Cheyenne job script examples

When you use any of these examples, remember to substitute your own job name and project code, and customize the other directives and commands as necessary. That includes the commands shown for setting TMPDIR in your batch scripts as recommended here: Storing temporary files with TMPDIR.

Specify an ompthreads value in your script to help ensure that any parallel job will run properly. This is particularly important in a couple of cases:

- If your code was compiled with the GNU or Intel compiler using the **-f openmp** option.
- If your code was compiled with a PGI compiler with option **-mp**.

For a parallel job that does not use OpenMP threads – for a pure MPI job, for example – specify **ompthreads=1** in your PBS select statement as shown below. Failure to do so may result in the job oversubscribing its nodes, resulting in poor performance or puzzling behavior such as exceeding its wallclock limit.

Load all modules that are necessary to run your program at the start of your batch scripts by including a line like this:

```
module load intel mpt
```

If you think you might run a particular compiled executable well into the future, we advise that you load specific versions of desired modules to ensure reproducibility. Follow this example:

```
module load intel/19.1.1 mpt/2.25
```

These examples are based on the following assumptions:

1. You are using HPE's Message Passing Toolkit (MPT) MPI library.
2. The programs being run were compiled with Intel 16.0.3 or a later version.

Contact the NCAR Research Computing help desk for assistance with adapting them for other cases.

When your script is ready, submit your batch job for scheduling as shown here.

Page contents

Batch script to run an MPI job

For tcsh users

```
#!/bin/tcsh
### Job Name
#PBS -N mpi_job
### Project code
#PBS -A project_code
#PBS -l walltime=01:00:00
#PBS -q queue_name
### Merge output and error files
#PBS -j oe
#PBS -k eod
### Select 2 nodes with 36 CPUs each for a total of 72 MPI processes
#PBS -l select=2:ncpus=36:mpiprocs=36:ompthreads=1
### Send email on abort, begin and end
#PBS -m abe
### Specify mail recipient
#PBS -M email_address

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run the executable
mpiexec_mpt ./executable_name.exe
```

For bash users

```
#!/bin/bash
### Job Name
#PBS -N mpi_job
### Project code
#PBS -A project_code
#PBS -l walltime=01:00:00
#PBS -q queue_name
### Merge output and error files
#PBS -j oe
#PBS -k eod
### Select 2 nodes with 36 CPUs each for a total of 72 MPI processes
#PBS -l select=2:ncpus=36:mpiprocs=36:ompthreads=1
### Send email on abort, begin and end
#PBS -m abe
### Specify mail recipient
#PBS -M email_address

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run the executable
mpiexec_mpt ./executable_name.exe
```

Batch script to run a pure OpenMP job

To run a pure OpenMP job, specify the number of CPUs you want from the node (ncpus). Also specify the number of threads (ompthreads) or OMP_NUM_THREADS will default to the value of ncpus, possibly resulting in poor performance.

You will be charged for use of all CPUs on the node when using an exclusive queue.

For tcsh users

```
#!/bin/tcsh
#PBS -A project_code
#PBS -N OpenMP_job
#PBS -j oe
#PBS -k eod
#PBS -m abe
#PBS -M email_address
#PBS -q queue_name
#PBS -l walltime=01:00:00
### Request 10 CPUS for 10 threads
#PBS -l select=1:ncpus=10:ompthreads=10

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run OpenMP program
./executable_name
```

For bash users

```
#!/bin/bash
#PBS -A project_code
#PBS -N OpenMP_job
#PBS -j oe
#PBS -k eod
#PBS -m abe
#PBS -M email_address
#PBS -q queue_name
#PBS -l walltime=01:00:00
### Request 10 CPUS for 10 threads
#PBS -l select=1:ncpus=10:ompthreads=10

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run OpenMP program
./executable_name
```

Batch script to run a hybrid MPI/OpenMP job

If you want to run a hybrid MPI/OpenMP configuration where each node uses threaded parallelism while the nodes communicate with each other using MPI, activate NUMA mode and run using the MPI launcher.

Specify the number of CPUs you want from each node (ncpus). Also specify the number of threads (ompthreads) or OMP_NUM_THREADS will default to the value of ncpus, possibly resulting in poor performance.

You will be charged for use of all CPUs on the node when using an exclusive queue.

