Running applications on the Cray XC30

Running on compute nodes

By default, users do not access compute nodes directly.

Instead they launch jobs on compute nodes using one of three available modes:

1. **Extreme Scaling Mode (ESM Mode)**
   The default high performance MPP mode
2. **Pre/Post Processing Mode**
   A throughput mode designed to allows efficient Multiple Applications Multiple User (MAMU) access for smaller applications.
3. **Cluster Compatibility Mode (CCM)**
   Emulation mode designed primarily to support 3rd party ISV applications which cannot be recompiled.

Extreme Scaling Mode (ESM)

ESM is a high performance mode designed to run larger applications at scale. Important features are:

- **Dedicated compute nodes for each user job**
  - No interference from other users sharing the node
  - Minimum quanta of compute is 1 node.
  - Nodes running with low-noise, low-overhead CNL OS
- **Inter-node communication is via native Aries API.**
  - The best latency and bandwidth comms are available using ESM.
  - Applications need to be linked with a Cray comms libraries (MPI, Shmem etc) or compiled with Cray language support (UPC, Coarrays)
- **The appropriate parallel runtime environment is automatically set up between nodes**

ESM is expected to be the default mode for the majority of applications that require at least one node to run.

Glossary of terms

PE/Processing Element

- A discrete software process with an individual address space. One PE is equivalent to:

1 MPI Rank, 1 Coarray Image, 1 UPC Thread, or 1 SHMEM PE

Threads

- A logically separate stream of execution inside a parent PE that shares the same address space

CPU

- The minimum piece of hardware capable of running a PE. It may share some or all of its hardware resources with other CPUs
  Equivalent to a single “Intel Hyperthread”

Compute Unit

- The individual unit of hardware for processing, may be seen described as a “core”. Each unit may provide multiple CPUs.

Launching ESM Parallel applications

- **ALPS : Application Level Placement Scheduler**
- **aprun is the ALPS application launcher**
  - It **must** be used to run application on the XC compute nodes in ESM mode, (either interactively or as a batch job)
  - If `aprun` is not used, the application will be run on the MOM node (and will most likely fail).
  - `aprun` launches sets of PEs on the compute nodes.
  - `aprun` man page contains several useful examples
  - The 4 most important parameters to set are:

Description	Option
Total Number of PEs used by the application	-n
Number of PEs per compute node	-N
Number of threads per PE (More precise, the “stride” between 2 PEs on a node)	-d
Number of to CPUs to use per Compute Unit	-j

Running applications on the Cray XC30: Some basic examples

Assuming an XC30 node with 12 core Ivybridge processors

- Each node has: 48 CPUs/Hyperthreads and 24 Compute Units/cores

- **Launching a basic MPI application:**
  - Job has 1024 total ranks/PEs, using 1 CPU per Compute Unit meaning a maximum of 24 PEs per node.

```
$ aprun -n 1024 -N 24 -j1 ./a.out
```

- **To launch the same MPI application but spread over twice as many nodes**

```
$ aprun -n 1024 -N 12 -j1 ./a.out
```

  - Can be used to increase the available memory for each PE

- **To use all availble CPUs on a single node**
  - (maximum now 48 PEs per node)

```
$ aprun -n 1024 -N 48 -j2 ./a.out
```

Some examples of hybrid invocation

- **To launch a Hybrid MPI/OpenMP application:**
  - 1024 total ranks, using 1 CPU per Compute Unit (Max 24 Threads)
  - Use 4 PEs per node and 6 Threads per PE
  - Threads set by exporting `OMP_NUM_THREADS`

```
$ export OMP_NUM_THREADS=6
$ aprun -n 1024 -N 4 -d $OMP_NUM_THREADS -j1 ./a.out
```

- **Launch the same hybrid application with 2 CPUs per CU**
  - 1024 total ranks, using 2 CPU per Compute Unit (Max 48 Threads)
  - Use 4 PEs per node and 12 Threads per PE

```
$ export OMP_NUM_THREADS=12
$ aprun -n 1024 -N 4 -d $OMP_NUM_THREADS -j2 ./a.out
```

Much more detail in later session on advance placement and binding.

