Introduction to Fish

Oralee Nudson
User Consultant

onudson@alaska.edu
Overview

- Connecting to Fish
- Hardware
- Programming Environment
- Compilers
- Queueing System
- Interactive Work
- Software Stack

Image 1: Fish Cabinets
Connecting to Fish

- ARSC supports ssh, sftp and scp clients
- Linux and OS come with native command line versions of ssh, sftp and scp.
- Windows requires you download a terminal and file transfer program (and optionally an X11 server). We have the most experience supporting PuTTY, FileZilla, and Xming.
Other File Transfer Alternatives

➤ File Transfer – Mac and Linux have GUI based file transfer programs (e.g. Fetch and Filezilla) which can be convenient. Other programs should work if they support either sftp or scp. Plain ftp will not work for file transfer files to ARSC systems.

➤ Rsync works too when started via ssh.
Image 2: Filezilla on a Mac
Other Windows Connection Options

- If you prefer the command line environment, Cygwin offers command line ssh, sftp and scp commands as well as an X11 Server.

- For more information on Cygwin see:
  - http://www.arsc.edu/arsc/support/howtos/usingcygwin/index.xml
FISH – Cray XK6m

- **2 Login Nodes:** `{fish1,fish2}.arsc.edu`
  - One six core, 2.6 GHz AMD Istanbul Processor
  - 16 GB of memory per node

- **48- GPU Enabled Sixteen Core Compute Nodes**
  - One sixteen core, 2.1 GHz AMD Interlagos Processor
  - 64 GB of memory per node (4 GB per core)
  - One nVIDIA Tesla X2090 GPU accelerator with 6GB RDDR5 memory

- **32- Twelve Core Compute Nodes**
  - 2- Six core 2.6 GHz AMD Istanbul Processors
  - 32 GB of memory per node (2.5 GB per core)

- **Cray Proprietary Gemini Interconnect**
- 20 TB Lustre $HOME$ file system
- 275 TB Lustre $CENTER$ file system (NEW)

- **Operating System**
  - Cray Linux Environment 4 (CLE4)

*Neat!*
Fish Storage

- Fish supports the ARSC standard storage locations
  - `$HOME`: small, backed up, not purged
  - `$CENTER`: large, not backed up, purged
  - `$ARCHIVE`: no quota, backed up, not purged.
- **NOTE:**
  - `$ARCHIVE` not available from compute nodes! `$ARCHIVE` is only accessible to the login nodes
  - ARSC recommends avoiding accessing `$HOME` in parallel jobs.
  - Purging is not currently enabled on `$CENTER` but may be enabled at a future date (with ample warning to users)
Monitoring Storage Usage

- Quotas are enabled on $HOME and $CENTER. Use the “show_storage” command to list your current usage:

```
fish1 % show_storage
Current Filesystem Quotas as of Thu Mar 8 14:07:47 AKST 2012:

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Used_GB</th>
<th>Soft_GB</th>
<th>Hard_GB</th>
<th>Files</th>
<th>Soft Files</th>
<th>Hard Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>$HOME</td>
<td>3.50</td>
<td>4.19</td>
<td>16.50</td>
<td>66712</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$CENTER</td>
<td>240.58</td>
<td>786.43</td>
<td>1048.58</td>
<td>143791</td>
<td>5000000</td>
<td>52000000</td>
</tr>
</tbody>
</table>
```
Parallel Programming Models

- Threads (OpenMP and pthreads)
- MPI- using the Cray’s MPI with Gemini support.
- Serial Application Autoparallelization (via the PGI and Cray compilers)
- PGAS languages (via the Cray compilers)
- GPU Acceleration
Parallel Programming Models

