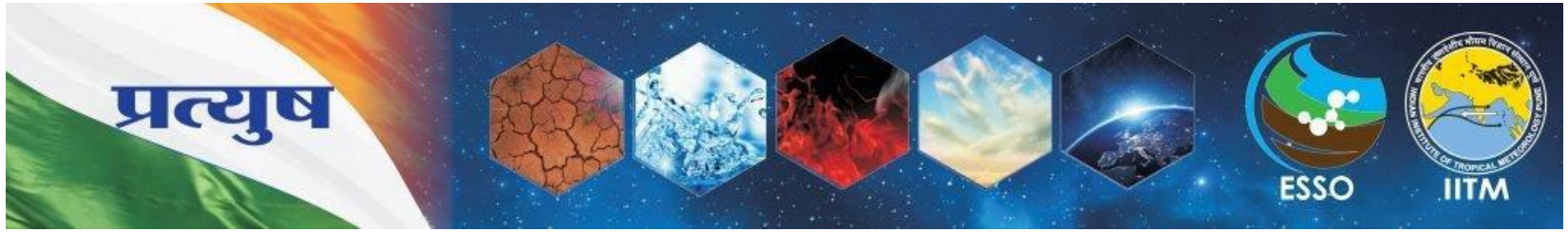
Compile applications on the Cray XC



Compiler Driver Wrappers (1)

- All applications that will run in parallel on the Cray XC should be compiled with the standard language wrappers.

  The compiler drivers for each language are:
  - cc – wrapper around the C compiler
  - CC – wrapper around the C++ compiler
  - ftn – wrapper around the Fortran compiler

- These scripts will choose the required compiler version, target architecture options, scientific libraries and their include files automatically from the current used module environment

- Use them exactly like you would the original compiler

  E.g. To compile prog1.f90:

  $> ftn -c prog1.f90

Compiler Driver Wrappers (2)

- The scripts choose which compiler to use from the PrgEnv module loaded

PrgEnv	Description	Real Compilers
PrgEnv-cray	Cray Compilation Environment	crayftn, craycc, crayCC
PrgEnv-intel	Intel Composer Suite	ifort, icc, icpc
PrgEnv-gnu	GNU Compiler Collection	gfortran, gcc, g++
PrgEnv-pgi	Portland Group Compilers	pgf90, pgcc, pgCC

- Use module swap to change PrgEnv, e.g.

```
$> module swap PrgEnv-cray PrgEnv-intel
```

- PrgEnv-cray is loaded by default at login. This may differ on other Cray systems.
  - use `module list` to check what is currently loaded
- The Cray MPI module is loaded by default (cray-mpich).
  - To support SHMEM load the cray-shmem module.

Which compiler do I use?

- **All compilers provide Fortran, C, C++ and OpenMP support**

- **Experiment with the various compilers**
  - Mixing binaries created by different compilers may cause issues

Compiler Versions

- **There are usually multiple versions of each compiler available to users.**
  - The most recent version is usually the default and will be loaded when swapping the `PrgEnv`.
  - To change the version of the compiler in use, swap the Compiler Module. e.g. `module swap cce cce/8.6.0`

PrgEnv	Compiler Module
PrgEnv-cray	cce
PrgEnv-intel	intel
PrgEnv-gnu	gcc
PrgEnv-pgi	pgi

EXCEPTION: Cross Compiling Environment

- The wrapper scripts, `ftn`, `cc`, and `CC`, will create a highly optimized executable tuned for the Cray XC's compute nodes (cross compilation).
- This executable may not run on the login nodes (nor pre/post nodes)
  - Login nodes do not support running distributed memory applications
  - Some Cray architectures may have different processors in the login and compute nodes. Typical error is '… illegal Instruction …'
- If you are compiling for the login nodes
  - You should use the original direct compiler commands,
    e.g. `ifort`, `pgcc`, `crayftn`, `gcc`
  - `PATH` will change with modules.
  - All libraries will have to be linked in manually.
  - Conversely, you can use the compiler wrappers {`cc`,`CC`,`ftn`} and use the `-target-cpu=` option among {abudhabi, haswell, interlagos, istanbul, ivybridge, mc12, mc8, sandybridge, shanghai, x86_64. The x86_64 is the most compatible but also less specific.

