



(© Cray Inc 2014)

INTRODUCTION TO COMPILATION, LAUNCHING AND PLACEMENT ON CRAY XC30

A STEP-BY-STEP GUIDE TO RUNNING ON THE CRAY XC30

OVERVIEW

Compiling and launching applications on a Massively Parallel Processing (MPP) supercomputer like the Cray XC30 requires some additional steps over running a standard serial application. Firstly, most supercomputers operate in batch processing mode, requiring users to submit jobs to the scheduler for execution in the future on dedicated resources. In order to run efficiently users must match the resources requested/reserved for a job with the number of ranks and threads used by a job running in parallel.

These exercises attempt to familiarise users with the basic concepts of compiling, scheduling and running parallel applications. By the end of these exercises you should be able to:

- Log in to the Cray XC30 supercomputer
- View the current loaded modules, swap between Programming Environments, add and remove modules from your environment.
- Compile a simple C, C++ or Fortran application.
- Submit a parallel job to the batch system.
- Construct a parallel launch command for pure MPI, SHMEM, Fortran 2008 or UPC application
- Construct a parallel launch command for a hybrid/threaded parallel application.

LOGGING IN AND DOWNLOADING EXERCISE MATERIAL

All access to Cray XC30 will be through a shell. If you do not already have a username and password you will be provided one by the course organisers. Once you have your username, password and the hostname of the Cray XC30 from the course organisers you can usually log in from a Unix host by running:

```
ssh -X -l <username> <hostname>
```

or by connecting via another Secure Shell Client (e.g. PuTTY).

Once you have logged in you will be required to download some example source code from an address that will be provided by your course organisers. Once you have the address, download the package with the following command:

```
wget <examples address>
```

Once you have successfully downloaded the package, extract the package into your scratch directory (this is the Lustre filesystem)

```
cd <work dir>  
tar xf running-on-xc30.tar.bz2
```

MANIPULATING YOUR PROGRAMMING ENVIRONMENT

The Cray Programming Environment is designed to simplify the process of building parallel applications for the Cray XC30. It supports multiple different underlying tool chains (compilers) from different vendors (e.g. Cray, Intel and GNU).

Only one on the available tool chains, or Programming Environments (PrgEnvs), can be active at any one time; to see the list of currently loaded modules type:

```
module list
```

A list will be displayed on screen, one of which will be of the form `PrgEnv-*`, e.g.

```
20) PrgEnv-cray/5.0.41
```

To see which other modules are available to load run the command

```
module avail
```

On most systems this may be a very long list, to be more selective run:

```
module avail PrgEnv
```

Which will list all available modules that start with `PrgEnv`.

To swap from the currently `PrgEnv` to another `PrgEnv` (e.g. to select a different tool chain), run:

```
module swap PrgEnv-cray PrgEnv-intel
```

This will unload the Cray `PrgEnv` and replace it with the Intel `PrgEnv`.

Modules are also used to automatically link extra libraries and software when compiling. To link the Fastest Fourier Transform in the West (FFTW) library into your application, load the `fftw` module by running:

```
module load fftw
```

For a specific version of the module, append the name with a slash and the version number e.g.

```
module load fftw/2.1.5.6
```

Should you no longer require a module in your path, you can unload it with:

```
module unload fftw
```

To see more information about a particular module, e.g. its release notes you can run:

```
module help fftw
```

Exercises

1. Log in to the XC30 supercomputer.
2. List the current modules, if not already loaded swap to PrgEnv-cray. Try swapping to the Intel and GNU PrgEnvs.
3. What happens when you try and load two PrgEnvs at once? What happens if you try to swap out a PrgEnv that is not already loaded?
4. Find out what does the cce module does? What is the equivalent for the PrgEnv-gnu and PrgEnv-intel modules?
5. Experiment with loading different library modules. Tip, most Cray supplied modules start with cray-*. Try loading different versions, try loading two versions of the same module, what happens?

COMPILING AN APPLICATION

To build applications to run on the XC30 users should always use the Cray compiler driver wrappers, `ftn` (Fortran), `cc` (C) and `CC` (C++) to compile their applications. These wrappers interface with the Cray Programming Environment and make sure the correct libraries and flags are supplied to the underlying library.

When compiling an application make sure that the appropriate modules are loaded before starting to compile, then modify the build system to call the appropriate wrapper scripts in place of the direct compilers.

