# Coherent Driver-based Memory Management (CDMM)

Whitepaper

V 1.0 | October 2025

# Document History

1.0

| Version | Date | Description of Change |
|---|---|---|
| 1.0 | October, 2025 | Initial release |

# Table of Contents



# List of Tables

# Introduction

NVIDIA systems such as GH200, GB200, and GB300 with an NVIDIA CPU and NVLink chip-to-chip connection provide "hardware coherent" memory [8]. This enables both CPU and GPU memory to be directly addressed from either CPU or GPU. This capability is not present on PCIe-connected systems.

On GH200+, the default shipping configuration presents GPU memory as a NUMA (non-uniform memory access) node to the operating system. This allows the OS to manage GPU memory together with CPU memory, which leads to some behavioral differences compared to a PCIe-attached GPU setup. In particular, the operating system may select GPU memory for unexpected or surprising uses, such as caching files or avoiding out-of-memory (OOM) conditions from an allocation request. For some applications and workflows, especially those that have been optimized for a particular layout of CPU and GPU memory (like Kubernetes), these differences may be undesirable.

To accommodate those situations, the NVIDIA Driver provides an alternative mode of operation, called CDMM (Coherent Driver-based Memory Management), that does not expose GPU memory to the operating system as a NUMA node. CDMM mode leaves the GPU NUMA memory nodes in place but reports to the OS that the amount of memory in each GPU node is zero. For the remainder of this document, the discussion will refer to either "CDMM mode" or "NUMA mode".

# NUMA Mode

The total memory on a CPU and GPU system consisting of sysmem + vidmem is exposed as NUMA nodes to the Linux kernel.

## Features

- Unified Memory Management: Standard Linux APIs like `malloc()` and `mmap()`, as well as CUDA APIs, can allocate memory on both CPU and GPU.
- Inter-process communication (IPC): Users can utilize OS-provided memory primitives to enable efficient inter-process communication for both CPU and GPU memory, facilitating complex multi-process applications.
- Dynamic Memory Migration: Memory can be moved between CPU and GPU memory nodes using user-space APIs or automatically by the kernel, optimizing resource utilization.

## Potential Side Effects

- While Linux NUMA offers flexibility by unifying device and system memory under native Linux memory management, it causes GPU memory to be treated as a generic memory pool. Some users may prefer to isolate GPU memory from general operating system functions. CDMM is designed for these users.
- Although Linux provides isolation mechanisms through a rich set of user-space tools, memory pressure can lead to memory usage spilling into GPU memory nodes. While this is typical NUMA behavior, it might be undesirable for users who closely monitor GPU memory usage and wish to reserve it for very specific, performance-critical tasks.

# CDMM Mode

The CDMM approach is inspired by the PCIe-attached model, in which GPU memory is separate from CPU system memory (sysmem). System memory is used for system tasks and operations, including running VMs and containers in the environment and GPU memory is used for accelerated computing. This mode allows the Linux kernel to operate without bringing online the GPU memory as a NUMA node and is designed for applications or users that want to manage and use GPU memory separately from system memory.

CDMM still allows applications to use the coherency features of the platform.

The CPU memory is managed by the Linux kernel while the GPU memory is managed by the NVIDIA device driver instead of the Linux kernel. CDMM is enabled using a module configuration parameter that is passed in during driver load.

## Features

- Isolated GPU Memory: CDMM allows GPU memory to function independently of the system's main memory, preventing the OS from making inefficient, potentially intrusive or unwanted allocation decisions for GPU-specific workloads. This isolation is important for performance-sensitive applications.

- Direct Driver Control: The NVIDIA device driver directly manages GPU memory. This means system administrators and application programmers have more granular control, bypassing potential OS interference and ensuring memory is optimally allocated for GPU operations.

- Platform Coherency: CDMM leverages platform coherency without exposing all memory as OS-managed. This allows for efficient data transfer between CPU and GPU while maintaining driver-level control over GPU memory.

# Guidelines for using NUMA and CDMM Modes

## Application-Specific Memory Management

**NUMA Mode:** Beneficial for applications that rely upon the OS NUMA APIs, and upon the OS's behavior of managing total system memory (system memory + video memory)

**CDMM Mode:** Ideal for applications that need direct control over GPU memory, bypassing (and avoiding) OS control.

## Memory Pooling

**NUMA Mode:** Allows memory pooling where both GPU memory and CPU memory can act as a larger single pool of memory. Workloads can benefit from being able to aggregate memory and rely on explicit or implicit bandwidth management to make best use of the pool.

**CDMM Mode:** CDMM's driver-managed approach prevents the OS from using GPU memory as a part of a larger memory pool. This prevents it from being used for page cache and other related memory management activities from the OS such as reclaim are not required. CDMM ensures GPU memory is primarily dedicated to GPU-specific data.

## GPU Memory Usage: Visibility and Measurement

**NUMA Mode:** Standard system monitoring tools report GPU memory utilization of the entire integrated memory pool of GPU memory and system memory and can be further filtered using the NUMA topology. The entire memory utilization of the system as a whole can be viewed.

**CDMM mode:** Provides fine-grained control and enhanced visibility into GPU memory. Because the driver directly manages GPU memory in CDMM, system administrators and developers gain a clearer understanding of actual GPU memory consumption, simplifying performance diagnosis and optimization.

# Cluster Administrator and CUDA Developer Considerations

In CDMM mode, user space applications do not have direct access to memory on the GPU but will need to use CUDA APIs to allocate and use GPU memory. Application code (kernels) executing on the GPU has direct access to CPU memory.

## Cluster Administrator

- When a system is configured in CDMM mode, the system will still have firmware tables and NUMA nodes corresponding to the GPU. The GPU NUMA node(s), however, will not have any memory present.