About the -I, -L and -l flags

- **For libraries and include files being triggered by module files, you should NOT add anything to your Makefile**
  - No additional MPI flags are needed (included by wrappers)
  - You do not need to add any `-I`, `-l` or `-L` flags for the Cray provided libraries
- **If your Makefile or Cmake needs an input for `-L` to work correctly, try using '.'**
- **If you really, really need a specific path, try checking '`module show <X>`' for some environment variables**

Dynamic vs. Static linking

- **Currently, static linking is default.**
- **To decide how to link,**
  - Either set `CRAYPE_LINK_TYPE` to `static` or `dynamic`
  - Or pass the `-static` or `-dynamic` option to the wrappers `cc, CC or ftn`.
    The `-shared` option is used to create shared libraries `*.so`
- **Features of `dynamic` linking :**
  - smaller executable, automatic use of new libs
  - Might need longer startup time to load and find the libs
  - Environment (loaded `modules`) should be the same between your compiler setup and your batch script (eg. when switching to `PrgEnv-intel`)
- **Features of `static` linking :**
  - Larger executable (usually not a problem)
  - Faster startup
  - Application will run the same code every time it runs (independent of environment)
- **If you want to hardcode the rpath into the executable use**
  - Set `CRAY_ADD_RPATH=yes` during compilation
  - This will always load the same version of the lib when running, independent of the version loaded by modules

The three styles of dynamic linking

Shared libraries mean applications may use a different versions of a library at runtime than was linked at compile time. On the Cray XC40 there are three ways to control which version is used

1. ***Default* –** Follow the default Linux policy and at runtime use the system default version of the shared libraries (so may change as and when system is upgraded)

2. ***pseudo-static* –** Hardcodes the path of each library into the binary at compile time. Runtime will attempt to use this version when the application start (as long as lib is still installed). Set CRAY_ADD_RPATH=yes at compile

3. ***Dynamic modules* –** Allow the currently loaded PE modules to select library version at runtime. App must not be linked with CRAY_ADD_RPATH=yes and must add “export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH” to run script

OpenMP

- **OpenMP is support by all of the PrgEnvs.**
  - CCE (PrgEnv-cray) recognizes and interprets OpenMP directives by default. If you have OpenMP directives in your application but do not wish to use them, disable OpenMP recognition with –hnoomp.

PrgEnv	Enable OpenMP	Disable OpenMP
PrgEnv-cray	-homp	-hnoomp
PrgEnv-intel	-openmp	
PrgEnv-gnu	-fopenmp	
PrgEnv-pgi	-mp	

- **Intel OpenMP spawns an extra helper thread which may cause oversubscription. Hints on that will follow.**

Compiler man Pages

- For more information on individual compilers

PrgEnv	C	C++	Fortran
PrgEnv-cray	man craycc	man crayCC	man crayftn
PrgEnv-intel	man icc	man icpc	man ifort
PrgEnv-gnu	man gcc	man g++	man gfortran
PrgEnv-pgi	man pgcc	man pgCC	man pgf90
Wrappers	man cc	man CC	man ftn

- To verify that you are using the correct version of a compiler, use:
  - -V option on a cc, CC, or ftn command with PGI, Intel and Cray
  - --version option on a cc, CC, or ftn command with GNU

Steps to compile application for CPU architecture

- Check list of modules needed for your program/ application
  Eg :- module switch PrgEnv-cray/6.0.4 PrgEnv-gnu

- Load related CPU architecture module. If not loaded, application will be compiled for x86_64 architecture
  Eg:- module load craype-broadwell

- Check Makefile for compilers and related flags
  Eg:- Use standard language wrappers and related compiler flags

Steps to compile application for KNL

- Check list of modules needed for your program/ application
  Eg :- module switch PrgEnv-cray/6.0.4 PrgEnv-gnu

- Load related KNL architecture module.
  Eg:- module load craype-mic-knl

- Use KNL specific compilation flags
  "-xMIC-AVX512" for Intel Compilers
  "-hcpu=mic-knl" for Cray compilers
  "-march=knl" for GNU compilers

- Check Makefile for compilers
  Eg:- Use standard language wrappers i.e cc, CC or ftn

