Example Batch Scripts

The following example batch scripts do NOT use Hyper-Threading unless explicitly mentioned. See the Edison **Hyper-Threading page** for an explanation of this technology and a discussion of its benefits.

Basic Scripts

Sample Job script

This script uses the default 24 cores per node. This job will run on 64 nodes, with 1,536 cores.

```
#PBS -q debug
#PBS -l mppwidth=1536
#PBS -l walltime=00:10:00
#PBS -N my_job
#PBS -j oe

cd $PBS_O_WORKDIR
aprun -n 1536 ./my_executable
```

Sample job script to run with Hyperthreading (HT)

With hyperthreading (HT), Edison compute nodes have 48 logical cores per node. Use the "-j 2" option of the "aprun" command to use Hyper-Threading; that's all that's required. The following job will run on 64 nodes, with 3,072 cores in total.

```
#PBS -q debug
#PBS -l mppwidth=1536
#PBS -l walltime=00:10:00
#PBS -N my_job
#PBS -j oe

cd $PBS_O_WORKDIR
aprun -j 2 -n 3072 ./my_executable
```

"Unpacked" Nodes Script

This example shows how to run a total of 768 MPI tasks using only 12 cores per node rather than all 24 using 64 nodes

```
#PBS -q regular
#PBS -l mppwidth=1536
#PBS -l walltime=12:00:00
#PBS -N my_job
#PBS -j oe

cd $PBS_O_WORKDIR
aprun -n 768 -N 12 -S 6 ./my_executable
```

Back to Top

Running Hybrid MPI/OpenMP Applications

Hybrid MPI/OpenMP Example

The -N and -d flags need to be passed to the aprun command to specify the number of cores per node to use and number of OpenMP threads to use. Notice the -S option for unpacked nodes usage, to specify the number of MPI tasks per NUMA node. The following example asks for 64 nodes to run a hybrid application with different OpenMP threads per MPI task. **Notice**

the usage a of "-cc numa_node" or "-cc none" or "-cc depth" options and comments below in the example scripts for the Intel compiled programs due to the conflict of the internal Intel thread affinity and the aprun thread affinity.

Sample job script to run hybrid applications built with **Intel compilers**:

```
#!/bin/bash -l
#PBS -q regular
#PBS -l mppwidth=1536
#PBS -l walltime=12:00:00
#PBS -N my_job
#PBS -j oe
cd $PBS_O_WORKDIR

#to run with 2 threads per task
export OMP_NUM_THREADS=2
aprun -n 768 -N 12 -S 6 -d 2 -cc depth ./my_executable

#to run with 6 threads per task
export OMP_NUM_THREADS=6
aprun -n 256 -N 4 -S 2 -d 6 -cc depth ./my_executable

#to run with 12 threads per task
export OMP_NUM_THREADS=12
aprun -n 128 -N 2 -S 1 -d 12 -cc numa_node ./my_executable

#to run with 24 threads per task
export OMP_NUM_THREADS=24
aprun -n 64 -N1 -d 24 -cc none ./my_executable
```

Note, it is very important to use the correct **-cc option** for an hybrid application built with an Intel compiler to avoid performance penelty due to the conflict of the internal Intel thread affinity and the aprun thread affinity. See aprun's man page for more information about the options used (type man aprun)

Sample job script to run hybrid applications built with **Cray or GNU compilers:**

```
#!/bin/bash -l
#PBS -q regular
#PBS -l mppwidth=1536
#PBS -l walltime=12:00:00
#PBS -N my_job
#PBS -j oe

cd $PBS_O_WORKDIR

#for Cray or GNU builds:

#to run with 2 threads per task
export OMP_NUM_THREADS=2
aprun -n 768 -N 12 -S 6 -d 2 ./my_executable

#to run with 6 threads per task
export OMP_NUM_THREADS=6
aprun -n 256 -N 4 -S 2 -d 6 ./my_executable

#to run with 12 threads per task
export OMP_NUM_THREADS=12
```

```
aprun -n 128 -N 2 -S 1 -d 12 ./my_executable

#to run with 24 threads per task
export OMP_NUM_THREADS=24
aprun -n 64 -N1 -d 24 ./my_executable
```

