

Lilac GPU Primer

What should I know when requesting GPU resources on the Lilac Cluster

This page documents the various GPU settings and modes for the Lilac cluster in more detail than the [Lilac Cluster Intro page](#) provides. This page looks specifically at the new GTX 1080 GPUs, although many of the options apply to other Nvidia GPUs as well. Where applicable, this primer will talk about how these properties interact with the Lilac Cluster

Topics Covered in this Primer

1. Looking at the status of the GPUs on a node through `nvidia-smi`
 - a. The difference between `process exclusive` and `shared` mode on nVidia GPUs (Compute Modes)
 - b. Important terminology notes
2. Identifying and running on specific GPUs by the `CUDA_VISIBLE_DEVICES` environment variable
 - a. How processes are distributed when `CUDA_VISIBLE_DEVICES` is set
3. Running multiple processes on the same GPU through the CUDA Multi-Process Service (MPS)
 - a. Changes MPS makes to GPU identification and control
 - b. Limitations of the MPS feature

Looking at the status of the GPUs on a node through `nvidia-smi`

nVidia provides a tool to view the status of GPUs, such as their current memory load, temperature, and operating mode. The command to see this info is `nvidia-smi`, but this can only be run on nodes which have GPUs, so this command fails on the head nodes. Here is a sample output from a Lilac node with 4 GPUs on it.

```

+-----+
| NVIDIA-SMI 367.48                Driver Version: 367.48
|
+-----+
| GPU  Name          Persistence-M| Bus-Id        Disp.A | Volatile Uncorr.
ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute
M. |
|=====+=====+=====+
====|
|  0  GeForce GTX 1080    Off   | 0000:02:00.0    Off   |
N/A |
|  0%  39C   P8     6W / 180W |    2MiB /  8113MiB |    0%   E.
Process |
+-----+
|  1  GeForce GTX 1080    Off   | 0000:03:00.0    Off   |
N/A |
|  0%  41C   P8    11W / 180W |    2MiB /  8113MiB |    0%   E.
Process |
+-----+
|  2  GeForce GTX 1080    Off   | 0000:83:00.0    Off   |
N/A |
|  0%  36C   P8     9W / 180W |    2MiB /  8113MiB |    0%   E.
Process |
+-----+
|  3  GeForce GTX 1080    Off   | 0000:84:00.0    Off   |
N/A |
|  0%  35C   P8    11W / 180W |    2MiB /  8113MiB |    0%   E.
Process |
+-----+

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

```

The entries at the top match with the entries per GPU by position, e.g. the Bus-Id entry in the top center cell matches to the 0000:02:00.0 in the

Here you can see the process is "python", the process ID you would see in `top` is 3505, it is running on the GPU with index 0, and it is using 115 MiB of that GPUs on board memory. The type field will either be "C" for compute (what most of your processes will be), "G" for graphics (if the GPU is doing rendering like being connected to a monitor), or "C+G" if its doing both.

Important Terminology Notes

- Because of the possible confusion involving the word "Default" as it relates to the `Compute Mode` of the GPUs, the `Compute M. = Default` case will be referred to as `shared` mode through this document.
- `Shared` in this document refers only to the computation mode of the GPU in that multiple Contexts can share this GPU. `Shared` does **not** mean that a GPU is shared between different jobs submitted to the LSF queue.
- `Exclusive process` will be shortened to `exclusive` since there are not multiple modes of exclusivity on the GTX 1080s.
- `Exclusive` in this document will only refer to the compute mode of the GPU.

Identifying and running on specific GPUs by the `CUDA_VISIBLE_DEVICES` environment variable

Each physical machine has some number of physical GPUs associated with it. When you start a process on a GPU, which physical GPU it start on depends on a few factors:

1. If the GPUs are in `shared` or `exclusive` mode
2. The value of the `CUDA_VISIBLE_DEVICES` environment variable.

`CUDA_VISIBLE_DEVICES` serves as the identifier as to which GPU *slots* are available for processes to start on. Lilac LSF jobs automatically assign you the `CUDA_VISIBLE_DEVICES` environment variable per node matching to the GPUs that are available to you job. For instance, you request 2 GPUs per node on 2 nodes, you are given the GPUs in slots 0 and 1 on NodeA, and the GPUs in slots 1 and 2 on NodeB. Your `CUDA_VISIBLE_DEVICES` variables will look like this:

- NodeA: `CUDA_VISIBLE_DEVICES=0,1`
- NodeB: `CUDA_VISIBLE_DEVICES=1,2`

These indexes match up with the **GPU** entry from the `nvidia-smi` output above. When your job starts, these are assigned to you, however, you can change them to control exactly what device your process starts on. If you tell your job to sign into NodeA and then export `CUDA_VISIBLE_DEVICES=0`, any process which uses the GPU can then only execute on the GPU in slot 0. You can only set this to devices assigned to your job, if you attempt to set this to GPUs not assigned to you, then your processes will not start.

