NVIDIA Multi-Instance GPU User Guide

User Guide

RN-08625-v2.0 _v01 | August 2024

Table of Contents

Chapter 1. Introduction

The new Multi-Instance GPU (MIG) feature allows GPUs (starting with NVIDIA Ampere architecture) to be securely partitioned into up to seven separate GPU Instances for CUDA applications, providing multiple users with separate GPU resources for optimal GPU utilization. This feature is particularly beneficial for workloads that do not fully saturate the GPU's compute capacity and therefore users may want to run different workloads in parallel to maximize utilization.

For Cloud Service Providers (CSPs), who have multi-tenant use cases, MIG ensures one client cannot impact the work or scheduling of other clients, in addition to providing enhanced isolation for customers.

With MIG, each instance's processors have separate and isolated paths through the entire memory system - the on-chip crossbar ports, L2 cache banks, memory controllers, and DRAM address busses are all assigned uniquely to an individual instance. This ensures that an individual user's workload can run with predictable throughput and latency, with the same L2 cache allocation and DRAM bandwidth, even if other tasks are thrashing their own caches or saturating their DRAM interfaces. MIG can partition available GPU compute resources (including streaming multiprocessors or SMs, and GPU engines such as copy engines or decoders), to provide a defined quality of service (QoS) with fault isolation for different clients such as VMs, containers or processes. MIG enables multiple GPU Instances to run in parallel on a single, physical NVIDIA Ampere GPU.

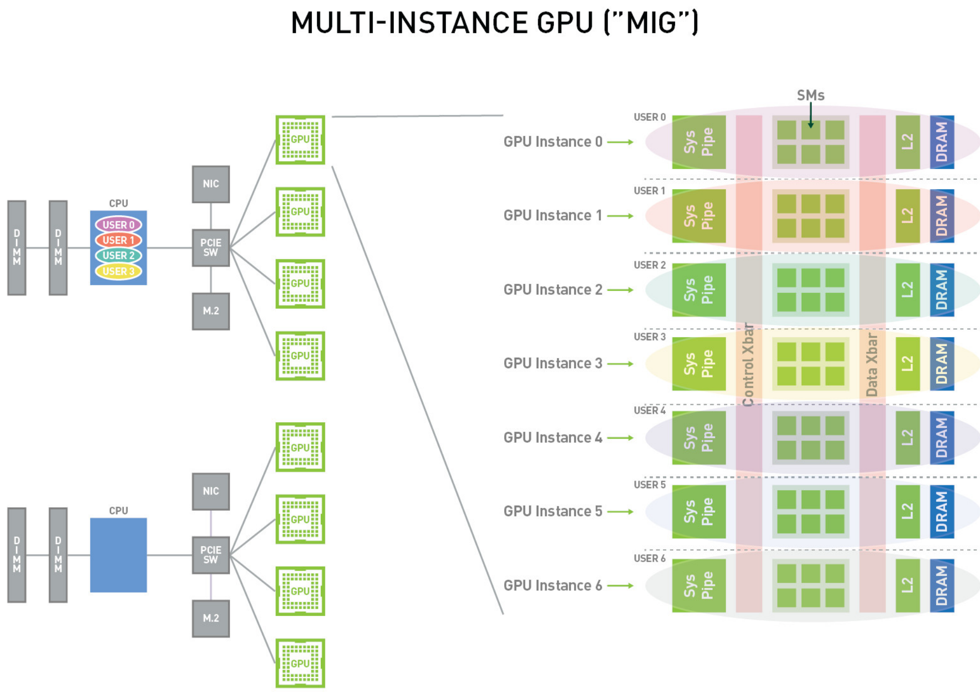
With MIG, users will be able to see and schedule jobs on their new virtual GPU Instances as if they were physical GPUs. MIG works with Linux operating systems, supports containers using Docker Engine, with support for Kubernetes and virtual machines using hypervisors such as Red Hat Virtualization and VMware vSphere.

MIG supports the following deployment configurations:

- Bare-metal, including containers
- GPU pass-through virtualization to Linux guests on top of supported hypervisors
- vGPU on top of supported hypervisors

MIG allows multiple vGPUs (and thereby VMs) to run in parallel on a single GPU, while preserving the isolation guarantees that vGPU provides. For more information on GPU partitioning using vGPU and MIG, refer to the technical brief.

Figure 1. MIG Overview



The purpose of this document is to introduce the concepts behind MIG, deployment considerations and provide examples of MIG management to demonstrate how users can run CUDA applications on MIG supported GPUs.

Chapter 2. Supported GPUs

MIG is supported on GPUs starting with the NVIDIA Ampere generation (i.e. GPUs with compute capability >= 8.0). The following table provides a list of supported GPUs:

Table 1. Supported GPU Products

Product	Architecture	Microarchitec	Compute Capability	Memory Size	Max Number of Instances
H100-SXM5	Hopper	GH100	9.0	80GB	7
H100-PCIE	Hopper	GH100	9.0	80GB	7
H100-SXM5	Hopper	GH100	9.0	94GB	7
H100-PCIE	Hopper	GH100	9.0	94GB	7
H100 on GH200	Hopper	GH100	9.0	96GB	7
H200-SXM5	Hopper	GH100	9.0	141GB	7
A100-SXM4	NVIDIA Ampere	GA100	8.0	40GB	7
A100-SXM4	NVIDIA Ampere	GA100	8.0	80GB	7
A100-PCIE	NVIDIA Ampere	GA100	8.0	40GB	7
A100-PCIE	NVIDIA Ampere	GA100	8.0	80GB	7
A30	NVIDIA Ampere	GA100	8.0	24GB	4

Additionally, MIG is supported on systems that include the supported products above such as DGX, DGX Station and HGX.

Chapter 3. Supported Configurations

Supported deployment configurations with MIG include

- Bare-metal, including containers and Kubernetes
- GPU pass-through virtualization to Linux guests on top of supported hypervisors
- vGPU on top of supported hypervisors

Chapter 4. Virtualization

MIG can be used with two types of virtualization:

- Under Linux guests on supported hypervisors, when MIG-supported GPUs are in GPU pass-through, the same workflows, tools and profiles available on bare-metal can be used.
- MIG allows multiple vGPUs (and thereby VMs) to run in parallel on a single MIG-supported GPU, while preserving the isolation guarantees that vGPU provides. To configure a GPU for use with vGPU VMs, refer to the chapter in the vGPU Software User Guide. Refer also to the technical brief for more information on GPU partitioning with vGPU.

Chapter 5. Concepts

5.1. Terminology

This section introduces some terminology used to describe the concepts behind MIG.

Streaming Multiprocessor

A streaming multiprocessor (SM) executes compute instructions on the GPU.

GPU Context

A GPU context is analogous to a CPU process. It encapsulates all the resources necessary to execute operations on the GPU, including a distinct address space, memory allocations, etc. A GPU context has the following properties:

- Fault isolation
- Individually scheduled
- Distinct address space

GPU Engine

A GPU engine is what executes work on the GPU. The most commonly used engine is the Compute/Graphics engine that executes the compute instructions. Other engines include the copy engine (CE) that is responsible for performing DMAs, NVDEC for video decoding, NVENC for encoding, etc. Each engine can be scheduled independently and execute work for different GPU contexts.

GPU Memory Slice

A GPU memory slice is the smallest fraction of the GPU's memory, including the corresponding memory controllers and cache. A GPU memory slice is roughly one eighth of the total GPU memory resources, including both capacity and bandwidth.

GPU SM Slice

A GPU SM slice is the smallest fraction of the SMs on the GPU. A GPU SM slice is roughly one seventh of the total number of SMs available in the GPU when configured in MIG mode.

GPU Slice

A GPU slice is the smallest fraction of the GPU that combines a single GPU memory slice and a single GPU SM slice.

GPU Instance

A GPU Instance (GI) is a combination of GPU slices and GPU engines (DMAs, NVDECs, etc.). Anything within a GPU instance always shares all the GPU memory slices and other GPU engines, but it's SM slices can be further subdivided into compute instances (CI). A GPU instance provides memory QoS. Each GPU slice includes dedicated GPU memory resources which limit both the available capacity and bandwidth, and provide memory QoS. Each GPU memory slice gets 1/8 of the total GPU memory resources and each GPU SM slice gets 1/7 of the total number of SMs.

Compute Instance

A GPU instance can be subdivided into multiple compute instances. A Compute Instance (CI) contains a subset of the parent GPU instance's SM slices and other GPU engines (DMAs, NVDECs, etc.). The CIs share memory and engines.

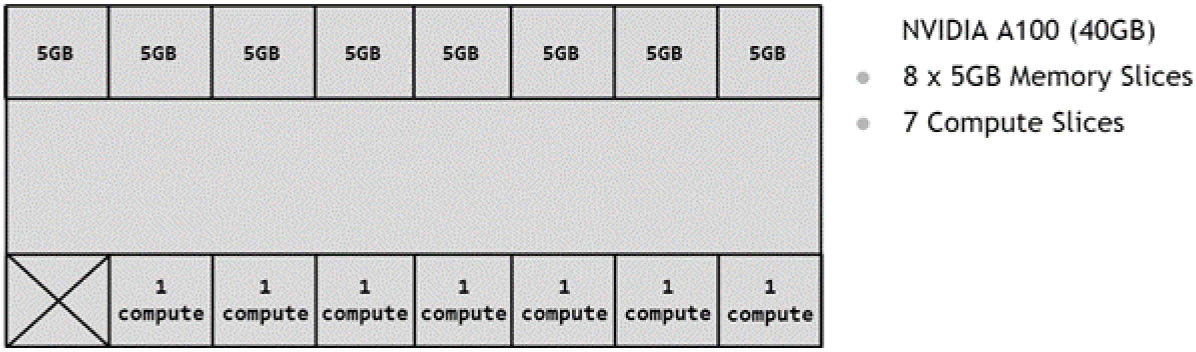
5.2. Partitioning

Using the concepts introduced above, this section provides an overview of how the user can create various partitions on the GPU. For illustration purposes, the document will use the A100-40GB as an example, but the process is similar for other GPUs that support MIG.

GPU Instance

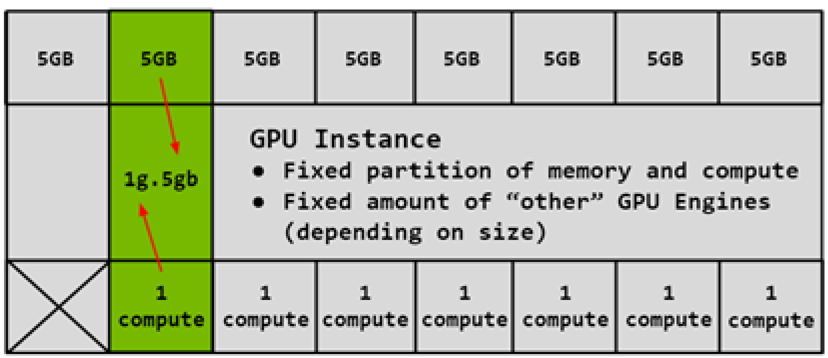
Partitioning of the GPU happens using memory slices, so the A100-40GB GPU can be thought of having 8x5GB memory slices and 7 SM slices as shown in the diagram below.