Requesting resources from a batch system

- **Most Cray XC30s are controlled by batch systems.**
  - Users submit batch job scripts to a scheduler from a login node (e.g. PBS) for execution at some point in the future.
    Each job requests resources and predicts how long it will run.
  - The scheduler (running on an external server) then chooses which jobs to run and when, allocating appropriate resources at the start.
  - The batch system will then execute a copy of the user's job script on an a one of the "MOM" nodes.
  - The scheduler monitors the job throughout it lifetime. Reclaiming the resources when job scripts finish or are killed for overrunning.
- **Each user job scripts will contain two types of command**
  1. Serial commands that are executed by the MOM node, e.g.
     - quick setup and post processing commands e.g. (`rm`, `cd`, `mkdir` etc)
  2. Parallel launches to run on the allocated compute nodes.
     1. Launched using the `aprun` command.

Requesting resources from PBS

Jobs provide a list of requirements as #PBS comments in the headers of the submission script, e.g.

```
#PBS -l walltime=12:00:00
```

These can be overriden or suplemented as submission by adding to the qsub command line, e.g.

```
> qsub -l walltime=11:59:59 run.pbs
```

Common options include:

Option	Description
`-N <name>`	A name for job,
`-q <queue>`	Submit job to a specific queues.
`-o <output file>`	A file to write the job's stdout stream in to.
`--error <error file>`	A file to write the job's stderr stream in to.
`-j oe`	Join stderr stream in to stdout stream as a single file
`-l walltime=<HH:MM:SS>`	Maximum wall time job will occupy
`-A <code>`	Account to run job unders (for controlling budgets)

Requesting parallel resources

Jobs must also request “chunks” of nodes:

This is done using the select option, e.g.

```
-l select=<numnodes>:mpiprocs=<pepn>: \
    ncpus=<ppn>:ompthreads=<threads>
```

Option	Description
`select=<numnodes>`	Requests `<numnodes>` nodes from the system.
`mpiprocs=<pepn>`	Requests `<pepn>` PEs run on each node
`ncpus=<ppn>`	Only select nodes that have at least `<ppn>` CPUs
`ompthreads=<nthreads>`	Run `<nthreads>` per PE, also sets `OMP_NUM_THREADS=<nthreads>` in runscript.

Example batch script

```
#!/bin/bash
#PBS -N xthi
#PBS -l select=3:ncpus=32:mpiprocs=16:ompthreads=1
#PBS -j oe
#PBS -o ./output
#PBS -l walltime=00:05:00

NODES=$( qstat -f ${PBS_JOBID} | awk '/Resource_List.nodect/{ print $3 }' )
NRANK=$( qstat -f ${PBS_JOBID} | awk '/Resource_List.mpiprocs/{ print $3 }' )
NTASK=$[ ${NRANK} / ${NODES} ]

export OMP_NUM_THREADS=1
aprun -n ${NRANK} -N ${NTASK} -d ${OMP_NUM_THREADS} -j1 ./a.out
```

A Note on the mppwidth, mppnppn and mppdepth

Casual examination of older Cray documentation or internet searches may suggest the alternatives, mppwidth, mppnppn and mppdepth for requesting resources.

These methods, while still functional, are Cray specific extensions to PBS Pro which have been deprecated by Altair.

Therefore we recommend all new scripts be structure using “select” notation describe previously

Lifecycle of a batch script

esLogin

qsub run.sh

Example Batch Job Script – run.sh

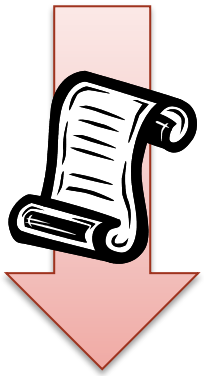
```
#!/bin/bash
#PBS -l select=16:ncpus=4:mpiprocs=16:ompthreads=1
#PBS -l walltime=1:00:00

cd $WORKDIR
aprun -n 64 -N 4 simulation.exe
rm -r $WORKDIR/tmp
```

