# First steps on using an HPC service

ARCHER







# ARCHER Service

## Overview and Introduction

EPSRC

NERC SCIENCE OF THE ENVIRONMENT

archer

CRAY THE SUPERCOMPUTER COMPANY

epcc

THE UNIVERSITY OF EDINBURGH

# ARCHER in a nutshell

- UK National Supercomputing Service
- Cray XC30 Hardware
  - Nodes based on 2×Intel Ivy Bridge 12-core processors
  - 64GB (or 128GB) memory per node
  - 3008 nodes in total (72162 cores)
  - Linked by Cray Aries interconnect (dragonfly topology)
- Cray Application Development Environment
  - Cray, Intel, GNU Compilers
  - Cray Parallel Libraries (MPI, SHMEM, PGAS)
  - DDT Debugger, Cray Performance Analysis Tools


# Storage

- /home – NFS, not accessible on compute nodes
  - For source code and critical files
  - Backed up
  - > 200 TB total
- /work – Lustre, accessible on all nodes
  - High-performance parallel filesystem
  - Not backed-up
  - > 4PB total
- RDF – GPFS, not accessible on compute nodes
  - Long term data storage


# Connecting to ARCHER

- We will use SSH:
  - Typically included in a UNIX installation (Mac, Linux, …)
  - For windows one can use the free ssh client: **putty**
    `http://www.chiark.greenend.org.uk/~sgtatham/putty/`
- Address: login.archer.ac.uk
  - On UNIX:
    ```
    ssh -l username login.archer.ac.uk
    ```
  - On windows, type this into the “Host Name” field of putty


# Batch Processing

- Very important on the HPC service - the only way to access the main compute node back-end
  - You will find a similar system on any HPC service
- Maximum job size:
  - MPI: 72,162 (Unlikely to get all without a long wait)
  - OpenMP: 24
  - 48 hours (up to 3072 cores)/24 hours (above 3072 cores)
- Uses PBS to schedule jobs:
  - use **`qsub`** command to enter jobs
  - use **`qstat`** to monitor the jobs
  - Use **`qdel`** to kill jobs


# The `qsub` Command

- Batch script (PBS script) specifies what resources are required for a jobs, where to charge time, etc…:

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

- Can also be done at the command line:

```
> 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) |


# A simple sample script

```
#!/bin/bash
#PBS -N thisjob
#PBS -l select=43
#PBS -l walltime=0:10:00
#PBS -A y14

cd /work/y14/y14/guest10/
aprun -n 1024 -N 24 ./myprogram
```

job name

number of nodes to use

maximum runtime

budget to charge

Parallel job launcher

program name

archer

epcc

THE UNIVERSITY OF EDINBURGH

# Running Jobs examples

- 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 ./a.out
```

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

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

- To use hyperthreading

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


# The `qstat` command

- Type **`qstat`** to monitor the queue:

```
adrianj@eslogin004:~> qstat
Job id              Name             User             Time Use S Queue
----------------  ---------------- ---------------  -------- - -----
428837.sdb          xkdea_build      chardacr                0 Q standard
428862.sdb          xkdea_build      chardacr                0 Q standard
438791.sdb          STDIN            maoqian                 0 Q standard
441088.sdb          lammpstest       maoqian                 0 Q standard
441092.sdb          lammpstest       maoqian                 0 Q standard
450356.sdb          lubm_benchmark.  s1340401                0 Q standard
458769.sdb          G4_stand         rp3e11                  0 Q low
459377.sdb          new_t_a          lc2a11                  0 Q low
460120.sdb          BIL76K6          gbrandan                0 Q standard
461088.sdb          v-job            mm339                   0 Q low
461192.sdb          v_job            jf298                   0 Q low
```

- `bjobs -u username`
  - See only your jobs


# The `qdel` command

- To delete a job:
  - Enquire its job-ID with qstat
  - Delete it with qdel (e.g. job-ID: 5789)

```
qdel 5789
```

# Compiling MPI Programs

- Most massively parallel scientific HPC codes use **Message Passing**
  - MPI: Message Passing Interface
- To compile message passing code on the gateway:
  - Fortran programmers use ftn
  - C programmers use cc
  - C++ programmers use CC
- There is nothing magic about these MPI compilers!
  - simply wrappers which automatically include various libraries etc
  - Compilation done by compilers you have loaded. ARCHER has 3 compilers installed; Cray, Intel, and GNU.
- Easiest to use them inside a “Makefile”


# Modules

- The Cray Programming Environment uses the GNU “modules” framework to support multiple software versions and to create integrated software packages
  - As new versions of the supported software and associated man pages become available, they are installed and added to the Programming Environment as a new version, while earlier versions are retained to support legacy applications
  - System administrators will set the default version of an application, or you can choose another version by using modules system commands
  - Users can create their own modules, or administrators can install site specific modules available to many users.