Figure 2. Available Slices on A100



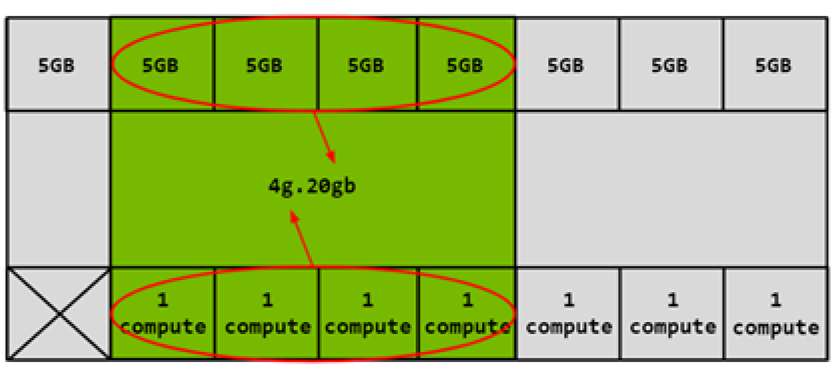
As explained above, then to create a GPU Instance (GI) requires combining some number of memory slices with some number of compute slices. In the diagram below, a 5GB memory slice is combined with 1 compute slice to create a `1g.5gb` GI profile:

Figure 3. Combining Memory and Compute Slices



Similarly, 4x5GB memory slices can be combined with 4x1 compute slices to create the `4g.5gb` GI profile:

Figure 4. Combining Memory and Compute Slices

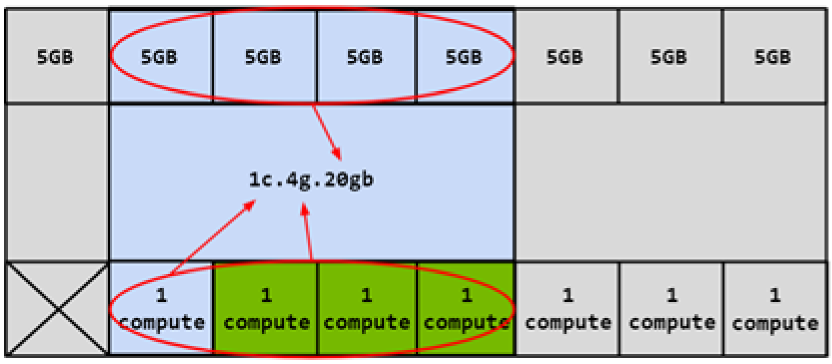


Compute Instance

The compute slices of a GPU Instance can be further subdivided into multiple Compute Instances (CI), with the CIs sharing the engines and memory of the parent GI, but each CI has dedicated SM resources.

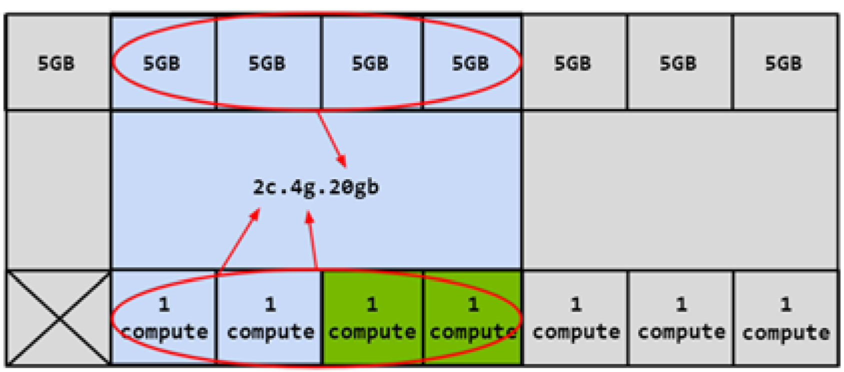
Using the same `4g.20gb` example above, a CI may be created to consume only the first compute slice as shown below:

Figure 5. Combining Memory and Compute Slices



In this case, 4 different CIs can be created by choosing any of the compute slices. Two compute slices can also be combined together to create a `2c.4g.20gb` profile:

Figure 6. Combining Memory and Compute Slices



In this example, 3 compute slices can also be combined to create a `3c.4g.20gb` profile or all 4 can be combined to create a `4c.4g.20gb` profile. When all 4 compute slices are combined, the profile is simply referred to as the `4g.20gb` profile.

Refer to the sections on the canonical naming scheme and the CUDA device terminology.

Profile Placement

The number of slices that a GI can be created with is not arbitrary. The NVIDIA driver APIs provide a number of "GPU Instance Profiles" and users can create GIs by specifying one of these profiles.

On a given GPU, multiple GIs can be created from a mix and match of these profiles, so long as enough slices are available to satisfy the request.

> Note:
>
> The table below shows the profile names on the A100-SXM4-40GB product. For A100-SXM4-80GB, the profile names will change according to the memory proportion - for example, `1g.10gb`, `2g.20gb`, `3g.40gb`, `4g.40gb`, `7g.80gb` respectively.
>
> For a list of all supported combinations of profiles on MIG-enabled GPUs, refer to the section on supported profiles.

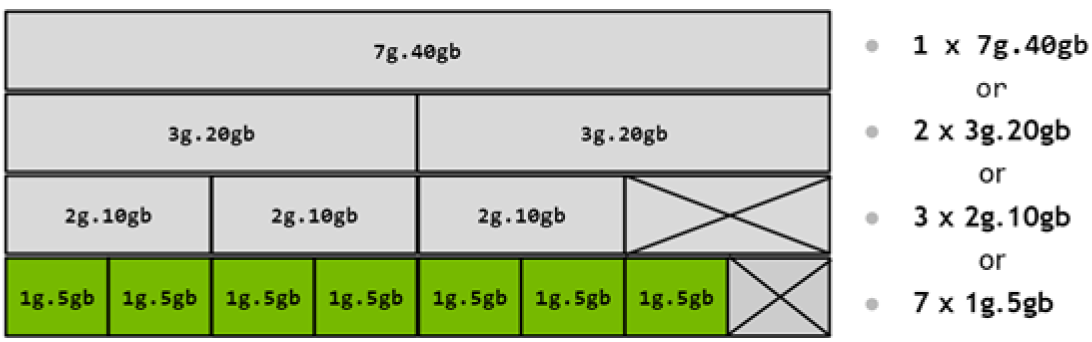
Table 2. GPU Instance Profiles on A100

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	L2 Cache Size	Copy Engines	Number of Instances Available
MIG 1g.5gb	1/8	1/7	0 NVDECs /0 JPEG /0 OFA	1/8	1	7
MIG 1g.5gb +me	1/8	1/7	1 NVDEC /1 JPEG /1 OFA	1/8	1	1 (A single 1g profile can include media extensions)
MIG 1g.10gb	1/8	1/7	1 NVDECs /0 JPEG /0 OFA	1/8	1	4
MIG 2g.10gb	2/8	2/7	1 NVDECs /0 JPEG /0 OFA	2/8	2	3
MIG 3g.20gb	4/8	3/7	2 NVDECs /0	4/8	3	2

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	L2 Cache Size	Copy Engines	Number of Instances Available
			JPEG /0 OFA			
MIG 4g.20gb	4/8	4/7	2 NVDECs /0 JPEG /0 OFA	4/8	4	1
MIG 7g.40gb	Full	7/7	5 NVDECs /1 JPEG /1 OFA	Full	7	1

The diagram below shows a pictorial representation of how to build all valid combinations of GPU instances.

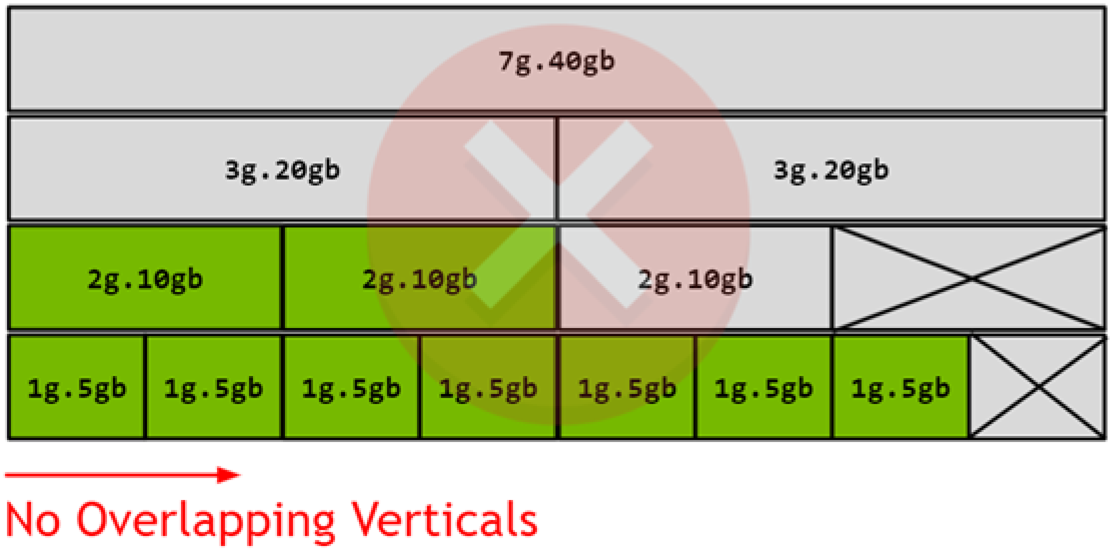
Figure 7. MIG Profiles on A100



In this diagram, a valid combination can be built by starting with an instance profile on the left and combining it with other instance profiles as you move to the right, such that no two profiles overlap vertically. For a list of all supported combinations and placements of profiles on A100 and A30, refer to the section on supported profiles.

Note that prior to NVIDIA driver release R510, the combination of a (4 memory, 4 compute) and a (4 memory, 3 compute) profile was not supported. This restriction no longer applies on newer drivers.

Figure 8. Profile Placements on A100



Note that the diagram represents the physical layout of where the GPU Instances will exist once they are instantiated on the GPU. As GPU Instances are created and destroyed at different locations, fragmentation can occur, and the physical position of one GPU Instance will play a role in which other GPU Instances can be instantiated next to it.

5.3. CUDA Concurrency Mechanisms

MIG has been designed to be largely transparent to CUDA applications - so that the CUDA programming model remains unchanged to minimize programming effort. CUDA already exposes multiple technologies for running work in parallel on the GPU and it is worthwhile showcasing how these technologies compare to MIG. Note that streams and MPS are part of the CUDA programming model and thus work when used with GPU Instances.

CUDA Streams are a CUDA Programming model feature where, in a CUDA application, different work can be submitted to independent queues and be processed independently by the GPU. CUDA streams can only be used within a single process and don't offer much isolation - the address space is shared, the SMs are shared, the GPU memory bandwidth, caches and capacity are shared. And lastly any errors affect all the streams and the whole process.

MPS is the CUDA Multi-Process service. It allows co-operative multi process applications to share compute resources on the GPU. It's commonly used by MPI jobs that cooperate, but it has also been used for sharing the GPU resources among unrelated applications, while accepting the challenges that such a solution brings. MPS currently does not offer error isolation between clients and while streaming multiprocessors used by each MPS client can be optionally limited to a percentage of all SMs, the scheduling hardware is still shared. Memory bandwidth, caches and capacity are all shared between MPS clients.

Lastly, MIG is the new form of concurrency offered by NVIDIA GPUs while addressing some of the limitations with the other CUDA technologies for running parallel work.

Table 3. CUDA Concurrency Mechanisms

	Streams	**MPS**	**MIG**
Partition Type	Single Process	Logical	Physical
Max Partitions	Unlimited	48	7
SM Performance Isolation	No	Yes (by percentage, not partitioning)	Yes
Memory Protection	No	Yes	Yes
Memory Bandwidth QoS	No	No	Yes
Error Isolation	No	No	Yes
Cross-Partition Interop	Always	IPC	Limited IPC
Reconfigure	Dynamic	Process Launch	When Idle

Chapter 6. Deployment Considerations

MIG functionality is provided as part of the NVIDIA GPU driver.

- H100 GPUs are supported starting with CUDA 12/R525 drivers.
- A100 and A30 GPUs are supported starting with CUDA 11/R450 drivers.

6.1. System Considerations

The following system considerations are relevant for when the GPU is in MIG mode.

- MIG is supported only on Linux operating system distributions supported by CUDA. It is also recommended to use the latest NVIDIA Datacenter Linux. Refer to the quick start guide.

  > Note:
  >
  > Also note the device nodes and `nvidia-capabilities` for exposing the MIG devices. The `/proc` mechanism for system-level interfaces is deprecated as of 450.51.06 and it is recommended to use the `/dev` based system-level interface for controlling access mechanisms of MIG devices through cgroups. This functionality is available starting with 450.80.02+ drivers.

- Supported configurations include
  - Bare-metal, including containers
  - GPU pass-through virtualization to Linux guests on top of supported hypervisors
  - vGPU on top of supported hypervisors

  MIG allows multiple vGPUs (and thereby VMs) to run in parallel on a single A100, while preserving the isolation guarantees that vGPU provides. For more information on GPU partitioning using vGPU and MIG, refer to the technical brief.
- Setting MIG mode on the A100/A30 requires a GPU reset (and thus super-user privileges). Once the GPU is in MIG mode, instance management is then dynamic. Note that the setting is on a per-GPU basis.

