NVIDIA GB200 NVL Partition

User Guide

DU-12143-001 | April 2025

Document History

DU-12143-001

Version	Date	Authors	Description of Change
0.1	February 21, 2025	MG and the MNNVL team	The initial release.
0.2	April 22, 2025		Updates to Trunk Link Fault handling, NVOS CLI, Multicast recommendation

Table of Contents

1. Overview

This chapter provides an overview of partitions in NVIDIA® GB200 NVL systems.

1.1 Introduction

NVIDIA NVLink™ is a high-speed interconnect that allows multiple GPUs to communicate directly. NVLink Multi-Node is an NVLink network where multiple systems are interconnected to form a large GPU memory fabric also known as an NVLink Domain.

NVLink Multi-Node is implemented using the NVIDIA GB200 reference architecture, and the architecture designs are provided in Table 3.

Table 1. Supporting Documentation and References

Term	Description
NVLink	A high-speed link to enable GPU-to-GPU communication.
Compute Node	An OS instance with at least one GPU.
Switch Node	An OS instance with at least one NVLink switch.
Access Link	An NVLink between an NVIDIA NVSwitch™ and a GPU.
Trunk Link	An NVLink between NVSwitches.
NVLink Domain	A set of nodes that can communicate over NVLink.
FM	Fabric Manager The NVLink Network Control Plane service is provided by the FM service.
SM/NVLSM	NVLink Subnet Manager A service that originates from NVIDIA InfiniBand Switches and has the modifications to effectively manage NVSwitches.
NMX-C	NMX-Controller Service that provides management capabilities for an NVLink domain.
Control Plane	The software suite that includes FM, SM, and NMX-C.
Tenant	A user or job scheduler that runs an NVIDIA CUDA® job.
Admin	Entity that is responsible for allocating partitions to a tenant.
NVLink Partition	A subset of GPUs in a hardware isolation boundary.
Default Partition	A Control Plane created an NVLink partition that allows all GPUs in the NVLink Domain to communicate.
User Partition	An NVLink partition that is created and managed by an admin.
Inter-Chassis Partition	A partition that uses switch-to-switch NVLinks (also known as trunk links).
Intra-Chassis Partition	A partition that uses switch-to-GPU NVLinks (also known as access links).
Partition Id	Logical identifier for a partition in an NVLink domain.

Term	Description
NVOS	NVIDIA Networking OS, which was previously known as MLNX-OS. NVOS is used as the Switch OS for L1 NVSwitch Trays.
OFR	Out For Repair.
Chassis Id	Unique logical identifier of a chassis in an NVLink Domain.
Slot Number	Unique identifier of a slot in a chassis.
Host ID	Logical identifier of a CPU instance in a slot.
Gpu ID	Logical identifier of a GPU in a host.
Gpu UID	Unique identifier of a GPU.
Gpu Location	Location of a GPU in an NVLink Domain that includes chassisId, slotNumber, HostId, and GpuId.

Table 2 provides a list of supporting documentation that are referred to in this guide.

Table 2. Supporting Documentation and References

No	Document Title	Documentation
1.	NMX-Controller gRPC APIs Documentation	https://docs.nvidia.com/networking/networking-nvlink/index.html
2.	NVIDIA GB200 NVL System Bring-up Guide for 0.9.00 Software	Contact your OEM vendor for support.
3.	Grace-Hopper I/O Virtualization Reference Code Release Notes	Contact your OEM vendor for support.
4.	NVIDIA NVOS User Manual: • QS version • PS version	https://docs.nvidia.com/networking/networking-nvlink/index.html
5.	NVIDIA GB200 NVL Service Flow User Guide	Contact your OEM vendor for support.

Table 3. GB200 Reference Architecture Designs

NVLink Domain	# Compute trays per chassis	# Switch Trays per chassis	Number of CPUs/GPUs in a tray	# Chassis
GB200 NVL36x2	9	9	C2G4	2
GB200 NVL36x2	18	9	C1G2	2
GB200 NVL72	18	9	C2G4	1

Figure 1 shows the GB200 2xNVL36 platform with 18 compute trays and nine switch trays per chassis.

Figure 1. The GB200 NVL36x2 Platform with 18 Compute Trays and Nine Switch Trays Per Chassis

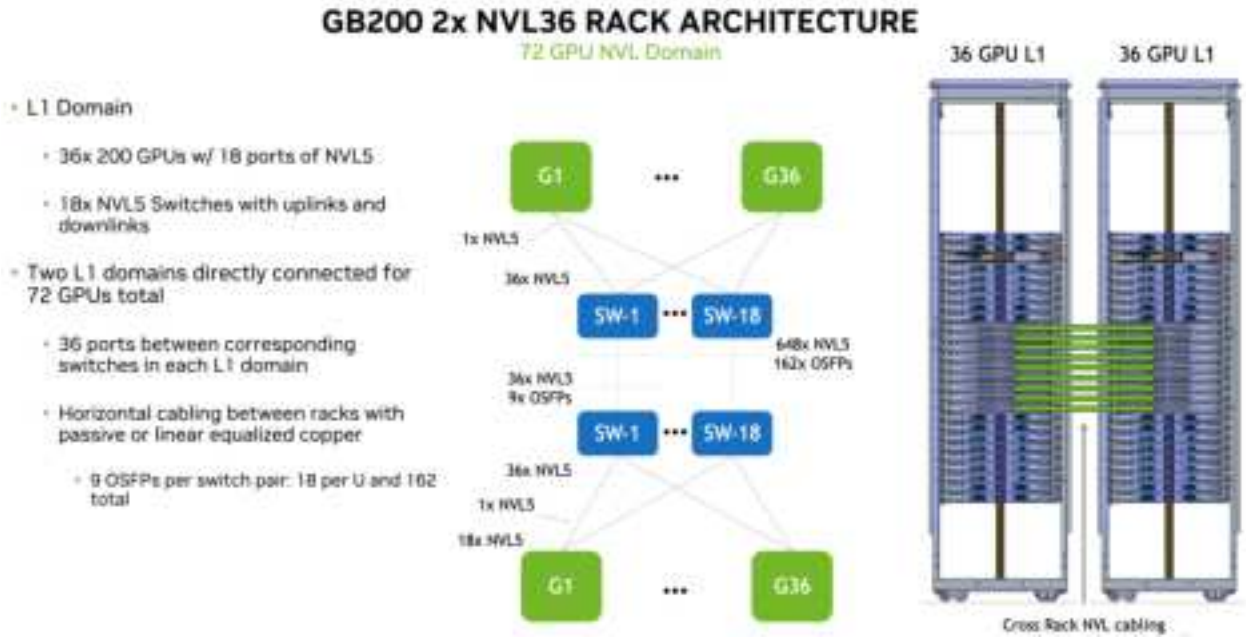
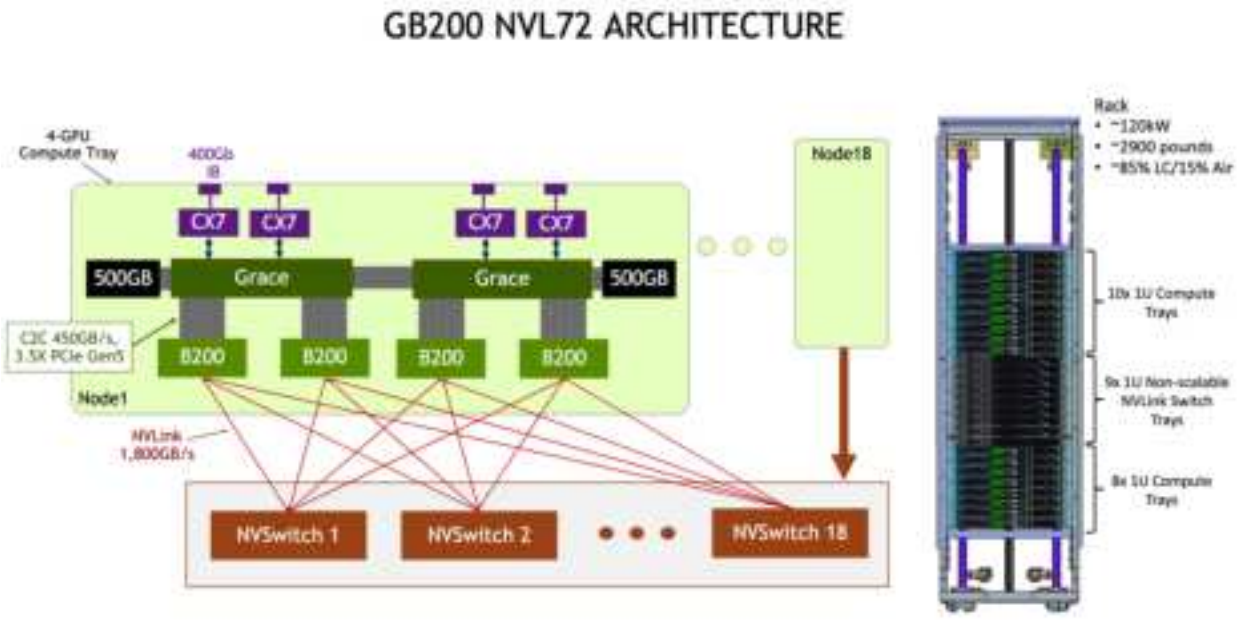


Figure 2 shows the GB200 NVL72 platform with 18 compute trays and nine switch trays.

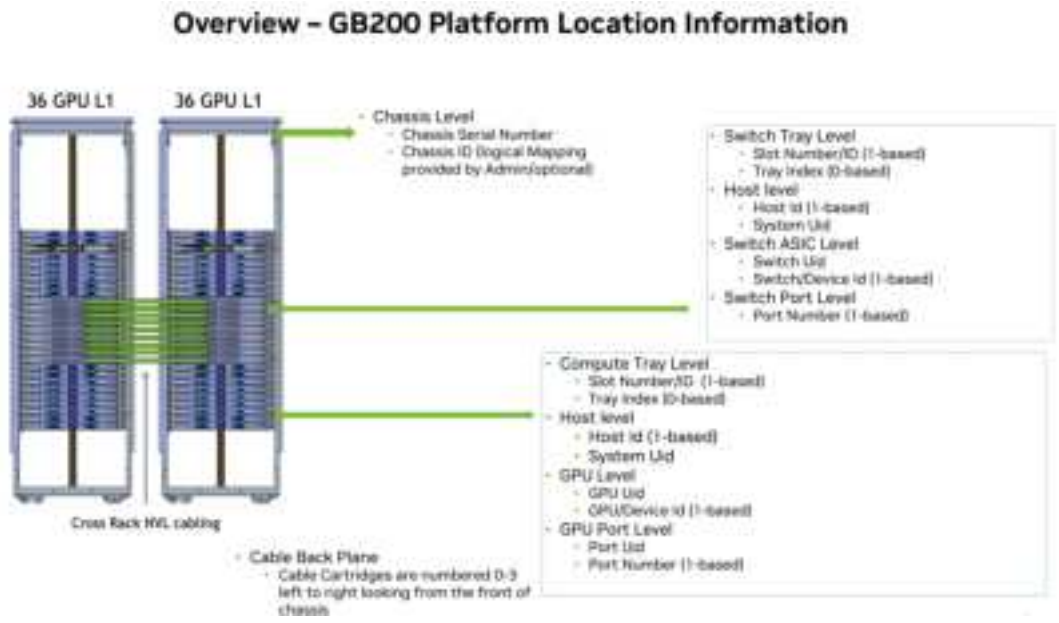
Figure 2. The GB200 NVL72 Platform with 18 Compute Trays and Nine Switch Trays



1.2 Platform Location Information

NVLink Multi-Node uses the platform location information as a building block to identify resources. Figure 3 illustrates the taxonomy of this information.

Figure 3. GB200 Platform Location Information



The platform location information is composed of layers of the following identifiers:

- Chassis
  - Serial Number
    - Unique identifier of a chassis.
      The serial number is read from an EEPROM on the cable cartridge in the backplane.
  - ID
    - A logical ID that maps to a chassis serial number.
      If users do not provide this optional mapping during provisioning, the Control Plane assigns the map during the resource discovery phase.
- Switch tray
  - Slot Number/ID
    Absolute physical number of a slot.
  - Tray Index
    Relative physical number of a slot according to its type(switch/compute).
  - UID
    Unique identifier of a switch tray.
  - Host ID
    Unique identifier of an OS domain that is running in the switch tray. This ID represents a switch node.

    - Switch ID
      Logical identifier of a switch ASIC in a switch node.
    - Switch UID
      Unique identifier of a switch ASIC in a switch node.
    - Switch Port Number
      Unique identifier of a switch port in the ASIC.
- Compute tray
    - Slot Number/ID
      Absolute physical number of a slot.
    - Tray Index
      Relative physical number of a slot according to its type(switch/compute).
    - UID
      Unique identifier of a compute tray.
    - Host ID
      Unique identifier of an OS domain that is running in the compute tray. This ID represents a compute node.
    - GPU ID
      Logical identifier of a GPU in a compute node.
    - GPU UID
      Unique identifier of a GPU in a compute node.
    - GPU Port Number
      Unique identifier of a port in the GPU.

The following links provide additional information about the Platform Location:

- nvidia-smi
- NVOS CLI
- NMX-C GRPC API

2. NVLink Partitioning

This chapter provides information about NVLink Partitioning.

2.1 Overview

NVLink Partitioning allows an NVLink domain to be divided into at least one hardware isolation domain, and each domain is referred to as an NVLink Partition, and isolation is configured by the Control Plane. The routing and fabric address assignment are configured on the GPUs so that memory transactions between the GPUs in the partition are confined to the partition boundary. NVLink Partitioning offers several use cases:

- The GPU fabric can be fragmented and used for small, medium, and large-sized workloads.
- Offers multi-tenancy.

NVLink Partitioning has the following properties:

- Ensures that data paths are isolated in each partition.
- Each NVLink is allocated to one tenant, which ensures exclusive access.
- Isolation is configured and is managed by the Control Plane.
- NVLinks are not shared across different partitions.
- Partitions can be created with an arbitrary set of nodes in the NVLink domain.
  - Partitions created with trunk links are called Inter-Chassis partitions.
  - Partitions created without trunk links are called Intra-Chassis partitions.
- Partitions cannot span across NVLink domains.
- A GPU belongs to only one partition at a time.
- Partitions can exist without any GPUs allocated to them.

