Programming Environments on SahasraT (Cray-XC40 system)

Dr. J. Lakshmi

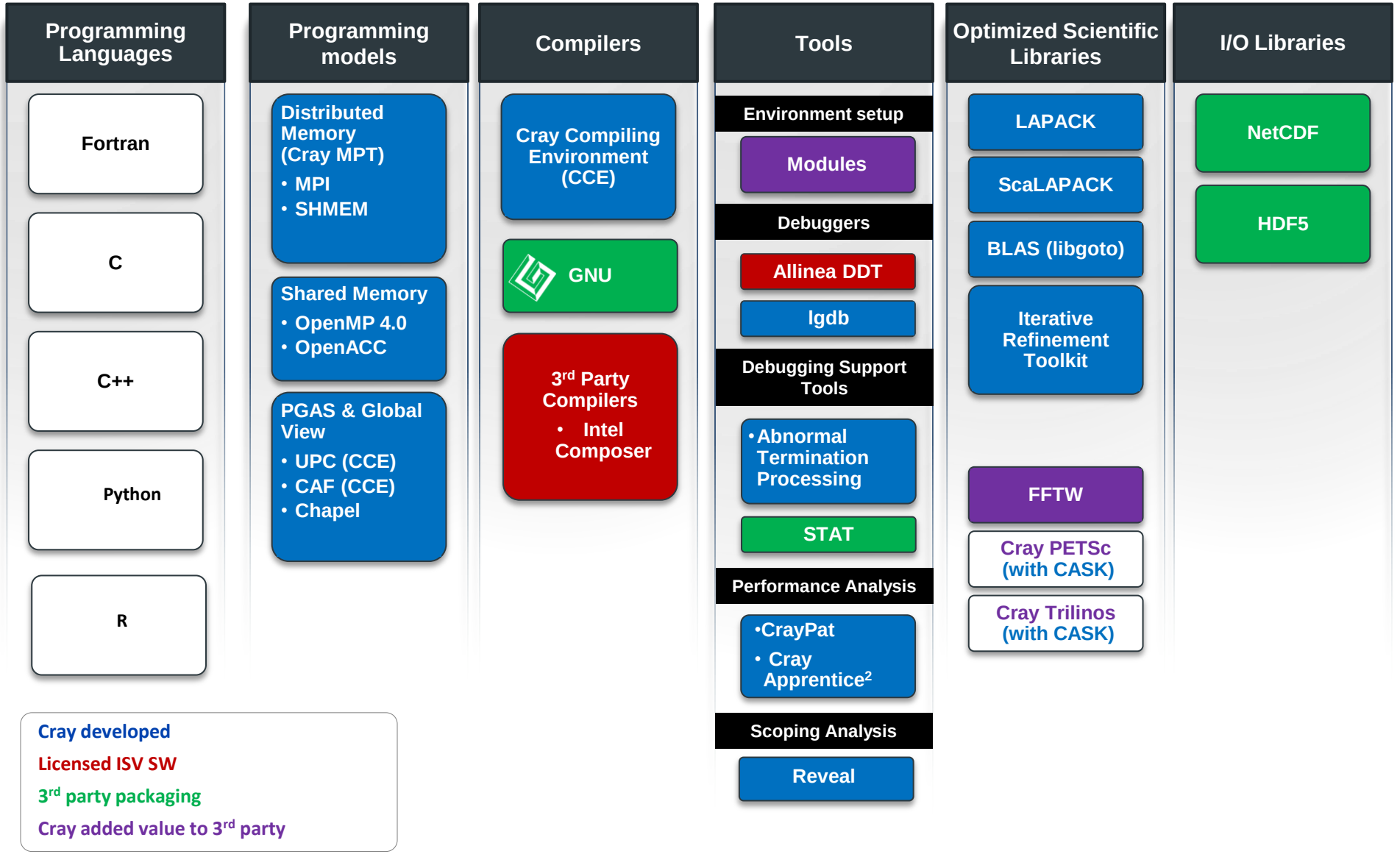
SERC, Indian Institute of Science

Bangalore-12, India

SahasraT: Massively Parallel Processing Class Machine

- Built to enable co-ordinated processing among multiple processing elements.
- Hardware composed of several independent distributed nodes interconnected with a high speed – high bandwidth network and storage.
- Each node complemented with system software and libraries to support parallel processing program executions.