The provided example application is a simple diagnostic MPI/OpenMP application called “`xthi`”. The build system has been designed to produce a different binary depending on the PrgEnv used, e.g. when `PrgEnv-intel` is loaded the binary will be called `xthi-intel`.

To compile the example application; first enter the build directory:

```
cd xthi
```

Then run make

```
make
```

If the `PrgEnv-cray` environment was loaded this will produce `xthi-cray`, otherwise it will produce either `xthi-gnu` or `xthi-intel`.

If you have your own application, you may wish to try and adapt it to the Cray build system now.

SUBMITTING JOBS TO THE BATCH SYSTEM

Nearly all XC30 system will run some form of batch scheduling system to fairly manage the nodes of the system between all the users. Therefore, users must submit all their work in the form of batch job scripts for the scheduler to run in the future.

An example script for the PBS Pro (version 12+) batch scheduler is provided in the example directory; it is called `run.pbs`. This script includes special `#PBS` comments which are used to provide information to the batch system about the required resources, e.g.

```
#PBS -N xthi
```

Specifies the name of the job (displayed in queues etc).

```
#PBS -l select=1
```

Specifies the number of nodes that are required (in this case one node).

```
#PBS -l walltime=0:05:00
```

Specifies the length of time the job will take (no longer than five minutes).

Some systems, like ARCHER, make required additional accounting information:

```
#PBS -A y11
```

Jobs are then submitted for execution using the `qsub` command:

```
qsub run.pbs
```

Which will return a job id that uniquely identifies the job. Any of the `#PBS` settings contained within a file can be overridden at submission time, e.g.

```
qsub -A y02 -l select=2 run.pbs
```

Changes the number of nodes requested and the project account the job will run under.

To view the status of all the jobs in the system use the `qstat` command. To find information about a specific job using the jobid:

```
qstat <jobid.sdb>
```

To find information about jobs belonging to a specific user run:

```
qstat -u <username>
```

By default, when a run has completed, the output from the run is left in the run directory in the form:

```
<jobname>.o<jobid>
```

for stdout and

```
<jobname>.e<jobid>
```

for stderr.

LAUNCHING APPLICATIONS ON THE CRAY XC30

Some time after a job is submitted the scheduler will assign the resources the job requested and start the job script running. At this point the supplied script will begin to execute on a PBS MOM node, and it is important to understand that at this stage most of the programs in the script are not yet running in parallel on the Cray XC30 compute nodes.

So statements a simple statement like this:

```
cd ${PBS_O_WORKDIR}
```

(which changes the current working directory to the same directory that the original qsub command) is actually run in serial on the PBS MOM node.

Only programs/commands that are preceded by an aprun statement will run in parallel on the XC30 compute nodes, e.g.

```
aprun -n 16 -N 16 ./xthi-cray
```

Therefore users should take care that long or computational intensive jobs are only ever run inside an appropriate aprun command.

Each batch job may contain multiple programs with aprun statements, so long as:

- An individual parallel program does not request more resources than have been allocated by the batch scheduler
- The program does not exceed the wall clock time limit specified in the submission script.

Looking at the provided run.pbs example for the xthi program we see the following lines:

```
for exe in xthi-cray xthi-intel xthi-gnu;
do
  if [[ -e "${exe}" ]]; then
    echo "Testing with ${exe}"
    aprun -n ${NPROC} -N ${NTASK} -d ${OMP_NUM_THREADS} -j1 ./${exe}
  fi
done
```

This loop checks to see if any of the xthi-* binaries created in the previous step exist, and if so will execute them on the backend using the aprun command.

See the lecture notes and the manual page, man aprun for a full explanation of the command line arguments to aprun.

Example output

The xthi example provides a simple output from each of the MPI ranks and threads and explains how each thread or rank is bound to its cpu, e.g.