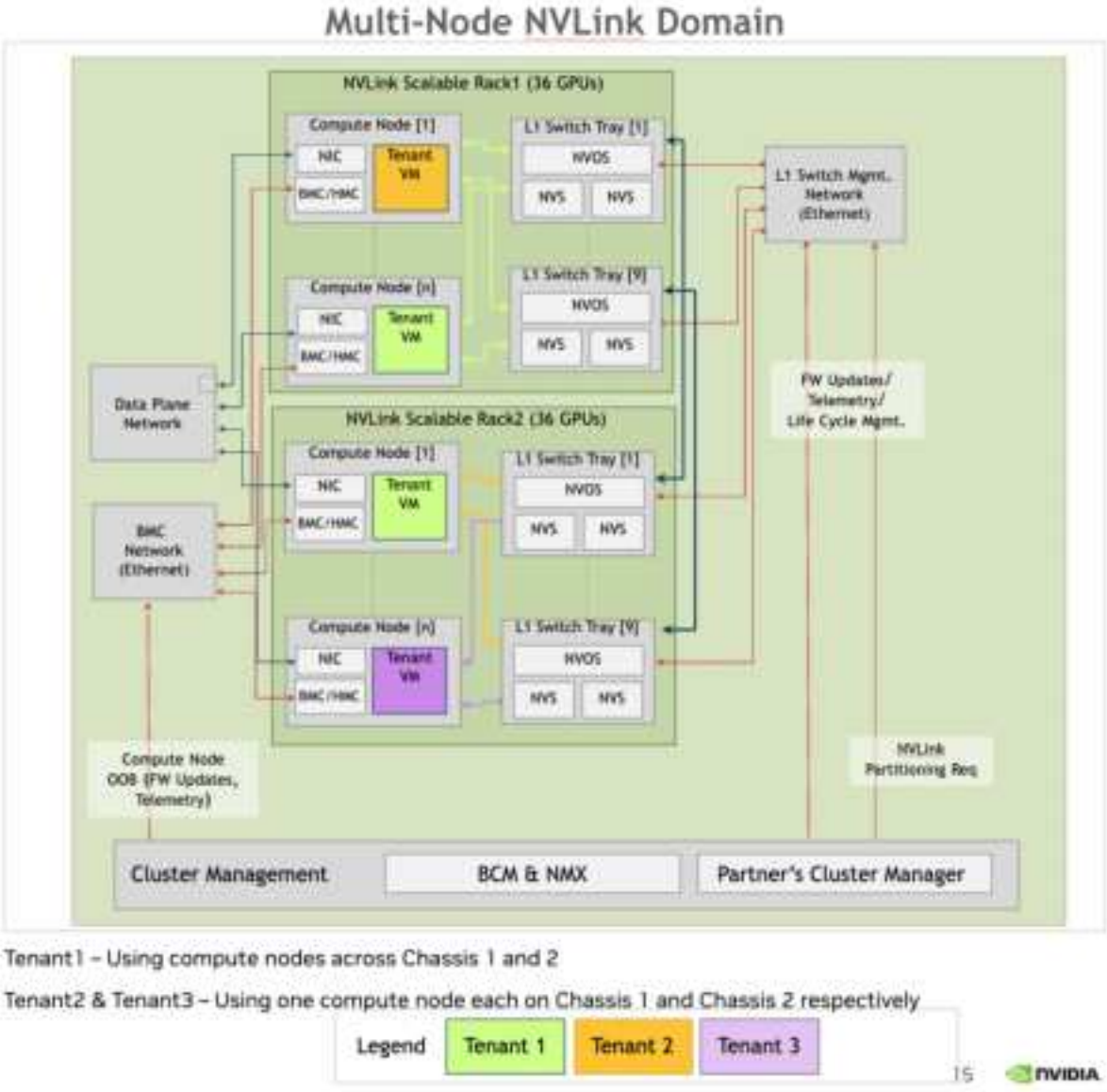
Here are the limits of an NVLink Partition:

- Partitions can exist with no resources allocated to them, so the maximum number of partitions in an NVLink Domain is the number of supported `partitionIds`.
  For GB200 platforms, this value is set to 32766(0x1- 0x7FFE).
- For partitions that are created with at least one GPU, the maximum number of partitions in an NVLink Domain is the total number of GPUs in the NVLink Domain.
  On a NVL36x2 system, for example, the maximum number of partitions of size one GPU is 72 because the NVLink Domain supports 72 GPUs.
- The size of the largest partition is the maximum number of GPUs in that NVLink Domain.
  In this case, one partition is assigned with all the GPUs in the NVLink domain. On a NVL36x2 system, for example, the largest partition is one with 72 GPUs because the NVLink Domain supports 72 GPUs.

Figure 4 illustrates an example of how partitions can be created in a Multi-Node NVLink Domain that has two chassis with 36 GPUs per rack. Chassis 1 and 2 host an Inter-Chassis

partition where multiple green VMs and one per compute node are spawned and are allocated to Tenant 1. Chassis 1 also hosts a partition where the orange VM is assigned to Tenant 2. Chassis 2 also hosts a partition where the purple VM is assigned to Tenant 3. Figure 4 illustrates how NVLink partitioning supports a multi-tenancy model that efficiently allocates resources in the rack.

Figure 4. Multi-Node NVLink Domain



2.1 Types of Partitions

A partition type indicates how the resources that belong to a partition are identified. Here are the partition types:

- Location Based
- GPU UID Based
- Zero GPU

When using the same partition type to manage a partition, there are strict rules (refer to Modify the Partition (AddGpusToPartition/RemoveGpusFromPartition).

2.1.1 Location-Based GPUs

Each GPU is uniquely identified by its Location Information (refer to Platform Location Information), which is a hierarchy of identifiers that are used to uniquely identify the GPU. Here is a list of the identifiers:

- Chassis ID
- Slot Number
- Host ID
- Gpu ID

A location-based partition is bound to the resources in the assigned location. The location is static, but the resources in the location are dynamic. This implies that if a compute tray that belongs to a partition is moved to a different slot, the partition loses the GPUs that migrated with the compute tray.

A location-based partition can also be created before GPUs are discovered in that location. However, GPUs are allocated to the partition, and routing is configured after discovery and initialization is complete.

2.1.2 GPU UID-Based Partitions

A GPU has a Unique Identifier (UID), which is a 64-bit value, assigned while it is manufactured. This is different from the GPU UUID, which is alphanumeric. A GPU UID-based partition is bound to the GPU UIDs when the UIDs move in the NVLink domain. The GPU location is only used to track the association of the GPU UIDs to its physical location. A UID must be assigned to the GPU before adding it to a partition.

Note Here is some additional information:

- To manage an NVLink Domain's partitions, we recommend that you stick to one partition type (Location or UID based).

  This helps an admin streamline partition workflows.
- When a compute tray moves from a UID-based partition to a created location-based partition, the tray is assigned to the location-based partition and the UID-based partition loses the compute tray (refer to Maintenance Flow for a Compute Tray for more information about these scenarios.

- Location-based partitions can identify a partition that uses the location information of its resources.

  GPU UID-based partitions can identify a partition using a specific list of GPUs. Selecting a partition type depends on the requirement of the application that manages partitions.

2.1.3 Zero-GPU Partitions

A partition can also be created with no GPUs, for example, without specifying UIDs or locations. The GPUs can be allocated to it later by specifying the type. When you create the partition, the partition type is undefined, but after the partition is allocated with a resource, the type is updated to UID or location depending on how users allocate resources. This also provides a transition path for a partition type from location-based to UID-based and vice versa. Users can modify a partition to make it a zero-GPU partition and switch its type after this point.

The primary use case for a zero-GPU partition is to facilitate compute tray maintenance. A partition that is reserved for compute tray maintenance is referred to as an Out For Repair (OFR) partition. Refer to Maintenance Flow for a Compute Tray for more information about OFR partitions.

3. Creating a Default Partition

An NVLink Domain can be configured to boot up with a Default Partition, so that all GPUs can communicate with each other. This special partition encompasses all GPUs in the NVLink Domain so that GPUs can share memory. The Default Partition can be modified or deleted using the NMX-C GRPC API or NVOS (refer to GRPC APIs and NVOS CLIs for more information).

The following Fabric Manager configurations are used to manage the Default Partition:

- `MNNVL_ENABLE_DEFAULT_PARTITION` (Default is 1)
  - Determines whether a Default Partition needs to be created at boot up.
  - Allowed values:
    - 0: Do not create a Default Partition.
    - 1: Create a Default Partition.
- `MNNVL_DEFAULT_PARTITION_TYPE` (Default is 2)
  - Sets the type of the Default Partition
  - Allowed values:
    - 1: Use GPU Location.
    - 2: Use GPU UID.
- `MNNVL_DEFAULT_RESILIENCY_MODE` (Default is 2)
  - Determines the resiliency mode for a Default Partition and when it is unspecified on a user partition.
  - Allowed values:
    - `RESILIENCY_MODE_FULL_BANDWIDTH` (1)
    - `RESILIENCY_MODE_ADAPTIVE_BANDWIDTH` (2)
    - `RESILIENCY_MODE_USER_ACTION_REQ` (3)

Refer to Partition Fault Handling for more information about the impact of selecting a resiliency mode based on the partition behavior.

The Default Partition automatically accumulates GPUs as they are discovered, and this is the *active phase* of the Default Partition. When the system boots, the partition does not need to have any of its compute trays populated. Compute trays might be inserted later and kickstart the *active phase*, which ensures that the inserted compute trays are allocated to the Default Partition.

In a multi-chassis NVLink domain, the Default Partition becomes an Inter-Chassis partition where all trunk links are already allocated to this partition. A trunk link addition, removal, or failure can disrupt the CUDA applications that are running in the partition. The preallocation of trunk links ensures that traffic disruption does not happen through the *active phase*.

The Default Partition is location-based or GPU UID-based.

3.1 UID-Based Default Partition

The partition accumulates GPUs (*active phase*) as new compute trays are discovered.

During the *active phase*, if no compute trays are discovered, the Default Partition can be an Undefined type. The addition of a discovered compute tray updates the type to UID based type. An API request to create, delete, or modify a partition halts the *active phase* of the partition.

The partition now enters the *passive phase*. In this phase, newly discovered GPUs are not allocated to the Default Partition and cannot be used to run a workload until they are allocated to a partition. When a zero-GPU OFR partition is created with a Default Partition, the Default Partition transitions to the *passive phase*.

3.1.1 Inter-Chassis UID-Based Default Partition

A Default Partition can span multiple chassis like an Inter-Chassis partition. In this case, the partition is allocated with the available trunk links in the NVLink domain after the first GPU is discovered. The allocation ensures that as additional compute trays are discovered, they are allocated to the Default Partition without having to adjust the required trunk links to maintain optimum bandwidth. This helps existing CUDA applications to continue running their workloads as more compute trays are added to the Default Partition. When a zero-GPU partition is created with a Default Partition, the partition transitions to the *passive phase*. The compute trays discovered from this point do not belong to a partition and must be explicitly added to one. If an Inter-chassis partition is created out of these compute trays, they might not have an adequate number of trunk links for optimum bandwidth. In this scenario, the admin explicitly runs a reroute operation (refer to Modify Partition (AddGpusToPartition/RemoveGpusFromPartition) on the Default Partition to adjust the number of trunk links again and free the unused ones to be used with the newly created partition.

Note This operation will disrupt CUDA applications that are running in the Default Partition.

3.2 Location-Based Default Partition

The partition accumulates GPUs (*active phase*) as new compute trays are discovered.

The partition never enters a *passive phase* because all resources (locations) were already created for this partition. Newly discovered GPUs are allocated to the Default Partition because of their location. As a result, when a zero-GPU OFR partition is created, the Default Partition continues to accumulate GPUs. In certain scenarios, a location-based Default Partition might be more suitable because the Default Partition always operates in the *active phase*.

Note In Maintenance Flow for a Compute Tray, allocating GPUs to the zero-gpu OFR partition requires sending requests to the Control Plane to remove a tray from the Default Partition and add a tray to the OFR partition.

4. Administrator-Defined Partitions

This chapter provides information about administrator-defined partitions.

When `MNNVL_ENABLE_DEFAULT_PARTITION` is set to 0, the NVLink domain is configured to not have any GPUs communicate with each other at boot time. In this configuration, memory sharing access is disabled. To enable memory access, the system must complete the Partition Creation flow, and memory sharing is limited based on the partition boundary. Partitions created this way are referred to as Administrator-Defined Partitions or User-Defined Partitions.

To manage partitions, the system must reach a certain state where managing partitions is allowed. When the system boots, Fabric Manager (FM) initiates the GPU and the switch discovery process. After being discovered and configured, FM declares the initialization as complete and makes the partitioning service available to the client, and the Control Plane State is set to `NMX_CONTROL_PLANE_STATE_CONFIGURED`.

To manage partitions, the client can make gRPC API requests to the NMX Controller. This section provides information about the GRPC APIs in the context of partition management. The detailed GRPC API specification, the REST APIs and the NVOS CLI support to manage partitions are also available.

Note We recommend that you use the Default or the User Partitions to manage an NVLink Domain but not both types simultaneously. This provides better control over partition behavior.

The User Partition can exist with the Default Partition when maintenance needs to be performed on the compute trays in a Default Partition.

4.1 GRPC APIs

This section provides information about the GRPC APIs.

4.1.1 Basic APIs

This section provides information about the Basic GRPC APIs.

4.1.1.1 Topology and Domain APIs

This section provides information about the Topology and Domain GRPC APIs.

4.1.1.1.1 GetDomainProperties

This API provides an overall view of the resources expected in the NVLink domain and provides the maximum number of expected resources (including but not limited to GPUs, switches, compute nodes, switch nodes, partitions, and NVLinks).

Here is the GRPC response message:

```
protobuf
message DomainProperties {
      uint32 maxComputeNodes
      uint32 maxComputeNodesPerChassis
      uint32 maxGpusPerComputeNode
      uint32 maxGpuNvLinks
      uint32 lineRateMBps

      uint32  maxSwitchNodes
      uint32  maxSwitchNodesPerChassis
      uint32  maxSwitchesPerSwitchNode
      uint32  maxSwitchNvLinks
```

```
      uint32  maxNumPartitions
      uint32  maxNumAlids

      uint32  maxMulticastGroups

      uint32 maxNumPorts
}
```

- **maxMulticastGroups**

  This is the maximum number of multicast groups that are available in the NVLink domain. During partition creation (refer to Compute Nodes and the GPU APIs), this field helps the admin set a limit to the number of multicast groups that are available to a partition.

- **maxNumPartitions**

  This is the maximum number of partitions of size at least one GPU that can be created in the NVLink domain. This is a useful upper limit to determine how many partitions can be created.

4.1.1.1.2 GetDomainStateInfo

This API provides dynamic state information about the NVLink domain.

GRPC response message:

```
protobuf

message DomainStateInfo{
      ControlPlaneState controlPlaneState
      uint32 availableMulticastGroups
      string configStatusDescription
 }
```

- **controlPlaneState**

  Provides the state of the Control Plane and the states that are relevant to the user partition:

  - `NMX_CONTROL_PLANE_STATE_STANDBY`

    When the controller state is `Standby`, the Control Plane does not initiate resource discovery. In this state, Partition operations are unsupported.

    The return code is `NMX_ST_NOT_CONFIGURED`.

- `NMX_CONTROL_PLANE_STATE_UNCONFIGURED`

  When the controller state is `Unconfigured`, the Control Plane is not ready to initiate resource discovery because the expected user configuration is pending. In this state, Partition operations are not ready to be supported.

  The return code is `NMX_ST_NOT_READY`.

- `NMX_CONTROL_PLANE_STATE_ERROR`

  When the controller state is `Error`, the Control Plane is not ready to initiate resource discovery until the incorrect configuration is addressed. In this state, Partition operations are not available.

  The return code is `NMX_ST_NOT_CONFIGURED`.

- `NMX_CONTROL_PLANE_STATE_DEGRADED`

  When the controller state is `Degraded`, the Control Plane operates with limited capability. In this state Partition operations are not ready to be supported.

  The return code is `NMX_ST_NOT_READY`.

- `NMX_CONTROL_PLANE_STATE_CONFIGURED`

  When the controller state is `Configured`, the Control Plane operates in normal capacity. In this state, partition operations are supported, and an appropriate return code is returned.