Requested Resources

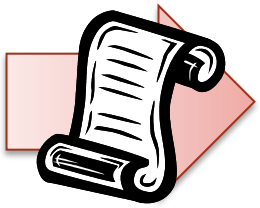
Serial

Parallel

Serial

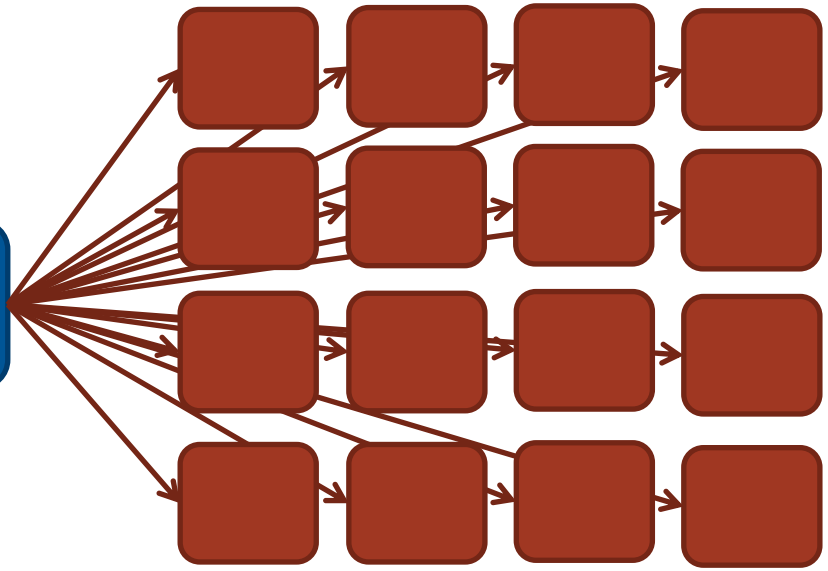


PBS Queue Manager



PBS MOM Node

Cray XC Compute Nodes



Watching a launched job on the Cray XE

- **`xtnodestat`**
  - Shows how XE nodes are allocated and corresponding `aprun` commands
- **`apstat`**
  - Shows `aprun` processes status
  - `apstat` overview
  - `apstat -a[ apid ]` info about all the applications or a specific one
  - `apstat -n` info about the status of the nodes
- **Batch `qstat` command**
  - shows batch jobs

Pre/Post Processing Mode

This is a new mode for the Cray XC30.

- **Designed to allow multi-user jobs to share compute nodes**
  - More efficient for apps running on less than one node
  - Possible interference from other users on the node
- **Uses the same fully featured OS as service nodes**
- **Multiple use cases, applications can be:**
  - entirely serial
  - embarrassingly parallel e.g. fork/exec, spawn + barrier.
  - shared memory using OpenMP or other threading model.
  - MPI (limited to intra-node MPI only*)
- **Scheduling and launching handled by PBS**
  - Similar to any normal PBS cluster deployment.

*Cray MPI Support coming soon.

Understanding Module Targets

- **The wrappers, cc, CC and ftn are cross compilation environments that by default target the compute nodes.**
  - This means compilers will build binaries explicitly targeting the CPU architecture in the compute nodes
  - It will also link distributed memory libraries by default.
  - Binaries built using the default settings will probably not work on serial nodes or pre/post processing nodes.
- **Users many need to switch the CPU target and/or opt to remove network dependencies.**
  - For example when compiling serial applications for use on esLogin or pre/post processing nodes.
- **Targets are changed by adding/removing appropriate modules**

Cray XC30 Target modules

CPU Architecture Targets	Network Targets
• `craype-ivybridge` (default) Tell compiler and system libraries to target Intel Ivybridge processors • `craype-sandybridge` Tell compiler and system libraries to build binaries targeting Intel Sandybridge processors	• <none> (default) Link network libraries into binary. • `craype-target-local_host` Do not link network libraries into the binary

- **We hope that future releases of the Cray MPI will include an intra-node mode that allows binaries compiled with Cray MPI will be used directly on Pre/Post Processing nodes without recompilation.**
- **Users will use "`mpiexec`" instead of "`aprun`" to distinguish between the two.**

Cluster Compatibility Mode (CCM)

- CCM creates small TCP/IP connected clusters of compute nodes.
- It was designed to support legacy ISV applications which could not be recompiled to use ESM mode.
- There is a small cost in performance due to the overhead of using TCP/IP rather than native Aries calls.
- Users also are also responsible for initialising the application on all the participating nodes, including any daemon processes.
- Still exclusive use of each node by a single users, so only recommended when recompilation unavailable.