NVIDIA MULTI-INSTANCE GPU USER GUIDE

RN-08625-v1.0 _v01 | November 2020

User Guide
TABLE OF CONTENTS

Chapter 1. Changelog ........................................................................................................ 1
Chapter 2. Introduction .................................................................................................... 2
Chapter 3. Concepts ........................................................................................................ 4
  3.1. Terminology ........................................................................................................ 4
  3.2. Partitioning .......................................................................................................... 5
  3.3. CUDA Concurrency Mechanisms .......................................................................... 7
Chapter 4. Deployment Considerations .......................................................................... 9
  4.1. System Considerations ....................................................................................... 9
  4.2. Application Considerations .............................................................................. 10
Chapter 5. Device Nodes and Capabilities .................................................................... 11
  5.1. /proc based nvidia-capabilities ....................................................................... 12
  5.2. /dev based nvidia-capabilities ........................................................................ 14
Chapter 6. MIG Device Names ....................................................................................... 17
Chapter 7. Running with MIG ...................................................................................... 19
  7.1. Prerequisites .................................................................................................... 19
  7.2. Enable MIG Mode ............................................................................................ 19
  7.3. List GPU Instance Profiles ............................................................................. 21
  7.4. Creating GPU Instances ................................................................................ 22
  7.5. Device Enumeration ....................................................................................... 25
  7.6. Running CUDA Applications on Bare-Metal .................................................... 25
    7.6.1. CUDA Device Enumeration ....................................................................... 26
    7.6.2. GPU Instances ....................................................................................... 26
    7.6.3. Compute Instances ................................................................................. 28
    7.6.4. Destroying GPU Instances ....................................................................... 30
  7.7. Running CUDA Applications as Containers .................................................... 30
    7.7.1. Install Docker ......................................................................................... 30
    7.7.2. Install NVIDIA Container Toolkit ............................................................. 31
    7.7.3. Running Containers ............................................................................... 31
Chapter 1.
CHANGELOG

- 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

Chapter 2.
INTRODUCTION

The new Multi-Instance GPU (MIG) feature allows the NVIDIA A100 GPU 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 A100 GPU.

With NVIDIA A100 GPU, 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 coming soon. For more information on vGPU, refer to the vGPU documentation.
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 the NVIDIA A100 with MIG.
3.1. 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 A100 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 A100 GPU. A GPU SM slice is roughly one seventh of the total number of SMs available in A100 when configured in MIG mode.

**GPU Slice**

A GPU slice is the smallest fraction of the A100 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 its 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.

### 3.2. Partitioning

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.

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.

<table>
<thead>
<tr>
<th>Profile Name</th>
<th>Fraction of Memory</th>
<th>Fraction of SMs</th>
<th>Hardware Units</th>
<th>Number of Instances Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIG 1g.5gb</td>
<td>1/8</td>
<td>1/7</td>
<td>0 NVDECs</td>
<td>7</td>
</tr>
<tr>
<td>MIG 2g.10gb</td>
<td>2/8</td>
<td>2/7</td>
<td>1 NVDECs</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1  GPU Instance Profiles
<table>
<thead>
<tr>
<th>Profile Name</th>
<th>Fraction of Memory</th>
<th>Fraction of SMs</th>
<th>Hardware Units</th>
<th>Number of Instances Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIG 3g.20gb</td>
<td>4/8</td>
<td>3/7</td>
<td>2 NVDECs</td>
<td>2</td>
</tr>
<tr>
<td>MIG 4g.20gb</td>
<td>4/8</td>
<td>4/7</td>
<td>2 NVDECs</td>
<td>1</td>
</tr>
<tr>
<td>MIG 7g.40gb</td>
<td>Full</td>
<td>7/7</td>
<td>5 NVDECs</td>
<td>1</td>
</tr>
</tbody>
</table>

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

**Figure 2  Valid GPU Instance Combinations**

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. The only exception to this rule is the combination of a (4 memory, 4 compute) and a (4 memory, 3 compute) profile, which is currently not supported. However, a combination of (4 memory, 3 compute) and (4 memory, 3 compute) again is supported and illustrated in this section.

An example of one such valid combination can be seen below:

**Figure 3  Example Configuration of GPU Instances**

It's important to note that the following two GPU Instance combinations are actually distinct due to the way the instances are created (referred to as placements) on the GPU:
3.3. 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 the NVIDIA A100 while addressing some of the limitations with the other CUDA technologies for running parallel work.

