GPU jobs

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

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 (eg "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 (eg "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.