The API availability based on the Control Plane state is summarized in Table 4

Table 4. Control Plane State and API Support

Control Plane State	Available APIs	Return Codes For Unavailable APIs
`NMX_CONTROL_PLANE_STATE_UNCONFIGURED`	Subscribe GetDomainStateInfo	NMX_ST_NOT_READY
`NMX_CONTROL_PLANE_STATE_DEGRADED`	GetDomainProperties GetDomainStateInfo GetTopologyInfo GetComputeNodeCount GetComputeNodeLocationList GetComputeNodeInfoList GetGpuInfoList GetSwitchNodeCount GetSwitchNodeLocationList GetSwitchNodeInfoList GetSwitchInfoList GetPartitionCount	NMX_ST_NOT_READY

Control Plane State	Available APIs	Return Codes For Unavailable APIs
	GetPartitionIdList GetPartitionInfoList GetConnCount GetConnInfoList GetConnInfoCombined	
NMX_CONTROL_PLANE_STATE_ERROR NMX_CONTROL_PLANE_STATE_ERROR	GetDomainStateInfo	NMX_ST_NOT_CONFIGURED

- **availableMulticastGroups**

  Provides the available multicast groups in the NVLink domain. This value varies because partitions with non-zero multicast group limits are created and destroyed.

- **configStatusDescription**

  Read with controlPlaneState, it provides a descriptive string for the Control Plane state.

  Here are a few examples:

  - NMX_CONTROL_PLANE_STATE_UNCONFIGURED
    - CONFIG_PENDING_UUID
    - CONFIG_PENDING_TOPOLOGY
    - CONFIG_PENDING_CHASSIS_ID_MAPPING
  - NMX_CONTROL_PLANE_STATE_ERROR
    - CONFIG_ERROR_MISSING_CHASSIS
    - CONFIG_ERROR_DUPLICATE_CHASSIS_SERIAL_NUMBER
    - CONFIG_ERROR_CHASSIS_ID_MAPPING_OUT_OF_RANGE
    - CONFIG_ERROR_INCORRECT_TOPOLOGY_FILE
    - CONFIG_ERROR_CHASSIS_ID_MAPPING_COUNT
    - CONFIG_ERROR_ADDITIONAL_CHASSIS_DETECTED
  - NMX_CONTROL_PLANE_STATE_DEGRADED
    - CONFIG_ERROR_MISWIRED_TRUNK_PORTS

The GRPC API specification contains more information about these states and description strings.

Refer to Miswired Trunk Links for more information about the NMX_CONTROL_PLANE_STATE_DEGRADED state with the CONFIG_ERROR_MISWIRED_TRUNK_PORTS description.

4.1.2 Compute Nodes and the GPU APIs

These APIs gather nuanced information about devices in the expected and discovered topology, which allows users to determine the number of devices that are discovered and online.

4.1.2.1 GetComputeNodeCount

This API queries the number of compute nodes based on several filters.

```
protobuf
```

```
message GetComputeNodeCountRequest {
    ComputeNodeAttr attr
    uint64 chassisId
    ComputeNodeHealth nodeHealth
}
```

Compute Node Attributes (attr)

Here are the filtering options:

- `NMX_COMPUTE_NODE_ATTR_ALL`

  Another way of specifying “No filter”. This option allows all compute nodes to match.

- `NMX_COMPUTE_NODE_ATTR_FREE`

  This filter matches only completely free compute nodes. If there are GPUs attached to a compute node that are part of a partition, the nodes will not be included in the count.

- `NMX_COMPUTE_NODE_ATTR_FULLY_ALLOCATED`

  Only matches nodes that have all of their GPUs assigned to a partition. This does not mean that the nodes belong to the same partition but that every GPU on the compute node is assigned to a partition.

- `NMX_COMPUTE_NODE_ATTR_PARTIALLY_ALLOCATED`

  Only matches nodes that have some GPUs free and some GPUs allocated to a partition. This can help narrow a search when users request a GPU partition. It might be beneficial to have two users share the same compute node.

Chassis ID
This is an optional field to limit searching to one chassis ID, and this ID is meaningful in multi-chassis systems. To allow all chassis to be included in the search, set the value to 0.

Node Health
Optional additional filter to only include compute nodes with specific health states in the result.

- `NMX_COMPUTE_NODE_HEALTH_UNKNOWN`

  Use this value if you do not want to filter on compute node health, and this is the default value.

- `NMX_COMPUTE_NODE_HEALTH_HEALTHY`
  This is used for filtering on compute nodes where the GPUs report that they have full NVLink capability. If a GPU is degraded, the compute node will no longer be healthy, and this will filter those compute nodes out of the result.

- `NMX_COMPUTE_NODE_HEALTH_DEGRADED`
  This filter gets compute nodes in which at least one, but not all of the GPUs, are in an unhealthy state.

- NMX_COMPUTE_NODE_HEALTH_UNHEALTHY
  This health state indicates that all GPUs on the compute node are unhealthy and cannot participate in NVLink traffic.

Here is the response message:

```
protobuf

message GetComputeNodeCountResponse {
    uint32 numNodes
}
```

Num Nodes

Returns the number of nodes that matched the filter that was specified by the parameters above.

4.1.2.2 GetComputeNodeLocationList

For this filter, many of the filters stay the same as the Count version above.

```
protobuf

message GetComputeNodeLocationListRequest {
    ComputeNodeAttr attr
    ComputeNodeHealth nodeHealth
    uint64 chassisId
    uint32 numNodes
}
```

The only new parameter is `numNodes`.

Num Nodes
In this context, `numNodes` limits the number of returned locations to the specified value.

Here is the response:

```
protobuf

message GetComputeNodeLocationListResponse {
    repeated Location locList
}
```

The difference between this API and the Count version is that this API returns the actual locations that were filtered for instead of just a count of the locations.

LocList
This specifies a list of Platform Location Information (for example `chassisId`, `slotNumber`, and `hostId`) to uniquely identify each compute node.

4.1.2.3 GetComputeNodeInfoList

This API provides a more detailed view of each compute node.

The request is structured so that you pass a list of locations and get more information for those locations.

```
protobuf

message GetComputeNodeInfoListRequest {
    repeated Location locList
}
```

LocList
This is a list of locations used as input to gather the information for each compute node.

For the response message, a parallel list of `ComputeNodeInfo` structures is returned:

```
protobuf

message ComputeNodeInfo {
    LocationInfo loc
    uint32 numGpus
    ComputeNodeHealth nodeHealth
    repeated PartitionId partitionIdList
}

message GetComputeNodeInfoListResponse {
    repeated ComputeNodeInfo nodeInfoList
}
```

Loc (LocationInfo)
In addition to having the normal `chassisId`, `slotNumber`, and `hostId` information, the `LocationInfo` structure also contains the chassis serial number and tray index for easy identification.

Num GPUs
This parameter defines the number of GPUs in this compute node. This will be the same as the maximum number of GPUs per compute node in the `DomainProperties` (refer to GetDomainProperties).

Compute Node Health
The realized health value on a compute node (refer to GetComputeNodeCount).

Partition ID List
This is a complete list of all partitions that touch the current compute node. A partition ID of 0 indicates the end of the list. For example, if partition ID 1 was on the first GPU in the compute node, and ID 2 was on the second, and the rest of the GPUs were unallocated, the list will be 1,2, and 0.

This information can locate affected partitions when a tray needs to be taken offline for maintenance. A list with only zeroes corresponds to NMX_COMPUTE_NODE_ATTR_FREE.

Extra Notes on Discovered/Undiscovered Compute Nodes
When a compute node is not discovered, it will have compute node health NMX_COMPUTE_NODE_HEALTH_UNKNOWN (0). The partition ID list can still be populated with an existing location-based partition.

4.1.2.4 GetGpuInfoList

This is the fundamental API that identifies the appropriate GPUs that can be allocated to a new partition. Many of the parameters to the request act as a filter for finding the correct GPUs.

Here is the request structure:

```
protobuf

message GetGpuInfoListRequest {
    GpuAttr attr
    uint32 numGpus
    Location loc
    PartitionId partitionId
}
```

GpuAttr (attr)
This attribute works like a selector on the location and partition ID parameters.

- NMX_GPU_ATTR_ALL

  Works as a “No Filter”, which allows all GPUs regardless of their location or partition ID.

- NMX_GPU_ATTR_LOCATION

  Only shows GPUs that match a specified location in the loc parameter.

- NMX_GPU_ATTR_PARTITION_ID

  Only shows GPUs that are assigned to a partition. When you specify this filter with partition ID 0, GPUs are not assigned to a partition.

Num GPUs
A limit on the number of GPUs returned.

Loc (Location)
An indicator that the only interesting GPUs should be from a compute node and must be used with the NMX_GPU_ATTR_LOCATION attribute.

PartitionId
An indicator that the only interesting GPUs should be from a partition ID and must be used with the NMX_GPU_ATTR_PARTITION_ID attribute.

Here is the response:

```
protobuf

message GpuInfo {
    LocationInfo loc
    uint32 gpuId
    uint64 gpuUid
    GpuHealth gpuHealth
    PartitionId partitionId
}

message GetGpuInfoListResponse {
    repeated GpuInfo gpuInfoList
}
```

Location and GPU ID
This information fully qualifies a GPU's location. If you need to add a GPU to a location-based partition, this information allows you to construct a `GpuLocation` structure.

GPU UID
This is a unique identifier for a GPU and can be used as an alternate way to specify partition creation.

GPU Health
This parameter can be used to ensure that only healthy GPUs are added to the partition you create and can also be used as an indicator for maintenance activities:

- `NMX_GPU_HEALTH_HEALTHY`

  The GPU is visible on the fabric and has all of its links in a working state.

- `NMX_GPU_HEALTH_DEGRADED`

  The GPU is seen and some, but not all, of its links are down.

- `NMX_GPU_HEALTH_NO_NVLINK`

  The GPU is unable to participate in NVLink partitioning.

- `NMX_GPU_HEALTH_DEGRADED_BW`

  The GPU can participate in NVLink partitioning but with degraded bandwidth.

Partition ID
Indicates the partition (if any) to which the GPU belongs. If the GPU is available and free, the value is set to 0.

Extra Notes on Discovered/Undiscovered GPUs
When a GPU is not yet visible on the fabric (undiscovered), the health state will be `NMX_GPU_HEALTH_UNKNOWN (0)`, and the `gpuUid` will be set to 0. The partition ID can still be set when there is an existing user-based partition.

4.1.3 Switch Node and Switch APIs

The switch (node) APIs provide insight into the state of the switch hardware. The information includes health states and the partitions that the switch node helps to maintain. The relevant APIs for partitioning are the switch node and switch information lists.

4.1.3.1 GetSwitchNodeInfoList

This API contains detailed information about each switch node and to which partitions it participates when routing traffic.

```
protobuf

message GetSwitchNodeInfoListRequest {
    repeated Location locList
}
```

Location List

If you do not provide this list, all switch node information will appear in the response. If a more specific set of locations is required, the locations can be set here and found using `GetSwitchNodeCount`/`GetSwitchNodeLocationList`, which is not in the scope of this guide (refer to the *NMX-Controller gRPC API Guide*, which is a part of the Switch Tray firmware package, for more information.

The response includes a list of `SwitchNodeInfo` structures:

```
protobuf

message SwitchNodeInfo {
    LocationInfo loc
    uint32 numSwitches
    SwitchNodeHealth nodeHealth
    repeated PartitionId partitionIdList
}

message GetSwitchNodeInfoListResponse {
    repeated SwitchNodeInfo nodeInfoList
}
```

Switch Node Health

This parameter indicates the ability of a switch to route traffic and support the partitions in its partition ID list.

- `NMX_SWITCH_NODE_HEALTH_HEALTHY`

  The switch is discovered on the fabric and has all its NVLinks up in a working state.

- `NMX_SWITCH_NODE_HEALTH_MISSING_NVLINK`

  At least one NVLink is down in the switch node

- NMX_SWITCH_NODE_HEALTH_UNHEALTHY

  No NVLinks are healthy in the switch node.

Partition ID List
Similar to the ComputeNodeInfo's partition ID list, this array is finalized by the size or a terminating partition ID of 0. In a single-chassis environment, all partitions are probably affected by all switch nodes. In a multi-chassis environment, however, a partition might be limited to the switches on one chassis.

Extra Notes on Discovered/Undiscovered Switch Nodes
When a switch node is undiscovered, it will have a node health of NMX_SWITCH_NODE_HEALTH_UNKNOWN. The partition list might have an entry in this scenario because all switch nodes on the same chassis, as an allocated GPU, facilitate NVLink traffic.

4.3.2 GetSwitchInfoList

For each switch node, there might be multiple switch ASICs. This API provides information about each ASIC on each switch node. A relation between partition and switch ASIC in the switch information list does not exist, but the information can narrow down which ports are causing issues for which partitions.

Here is the request:

```
protobuf

message GetSwitchInfoListRequest {
    Location loc
    uint32 numSwitches
}
```

If no parameters are provided, the API will return information about the known switches:

```
protobuf
message SwitchInfo {
    Location loc
    uint32 switchId
    uint64 switchUid
    uint32 numPorts
    SwitchHealth health
}

message GetSwitchInfoListResponse {
    repeated SwitchInfo switchInfoList
}
```

Location and Switch ID
Specifies the device on the switch node that is being referenced.

Switch UID
The unique identifier for a switch ASIC. To narrow the issues, this UID can be correlated with topology information.

Switch Health
A high-level overview of each switch's ability to route traffic. This parameter is more useful when diagnosing issues with partition communication and should be used as a deeper look into a switch node with a known partition ID list.

- `NMX_SWITCH_HEALTH_HEALTHY`

  The switch is discovered on the fabric and has all of its NVLinks up in a working state.

- `NMX_SWITCH_HEALTH_MISSING_NVLINK`

  Some, but not all, NVLinks are down for the ASIC.

- `NMX_SWITCH_HEALTH_UNHEALTHY`

  No NVLinks are healthy in the switch.

Extra Notes on Discovered/Undiscovered Switches
An undiscovered switch has its health set to `NMX_SWITCH_HEALTH_UNKNOWN (0)` and its switchUid set to 0.

4.1.4 Partition APIs

This section provides information about the APIs that you can use to view partition states and how to manage these partitions.

4.1.4.1 GetPartitionCount

This API queries for partition counts based on a filter.

```
protobuf
message GetPartitionCountRequest {
    PartitionInfoAttr infoAttr
    uint32 numGpus
    uint32 numNodes
    PartitionHealth health
}
```

Partition Information Attributes
This parameter selects the parameters that can be used as a filter.

- `NMX_PARTITION_INFO_ATTR_ALL`

  This acts like a "No Filter" and allows partitions of any criteria to pass through to the response.

- `NMX_PARTITION_INFO_ATTR_NUM_GPUS`

  Only shows information for partitions that contain a certain number of GPUs.

- `NMX_PARTITION_INFO_ATTR_NUM_COMPUTE_NODES`

  Only shows information for partitions that touch a specified number of compute nodes. This attribute does not allow you to filter on chassis where those compute nodes are located.

Partition Health
This is used as a supplementary filter to the above attributes. If specified, it will only allow partitions that are in the specified health state to be counted.