Table 2  CUDA Concurrency Mechanisms

<table>
<thead>
<tr>
<th>Partition Type</th>
<th>Streams</th>
<th>MPS</th>
<th>MIG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Single Process</td>
<td>Logical</td>
<td>Physical</td>
</tr>
<tr>
<td>Max Partitions</td>
<td>Unlimited</td>
<td>48</td>
<td>7</td>
</tr>
<tr>
<td>Feature</td>
<td>Streams</td>
<td>MPS</td>
<td>MIG</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------</td>
<td>-------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>SM Performance Isolation</td>
<td>No</td>
<td>Yes (by percentage, not partitioning)</td>
<td>Yes</td>
</tr>
<tr>
<td>Memory Protection</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Memory Bandwidth QoS</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Error Isolation</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Cross-Partition Interop</td>
<td>Always</td>
<td>IPC</td>
<td>Limited IPC</td>
</tr>
<tr>
<td>Reconfigure</td>
<td>Dynamic</td>
<td>Process Launch</td>
<td>When Idle</td>
</tr>
</tbody>
</table>
MIG functionality is provided as part of the NVIDIA GPU driver starting with the CUDA 11.0 / R450 release.

### 4.1. System Considerations

The following system considerations are relevant for NVIDIA A100 when the GPU is in MIG mode.

- MIG is supported only on Linux operating system distributions supported by CUDA 11/R450. It is also recommended to use at least to use the NVIDIA Datacenter Linux driver 450.80.02 or higher. This driver can either be installed by using the .run installer package or by using the `cuda-drivers-450` meta-package from the CUDA network repository.

  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
  - GPU pass-through to Linux guests on top of supported hypervisors
  - vGPU on top of supported hypervisors

  MIG allows multiple vGPUs to run in parallel on a single A100, while preserving the isolation guarantees that vGPU provides. For more information on vGPU, refer to the software documentation.

- Setting MIG mode on the A100 requires a GPU reset and super-user privileges. Once A100 is in MIG mode, instance management is then dynamic (i.e. does not require a GPU reset). Note that the setting is on a per-GPU basis.

- Similar to ECC mode, MIG mode setting is persistent across reboots until the user toggles the setting explicitly
Deployment Considerations

- 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`.

4.2. 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
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
├── nvidia-ctl
├── 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 `/proc` and the second is via `/dev`. 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 \texttt{nv\_cap\_enable\_devfs} parameter on the \texttt{nvidia.ko} kernel module:

- When \texttt{nv\_cap\_enable\_devfs}=0 the /proc based interface is enabled.
- When \texttt{nv\_cap\_enable\_devfs}=1 the /dev based interface is enabled.
- A setting of \texttt{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 \texttt{nv\_cap\_enable\_devfs}=1.

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

\begin{verbatim}
$ modprobe nvidia nv_cap_enable_devfs=1
\end{verbatim}

### 5.1. /proc based nvidia-capabilities

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 \texttt{mig-config} capability (which allows a user to create and destroy MIG devices) is represented as follows:

\begin{verbatim}
/proc/driver/nvidia/capabilities
    mig
    config
\end{verbatim}

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):

\begin{verbatim}
/proc/driver/nvidia/capabilities
    g0
    mig
        q0
            access
        ci0
            access
    g1
\end{verbatim}
And the corresponding file system layout is shown below with read permissions:

```bash
$ 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.

5.2. /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-capabilities and can be read from the standard /proc/devices file.

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

Second, the exact same set of files exist under /proc/driver/nvidia/capabilities as they did for /proc based capabilities, except that these files no longer control access to the capability directly. 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:

```
$ ll /dev/nvidia-caps
total 0
cr--------  1 root root 238,   1 May 30 20:41 nvidia-cap1
cr--r--r--  1 root root 238,   2 May 30 20:41 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
```

```
$ ll /dev/nvidia-caps
```
```
total 0
  cr--------  1 root root 238,  1 May 30 20:41 nvidia-cap1
  cr--r--r--  1 root root 238,  2 May 30 20:41 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 (0400) 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.

