensive overview of the most common allocators used on Grace Hopper systems.

Table 9-1.  Overview of Allocators on a Grace Hopper System

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cudaMallocHost / cudaHostAlloc / cuMemCreate</td>
<td>CPU, GPU</td>
<td>CPU</td>
<td>No</td>
<td></td>
<td>ATS/EG M</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
<td>-----------------------------</td>
<td>----------------</td>
<td>----------------------</td>
<td>------------</td>
</tr>
<tr>
<td>[location type HOST*]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cudaMalloc Managed</td>
<td>CPU, GPU</td>
<td>First touch / hint</td>
<td>Yes</td>
<td>System (CPU-resident) 2MB (GPU-resident)</td>
<td>ATS/EGM (CPU-resident) GMMU (GPU-resident)</td>
<td>Yes [7]</td>
</tr>
</tbody>
</table>

[1] Can be overridden with cudaMemAdvise. Even if access-based migrations are disabled, if the backing memory space is full, memory might migrate.

[2] The default system page size is 4K or 64K (refer to Page Size), the default page size for GPU translations is 2MB.

[3] For mmap, file-backed memory is placed on the CPU by default, unless specified otherwise through mbind or cudaMemAdviseSetPreferredLocation.


[5] Default system page size, unless huge page size was explicitly specified (for example, with mmap MAP_HUGETLB / MAP_HUGE_SHIFT). In this case, any huge page size configured on the system is supported (refer to Huge Pages).

[6] Avoiding migrations due to access might require hardware support (for example, NIC with On Demand Paging).

[7] NIC access to managed memory requires a NIC with On Demand Paging support.

[8] Access to CPU-resident managed memory through NVLink requires EGM.

[9] No translation between CPU and GPU address spaces.

[10] Requires nvidia-peerem kernel module or kernel with dmabuf support.

### 9.2.3. Page Sizes

Tasks that involve excessive random memory access to a large working set might encounter limitations because of TLB misses. You can minimize TLB misses by using larger page sizes.

This section provides information about major memory allocation methods that can use larger page sizes. Refer to Operating System Settings for more information about the set up and operation system specific details.
9.2.3.1. Assessing the Need for Huge Pages

Using huge pages can lead to occasional latency spikes and larger memory fragmentation. Therefore, developers should carefully analyze their workload before enabling larger page sizes. `perf stat` can be used to fetch TLB misses. Refer to Measuring Workload Performance with Hardware Performance Counters for more information about `perf`.

Tracking page fault events (faults), TLB load and miss events (dTLB-loads, dTLB-load-misses), and LLC events (cache-misses, cache-references) provide enough information to help you decide whether you need huge pages. All trackable events can be listed using `perf list`.

Here is the sample output:

```
perf stat -e 'faults,dTLB-loads,dTLB-load-misses,cache-misses,cache-references'
<EXECUTABLE> <ARGUMENTS>

Performance counter stats for '<EXECUTABLE> <ARGUMENTS>':

    70,080 faults:u
    34,285,589 dTLB-loads:u
    30,657,940 dTLB-load-misses:u  #  89.42% of all dTLB cache hits
    51,676,464 cache-misses:u      #  44.81% of all cache refs
    115,346,452 cache-references:u
...
```

When you encounter significant TLB miss to LLC miss ratios, we recommend that you explore the potential performance improvement by using huge pages.

9.2.3.2. Using Transparent Huge Pages

To enable Transparent Huge (THP) for a specific memory range, the MADV_HUGEPAGE flag has to be defined in madvice mode.

```
Note: Linux THP support does not ensure the automatic allocation of huge pages. Allocating memory using posix_memalign increases the likelihood of the memory being backed by huge pages.
```

Here is the sample output:

```
void *ptr = nullptr;
size_t huge_page_size = 1<<21; // 2MiB;
```
size_t nbytes = 1<<30; // 1GiB;
posix_memalign(&ptr, huge_page_size, nbytes);
madvise(ptr, nbytes, MADV_HUGEPAGE);

**Note:** posix_memalign is only useful for allocations that have at least one huge page.

9.2.3.3. Using mmap

You can allocate huge pages using `mmap`, the allocated memory size must be perfectly divisible by the page size, and the MAP_HUGETLB flag has to be set.

Here is the sample output:

```c
// Defining MAP_HUGE_2MB flag which is not defined by default.
#define MAP_HUGE_SHIFT 26 // First 5 bits define the log2 of page size.
#define MAP_HUGE_2MB (21 << MAP_HUGE_SHIFT)
size_t huge_page_size = 1<<21; // 2MiB;
nsize_t npages = 1<<30 / huge_page_size + 1;
void *ptr = mmap(NULL, npages * huge_page_size,
                   PROT_READ | PROT_WRITE,
                   MAP_PRIVATE | MAP_ANON | MAP_HUGETLB | MAP_HUGE_2MB,
                   -1, 0);
```

Refer to [Huge Pages](#) for more information about how huge pages are obtained from designated pools.

9.3. NVIDIA NVLink C2C System

This section provides information about the NVIDIA NVLink C2C System.