- In CDMM mode, using NUMA aware allocation tools such as `numactl` or using `mbind()` will have no impact when applied to GPU memory. Even though nodes corresponding to the GPU are visible, they have no memory associated with them. It is recommended that these tools not be used in CDMM mode for any GPU memory management, they can be used to manage system memory.

## CUDA Developer

- Migration of system allocated memory: In the current implementation of CDMM, system allocated memory will not be migrated to the GPU. System allocated memory can still be accessed by code running on the GPU over the C2C link. Pages will not be migrated even when an application uses the hints below to encourage migration. The impacted APIs include:

  - `cudaMemPrefetchAsync()`
  - `cudaMemPrefetchBatchAsync()`
  - `cudaMemDiscardAndPrefetchBatchAsync()`
  - `cudaMemAdvise(SetPreferredLocation)`

Table 1. CDMM on CUDA Device Attributes in NUMA and CDMM Modes

| Attribute | NUMA Mode | CDMM Mode |
|---|---|---|
| CU_DEVICE_ATTRIBUTE_NUMA_CONFIG | CU_DEVICE_NUMA_CONFIG_NUMA_NODE | CU_DEVICE_NUMA_CONFIG_NONE |
| CU_DEVICE_ATTRIBUTE_NUMA_ID | The current GPU NUMA node ID | The attribute will return an invalid NUMA ID |
| CU_DEVICE_ATTRIBUTE_PAGEABLE_MEMORY_ACCESS | True | True. CDMM will not migrate system allocated memory to the GPU |
| CU_DEVICE_ATTRIBUTE_DIRECT_MANAGED_MEM_ACCESS_FROM_HOST | True | False |

# Kubernetes Implications in NUMA mode

- **Memory over-reporting: Kubernetes** enumerates the amount of system memory it makes available to pods by summing up the total amount of memory it sees on each NUMA node it has access to. Since GPU memory is also exposed as NUMA nodes, Kubernetes mistakenly includes GPU memory in this count. This, in turn, allows pods to request more system memory than is available on a given node, causing out-of-memory (OOM) failures rather than waiting in a pending state until more memory becomes available.

- **Pod memory limits apply to GPU memory, not just system memory:** The Kubernetes pod spec provides a configuration to set a memory limit on the amount of system memory a pod has access to. Since each GPU's memory is exposed as a standard NUMA node, these limits apply to both system memory and GPU memory when using system allocated memory. This breaks the intended contract of the Pod spec API since it was designed to refer only to system memory, not GPU memory. NOTE: If CUDA API is used for allocation, memory limit accounting and enforcement may vary depending on the allocator used.

- **Isolating GPU memory amongst pods:** By default, all pods in Kubernetes have access to all memory across all NUMA nodes. Since each GPU's memory is exposed as a standard NUMA node, all containers can allocate memory on all GPUs in the system, even if they don't have access to those GPUs, thus breaking the desired isolation.

# CDMM Mode Configuration

To enable CDMM, pass a kernel module parameter and value `NVreg_CoherentGPUMemoryMode=driver`, to nvidia.ko.

```
$ sudo modprobe nvidia NVreg_CoherentGPUMemoryMode=driver
```

This can be made persistent across reboots by adding it to `/etc/modprobe.d/` as follows:

```
$ echo options nvidia NVreg_CoherentGPUMemoryMode=driver > /etc/modprobe.d/nvidia-openrm.conf
```

Reloading the driver, typically by rebooting, is required for the kernel module parameter to take effect.

To verify that CDMM is active, run the following:

```
$ grep Coherent /proc/driver/nvidia/params
```

Expected output:

```
$ CoherentGPUMemoryMode: "driver"
```

# CDMM Compatibility Notes

## Default Mode (NUMA vs CDMM)

The current default mode of the driver is CDMM Mode = OFF. However, in Kubernetes-based GPU operator deployments on k8s that use the NVIDIA driver container, CDMM is enabled by default starting from driver 580.65.06 and greater.

## GPU Direct RDMA

CDMM is only supported on the nv-p2p persistent APIs and not on the legacy ones. Please see [9] for more information. With ODP and managed memory, using `CU_MEM_ADVISE_SET_PREFERRED_LOCATION` and setting the preferred location to device memory, the system will observe the behavior mentioned in [10].

## GPU Direct Storage (GDS)

GDS support is available with driver release R580TRD3 and greater.

## vGPU Support

vGPU is currently not supported in CDMM mode. A future driver version will enable vGPU support in CDMM mode.

## MIG Support

MIG is currently not supported in CDMM mode. A future driver version will enable MIG support in CDMM mode.

## CUDA API

Future implementations of CDMM might support remote mapping of pages from the CPU to GPU memory and migration of system allocated memory as well. This will impact the following CUDA attributes `CU_DEVICE_ATTRIBUTE_PAGEABLE_MEMORY_ACCESS` and `CU_DEVICE_ATTRIBUTE_DIRECT_MANAGED_MEM_ACCESS_FROM_HOST`.

# References


1. NVIDIA. NVIDIA Tesla Release Notes, Version 580.65.06
2. NVIDIA. NVIDIA GPU Operator Release Notes.
3. NVIDIA Driver README
4. NVIDIA. CUDA C++ Programming Guide
5. The Linux Kernel: What is NUMA? — The Linux Kernel documentation
6. The Linux Kernel: NUMA Memory Policy NUMA Memory Policy — The Linux Kernel documentation
7. CUDA Runtime API, Version 13.0, Memory Management
8. NVIDIA GH200 Grace Hopper Superchip Architecture Whitepaper
9. NV P2P API. Developing a Linux Kernel Module using GPUDirect RDMA
10. CUDA Driver API Documentation, Version 13.0, Unified Addressing

## 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 and the NVIDIA logo 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.

## Copyright



NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com