Overview of Programming Envs. on SahasraT



SahasraT Programming Environment

http://www.serc.iisc.ac.in/software/programming-environment-cray/

- A cross-compiler environment
  - Compiler runs on a login node
  - Executable runs on the compute nodes

- Cray written compiler driver scripts
  - Compute Node Linux (CNL) compiler options
  - CNL system libraries and header files
  - Compiler specific programming environment libraries
  - Latest standards compliant

- Modules utility
  - Consists of the module command and module files
  - Initialized environment for a specific compiler
  - Allows easy swapping of compilers and compiler versions

Cray Programming Environment

- The default compiler on XC system
  - Specifically designed for HPC applications
  - Takes advantage of Cray's experience with automatic vectorization and shared memory parallelization
- Excellent standards support for multiple languages and programming models
  - Fortran 2008 standards compliant
  - C++11 compliant
  - OpenMP 4.5
  - OpenACC 2.0 compliant
- Full integrated and optimized support for PGAS languages
  - UPC 1.2 and Fortran 2008 coarray support
  - No preprocessor involved
  - Full debugger support (With Allinea DDT)
- OpenMP and automatic multithreading fully integrated
  - Share the same runtime and resource pool
  - Aggressive loop restructuring and scalar optimization done in the presence of OpenMP
  - Consistent interface for managing OpenMP and automatic multithreading

Introduction to modules

- On SahasraT runtime environment is established using environment modules
- Provides for the dynamic modification of a user's environment via modulefiles
- Each modulefile contains the information needed to configure the shell for an application
  - Typically alter or set shell environment variables such as PATH, MANPATH, etc.
- Modules can be loaded and unloaded dynamically and atomically, in an clean fashion
- All popular shells are supported
  - including *bash*,*ksh*, *zsh*, *sh*, *csh*, *tcsh*, as well as some scripting languages such as *perl* and *python*
- Useful in managing different applications and versions of applications
- Can be bundled into metamodules
  - load an entire suite of different applications

Runtime Environment Setup

- The Cray XC system uses modules in the user environment 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 added automatically to the Programming Environment as a new version, while earlier versions are retained to support legacy applications
- You can use the default version of an application, or you can choose another version by using Modules system commands
- Users can create their own modules or admins can install site specific modules available to many users

Module commands

`$> module list`	• Prints actual loaded modules
`$> module avail [-S str]`	• Prints all module available containing the specified string
`$> module (un)load [mod_name/version]`	• Adds or remove a module to the actual loaded list • If no version specified, loading the default version
`$> module switch [mod1] [mod2]`	• Unload mod1 and load mod2 • E.g. to change versions of loaded modules
`$> module whatis/help [mod]`	• Prints the module (short) description
`$> module show [mod]`	• Prints the environmental modification
`$> module load user_own_modules`	• add $HOME/privatemodules to the list of directories that the module command will search for modules

SahasraT Modules

```
Currently Loaded Modulefiles:
  1) modules/3.2.10.6
  2) alps/6.4.3-6.0.4.1_2.1__g92a2fc0.ari
  3) nodestat/2.3.78-6.0.4.0_7.2__gbe57af8.ari
  4) sdb/3.3.729-6.0.4.0_16.2__gb405b22.ari
  5) udreg/2.3.2-6.0.4.0_12.2__g2f9c3ee.ari
  6) ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari
  7) gni-headers/5.0.11-6.0.4.0_7.2__g7136988.ari
  8) dmapp/7.1.1-6.0.4.0_46.2__gb8abda2.ari
  9) xpmem/2.2.2-6.0.4.1_18.2__g43b0535.ari
 10) llm/21.3.446-6.0.4.0_20.1__gbe30105.ari
 11) nodehealth/5.4.0-6.0.4.0_12.4__g3427370.ari
 12) system-config/3.4.2456-6.0.4.1_13.1__g56652eb.ari
 13) sysadm/2.4.119-6.0.4.0_14.2__gcab7125.ari
 14) lustre-utils/2.3.5-6.0.4.0_10.2__g3d4bf80.ari
 15) Base-opts/2.4.123-6.0.4.0_10.1__g6460790.ari
 16) cce/8.7.3
 17) craype-network-aries
 18) craype/2.5.15
 19) cray-libsci/18.07.1
 20) pmi/5.0.14
 21) rca/2.2.11-6.0.4.0_13.2__g84de67a.ari
 22) atp/2.1.2
 23) perftools-base/7.0.2
 24) PrgEnv-cray/6.0.4
 25) cray-mpich/7.7.2
 26) pbs/default
```