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
```
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.
Chapter 6.
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.

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 3  Device names when using a single CI

<table>
<thead>
<tr>
<th>Memory</th>
<th>20gb</th>
<th>10gb</th>
<th>5gb</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Instance</td>
<td>3g</td>
<td>2g</td>
<td>1g</td>
</tr>
<tr>
<td>Compute Instance</td>
<td>3c</td>
<td>2c</td>
<td>1c</td>
</tr>
<tr>
<td>MIG Device</td>
<td>3g.20gb</td>
<td>2g.10gb</td>
<td>1g.5gb</td>
</tr>
<tr>
<td>GPC GPC GPC</td>
<td>GPC GPC GPC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 4  Device names when using multiple CIs

<table>
<thead>
<tr>
<th>Memory</th>
<th>20gb</th>
<th>20gb</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Instance</td>
<td>3g</td>
<td>3g</td>
</tr>
<tr>
<td>Compute Instance</td>
<td>1c</td>
<td>1c</td>
</tr>
<tr>
<td>MIG Device</td>
<td>1c.3g.20gb</td>
<td>1c.3g.20gb</td>
</tr>
<tr>
<td>GPC</td>
<td>GPC</td>
<td>GPC</td>
</tr>
</tbody>
</table>
Chapter 7.
RUNNING WITH MIG

7.1. Prerequisites

The following prerequisites and minimum software versions are recommended when using A100 in MIG mode.

- MIG is supported only on NVIDIA A100 products and associated systems using A100 (e.g. DGX A100 and HGX A100)
- CUDA 11 and NVIDIA driver 450.80.02 or later
- CUDA 11 supported Linux operating system distributions
- 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.11.0.h) included in CUDA 11.

7.2. Enable MIG Mode

By default, MIG mode is not enabled on the NVIDIA A100. 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     |
+-----------------------------------------------------------------------------+
```

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.

```bash
$ 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
```

If you are using MIG inside a VM with GPU 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:

```bash
$ 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.
```

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:

```bash
$ 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 A100 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
```

### 7.3. 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.

```
$ sudo nvidia-smi mig -lgip
+--------------------------------------------------------------------------+
| GPU instance profiles:                                                   |
| GPU   Name          ID    Instances Free/Total Memory     P2P    SM    DEC   ENC  |
|                           GiB              CE    JPEG  OFA  |
|==========================================================================|
|   0  MIG 1g.5gb     19     7/7        4.95       No     14     0     0   |
|                                                          1     0     0   |
|   0  MIG 2g.10gb    14     3/3        9.90       No     28     1     0   |
|                                                          2     0     0   |
|   0  MIG 3g.20gb     9     2/2        19.79      No     42     2     0   |
|                                                          3     0     0   |
|   0  MIG 4g.20gb     5     1/1        19.79      No     56     2     0   |
|                                                          4     0     0   |
|   0  MIG 7g.40gb     0     1/1        39.59      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.

```bash
$ sudo nvidia-smi mig -lgipp
GPU  0 Profile ID 19 Placements: {0,1,2,3,4,5,6}:1
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).

### 7.4. 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.

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:

```bash
$ 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:

```bash
$ sudo nvidia-smi mig -lgi
+----------------------------------------------------+
| GPU instances:                                     |
| GPU   Name          Profile  Instance   Placement  |
+----------------------------------------------------+
Now verify that the GIs and corresponding CIs are created:

```
$ nvidia-smi
```

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
```

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

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.

```bash
$ 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.

```bash
$ 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)
```

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

<table>
<thead>
<tr>
<th>GPU instances:</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU   Name          Profile  Instance   Placement</td>
</tr>
<tr>
<td>ID       ID       Start:Size</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>0  MIG 1g.5gb       19        7          0:1</td>
</tr>
<tr>
<td>0  MIG 1g.5gb       19        8          1:1</td>
</tr>
<tr>
<td>0  MIG 1g.5gb       19        9          2:1</td>
</tr>
<tr>
<td>0  MIG 1g.5gb       19       10          3:1</td>
</tr>
<tr>
<td>0  MIG 1g.5gb       19       13          6:1</td>
</tr>
<tr>
<td>0  MIG 2g.10gb      14        5          4:2</td>
</tr>
</tbody>
</table>

7.5. Device Enumeration

GPU Instances (GI)s and Compute Instances (CI)s are enumerated in the new `/proc` filesystem layout for MIG

$ ls -l /proc/driver/nvidia/capabilities/gpu0/mig/gi*
/proc/driver/nvidia/capabilities/gpu0/mig/gi1:
total 0
-r---r--- 1 root root 0 Nov 6 20:34 access
dr-xr-xr-x 2 root root 0 Nov 6 20:34 ci0

/proc/driver/nvidia/capabilities/gpu0/mig/gi2:
total 0
-r---r--- 1 root root 0 Nov 6 20:34 access
dr-xr-xr-x 2 root root 0 Nov 6 20:34 ci0

7.6. Running CUDA Applications on Bare-Metal
7.6.1. CUDA Device Enumeration

MIG supports running CUDA applications by specifying the CUDA device on which the application should be run. With CUDA 11, only enumeration of a single MIG instance is supported.