- On NVIDIA Ampere GPUs, similar to ECC mode, MIG mode setting is persistent across reboots until the user toggles the setting explicitly
- All daemons holding handles on driver modules need to be stopped before MIG enablement.
- This is true for systems such as DGX which may be running system health monitoring services such as nvsm or GPU health monitoring or telemetry services such as DCGM.
- Toggling MIG mode requires the `CAP_SYS_ADMIN` capability. Other MIG management, such as creating and destroying instances, requires superuser by default, but can be delegated to non-privileged users by adjusting permissions to MIG capabilities in `/proc/`.

6.2. Application Considerations

Users should note the following considerations when the A100 is in MIG mode:

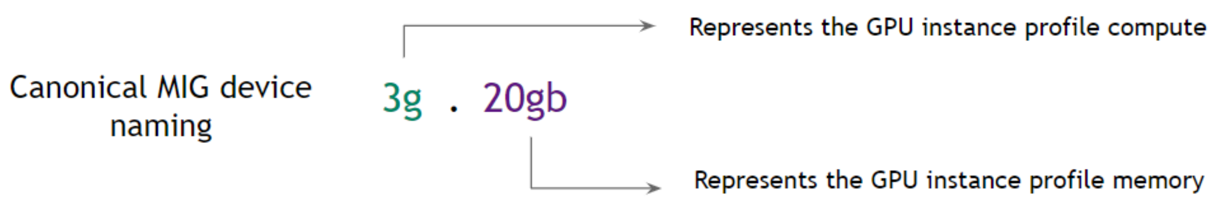
- No graphics APIs are supported (e.g. OpenGL, Vulkan etc.)
- No GPU to GPU P2P (either PCIe or NVLink) is supported
- CUDA applications treat a Compute Instance and its parent GPU Instance as a single CUDA device. See this section on device enumeration by CUDA
- CUDA IPC across GPU instances is not supported. CUDA IPC across Compute instances is supported
- CUDA debugging (e.g. using cuda-gdb) and memory/race checking (e.g. using cuda-memcheck or compute-sanitizer) is supported
- CUDA MPS is supported on top of MIG. The only limitation is that the maximum number of clients (48) is lowered proportionally to the Compute Instance size
- GPUDirect RDMA is supported when used from GPU Instances

Chapter 7. MIG Device Names

By default, a MIG device consists of a single “GPU Instance” and a single “Compute Instance”. The table below highlights a naming convention to refer to a MIG device by its GPU Instance's compute slice count and its total memory in GB (rather than just its memory slice count).

When only a single CI is created (that consumes the entire compute capacity of the GI), then the CI sizing is implied in the device name.

Figure 9. MIG Device Name



> Note:
>
> The description below shows the profile names on the A100-SXM4-40GB product. For A100-SXM4-80GB, the profile names will change according to the memory proportion - for example, `1g.10gb`, `2g.20gb`, `3g.40gb`, `4g.40gb`, `7g.80gb` respectively.

Table 4. Device names when using a single CI

Memory	20gb			10gb		5gb
GPU Instance	3g			2g		1g
Compute Instance	3c			2c		1c
MIG Device	3g.20gb			2g.10gb		1g.5gb
	GPC	GPC	GPC	GPC	GPC	GPC

Each GI can be further sub-divided into multiple CIs as required by users depending on their workloads. The table below highlights what the name of a MIG device would look like in this case. The example shown is for subdividing a 3g.20gb device into a set of sub-devices with different Compute Instance slice counts.

Table 5. Device names when using multiple CIs

Memory	20gb			20gb		
GPU Instance	3g			3g		
Compute Instance	1c	1c	1c	2c		1c
MIG Device	1c.3g.20gb	1c.3g.20gb	1c.3g.20gb	2c.3g.20gb		1c.3g.20gb
	GPC	GPC	GPC	GPC	GPC	GPC

7.1. Device Enumeration

GPU Instances (GIs) and Compute Instances (CIs) are enumerated in the new `/proc` filesystem layout for MIG

```
$ ls -l /proc/driver/nvidia-caps/
-r--r--r-- 1 root root 0 Nov 21 21:22 mig-minors
-r--r--r-- 1 root root 0 Nov 21 21:22 nvlink-minors
-r--r--r-- 1 root root 0 Nov 21 21:22 sys-minors
```

The corresponding device nodes (in `mig-minors`) are created under `/dev/nvidia-caps`. Refer to the chapter on device nodes and capabilities for more information.

7.2. CUDA Device Enumeration

MIG supports running CUDA applications by specifying the CUDA device on which the application should be run. With CUDA 11/R450 and CUDA 12/R525, only enumeration of a single MIG instance is supported. In other words, regardless of how many MIG devices are created (or made available to a container), a single CUDA process can only enumerate a single MIG device.

CUDA applications treat a CI and its parent GI as a single CUDA device. CUDA is limited to use a single CI and will pick the first one available if several of them are visible. To summarize, there are two constraints:

1. CUDA can only enumerate a single compute instance
2. CUDA will not enumerate non-MIG GPU if any compute instance is enumerated on any other GPU

Note that these constraints may be relaxed in future NVIDIA driver releases for MIG.

`CUDA_VISIBLE_DEVICES` has been extended to add support for MIG. Depending on the driver versions being used, two formats are supported:

1. With drivers >= R470 (`470.42.01`+), each MIG device is assigned a GPU UUID starting with `MIG-<UUID>`.
2. With drivers < R470 (e.g. R450 and R460), each MIG device is enumerated by specifying the CI and the corresponding parent GI. The format follows this convention: `MIG-<GPU-UUID>/<GPU instance ID>/<compute instance ID>`.

Note:

With the R470 NVIDIA datacenter drivers (`470.42.01`+), the example below shows how MIG devices are assigned GPU UUIDs in an 8-GPU system with each GPU configured differently.

```
$ nvidia-smi -L

GPU 0: A100-SXM4-40GB (UUID: GPU-5d5ba0d6-d33d-2b2c-524d-9e3d8d2b8a77)
  MIG 1g.5gb      Device  0: (UUID: MIG-c6d4f1ef-42e4-5de3-91c7-45d71c87eb3f)
  MIG 1g.5gb      Device  1: (UUID: MIG-cba663e8-9bed-5b25-b243-5985ef7c9beb)
  MIG 1g.5gb      Device  2: (UUID: MIG-1e099852-3624-56c0-8064-c5db1211e44f)
  MIG 1g.5gb      Device  3: (UUID: MIG-8243111b-d4c4-587a-a96d-da04583b36e2)
  MIG 1g.5gb      Device  4: (UUID: MIG-169f1837-b996-59aa-9ed5-b0a3f99e88a6)
  MIG 1g.5gb      Device  5: (UUID: MIG-d5d0152c-e3f0-552c-abee-ebc0195e9f1d)
  MIG 1g.5gb      Device  6: (UUID: MIG-7df6b45c-a92d-5e09-8540-a6b389968c31)
GPU 1: A100-SXM4-40GB (UUID: GPU-0aa11ebd-627f-af3f-1a0d-4e1fd92fd7b0)
  MIG 2g.10gb     Device  0: (UUID: MIG-0c757cd7-e942-5726-a0b8-0e8fb7067135)
  MIG 2g.10gb     Device  1: (UUID: MIG-703fb6ed-3fa0-5e48-8e65-1c5bdcfe2202)
  MIG 2g.10gb     Device  2: (UUID: MIG-532453fc-0faa-5c3c-9709-a3fc2e76083d)
GPU 2: A100-SXM4-40GB (UUID: GPU-08279800-1cbe-a71d-f3e6-8f67e15ae54a)
  MIG 3g.20gb     Device  0: (UUID: MIG-aa232436-d5a6-5e39-b527-16f9b223cc46)
  MIG 3g.20gb     Device  1: (UUID: MIG-3b12da37-7fa2-596c-8655-62dab88f0b64)
GPU 3: A100-SXM4-40GB (UUID: GPU-71086aca-c858-d1e0-aae1-275bed1008b9)
  MIG 7g.40gb     Device  0: (UUID: MIG-3e209540-03e2-5edb-8798-51d4967218c9)
GPU 4: A100-SXM4-40GB (UUID: GPU-74fa9fb7-ccf6-8234-e597-7af8ace9a8f5)
  MIG 1c.3g.20gb  Device  0: (UUID: MIG-79c62632-04cc-574b-af7b-cb2e307120d8)
  MIG 1c.3g.20gb  Device  1: (UUID: MIG-4b3cc0fd-6876-50d7-a8ba-184a86e2b958)
  MIG 1c.3g.20gb  Device  2: (UUID: MIG-194837c7-0476-5b56-9c45-16bddc82e1cf)
  MIG 1c.3g.20gb  Device  3: (UUID: MIG-291820db-96a4-5463-8e7b-444c2d2e3dfa)
  MIG 1c.3g.20gb  Device  4: (UUID: MIG-5a97e28a-7809-5e93-abae-c3818c5ea801)
  MIG 1c.3g.20gb  Device  5: (UUID: MIG-3dfd5705-b18a-5a7c-bcee-d03a0ccb7a96)
GPU 5: A100-SXM4-40GB (UUID: GPU-3301e6dd-d38f-0eb5-4665-6c9659f320ff)
  MIG 4g.20gb     Device  0: (UUID: MIG-6d96b9f9-960e-5057-b5da-b8a35dc63aa8)
GPU 6: A100-SXM4-40GB (UUID: GPU-bb40ed7d-cbbb-d92c-50ac-24803cda52c5)
  MIG 1c.7g.40gb  Device  0: (UUID: MIG-66dd01d7-8cdb-5a13-a45d-c6eb0ee11810)
  MIG 2c.7g.40gb  Device  1: (UUID: MIG-03c649cb-e6ae-5284-8e94-4b1cf767e06c)
  MIG 3c.7g.40gb  Device  2: (UUID: MIG-8abf68e0-2808-525e-9133-ba81701ed6d3)
GPU 7: A100-SXM4-40GB (UUID: GPU-95fac899-e21a-0e44-b0fc-e4e3bf106feb)
  MIG 4g.20gb     Device  0: (UUID: MIG-219c765c-e07f-5b85-9c04-4afe174d83dd)
  MIG 2g.10gb     Device  1: (UUID: MIG-25884364-137e-52cc-a7e4-ecf3061c3ae1)
  MIG 1g.5gb      Device  2: (UUID: MIG-83e71a6c-f0c3-5dfc-8577-6e8b17885e1f)
```

Chapter 8. Supported MIG Profiles

This section provides an overview of the supported profiles and possible placements of the MIG profiles on supported GPUs.

8.1. A30 MIG Profiles

The following diagram shows the profiles supported on the NVIDIA A30:

Figure 10. Profiles on A30

Config	GPC Slice #0	GPC Slice #1	GPC Slice #2	GPC Slice #3	OFA	NVDEC	NVJPG	P2P	GPU Direct RDMA
1	4				1	4	1	No	Supported MemBW proportional to size of the instance
2	2		2		0	2+2	0	No	
3	2		1	1	0	2+1+1	0	No	
4	1	1	2		0	1+1+2	0	No	
5	1	1	1	1	0	1+1+1+1	0	No	

The table below shows the supported profiles on the A30-24GB product.

Table 6. GPU Instance Profiles on A30

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	L2 Cache Size	Copy Engines	Number of Instances Available
MIG 1g.6gb	1/4	1/4	0 NVDECs /0 JPEG /0 OFA	1/4	1	4
MIG 1g.6gb +me	1/4	1/4	1 NVDEC /1 JPEG /1 OFA	1/4	1	1 (A single 1g profile can include media extensions)
MIG 2g.12gb	2/4	2/4	2 NVDECs /0 JPEG /0 OFA	2/4	2	2

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	L2 Cache Size	Copy Engines	Number of Instances Available
MIG 2g.12gb +me	2/4	2/4	2 NVDECs /1 JPEG /1 OFA	2/4	2	1 (A single 2g profile can include media extensions)
MIG 4g.24gb	Full	4/4	4 NVDECs /1 JPEG /1 OFA	Full	4	1

Note:

The `1g.6gb+me` profile is only available starting with R470 drivers.

The `2g.12gb+me` profile is only available starting with R525 drivers.