<table>
<thead>
<tr>
<th>Level</th>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Auto</td>
<td>Automatic parallelization of basic loops. Only available with PGI compilers (use \texttt{–Mconcur=} option)</td>
</tr>
<tr>
<td>Node</td>
<td>OpenMP</td>
<td>Explicit shared memory model using directives to achieve loop level parallelism (use \texttt{–mp} option with PGI compilers)</td>
</tr>
<tr>
<td>System</td>
<td>MPI</td>
<td>Most common and portable method for scalable distributed memory parallelism</td>
</tr>
<tr>
<td>Node</td>
<td>OpenACC</td>
<td>Compiler directives to specify loops and regions of code to be offloaded from CPU to GPU accelerators.</td>
</tr>
<tr>
<td>Node</td>
<td>GPU</td>
<td>Fish has 48 nodes with Nvidia GPU accelerators. The CUDA toolkit 4.0 is available.</td>
</tr>
</tbody>
</table>
Modules

- Fish uses a package called “modules” which allows you to quickly and easily switch from one software version to another.
- This makes it possible for us to provide multiple versions of a software package.
- One person can access the newest version of a package while someone else has an unchanging environment.
Modules

Modules do the following:

- Set your $PATH, $MANPATH, $LD_LIBRARY_PATH
- Set environment variables needed by packages (e.g. NCARG_ROOT for NCL, etc).
- Keep track of what they set so the environment variables and aliases can be unset when a module is unloaded.
By default all accounts have a compiler stack loaded which is set using the modules package.

You can put alternate and additional modules commands in shell login files.

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrgEnv-pgi</td>
<td>Programming environment using the PGI compilers &amp; Cray’s MPI stack (default module).</td>
</tr>
<tr>
<td>PrgEnv-cray</td>
<td>Programming environment using the Cray compiler and Cray’s MPI stack.</td>
</tr>
<tr>
<td>PrgEnv-gnu</td>
<td>Programming environment using GNU compilers &amp; Cray’s MPI stack.</td>
</tr>
</tbody>
</table>
# Sample Module Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Example Use</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>module avail</td>
<td>module avail</td>
<td>lists all available modules for the system.</td>
</tr>
<tr>
<td>module load &lt;pkg&gt;</td>
<td>module load PrgEnv</td>
<td>loads a module file from the environment</td>
</tr>
<tr>
<td>module unload &lt;pkg&gt;</td>
<td>module unload PrgEnv</td>
<td>unloads a module file from the environment</td>
</tr>
<tr>
<td>module list</td>
<td>module list</td>
<td>displays the modules which are currently loaded.</td>
</tr>
<tr>
<td>module switch old new</td>
<td>module switch PrgEnv-mpi PrgEnv-gnu</td>
<td>replaces the module old with module new in the environment</td>
</tr>
<tr>
<td>module purge</td>
<td>module purge</td>
<td>unload all module settings, restoring the environment to the state before any modules were loaded. (Avoid this command on fish!)</td>
</tr>
</tbody>
</table>
Module Use

- Modules will include:
  - abaqus, cuda, fftw, ferret, grads, idl, java, matlab, nco, ncl, petsc, xt-totalview.

- If the default version of a package is not what you want, check to see if a newer version is available.

- The “modules avail” command lets you see all versions of a package (e.g. “module avail ncl” lists all ncl modules)
Module Command Examples

```
fish1 % module avail PrgEnv-pgi
fish1 % module list
fish1 % module load PrgEnv-pgi
fish1 % module list
fish1 % module swap PrgEnv-pgi/4.0.46 PrgEnv-cray/4.0.46
fish1 % module list
```
If you compile with XTPE_LINK_TYPE=dynamic and used non-default modules, be sure to load those same modules in your job script.

```bash
#!/bin/bash
#PBS -q standard
...

source /opt/modules/default/init/modules.sh
module swap pgi pgi/12.6.0
aprun -n 64 ./wrf.exe
```
Last Words on Modules

➢ You can create your own modules if you want to maintain multiple versions of programs you build yourself.