- `NMX_PARTITION_HEALTH_UNKNOWN`
  A partition that does not have any discovered GPUs will start with this option as the initial value. This is used as a "No filter" and partitions of any health state will be passed as part of the result. This is the default value.
- `NMX_PARTITION_HEALTH_HEALTHY`

  A partition where all the available GPUs can communicate at full bandwidth. To maintain bandwidth, in this state, no GPUs are intentionally excluded from the fabric. This filter only counts partitions in this state.
- `NMX_PARTITION_HEALTH_DEGRADED_BANDWIDTH`
  A partition that operates with a subset of required trunk links (applies only to Inter-chassis partitions). This means that the cross-chassis traffic will have a lower bandwidth than optimal. This filter only counts partitions in this state.
- `NMX_PARTITION_HEALTH_DEGRADED`
  A partition is operating with intentionally excluded GPUs. The GPUs are excluded from the NVLink fabric because of trunk link failures and to maintain full bandwidth. This filter only counts partitions in this state.
- `NMX_PARTITION_HEALTH_UNHEALTHY`
  A partition where the fabric health is detected as unhealthy, so traffic might not pass. This state is considered non-operational.

  The partition can go into this state in scenarios including trunk miswirings, unhandled trunk failures (specifically in user action required resiliency mode), and other internal failures. This filter only counts partitions in this state.

In the response, only a `numPartitions` parameter is considered:

```
protobuf

message GetPartitionCountResponse {
    uint32 numPartitions
}
```

Num Partitions
The returned number of partitions that match the filter that was provided in the request.

4.1.4.2 GetPartitionIdList

This API is similar to the Count API above except in two key areas:

```
protobuf
message GetPartitionIdListRequest {
    PartitionInfoAttr infoAttr
    uint32 numGpus
    uint32 numNodes
    uint32 numPartitions
    PartitionHealth health
}
```

Num Partitions
This is the only parameter on the request that is different from the `GetPartitionCount` API in GetPartitionCount. It limits the number of partition IDs to a maximum value in the response.

The response returns the realized list of partition IDs based on the filter instead of their count:

```
protobuf
message Partition {
    PartitionId partitionId
    uint32 numGpus
}

message GetPartitionIdListResponse {
    repeated Partition partitionList
}
```

Partition ID
The partition ID is defined as a uint32 because of limitations in protobuf. There is no uint16 representation in the language, the limited pool of partition IDs is 1 to 0x7FFD, and the ID 0x7FFE is reserved only for the Default Partition.

Num GPUs
This is the number of GPUs in the partition as specified by the ID.

4.1.4.3 GetPartitionInfoList

This API provides detailed and more useful information on each partition, and the request provides a list that will be filled in with specified partition IDs:

```
protobuf

message GetPartitionInfoListRequest {
    repeated PartitionId partitionIdList
```

```
}
```

Partition ID List
If this list is empty, the full unfiltered list of all partitions is returned in the response.

The response is defined by the following structures:

```
protobuf

message PartitionAttr {
    ResiliencyMode resiliencyMode
    uint32 multicastGroupsLimit
}

message PartitionInfo {
    PartitionId partitionId
    string name
    uint32 numGpus
    repeated GpuLocation gpuLocationList
    repeated uint64 gpuUidList
    PartitionHealth health
    PartitionType partitionType
    uint32 numAllocatedMulticastGroups
    PartitionAttr attr
}

message GetPartitionInfoListResponse {
    repeated PartitionInfo partitionInfoList
}
```

Partition ID
Refer to GetPartitionIdList for more information about the partition ID

Partition Name
The partition name is a unique string that can be used as an alternate way to associate data to a partition.

Num GPUs
This is the number of GPUs in the partition, and it is the size of the `gpuUidList` (UID-based) or the `gpuLocationList` (location-based) depending on whether the partition is UID-based or location-based.

GPU Location List
The locations that are associated with a partition. This parameter is populated for location- and GPU UID-based partitions and is treated as a parallel array with the GPU UID List parameter. In a location-based partition, this array will remain constant, but the other might change.

GPU UID List
This is the unique identifier for the GPU in the location slot as specified by the parallel location list array. In a GPU UID-based partition, this array remains relatively constant (refer to GPU UID-Based Partitions), while the other might change.

Partition Health
This is the realized value of the partition health (refer to GetPartitionCount for more information).

Partition Type
This is an important indicator for a partition and its behavior, and here are the partition types:

- `NMX_PARTITION_TYPE_UNDEFINED`

  Refer to Zero-GPU Partitions for more information about this partition type.

- `NMX_PARTITION_TYPE_LOCATION_BASED`

  Refer to Location-Based Partition for more information about this partition type.

- `NMX_PARTITION_TYPE_GPUUID_BASED`

  Refer to GPU UID-Based Partitions, for more information about this partition type.

Num Allocated Multicast Groups
This is the real-time indicator of how many multicast groups are active for a partition.

Multicast Groups Limit
There is a hardware limit of 1024 total multicast groups that can be on the fabric at a time. This parameter specifies how many multicast groups were reserved for this partition out of the pool of 1024 for all partitions (refer to Multi-Cast Support for more information).

Resiliency Mode
Depending on the resiliency mode for the partition, the partition can enter one of the following health states:

- Full Bandwidth Mode:
  - `NMX_PARTITION_HEALTH_HEALTHY`: When the partition is marked as healthy, it is expected to be in full bandwidth and in a full compute capacity state.
    This is the optimal state.
  - `NMX_PARTITION_HEALTH_DEGRADED`: In this state, some of the GPUs might be marked as "parked" and their GPU health might be `NO_NVLINK`.
    In this state, the rest of the GPUs will be able to communicate with full bandwidth, and this is an operational state.
  - `NMX_PARTITION_HEALTH_UNHEALTHY`: There might be various reasons that cause a partition to enter an unhealthy state such as the loss of a switch or other internal failures.

    This state is not an operational state.

- Adaptive Bandwidth Mode:
  - `NMX_PARTITION_HEALTH_HEALTHY`: When the partition is marked as healthy, it is expected to be in full bandwidth and in a full compute capacity state.
    This is the optimal state.
  - `NMX_PARTITION_HEALTH_DEGRADED_BANDWIDTH`: In this state, some of the trunk links might be missing.
    However, all of the operational GPUs will be able to communicate with each other in a degraded bandwidth capacity. This is an operational state.
  - `NMX_PARTITION_HEALTH_UNHEALTHY`: There might be various reasons that cause a partition to go into an unhealthy state such as the loss of a switch or other internal failures.
    This is not an operational state.
- User Action Required Mode:
  - `NMX_PARTITION_HEALTH_HEALTHY`: When the partition is marked as healthy, it is expected to be in full-bandwidth and full compute capacity state.
    This state is considered to be optimal.
- `NMX_PARTITION_HEALTH_UNHEALTHY`: Here are some of the reasons why a partition might enter an unhealthy (and non-operational) state:
  - There might be a loss of a switch or other internal failures.
  - In this resiliency mode, the unhealthy state might be a result of an unhandled trunk link failure.
    When a trunk link fails, if the control plane cannot find spare links to restore full bandwidth, it will immediately go to this state. To reduce the number of required links, recovery actions include fixing or freeing trunk links or removing GPUs from this partition

For a partition that is in the `DEGRADED`, `DEGRADED_BANDWIDTH`, or `UNHEALTHY` state due to a trunk failure, after the required trunk links are available, a manual reroute operation can be used to restore the partition to a healthy state (refer to Modify the Partition (AddGpusToPartition) for more information.

Refer to Partition Fault Handling for more information about this parameter.

4.1.4.4 CreatePartition

Before you can use inter-GPU traffic over an NVLink, you need to create a partition .

Here is the request:

```
protobuf
message GpuLocation {
    Location loc
    uint32 gpuId
}

message GpuResourceId {
    oneof resourceId {
        GpuLocation gpuLocation
        uint64 gpuUid
    }
}
```

```
message CreatePartitionRequest {
    string name
    repeated GpuResourceId gpuResourceId
    PartitionAttr attr
    PartitionId partitionId
}
```

Partition Name
Name your partition something unique so that you can correlate it to another item in your database.

Note Unique partition names are currently enforced.

In general, the `Default Partition` name is reserved.

GPU Resource ID
This parameter is used to identify a GPU by its location, or by its GPU UID, and only one type of specification is allowed at a time. The list cannot contain mixed locations and UIDs because homogeneity determines the partition type. The behavior of a partition is determined by its type (refer to Switch Node and Switch API for more information).

You can create a partition with 0 GPUs by not specifying any partitions in the lists, and the partition type will be `UNDEFINED` until the partition is modified. The partition takes on the type of the modification. When removing GPUs, a partition type can transition back to `UNDEFINED`.

Partition Attributes (attr)
The attributes of a partition are specified once at creation. The attributes are the same as the attributes in GetPartitionInfoList.

Partition ID
Users can also specify the partition ID, but this partition ID must be unique across the domain.

Return Codes
Refer to the GRPC APIs for more information.

4.1.4.5 DeletePartition

Deleting a partition is the release of associated partition resources such as GPUs, a unique name (if specified), multicast reservations, and the partition ID.

```
protobuf

message DeletePartitionRequest {
    PartitionId partitionId

    string name
}

message DeletePartitionResponse {
    PartitionId partitionId
}
```

Partition ID
To ensure that the correct partition ID is deleted, the ID is specified in the request and response.

Name
An alternate way to specify the partition that will be deleted.

Return Codes
Refer to the *NMX-Controller gRPC API Guide*, which is a part of the Switch Tray firmware package, for more information.

4.1.4.6 Modifying the Partition (AddGpusToPartition/RemoveGpusFromPartition)

You can create a partition or change the partition to add or remove GPU resources.

Here is an example of how to update a partition:

```
protobuf

message UpdatePartitionRequest {
    PartitionId partitionId
    repeated GpuLocation locationList
    repeated uint64 gpuUid

    string name

    bool reroute
}

message UpdatePartitionResponse {
    PartitionId partitionId
```

```
}
```

Partition ID
This ID specifies the partition that needs to be modified.

Location List
This list is only specified when the partition ID refers to a location-based partition and describes the locations that need to be added or removed from the partition.

GPU UID List
This list is only specified when the partition ID refers to a GPU UID-based partition and describes which GPUs to add or remove from the partition.

Name
An alternate way to specify the partition that needs to be updated.

Rerouting
This flag is set to true by default and is only set to false in certain maintenance scenarios. `True` means that the trunk links between chassis for multi-chassis partitions will be allocated again to provide full bandwidth for the GPUs or free up excess trunk links from this partition. During a reroute, existing cross-chassis traffic will be affected (refer to Maintenance for more information). The Rerouting flag is used only with Inter-Chassis partitions.

A special invocation of this request involves specifying a partition ID but leaving the `locationList` and `gpuUid` empty. This instructs the Control Plane to perform a reroute and evaluate the assigned trunk links for this partition again without modifying the GPUs in the partition. This is useful in maintenance scenarios such as restoring the partitions to an optimal state.

4.2 NVOS CLIs

This section provides information about partition NVOS CLIs.

Note NVOS CLI commands are asynchronous, so after a command is initiated, the operation might continue running in the background. Ensure you verify the status or completion of the process before proceeding with any dependent tasks.

4.2.1 Creating SDN Partitions

Partitions can be created to define the allocation of GPUs with attributes such as the following:

```
admin@nvos:~$ nv action create sdn partition <partition-id> name <name>
resiliency-mode <resiliency-mode> mcast- limit <mcast-limit> [uuid <uuid>] [location
<location-id>]
```

```
$ nv action create sdn partition 1 name example_partition1 resiliency-mode
adaptive_bandwidth mcast-limit 0
Action executing ...
Creating a partition: 1
Action executing ...
Partition 1 is successfully created
Action succeeded
```

A known limitation of the partition creation CLI is that only a single resource such as the uuid or the location-id can be specified. If a partition needs to be created with multiple resources, you must add GPUs to the partition.

4.2.2 Adding GPUs To a Partition

You can add GPUs to a partition in one of the following ways:

Location-based

```
admin@nvos:~$ nv action update sdn partition <partition-id> location <location-id>

[no-reroute]
```

UUID-based

```
admin@nvos:~$ nv action update sdn partition <partition-id> uuid <uuid> [no-reroute]
```

Here is the output:

```
$ nv action update sdn partition 1  uuid  12655913173760303306
Action executing ...
Updating uuid 12655913173760303306 in partition 1
Action executing ...
Partition 1 uuid 12655913173760303306 has been successfully updated
Action succeeded

$ nv show sdn partition 1
                 operational
---------------  ------------------
name             example_partition1
num-gpus         1
health           healthy
resiliency-mode  adaptive_bandwidth
mcast-limit      0
```

```
partition-type   gpuuid_based



locations
============
    GPU Location  UUID
    ------------  --------------------
    1.1.1.1       12655913173760303306
```

Note The GPU UUID used for partition manipulations must be specified in decimal format. If you currently have a hexadecimal GPU UUID, please convert it to decimal before proceeding.

4.2.3 Removing GPUs From a Partition

You can remove GPUs from a partition in one of the following ways:

Location-based

```
 admin@nvos:~$ nv action restore sdn partition <partition-id> location <location-id>
[no-reroute]
```

UUID-based

```
admin@nvos:~$ nv action restore sdn partition <partition-id> uuid <uuid> [no-reroute]
```

Here is the output:

```
$ nv action restore sdn partition 32766 uuid 12655913173760303306
Action executing ...
Restoring uuid 12655913173760303306 in partition 32766
Action executing ...
Partition 32766 uuid 12655913173760303306 has been successfully restored
Action succeeded
```

4.2.4 Viewing the SDN Partition

To display all partitions with details such as resiliency mode, multicast limit, and type, run the following command:

```
admin@nvos:~$ nv show sdn partition
```

Here is the output:

```
$ nv show sdn partition

ID     Name               Num of GPUs  Health   Resiliency mode     Multicast groups
limit  Partition type  Summary

-----  -----------------  -----------  -------  ------------------
----------------------  --------------  -------
```

```
32766  Default Partition  64           healthy  adaptive_bandwidth  1024
gpuuid_based
```

To display detailed information about a specific partition, including associated GPUs and health status, run the following command:

```
admin@nvos:~$ nv show sdn partition <partition-id>
```

Here is the output:

```
$ nv show sdn partition 32766
                 operational
---------------  ------------------
name             Default Partition
num-gpus         64
health           healthy
resiliency-mode  adaptive_bandwidth
mcast-limit      1024
partition-type   gpuuid_based



locations
=============
    GPU Location  UUID
    ------------  --------------------
    1.1.1.1       12655913173760303306
    1.1.1.2       13251666084076697626
    1.1.1.3       1764344441350953703
    1.1.1.4       5485166907021013365
    1.2.1.1       1450422857593169538
    1.2.1.2       6627118739306428495
    1.2.1.3       7806904066629993257
    1.2.1.4       6716622511424029579
    1.3.1.1       10529188649349609227
    1.3.1.2       2262711745376834792
```

4.2.5 Deleting an SDN Partition

To delete an existing partition.

```
admin@nvos:~$ nv action delete sdn partition <partition-id>
```

Here is the output:

```
$ nv action delete sdn partition 1

Action executing ...

Deleting a partition: 1
```

```
Action executing ...

Partition 1 is successfully deleted

Action succeeded
```

This section will be updated in a future release with the NVOS CLIs that are relevant to partition management.

