# Running jobs - GPU

<table>
<thead>
<tr>
<th>Quick Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC at NYU</td>
</tr>
<tr>
<td>HPC Newsletter</td>
</tr>
<tr>
<td>Getting started on Prince</td>
</tr>
<tr>
<td>Getting an account</td>
</tr>
<tr>
<td>Prince How-to Articles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Clusters and Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mercer (HPC, retired)</td>
</tr>
<tr>
<td>Prince (HPC)</td>
</tr>
<tr>
<td>Dumbo (Hadoop)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logging in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
</tr>
<tr>
<td>Mac / Linux</td>
</tr>
</tbody>
</table>
Dlama (NYU Abu Dhabi)

Transferring data to/from the clusters

Transferring data to/from Prince cluster using Globus

Submitting jobs with sbatch

Available software

Building Software Tutorials

Programming for Biologists

Research Gallery

FAQs

NYU HPC Cheat Sheet

Wiki Map

HPC Policies
Running jobs on the NYU HPC clusters

Job Scheduling Overview

Login and compute nodes

Queues

Writing a job script

Job stdout and stderr

Which filesystems should I use?

Submitting a Job

Basic qsub options and directives

Setting resource limits

Running MPI jobs

Running GPU jobs

Running R jobs

Running Matlab, Gaussian, etc jobs

Working interactively
Job dependencies and delaying starting

Running many similar jobs

Setting job priorities

Where in the queue is my job, and why?

Monitoring jobs with qstat

qsstat in more det

See what is running where with pbstop

When will my job start?

Why hasn't my job started?

Canceling a job

GPU jobs

You can request GPUs just like requesting ppp (processors per node):

Options to request compute resources:

- `-l walltime=walltime`
  Maximum wallclock time the job will need. Default is 1 hour. Walltime is specified in seconds or as `hh:mm:ss`.

- `-l mem=memory`
  Maximum memory per node the job will need. Default depends on queue, normally 2GB for serial jobs and the full node for parallel jobs. Memory should be specified with units, eg `500MB` or `8GB`.

- `-l nodes=num:ppn=num`
  Number of nodes and number of processors per node required. Default is 1 node and 1 processor per node. The `:ppn=num` can be omitted, in which case (at NYU HPC) you will get full nodes. When using multiple nodes the job script will be executed on the first
allocated node.

- `-q queue`
  
  Submit to a specific queue. If not specified, Torque will choose a queue based on the resources requested.

  A job submitted without requesting a specific queue or resources will go to the default serial queue (s48 on Mercer) with the default resource limits for that queue.

  Requesting the resources you need, as accurately as possible, allows your job to be started at the earliest opportunity as well as helping the system to schedule work efficiently to everyone’s benefit.

To request GPU nodes:

- `-l nodes=1:ppn=1:gpus=1`
  
  1 node with 1 core and 1 GPU

- `-l nodes=1:ppn=1:gpus=1:titan`
  
  1 node with 1 core and 1 GPU, specifically a Titan Black GPU

- `-l nodes=1:ppn=1:gpus=1:k80`
  
  1 node with 1 core and 1 GPU, specifically an Nvidia K80 GPU

- `-l nodes=1:ppn=4:gpus=4:titan`
  
  1 node with 4 Titan GPUs. Note that we request ppn=4 too, it is always best to request at least as many CPU cores are GPUs

The available GPU node configurations are shown [here](#).

When you request GPUs, the system will set two environment variables - we strongly recommend you **do not change** these:

- `CUDA_VISIBLE_DEVICES` has a comma-separated list of the device IDs this job is allowed to use (e.g. "2,3"). The CUDA library within the application will use this to prevent multiple GPU jobs on the same node from interfering with each other.

- `CUDA_DEVICES` has a zero-based sequence of the "logical device IDs" for your job (e.g. "0 1"). So, if your application expects a list of GPU IDs starting at zero, and you have been allocated GPU numbers 2 and 3, then you can pass `$CUDA_DEVICES` to your application and it will see 2 devices, named "0" and "1", which happen to correspond (via `$CUDA_VISIBLE_DEVICES`) to the GPUs whose physical IDs are "2" and "3"

To your application, it will look like you have GPU 0, 1... (up to as many GPUs as you requested). So if for example, you request 2 GPUs, and are allocated GPU 2 and GPU 3, you will have:

```bash
# echo $CUDA_VISIBLE_DEVICES
2,3
# echo $CUDA_DEVICES
0,1
```

Now if your application calls "cudaSetDevice(0)", you will use the **GPU that appears as device 0, but is actually device 2**.

And a call to "cudaSetDevice(3)" will return an error, because as far as the application can see, the node only has 2 GPUs, numbered 0 and 1.