Compiler and Version

Compiler driver and version

Modifying the default environment

Loading, swapping or unloading modules:

- The default version of any individual module can be loaded by name
  e.g.: `module load cce`

- A specific version can be specified after the forward slash
  e.g.: `module load cce/8.7.7`

- Modules can be swapped out in place
  e.g.: `module swap cce cce/8.7.0`

- Or removed entirely
  e.g.: `module unload perftools`

Modifying the default environment

- **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

What module does ?

```
crayadm@login2:~> module show cce
-------------------------------------------------------------------
/opt/cray/pe/modulefiles/cce/8.7.3:

conflict         cce
setenv           GCC_X86_64 /opt/gcc/6.1.0/snos
setenv           CRAY_BINUTILS_ROOT_X86_64 /opt/cray/pe/cce/8.7.3/binutils/x86_64/x86_64-pc-linux-gnu/../
setenv           CRAY_BINUTILS_BIN_X86_64 /opt/cray/pe/cce/8.7.3/binutils/x86_64/bin
setenv           LINKER_X86_64 /opt/cray/pe/cce/8.7.3/binutils/x86_64/x86_64-pc-linux-gnu/bin/ld
setenv           ASSEMBLER_X86_64 /opt/cray/pe/cce/8.7.3/binutils/x86_64/x86_64-pc-linux-gnu/bin/as
setenv           FTN_X86_64 /opt/cray/pe/cce/8.7.3/cce/x86_64
setenv           CC_X86_64 /opt/cray/pe/cce/8.7.3/cce/x86_64
setenv           CRAY_CXX_IPA_LIBS_X86_64 /opt/cray/pe/cce/8.7.3/cce/x86_64/lib/libcray-c++-rts.a
setenv           CRAYLIBS_X86_64 /opt/cray/pe/cce/8.7.3/cce/x86_64/lib
prepend-path     INCLUDE_PATH_X86_64 /opt/cray/pe/cce/8.7.3/cce/x86_64/include/craylibs
setenv           GCC_AARCH64 /opt/gcc-cross-aarch64/6.1.0/aarch64
setenv           CRAY_BINUTILS_ROOT_AARCH64 /opt/cray/pe/cce/8.7.3/binutils/cross/x86_64-aarch64/aarch64-linux-gnu/../
setenv           CRAY_BINUTILS_BIN_AARCH64 /opt/cray/pe/cce/8.7.3/binutils/cross/x86_64-aarch64/aarch64-linux-gnu/bin
setenv           LINKER_AARCH64 /opt/cray/pe/cce/8.7.3/binutils/cross/x86_64-aarch64/aarch64-linux-gnu/bin/ld
setenv           ASSEMBLER_AARCH64 /opt/cray/pe/cce/8.7.3/binutils/cross/x86_64-aarch64/aarch64-linux-gnu/bin/as
setenv           CRAY_CXX_IPA_LIBS_AARCH64 /opt/cray/pe/cce/8.7.3/cce/aarch64/lib/libcray-c++-rts.a
setenv           CRAYLIBS_AARCH64 /opt/cray/pe/cce/8.7.3/cce/aarch64/lib
prepend-path     INCLUDE_PATH_AARCH64 /opt/cray/pe/cce/8.7.3/cce/aarch64/include/craylibs
setenv           CRAYLMD_LICENSE_FILE /opt/cray/pe/cce/cce.lic
setenv           CRAY_BINUTILS_ROOT /opt/cray/pe/cce/8.7.3/binutils/x86_64/x86_64-pc-linux-gnu/../
setenv           CRAY_BINUTILS_VERSION /opt/cray/pe/cce/8.7.3
setenv           CRAY_BINUTILS_BIN /opt/cray/pe/cce/8.7.3/binutils/x86_64/bin
setenv           CRAY_CCE_SHARE /opt/cray/pe/cce/8.7.3/cce/x86_64/share
setenv           CRAY_CXX_IPA_LIBS /opt/cray/pe/cce/8.7.3/cce/x86_64/lib/libcray-c++-rts.a
setenv           CRAY_FTN_VERSION 8.7.3
setenv           CRAY_CC_VERSION 8.7.3
```