5. Control Plane Software High Availability

This chapter provides information about high availability.

5.1 Persistence and Recovery Overview

Here are some of the reasons why the NMX-Controller (NMX-C) and its FM service can go down:

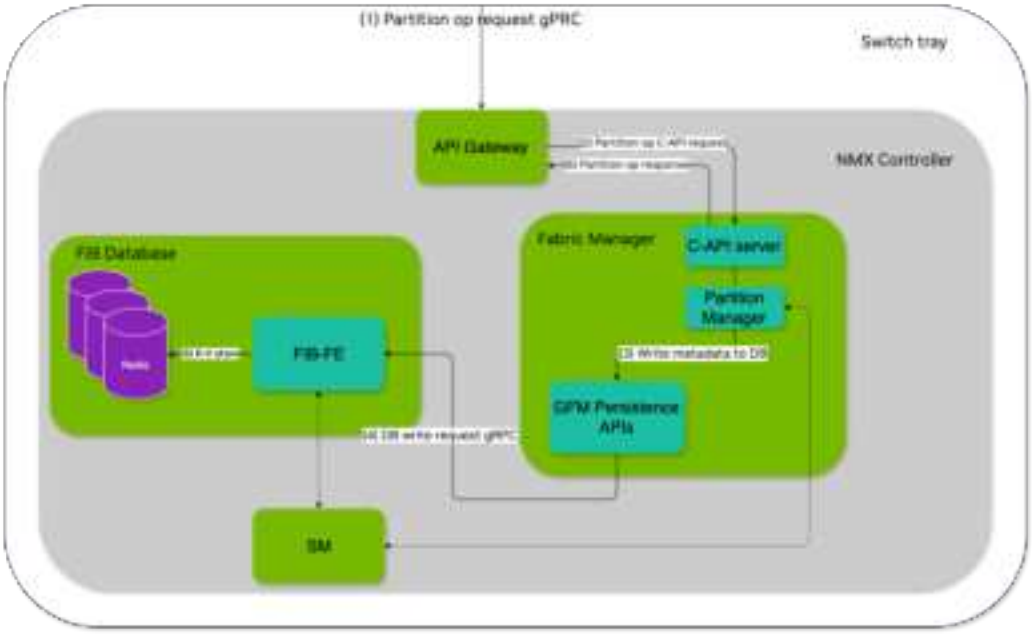
- A planned NMX-C restart.
- An NMX-C hosting switch tray reboot.
- A FM crash/restart.

To provide continuity of service, if FM crashes, it is automatically restarted and recovers its previous settings. This is achieved by saving the metadata that was generated during its runtime and restoring the metadata by using the database infrastructure provided by NMX-C.

The runtime metadata includes all aspects in FM including the partition metadata, which is the information that describes a partition's configurations and states (partition ID, list of GPUs, health state, multicast teams, and so on).

Figure 5 shows an example of partition metadata.

Figure 5. Partition Metadata



5.2 Partition Metadata Availability

In the TTM phase, metadata is stored locally on the switch tray where FM was running and is used by FM when it is restarted on the same switch tray.

Note: The NMX-C/FM that is restarted on another switch tray cannot reconstruct its configurations using the previously saved metadata.

5.3 Restarting Fabric Manager for Established Partitions

Default, or user partitions that existed before FM went down are recreated based on the saved partition metadata after FM restarts. Obsolete configurations or states will be refreshed.

Here is some additional information:

- NMX-C restarts, FM crashes, and restarts.
  - The existing CUDA app unicast and multicast NVLink traffic are not disrupted during the NMX-C/FM restart.

    - Multicast teams are created by tenants that use GPUs.
    - After FM restarts, the established multicast teams are restored in FM.
- An NMX-C hosting switch tray reboots.

  If the switch tray reboots, it is disruptive to existing unicast and multicast NVLink traffic through the impacted switch trays.
- The existing CUDA application aborts.
  Unicast NVLink traffic can be restarted after NMX-C/FM restarts, and the GPU is reset.
- Multicast teams cannot continue to function.
  After NMX-C/FM is restarted, the broken multicast teams are eventually destroyed. You can restart the tenants' applications and recreate the multicast teams.

5.4 Restarting Fabric Manager During Ongoing Partition Operations

The NMX-C/FM can go down when partition operations, such as partition creation or deletion and GPU addition or removal.

Partition operations are handled as transactions, and partitions are always restored to a consistent state after a FM restart., Depending on the timing and operation type, unfinished operations are continued or rolled back. Administrators can query the latest settings after NMX-C/FM restart.

Here is some information about creating partitions or handling GPU addition requests:

- If FM goes down **before** it starts the transaction, the request will time out, but no changes are made.
  After FM restarts, the administrator can retry the failed requests.
- If FM goes down **during** the transaction, the request will time out.
  After FM restarts, FM rolls back the partitions to the state before the transaction, and the administrator can retry the failed requests.
- If FM goes down **after** the transaction finishes, but before the request is acknowledged, there will be a request time out.
  After FM restarts and the administrator queries the current list of partitions, the updated partitions are displayed. No additional action is needed.

Here is some additional information about deleting partitions or handling GPU removal requests:

- If FM goes down **before** it starts the transaction, the request will time out, but no changes are made.
  After FM restarts, the administrator can retry the failed requests.
- If FM goes down **during** the transaction, the request will time out.
  After FM restarts, FM replays the previous failed transaction. The administrator queries the current list of partitions, and the updated partitions are displayed. No additional action is needed.

- If FM goes down **after** the transaction finishes, but before the request is acknowledged, the request will time out.
  After FM restarts, the administrator queries the current list of partitions, and the updated partitions are shown. No additional action is needed.

5.5 GFM Restart and a Partition Wipeout

By default, FM recovers partition configurations after a restart. When the FM static configuration option `FABRIC_MODE_RESTART` is set to 1, it follows the default behavior. If users expect the partition and other runtime configurations to be wiped out after restart, they should set `FABRIC_MODE_RESTART` to 0 in the FM static configuration.

```
#     Description: Restart Fabric Manager after exit

#                This can always be set to 1. If persisted runtime data is to be
deleted, then set it to 0

#     Possible Values:

#       0 - Start Fabric Manager and erase persisted runtime data

#       1 - (Recommended) Start Fabric Manager and try to restore system state on
restart and dynamically handles first time boot

FABRIC_MODE_RESTART=1
```

To wipe out runtime configurations:

1. Set `FABRIC_MODE_RESTART=0` in FM static configuration.
2. Stop `NMX-C`.
3. Start `NMX-C`.
4. (Optional) If users expect a one-off operation, set `FABRIC_MODE_RESTART=1`.

5.6 Default Partition and Fabric Mode Restart

Table 5 captures the interaction between the `MNNVL_ENABLE_DEFAULT_PARTITION` and `FABRIC_MODE_RESTART` configurations and documents the GFM behavior in various restart scenarios.

Table 5. Interaction Between Two Configurations

GFM Init Time Config	Admin Runtime Action	Impact on the Default Partition After a GFM Restart
MNNVL_ENABLE_DEFAULT_PARTITION = 1 FABRIC_MODE_RESTART = 1	Delete Default Partition	The Default partition is not created.
	Create User Partition	The Default Partition is created but will not operate in the active phase.

GFM Init Time Config	Admin Runtime Action	Impact on the Default Partition After a GFM Restart
	Set MNNVL_ENABLE_DEFAULT_PARTITION = 0	The Default Partition is created and continues to operate in the active phase.
MNNVL_ENABLE_DEFAULT_PARTITION = 0 FABRIC_MODE_RESTART = 1	Create User Partition and set MNNVL_ENABLE_DEFAULT_PARTITION = 1	The Default Partition is not created.
	Set MNNVL_ENABLE_DEFAULT_PARTITION = 1	The Default Partition is created.
MNNVL_ENABLE_DEFAULT_PARTITION = 1 FABRIC_MODE_RESTART = 0	Set MNNVL_ENABLE_DEFAULT_PARTITION = 0	The Default Partition is not created.
	No action	The Default Partition is created.
MNNVL_ENABLE_DEFAULT_PARTITION = 0 FABRIC_MODE_RESTART = 0	Create User Partition and set MNNVL_ENABLE_DEFAULT_PARTITION = 1	The Default Partition is created and the user partition is deleted.

6. Partition Fault Handling

Hardware faults can happen at any time during the lifetime of a partition. This chapter provides information about the types of hardware faults and their impact on a partition.

6.1 GPU

Here is how partition fault handling is managed in a GPU:

1. A GPU failure causes a workload that is running in a partition to fail.
2. The Control Plane detects the failure and marks the GPU health as `NO_NVLINK`. Use the GetGpuInfoList GRPC API to determine the health of the GPU.
3. The state of the GPU on the compute node is available through the tools described in GPU Fabric State.

4. After determining that the failed GPU is in `NO_NVLINK` state, the administrator can use the partition to run workloads on the remaining NVLink capable GPUs

6.2 Access Link

Here is how partition fault handling is managed in an access link:

1. An access link failure causes the GPU in the partition to lose NVLink connectivity This causes the workload in the partition to run into errors.
2. The Control Plane detects the failure through GPU link down events and that causes the GPU health to be marked `NO_NVLINK`.
3. After determining that the failed GPU is in `NO_NVLINK` state, the administrator can use the partition to run workloads on the healthy GPUs.

Note When a GPU is reset, the handling of the health state update is the same as when it experiences an access link failure. However, a GPU reset typically lasts for a few seconds after which the GPU is rediscovered into the NVLink fabric. In both cases, the partition health remains unchanged.

6.3 Trunk Link

When the trunk link that is assigned to a partition fails, the partition behavior is determined by the resiliency mode in which it operates. Trunk link failures will cause interruptions in cross-chassis workload in the partition.

6.3.1 Faulty Trunk Link

Depending on the resiliency mode that you selected for the partition, here is how the partition will react when a trunk link failure occurs:

- **Full Bandwidth Mode**
  - If spare trunk links are found, a replacement trunk link will be assigned to the partition.
    - The partition health will be marked as healthy.
    - The partition will continue to operate with all the present GPUs and at full bandwidth.
  - If replacement trunk links are not found, at least one GPU will be excluded from the NVLink fabric to maintain full bandwidth for the remaining GPUs.
    - The partition health is set to degraded to show that at least one GPU has been excluded from the fabric.
    - The number of excluded GPUs is proportional to the number of trunk links that are lost.
    - The excluded GPUs will be marked as NVLink disabled.
    - To minimize the total number of excluded GPUs, the excluded GPUs will be selected from the chassis with a lower number of GPUs.

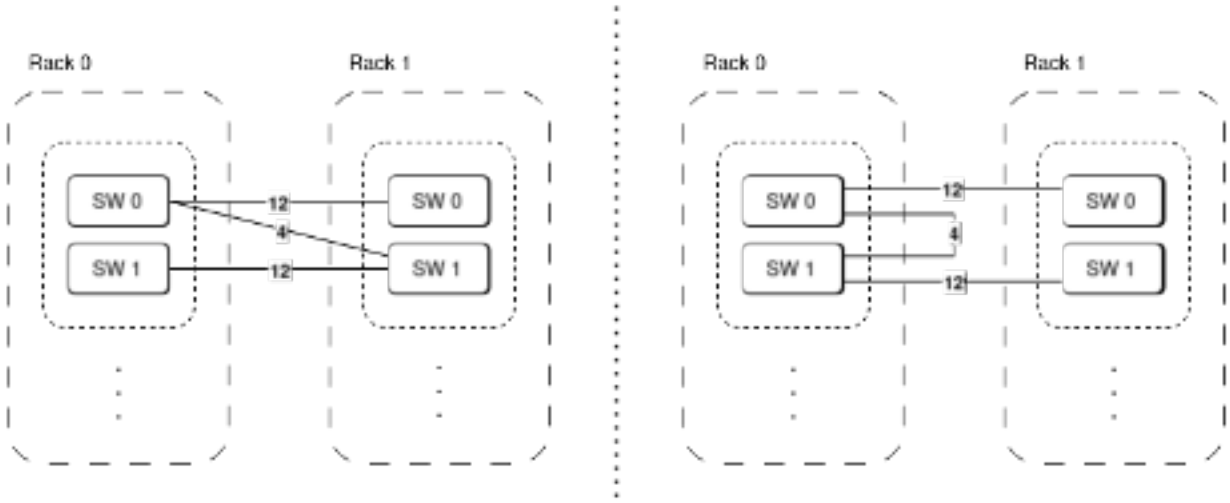
        - After enough trunk links are available, a manual reroute operation can be used to restore the partition to a healthy state (refer to Modify the Partition (AddGpusToPartition/RemoveGpusFromPartition) for more information). .
- **Adaptive Bandwidth Mode**
    - If spare trunk links **are found**, a replacement trunk link will be assigned to the partition.
        - The partition health will be marked as healthy.
        - The partition will continue to operate with full bandwidth and with all the present GPUs.
    - If replacement trunk links **are not found**, the GPUs in the partition will operate with a degraded bandwidth.
        - The partition health will be set to `DEGRADED_BANDWIDTH` to show that the partition is operating with all the available GPUs but with limited bandwidth.
        - The GPUs in the partition will also be marked as having `DEGRADED_BANDWIDTH`.
        - After enough trunk links are available, a manual reroute operation can be used to restore the partition to a healthy state (refer to Modify the Partition (AddGpusToPartition) for more information.
- **User Action Required Mode**
    - If spare trunk links are found, a replacement trunk link will be assigned to the partition.
        - The partition health will be marked as healthy.
        - The partition will continue to operate with all the present GPUs and at full bandwidth.
    - If replacement trunk links are not found, the partition will go into a non-operational state which requires user action to recover.
        - The partition health will be marked as unhealthy.
        - Cross-rack traffic will likely fail in this scenario.
        - All the GPUs in the partition will be notified that the fabric is unhealthy.
        - After enough trunk links are available, a manual reroute operation can be used to restore the partition to a healthy state (refer to Modify the Partition (AddGpusToPartition) for more information.

6.3.2 Miswired Trunk Links

Here is the pattern for trunk link wiring in topologies with multiple chassis:

- The slot number of the switches connected using trunk links shall match.
  This means that the second switch tray of the first chassis can only be connected to the second switch tray of the second chassis.
- Cage numbers 1-9 on one side of the trunk cable can only be connected to cage numbers 1-9 on the other side.
- Cage numbers 10-18 on one side of the trunk cable can only be connected to cage numbers 10-18 on the other side.
- A switch tray can not be connected to itself.

Figure 6. Trunk Link Miswiring



Examples of trunk miswiring are shown in Figure 6.

From a control plane perspective, if any of the guidelines are not followed, the control plane state (refer to GetDomainStateInfo) will be marked `NMX_CONTROL_PLANE_STATE_DEGRADED` with the `CONFIG_ERROR_MISWIRED_TRUNK_PORTS` description.

In this state, the control plane can be considered mostly non-operational. Partitioning API calls such as `CreatePartition`, `AddGpusToPartition`, and `RemoveGpusToPartition` will be rejected by FM. Additionally, all existing partitions will be marked `NMX_PARTITION_HEALTH_UNHEALTHY` (refer to 4.1.4.1 GetPartitionCount).