Testing with xthi-cray

```
./xthi-cray compiled with: Cray C++ : Version 8.2.x.x (u82093c82157)
Hello from rank    0, thread 0, on nid00242. (core affinity = 0)
Hello from rank    1, thread 0, on nid00242. (core affinity = 1)
Hello from rank    2, thread 0, on nid00242. (core affinity = 2)
Hello from rank    3, thread 0, on nid00242. (core affinity = 3)
Hello from rank    4, thread 0, on nid00242. (core affinity = 4)
Hello from rank    5, thread 0, on nid00242. (core affinity = 5)
Hello from rank    6, thread 0, on nid00242. (core affinity = 6)
Hello from rank    7, thread 0, on nid00242. (core affinity = 7)
Hello from rank    8, thread 0, on nid00242. (core affinity = 8)
Hello from rank    9, thread 0, on nid00242. (core affinity = 9)
Hello from rank   10, thread 0, on nid00242. (core affinity = 10)
Hello from rank   11, thread 0, on nid00242. (core affinity = 11)
Hello from rank   12, thread 0, on nid00242. (core affinity = 12)
Hello from rank   13, thread 0, on nid00242. (core affinity = 13)
Hello from rank   14, thread 0, on nid00242. (core affinity = 14)
Hello from rank   15, thread 0, on nid00242. (core affinity = 15)
```

Launching Hybrid MPI/OpenMP applications

The aprun command cannot launch application threads directly. This is done by the application itself, commonly via the OpenMP runtime. However, the aprun can reserve space on the CPUs for threads to execute via the `-d` parameter.

For a typical OpenMP application the script should include the following settings:

```
export OMP_NUM_THREADS=4
aprun -n ${NPROC} -N ${NTASK} -d ${OMP_NUM_THREADS} ./xthi-cray
```

Exercises

- Check the application is launching the right number of ranks and threads.
 - Is the application using all the available cores on the node?
 - If not how can you adjust it to fill the node?
 - What happens when you request more cores or nodes than there are nodes?
 - Are the cores balanced across the numa-nodes? Try using the `-S` option to balance the ranks across the numa-nodes.
- Look in the manual at the `-B` option to aprun, how might this be used and what is the expected output?
- Try running the application across more than one node.
 - What changes do you need to make to the PBS run script?
 - What changes do you need to make to the aprun statement?
 - What happens when you get these wrong?
- Try running the application with some OpenMP threads.
 - What changes do you have to make to the run script?
 - What changes do you have to make the aprun command line?
 - What if the OMP_NUM_THREADS and the `-d` argument do not match? How does this differ to getting MPI ranks and numbers of nodes wrong?

- Are there any differences between the outputs from different compilers?
 - Look at the output from the Intel Compiler with OpenMP and default binding.
 - What could be causing this (remember the Intel compiler generates an additional helper thread when using OpenMP).

ADVANCED APPLICATION BINDING

Using the `xthi` application provided in the introduction, try and understand the following situations:

- Try limiting or switching off the binding by passing the `-cc none` or `-cc numa_node` options to `aprun`.
 - What happens to the core affinity? How might this affect application performance?
 - Try using the custom ordering (see `-cc` in `man aprun`) to produce a non-standard binding of cores/threads.
- Enable the Intel Hyperthreads on the system by adding `-j2` to the `aprun` line?
 - What are the Hyperthread pairs? Which pairs of ranks/threads are being assigned to the same threads?
- Look at the output from the threads when using the Intel PrgEnv. How could you fix this?
 - Think about a custom binding list, or perhaps one of the other default (or non-) binding options.
- What happens if you use MPMD mode?
 - It is legal to use the same binary as the argument to the `aprun` command line. What might this potentially allow you to do in your application?

RANK REORDERING

Using the `xthi` example provided, investigate how rank reordering affects the placement of tasks on the node.

- If you have not done so already, adjust the default job script so it runs across multiple nodes (i.e. four or more).
- Add `export MPICH_RANK_REORDER_DISPLAY=1` to the script.
- Run experiments changing the value of `export MPICH_RANK_REORDER_METHOD=X` to 0 (Round-robin), 1 (default SMP) and 2 (folded). Note how this changes the hostname each rank ends up on.
- Create a custom rank-reorder file in `MPICH_RANK_ORDER` and add `export MPICH_RANK_REORDER_METHOD=3` to the run script.
 - What happens if this file specifies the wrong number of values?
 - Load the `perftools` module, see `man grid_order` for generating `MPICH_RANK_ORDER` files for Cartesian grid layouts.
- Consider how you might combine using MPMD mode and rank reordering to optimise the behaviour of applications that have individual ranks with different memory or IO requirements (E.g. IO Servers or gather/scatter models).