8.2. A100 MIG Profiles

The following diagram shows the profiles supported on the NVIDIA A100:

Figure 11. Profiles on A100

Config	GPC Slice #0	GPC Slice #1	GPC Slice #2	GPC Slice #3	GPC Slice #4	GPC Slice #5	GPC Slice #6	OFA	NVDEC	NVJPG	P2P	GPU Direct RDMA
1	7							1	5	1	No	
2	4				3			0	2+2	0	No	Supported MemBW proportional to size of the instance
3	4				2		1	0	2+1+0	0	No	
4	4				1	1	1	0	2+0+0+0	0	No	
5	3			3				0	2+2	0	No	
6	3			2		1		0	2+1+0	0	No	
7	3			1	1	1		0	2+0+0+0	0	No	
8	2		2		3			0	1+1+2	0	No	
9	2		1	1	3			0	1+0+0+2	0	No	
10	1	1	2		3			0	0+0+1+2	0	No	
11	1	1	1	1	3			0	0+0+0+0+2	0	No	
12	2		2		2		1	0	1+1+1+0	0	No	
13	2		1	1	2		1	0	1+0+0+1+0	0	No	
14	1	1	2		2		1	0	0+0+1+1+0	0	No	
15	2		1	1	1	1	1	0	1+0+0+0+0	0	No	
16	1	1	2		1	1	1	0	0+0+1+0+0+0	0	No	
17	1	1	1	1	2		1	0	0+0+0+0+1+0	0	No	
18	1	1	1	1	1	2		0	0+0+0+0+0+1	0	No	
19	1	1	1	1	1	1	1	0	0+0+0+0+0+0+0	0	No	

The table below shows the supported profiles on the A100-SXM4-40GB product. For A100-SXM4-80GB, the profile names will change according to the memory proportion - for example, `1g.10gb`, `1g.10gb+me`, `1g.20gb`, `2g.20gb`, `3g.40gb`, `4g.40gb`, `7g.80gb` respectively.

Table 7. GPU Instance Profiles on A100

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	L2 Cache Size	Copy Engines	Number of Instances Available
MIG 1g.5gb	1/8	1/7	0 NVDECs /0 JPEG /0 OFA	1/8	1	7
MIG 1g.5gb +me	1/8	1/7	1 NVDEC /1 JPEG /1 OFA	1/8	1	1 (A single 1g profile can include media extensions)
MIG 1g.10gb	1/8	1/7	1 NVDECs /0 JPEG /0 OFA	1/8	1	4
MIG 2g.10gb	2/8	2/7	1 NVDECs /0 JPEG /0 OFA	2/8	2	3
MIG 3g.20gb	4/8	3/7	2 NVDECs /0 JPEG /0 OFA	4/8	3	2
MIG 4g.20gb	4/8	4/7	2 NVDECs /0 JPEG /0 OFA	4/8	4	1
MIG 7g.40gb	Full	7/7	5 NVDECs /1 JPEG /1 OFA	Full	7	1

Note:

The `1g.5gb+me` profile is only available starting with R470 drivers.

The `1g.10gb` profile is only available starting with R525 drivers.

8.3. H100 MIG Profiles

The following diagram shows the profiles supported on the NVIDIA H100:

Figure 12. Profiles on H100

Config	GPC Slice #0	GPC Slice #1	GPC Slice #2	GPC Slice #3	GPC Slice #4	GPC Slice #5	GPC Slice #6	OFA	NVDEC	NVJPG	P2P	GPU Direct RDMA
1	7							1	7	7	No	Supported MemBW proportional to size of the instance
2	4				3			0	4+3	4+3	No	
3	4				2		1	0	4+2+1	4+2+1	No	
4	4				1	1	1	0	4+1+1+1	4+1+1+1	No	
5	3			3				0	3+3	3+3	No	
6	3			2		1		0	3+2+1	3+2+1	No	
7	3			1	1	1		0	3+1+1+1	3+1+1+1	No	
8	2		2		3			0	2+2+3	2+2+3	No	
9	2		1	1	3			0	2+1+1+3	2+1+1+3	No	
10	1	1	2		3			0	1+1+2+3	1+1+2+3	No	
11	1	1	1	1	3			0	1+1+1+1+3	1+1+1+1+3	No	
12	2		2		2		1	0	2+2+2+2+1	2+2+2+2+1	No	
13	2		1	1	2		1	0	2+1+1+2+1	2+1+1+2+1	No	
14	1	1	2		2		1	0	1+1+2+2+1	1+1+2+2+1	No	
15	2		1	1	1	1	1	0	2+1+1+1+1+1	2+1+1+1+1	No	
16	1	1	2		1	1	1	0	1+1+2+1+1+1	1+1+2+1+1+1	No	
17	1	1	1	1	2		1	0	1+1+1+1+2+1	1+1+1+1+2+1	No	
18	1	1	1	1	1	2		0	1+1+1+1+1+2	1+1+1+1+1+2	No	
19	1	1	1	1	1	1	1	0	1+1+1+1+1+1+1	1+1+1+1+1+1+1	No	

The table below shows the supported profiles on the H100 80GB product (PCIe and SXM5).

Table 8. GPU Instance Profiles on H100

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	L2 Cache Size	Copy Engines	Number of Instances Available
MIG 1g.10gb	1/8	1/7	1 NVDECs /1 JPEG /0 OFA	1/8	1	7
MIG 1g.10gb +me	1/8	1/7	1 NVDEC /1 JPEG /1 OFA	1/8	1	1 (A single 1g profile can include media extensions)
MIG 1g.20gb	1/4	1/7	1 NVDECs /1 JPEG /0 OFA	1/8	1	4
MIG 2g.20gb	2/8	2/7	2 NVDECs /2 JPEG /0 OFA	2/8	2	3
MIG 3g.40gb	4/8	3/7	3 NVDECs /3 JPEG /0 OFA	4/8	3	2
MIG 4g.40gb	4/8	4/7	4 NVDECs /4 JPEG /0 OFA	4/8	4	1

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	L2 Cache Size	Copy Engines	Number of Instances Available
MIG 7g.80gb	Full	7/7	7 NVDECs /7 JPEG /1 OFA	Full	8	1

The table below shows the supported profiles on the H100 94GB product (PCIe and SXM5).

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	L2 Cache Size	Copy Engines	Number of Instances Available
MIG 1g.12gb	1/8	1/7	1 NVDECs /1 JPEG /0 OFA	1/8	1	7
MIG 1g.12gb +me	1/8	1/7	1 NVDEC /1 JPEG /1 OFA	1/8	1	1 (A single 1g profile can include media extensions)
MIG 1g.24gb	1/4	1/7	1 NVDECs /1 JPEG /0 OFA	1/8	1	4
MIG 2g.24gb	2/8	2/7	2 NVDECs /2 JPEG /0 OFA	2/8	2	3
MIG 3g.47gb	4/8	3/7	3 NVDECs /3 JPEG /0 OFA	4/8	3	2
MIG 4g.47gb	4/8	4/7	4 NVDECs /4 JPEG /0 OFA	4/8	4	1
MIG 7g.94gb	Full	7/7	7 NVDECs /7 JPEG /1 OFA	Full	8	1

The table below shows the supported profiles on the H100 96GB product (H100 on GH200).

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	L2 Cache Size	Copy Engines	Number of Instances Available
MIG 1g.12gb	1/8	1/7	1 NVDECs /1 JPEG /0 OFA	1/8	1	7
MIG 1g.12gb +me	1/8	1/7	1 NVDEC /1 JPEG /1 OFA	1/8	1	1 (A single 1g profile can include media extensions)
MIG 1g.24gb	1/4	1/7	1 NVDECs /1 JPEG /0 OFA	1/8	1	4
MIG 2g.24gb	2/8	2/7	2 NVDECs /2 JPEG /0 OFA	2/8	2	3
MIG 3g.48gb	4/8	3/7	3 NVDECs /3 JPEG /0 OFA	4/8	3	2
MIG 4g.48gb	4/8	4/7	4 NVDECs /4 JPEG /0 OFA	4/8	4	1
MIG 7g.96gb	Full	7/7	7 NVDECs /7 JPEG /1 OFA	Full	8	1

8.4. H200 MIG Profiles

The following diagram shows the profiles supported on the NVIDIA H200:

Figure 13. Profiles on H200

Config	GPC Slice #0	GPC Slice #1	GPC Slice #2	GPC Slice #3	GPC Slice #4	GPC Slice #5	GPC Slice #6	OFA	NVDEC	NVJPG	P2P	GPU Direct RDMA
1	7							1	7	7	No	
2	4				3			0	4+3	4+3	No	Supported MemBW proportional to size of the instance
3	4				2		1	0	4+2+1	4+2+1	No	
4	4				1	1	1	0	4+1+1+1	4+1+1+1	No	
5	3			3				0	3+3	3+3	No	
6	3			2		1		0	3+2+1	3+2+1	No	
7	3			1	1	1		0	3+1+1+1	3+1+1+1	No	
8	2		2		3			0	2+2+3	2+2+3	No	
9	2		1	1	3			0	2+1+1+3	2+1+1+3	No	
10	1	1	2		3			0	1+1+2+3	1+1+2+3	No	
11	1	1	1	1	3			0	1+1+1+1+3	1+1+1+1+3	No	
12	2		2		2		1	0	2+2+2+2+1	2+2+2+2+1	No	
13	2		1	1	2		1	0	2+1+1+2+1	2+1+1+2+1	No	
14	1	1	2		2		1	0	1+1+2+2+1	1+1+2+2+1	No	
15	2		1	1	1	1	1	0	2+1+1+1+1+1	2+1+1+1+1	No	
16	1	1	2		1	1	1	0	1+1+2+1+1+1	1+1+2+1+1+1	No	
17	1	1	1	1	2		1	0	1+1+1+1+2+1	1+1+1+1+2+1	No	
18	1	1	1	1	1	2		0	1+1+1+1+1+2	1+1+1+1+1+2	No	
19	1	1	1	1	1	1	1	0	1+1+1+1+1+1+1	1+1+1+1+1+1+1	No	

The table below shows the supported profiles on the H200 141GB product.

Table 9. GPU Instance Profiles on H200

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	L2 Cache Size	Copy Engines	Number of Instances Available
MIG 1g.18gb	1/8	1/7	1 NVDECs /1 JPEG /0 OFA	1/8	1	7
MIG 1g.18gb +me	1/8	1/7	1 NVDEC /1 JPEG /1 OFA	1/8	1	1 (A single 1g profile can include media extensions)
MIG 1g.35gb	1/4	1/7	1 NVDECs /1 JPEG /0 OFA	1/8	1	4
MIG 2g.35gb	2/8	2/7	2 NVDECs /2 JPEG /0 OFA	2/8	2	3
MIG 3g.71gb	4/8	3/7	3 NVDECs /3 JPEG /0 OFA	4/8	3	2
MIG 4g.71gb	4/8	4/7	4 NVDECs /4 JPEG /0 OFA	4/8	4	1

Profile Name	Fraction of Memory	Fraction of SMs	Hardware Units	L2 Cache Size	Copy Engines	Number of Instances Available
MIG 7g.141gb	Full	7/7	7 NVDECs /7 JPEG /1 OFA	Full	8	1

Chapter 9. Getting Started with MIG

9.1. Prerequisites

The following prerequisites and minimum software versions are recommended when using supported GPUs in MIG mode.

- MIG is supported only on GPUs and systems listed here
- It is recommended to install the latest NVIDIA datacenter driver. The minimum versions are provided below:
  - If using H100, then CUDA 12 and NVIDIA driver R525 (`>= 525.53`) or later
  - If using A100/A30, then CUDA 11 and NVIDIA driver R450 (`>= 450.80.02`) or later
- Linux operating system distributions supported by CUDA
- If running containers or using Kubernetes, then:
  - NVIDIA Container Toolkit (`nvidia-docker2`): v2.5.0 or later
  - NVIDIA K8s Device Plugin: v0.7.0 or later
  - NVIDIA gpu-feature-discovery: v0.2.0 or later

MIG can be managed programmatically using NVIDIA Management Library (NVML) APIs or its command-line-interface, `nvidia-smi`. Note that for brevity, some of the `nvidia-smi` output in the following examples may be cropped to showcase the relevant sections of interest.