Run applications on XC

How applications are generally run on a XC

- **Most Cray XCs are batch systems.**
  - Users submit batch job scripts to a scheduler from a login node (e.g. PBS, MOAB, SLURM) for execution at some point in the future.
    Each job requires resources and a prediction how long it will run.
  - The scheduler (running on an external server) chooses which jobs to run and allocates appropriate resources
  - The batch system will then execute the user's job script on an a different node as the login node.
  - The scheduler monitors the job and kills any that overrun their runtime prediction.
- **User job scripts typically contain two types of statements.**
  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 executables that run on compute nodes.
     1. Launched using a special command (`aprun`)

The Three types of Cray XC Nodes

External Login (esLogin)

- This is the node you access when you first log in to the system.
- It runs a full version of the CLE operating system (all libraries and tools available)
- They are used for editing files, compiling code, submitting jobs to the batch queue and other interactive tasks.
- They are shared resources that may be used concurrently by multiple users.

service nodes

- Its purpose is managing running jobs, but you can access using an interactive session.
- It runs a full version of the CLE operating system (all libraries and tools available)
- They are shared resources, mistakes and misbehaviour can effect jobs of other users(!).

compute nodes

- These are the nodes on which jobs are executed
- It runs Compute Node Linux, a version of the OS optimised for running batch workloads
- They can only be accessed by starting jobs with **aprun** (in conjunction with a batch system)
- They are exclusive resources that may only be used by a single user
- Always connects to the Aries network.

PBS and ALPS on the XC (Introduction)

- Cray system uses the PBS workload manager and the Application Level Placement Scheduler (ALPS)
- In your daily work you will mainly encounter the following commands:

```
qsub  – Submit a batch script to PBS.
aprun – Run parallel jobs within the script.
qdel  – Signal jobs under the control of PBS
qstat – information about running jobs
```

- Plenty of information can be found in the corresponding man pages on the system
- The entire information about your simulation execution is contained in a batch script which is submitted via qsub.
- The batch script contains one or more parallel job runs executed via aprun. Nodes are used exclusively.

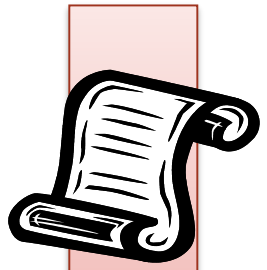
Sample Job Script:

```
#! /bin/bash

#PBS -N Test
#PBS –q <queue_name>
#PBS -l nodes=4:ppn=36
#PBS -l walltime=00:10:00
#PBS -V
#PBS -j oe

export OMP_NUM_THREADS=1

aprun -n 144 -N 36 <exe> <arguments if any>
```

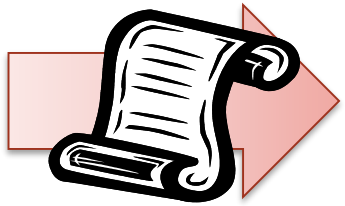
Lifecycle of a batch script

Login Node

qsub job.pbs



MOAB Queue Manager



MOM Node

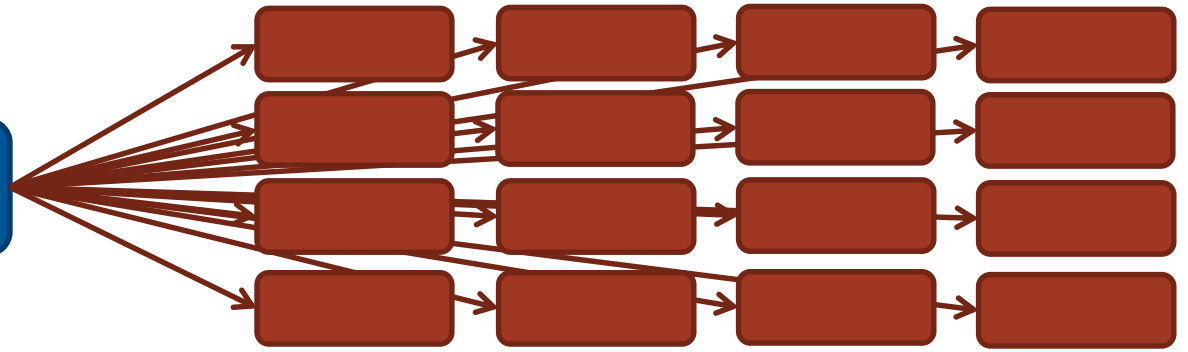
Scheduler Resources

Serial

Parallel

```
$> cat job.pbs
#!/bin/bash
#PBS -q small
#PBS -l nodes=16:ncpus=36
#PBS -l walltime=00:10:00
#PBS -j oe

cd <my_work_dir>
aprun -n 384 -N 36 <exe>
rm -r <my_work_dir>/<tmp_files>
```