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 by specifying the CI and the corresponding parent GI. The new format follows this convention: `MIG-G<GPU-UUID>/G<GPU instance ID>/C<compute instance ID>`.

7.6.2. 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.

```bash
$ nvidia-smi -L
GPU 0: A100-SXM4-40GB (UUID: GPU-e86cb44c-6756-fd30-cd4a-1e6da3ca9b0)
  MIG 3g.20gb Device 0: (UUID: MIG-GPU-e86cb44c-6756-fd30-cd4a-1e6da3ca9b0/1/0)
  MIG 3g.20gb Device 1: (UUID: MIG-GPU-e86cb44c-6756-fd30-cd4a-1e6da3ca9b0/2/0)

$ CUDA_VISIBLE_DEVICES=MIG-GPU-e86cb44c-6756-fd30-cd4a-1e6da3ca9b0/1/0 ./. BlackScholes &
$ CUDA_VISIBLE_DEVICES=MIG-GPU-e86cb44c-6756-fd30-cd4a-1e6da3ca9b0/2/0 ./. BlackScholes &

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

```
### 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
```

For monitoring MIG devices on 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 documentation for more details.
7.6.3. 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       SM       DEC   ENC   OFA  |
|       Instance                       ID     Free/Total     SM       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 ID  0
Successfully created compute instance on GPU  0 GPU instance ID  1 using profile ID  0
Successfully created compute instance on GPU  0 GPU instance ID  1 using profile 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     |
+=======================================================|
|   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
```
Now, three BlackScholes applications can be created and run in parallel:

```
$ CUDA_VISIBLE_DEVICES=MIG-GPU-e86cb44c-6756-fd30-cd4a-1e6da3caf9b0/1/0 ./
BlackScholes &
$ CUDA_VISIBLE_DEVICES=MIG-GPU-e86cb44c-6756-fd30-cd4a-1e6da3caf9b0/1/1 ./
BlackScholes &
$ CUDA_VISIBLE_DEVICES=MIG-GPU-e86cb44c-6756-fd30-cd4a-1e6da3caf9b0/1/2 ./
BlackScholes &
```

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

```
$ nvidia-smi
```

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

```
```
7.6.4. Destroying GPU Instances

Once the A100 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.

```
$ 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 MIG devices have now been torn down on the A100:

```
$ nvidia-smi
+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
|------------------+----------------------+-----------+-----------------------|
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |                      | SM        | 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                                                 |
+-----------------------------------------------------------------------------+
```

7.7. 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.

7.7.1. 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
```
7.7.2. 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:

```
$ curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -

   && distribution=$(./etc/os-release;echo $ID$VERSION_ID) 
   && curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list 
   && sudo apt-get update
```

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

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

7.7.3. 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 two formats to specify MIG devices:

1. MIG-<GPU-UUID>/<GPU instance ID>/<compute instance ID>
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 \
   --device=MIG-GPU-e86cb44c-6756-fd30-cd4a-1e6da3caf9b0/1/0 \
   nvidia/cuda nvidia-smi
```

+-----------------------------------------------------------------------------+
### MIG devices:

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

### Processes:

<table>
<thead>
<tr>
<th>GPU ID</th>
<th>GI ID</th>
<th>PID</th>
<th>Type</th>
<th>Process name</th>
<th>GPU Memory Usage</th>
</tr>
</thead>
</table>

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-GPU-e86cb44c-6756-fd30-cd4a-1e6da3caf9b0/1/0)
```

# 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-GPU-e86cb44c-6756-fd30-cd4a-1e6da3caf9b0/1/0)
```

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 Marietho"

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]

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.
Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS GUIDE IS PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this guide. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2020 NVIDIA Corporation. All rights reserved.

www.nvidia.com