## Requesting resources via Slurm

### Quick Links

<table>
<thead>
<tr>
<th>Category</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HPC Home</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Getting an account</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Getting started on Prince</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Prince How-to Articles</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Logging in</strong></td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td>Mac / Linux</td>
<td></td>
</tr>
<tr>
<td><strong>Clusters and Storage</strong></td>
<td></td>
</tr>
<tr>
<td>Prince (HPC)</td>
<td></td>
</tr>
<tr>
<td>Dumbo (Hadoop)</td>
<td></td>
</tr>
<tr>
<td>Brooklyn (OpenStack)</td>
<td></td>
</tr>
<tr>
<td>Dalma (NYU Abu Dhabi)</td>
<td></td>
</tr>
</tbody>
</table>
Transferri ng data to/from the clusters
Transferri ng data to/from Prince cluster using Globus
Submittin g jobs with sbatch
Available software
Licensed Software Available on the HPC Cluster
Building Software
Slurm Tutorial
Tutorials
FAQs
Scratch Area Cleanup
Programming for Biologist s
Acknowledg e Statement
Research Gallery
HPC People
# Running jobs on the Prince Cluster

## Accessing the Prince Cluster
- From Windows workstation
- From Mac workstation

## Software and Environment Module

## Job script and resource request
- Introduction to job scheduling
- Submitting jobs with sbatch
- Requesting resources
- Using computing nodes interactively

## Monitoring batch jobs
- Monitoring batch jobs - squeue
What is running and where?

slurmtop

Canceling your jobs

Compiling your own software

Putting all pieces together

An Amber example

A R example

Summary

At a minimum, the scheduler needs to know:

- How many CPU cores you need, and whether they must be on the same node
  - If you don’t know, the answer is probably “1 core”. If the program supports parallel, it probably supports multithreading - multiple cores on a single node. To use multiple nodes, a program generally needs MPI, so you should only request multiple nodes if you are sure the program can use them.
- How much memory you need
  - NYU has nodes with 60GB, 124GB and 252GB available to jobs. (The remaining memory is needed by the operating system)
- How long the job is expected to take
  - NYU HPC users can request up to 168 hours (1 week) for a single job. But priority is given to jobs requesting less time

**CPUs - nodes and cores**

HPC is associated with parallel processes - but it's not magic! To use multiple CPUs, the program must have been written with either threading (e.g. OpenMP) or message passing (MPI).
How much do I need?

The HPC cluster is not magic, its CPUs are only as fast as any other contemporary CPU. In fact, some nodes on Prince are a few years old, and may be slower than your desktop (see Clusters July 2017 for a table of node types and when we installed them).

The performance of the HPC cluster comes from its scale. Most nodes in the cluster have 12 or 20 cores, 48GB up to 192GB of RAM, access to a large fast parallel filesystem and there is a 40Gb/s dedicated network link between any two nodes in each of the main groups. And there are thousands of nodes.

So although the resources your job needs depend very much on your job, and there is no simple rule for estimating requirements, you can make some initial guesses based on why you need the HPC cluster:

- My desktop has not enough RAM
  You should request at least as much RAM as your desktop possesses. The time required will probably be similar to the time required on your desktop. Be aware though that many problems scale with $O(n^2)$ or more, so doubling the number of data points might require 4x the RAM and 8x the compute time.
- Each run takes 4 hours on my 32GB desktop, and I have 1000 experiments to run
  Each experiment will probably take 32GB of memory and 4 hours on the HPC cluster too - but you can submit 1000 of them at once and a few hundred might run simultaneously

For a few one-off jobs, you can safely request much more than you need. When jobs are done, run `sacct` to check actual usages (`un 'sacct -j <jobID> -i'` would show much more information).:
```
$ sacct -j 6891 -oJobID,JobName,ExitCode,NNodes,NCPUS,MaxRSS,Elapsed

<table>
<thead>
<tr>
<th>JobID</th>
<th>JobName</th>
<th>ExitCode</th>
<th>NNodes</th>
<th>NCPUS</th>
<th>MaxRSS</th>
<th>Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>6891</td>
<td>mpiexec-t+</td>
<td>0:0</td>
<td>3</td>
<td>84</td>
<td></td>
<td>00:41:21</td>
</tr>
<tr>
<td>6891.batch</td>
<td>batch</td>
<td>0:0</td>
<td>1</td>
<td>28</td>
<td>647076K</td>
<td>00:41:21</td>
</tr>
<tr>
<td>6891.extern</td>
<td>extern</td>
<td>0:0</td>
<td>3</td>
<td>84</td>
<td></td>
<td>00:41:21</td>
</tr>
<tr>
<td>6891.0</td>
<td>orted</td>
<td>0:0</td>
<td>2</td>
<td>14</td>
<td>639040K</td>
<td>00:41:19</td>
</tr>
</tbody>
</table>
```