“Meta”-Module PrgEnv-X

- PrgEnv-X is a “meta”-module
  - loading several modules,
    - including the compiler,
    - the corresponding mathematical libs,
    - MPI,
    - system environment needed for the compiler wrappers

```
crayadm@login2:~> module show PrgEnv-cray
-------------------------------------------------------------------
/opt/cray/pe/modulefiles/PrgEnv-cray/6.0.4:

conflict        PrgEnv
conflict        PrgEnv-x1
conflict        PrgEnv-x2
conflict        PrgEnv-gnu
conflict        PrgEnv-intel
conflict        PrgEnv-pgi
conflict        PrgEnv-pathscale
conflict        PrgEnv-cray
setenv          PE_ENV CRAY
prepend-path    PE_PRODUCT_LIST CRAY
setenv          cce_already_loaded 1
module          load cce/8.7.3
setenv          craype_already_loaded 1
module          swap craype/2.5.15
module          swap cray-mpich cray-mpich/7.7.2
module          load cray-libsci
module          load pmi
module          load rca
module          load atp
module          load perftools-base
setenv          CRAY_PRGENVCRAY loaded
```

Cray modules

- **cray-mpich : Loads optimized MPICH3 module**
- **cray-shmem : Loads optimized SHMEM module**
- **cray-libsci : Loads BLAS, LAPACK, BLACS, FFT, ScaLAPACK etc pacages**
- **Debuggers : cray lgdb**
- **Perf tools : craypat, Apprentice2, Reveal etc.**
- **Other libs : cray-petsc, cray-trillinos etc.**

Processor modules

- **Cray XC system :**
  - **module load craype-haswell** (Sahasrat – For CPU)
  - **module load craype-ivybridge** (Sahasrat – For GPU Host)

    Note : x86_64 instruction will be loaded by default
- **Accelerators**
  - **module load craype-mic-knl** (Sahasrat – For KNL)
  - **module load craype-accel-nvidia35 (Sahasrat - Kepler GPU)**

Summary

- Various applications in various versions available
  ```
  $> module avail # lists all
  $> module avail cce    # cce*
  ```
- Dynamic modification of a user's environment
  ```
  $> module (un)load PRODUCT/MODULE
  ```
  - E.g. PrgEnv-xxx changes compilers, linked libraries, and environment variables
- Version management
  ```
  $> module switch prod_v1 prod_v2
  $> module switch PrgEnv-cray PrgEnv-gnu
  $> module switch cce  cce/8.7.0
  ```

- Metamodules feature bundles multiple modules
- Can create your own (meta)modules
- Module tool takes care
  - Environment variables
    - PATH, MANPATH, LD_LIBRARY_PATH, LM_LICENSE_FILE,....
  - Compiler and linker arguments of loaded products
    - Include paths, linker paths, …

Compiling Applications on SahasraT

Compiler Driver Wrappers

- All applications that will run in parallel on the Cray XC should be compiled with the standard language wrappers.
- Compiler drivers for each language are:
  - cc – wrapper around the C compiler
  - CC – wrapper around the C++ compiler
  - ftn – wrapper around the Fortran compiler
- 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 use the original compiler