9.3.1. Accessing System Memory

As described in [Memory Allocators](#), CPU and GPU threads can access memory that is allocated on Grace Hopper, with the exception of CPU threads that access memory allocated using cudaMalloc. To achieve best possible performance for data accesses, some considerations are listed in this section.
If an application allows the GPU work on data and shares a result with the CPU, here are some possible options:

- The GPU writes its result to GPU-local memory.
  
The result is transferred by using cudaMemcpyAsync (or similar), and the CPU accesses the transferred data.
- The GPU directly writes its result to CPU-local memory.
  
The CPU accesses this data.
- The GPU writes to GPU-local memory.
  
The CPU directly accesses this data.

If independent work can be scheduled on the GPU while the result is transferred/accessed by the CPU, options 1 or 3 are preferred. If the GPU is starved until the CPU has accessed the result, option 2 might be preferred. The GPU can generally write at a higher bandwidth than the CPU can read. Unless multiple CPU cores are used to access results from the GPU, the best option is to let the GPU write the result through C2C before accessing it by the CPU.

In the scenario above, the CPU and GPU also need to synchronize before the CPU can access the results from work on the GPU. If the GPU continues to work while the CPU should read/access previous results, atomic counters or system-wide barriers have to be used to signal the CPU that results are available. A question that might come up is whether those atomic counters or system-wide barriers should be allocated and populated in CPU-backed memory or GPU-backed memory.

Benchmarks show that, as long as a CPU core participates in system-wide atomics or barriers, there is no significant difference whether the memory for this is backed by the CPU or GPU.

### 9.3.2. Transferring Data Across C2C

This section provides information about transferring data across C2C.

#### 9.3.2.1. Using CUDA Copy Engines

In general, for contiguous data, using the CUDA copy-engines (CEs) is preferred because operations performed by CEs can be overlapped with the CPU and the GPU processor work. CEs are used in all cudaMemcpy* and cudaMemcpyAsync APIs from the CUDA toolkit.

Currently, the performance of cudaMemcpyAsync is low and will be improved in a future CUDA toolkit. Until then, we recommend that you use cudaMemcpyAsync and explicitly specify the direction of transfer.
CEs are generally used for transfers between CPU and GPU and between peer GPUs in an NVLink-connected system. Due to the limited total number of CEs, device to host transfers might be limited to a lower bandwidth than using the GPU SMs to perform the transfer. In specific cases, CPU- or GPU-based transfers might be preferred even for contiguous data (refer to Accessing System Memory from the CPU for more information).

### 9.3.2.2. Atomic Accesses

In general, atomic accesses can be performed by the CPU using `std::atomic` or `std::atomic_ref` from the C++ standard library header `<atomic>`. When allocating `std::atomic` through a specialized allocator, such as `cudaMallocManaged`, call `std::atomic_init` to initialize the allocated memory.

Similarly, on the GPU, atomic accesses can be performed using `cuda::atomic` or `cuda::atomic_ref` from the CUDA toolkit as defined in `<cuda/atomic>`. When allocating `cuda::atomic` through a specialized allocator such as `cudaMallocManaged`, to initialize the allocated memory, you can call `cuda::atomic_init`.

On Grace Hopper, because GPU threads can access system-allocated memory like any other type of memory, atomics can easily be shared between CPU and GPU threads. However, atomics using `std::atomic*` and `cuda::atomic*` cannot be mixed. If you want atomic accesses from both CPU and GPU threads, we recommend that you use the `cuda::atomic*` variants and allocate the backing memory using an allocator that allows access from the CPU and the GPU. If most accesses are made by GPU threads, we recommend that the physical backing memory is placed on the GPU memory.

On Grace Hopper, because of coherency in the memory spaces between the CPU and GPU, atomic latencies can be larger than on systems without coherent memory spaces.

### 9.3.2.3. Pseudo-Random Access

When transferring data across C2C where the access pattern is random or unknown across a working set, the application puts more pressure on the TLB caches. If the working set size is larger than the maximum range supported by the entire TLB, the transfer will be limited by the address translation rate.

If TLB cache miss rates are a potential limiter of your application, an important resource is to increase the page size of the allocations used by your application (refer to Page sizes).
Appendix A: References to Help Improve Performance

The following guides will help you improve application performance:

- NVIDIA Grace CPU Benchmarking Guide.
- Dynamic programming instructions in this blog.
- Hopper Tuning Guide
Notice
This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality. NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.
Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.
NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.
NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.
NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.
No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.
Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.
THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.
Trademarks
NVIDIA, the NVIDIA logo, NVLink are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.
VESADisplayPort
DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.
HDMI
HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.
Arm
Arm, AMBA, and ARM Powered are registered trademarks of Arm Limited. Cortex, MPCore, and Mali are trademarks of Arm Limited. All other brands or product names are the property of their respective holders. "Arm" is used to represent ARM Holdings plc; its operating company Arm Limited; and the regional subsidiaries Arm Inc.; Arm KK; Arm Korea Limited; Arm Taiwan Limited; Arm France SAS; Arm Consulting (Shanghai) Co. Ltd.; Arm Germany GmbH; Arm Embedded Technologies Pvt. Ltd.; Arm Norway, AS, and Arm Sweden AB.
OpenCL
OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.
Copyright
© 2023 NVIDIA Corporation & Affiliates. All rights reserved.