➢ Contact consult@arsc.edu if you’d like to learn how.
## Compilers on Fish

<table>
<thead>
<tr>
<th>Item</th>
<th>Cray</th>
<th>PGI</th>
<th>GNU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran 77</td>
<td>ftn</td>
<td>ftn</td>
<td>ftn</td>
</tr>
<tr>
<td>Fortran 90/95</td>
<td>ftn</td>
<td>ftn</td>
<td>ftn</td>
</tr>
<tr>
<td>C</td>
<td>cc</td>
<td>cc</td>
<td>cc</td>
</tr>
<tr>
<td>C++</td>
<td>CC</td>
<td>CC</td>
<td>CC</td>
</tr>
<tr>
<td>Serial Debugger</td>
<td>lgdb</td>
<td>lgdb</td>
<td>lgdb</td>
</tr>
<tr>
<td>Parallel Debugger</td>
<td>totalview</td>
<td>totalview</td>
<td>totalview</td>
</tr>
<tr>
<td>Performance Analysis</td>
<td>Cray</td>
<td>Cray</td>
<td>Cray</td>
</tr>
<tr>
<td></td>
<td>perftools</td>
<td>perftools</td>
<td>perftools</td>
</tr>
<tr>
<td>Default MPI Module</td>
<td>PrgEnv-cray</td>
<td>PrgEnv-pgi</td>
<td>PrgEnv-gnu</td>
</tr>
</tbody>
</table>
### A few PGI compiler options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>Generate object file but don’t link</td>
</tr>
<tr>
<td>-g</td>
<td>Add debugging information</td>
</tr>
<tr>
<td>-O3</td>
<td>Higher level of optimization (default is –O2)</td>
</tr>
<tr>
<td>-fast</td>
<td>Higher level of optimization than –O3</td>
</tr>
<tr>
<td>-Mipa</td>
<td>Perform interprocedural analysis</td>
</tr>
<tr>
<td>-Mconcur</td>
<td>Enable autoparallelization</td>
</tr>
<tr>
<td>-mp</td>
<td>Enables parallelization via OpenMP directives</td>
</tr>
<tr>
<td>-Minfo</td>
<td>Provides additional information on optimizations done by the compiler. This flag has numerous options (e.g. –Minfo=mp,par)</td>
</tr>
<tr>
<td>-Mneginfo</td>
<td>Provides additional information on why optimizations are <em>not done</em> by the compiler.</td>
</tr>
<tr>
<td>-ta=nvidia</td>
<td>Enables GPU parallelization via OpenACC directives</td>
</tr>
</tbody>
</table>
Batch System

- **Batch queueing**
  - Allows job scheduling on shared compute nodes
  - Requires users to specify resource requests
  - Ensures that nodes are shared fairly
  - Manages resources to maximize throughput

- Fish uses the Torque/Moab/ALPS batch queueing system, which is based on PBS, the Portable Batch System
Batch Script Contents

1. **Queueing Commands**
   - Shell dialect (e.g. bash, ksh)
   - Execution queue to use
   - Job walltime
   - Number of nodes required

2. **Commands to Execute**
   - File/Directory Manipulations
   - Code to execute (call with aprun)
Torrent Script - MPI Example

#!/bin/bash
#PBS -q standard
#PBS -l walltime=8:00:00
#PBS -l nodes=4:ppn=12
#PBS -j oe

cd $PBS_O_WORKDIR

NP =${( $PBS_NUM_NODES * $PBS_NUM_PPN )}
aprun -n $NP ./myprog
Torque Script - OpenMP

#!/bin/bash
#PBS -q standard
#PBS -l walltime=8:00:00
#PBS -l nodes=1:ppn=12
#PBS -j oe

cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=12
aprun -n 1 -d 12 ./myprog
Torque Script – MPI w/ OpenMP

#!/bin/bash
#PBS -q gpu
#PBS -l walltime=8:00:00
#PBS -l nodes=4:ppn=16
#PBS -j oe

cd $PBS_O_WORKDIR

export OMP_NUM_THREADS=$PBS_NUM_PPN
NP=$PBS_NUM_NODES
aprun -n $NP -d ${OMP_NUM_THREADS} ./myprog
Common PBS commands

- `qsub job.pbs` - queue the script “job.pbs” to be run by PBS.

- `qstat` - list jobs which are queued, running or recently completed.