For tcsh users

```
#!/bin/tcsh
#PBS -A project_code
#PBS -N hybrid_job
#PBS -j oe
#PBS -k eod
#PBS -m abe
#PBS -M email_address
#PBS -q queue_name
#PBS -l walltime=01:00:00
### Request two nodes, each with one MPI task and 36 threads
#PBS -l select=2:ncpus=36:mpiprocs=1:ompthreads=36

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run the hybrid OpenMP/MPI program
mpiexec_mpt omplace ./executable_name
```

For bash users

```
#!/bin/bash
#PBS -A project_code
#PBS -N hybrid_job
#PBS -j oe
#PBS -k eod
#PBS -m abe
#PBS -M email_address
#PBS -q queue_name
#PBS -l walltime=01:00:00
### Request two nodes, each with one MPI task and 36 threads
#PBS -l select=2:ncpus=36:mpiprocs=1:ompthreads=36

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run the hybrid OpenMP/MPI program
mpiexec_mpt omplace ./executable_name
```

Batch script to run a job array

Job arrays are useful when you want to run the same program repeatedly on different input files. PBS can process a job array more efficiently than it can process the same number of individual non-array jobs. The elements in a job array are known as "sub-jobs."

Before submitting this batch script: Place files input.1 through input.18 in the same directory where you have the sequential **cmd** command. The batch job specifies 18 sub-jobs indexed 1-18 that will run in the "share" queue. The Nth sub-job uses file input.N to produce file output.N. The "share" queue is recommended for running job arrays of sequential sub-jobs, or parallel sub-jobs each having from two to nine tasks. The share queue has a per-user limit of 18 sub-jobs per array.

For tcsh users

```
#!/bin/tcsh
### Job Name
#PBS -N job_arrays
### Project code
#PBS -A project_code
#PBS -l walltime=01:00:00
#PBS -q share
### Merge output and error files
#PBS -j oe
#PBS -k eod
### Select one CPU
#PBS -l select=1:ncpus=1
### Specify index range of sub-jobs
#PBS -J 1-18

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

# Execute subjob for index PBS_ARRAY_INDEX
cmd input.$PBS_ARRAY_INDEX > output.$PBS_ARRAY_INDEX
```

If you need to include a job ID in a subsequent **qsub** command as in the following example, be sure to use quotation marks to preserve the **[]** brackets:

```
qsub -W "depend=afterok:317485[]" postprocess.pbs
```

For bash users

```
#!/bin/bash
### Job Name
#PBS -N job_arrays
### Project code
#PBS -A project_code
#PBS -l walltime=01:00:00
#PBS -q share
### Merge output and error files
#PBS -j oe
#PBS -k eod
### Select one CPU
#PBS -l select=1:ncpus=1
### Specify index range of sub-jobs
#PBS -J 1-18

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

# Execute subjob for index PBS_ARRAY_INDEX
cmd input.$PBS_ARRAY_INDEX > output.$PBS_ARRAY_INDEX
```

Batch script to run a command file (MPMD) job

Multiple Program, Multiple Data (MPMD) jobs run multiple independent, sequential executables simultaneously. The executable commands appear in the command file (cmdfile) on separate lines. The command file, the executable files, and the input files should reside in the directory from which the job is submitted. If they don't, you need to specify adequate relative or full pathnames in both your command file and job scripts.

The command file used in the example job scripts has these four lines.

```
./cmd1.exe < input1 > output1
./cmd2.exe < input2 > output2
./cmd3.exe < input3 > output3
./cmd4.exe < input4 > output4
```

The job will produce output files that reside in the directory in which the job was submitted.

In place of executables, you can specify independent shell scripts, MATLAB scripts, or others, or you can mix and match executables with scripts. Each task should execute in about the same wall-clock time as the others.

If any of your command file lines invoke a utility such as IDL, MATLAB, NCL, R and so on, invoke it in batch mode rather than interactive mode or your job will hang until it reaches the specified walltime limit. See the user guide for the utility for how to invoke it in batch mode.

For tcsh users

```
#!/bin/tcsh
#PBS -A project_code
#PBS -N cmd_file
#PBS -j oe
#PBS -k eod
#PBS -m abe
#PBS -M email_address
#PBS -q queue_name
#PBS -l walltime=01:00:00
### Request one chunk with ncpus and mpiprocs set to
### the number of lines in the command file
#PBS -l select=1:ncpus=4:mpiprocs=4:ompthreads=1

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

# yyyy-mm-dd Context: Cheyenne MPT command file job.
# Do not propagate this use of MPI_SHEPHERD
# to other MPT jobs as it may cause
# significant slowdown or timeout.
# Contact the CISL Consulting Services Group
# if you have questions about this.
setenv MPI_SHEPHERD true

mpiexec_mpt launch_cf.sh cmdfile
```

For bash users

```
#!/bin/bash
#PBS -A project_code
#PBS -N cmd_file
#PBS -j oe
#PBS -k eod
#PBS -m abe
#PBS -M email_address
#PBS -q queue_name
#PBS -l walltime=01:00:00
### Request one chunk with ncpus and mpiprocs set to
### the number of lines in the command file
#PBS -l select=1:ncpus=4:mpiprocs=4:ompthreads=1

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

# yyyy-mm-dd Context: Cheyenne MPT command file job.
# Do not propagate this use of MPI_SHEPHERD
# to other MPT jobs as it may cause
# significant slowdown or timeout.
# Contact the CISL Consulting Services Group
# if you have questions about this.
export MPI_SHEPHERD=true

mpiexec_mpt launch_cf.sh cmdfile
```

Pinning tasks/threads to CPUs

The **omplace** wrapper script pins processes or threads to specific CPUs. This is particularly useful if you want to use threads to parallelize at the socket level (using 18 CPUs per socket, two sockets per node, for example), which can improve performance if your program depends strongly on memory locality.