FM will continue to monitor any changes in trunk link connections until the miswiring is fixed before enabling the APIs evaluating the health of the partitions again. After the miswiring is fixed, all the intra-chassis partitions are expected to become healthy, and each partition will be evaluated based on the latest state of the trunk links. A manual reroute (refer to 4.1.4.6 Modifying the Partition (AddGpusToPartition/RemoveGpusFromPartition)) operation might be required to return the inter-chassis partitions to a healthy state after the trunk link is fixed.

If FM detects the miswiring during the initialization, it will continue to monitor changes in trunk link connections until the miswiring is fixed. In such a state, the administrator must fix the trunk ports before the control plane can continue its operation.

6.4 Compute Tray

Here is how partition fault handling is managed in a compute tray:

1. When one of the compute trays assigned to a partition fails, it causes errors on the partition workload.
2. The Control Plane detects the failure that causes the health of all GPUs in the tray to be marked `NO_NVLINK`.

3. The health of the compute tray is marked `UNHEALTHY` and the health of the partition remains unchanged.
4. After determining that the failed GPUs are in `NO_NVLINK` state, the administrator can use the partition to run workloads on the GPUs from the remaining compute trays

Note When a compute tray is rebooted, the handling of the health state update is the same as when the compute tray fails. However, a reboot typically lasts for a few seconds after which the compute tray is rediscovered in the NVLink fabric.

6.5 Switch Tray/Switch

Here are the differences between single-chassis and inter-/intra-chassis platforms:

- In single-chassis platforms that consist of only Intra-Chassis partitions, a switch tray/switch failure causes all GPUs in all partitions to lose NVLink connectivity.
- In multi-chassis platforms that consist of Intra- and Inter-Chassis partitions, a switch tray/switch failure affects all GPUs in all partitions in the chassis on which the failure occurs.
  If trunk ports on the failed switch are part of a partition, partitions on other chassis are affected.

Note: A partition that does not use the affected switch will not get affected.

Here is how partition fault handling is managed in a switch tray or switch:

1. The failure causes the workload in the partition to experience errors.
2. The Control Plane detects the failure through GPU link down events and that causes the health of all GPUs connected to the switch to be marked `NO_NVLINK`.
3. The partition health is set to `UNHEALTHY` to show that none of the GPUs are NVLink capable.
4. Administrators cannot use the partition to run workloads that use NVLink.
   They can use standalone GPUs in the partition to run workloads.

7. Maintenance

Hardware faults can impact the partition's ability to run workloads. Performing maintenance on faulty hardware helps users identify the potential root cause, take corrective action, and in certain cases, isolate the failure. After the failure is addressed, or a replacement hardware is found, users can resume the normal workflow.

This chapter discusses maintenance workflows for several use cases and provides information about their impact on partition management..

7.1 Maintenance Flow for a Compute Tray

After a faulty compute tray is identified, it needs to be isolated from the partition by using an Out for Repair (OFR) partition. An OFR partition's life cycle can be managed in one of the following ways based on your maintenance flow:

- A `zero-gpu` OFR partition.
  1. A `zero-gpu` partition is created using `CreatePartition` and is unallocated without any resources.
  2. When a maintenance flow is initiated, the partition is allocated with GPUs or a compute tray using `AddGpusToPartition`.
  3. After the maintenance flow is completed, the GPUs or a compute tray are removed using `RemoveGpusFromPartition`.
  4. The OFR partition becomes a `zero-gpu` partition again.
- A location- or UID-based OFR partition.
  1. When a maintenance flow is initiated, an OFR partition is created using `CreatePartition` and is allocated with a location or GPU UIDs.
  2. When a maintenance flow is completed, an OFR partition is deleted using `DeletePartition`.

7.1.1 Maintenance in a Default Partition

In a UID-based default partition, the active phase stops when a user partition is created for maintenance purposes.

1. If the Active Phase needs to be preserved, the Default Partition type must be set to `Location Based`.
2. During NMX-C initial provisioning, set the following configurations:
   - `MNNVL_ENABLE_DEFAULT_PARTITION=1` to enable the Default Partition.
   - `MNNVL_DEFAULT_PARTITION_TYPE=1` to make it location based.

To preserve the active phase:

1. Create an OFR partition with zero GPUs using `CreatePartition(numGpus=0)`.
2. When a faulty compute tray needs to be isolated, remove the faulty tray from the Default Partition using `RemoveGpusFromPartition(Location, reroute=false)`.
3. Add the tray to the OFR partition using `AddGpusToPartition(Location)`.
4. (**Optional**) Power off the compute tray and remove it from the slot.
5. (**Optional**) Insert the tray back into the same slot and power on the tray.
6. Remove the faulty tray from OFR partition using `RemoveGpusFromPartition(Location)`.
7. Add the tray back to the Default Partition using `AddGpusToPartition(Location, reroute=false)`.

7.1.2 Maintenance in a User Partition

This section provides information about the maintenance process in a user partition.

7.1.2.1 Location-Based User Partition

This section provides information about maintaining location-based user partitions.

7.1.2.1.1 Using A Zero-GPU OFR Partition

1. Create an OFR partition with `numgpus=0` using `CreatePartition(numGpus=0)`.
2. When a faulty compute tray needs to be isolated, remove the tray from the partition using `RemoveGpusFromPartition(Location, reroute=false)`.
3. Add the tray to the OFR partition using `AddGpusToPartition(Location)`.
4. (**Optional**) Power off the compute tray and remove it from the slot.
5. (**Optional**) Insert the tray back into the same slot and power on the tray
6. After you complete the maintenance, complete the following tasks:
    1. Remove the faulty tray from the OFR partition using `RemoveGpusFromPartition(Location)`.
    2. Add the tray back to the Default Partition using `AddGpusToPartition(Location, reroute=false)`.

7.1.2.1.2 Using A Location-Based OFR Partition

1. When a faulty compute tray needs to be isolated, remove the faulty tray from the partition using `RemoveGpusFromPartition(Location, reroute=false)`.
2. Create an OFR partition with the location of the faulty compute tray using `CreatePartition(Location)`.
3. (**Optional**) Power off the compute tray and remove the tray from the slot.
4. (**Optional**) Insert the tray back into the same slot and power on the tray
5. After you complete the maintenance, complete the following tasks:
    1. Delete the OFR partition using `DeletePartition(Location)`.
    2. Add the tray back to the Default Partition using `AddGpusToPartition(Location, reroute=false)`.

7.1.2.2 UID-Based User Partition

1. Create an OFR partition with `numgpus=0` using `CreatePartition(numGpus=0)`.
2. When a faulty compute tray needs to be isolated, remove the tray from the partition using `RemoveGpusFromPartition(UID, reroute=false)`.
3. Add the tray to the OFR partition using `AddGpusToPartition(UID)`.
4. (**Optional**) Power off the compute tray and remove it from the slot.
5. (**Optional**) Insert the tray back into the same slot and power on the tray.
6. After you complete the maintenance, complete the following tasks:
    1. Power off the compute tray in the OFR partition.
    2. Remove from OFR partition using `RemoveGpusFromPartition(UID)`.
    3. Add tray back to Default Partition using `AddGpusToPartition(UID, reroute=false)`.

7.1.3 Additional Maintenance Flows

Compute trays can be moved and/or replaced when Control Plane is up or down, and the Control Plane might go down for the following reasons:

- The switch tray/chassis that hosts the Control Plane is powered down.
- An NMX-C crash or restart.
- A GFM crash or restart.

A replacement action involves the following sequence:

1. (**Optional**) Power off one or more compute trays.
2. Removing and/or moving the tray to other available slots.
3. (**Optional**) Insert another compute tray into a slot from which a compute tray was removed.
4. (**Optional**) Power on the affected compute trays.

If the Control Plane is up when the replacement action is performed, it is notified about the events that pertain to the addition and removal of the compute trays. If the Control Plane is down when the replacement action is performed, it needs to come back up and reconcile the newly discovered topology with the snapshot of the topology it had saved before going down. In both scenarios, the Control Plane refreshes the state of the resources in the partition.

To better understand the impact, the following assumptions are made:

- Associated partitions are healthy before the compute trays are powered off.
- All GPUs in the tray are healthy.

7.1.3.1 Location-Based Partition

This section provides information about location-based partitions.

Replacement action: Remove the compute tray

- The removed UIDs continue to belong to the partition information and are tagged to their existing locations.
- The health of the removed GPU locations is set to `NO_NVLINK`.
- Partition Health is healthy.

Replacement action: Remove the compute tray from the existing location and insert a new tray in the same location.

- The removed GPUs are replaced with the GPUs from the new compute tray in the partition information.
- The locations are updated with the UIDs that correspond to the new compute tray.
- The health of the locations to which new GPUs were added is set to Healthy.
- The partition is healthy.

Replacement action: Remove the compute tray from the existing location and insert the tray into another partition.

- The UIDs of the removed GPUs are set to invalid (zero) against their locations in the original partition information.
- The health of the removed GPU locations is set to `NO_NVLINK`.
- The partition is healthy.

Replacement action: Remove the compute tray from the existing location and insert the tray into another location in the same partition.

- The UIDs of the removed GPUs are set to invalid (zero) against their locations in the partition information.
- The UIDs of the GPUs in their new locations are updated in the partition information.
- The health of the removed GPU locations is set to `NO_NVLINK`.
- The health of the GPUs new locations is set to Healthy.
- The partition is healthy.

7.1.3.2 UID-Based Partition

This section provides information about UIU-based partitions.

Replacement action: Remove compute tray

- The removed UIDs continue to belong to the partition information and are tagged to their existing locations.
- The health of the removed GPU UIDs is set to `NO_NVLINK`.
- The partition is healthy.

Replacement action: Remove the compute tray from the existing location and insert the new tray into the same location.

- The locations of the removed GPUs are set to invalid(zero) against their UIDs in the partition information.
- The health of the removed GPU UIDs is set to NO_NVLINK.
- The partition is healthy.

Replacement action: Remove the compute tray from the existing location and insert the tray into a new location that does not belong to a partition.

- The locations of the moved GPUs are updated in the partition information.
- The health of the moved GPU UIDs is set to Healthy.
- The partition is healthy.

Replacement action: Remove compute tray from existing location and insert the tray into a location that belongs to another partition.

- GPUs that moved are no longer in the original partition.
- The locations and UIDs of the moved GPUs are erased from the partition information
- The partition is healthy.

7.1.4 OFR Partition Example

Refer to OFR Partition for more information.

7.2 Maintenance Flow for Trunk Link Failures

This section will be updated in a future release of the guide.

7.3 Maintenance Flow for Switch Trays

Before you start to maintain the switch tray, ensure that all partitions in the NVLink domain are deleted. After a replacement switch tray is installed, the partitions need to be recreated. This section will be updated in a future release of the guide.

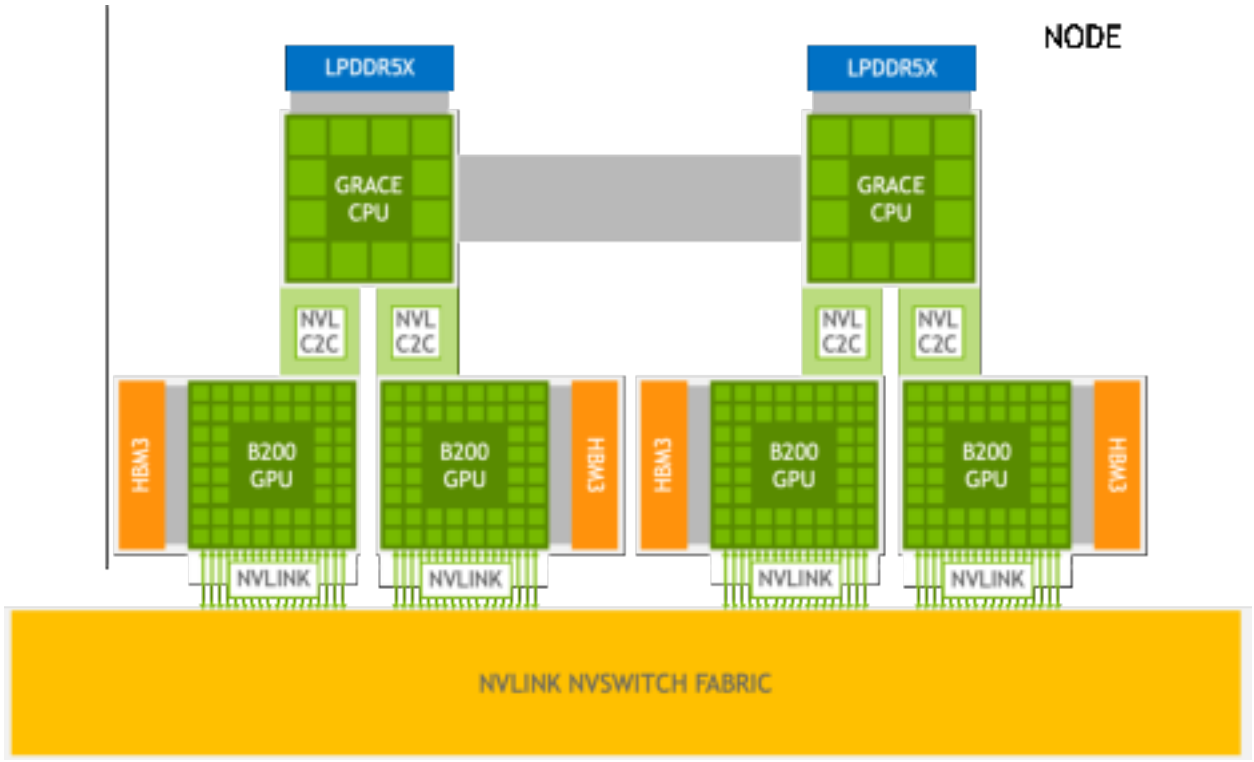
7.4 Maintenance Flow for Cable Cartridge

Before you start to maintain the cable cartridges, ensure that all partitions in the NVLink domain are deleted. After a replacement cable cartridge is installed, the partitions need to be recreated. This section will be updated in a future release of the guide.

8. EGM Support

Starting from the NVIDIA Hopper™, LPDDR5X memory is attached to the NVIDIA Grace™ CPU chiplet, which operates as a memory controller and is attached to NVIDIA GH100 using a chip-to-chip GRS interconnect. This allows the GPU to access the host memory carveout for the GPU. The feature is known as Extended GPU Memory (EGM), and it provides an increase in per-GPU memory capacity over what the HBM vidmem that is attached to GPU can provide.

Figure 7. A GB200 compute tray



8.1 Security Considerations

In NVIDIA GB200, the two NVIDIA Blackwell GPUs that are connected to the same Grace CPU using a C2C link, have access to this CPU's EGM carveout, which bypasses the stage 2 SMMU translation. As a result, if the two GPUs are isolated from each other because they are from different VMs, and/or because they are part of different NVL partitions, the isolation boundary is violated.

While the local C2C link access does not go through NVLink, we recommend that you use NVL partition as the security boundary and assign these two GPUs to the same NVLink partition when EGM is enabled. If security is a concern, VMs should also be aligned with NVLink partition boundaries.

Table 6 shows the recommended EGM configuration based on security constraints for partitions that are based on the number of compute trays/GPUs allocated to the configuration.