For more information on the MIG commands, see the `nvidia-smi` man page or `nvidia-smi mig --help`. For information on the MIG management APIs, see the NVML header (`nvml.h`) included in the CUDA Toolkit packages (`cuda-nvml-dev-*`; installed under `/usr/local/cuda/include/nvml.h`) For automated tooling support with configuring MIG, refer to the NVIDIA MIG Partition Editor (or `mig-parted`) tools.

9.2. Enable MIG Mode

By default, MIG mode is not enabled on the GPU. For example, running nvidia-smi shows that MIG mode is disabled:

```
$ nvidia-smi -i 0
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.80.02    Driver Version: 450.80.02    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  A100-SXM4-40GB      Off  | 00000000:36:00.0 Off |                    0 |
| N/A   29C    P0    62W / 400W |      0MiB / 40537MiB |      6%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
```

MIG mode can be enabled on a per-GPU basis with the following command: `nvidia-smi -i <GPU IDs> -mig 1`. The GPUs can be selected using comma separated GPU indexes, PCI Bus Ids or UUIDs. If no GPU ID is specified, then MIG mode is applied to all the GPUs on the system.

When MIG is enabled on the GPU, depending on the GPU product, the driver will attempt to reset the GPU so that MIG mode can take effect.

```
$ sudo nvidia-smi -i 0 -mig 1
Enabled MIG Mode for GPU 00000000:36:00.0
All done.


$ nvidia-smi -i 0 --query-gpu=pci.bus_id,mig.mode.current --format=csv
pci.bus_id, mig.mode.current
00000000:36:00.0, Enabled
```

9.2.1. GPU Reset on Hopper+ GPUs

Starting with the Hopper generation of GPUs, enabling MIG mode no longer requires a GPU reset to take effect (and thus the driver does not attempt to reset the GPU in the background).

Note that MIG mode (`Disabled` or `Enabled` states) is only persistent as long as the driver is resident in the system (i.e. the kernel modules are loaded). MIG mode is no longer persistent across system reboots (there is no longer a status bit stored in the GPU InfoROM).

Thus, an unload and reload of the driver kernel modules will disable MIG mode.

9.2.2. GPU Reset on Ampere GPUs

On NVIDIA Ampere GPUs, when MIG mode is enabled, the driver will attempt to reset the GPU so that MIG mode can take effect.

Note that MIG mode (`Disabled` or `Enabled` states) is persistent across system reboots (there is a status bit stored in the GPU InfoROM). Thus MIG mode has to be explicitly disabled to return the GPU to its default state.

Note:

> If you are using MIG inside a VM with NVIDIA Ampere GPUs (A100 or A30) in passthrough, then you may need to reboot the VM to allow the GPU to be in MIG mode as in some cases, GPU reset is not allowed via the hypervisor for security reasons. This can be seen in the following example:
>
> ```
> $ sudo nvidia-smi -i 0 -mig 1
> Warning: MIG mode is in pending enable state for GPU 00000000:00:03.0:Not
>  Supported
> Reboot the system or try nvidia-smi --gpu-reset to make MIG mode effective on
>  GPU 00000000:00:03.0
> All done.
>
> $ sudo nvidia-smi --gpu-reset
> Resetting GPU 00000000:00:03.0 is not supported.
> ```

9.2.3. Driver Clients

In some cases, if you have agents on the system (e.g. monitoring agents) that use the GPU, then you may not be able to initiate a GPU reset. For example, on DGX systems, you may encounter the following message:

```
$ sudo nvidia-smi -i 0 -mig 1
Warning: MIG mode is in pending enable state for GPU 00000000:07:00.0:In use by
 another client
00000000:07:00.0 is currently being used by one or more other processes (e.g. CUDA
 application or a monitoring application such as another instance of nvidia-smi).
 Please first kill all processes using the device and retry the command or reboot
 the system to make MIG mode effective.
All done.
```

In this specific DGX example, you would have to stop the `nvsm` and `dcgm` services, enable MIG mode on the desired GPU and then restore the monitoring services:

```
$ sudo systemctl stop nvsm

$ sudo systemctl stop dcgm

$ sudo nvidia-smi -i 0 -mig 1
Enabled MIG Mode for GPU 00000000:07:00.0
All done.
```

The examples shown in the document use super-user privileges. As described in the Device Nodes section, granting read access to `mig/config` capabilities allows non-root users to manage instances once the GPU has been configured into MIG mode. The default file permissions on the `mig/config` file is shown below.

```
$ ls -l /proc/driver/nvidia/capabilities/*
/proc/driver/nvidia/capabilities/mig:
total 0
-r-------- 1 root root 0 May 24 16:10 config
-r--r--r-- 1 root root 0 May 24 16:10 monitor
```

9.3. List GPU Instance Profiles

The NVIDIA driver provides a number of profiles that users can opt-in for when configuring the MIG feature in A100. The profiles are the sizes and capabilities of the GPU instances that can be created by the user. The driver also provides information about the placements, which indicate the type and number of instances that can be created.

```
$ nvidia-smi mig -lgip
+-----------------------------------------------------------------------------+
| GPU instance profiles:                                                      |
| GPU   Name             ID    Instances   Memory     P2P    SM    DEC   ENC  |
|                              Free/Total   GiB              CE    JPEG  OFA  |
|=============================================================================|
|   0  MIG 1g.5gb        19     7/7        4.75       No     14     0     0   |
|                                                             1     0     0   |
+-----------------------------------------------------------------------------+
|   0  MIG 1g.5gb+me     20     1/1        4.75       No     14     1     0   |
|                                                             1     1     1   |
+-----------------------------------------------------------------------------+
|   0  MIG 1g.10gb       15     4/4        9.62       No     14     1     0   |
|                                                             1     0     0   |
+-----------------------------------------------------------------------------+
|   0  MIG 2g.10gb       14     3/3        9.62       No     28     1     0   |
|                                                             2     0     0   |
+-----------------------------------------------------------------------------+
|   0  MIG 3g.20gb        9     2/2        19.50      No     42     2     0   |
|                                                             3     0     0   |
+-----------------------------------------------------------------------------+
|   0  MIG 4g.20gb        5     1/1        19.50      No     56     2     0   |
|                                                             4     0     0   |
+-----------------------------------------------------------------------------+
|   0  MIG 7g.40gb        0     1/1        39.25      No     98     5     0   |
|                                                             7     1     1   |
+-----------------------------------------------------------------------------+
```

List the possible placements available using the following command. The syntax of the placement is `{<index>}:<GPU Slice Count>` and shows the placement of the instances on the GPU. The placement index shown indicates how the profiles are mapped on the GPU as shown in the supported profiles tables.

```
$ nvidia-smi mig -lgipp
GPU  0 Profile ID 19 Placements: {0,1,2,3,4,5,6}:1
GPU  0 Profile ID 20 Placements: {0,1,2,3,4,5,6}:1
GPU  0 Profile ID 15 Placements: {0,2,4,6}:2
GPU  0 Profile ID 14 Placements: {0,2,4}:2
GPU  0 Profile ID  9 Placements: {0,4}:4
GPU  0 Profile ID  5 Placement : {0}:4
GPU  0 Profile ID  0 Placement : {0}:8
```

The command shows that the user can create two instances of type `3g.20gb` (profile ID 9) or seven instances of `1g.5gb` (profile ID 19).

9.4. Creating GPU Instances

Before starting to use MIG, the user needs to create GPU instances using the `-cgi` option. One of three options can be used to specify the instance profiles to be created:

1. Profile ID (e.g. 9, 14, 5)
2. Short name of the profile (e.g. `3g.20gb`
3. Full profile name of the instance (e.g. `MIG 3g.20gb`)

Once the GPU instances are created, one needs to create the corresponding Compute Instances (CI). By using the `-C` option, `nvidia-smi` creates these instances.

Note:

Without creating GPU instances (and corresponding compute instances), CUDA workloads cannot be run on the GPU. In other words, simply enabling MIG mode on the GPU is not sufficient. Also note that, the created MIG devices are not persistent across system reboots. Thus, the user or system administrator needs to recreate the desired MIG configurations if the GPU or system is reset. For automated tooling support for this purpose, refer to the NVIDIA MIG Partition Editor (or `mig-parted`) tool, including creating a `systemd` service that could recreate the MIG geometry at system startup.

The following example shows how the user can create GPU instances (and corresponding compute instances). In this example, the user can create two GPU instances (of type `3g.20gb`), with each GPU instance having half of the available compute and memory capacity. In this example, we purposefully use profile ID and short profile name to showcase how either option can be used:

```
$ sudo nvidia-smi mig -cgi 9,3g.20gb -C
Successfully created GPU instance ID  2 on GPU  0 using profile MIG 3g.20gb (ID  9)
Successfully created compute instance ID  0 on GPU  0 GPU instance ID  2 using
 profile MIG 3g.20gb (ID  2)
Successfully created GPU instance ID  1 on GPU  0 using profile MIG 3g.20gb (ID  9)
Successfully created compute instance ID  0 on GPU  0 GPU instance ID  1 using
 profile MIG 3g.20gb (ID  2)
```

Now list the available GPU instances:

```
$ sudo nvidia-smi mig -lgi
+----------------------------------------------------+
| GPU instances:                                     |
| GPU   Name            Profile  Instance   Placement  |
|                         ID       ID       Start:Size |
|====================================================|
|   0  MIG 3g.20gb       9        1          4:4     |
+----------------------------------------------------+
|   0  MIG 3g.20gb       9        2          0:4     |
+----------------------------------------------------+
```

Now verify that the GIs and corresponding CIs are created:

```
$ nvidia-smi
+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |                      | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  0    1   0   0  |     11MiB / 20224MiB | 42      0 |  3   0    2    0    0 |
+------------------+----------------------+-----------+-----------------------+
|  0    2   0   1  |     11MiB / 20096MiB | 42      0 |  3   0    2    0    0 |
+------------------+----------------------+-----------+-----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI         PID   Type   Process name                GPU Memory |
|        ID   ID                                                  Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Instance Geometry

As described in the section on Partitioning, the NVIDIA driver APIs provide a number of available GPU Instance profiles that can be chosen by the user.

If a mixed geometry of the profiles is specified by the user, then the NVIDIA driver chooses the placement of the various profiles. This can be seen in the following examples.

Example 1: Creation of a 4-2-1 geometry. After the instances are created, the placement of the profiles can be observed:

```
$ sudo nvidia-smi mig -cgi 19,14,5
Successfully created GPU instance ID 13 on GPU  0 using profile MIG 1g.5gb (ID 19)
Successfully created GPU instance ID  5 on GPU  0 using profile MIG 2g.10gb (ID 14)
Successfully created GPU instance ID  1 on GPU  0 using profile MIG 4g.20gb (ID  5)


$ sudo nvidia-smi mig -lgi
+----------------------------------------------------+
| GPU instances:                                     |
| GPU   Name          Profile  Instance   Placement  |
|                       ID       ID       Start:Size |
|====================================================|
|   0  MIG 1g.5gb       19       13          6:1     |
+----------------------------------------------------+
|   0  MIG 2g.10gb      14        5          4:2     |
+----------------------------------------------------+
|   0  MIG 4g.20gb       5        1          0:4     |
+----------------------------------------------------+
```

Example 2: Creation of a 3-2-1-1 geometry.

Note:

> Due to a known issue with the APIs, the profile ID `9` or `3g.20gb` must be specified first in order. Not doing so, will result in the following error.
>
> ```
> $ sudo nvidia-smi mig -cgi 19,19,14,9
> Successfully created GPU instance ID 13 on GPU  0 using profile MIG 1g.5gb (ID
>  19)
> Successfully created GPU instance ID 11 on GPU  0 using profile MIG 1g.5gb (ID
>  19)
> Successfully created GPU instance ID  3 on GPU  0 using profile MIG 2g.10gb
>  (ID 14)
> Unable to create a GPU instance on GPU  0 using profile 9: Insufficient
>  Resources
> Failed to create GPU instances: Insufficient Resources
> ```

Specify the correct order for the `3g.20gb` profile. The remaining combinations of the profiles do not have this requirement.