Cray XC Compute Nodes



Requesting Resources

- Job requirements as #PBS comments in the headers of the batch script
- Common options:

Option	Description
`-l nodes=<nnodes>:ppn=24`	Requests X full nodes (only full nodes are available on HazelHen)
`-l walltime <HH:MM:SS>`	Maximum wall time job will occupy
`-N <job_name>`	Name of the job
`-A <code>`	Account to run job under (for controlling budgets)
`-j oe`	collect both `stderr` and `stdout` to a single file specified by the `-o` option or the default file for `stdout`.
`-o <my_output_file_name>` `-e <my_error_file_name>`	Redirects stdout and stderr to two separate files. If not specified, the script output will be written to files of the form <script_name>.e<JOBID> and <script_name>.o<JOBID>.
`-q <queue>`	Submit job to a specific queues

These can be overridden or supplemented by adding arguments to the qsub command line, e.g.

```
$> qsub -l nodes=20:ncpus=36 run.pbs
```

Running an application using ALPS + aprun

- **aprun is the ALPS application launcher**
  - Runs applications on the XC compute nodes.
    aprun launches groups of Processing Elements (PEs) on the compute nodes
    (PE == (MPI RANK || Coarray Image || UPC Thread || ..))
  - Cannot get more resources for aprun than requested via WLM.
  - The most important parameters (manpage for more examples)

Option	Description
-n	Total Number of PEs used by the application
-N	Number of PEs per compute node
-d	“stride” between 2 PEs on a node, usually used for: Number of threads per PE
-S	Pes per numa node (can have effects for memory bandwidth)
-j	-j 2 enables hyperthreading

  - Applications started without aprun, are executed on mom nodes and can affect other users jobs

Cray XC Basic MPI-Jobs Examples

Single node, Single task
Run a job on one task on one node with full memory.

```
…
#PBS -l nodes=1:ncpus=36
…
aprun -n 1 ./<exe>
```

Single node, Multiple Ranks
Run a pure MPI job with 36 Ranks or less on one node.

```
…
#PBS -l nodes=1:ncpus=36
…
aprun -n 36 ./<exe>
```

Multiple nodes, Multiple Ranks
Run a pure MPI job on 4 nodes with 36 MPI ranks or less on each node.

```
…
#PBS -l nodes=4:ncpus=36
…
aprun -n 144 -N 36 ./<exe>
```

Hyperthreads on the XC with ALPS

- **Intel Hyper-Threading is a method of improving the throughput of a CPU by allowing two independent program threads to share the execution resources of one CPU**
  - When one thread stalls the processor can execute read instructions from a second thread instead of sitting idle
  - Because only the thread context state and a few other resources are replicated (unlike replicating entire processor cores), the throughput improvement depends on whether the shared execution resources are a bottleneck
  - Typically much less than 2x with two hyperthreads
  - With `aprun`, hyper-threading is controlled with **`-j`**
    - **-j 1** = no hyper-threading (default)
      (a node is treated to contain 24 cores)
    - **-j 2** = hyper-threading enabled
      (a node is treated to contain 48 cores)
  - Try it, if it does not help, turn it off.

```
…
#PBS -l nodes=1:ncpus=36
aprun -n 72 -j2 ./<exe>
```

XC Hybrid MPI/OpenMP Jobs (PBS Example)

Pure OpenMP Job
Using 4 threads on one a single node

```
…
#PBS -l nodes=1:ncpus=36
…
export OMP_NUM_THREADS=4
echo "OMP_NUM_THREADS: $OMP_NUM_THREADS"
aprun -n 1 -d $OMP_NUM_THREADS ./<omp_exe>
```

Hybrid MPI/OpenMP job on 3 nodes with 12 MPI ranks per node, 4 threads for each rank, using Hyperthreads.

```
…
#PBS -l nodes=3:ncpus=36
…
export OMP_NUM_THREADS=4
echo "OMP_NUM_THREADS: $OMP_NUM_THREADS"
aprun -n 54 -N 12 -d $OMP_NUM_THREADS -j 2 ./<hybrid_exe>
```