# Viewing the current module state

- Each login session has its own module state which can be modified by loading, swapping or unloading the available modules.
- This state affects the functioning of the compiler wrappers and in some cases runtime of applications.
- A standard, default set of modules is always loaded at login for all users.
- Current state can be viewed by running:

```
$> module list
```


# Default modules example

```
adrianj@eslogin001:~> module list
Currently Loaded Modulefiles:
  1) modules/3.2.6.7
  2) nodestat/2.2-1.0500.41375.1.85.ari
  3) sdb/1.0-1.0500.43793.6.11.ari
  4) alps/5.0.3-2.0500.8095.1.1.ari
  5) MySQL/5.0.64-1.0000.7096.23.1
  6) lustre-cray_ari_s/2.3_3.0.58_0.6.6.1_1.0500.7272.12.1-1.0500.44935.7.1
  7) udreg/2.3.2-1.0500.6756.2.10.ari
  8) ugni/5.0-1.0500.0.3.306.ari
  9) gni-headers/3.0-1.0500.7161.11.4.ari
 10) dmapp/6.0.1-1.0500.7263.9.31.ari
 11) xpmem/0.1-2.0500.41356.1.11.ari
 12) hss-llm/7.0.0
 13) Base-opts/1.0.2-1.0500.41324.1.5.ari
 14) craype-network-aries
 15) craype/1.06.05
 16) cce/8.2.0.181
...
```


# Viewing available modules

- There may be many hundreds of possible modules available to users.
  - Beyond the pre-loaded defaults there are many additional packages provided by Cray
  - Sites may choose to install their own versions.
- Users can see all the modules that can be loaded using the command:
  - `module avail`
- Searches can be narrowed by passing the first few characters of the desired module, e.g.

```
adrianj@eslogin001 :~> module avail gc

------------------------------ /opt/modulefiles ----------------------------
gcc/4.6.1          gcc/4.7.2          gcc/4.8.0
gcc/4.6.3          gcc/4.7.3          gcc/4.8.1(default)
```


# Modifying the default environment

- Loading, swapping or unloading modules:
  - The default version of any inidividual modules can be loaded by name
    - e.g.: `module load perftools`
  - A specific version can be specified after the forward slash.
    - e.g.: `module load perftools/6.1.0`
  - Modules can be swapped out in place
    - e.g.: `module swap intel intel/13.1.1.163`
  - Or removed entirely
    - e.g.: `module unload perftools`
- Modules will automatically change values of variables like `PATH`, `MANPATH`, `LM_LICENSE_FILE`... etc
  - Modules also provide a simple mechanism for updating certain environment variables, such as `PATH`, `MANPATH`, and `LD_LIBRARY_PATH`
  - In general, you should make use of the modules system rather than embedding specific directory paths into your startup files, makefiles, and scripts


```
adrianj@eslogin008:~> module show fftw
-------------------------------------------------------------------
/opt/cray/modulefiles/fftw/3.3.0.4:

setenv           FFTW_VERSION 3.3.0.4
setenv           CRAY_FFTW_VERSION 3.3.0.4
setenv           FFTW_DIR /opt/fftw/3.3.0.4/sandybridge/lib
setenv           FFTW_INC /opt/fftw/3.3.0.4/sandybridge/include
prepend-path     PATH /opt/fftw/3.3.0.4/sandybridge/bin
prepend-path     MANPATH /opt/fftw/3.3.0.4/share/man
prepend-path     CRAY_LD_LIBRARY_PATH /opt/fftw/3.3.0.4/sandybridge/lib
setenv           PE_FFTW_REQUIRED_PRODUCTS PE_MPICH
prepend-path     PE_PKGCONFIG_PRODUCTS PE_FFTW
setenv           PE_FFTW_TARGET_interlagos interlagos
setenv           PE_FFTW_TARGET_sandybridge sandybridge
setenv           PE_FFTW_TARGET_x86_64 x86_64
setenv           PE_FFTW_VOLATILE_PKGCONFIG_PATH
/opt/fftw/3.3.0.4/@PE_FFTW_TARGET@/lib/pkgconfig
prepend-path     PE_PKGCONFIG_LIBS
fftw3f_mpi:fftw3f_threads:fftw3f:fftw3_mpi:fftw3_threads:fftw3
module-whatis    FFTW 3.3.0.4 - Fastest Fourier Transform in the West
-------------------------------------------------------------------
```


# Summary of Useful module commands

- Which modules are available?
  - `module avail`, `module avail cce`
- Which modules are currently loaded?
  - `module list`
- Load software
  - `module load perftools`
- Change programming environment
  - `module swap PrgEnv-cray PrgEnv-gnu`
- Change software version
  - `module swap cce/8.0.2  cce/7.4.4`
- Unload module
  - `module unload cce`
- Display module release notes
  - `module help cce`
- Show summary of module environment changes
  - `module show cce`


# 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 module environment.

- Use them exactly like you would the original compiler, e.g. To compile `prog1.f90` run

```
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++` |

- 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.
- The drivers automatically support an MPI build
  - No need to use specific wrappers such as mpiifort, mpicc

# PLEASE NOTE : Cross Compiling Environment

- You are compiling on a Linux login node but generating an executable for a CLE compute node

- Do not use crayftn, craycc, ifort, icc, gcc, g++… unless you want a Linux executable for the service node
  - ALWAYS Use **`ftn`**, **`cc`**, or **`CC`** instead
  - Use the direct compiler commands if the executable is supposed to run on the service nodes (utilities, setup, …)


# Documentation

- Up to date version of local documentation is always available at:
  - **`http://www.archer.ac.uk/`**