```
$ sudo nvidia-smi mig -cgi 9,19,14,19
Successfully created GPU instance ID  2 on GPU  0 using profile MIG 3g.20gb (ID  9)
Successfully created GPU instance ID  7 on GPU  0 using profile MIG 1g.5gb (ID 19)
Successfully created GPU instance ID  4 on GPU  0 using profile MIG 2g.10gb (ID 14)
Successfully created GPU instance ID  8 on GPU  0 using profile MIG 1g.5gb (ID 19)

$ sudo nvidia-smi mig -lgi
+----------------------------------------------------+
| GPU instances:                                     |
| GPU   Name          Profile  Instance   Placement  |
|                       ID       ID       Start:Size |
|====================================================|
|   0  MIG 1g.5gb       19        7          0:1     |
+----------------------------------------------------+
|   0  MIG 1g.5gb       19        8          1:1     |
+----------------------------------------------------+
|   0  MIG 2g.10gb      14        4          2:2     |
+----------------------------------------------------+
|   0  MIG 3g.20gb       9        2          4:4     |
+----------------------------------------------------+
```

Example 3: Creation of a 2-1-1-1-1-1 geometry:

```
$ sudo nvidia-smi mig -cgi 14,19,19,19,19,19
Successfully created GPU instance ID  5 on GPU  0 using profile MIG 2g.10gb (ID 14)
Successfully created GPU instance ID 13 on GPU  0 using profile MIG 1g.5gb (ID 19)
Successfully created GPU instance ID  7 on GPU  0 using profile MIG 1g.5gb (ID 19)
Successfully created GPU instance ID  8 on GPU  0 using profile MIG 1g.5gb (ID 19)
Successfully created GPU instance ID  9 on GPU  0 using profile MIG 1g.5gb (ID 19)
Successfully created GPU instance ID 10 on GPU  0 using profile MIG 1g.5gb (ID 19)

$ sudo nvidia-smi mig -lgi
+----------------------------------------------------+
| GPU instances:                                     |
| GPU   Name          Profile  Instance   Placement  |
|                       ID       ID       Start:Size |
|====================================================|
|   0  MIG 1g.5gb       19        7          0:1     |
+----------------------------------------------------+
```

```
|   0  MIG 1g.5gb       19         8            1:1      |
+--------------------------------------------------------+
|   0  MIG 1g.5gb       19         9            2:1      |
+--------------------------------------------------------+
|   0  MIG 1g.5gb       19        10            3:1      |
+--------------------------------------------------------+
|   0  MIG 1g.5gb       19        13            6:1      |
+--------------------------------------------------------+
|   0  MIG 2g.10gb      14         5            4:2      |
+--------------------------------------------------------+
```

9.5. Running CUDA Applications on Bare-Metal

9.5.1. GPU Instances

The following example shows how two CUDA applications can be run in parallel on two different GPU instances. In this example, the BlackScholes CUDA sample is run simultaneously on the two GIs created on the A100.

```
$ nvidia-smi -L
GPU 0: A100-SXM4-40GB (UUID: GPU-e86cb44c-6756-fd30-cd4a-1e6da3caf9b0)
  MIG 3g.20gb Device 0: (UUID: MIG-c7384736-a75d-5afc-978f-d2f1294409fd)
  MIG 3g.20gb Device 1: (UUID: MIG-a28ad590-3fda-56dd-84fc-0a0b96edc58d)


$ CUDA_VISIBLE_DEVICES=MIG-c7384736-a75d-5afc-978f-d2f1294409fd ./BlackScholes &
$ CUDA_VISIBLE_DEVICES=MIG-a28ad590-3fda-56dd-84fc-0a0b96edc58d ./BlackScholes &
```

Now verify the two CUDA applications are running on two separate GPU instances:

```
$ nvidia-smi
+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |                      | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  0    1   0   0  |    268MiB / 20224MiB | 42      0 |  3   0    2    0    0 |
+------------------+----------------------+-----------+-----------------------+
|  0    2   0   1  |    268MiB / 20096MiB | 42      0 |  3   0    2    0    0 |
+------------------+----------------------+-----------+-----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI          PID   Type   Process name                GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0    1    0       58866      C   ./BlackScholes                  253MiB |
|    0    2    0       58856      C   ./BlackScholes                  253MiB |
+-----------------------------------------------------------------------------+
```

GPU Utilization Metrics

NVML (and `nvidia-smi`) does not support attribution of utilization metrics to MIG devices. From the previous example, the utilization is displayed as `N/A` when running CUDA programs:

```
$ nvidia-smi

+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |           BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  0    1   0   0  |    268MiB / 20096MiB | 42      0 |  3   0    2    0    0 |
|                  |      4MiB / 32767MiB |           |                       |
+------------------+----------------------+-----------+-----------------------+
|  0    2   0   1  |    268MiB / 20096MiB | 42      0 |  3   0    2    0    0 |
|                  |      4MiB / 32767MiB |           |                       |
+------------------+----------------------+-----------+-----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                 GPU Memory |
|        ID   ID                                                  Usage      |
|=============================================================================|
|    0    1    0       6217      C   ...inux/release/BlackScholes    253MiB |
|    0    2    0       6223      C   ...inux/release/BlackScholes    253MiB |
+-----------------------------------------------------------------------------+
```

For monitoring MIG devices on MIG capable GPUs such as the A100, including attribution of GPU metrics (including utilization and other profiling metrics), it is recommended to use NVIDIA DCGM v2.0.13 or later. See the Profiling Metrics section in the DCGM User Guide for more details on getting started.

9.5.2. Compute Instances

As explained earlier in this document, a further level of concurrency can be achieved by using Compute Instances (CIs). The following example shows how 3 CUDA processes (BlackScholes CUDA sample) can be run on the same GI.

First, list the available CI profiles available using our prior configuration of creating 2 GIs on the A100.

```
$ sudo nvidia-smi mig -lcip -gi 1
+--------------------------------------------------------------------------------------+
| Compute instance profiles:                                                           |
| GPU     GPU       Name             Profile  Instances   Exclusive       Shared       |
|       Instance                       ID     Free/Total     SM       DEC   ENC   OFA  |
|         ID                                                          CE    JPEG      |
|======================================================================================|
|   0      1       MIG 1c.3g.20gb       0      0/3           14        2     0     0   |
|                                                                      3     0         |
+--------------------------------------------------------------------------------------+
|   0      1       MIG 2c.3g.20gb       1      0/1           28        2     0     0   |
|                                                                      3     0         |
```

```
+-------------------------------------------------------------------------------------+
|   0      1       MIG 3g.20gb         2*     0/1          42       2     0     0    |
|                                                                   3     0          |
+-------------------------------------------------------------------------------------+
```

Create 3 CIs, each of type 1c compute capacity (profile ID 0) on the first GI.

```
$ sudo nvidia-smi mig -cci 0,0,0 -gi 1
Successfully created compute instance on GPU  0 GPU instance ID  1 using profile MIG
 1c.3g.20gb (ID  0)
Successfully created compute instance on GPU  0 GPU instance ID  1 using profile MIG
 1c.3g.20gb (ID  0)
Successfully created compute instance on GPU  0 GPU instance ID  1 using profile MIG
 1c.3g.20gb (ID  0)
```

Using nvidia-smi, the following CIs are now created on GI 1.

```
$ sudo nvidia-smi mig -lci -gi 1
+-------------------------------------------------------+
| Compute instances:                                    |
| GPU     GPU       Name             Profile   Instance |
|       Instance                       ID        ID     |
|         ID                                            |
|=======================================================|
|   0      1       MIG 1c.3g.20gb       0         0     |
+-------------------------------------------------------+
|   0      1       MIG 1c.3g.20gb       0         1     |
+-------------------------------------------------------+
|   0      1       MIG 1c.3g.20gb       0         2     |
+-------------------------------------------------------+
```

And the GIs and CIs created on the A100 are now enumerated by the driver:

```
$ nvidia-smi
+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |                      | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  0    1   0   0  |     11MiB / 20224MiB | 14      0 |  3   0    2    0    0 |
+------------------+                      +-----------+-----------------------+
|  0    1   1   1  |                      | 14      0 |  3   0    2    0    0 |
+------------------+                      +-----------+-----------------------+
|  0    1   2   2  |                      | 14      0 |  3   0    2    0    0 |
+------------------+----------------------+-----------+-----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                 GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Now, three BlackScholes applications can be created and run in parallel:

```
$ CUDA_VISIBLE_DEVICES=MIG-c7384736-a75d-5afc-978f-d2f1294409fd ./BlackScholes &
$ CUDA_VISIBLE_DEVICES=MIG-c376546e-7559-5610-9721-124e8dbb1bc8 ./BlackScholes &
$ CUDA_VISIBLE_DEVICES=MIG-928edfb0-898f-53bd-bf24-c7e5d08a6852 ./BlackScholes &
```

And seen using `nvidia-smi` as running processes on the three CIs:

```
$ nvidia-smi
+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |                      | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  0    1   0   0  |    476MiB / 20224MiB | 14      0 |  3   0    2    0    0 |
+------------------+                      +-----------+-----------------------+
|  0    1   1   1  |                      | 14      0 |  3   0    2    0    0 |
+------------------+                      +-----------+-----------------------+
|  0    1   2   2  |                      | 14      0 |  3   0    2    0    0 |
+------------------+----------------------+-----------+-----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0    1    0      59785      C   ./BlackScholes                    153MiB |
|    0    1    1      59796      C   ./BlackScholes                    153MiB |
|    0    1    2      59885      C   ./BlackScholes                    153MiB |
+-----------------------------------------------------------------------------+
```

9.6. Destroying GPU Instances

Once the GPU is in MIG mode, GIs and CIs can be configured dynamically. The following example shows how the CIs and GIs created in the previous examples can be destroyed.

Note:

If the intention is to destroy all the CIs and GIs, then this can be accomplished with the following commands:

```
$ sudo nvidia-smi mig -dci && sudo nvidia-smi mig -dgi
Successfully destroyed compute instance ID  0 from GPU  0 GPU instance ID  1
Successfully destroyed compute instance ID  1 from GPU  0 GPU instance ID  1
Successfully destroyed compute instance ID  2 from GPU  0 GPU instance ID  1
Successfully destroyed GPU instance ID  1 from GPU  0
Successfully destroyed GPU instance ID  2 from GPU  0
```

In this example, we delete the specific CIs created under GI 1.

```
$ sudo nvidia-smi mig -dci -ci 0,1,2 -gi 1
Successfully destroyed compute instance ID  0 from GPU  0 GPU instance ID  1
Successfully destroyed compute instance ID  1 from GPU  0 GPU instance ID  1
Successfully destroyed compute instance ID  2 from GPU  0 GPU instance ID  1
```

It can be verified that the CI devices have now been torn down on the GPU:

```
$ nvidia-smi
+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |                      | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  No MIG devices found                                                       |
+-----------------------------------------------------------------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                 GPU Memory |
|        ID   ID                                                  Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Now the GIs have to be deleted:

```
$ sudo nvidia-smi mig -dgi
Successfully destroyed GPU instance ID  1 from GPU  0
Successfully destroyed GPU instance ID  2 from GPU  0
```

9.7. Monitoring MIG Devices

For monitoring MIG devices on including attribution of GPU metrics (including utilization and other profiling metrics), it is recommended to use NVIDIA DCGM v3 or later. See the Profiling Metrics section in the DCGM User Guide for more details on getting started.

> Note:
>
> On Ampere GPUs (A100 or A30), NVML (and `nvidia-smi`) does not support attribution of utilization metrics to MIG devices. From the previous example, the utilization is displayed as `N/A` when running CUDA programs:

```
$ nvidia-smi

+-----------------------------------------------------------------------------
+
| MIG devices:
 |
+------------------+----------------------+-----------+-----------------------
+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared
 |
|      ID  ID  Dev |           BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA
 JPG|
```

```
|                  |                      |        ECC|
  |
|==================+======================+===========
+=======================|
|  0    1    0    0  |    268MiB / 20096MiB | 42      0 |  3    0    2    0    0
  |
|                  |      4MiB / 32767MiB |           |
  |
+------------------+----------------------+-----------+-----------------------
+
|  0    2    0    1  |    268MiB / 20096MiB | 42      0 |  3    0    2    0    0
  |
|                  |      4MiB / 32767MiB |           |
  |
+------------------+----------------------+-----------+-----------------------
+