Table 6. The Recommended EGM Configuration

Partition Contains	Partition is at the Compute Tray Boundary	Recommended EGM Setting	Example Configuration (x denotes a GPU)
Fully allocated Compute Trays	Yes	Enable	GB200 NVL72: 4x, 8x, 12x, ...
Partially allocated Compute Trays	No	Disable	GB200 NVL72: 1x, 2x, 3x

Note EGM can be enabled or disabled on each compute tray using the BIOS Redfish API.

To enable EGM:

```
curl -k -X PATCH \
     -u "[username]:[password]" \
     -H "Content-Type: application/json" \
     -H "Accept: application/json" \
     -d '{"Attributes": {"EGM":false}}' \
     "https://[BMC IP address]/redfish/v1/Systems/System_0/Bios/Settings"
```

To disable EGM:

```
curl -k -X PATCH \
     -u "[username]:[password]" \
     -H "Content-Type: application/json" \
     -H "Accept: application/json" \
     -d '{"Attributes": {"EGM":false}}' \
     "https://[BMC IP address]/redfish/v1/Systems/System_0/Bios/Settings"
```

On compute tray, to check the EGM status of each GPU:

```
# nvidia-smi -q

...

GPU 00000008:01:00.0

....
    Capabilities
        EGM                                 : enabled
```

```
...
```

9. Multi-Cast Support

This chapter provides information about multi-cast support.

The GB200 system supports setting up 1,024 multicast teams, and this is a global resource that has to be shared across all the partitions. To manage how these resources are shared, you can reserve resources against each partition. After resources are reserved, applications running in the partition can allocate the reserved teams.

9.1 Reserving Partitions

When the partition is created, the administrator can specify the number of available groups to be used by applications in the partition by specifying `multicastGroupsLimit` (refer to Administrator-Defined Partitions). This value must be 0 or a multiple of 4, and the partitioning request will fail if an invalid value is specified or if there are no available resources to reserve. The partition information will show the current groups reserved to each partition of the domain.

```
admin@nvos:~$ nv show sdn partition

ID     Name               Num of GPUs  Health   Resiliency mode     Multicast groups
limit  Partition type  Summary

-----  -----------------  -----------  -------  ------------------
----------------------  --------------  -------

32766  Default Partition  36           healthy  adaptive_bandwidth  1024
gpuuid_based
```

Note For Default Partition users, the partition defaults to the pre-allocation of all 1,024 multicast teams. To allow user partitions to be created with pre-allocated teams, the Default Partition should be deleted and user partitions be created.

9.2 Suggested Sizing

The number of teams to reserve to a partition is determined on the expected usage by the partition's tenant. Smaller partitions should be allocated fewer multicast groups than larger partitions. The performance benefit with multicast teams is limited for smaller sized partitions. For example, partitions that have less than or equal to four GPUs will not benefit from multicast and can accomplish all traffic through unicast.

Here is a simple allocation algorithm for each partition:

`floor(3.55*NGPUs) * 4`

In this case, the allocation increases proportional to the number of GPUs in the partition. For example, a 72-GPU partition will get 1024 allocations, a 68-GPU partition will get 964 allocations, and a 64-GPU partition will get 908 allocations, and so on.

9.3 Administrator Notification

The currently consumed multicast groups can be polled from `GetPartitionInfoList: allocatedMulticastGroups` (refer to GetPartitionInfoList).

10. Admin and Tenant Workflows

An NVLink domain is configured and managed by an administrator. For example, the administrator creates partitions and performs maintenance operations, but a tenant uses the resources that the admin provides to run CUDA workloads. The tenants and admins have different views of the GPU memory fabric, and they use different tools that serve their workflows.

An admin uses the following tools to create and manage partitions and check resource states:

- NVOS CLI
- GRPC API

These tools allow the admin to look at Platform Location information, GPU, switch and partition states from the fabric side.

A tenant uses the following tools to check the availability of GPUs including their fabric state and association to partitions:

- nvidia-smi
- NVML API

These tools allow the tenant to look at Platform Location information, GPU states on the compute nodes.

10.1 Platform-Location Information

This section provides information about platform-location information.

10.1.1 Tenant View

Run the following command:

```
$ nvidia-smi -q | grep -A6 "Platform"
        Platform Info
        Chassis Serial Number             : 1783724160070
        Slot Number                       : 3
        Tray Index                        : 2
        Host ID                           : 1
        Peer Type                         : Switch Connected
        Module Id                         : 2
       GPU Fabric GUID                   : 0x168b9f9b49c971b2
```

10.1.2 Admin View

Run one of the following commands:

- NVOS CLI: `nv show platform chassis-location`
- GRPC API: `LocationInfo field in GetGpuInfoList()`

10.2 NVLink State

This section provides information about the state of NVLink.

10.2.1 Admin View

GRPC API: Health field in `GetGpuInfoList`.

10.2.2 Tenant View

Run the following command:

```
$ nvidia-smi nvlink -s

GPU 0: NVIDIA Graphics Device (UUID: GPU-7f32a571-50ca-6915-b01d-90369ce50c9d)
         Link 0: 50 GB/s
         Link 1: 50 GB/s
         Link 2: 50 GB/s
         Link 3: 50 GB/s
```

```
        Link 4: 50 GB/s
        Link 5: 50 GB/s
        Link 6: 50 GB/s
        Link 7: 50 GB/s
        Link 8: 50 GB/s
        Link 9: 50 GB/s
        Link 10: 50 GB/s
        Link 11: 50 GB/s
        Link 12: 50 GB/s
        Link 13: 50 GB/s
        Link 14: 50 GB/s
        Link 15: 50 GB/s
        Link 16: 50 GB/s
        Link 17: 50 GB/s
```

10.3 GPU Fabric State

This section provides information about the GPU fabric state.

10.3.1 Admin View

Admins cannot query the fabric state of the GPU, but they can determine whether the GPU is part of a partition using the **`partitionId`** field in the `GetGpuInfoList` API. A GPU that is part of a partition is set to a non-zero value for this field. The `ClusterUUID` state can also be determined from the **`uuid`** field of the `GetDomainStateInfo` API.

10.3.2 Tenant View

Run the following command:

```
$ nvidia-smi -q | grep -A9 "Fabric"
    Fabric
        State                             : Completed
        Status                            : Success
        CliqueId                          : 32766
        ClusterUUID                       : e951cfb5-f8d8-4226-b3e4-24e31cfdee9e
        Health
            Bandwidth                     : N/A
```

```
            Route Recovery                : N/A
            Route Unhealthy               : N/A
            Access Timeout Recovery       : N/A
```

- **State**
  - Indicates the state of the fabric probe completion.
  - Here is a list of the possible values:
    - Completed
    - In Progress
    - Not Started
    - Not Supported
- **Status**
  - Status of the fabric probe response from FM.
  - Here is a list of the possible values:
    - Success
    - Not Supported
    - Insufficient Resources
- **Clique Id**
  Clique to which this GPU belongs. Indicates NVLink Partition ID.
- **Cluster UUID**
  UUID of NVLink Domain to which this GPU belongs.
- **GPU Health**
  - Bandwidth: Is the fabric bandwidth degraded?
    The possible values are True or False.
  - Route Recovery: Is route recovery in progress?
    The possible values are True or False.
  - Route Unhealthy: Did the Fabric rerouting fail or abort, or did any other fabric error occur?
    The possible values are True or False.
  - Access Timeout Recovery: Is timeout recovery happening on the NVLink Fabric?
    The possible values are True or False.

10.4 GPU Recovery State

This section provides information about the GPU recovery state.

10.4.1 Tenant View

Here is the tenant view:

```
$ nvidia-smi -q | grep "GPU Recovery"
    GPU Recovery Action                : None
    GPU Recovery Action                : None
    GPU Recovery Action                : None
    GPU Recovery Action                : None
```

- **GPU Recovery Action**
  - Possible Values
    - **None**
      No recovery action is needed.
    - **GPU_RESET**
      - A reset is required.
      - Do not restart the application without completing a GPU reset.
    - **DRAIN_AND_RESET**
      - We recommend a reset.
      - It is safe to restart the application (memory capacity will be reduced because of dynamic page offlining), but you need to eventually reset (to get row remap).
    - **NODE_REBOOT**
      - A compute node reboot is required.
      - The application cannot restart without a node reboot.
      - A warm reboot is sufficient, and a power cycle is not necessary.
    - **DRAIN_P2P**
      - Disable job scheduling and stop all applications when convenient.
      - If persistence mode is enabled, disable it.
      - If the recovery action is still DRAIN_P2P, complete a GPU reset.

10.5 GPU Probe and Recovery Actions

Table 7 provides information about the GPU probe and the recovery actions.

Table 7. GPU Probe and Recovery Actions

Scenario	NVLink State	Fabric State (State:Status)	Recovery State	Recommended Action from the User
GPUs are not part of any partitions.	All NVLinks are down.	Not Started: Not Started	None	N/A
	One or more NVLinks are inactive.	In Progress: N/A	None	N/A
GPUs are part of an active partition **(after partition creation).**	All NVLinks are down.	Not Started: N/A	None	Reset the GPU

Scenario	NVLink State	Fabric State (State:Status)	Recovery State	Recommended Action from the User
	Some NVLinks are inactive.	Completed: Insufficient resources	None	Check NVLink state using the following command: `nvidia-smi nvlink -s`
	A missing GPU configuration.	In Progress: N/A	None	Reset the GPU
	All NVLinks are active.	Completed: Success	None	N/A
GPUs are part of an active partition **(during run-time).**	Some NVLinks go down as the result of a switch tray reboot.	Completed: Success (Stale)	Reset	Reset the GPU
	All NVLinks are active after a GPU reset.	Completed: Success	None	N/A
	GPU hits an internal error.	Completed: Success (Stale)	Drain_and_Reset	Reset the GPU
	All NVLinks are active after a GPU reset.	Completed: Success	None	N/A
	During rerouting	Completed: Success (Stale)	None	N/A Reroute is automatic in all resiliency modes

Scenario	NVLink State	Fabric State (State:Status)	Recovery State	Recommended Action from the User
		Route Recovery: Yes Route Unhealthy: N/A Bandwidth: N/A		
	After rerouting	Route Recovery: N/A Route Unhealthy: N/A Bandwidth: N/A	None	N/A
GPUs are deleted or removed from a partition	All NVLinks are active.	Completed: Success (Stale)	None	N/A

Note

- Rerouting can be initiated for reasons such as a failed trunk link in a partition, recovering a partition, or updating a partition.
- After rerouting, **`Route Unhealthy:`** can be set to **`Yes`** to indicate a routing plane loss, an internal failure during rerouting, a parked GPU (only in the Full Bandwidth Mode), non-optimal trunk link allocation (only in User Action Required Mode).

  `Bandwidth:` can be set to **`Degraded`** (only in Adaptive Bandwidth mode) to indicate a non-optimal trunk link allocation.

10.6 Virtualization

This section will be updated in future versions with additional information.

11. FAQs

1. Can a partition exist with zero GPUs?
   Yes, a partition can exist without any resources (location or gpu UIDs) associated with it.
2. Do partitions have specific features that can be enabled or disabled per tenant?
   Partition creation allows attributes like the resiliency policy, number of multicast groups, and so on to be configured. This applies to the lifetime of the partition and cannot be changed.
3. Are partitions isolated to a chassis?
   Partitions can be contained in a chassis (only access links are configured) or across chassis (access and trunk links are configured).
4. Can partitions span across NVLink domains?
   No. A Control Plane (NMX-C, SM, and FM) manages partitions in an NVLink domain.
5. What is the guideline for maximum partitions per chassis?
   The theoretical limit for the number of partitions in a chassis is the number of maximum GPUs in that chassis.
6. Is there a REST API exposed for partition management?
   Yes. Refer to GRPC APIs for more information.
7. Is storage partitioning supported?
   The scope of NVLink partitioning is at the GPU memory fabric.
8. Is Kubernetes supported?
   Users can select any technology for GPU workload orchestration.
9. Can partitions share a CPU?

   No. More information will be available in the next release of this guide.
10. How is isolation enforced in a partition?
    Control Plane configures routing on the GPUs and switches so that the memory access of the GPUs is limited to the partition boundary.
11. Does a partition become a NVLink domain of its own?
    No
12. Does a partition appear as one node or as a cluster?
    Partitions can be configured to span a group of nodes or one node.
13. How long does it take to create or set up a partition?
    It takes a few seconds to set up a partition.
14. Is there a default software stack for a partition to start functioning?
    At the minimum, we recommend that you have a virtual machine (VM) that has the GPUs assigned to it with the GPU driver.

15. What happens to the compute nodes and links when a partition is deleted?

    The resources in the partition are reclaimed and are no longer available to run workloads.

16. Is running a VM required for isolation or recommended?
    TBD. More information will be available in the next release of this guide.

17. What is the smallest granularity for creating a partition?
    TBD**.** More information will be available in the next release of this guide.

18. What is the underlying mechanism of isolation through partition?
    To enforce isolation, GPUs are associated with Pkeys. Only GPUs that belong to the same Pkey can communicate.

19. After a partition ID is created, can it be modified?

    No.

20. What happens when a GPU dies in a partition?
    The GPU continues to be part of the partition until it is explicitly removed for servicing.

21. How can I get the partition properties?
    Use the `GetPartitionInfoList` GRPC API.

22. What aspects of a partition can be updated?
    In a partition, you can add or remove GPUs, which causes it to allocate/deallocate a proportional number of NVLinks (access and trunk).

23. What happens when a compute or switch tray is replaced in a partition?
    Metadata from the old compute tray is purged from the partition, and the partition is updated with the metadata from the new compute tray (refer to Maintenance).

24. Can traffic in one partition affect another?
    No.

12. Appendix

The appendix provides additional information for this guide.

12.1 Partitioning Examples

This section provides information about partitioning examples.