For purposes of many Cheyenne users, using **omplace** as shown below is sufficient to ensure that processes do not migrate among CPUs and adversely affect performance. To learn about using omplace and dplace for more precise control of process placement, see Process binding.

For tcsh users

```
#!/bin/tcsh
#PBS -A project_code
#PBS -N hybrid_job
#PBS -j oe
#PBS -k eod
#PBS -m abe
#PBS -M email_address
#PBS -q queue_name
#PBS -l walltime=01:00:00
### Request two nodes, each with two MPI tasks and 18 threads per task
#PBS -l select=2:ncpus=36:mpiprocs=2:ompthreads=18

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run hybrid program with explicit task/thread placement using omplace
### to ensure that each group of 18 threads runs on a separate socket
mpiexec_mpt omplace ./executable_name
```

For bash users

```
#!/bin/bash
#PBS -A project_code
#PBS -N hybrid_job
#PBS -j oe
#PBS -k eod
#PBS -m abe
#PBS -M email_address
#PBS -q queue_name
#PBS -l walltime=01:00:00
### Request two nodes, each with two MPI tasks and 18 threads per task
#PBS -l select=2:ncpus=36:mpiprocs=2:ompthreads=18

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run hybrid program with explicit task/thread placement using omplace
### to ensure that each group of 18 threads runs on a separate socket
mpiexec_mpt omplace ./executable_name
```

Dependent jobs

It is possible to schedule jobs to run based on the status of other jobs. For example, you might schedule a preprocessing job to run; start a computation job when the preprocessing job is complete; then start a post-processing job when the computation is done.

One way to schedule such a series or chain of jobs is to use **qsub -W [job-dependency-expression]** to specify the job dependencies you need.

Example 1 below shows how to do this.
Example 2 illustrates another method.

Dependent jobs - example 1

Let's say you have you have three scripts to submit and run consecutively:

1. **pre.pbs**: a preprocessing job
2. **main.pbs**: a computation job
3. **post.pbs**: a post-processing job

The main job can be run only when the preprocessing job finishes, and the post-processing job can be run only when the computation job finishes.

Follow this example, making sure to use **backticks** as shown.

- Place the first job on hold. If it starts before the dependent jobs are submitted, the dependent jobs might never start.

```
bash users:
JID1=`qsub -h pre.pbs`

tcsh users:
set JID1=`qsub -h pre.pbs`
```

- Make starting of the second job dependent on successful completion of the first.

```
bash users:
JID2=`qsub -W depend=afterok:$JID1 main.pbs`

tcsh users:
set JID2=`qsub -W depend=afterok:$JID1 main.pbs`
```

- Make starting of the post-processing job dependent on successful completion of the second job.

```
qsub -W depend=afterok:$JID2 post.pbs
```

- Release the first job to initiate the sequence.

```
qrls $JID1
```

To monitor the jobs you submitted, use the qstat command.

```
qstat -u $USER
```

To determine why an individual job is pending, enter **qstat** followed by the **job ID**. Such a job typically would be pending because the job on which it depends has not met the required conditions.

Dependent jobs - example 2

This script starts a second job after successful completion of the initial job. When your job scripts are ready, submit the first one with the qsub command:

```
qsub script_name
```

For tcsh users

```
#!/bin/tcsh
#PBS -A project_code
#PBS -N job1
#PBS -j oe
#PBS -k eod
#PBS -m abe
#PBS -M email_address
#PBS -q queue_name
#PBS -l walltime=01:00:00

### Request 2 nodes, each with two MPI tasks and 18 threads per task
#PBS -l select=2:ncpus=36:mpiprocs=2

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run job1
mpiexec_mpt job1.exe

### Run job 2 if program runs successfully
if ( $status == 0 ) then
    qsub job2.pbs
endif
```

For bash users

```
#!/bin/bash
#PBS -A project_code
#PBS -N job1
#PBS -j oe
#PBS -k eod
#PBS -m abe
#PBS -M email_address
#PBS -q queue_name
#PBS -l walltime=01:00:00

### Request 2 nodes, each with two MPI tasks and 18 threads per task
#PBS -l select=2:ncpus=36:mpiprocs=2

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run job1
mpiexec_mpt job1.exe

### Run job 2 if program runs successfully
if [[ $? == 0 ]]; then
    qsub job2.pbs
fi
```