+-----------------------------------------------------------------------------
+
| Processes:
  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory
  |
|        ID   ID                                                   Usage
  |
|
=============================================================================|
|    0    1    0       6217      C   ...inux/release/BlackScholes      253MiB
  |
|    0    2    0       6223      C   ...inux/release/BlackScholes      253MiB
  |
+-----------------------------------------------------------------------------
+
```

9.8. MIG with CUDA MPS

As described in the section on CUDA concurrency mechanisms, CUDA Multi-Process Service (MPS) enables co-operative multi-process CUDA applications to be processed concurrently on the GPU. MPS and MIG can work together, potentially achieving even higher levels of utilization for certain workloads.

Refer to the MPS documentation to understand the architecture and provisioning sequence for MPS.

In the following sections, we will walk through an example of running MPS on MIG devices.

Workflow

In summary, the workflow for running with MPS is as follows:

- Configure the desired MIG geometry on the GPU.
- Setup the `CUDA_MPS_PIPE_DIRECTORY` variable to point to unique directories so that the multiple MPS servers and clients can communicate with each other using named pipes and Unix domain sockets.

- Launch the application by specifying the MIG device using `CUDA_VISIBLE_DEVICES`.

> Note:
>
> The MPS documentation recommends setting up `EXCLUSIVE_PROCESS` mode to ensure that a single MPS server is using the GPU. However, this mode is not supported when the GPU is in MIG mode as we use multiple MPS servers (one per MIG GPU instance).

Configure GPU Instances

Follow the steps outlined in the previous sections to configure the desired MIG geometry on the GPU. For this example, we configure the GPU into a `3g.20gb`,`3g.2gb` geometry:

```
$ nvidia-smi

+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.73.01    Driver Version: 460.73.01    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  A100-PCIE-40GB      On   | 00000000:65:00.0 Off |                   On |
| N/A   37C    P0    66W / 250W |    581MiB / 40536MiB |     N/A      Default |
|                               |                      |              Enabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |           BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  0    1   0   0  |    290MiB / 20096MiB | 42      0 |  3   0    2    0    0 |
|                  |      8MiB / 32767MiB |           |                       |
+------------------+----------------------+-----------+-----------------------+
|  0    2   0   1  |    290MiB / 20096MiB | 42      0 |  3   0    2    0    0 |
|                  |      8MiB / 32767MiB |           |                       |
+------------------+----------------------+-----------+-----------------------+
```

Setup the MPS Control Daemons

In this step, we start an MPS control daemon (with admin privileges) and ensure we use a different socket for each daemon:

```
export CUDA_MPS_PIPE_DIRECTORY=/tmp/<MIG_UUID>
mkdir -p $CUDA_MPS_PIPE_DIRECTORY

CUDA_VISIBLE_DEVICES=<MIG_UUID> \
CUDA_MPS_PIPE_DIRECTORY=/tmp/<MIG_UUID> \
nvidia-cuda-mps-control -d
```

Launch the Application

Now we can launch the application by specifying the desired MIG device using `CUDA_VISIBLE_DEVICES`:

```
CUDA_VISIBLE_DEVICES=<MIG_UUID> \
   my-cuda-app
```

A Complete Example

We now provide a script below where we attempt to run the `BlackScholes` from before on the two MIG devices created on the GPU:

```
#!/usr/bin/env bash

set -euo pipefail

#GPU 0: A100-PCIE-40GB (UUID: GPU-63feeb45-94c6-b9cb-78ea-98e9b7a5be6b)
#  MIG 3g.20gb Device 0: (UUID: MIG-GPU-63feeb45-94c6-b9cb-78ea-98e9b7a5be6b/1/0)
#  MIG 3g.20gb Device 1: (UUID: MIG-GPU-63feeb45-94c6-b9cb-78ea-98e9b7a5be6b/2/0)

GPU_UUID=GPU-63feeb45-94c6-b9cb-78ea-98e9b7a5be6b
for i in MIG-$GPU_UUID/1/0 MIG-$GPU_UUID/2/0; do

   # set the environment variable on each MPS
   # control daemon and use different socket for each MIG instance
   export CUDA_MPS_PIPE_DIRECTORY=/tmp/$i
   mkdir -p $CUDA_MPS_PIPE_DIRECTORY
   sudo CUDA_VISIBLE_DEVICES=$i \
        CUDA_MPS_PIPE_DIRECTORY=/tmp/$i \
        nvidia-cuda-mps-control -d

   # now launch the job on the specific MIG device
   # and select the appropriate MPS server on the device
   CUDA_MPS_PIPE_DIRECTORY=/tmp/$i \
   CUDA_VISIBLE_DEVICES=$i \
   ./bin/BlackScholes &
done
```

When running this script, we can observe the two MPS servers on each MIG device and the corresponding CUDA program started as an MPS client when using `nvidia-smi`:

```
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0    1    0      46781    M+C   ./bin/BlackScholes                251MiB |
|    0    1    0      46784      C   nvidia-cuda-mps-server             29MiB |
|    0    2    0      46797    M+C   ./bin/BlackScholes                251MiB |
|    0    2    0      46798      C   nvidia-cuda-mps-server             29MiB |
+-----------------------------------------------------------------------------+
```

9.9. Running CUDA Applications as Containers

NVIDIA Container Toolkit has been enhanced to provide support for MIG devices, allowing users to run GPU containers with runtimes such as Docker. This section provides an overview of running Docker containers on A100 with MIG.

9.9.1. Install Docker

Many Linux distributions may come with Docker-CE pre-installed. If not, use the Docker installation script to install Docker.

```
$ curl https://get.docker.com | sh \
    && sudo systemctl start docker \
    && sudo systemctl enable docker
```

9.9.2. Install NVIDIA Container Toolkit

Now install the NVIDIA Container Toolkit (previously known as `nvidia-docker2`). MIG support is available starting with v2.3 of `nvidia-docker2` (or v1.1.1 of the `nvidia-container-toolkit` package).

To get access to the `/dev nvidia capabilities`, it is recommended to use at least v2.5.0 of `nvidia-docker2`. See the Installation Guide for more information.

For brevity, the installation instructions provided here are for Ubuntu 18.04 LTS. Refer to the NVIDIA Container Toolkit page for instructions on other Linux distributions.

Setup the repository and the GPG key:

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
      && curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor
 -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \
      && curl -s -L https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-
container.list | \
            sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-
keyring.gpg] https://#g' | \
            sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

Install the NVIDIA Container Toolkit packages (and their dependencies):

```
$ sudo apt-get install -y nvidia-docker2 \
    && sudo systemctl restart docker
```

9.9.3. Running Containers

To run containers on specific MIG devices - whether these are GIs or specific underlying CIs, then the `NVIDIA_VISIBLE_DEVICES` variable (or the `--gpus` option with Docker 19.03+) can be used.

`NVIDIA_VISIBLE_DEVICES` supports the following formats to specify MIG devices:

1. `MIG-<GPU-UUID>/<GPU instance ID>/<compute instance ID>` when using R450 and R460 drivers or `MIG-<UUID>` starting with R470 drivers.
2. `GPUDeviceIndex>:<MIGDeviceIndex>`

If using Docker 19.03, the `--gpus` option can be used to specify MIG devices by using the following format: `'"device=MIG-device"'`, where MIG-device can follow either of the format specified above for `NVIDIA_VISIBLE_DEVICES`.

The following example shows running `nvidia-smi` from within a CUDA container using both formats. As can be seen in the example, only one MIG device as chosen is visible to the container when using either format.

```
$ sudo docker run --runtime=nvidia \
    -e NVIDIA_VISIBLE_DEVICES=MIG-c7384736-a75d-5afc-978f-d2f1294409fd \
    nvidia/cuda nvidia-smi

+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |                      | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  0    1   0   0  |     11MiB / 20224MiB | 42      0 |  3   0    2    0    0 |
+------------------+----------------------+-----------+-----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+


# For Docker versions < 19.03
$ sudo docker run --runtime=nvidia \
    -e NVIDIA_VISIBLE_DEVICES="0:0" \
    nvidia/cuda nvidia-smi -L
GPU 0: A100-SXM4-40GB (UUID: GPU-e86cb44c-6756-fd30-cd4a-1e6da3caf9b0)
  MIG 3g.20gb Device 0: (UUID: MIG-c7384736-a75d-5afc-978f-d2f1294409fd)


# For Docker versions >= 19.03
$ sudo docker run --gpus '"device=0:0"' \
    nvidia/cuda nvidia-smi -L
GPU 0: A100-SXM4-40GB (UUID: GPU-e86cb44c-6756-fd30-cd4a-1e6da3caf9b0)
  MIG 3g.20gb Device 0: (UUID: MIG-c7384736-a75d-5afc-978f-d2f1294409fd)
```

A more complex example is to run a TensorFlow container to do a training run using GPUs on the MNIST dataset. This is shown below:

```
$ sudo docker run --gpus '"device=0:1"' \
    nvcr.io/nvidia/pytorch:20.11-py3 \
    /bin/bash -c 'cd /opt/pytorch/examples/upstream/mnist && python main.py'

=============
== PyTorch ==
=============

NVIDIA Release 20.11 (build 17345815)
PyTorch Version 1.8.0a0+17f8c32

Container image Copyright (c) 2020, NVIDIA CORPORATION.  All rights reserved.

Copyright (c) 2014-2020 Facebook Inc.
Copyright (c) 2011-2014 Idiap Research Institute (Ronan Collobert)
Copyright (c) 2012-2014 Deepmind Technologies    (Koray Kavukcuoglu)
Copyright (c) 2011-2012 NEC Laboratories America (Koray Kavukcuoglu)
Copyright (c) 2011-2013 NYU                      (Clement Farabet)
Copyright (c) 2006-2010 NEC Laboratories America (Ronan Collobert, Leon Bottou, Iain
 Melvin, Jason Weston)
Copyright (c) 2006      Idiap Research Institute (Samy Bengio)
Copyright (c) 2001-2004 Idiap Research Institute (Ronan Collobert, Samy Bengio,
 Johnny Mariethoz)
Copyright (c) 2015      Google Inc.
Copyright (c) 2015      Yangqing Jia
Copyright (c) 2013-2016 The Caffe contributors
All rights reserved.

NVIDIA Deep Learning Profiler (dlprof) Copyright (c) 2020, NVIDIA CORPORATION.  All
 rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying
 project or file.

NOTE: Legacy NVIDIA Driver detected.  Compatibility mode ENABLED.

9920512it [00:01, 7880654.53it/s]
32768it [00:00, 129950.31it/s]
1654784it [00:00, 2353765.88it/s]
8192it [00:00, 41020.33it/s]
/opt/conda/lib/python3.6/site-packages/torchvision/datasets/mnist.py:480:
 UserWarning: The given NumPy array is not writeable, and PyTorch does not support
 non-writeable tensors. This means you can write to the underlying (supposedly
 non-writeable) NumPy array using the tensor. You may want to copy the array to
 protect its data or make it writeable before converting it to a tensor. This type
 of warning will be suppressed for the rest of this program. (Triggered internally
 at  ../torch/csrc/utils/tensor_numpy.cpp:141.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../data/
MNIST/raw/train-images-idx3-ubyte.gz
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/
MNIST/raw/train-labels-idx1-ubyte.gz
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../data/
MNIST/raw/t10k-images-idx3-ubyte.gz
Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../data/
MNIST/raw/t10k-labels-idx1-ubyte.gz
Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw
Processing...
```

```
Done!
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.320747
Train Epoch: 1 [640/60000 (1%)] Loss: 1.278727
```

9.10. MIG with Kubernetes

MIG support in Kubernetes is available starting with v0.7.0 of the NVIDIA Device Plugin for Kubernetes. Visit the documentation on getting started with MIG and Kubernetes.

9.11. MIG with Slurm

Slurm is a workload manager that is widely used at high performance computing centers such as government labs, universities.

Starting with 21.08, Slurm supports the usage of MIG devices. Refer to the official documentation on getting started.

Chapter 10. Device Nodes and Capabilities

Currently, the NVIDIA kernel driver exposes its interfaces through a few system-wide device nodes. Each physical GPU is represented by its own device node - e.g. nvidia0, nvidia1 etc. This is shown below for a 2-GPU system.

