Handout for HiPerGator:

SLURM Scripts for MPI Jobs

Presented 2/2/2017 by Matt Gitzendanner: magitz@ufl.edu

The SLURM Scheduler

- SLURM is the scheduling software that controls all the jobs that run on the HiPerGator cluster.
- Users need to tell SLURM what resources are needed to run each job, and then SLURM takes care of figuring out where to run the job on the compute resources
- The information SLURM needs is:
  - How many CPUs you want and how you want them grouped
  - How much RAM your job will use
  - How long your job will run
  - The commands that will be run

SLURM CPU Requests for Parallel Applications

- Applications can be parallelized in different ways, knowing which method was used for your application is critical
- Common categories of parallelization:
  - OpenMP, Threaded, Pthreads
    - All cores on one sever, shared memory
  - MPI
    - Can use multiple servers

SLURM CPU Request for Threaded Applications

- For threaded applications, all cores need to be on a single node
- The request could look like:

```
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
```

This would request 1 node (physical computer) with 1 task and 8 cores for that task—so 8 cores all together for the application to use.

SLURM Terminology and Directives

- Each server is referred to as a node and is associated with the --nodes directive
- A task corresponds with MPI ranks and is associated with the --ntasks directive
- Each server has 2-4 physical CPUs, referred to as sockets and these are associated with the --ntasks-per-node and --ntasks-per-socket directives.
- Each task can use one or more cores (threads) and is associated with the --cpus-per-task directive
- The memory for MPI applications should be requested with the --mem-per-cpu directive, where CPU is actually core in this case.
- Lastly, the --distribution directive controls how processes are distributed across nodes and sockets.

SLURM CPU Requests for MPI Applications

- Here is an example of an MPI job resource request:

```
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16
#SBATCH --ntasks-per-socket=8
#SBATCH --distribution=cyclic:cyclic
#SBATCH --constraint=infiniband
srun --mpi=pmi2 myApp
```

This would request 16 tasks, corresponding to 16 MPI ranks. These will each have 1 core. All 16 tasks will be constrained to be on a single server. Each socket (assuming a HiPerGator 2 node with 2 sockets) will have 8 tasks and the tasks will be distributed cyclically on the cores within the nodes. The --constraint=infiniband, while not technically needed since all HiPerGator 2 nodes have infiniband will ensure that if you run in a partition with nodes that don't have infiniband, this job would only run on infiniband connected nodes.

Note that we launch the application with "srun --mpi=pmi2" and not mpirun or mpiexec. This helps to ensure that the process mapping is consistent between SLURM and MPI.

- A second example :

```
#SBATCH --ntasks=32
#SBATCH --cpus-per-task=1
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=16
#SBATCH --ntasks-per-socket=8
#SBATCH --distribution=cyclic:cyclic
#SBATCH --constraint=infiniband
srun --mpi=pmi2 myApp
```

In this second example, 32 tasks are requested, and these are split between 2 nodes, 16 tasks on each. Again, the 16 tasks on each node are split 8 per socket. Being explicit down to the socket level ensures consistent mapping of processes to cores between SLURM and MPI and prevents time-slicing caused by incorrect mapping.

SLURM CPU Requests for Hybrid OpenMP/MPI Jobs

- Here is an example of a Hybrid OpenMP/MPI job resource request:

```
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=4
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4
#SBATCH --ntasks-per-socket=2
#SBATCH --distribution=cyclic:cyclic
#SBATCH --constraint=infiniband
srun --mpi=pmi2 myApp
```

In this case, we are running an application that uses multiple core for each MPI task or rank. We specify 16 tasks and 4 cores per task, for a total of 64 cores. The 16 tasks are split across 4 nodes, with the --ntasks-per-node ensuring that they are distributed evenly across the nodes. Each socket on each node will get 2 tasks, using a total of 8 cores on each socket.

The distribution here is cyclic for tasks on nodes, but grouped (fcyclic) for the cores of each tasks with-in a node. This ensures that the cores for each task are within the same socket.

Additional Considerations

- It is important to test scaling--how well does the application perform as you increase the num ber of processors.
    - If an application runs in 2 hours on 8 cores and only speeds up to 1.5 hours on 24 cores, is that a reasonable use of resources?
- Memory requests should avoid the --mem directive, instead using the --mem-per-cpu directive.
- There are currently 2 compute partitions on HiPerGator
    - hpg2-compute (the default)
        - 30,000 cores
        - Nodes have 2 sockets with 16 cores each
        - Nodes have about 120 GB RAM available
    - hpg1-compute
        - ~16,000 cores
        - Specify with: -p hpg1-compute
        - Several types of nodes:
            - Most:

                - Nodes have 4 sockets with 16 cores each

                - Nodes have 250 GB RAM available
            - Several other configurations, not all are infiniband connected

                - See wiki page for sinfo command usage to view node features