- `qdel jobid` - delete a job with ID=jobid from the qstat command. The jobid is provided following a successful qsub.
Common Queues

- **standard** - regular CPU only work. Uses 12 core nodes. 24 hour max walltime.
- **gpu** - CPU + GPU work. Uses 16 core nodes. 24 hour max walltime.
- Use “**news queues**” to find out more details on the number of nodes and walltime allowed per queue.
Example Batch Queue Use

```plaintext
# Submit job
fish1 % qsub run_hello.cmd
2067.sdb

# Check job status
fish1 % qstat 2067.sdb
Job id     Name             User            Time Use S Queue
------------------ --------------- -------- - ----- 
2067.sdb     run_hello.cmd    fred                     0 Q gpu

# Check job status a bit later
fish1 % qstat 2067.sdb
Job id     Name             User            Time Use S Queue
------------------ --------------- -------- - ----- 
2067.sdb     run_hello.cmd    fred          00:00:01 C  gpu

# Output from job is returned to the working directory by default.
fish1 % ls run_hello.cmd.o2067
run_hello.cmd.o206782
```
Select Torque Environment Variables

- Torque sets environment variables for use within job scripts.
  - `$PBS_JOBID` – Job id assigned by torque.
  - `$PBS_NUM_NODES` – Number of nodes assigned to job.
  - `$PBS_NUM_PPN` – Number of cores assigned per node.
  - `$PBS_O_WORKDIR` – Original working directory on job submission.
  - `$PBS_ARRAYID` – Assigned to the array index when `-t` is used.
Example Torque Environment Variable Use

```
#!/bin/bash
#PBS -l nodes=4:ppn=12
#PBS -q standard
#PBS -j oe
#PBS -l walltime=1:00:00

cd $PBS_O_WORKDIR

NP=$(( $PBS_NUM_NODES * $PBS_NUM_PPN ))
aprun -n ${NP} ./a.out > job.${PBS_JOBID}.out 2>&1
```
More Torque Features

➢ There a number of other features available and may be useful:

➢ **Job dependencies** – Make one job run after another.

➢ **Job arrays** – Submit an array of jobs.

➢ **Command line options** – Override settings in a torque script when a job is submitted.
Interactive Work

- There are two fish login nodes: fish1.arsc.edu and fish2.arsc.edu
- You may run short serial work on fish1 and fish2, however CPU intensive, memory intensive, or parallel work should be run on the compute nodes using an interactive job.
- Interactive compute nodes are subject to availability.
Example Interactive Work

# Load a Programming Environment
```bash
fish2 % module load PrgEnv-gnu
```

# Compile the code
```bash
fish2 % make
cal hello.c -O3 -o hello.exe
```

# Start an interactive job
```bash
fish2 % qsub -lnodes=2:ppn=12 -q standard -I
```

# run the program on the compute nodes.
```bash
nid00186 % cd mpi/
# the “aprun” is important! Even if you run a serial job!
nid00186 % aprun -n 4 ./hello.exe
```
```
nid00064: hello world- 0
nid00064: hello world- 1
nid00064: hello world- 3
nid00064: hello world- 2
```
Fish has a software stack based on requests from people on campus. If we have a package installed there’s likely already someone on campus using that package.

When possible we put example job scripts and test cases in `$SAMPLES_HOME`

If you want a test case for a package, let us know and we will make one.
Additional Information

- Watch [www.arsc.edu](http://www.arsc.edu) for upcoming training and additional information about ARSC.
- More info at
  - ARSC How to: [http://www.arsc.edu/arsc/support/howtos/usingfish](http://www.arsc.edu/arsc/support/howtos/usingfish)
  - PGI Compilers: [http://www.pgroup.com/resources/docs.htm](http://www.pgroup.com/resources/docs.htm)
  - Cray Compilers: [http://docs.cray.com](http://docs.cray.com)
  - Free CUDA training: [https://www.udacity.com/course/cs344](https://www.udacity.com/course/cs344)
Questions and Feedback

If you have additional questions, feel free to contact us:

ARSC Help Desk
Phone: (907) 450-8602
Email: consult@arsc.edu
Web: http://www.arsc.edu/support

We welcome your feedback on this class!

http://www.arsc.edu/arsc/support/training/trainingevalform/index.xml
Practice Exercises

- cd $CENTER
- mkdir fishExercises; cd fishExercises;
- cp –r $SAMPLES_HOME/training/introToFish .