```
/dev
├── nvidiactl
├── nvidia-modeset
├── nvidia-uvm
├── nvidia-uvm-tools
├── nvidia-nvswitchctl
├── nvidia0
└── nvidia1
```

Starting with CUDA 11/R450, a new abstraction known as `nvidia-capabilities` has been introduced. The idea being that access to a specific *capability* is required to perform certain actions through the driver. If a user has access to the *capability*, the action will be carried out. If a user does not have access to the *capability*, the action will fail. The one exception being if you are the root-user (or any user with `CAP_SYS_ADMIN` privileges). With `CAP_SYS_ADMIN` privileges, you implicitly have access to all `nvidia-capabilities`.

For example, the `mig-config` capability allows one to create and destroy MIG instances on any MIG-capable GPU (e.g. the A100 GPU). Without this capability, all attempts to create or destroy a MIG instance will fail. Likewise, the `fabric-mgmt` capability allows one to run the Fabric Manager as a non-root but privileged daemon. Without this capability, all attempts to launch the Fabric Manager as a non-root user will fail.

The following sections walk through the system level interface for managing these new `nvidia-capabilities`, including the steps necessary to grant and revoke access to them.

System Level Interface

There are two different system-level interfaces available to work with `nvidia-capabilities`. The first is via `/dev` and the second is via `/proc`. The `/proc` based interface relies on user-permissions and mount namespaces to limit access to a particular capability, while the `/dev` based interface relies on *cgroups*. Technically, the

/dev based interface also relies on user-permissions as a second-level access control mechanism (on the actual device node files themselves), but the primary access control mechanism is cgroups. The current CUDA 11/R450 GA (Linux driver 450.51.06) supports both mechanisms, but going forward the /dev based interface is the preferred method and the /proc based interface is deprecated. For now, users can choose the desired interface by using the nv_cap_enable_devfs parameter on the nvidia.ko kernel module:

- When nv_cap_enable_devfs=0 the /proc based interface is enabled.
- When nv_cap_enable_devfs=1 the /dev based interface is enabled.
- A setting of nv_cap_enable_devfs=0 is the default for the R450 driver (as of Linux 450.51.06).
- All future NVIDIA datacenter drivers will have a default of nv_cap_enable_devfs=1.

An example of loading the nvidia kernel module with this parameter set can be seen below:

```
$ modprobe nvidia nv_cap_enable_devfs=1
```

10.1. /dev based nvidia-capabilities

The system level interface for interacting with /dev based capabilities is actually through a combination of /proc and /dev.

First, a new major device is now associated with nvidia-caps and can be read from the standard /proc/devices file.

```
$ cat /proc/devices | grep nvidia-caps
508 nvidia-caps
```

Second, the exact same set of files exist under /proc/driver/nvidia/capabilities. These files no longer control access to the capability directly and instead, the contents of these files point at a device node under /dev, through which cgroups can be used to control access to the capability.

This can be seen in the example below:

```
$ cat /proc/driver/nvidia/capabilities/mig/config
DeviceFileMinor: 1
DeviceFileMode: 256
DeviceFileModify: 1
```

The combination of the device major for nvidia-caps and the value of DeviceFileMinor in this file indicate that the mig-config capability (which allows a user to create and destroy MIG devices) is controlled by the device node with a major:minor of 238:1. As such, one will need to use cgroups to grant a process read access to this device in order to configure MIG devices. The purpose of the `DeviceFileMode` and `DeviceFileModify` fields in this file are explained later on in this section.

The standard location for these device nodes is under `/dev/nvidia-caps` as seen in the example below:

```
$ ls -l /dev/nvidia-caps
total 0
cr-------- 1 root root 508,  1 Nov 21 17:16 nvidia-cap1
cr--r--r-- 1 root root 508,  2 Nov 21 17:16 nvidia-cap2
...
```

Unfortunately, these device nodes cannot be automatically created/deleted by the NVIDIA driver at the same time it creates/deletes files underneath `/proc/driver/nvidia/capabilities` (due to GPL compliance issues). Instead, a user-level program called `nvidia-modprobe` is provided, that can be invoked from user-space in order to do this. For example:

```
$ nvidia-modprobe \
    -f /proc/driver/nvidia/capabilities/mig/config \
    -f /proc/driver/nvidia/capabilities/mig/monitor

$ ls -l /dev/nvidia-caps
total 0
cr-------- 1 root root 508,  1 Nov 21 17:16 nvidia-cap1
cr--r--r-- 1 root root 508,  2 Nov 21 17:16 nvidia-cap2
```

`nvidia-modprobe` looks at the `DeviceFileMode` in each capability file and creates the device node with the permissions indicated (e.g. `+ur` from a value of `256 (o400)` from our example for `mig-config`).

Programs such as `nvidia-smi` will automatically invoke `nvidia-modprobe` (when available) to create these device nodes on your behalf. In other scenarios it is not necessarily required to use `nvidia-modprobe` to create these device nodes, but it does make the process simpler.

If you actually want to prevent `nvidia-modprobe` from ever creating a particular device node on your behalf, you can do the following:

```
# Give a user write permissions to the capability file under /proc
$ chmod +uw /proc/driver/nvidia/capabilities/mig/config

# Update the file with a “DeviceFileModify” setting of 0
$ echo "DeviceFileModify: 0" > /proc/driver/nvidia/capabilities/mig/config
```

You will then be responsible for managing creation of the device node referenced by `/proc/driver/nvidia/capabilities/mig/config` going forward. If you want to change that in the future, simply reset it to a value of `"DeviceFileModify: 1"` with the same command sequence.

This is important in the context of containers because we may want to give a container access to a certain capability even if it doesn't exist in the `/proc` hierarchy yet.

For example, granting a container the `mig-config` capability implies that we should also grant it capabilities to access all possible gis and cis that could be created for any GPU on the system. Otherwise the container will have no way of working with those gis and cis once they have actually been created.

One final thing to note about `/dev` based capabilities is that the minor numbers for all possible capabilities are predetermined and can be queried under various files of the form:

```
/proc/driver/nvidia-caps/*-minors
```

For example, all capabilities related to MIG can be looked up as:

```
$ cat /proc/driver/nvidia-caps/mig-minors
config 1
monitor 2
gpu0/gi0/access 3
gpu0/gi0/ci0/access 4
gpu0/gi0/ci1/access 5
gpu0/gi0/ci2/access 6
...
gpu31/gi14/ci6/access 4321
gpu31/gi14/ci7/access 4322
```

The format of the content follows: `GPU<deviceMinor>/gi<GPU instance ID>/ci<compute instance ID>`

Note that the GPU device minor number can be obtained by using either of these mechanisms:

- The NVML API `nvmlDeviceGetMinorNumber()` so it returns the device minor number
- Or use the PCI BDF available under `/proc/driver/nvidia/gpus/domain:bus:device:function/information`. This file contains a "Device Minor" field.

> Note:
>
> The NVML device numbering (e.g. through `nvidia-smi`) is not the device minor number.

For example, if the MIG geometry was created as below:

```
+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |           BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  0    1   0   0  |     19MiB / 40192MiB | 14      0 |  3   0    3    0    3 |
|                  |      0MiB / 65535MiB |           |                       |
+------------------+                      +-----------+-----------------------+
|  0    1   1   1  |                      | 14      0 |  3   0    3    0    3 |
|                  |                      |           |                       |
+------------------+                      +-----------+-----------------------+
```

```
|  0    1   2   2  |                     | 14       0 |  3   0    3    0    3 |
|                  |                     |            |                        |
+------------------+---------------------+------------+------------------------+
```

Then the corresponding device nodes: `/dev/nvidia-cap12`, `/dev/nvidia-cap13` and `/dev/nvidia-cap14` and `/dev/nvidia-cap15` would be created.

10.2. /proc based nvidia-capabilities (**Deprecated**)

The system level interface for interacting with `/proc` based `nvidia-capabilities` is rooted at `/proc/driver/nvidia/capabilities`. Files underneath this hierarchy are used to represent each capability, with read access to these files controlling whether a user has a given capability or not. These files have *no content* and only exist to represent a given capability.

For example, the `mig-config` capability (which allows a user to create and destroy MIG devices) is represented as follows:

```
        /proc/driver/nvidia/capabilities
        └── mig
            └── config
```

Likewise, the capabilities required to run workloads on a MIG device once it has been created are represented as follows (namely as access to the GPU Instance and Compute Instance that comprise the MIG device):

```
        /proc/driver/nvidia/capabilities
        └── gpu0
            └── mig
                ├── gi0
                │   ├── access
                │   └── ci0
                │       └── access
                ├── gi1
                │   ├── access
                │   └── ci0
                │       └── access
                └── gi2
                    ├── access
                    └── ci0
                        └── access
```

And the corresponding file system layout is shown below with read permissions:

```
$ ls -l /proc/driver/nvidia/capabilities/gpu0/mig/gi*
/proc/driver/nvidia/capabilities/gpu0/mig/gi1:
total 0
-r--r--r-- 1 root root 0 May 24 17:38 access
dr-xr-xr-x 2 root root 0 May 24 17:38 ci0
```

```
/proc/driver/nvidia/capabilities/gpu0/mig/gi2:
total 0
-r--r--r-- 1 root root 0 May 24 17:38 access
dr-xr-xr-x 2 root root 0 May 24 17:38 ci0
```

For a CUDA process to be able to run on top of MIG, it needs access to the Compute Instance capability and its parent GPU Instance. Thus a MIG device is identified by the following format:

```
MIG-<GPU-UUID>/<GPU instance ID>/<compute instance ID>
```

As an example, having read access to the following paths would allow one to run workloads on the MIG device represented by `<gpu0, gi0, ci0>`:

```
/proc/driver/nvidia/capabilities/gpu0/mig/gi0/access
/proc/driver/nvidia/capabilities/gpu0/mig/gi0/ci0/access
```

Note, that there is no access file representing a capability to run workloads on gpu0 (only on gi0 and ci0 that sit underneath gpu0). This is because the traditional mechanism of using cgroups to control access to top level GPU devices (and any required meta devices) is still required. As shown earlier in the document, the cgroups mechanism applies to:

```
/dev/nvidia0
/dev/nvidiactl
/dev/nvidiactl-uvm
...
```

In the context of containers, a new mount namespace should be overlaid on top of the path for `/proc/driver/nvidia/capabilities`, and only those capabilities a user wishes to grant to a container should be bind-mounted in. Since the host's user/group information is retained across the bind-mount, it must be ensured that the correct user permissions are set for these capabilities on the host before injecting them into a container.

Chapter 11. Changelog

- 11/17/2022 (author: PR): Includes the following changes:
  - Updates for Hopper, CUDA 12.0/R525
  - Reorginzation of several chapters
  - Added more information on `/dev` based capabilities
- 7/19/2022 (author: PR): Includes the following changes:
  - Added a chapter on virtualization.
- 6/6/2022 (author: PR): Includes the following changes:
  - Fix table that lists A30 profiles.
  - Update Slurm documentation link.
- 8/26/2021 (author: PR): Includes the following changes:
  - Improve explanation of GPU Partitioning.
- 6/30/2021 (author: PR): Includes the following changes:
  - Add info on unique UUIDs for MIG devices.
  - Update supported profiles.
- 4/22/2021 (author: PR): Includes the following changes:
  - Added information for Slurm and CUDA MPS.
- 4/14/2021 (author: PR): Includes the following changes:
  - Add additional supported products.
  - Update diagrams.
  - Add link to vGPU documentation.
- 2/17/2021 (author: PR): Includes the following changes:
  - Add note about persistence of MIG devices.
  - Add link to gathering telemetry for MIG.

  - Add link to K8s documentation.
- 11/24/2020 (author: PR): Includes the following changes:
  - Fix broken container example.
  - Added link to Kubernetes documentation.
  - Added minimum required software versions.
  - Added MIG mode enablement example on DGX A100.
- 11/06/2020 (author: PR): Includes the following changes:
  - Updated examples.
  - Added documentation for new CLI options.
  - Added doc links for vGPU.
  - Added doc links for Kubernetes support.
  - Fixed typos.
- 8/7/2020 (author: PR):
  - Added information on device nodes and nvidia-capabilities with CUDA 11.0 GA.
- 5/28/2020 (author: PR):
  - Initial Version.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051

https://www.nvidia.com