Hybrid MPI/OpenMP Example with Hyperthreading

The -N and -d flags need to be passed to the aprun command to specify the number of cores per node to use and number of OpenMP threads to use. Notice the -S option for unpacked nodes usage, to specify the number of MPI tasks per NUMA node. The following example asks for 64 nodes, 2 MPI tasks per node, 24 OpenMP threads per MPI task, with HT.

```
#PBS -q regular
#PBS -l mppwidth=1536
#PBS -l walltime=12:00:00
#PBS -N my_job
#PBS -j oe

cd $PBS_O_WORKDIR
setenv OMP_NUM_THREADS 24

# for Intel compiled programs
# the "-cc numa_node" option should be used if the number of threads is less
   than or equal 24 with Hyperthreading
# (note: use "-cc none" instead for other number of threads)
aprun -j 2 -n 128 -N 2 -S 1 -d 24 -cc numa_node ./my_executable
# for Cray or GNU compiled programs
# aprun -j 2 -n 128 -N 2 -S 1 -d 24 ./my_executable
```

Pure OpenMP Example

Make sure to compile your application with the the appropriate OpenMP compiler flags.

```
#PBS -q regular
#PBS -l mppwidth=24
#PBS -l walltime=12:00:00
#PBS -N my_job
#PBS -j oe

cd $PBS_O_WORKDIR
setenv OMP_NUM_THREADS 24
# for Intel compiled programs
# the "-cc none" option should be used if the number of threads is larger
   than 12
aprun -n 1 -N 1 -d 24 -cc none ./my_executable
# for GNU or Cray compiled programs
aprun -n 1 -N 1 -d 24 ./my_executable
```

Pure OpenMP Example with Hyperthreading

With HT, you can run pure OpenMP with up to 48 threads. Make sure to compile your application with the the appropriate OpenMP compiler flags.

```
#PBS -q regular
#PBS -l mppwidth=24
#PBS -l walltime=12:00:00
#PBS -N my_job
#PBS -j oe

cd $PBS_O_WORKDIR
setenv OMP_NUM_THREADS 48
```

```
# for Intel compiled programs
# the "-cc none" option should be used if the number of threads is larger
   than 24 with hyperthreading
aprun -j 2 -n 1 -N 1 -d 48 -cc none ./my_executable
# for GNU or Cray compiled programs
aprun -j 2 -n 1 -N 1 -d 48 ./my_executable
```

The correct process/memory/thread affinity must be used for application binaries that are compiled with the Intel compilers. We recommend that you experiment with the code posted here, which can tell you where your tasks/threads are placed on the node.

Back to Top

Running Dynamic and Shared Library Applications

System Supported Dynamic and Shared Library Script

Note the environment variable CRAY_ROOTFS which must be set and codes must be compiled with the -dynamic flag.

```
#PBS -q regular
#PBS -l mppwidth=144
#PBS -l walltime=12:00:00
#PBS -N my_job
#PBS -e my_job.$PBS_JOBID.err
#PBS -o my_job.$PBS_JOBID.out

cd $PBS_O_WORKDIR
setenv CRAY_ROOTFS DSL
aprun -n 144 ./my_executable
```

Back to Top

Running Multiple Parallel Jobs Sequentially

```
#PBS -q regular
#PBS -l mppwidth=96
#PBS -l walltime=12:00:00
#PBS -N my_job
#PBS -e my_job.$PBS_JOBID.err
#PBS -o my_job.$PBS_JOBID.out

cd $PBS_O_WORKDIR
aprun -n 96 ./a.out
aprun -n 96 ./b.out
aprun -n 96 ./c.out
```

Back to Top

Running Multiple Parallel Jobs Simultaneously

Be sure to specify the total number of nodes needed to run all jobs at the same time. Note that multiple executables cannot be share the same nodes. If the required number of cores to launch an aprun command is not divisible by 24, an extra node needs to be added for each aprun command. In this example, the first executable needs 2 nodes, the second executable needs 5 nodes, and the last executable needs 2 nodes. The mppwidth requested is the total of 9 nodes * 24 cores/node = 216.