E.g. To compile prog1.f90:

```
$> ftn –o myprog myprog.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/8.7.0`

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 …'
- You must not build for execution of codes on login nodes

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

- Do not call compilers directly. Use cray compiler drivers
  - cc : For c programming language
  - CC : For C++ programming language
  - ftn : For fortran programming language

- 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-haswell

- Check Makefile for compilers and related flags

  Eg:- Use standard language wrappers and related compiler flags

For details on the programming environment on SahasraT access Link:

http://www.serc.iisc.ac.in/software/programming-environment-cray/

Application Execution on SahasraT

http://www.serc.iisc.ac.in/pbspro-batch-scheduler-cray/

Modes of program execution

- Interactive program execution:
  - Most of us know of executing our program by typing out the executable name on the cursor prompt in a terminal window and hitting the return key!
  - The current state of the execution is observed by the contents displayed in the terminal window where program execution was initiated.
  - Most probably the output, if any, generated by the program is also displayed in this window.
  - Return to displaying the cursor prompt is an indication that the program execution is completed.
- Whatif there are other programs already running on the system?
- Whatif your program initiates execution on multiple machines/nodes of a system?
- Many HPC systems do not support interactive program execution!

Batch execution

- Batch mode of execution refers to program execution in background.
- Most HPC jobs can use resources of multiple nodes.
- Time sharing a resource across multiple jobs is possible, it may not however yield deterministic performance for all jobs.
- Given the HPC systems cost to acquire and maintain, it makes economic sense to reduce idle time on the system.
- Most resource policies on the HPC systems tend to be exclusive.
- User jobs are accepted and typically wait in queues before adequate resources are available to the job and then it is scheduled for execution.

Jobs on SahasraT

- SahasraT accepts only jobs in batch mode through PBS job scripts.
- A job script contains all that is necessary for a job to execute, like
  - Environment setup
  - Execution resources and how the job wants to use the resources
  - Description of input, output and error files associated with the job.
- Procedure to execute:
  - Prepare a job script
  - Submit to PBSPro workload manager
  - PBSPro puts your job in job queues waiting for necessary resources to become available
  - Once resources are available job is released for execution based on the system policy of usage
  - After a job is completed it is deleted from the PBSPro batch system.

PBS resources on SahasraT

- There are different types of resources that can be used by a job on SahasraT.
  - Login nodes: mostly used to prepare and submit jobs: DO NOT EXECUTE on login nodes
  - Service nodes: invisible to users but necessary for jobs to successfully get submitted and execute applications
  - Compute nodes: nodes which provide the resources for submitted jobs
    - CPU nodes: Pure Intel Haswell cores used mainly by OpenMP/MPI jobs
    - GPU nodes: Nvidia K40 nodes meant to execute CUDA codes
    - KNL nodes: Intel KNL nodes meant to execute OpenMP/MPI codes

PBS Job Scripts

- User job scripts typically contain two types of statements
  1. PBS directives that provide information to the workload manager on the type, number and nature of resources your job requires. Other information associated with aspects of job requirements also get to be specified by these directives.
  2. Serial commands that are executed by the login node, e.g.
     - Job environment setup and I/O staging commands
     - e.g. (`rm`, `cd`, `mkdir` etc.)
  3. Parallel executables that run on compute nodes
     - Launched using a special command (`aprun`)

Sample PBS job script

```
#!/bin/sh
#PBS -N jobname
#PBS -l select=10:ncpus=24 //select 10 compute nodes with 24 cores each
#PBS -l walltime=4:00:00   //maximum walltime for a job to run 4 hours
#PBS -l place=scatter     // node topology description asked for job
#PBS -l accelerator_type="None“ // type of compute node specification
//add the above line only for idqueue,small,small72,medium queue

. /opt/modules/default/init/sh //can include module load command for env
cd {/path of executable}  // restrict this to use /mnt/lustre on this machine
#Launch the parallel job
        aprun -j 1 -n 240 -N 24 ./name_of_executable
//Using 240 MPI processes and 24 MPI processes per node
```

PBS and ALPS on the SahasraT (Introduction)

- SahasraT uses the PBS workload manager and the Application Level Placement Scheduler (ALPS)

- Most useful commands for your daily work:

```
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
```

- Detailed 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.

Any questions?
mailto: helpdesk.serc@auto.iisc.ac.in

Thankyou and Happy computing!