Using the Cray Programming Environment

Alistair Hart
Cray Exascale Research Initiative Europe

Purpose of this talk

- **How to compile and run jobs**
  - If you have already used tödi, you may know some of this already
- **The practical exercises automate a lot of this**
  - The practicals are about learning OpenACC,
    - not remembering system-specific details
    - The Makefiles, jobscripts etc. can be used as templates for your projects

The system

- **You are using a Cray system called "tödi"**
  - it is a Cray XK7 system
  - you log in and compile on a front end node
    - these nodes have no GPU, so you can't run jobs here
  - jobs run on the compute nodes
    - each node has one AMD Interlagos CPU and one Nvidia GPU
      - 268 nodes contain Nvidia Kepler K20x GPUs
      - (4 nodes contain older Nvidia Fermi X2090 GPUs)
  - you run the jobs by submitting a jobscript to the SLURM batch system
    - compute jobs can't be run from the front end command line
  - there are two filesystems
    - home directories; yours is $HOME
    - the lustre filesystem; your directory is /scratch/todi/$USER
  - you should submit jobs from a directory on the lustre filesystem
  - home directories are backed up; the lustre filesystem is not and old files are periodically purged

Getting started

- **Cray uses a linux-based environment on the login nodes**
  - You will have a bash login shell by default
  - All the usual linux commands are available
  - Software versions are loaded and unloaded using the Gnu module command (see man module)
    - To see which modules are currently loaded, type: module list
    - To see which modules are available, type: module avail
      - You can wildcard the end of the names, e.g.: module avail PrgEnv*
      - For more complicated grepping, you need to redirect stderr to stdout, e.g.
        - module avail 2>&1 | grep "Env"
    - You load a new module by typing: module load <module name>
    - Some modules (e.g. different compiler versions) conflict, so you should first "module unload" the old version (or use "module swap")

Programming Environments

- **A number of different compilers are supported**
  - You select these by loading a Programming Environment module
    - PrgEnv-cray for CCE (the default)
    - PrgEnv-pgi for PGI
    - PrgEnv-gnu for gcc, gfortran

  - Once one of these is loaded, you can then select a compiler version
    - CCE: module avail cce
    - PGI: module avail pgi
    - Gnu: module avail gcc
  - Swap to the most up to date version in each case
    - e.g. "module avail cce" to see the versions available
    - then "module swap cce cce/<whatever>"

  - For any GPU programming (CUDA, OpenCL, OpenACC...)
    - make sure you always: "module load craype-accel-nvidia35"
    - it is not loaded by default

Using the compilers

- **You use the compilers via wrapper functions**
  - ftn for Fortran; cc for C; CC for C++
  - it doesn't matter which PrgEnv is loaded (same wrapper name)

  - the wrappers add optimisation options, architecture-specific stuff and all the important library paths
    - make sure module xtpe-interlagos is loaded so these are correct
    - in many cases, you don't need any other compiler options
    - if you really want unoptimised code, you must use option -O0

- **Further information**
  - man pages for the wrapper commands give you general information
  - For more detail see the compiler-specific man pages
    - CCE: crayftn, craycc, crayCC
    - PGI: pgfortran, pgcc
    - GNU: gfortran, gcc
  - You will need the appropriate PrgEnv module loaded to see these

Some Cray Compilation Environment basics

- **CCE-specific features:**
  - Optimisation: -O2 is the default and you should usually use this
    - -O3 activates more aggressive options; could be faster or slower
  - OpenMP: is supported by default.
    - if you *don't* want it, use -hnoomp compiler flag
  - CCE only gives minimal information to stderr when compiling
    - to see more information, you should request a compiler listing file
      - flag -hlist=a for ftn and cc
      - writes a file with extension .lst
      - contains annotated source listing, followed by explanatory messages
    - each message is tagged with an identifier, e.g.: ftn-6430
      - to get more information on this, type: explain <identifier>

  - For a full description of the Cray compilers, see the reference manuals at http://docs.cray.com.

Submitting jobs and the lustre filesystem

- **You should submit jobs from the lustre filesystem**
  - you can compile there as well if you wish
  - Create a unique directory for yourself: `mkdir -p /scratch/todi/$USER`
    - and subdirectories if you want

- **To submit a job, create a SLURM jobscript**
  - there is a skeleton script provided as part of the tutorial materials
  - just rename the executable
    - note that command `aprun` is used within the jobscript to run the executable

  - submit the job using command: `sbatch <jobscript name>`
    - other options are specified in the jobscript header
    - a job number is returned
  - to view the queued and running jobs: `squeue`
    - to see just your jobs: `squeue -u $USER`
  - to stop a queued or running job: `scancel <job number>`

A sample SLURM script

- **SLURM parameters:**
  - nodes is the total number of nodes required
  - ntasks is the number of MPI ranks
  - ntasks-per-node is the number of ranks per node
    - Could be up to 32 for XE6
    - Usually set to 1 for XK7 GPU jobs (unless you are using CUDA_PROXY)
  - cpus-per-task is number of threads per rank (usually 1 for GPU jobs)
    - ntasks-per-node*cpus-per-task <= 32 for XE6
  - See "man sbatch" for more details
- **Within the script:**
  - you must manually set OMP_NUM_THREADS to cpus-per-task value
    - if you are using OpenMP
  - run job using aprun command
    - "aprun -B <executable>" is a good shortcut
    - avoids need to specify "-n", "-N", "-d" options
      - -n should match ntasks
      - -N should match ntasks-per-node
      - -d should match cpus-per-task

A sample script

```
#!/bin/bash
#SBATCH --job-name="myjob"
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=00:01:00
#SBATCH --output="myjob.log"

# If you are using OpenMP, set the number of threads here
#export OMP_NUM_THREADS=<whatever>

# Now run the job with aprun
#   aprun -B uses the same parameters (-N -n -d) as specified
in PBS

# As an example, we'll run the PGI pgaccelinfo program to give
# information about the GPU. To access this, we need to load
# PrgEnv-pgi:
module swap PrgEnv-cray PrgEnv-pgi
# In general, you don't need to do this.

# Run the command:
aprun -B pgaccelinfo
```

For information: Compiling CUDA

- **Compilation:**
  - module load craype-accel-nvidia35
  - Main CPU code compiled with PrgEnv "cc" wrapper
    - either PrgEnv-gnu for gcc; or PrgEnv-cray for craycc
  - GPU CUDA-C kernels compiled with nvcc
    - nvcc -O3 -arch=sm_35
  - PrgEnv "cc" wrapper used for linking
    - Only GPU flag needed: `-lcudart`
      - e.g. no CUDA `-L` flags needed (added in cc wrapper)

For information: Compiling OpenCL

- **Compilation:**
  - module load craype-accel-nvidia35
  - Main CPU code compiled with PrgEnv "cc" wrapper
    - either PrgEnv-gnu for gcc; or PrgEnv-cray for craycc
  - GPU OpenCL kernels compiled with nvcc
  - PrgEnv "cc" wrapper used for linking
    - Only GPU flag needed: `-lOpenCL`

- **Alternatively:**
  - Use PrgEnv-gnu for all compilation
    - still need `-lOpenCL` at linktime