From which you can deduce that the job took 42 minutes of wall clock time and about 647MB of memory. So a sensible resource request for the next job is (see RESOURCES for more about the options):

```
$ sbatch -t1:00:00 --mem=1000 ... testme.s
```

**Requesting resources**

Slurm schedules jobs based on the resources they expect to need. The default at NYU is 1 CPU core and 2GB of memory for 1 hour. To specify what your job will actually need, use these #SBATCH option:

**Options to request compute resources:**

- `--nodes` - number of nodes
- `--ntasks-per-node` - number of tasks per node
- `--cpus-per-task` - number of CPU core per task required. Default is, one node, one task per node, one CPU core per task, so is 1 CPU core on 1 node
- `--mem` - memory required per node in MegaBytes
- `--mem-per-cpu` - memory required per CPU in MegaBytes
- `--time` - maximum run time of a job. Time formats include "minutes", "minutes:seconds", "hours:minutes:seconds", "days-hours", "days-hours:minutes" and "days-hours:minutes:seconds".

A job submitted without requesting any specific resources will get the default resource limits for the default partition.

```
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.
```

A small amount of memory on each node is needed by the operating system, so for example on a 128GB node, only about 125GB is available to jobs. A job requesting 128GB of memory will therefore be too big for a 128GB node, and Slurm will schedule it on other nodes with larger memory instead. We have fewer nodes with so much memory, so the job is likely to spend longer waiting in the queue. Tip: try requesting 125GB instead.

```
Nodes on Prince have memory of 62 GB, 125 GB and 250 GB. Requesting a large portion of the memory on a node will cause Slurm to reserve an entire node for your job even if you only request 1 CPU, since there will be insufficient remaining memory to run other jobs.
```

The serial queues on NYU HPC clusters are limited to a single node, but allow multiple processors on that node to be used. Therefore, parallel jobs using only one node, such as OpenMP or multithreaded jobs can be submitted to a serial queue.
When using more than one node, the job script is executed only on the first node. To make use of the other nodes you must use MPI.

**Requesting GPUs**

To request GPU nodes:

- `--gres=gpu:1`
  1 node with 1 core and 1 GPU card
- `--gres=gpu:2 -c2`
  1 node with 2 cores and 2 GPU cards
- `--gres=gpu:k80:3 -c3`
  1 node with 3 cores and 3 GPU cards, specifically the type of Tesla K80 cards. Note that 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`
- `GPU_DEVICE_ORDINAL`

To your application, it will look like you have GPU 0,1,... (up to as many GPUs as you requested). So if for example, there are two jobs from different users: the first one requesting 1 GPU card, the second 3 GPU cards, and they happen landing on the same node `gpu-08`:

The 1st job

```
$ env | grep DEV
CUDA_VISIBLE_DEVICES=0
GPU_DEVICE_ORDINAL=0

$ nvidia-smi -L
GPU 0: Tesla K80 (UUID: GPU-c448f648-2759-5ec0-0fde-0b72a857cd09)
```

The 2nd job

```
$ env | grep DEV
CUDA_VISIBLE_DEVICES=0,1,2
GPU_DEVICE_ORDINAL=0,1,2

$ nvidia-smi -L
GPU 0: Tesla K80 (UUID: GPU-6e023c21-2644-aec5-5aa9-78a0874bda48)
GPU 1: Tesla K80 (UUID: GPU-f2735f79-615d-bf15-00eb-54b976b944b8)
GPU 2: Tesla K80 (UUID: GPU-b9bf5b2d-d7d5-85aa-75a5-68d0b5532a4e)
```

The above boxes shown that there is an overlapped value of '0' between the two jobs for the two environment variables. But the GPU cards assigned are actually different for the jobs, as we should trust UUIDs the nvidia-smi command display to us.
Exercise
Try submitting a simple job. Here's an example script:

```bash
#!/bin/bash
#SBATCH -N1
#SBATCH -n1
#SBATCH -t5:00
#SBATCH --mem=500
mkdir -p $SCRATCH/my_run_dir
cd $SCRATCH/my_run_dir
echo "running a job on node $(hostname) in directory $(pwd)"
sleep 120
echo "finishing"
```