Notice the "&" at the end of each aprun command. Also the "wait" command at the end of the script is very important. It makes sure the batch job won't exit before all the simultaneous apruns are completed. Please limit the number of simultaneous apruns less than 50, as too many simultaneous apruns could overload the MOM node from which your job and other users' jobs get launched.

```
#PBS -q regular
#PBS -l mppwidth=216
#PBS -l walltime=12:00:00
#PBS -N my_job
```

```
#PBS -e my_job.$PBS_JOBID.err
#PBS -o my_job.$PBS_JOBID.out

cd $PBS_O_WORKDIR
aprun -n 44 ./a.out &
aprun -n 108 ./b.out &
aprun -n 40 ./c.out &
wait
```

Back to Top

Running MPMD (Multiple Program Multiple Data) Jobs

Note that more than one executable cannot be run on a given node, so make sure to add an extra node for each executable for which the number of cores needed is not divisible by 24. See example below. The executable a.out needs 2 nodes (36 tasks / 24 cores per node + 1). The executable b.out needs 3 nodes (60 tasks / 24 cores per node + 1). mppwidth is set to (2 nodes + 3 nodes) * 24 cores per node = 120 cores.

```
#PBS -q regular
#PBS -l mppwidth=120
#PBS -l walltime=02:00:00
#PBS -N my_job
#PBS -e my_job.$PBS_JOBID.err
#PBS -o my_job.$PBS_JOBID.out

cd $PBS_O_WORKDIR
aprun -n 36 ./a.out : -n 60 ./b.out
```

Please note that the SPMD components (a.out and b.out above) share MPI_COMM_WORLD. So this run method is not for running multiple copies of the same application simultaneously, just to increase throughput.

Back to Top

Serial Jobs

Using the "serial" queue

Nodes used for the "serial" queue are shared among multiple users. Jobs in the "serial" queue are charged by core, instead of the entire node. Please see **details on how to compile and run**.

Using the regular queues

Serial jobs can run in the regular queues by requesting 1 core only. Nodes in the regular queues (regular, debug, premium, low, etc,) are not shared. Jobs will be charged for the entire node, even only 1 core is sued. A sample script is as follows:

```
#PBS -q regular
#PBS -l mppwidth=1
#PBS -l walltime=02:00:00
#PBS -N my_job
#PBS -e my_job.$PBS_JOBID.err
#PBS -o my_job.$PBS_JOBID.out

cd $PBS_O_WORKDIR
aprun -n 1 ./my_executable
```

Back to Top

Running xfer Jobs

The intended use of the xfer queue is to transfer data between Edison and **HPSS**. The xfer jobs run on one of the login nodes, edison06, so they are free of charge. If you want to transfer data to the archive system (HPSS) at the end of a job, you can submit an xfer job at the end of your batch job script, so that you will not get charged for the duration of the data trasfer. See the sample job script below:

```
#PBS -q xfer
```

```
#PBS -l walltime=12:00:00
#PBS -N my_transfer
#PBS -j oe

cd $PBS_O_WORKDIR

#Archive run01 to HPSS
htar -cvf run01.tar run01
```

Note, there is no "#PBS -l mppwidth" line in the above example. The xfer jobs specifying "-l mppwidth" will be rejected at submission time.

Back to Top

STDOUT and STDERR

While your job is running, standard output (STDOUT) and standard error (STDERR) are written to temporary files in your submit directory (for example: 164894.edique02.ER and 164894.edique02.OU). The system will append to these files in real time as the job runs so you can check the contents of these files for easier job monitoring. If you merge the stderr/stdout via "#PBS -j eo" or "#PBS -j oe" option, then only one such spool file will appear. IMPORTANT: Do not remove, rename, or otherwise perturb these spool files while the job is still running!