12.1.1 Common Code

Here is the common code:

```
// gRPC client used to talk with NMX-C

#include <iostream>

// Assumes implementation of service functions from the proto file
```

```
#include <grpc-client.h> // Assumes defined global stub grpc ptr to do calls
#define UID_BASED true


int initAndWaitForConfigured()
{
    // Initial RPC call for handshake
    ServerHello sh = grpc->Hello();
    if(sh.serverheader().returncode() != NMX_ST_SUCCESS)
        return sh.serverheader().returncode();


    // Subscribe to state change notifications (optional)
    // This is a stream and needs to be handled accordingly,
    // As this is a partitioning document that is not covered here
    ServerNotification sn = grpc->Subscribe();



    // Sleep until fabric reaches config done state (can be polled or found from the
stream above)
    bool configured = false;
    do {
        GetDomainStateInfoRequest dsiRequest;
        DomainStateInfo dsiResp = grpc->GetDomainStateInfo(dsiRequest);
        configured = dsiResp.serverheader().returncode() == NMX_ST_SUCCESS
                        && dsiResp.controlplanestate() ==
NMX_CONTROL_PLANE_STATE_CONFIGURED;
        if(!configured)
            sleep(10);
    } while(!configured);


    return NMX_ST_SUCCESS;
```

```
}


// Filter examples
// This filter allows through all healthy GPUs
// makes a big single partition for all discovered unallocated GPUs
bool filterAllHealthy(const GpuInfo & gpu)
{
    return gpu.health() == NMX_GPU_HEALTH_HEALTHY;
}


// Filter to make a partition for a single compute node
// will make a partition local only to 1/2/1 chassis/slot/host
bool filterOnlyOneComputeNode(const GpuInfo & gpu)
{
    const uint32_t chassis = 1;
    const uint32_t slot = 2;
    const uint32_t host = 1;
    auto & location = gpu.loc().location();
    return location.chassisid() == chassis && location.slotNumber() == slot &&
location.hostid() == host;
}


// This filter makes a cross chassis partition by ignoring the chassisId
// the resulting partition will have trunk links allocated to it
bool filterSmallInterChassis(const GpuInfo & gpu)
{
    // Purposefully ignore chassis so we get chassis 1 and 2
    const uint32_t slot = 2;
    const uint32_t host = 1;
    auto & location = gpu.loc().location();
```

```
    return location.slotNumber() == slot && location.hostid() == host;
}
```

12.1.2 Creating a GetGpuInfoList-Based Partition

Here is an example of how to create a `GetGpuInfoList`-based partition:

```
// GetGpuInfoList Based partition creation
int main() {
    // Do Init specified above
    int ret = initAndWaitForConfigured();
    if(ret != NMX_ST_SUCCESS)
        return ret;

    // Get all GPU info
    GetGpuInfoListRequest gpuInfoReq;
    gpuInfoReq.set_attr(NMX_GPU_ATTR_ALL);
    GetGpuInfoListResponse gpuInfoResp = grpc->GetGpuInfoList(gpuInfoReq);
    if(gpuInfoResp.serverheader().returncode() != NMX_ST_SUCCESS)
        return gpuInfoResp.serverheader().returncode();

    CreatePartitionRequest createPartitionReq;
    // Start filling up createPartitionReq with all the free GPUs
    for(int i = 0; i < gpuInfoResp.gpuinfolist_size(); i++) {
        auto & currGpu = gpuInfoResp.gpuinfolist(i);
        // Filter out all GPUs that are already allocated (they cannot be used)
        if(currGpu.partitionid() != 0)
            continue;

        // Write filter implementation here if desired.
```

```
        // Examples are given below should return true if we want GPU in the partition
        if(filter(currGpu))
        {
            GpuResourceId * res = createPartitionReq.add_gpuresourceid();
            if(UID_BASED)
                res->gpuUid = currGpu.gpuuid();
            else {
                GpuLocation * gpuloc = new GpuLocation;
                Location * nodeLoc = new Location;
                nodeLoc->set_chassisid(currGpu.loc().location().chassisid());
                nodeLoc->set_slotNumber(currGpu.loc().location().slotNumber());
                nodeLoc->set_hostid(currGpu.loc().location.hostid());
                gpuLoc->set_loc(nodeLoc);
                gpuLoc->set_gpuid(currGpu.gpuid());
                res->set_gpulocation(gpuloc);
            }
        }
    }

    // Do actual create partition call
    CreatePartitionResponse createPartitionResp =
grpc->CreatePartition(createPartitionReq);
    if(createPartitionResp.serverheader().returncode() != NMX_ST_SUCCESS)
        return createPartitionResp.serverheader().returncode();

    std::cout << "Created partition: " << createPartitionResp.partitionid() <<
std::endl;

    return 0;
}
```

12.1.3 Creating a GetComputeLocationList-Based Partition

Here is an example of how to create a `GetComputeLocationList`-based partition:

```
// Alternate way to find GPUs by just looking at compute node free lists

int main() {

    // Do Init specified above

    int ret = initAndWaitForConfigured();

    if(ret != NMX_ST_SUCCESS)

        return ret;


    // Get number of gpus per compute node

    GetDomainPropertiesRequest dpRequest;

    DomainProperties dp = grpc->GetDomainProperties(dpRequest);

    if(dp.serverheader().returncode() != NMX_ST_SUCCESS)

        return dp.serverheader().returncode();


    uint32_t gpusPerCn = dp.maxgpuspercomputenode();


    // Get compute node list filtering on free compute nodes that are healthy

    GetComputeNodeLocationListRequest cnlRequest;

    cnlRequest.set_attr(NMX_COMPUTE_NODE_ATTR_FREE);

    cnlRequest.set_health(NMX_COMPUTE_NODE_HEALTH_HEALTHY);

    // Optional if you only want nodes from chassis 1

    cnlRequest.set_chassisid(1);


    // Do the actual request

    GetComputeNodeLocationListResponse cnlResp =
grpc->GetComputeNodeLocationList(cnlRequest);

    if(cnlResp.serverheader().returncode() != NMX_ST_SUCCESS)
```

```
        return cnlResp.serverheader().returncode();


    CreatePartitionRequest cpRequest;
    for(size_t i = 0; i < cnlResp.loclist_size(); i++)
    {
        auto & loc = cnlResp.loclist(i);


        // Write filter implementation here if desired.
        // return true if we want the location in the partition
        if(filter(loc))
        {
            for(uint32_t j = 0; j < gpusPerCn; j++)
            {
                GpuResourceId * res = createPartitionReq.add_gpuresourceid();
                GpuLocation * gpuloc = new GpuLocation;
                Location * nodeLoc = new Location;
                nodeLoc->set_chassisid(loc.chassisid());
                nodeLoc->set_slotNumber(loc.slotNumber());
                nodeLoc->set_hostid(loc.hostid());
                gpuLoc->set_loc(nodeLoc);
                gpuLoc->set_gpuid(j);
                res->set_gpulocation(gpuloc);
            }
        }
    }


    // Create Partition
    CreatePartitionResponse createPartitionResp =
grpc->CreatePartition(createPartitionReq);
    if(createPartitionResp.serverheader().returncode() != NMX_ST_SUCCESS)
```

```
        return createPartitionResp.serverheader().returncode();


    std::cout << "Created partition: " << createPartitionResp.partitionid() <<
std::endl;


    return 0;
}
```

12.1.4 Deleting the Default Partition

Here is an example of deleting a Default Partition:

```
// Delete partition example

#define DEFAULT_PARTITION_ID (0x7FFF - 1)

int main() {

    // Do Init specified above

    int ret = initAndWaitForConfigured();

    if(ret != NMX_ST_SUCCESS)

        return ret;


    // Set partition id to Default Partition

    DeletePartitionRequest dpReq;

    dpReq.set_partitionid(DEFAULT_PARTITION_ID);


    // Do the actual deletion call

    DeletePartitionResponse dpResp = grpc->DeletePartition(dpReq);

    if(dpResp.serverheader().returncode() != NMX_ST_SUCCESS)

        return dpResp.serverheader().returncode();

}
```

12.1.5 Adding or Removing GPUs from the Default Partition

Here is an example of adding or removing GPUs from the Default Partition:

```
// Add/Remove GPUs from location based Default Partition example

#define DEFAULT_PARTITION_ID (0x7FFF - 1)

int main() {

    // Do Init specified above

    int ret = initAndWaitForConfigured();

    if(ret != NMX_ST_SUCCESS)

        return ret;


    // Assume we want to remove a specific GPU from a location based Default Partition

    // Want to remove 1/1/1/2 (chassis/slot/host/gpu)

    UpdatePartitionRequest upReq;

    upReq.set_partitionid(DEFAULT_PARTITION_ID);

    GpuLocation * gpuLoc = upReq.add_locationlist();

    Location * nodeLoc = new Location;

    nodeLoc->set_chassisid(1);

    nodeLoc->set_slotNumber(1);

    nodeLoc->set_hostid(1);

    gpuLoc->set_loc(nodeLoc);

    gpuLoc.set_gpuid(2);


    // Remove the location from the location based Default Partition

    UpdatePartitionResponse upResp = grpc->RemoveGpusFromPartition(upReq);

    if(upResp.serverheader().returncode() != NMX_ST_SUCCESS)

        return upResp.serverheader().returncode();


    // Add the location back
```

```
    upResp = grpc->AddGpusToPartition(upReq);
    if(upResp.serverheader().returncode() != NMX_ST_SUCCESS)
        return upResp.serverheader().returncode();
}
```

12.1.6 Printing Currently Active Partitions

Here is an example of printing the current, active partitions:

```
// List current partitions
int main() {
    // Do Init specified above
    int ret = initAndWaitForConfigured();
    if(ret != NMX_ST_SUCCESS)
        return ret;

    GetPartitionInfoListRequest pilReq;
    // Note that you can specify which IDs you want as part of the request,
    // but specifying empty list returns all partitions

    // Do the actual request
    GetPartitionInfoListResponse pilResp = grpc->GetPartitionInfoList(pilReq);

    for(size_t i = 0; i < pilResp.partitioninfolist_size(); i++)
    {
        auto & info = pilResp.partitioninfolist(i);
        static const char* healthStr[] = {"Unknown", "Healthy", "Degraded", "No
NVLink", "Unhealthy"};
        static const char* typeStr[] = {"Undefined", "Location", "GPU UID"};
        static const char* resiliencyStr[] = {"Undefined", "Full Bandwidth", "Adaptive
Bandwidth", "User Action Required"};
```

```
        std::cout << "Printing partition ID " << info.partitionid().partitionid() <<
std::endl
            << "\tName: " << (info.has_name() ? info.name() : std::string("N/A")) <<
std::endl
            << "\tNum GPUs: " << info.numgpus() << std::endl
            << "\tHealth: " << healthStr[info.health()] << std::endl
            << "\tType: " << typeStr[info.type()] << std::endl
            << "\tResiliency Mode: " << resiliencyStr[info.attr().resiliencymode()] <<
std::endl
            << "\tMax Multicast Groups: " << info.attr().multicastgrouplimit() <<
std::endl
            << "\tAllocated Multicast Groups: " << info.numallocatedmulticastgroups()
<< std::endl
            << "\tGPU\tLocation\tUID" << std::endl;
        for(size_t j = 0; j < info.gpulocationlist_size(); j++)
        {
            auto & gpuLoc = info.gpulocationlist(j);
            std::cout << "\t" << j
                << "\t" << gpuLoc.loc().chassisid() << "/" <<
gpuLoc.loc().slotNumber() << "/" << gpuLoc.loc().hostid() << "/" << gpuLoc.gpuid()
                << "\t" << info.gpuuidlist(j) << std::endl;
        }
    }
}
```

12.1.7 OFR Partition

Here is an example of of an OFR partition:

```
// Out For Repair (OFR) Partition example (assumes all gpus are in location based
Default Partition)
#define DEFAULT_PARTITION_ID (0x7FFF - 1)
int main() {
```

```
    // Do Init specified above
    int ret = initAndWaitForConfigured();
    if(ret != NMX_ST_SUCCESS)
        return ret;

    // Get number of gpus per compute node
    GetDomainPropertiesRequest dpRequest;
    DomainProperties dp = grpc->GetDomainProperties(dpRequest);
    if(dp.serverheader().returncode() != NMX_ST_SUCCESS)
        return dp.serverheader().returncode();

    uint32_t gpusPerCn = dp.maxgpuspercomputenode();

    // Create Partition with 0 GPUs
    CreatePartitionRequest ofrRequest;
    CreatePartitionResponse ofrResp = grpc->CreatePartition(ofrRequest);
    if(ofrResp.serverheader().returncode() != NMX_ST_SUCCESS)
        return ofrResp.serverheader().returncode();

    const uint32_t ofrPartitionId = ofrResp.partitionid();
    std::cout << "Created OFR partition: " << ofrPartitionId << std::endl;

    // Create a structure to compare the grpc structs (basically so we can use an
std::set)
    struct LocationWithOps {
        uint32_t chassisId;
        uint32_t slotNumber;
        uint32_t hostId;
        bool operator<(const LocationWithOps & rhs) const
        {
```

```
            if(chassisId != rhs.chassisId)
                return chassisId < rhs.chassisId;
            if(slotNumber != rhs.slotNumber)
                return slotNumber < rhs.slotNumber;
            return hostId < rhs.hostId;
        }
    }

    // Loop forever looking for compute nodes that need maintainence
    while(true)
    {
        // std::set to get healthy compute nodes (no filter for multiple non-healthy
health states)
        std::set<LocationWithOps> healthyComputeNodes;

        // Request all compute nodes first
        GetComputeNodeLocationListRequest cnlRequest;
        GetComputeNodeLocationListResponse cnlAll =
grpc->GetComputeNodeLocationList(cnlRequest);
        if(cnlAll.serverheader().returncode() != NMX_ST_SUCCESS)
            return cnlAll.serverheader().returncode();

        // Request healthy compute nodes only
        cnlRequest.set_health(NMX_COMPUTE_NODE_HEALTH_HEALTHY);
        GetComputeNodeLocationListResponse cnlHealthy =
grpc->GetComputeNodeLocationList(cnlRequest);
        if(cnlHealthy.serverheader().returncode() != NMX_ST_SUCCESS)
            return cnlHealthy.serverheader().returncode();

        // Insert all the healthy compute nodes to a set
```

```
        for(uint32_t i = 0; i < cnlHealthy.loclist_size(); i++)
        {
            auto & node = cnlHealthy.loclist(i);
            healthyComputeNodes.insert({node.chassisid(), node.slotNumber(),
node.hostid()});
        }

        // Loop through all the compute nodes
        for(uint32_t i = 0; i < cnlAll.loclist_size(); i++)
        {
            auto & node = cnlAll.loclist(i);
            LocationWithOps currLoc = {
                node.chassisid(), node.slotNumber(), node.hostid()
            };
            // If the compute node is in healthy list we dont need to perform
maintainence
            // Just skip it.
            if(cnlHealthy.count(currLoc) != 0)
                continue;

            // Make an update request with all the gpus in the unhealthy location
            UpdatePartitionRequest upReq;
            upReq.set_partitionid(DEFAULT_PARTITION_ID);
            for(uint32_t j = 0; j < gpusPerCn; j++)
            {
                GpuLocation * gpuLoc = upReq.add_locationlist();
                Location * nodeLoc = new Location;
                nodeLoc->set_chassisid(node.chassisid());
                nodeLoc->set_slotNumber(node.slotNumber());
                nodeLoc->set_hostid(node.hostid());
```

```
            gpuLoc->set_loc(nodeLoc);
            gpuLoc.set_gpuid(j);
        }

        // Remove the unhealthy location from the location based Default Partition
        UpdatePartitionResponse upResp = grpc->RemoveGpusFromPartition(upReq);
        if(upResp.serverheader().returncode() != NMX_ST_SUCCESS)
            return upResp.serverheader().returncode();

        // Change partition id to OFR and add same GPUs that were just removed
        upReq.set_partitionid(ofrPartitionId);
        upResp = grpc->AddGpusToPartition(upReq);
        if(upResp.serverheader().returncode() != NMX_ST_SUCCESS)
            return upResp.serverheader().returncode();

        // Actually perform the maintainence here
        do_maintainence_for_ofr();

        // Remove GPUs from the OFR partition (maintainence is done)
        upResp = grpc->RemoveGpusFromPartition(upReq);
        if(upResp.serverheader().returncode() != NMX_ST_SUCCESS)
            return upResp.serverheader().returncode();

        // Add them back to the Default Partition (should be healthy now)
        upReq.set_partitionid(DEFAULT_PARTITION_ID);
        upResp = grpc->AddGpusToPartition(upReq);
        if(upResp.serverheader().returncode() != NMX_ST_SUCCESS)
            return upResp.serverheader().returncode();
    }
```

```
    }

    return 0;
```

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, Grace, Hopper, NVLink, NVSwitch, and CUDA 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.

VESA DisplayPort

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

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