Lilac is configured to only give you access to the devices you request, and renumbers the GPUs starting at 0. So if you request 2 GPUs and you get the GPUs in physical slots 1 and 3, you will only see 2 GPUs though `nvidia-smi` and they will be given the indices 0 and 1.

Unsetting `CUDA_VISIBLE_DEVICES` is the same as setting it to the list of indices for all available GPUs. E.g. if you have 3 GPUs assigned to you in slots 0, 1 and 2; then unsetting `CUDA_VISIBLE_DEVICES` is the same as setting `CUDA_VISIBLE_DEVICES=0,1,2`

How processes are distributed when `CUDA_VISIBLE_DEVICES` is set

This section covers the behavior of how processes are distributed to GPUs with the `CUDA_VISIBLE_DEVICES` flag. Remember that the behavior for an unset `CUDA_VISIBLE_DEVICES` is also defined. Although Lilac jobs are started in `exclusive` mode, its important to cover the `shared` mode behavior because of the Multi-Process Service (MPS) covered in the next section.

- `exclusive` mode:
 - `CUDA_VISIBLE_DEVICES` is checked
 - for each device in `CUDA_VISIBLE_DEVICES`
 - If no process on device, start process
 - Loop until process has started
 - If process has not started and all devices have been looped through
 - Fail state, "No Available Devices"
- `shared` mode
 - `CUDA_VISIBLE_DEVICES` is checked
 - Pick the first device in `CUDA_VISIBLE_DEVICES`
 - Start process on device
 - If out of memory
 - Fail

As you can see, the behavior of `shared` mode requires some management of `CUDA_VISIBLE_DEVICES` otherwise only one GPU will be used.

Running multiple processes on the same GPU through the CUDA Multi-Process Service (MPS)

It is possible to run multiple GPU Contexts (rough equivalent to a CPU process on the GPU) on a single device, even if the device is in `exclusive` mode through the [CUDA Multi-Process Service \(MPS\)](#). In short, this feature allows you to emulate `shared` mode on the GPU by running a single MPS daemon on the GPU which processes your job's processes to run them on the GPU together.

You can enable this feature with your Lilac jobs by adding the following flag to your `bsub` submission, or as a `#BSUB` directive in your job script:

```
-gpu "num=1:mode=exclusive_process:mps=yes:j_exclusive=yes"
```

This sets the environment variable that tells LSF to start the CUDA MPS daemon on the nodes and GPUs assigned to your job.

When you start processes with MPS, you will no longer see your own processes in the process list from `nvidia-smi`, instead, you will see that each GPU will have a single process called `nvidia-cuda-mps-server`, independent of how many processes that are running on that GPU.

Important: MPS does alter the behavior of `CUDA_VISIBLE_DEVICES`, your GPU indices, and has a few limitations as to what can be run on it.

Changes MPS makes to GPU identification and control

MPS does causes two major changes to your GPUs you should be aware of

1. Devices are re-indexed starting from 0, no matter what index they were before
2. `CUDA_VISIBLE_DEVICES` is unset

Since GPUs are re-indexed, your `CUDA_VISIBLE_DEVICES` variables should start at 0, however, your job will need to figure out how many GPUs per node you have either from the job environment variables, or hard coded in your script since you cannot determine how many GPUs you have from the `CUDA_VISIBLE_DEVICES` variable. One option is to look at the `LSB_HOSTS` variable once your job has started which is a space separated list of the hosts your job is running on, where each host is repeated for each task on that host. E.g. If you have 3 processes on host "ls02" and 2 processes on "ls05" with equal GPUs on each one, your `LSB_HOSTS` would look like

```
LSB_HOSTS="ls02 ls02 ls02 ls05 ls05"
```

where you can then infer the what `CUDA_VISIBLE_DEVICES` per host should be by counting up from 0 on each host, so

- ls02: `CUDA_VISIBLE_DEVICES=0,1,2`
- ls05: `CUDA_VISIBLE_DEVICE=0,1`

Limitations of the MPS feature

Although MPS allows multiple processes to run on an `exclusive` GPU, there are a few limitations to this service.

1. A *maximum* of 16 Contexts per GPU are supported
2. You cannot run OpenCL Contexts on the GPU, only CUDA Contexts
3. There is a small GPU memory overhead to running the MPS server on each GPU
 - This appears to be about 50-100 MiB per GPU total
 - This memory cost is per GPU, **not** per process
4. Running MPS on `shared` GPUs is pointless

These will probably not affect most jobs and users, but please contact the HPC team if you feel these limitations will affect your work. Remember that MPS is optional and is off by default.

Lastly, if you find yourself running jobs on `shared` mode GPUs, *do not run MPS*. You can technically run MPS on `shared` mode GPUs, but there is no gain from this since you can already run multiple processes per GPU, but you do lose some memory on the GPU, there is a small computational overhead to running the MPS service, and you limit the number and type of Contexts you can start, so there really is no point in doing so.

Related articles

- [Lilac GPU Primer](#)