After the batch job completes, the above files will be renamed to the corresponding stderr/stdout files (for example: jobscript.e164894 and jobscript.o164894). If you rename your own stdout/stderr file names, or merge stderr file to stdout file (with Torque keyword) and redirect your output to a file as follows, the temporary file names will be renamed to the file names of your choice. For example, if you have:

```
...
#PBS -j oe
...
aprun -n 48 ./a.out >& my_output_file          (for csh/tcsh)
or:  aprun -n 48 ./a.out > my_output_file 2>&1      (for bash)
```

Then the temporary files will be copied to "my_output_file" instead of the "jobscript.o164894" at job completion time.

Back to Top

Job Steps and Dependencies

There is a qsub option -W depend=dependency_list or a Torque Keyword #PBS -W depend=dependency_list for job dependencies. The most commonly used dependency_list would be afterok:jobid[:jobid...], which means the job just submitted will be executed only after the dependent job(s) have terminated without an error. Another option would be afterany:jobid[:jobid...], which means the job just submitted will be executed only after the dependent job(s) have terminated either with or without an error. The second option could be useful in many restart runs since it is the user's intention to exceed wall clock limit for the first job.

Note that the job id in the "-W depend=" line, must be in the format of a complete job (jobid@torque_server), such as 164894.edique02@edique02.

For example, to run batch job2 only after batch job1 succeeds,

```
edison% qsub job1
164894.edique02

edison06% qsub -W depend=afterok:164894 job2
or
edison06% qsub -W depend=afterany:164894 job2
```

or:

```
edison06% qsub job1
164894.edique02
```

```
edison06% cat job2
#PBS -q regular
#PBS -l mppwidth=24
#PBS -l walltime=0:30:00
#PBS -W depend=afterok:164894.edique02
#PBS -j oe

cd $PBS_O_WORKDIR
aprun -n 24 ./a.out
```

```
edison06% qsub job2
```

The second job will be in batch "Held" status until job1 has run successfully. Note that job2 has to be submitted while job1 is still in the batch system, either running or in the queue. If job1 has exited before job2 is submitted, job2 will not be released from the "Held" status.

It is also possible to submit the second job in its dependent job (job1) batch script using Torque keyword "$PBS_JOBID":

```
#PBS -q regular
#PBS -l mppwidth=24
#PBS -l walltime=0:30:00
#PBS -j oe

cd $PBS_O_WORKDIR
qsub -W depend=afterok:$PBS_JOBID job2
aprun -n 24 ./a.out
```

Please refer to qsub man page for other -W depend=dependency_list options including afterany:jobid[:jobid...], afternotok:jobid[:jobid...], before:jobid[:jobid...], etc.

Sample Scripts for Submitting Chained Dependency Jobs

Below is a simple batch script, 'runit', for submitting three chained jobs in total (job_number_max=3). It sets the job sequence number (job_number) to 1 if this variable is undefined (that is, in the first job). When the value is less than job_number_max, the current job submits the next job. The value of job_number is incremented by 1, and the new value is provided to the subsequent job.

```
#!/bin/bash
#PBS -q regular
#PBS -l mppwidth=1
#PBS -l walltime=0:05:00
#PBS -j oe

 : ${job_number:="1"}           # set job_nubmer to 1 if it is undefined
 job_number_max=3

 cd $PBS_O_WORKDIR

 echo "hi from ${PBS_JOBID}"

 if [[ ${job_number} -lt ${job_number_max} ]]
 then
   (( job_number++ ))
   next_jobid=$(qsub -v job_number=${job_number} -W
      depend=afterok:${PBS_JOBID} runit)
   echo "submitted ${next_jobid}"
 fi
```

```
20.
21.  sleep 15
22.  echo "${PBS_JOBID} done"
```

Using the above script, three batch jobs are submitted.