Tips about Job Submission

Last update: Jul 22, 2024

Basics

- Samples for each application can be found at /apl/(application name)/(version/revision)/samples directory. Those sample files can be used as template files for your own software.

Jobscript skeleton for /bin/sh or /bin/bash:

```
#!/bin/sh
#PBS -l select=...
#PBS -l walltime=(walltime; 24:00:00 for example)

# PBS_O_WORKDIR corresponds to the working directory when you submit the job.
# If you want to use the same working directory when running the job on the computation node(s),
# you need to "cd" to the directory beforehand.
if [ ! -z ${PBS_O_WORKDIR} ]; then
  cd ${PBS_O_WORKDIR} # if PBS_O_WORKDIR exists, cd to that dir.
fi

(actual commands; file copy, setenv, run, etc.)
```

Joscript skeleton for csh/tcsh:

```
#!/bin/csh -f
#PBS -l select=...
#PBS -l walltime=(walltime; 24:00:00 for example)

if ( $?PBS_O_WORKDIR ) then
  cd $PBS_O_WORKDIR # same as sh/bash case.
endif

(actual commands; file copy, setenv, run, etc.)
```

Sample Job Header (select line)

- The number following select= represents the number of nodes (1 if omitted).
- Other numbers (ncpus, mpiprocs, ompthreads, ngpus) are the resource amounts per node.
- OMP_NUM_THREADS environment variable is automatically set to the number specified by ompthreads.
- If you employ one of MPI environments installed under /apl (OpenMPI, IntelMPI, MVAPICH), you don't need to specify hostlist (or machinefile) in mpirun argument.
  - "mpirun -np (number specified for "mpiprocs") command options" will work.
  - (If you add --with-tm=/apl/pbs/22.05.11 option to "configure" upon your OpenMPI build, you may not need to specify hostlist.)
- When you request GPUs, you don' need to pay special attention on CUDA_VISIBLE_DEVICES environment variable. (The job server will handle resources wisely.)
  - (Some software may need a special setting when multiple GPU cards are employed.)

1 vnode, for each vnode: 64 CPU cores, MPI*64, no OpenMP (Flat MPI) (1 node and 64 CPU cores in total), 72 hours (= 3 days) (jobtype=vnode implicitly)

```
#PBS -l select=1:ncpus=64:mpiprocs=64:ompthreads=1
#PBS -l walltime=72:00:00
```

jobtype=small, 4 nodes, for each node: 128 CPU cores, MPI*128, no OpenMP (4 nodes and 512 CPU cores in total), 168 hours (= a week) (jobtype=vnode implicitly)

```
#PBS -l select=4:ncpus=128:mpiprocs=128:ompthreads=1
#PBS -l walltime=168:00:00
```

jobtype=largemem, 1 node, for each node: 128 CPU cores, MPI*64, OpenMP*2 (1 node and 128 CPU cores in total), 30 minutes

```
#PBS -l select=1:ncpus=128:mpiprocs=64:ompthreads=2:jobtype=largemem
#PBS -l walltime=00:30:00
```

jobtype=largemem, 2 vnodes, for each vnode: 64 CPU cores, MPI*64, no OpenMP (2 vnodes and 128 CPU cores in total), 1 hour

```
#PBS -l select=2:ncpus=64:mpiprocs=64:ompthreads=1:jobtype=largemem
#PBS -l walltime=01:00:00
```

This job may be performed on two separate nodes or on only one node. For the example one above, that will run on only one node.

1 CPU core, 168 hours (jobtype=core implicitly)

```
#PBS -l select=1:ncpus=1:mpiprocs=1:ompthreads=1
#PBS -l walltime=168:00:00
```

12 CPU cores, MPI*4, no OpenMP, 12 hours (jobtype=core implicitly)

```
#PBS -l select=1:ncpus=12:mpiprocs=4:ompthreads=1
#PBS -l walltime=12:00:00
```

(Available memory amount is proportional to the number of cpu cores employed. If you want to increase memory amount but not mpiprocs, this types of description is useful/necessary for jobtype=core.)

18 CPU cores, MPI*9, OpenMP*2, 3 hours (jobtype=core implicitly)

```
#PBS -l select=1:ncpus=18:mpiprocs=9:ompthreads=2
#PBS -l walltime=03:00:00
```

32 CPU cores, MPI*8, OpenMP*4, 168 hours (jobtype=core implicitly)

```
#PBS -l select=1:ncpus=32:mpiprocs=8:ompthreads=4
#PBS -l walltime=168:00:00
```

60 CPU cores, OpenMP*60, 168 hours (jobtype=core implicitly)

```
#PBS -l select=1:ncpus=60:mpiprocs=1:ompthreads=60
#PBS -l walltime=168:00:00
```

64 CPU cores, OpenMP*60, 168 hours (jobtype=vnode implicitly)

```
#PBS -l select=1:ncpus=64:mpiprocs=1:ompthreads=60
#PBS -l walltime=168:00:00
```

The calculation itself is the same as the example one above; 60 cores are used with OpenMP. However, CPU points per hour of this job is 45, smaller than the example one above (60 points/hour). This difference comes from the jobtype of these two jobs.

1 CPU core, 1 GPU, 24 hours (jobtype=gpu, implicitly)

```
#PBS -l select=1:ncpus=1:mpiprocs=1:ompthreads=1:ngpus=1
#PBS -l walltime=24:00:00
```

12 CPU cores, MPI*12, no-OpenMP, 1 GPU, 12 hours (jobtype=gpu, implicitly)

```
#PBS -l select=1:ncpus=12:mpiprocs=12:ompthreads=1:ngpus=1
#PBS -l walltime=12:00:00
```

(In case of ngpus=1, ncpus must be <= 16.)

4 nodes, for each node: 8 CPU cores, MPI*2, OpenMP*4, 2 GPUs (4 nodes, 32 CPU cores, 8 GPUs), 12 hours (jobtype=gpu, implicitly)

```
#PBS -l select=4:ncpus=8:mpiprocs=2:ompthreads=4:ngpus=2
#PBS -l walltime=12:00:00
```

(Value of "mpiprocs" must be multiple of that of "ngpus". Even if you use only 1 CPU core and multiple GPUs, your

"ncpus" specification must be equal to number of GPUs at least.)

> 2 nodes, for each node: 24 CPU cores, MPI*24, no OpenMP, 2 GPUs (2 nodes, 48 CPU cores, 4 GPUs), 24 hours (jobtype=gpu, implicitly)

```
#PBS -l select=2:ncpus=24:mpiprocs=24:ompthreads=1:ngpus=2
#PBS -l walltime=24:00:00
```

(The MPI processes might be scattered up to 4 nodes.)