Monitoring your Job

- **After submitting your job, you can monitor its status**

Command	Description
`$> qsub <batch_script> <JOBID>`	Start your job with from the shell with qsub. The <JOBID> is printed.
`$> qstat -u $USER`	Prints status of all your jobs. Always check that the reported resources are what you expect.
`$> showq -u $USER`	information of active, eligible, blocked and completed jobs
`$> checkjob <JOBID>`	Detailed job state information and diagnostic output
`$> qdel <JOBID>`	Only if you think that your job is not running properly after inspecting your output files, you can cancel it with qdel.

Interactive Sessions

request an **interactive session**.

- use qsub option `-I`
- typically used for small jobs which have to be run frequently for testing or for debugging sessions with STAT, ATP, DDT etc. and usually used with small amount of nodes.

```
eslogin01$> qsub -I -l nodes=2,walltime=00:19:00
qsub: waiting for job 123456.XXX-batch.YYY.com to start
...
qsub: job 123456.XXX-batch.YYY.de ready
Welcome to XXX (Cray XC40) at XXX.
Directory: /home/userxyz
Fri Jan 01 08:15:00 CEST 2018
mom15$> aprun -n 72 -N 36 … <my_application>
```

Once the Job is executed by PBS, the user receives a shell prompt where commands like `aprun` can be executed directly. An entire batch script could be executed with `source <bath_scipt>`.
(!) interactive sessions are executed on MOM nodes. Every compute intense calculation has to be executed with `aprun`.

Environment variables

- Job specific environmental variables are available

Environment Variable	Description
PBS_O_WORKDIR	Directory where qsub has been executed
PBS_JOBID	Job ID
PBS_JOBNAME	Job name as specified by the user
PBS_NODEFILE	List of allocated nodes.

- E.g. using the maximum allocated resources

```
#!/bin/bash
#PBS -N xthi
#PBS -l nodes=3:ncpus=36
#PBS -l walltime=00:05:00
...
NS=$( qstat -f ${PBS_JOBID} | awk '/Resource_List.nodect/{ print $3 }' )
NRANK=$[ ${NS} * 36 ]

aprun -n ${NRANK} -N 36 -d ${OMP_NUM_THREADS} -j1 ./a.out
```

What resources did it use?

- **Can be good to record contents of $PBS_NODEFILE during batch session to note what nodes were used (though list will be long if use lots of nodes!)**
  - cat $PBS_NODEFILE | sort | uniq -c
  - Or look at “apstat –avv apid” when job is running to see placement
- **See upcoming information on Cray Performance Tools**
  - perftools-lite is good place to start
- **For accelerators, environment variables are available to produce job statistics**

More info about my running job....

```
>apstat -avv  614502

Total (filtered) placed applications: 1
  Apid  ResId   User  PEs Nodes   Age State  Command
614502 355534 casnan 2400   100 8h36m   run cesm.exe

Application detail
Ap[101]: apid 614502, pagg 0xc6000012c5, resId 355534, user casnan,
      gid 208, account 0, time 0, normal
  Batch System ID = 70460.sdb
  Reservation flags = 0x100000
  Application flags = 0x142001
  Created at Mon Oct  5 00:47:19 2015
  Originator: aprun on NID 198, pid 5168
  Number of commands 1, control network fanout 32
  Network: cookies 0x2a80000/0x2a90000, NTTgran/entries 0/0
  Cmd[0]: cesm.exe -n 2400 -N 24 -j 1, 2730MB, XT, nodes 100, exclusive
  Placement list entries: 2400
  Placement list: 168-172,187-189,212-219,221-227,249,292-319,324-353,374-382,414-422
```

Point to Note:

- All to All communication latency is ~1 micro sec
- Each login session has its own module state which can be modified by loading, swapping or unloading the available modules
- Use specific target system modules to load instruction
- On cray machine, all applications should be compiled with the standard language wrappers.
  - cc – wrapper around the C compiler
  - CC – wrapper around the C++ compiler
  - ftn – wrapper around the Fortran compilee
- On cray machine, use ***aprun*** for application launching instead of mpirun or mpi.hydra etc