Introduction to SahasraT

RAVITEJA K
Applications Analyst, Cray inc
E Mail : raviteja@cray.com

1. Introduction to SahasraT
2. Cray Software stack
3. Compile applications on XC
4. Run applications on XC

What is Supercomputer?

- Broad term for one of the fastest computer currently available.

- Designed and built to solve difficult computational problems on extremely large jobs that could not be handled by no other types of computing systems.

Characteristics :

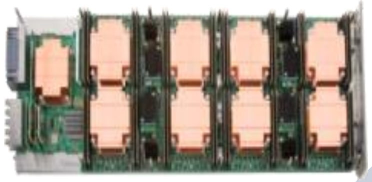
- The ability to process instructions in parallel (Parallel processing)
- The ability to automatically recover from failures (Fault tolerance)

sahasrat

What is SahasraT?

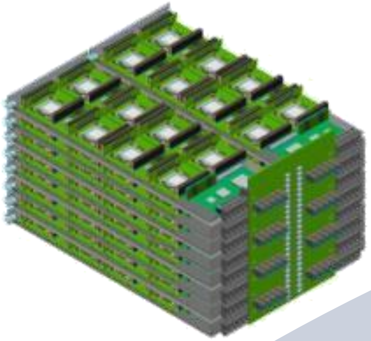
- SahasraT is Country’s first petaflops supercomputer.
- SahasraT : Sahasra means “Thousand” and T means “Teraflop”
- Built and designed by Cray (XC40 Series)

Cray XC System Building Blocks



Compute Blade

4 Compute Nodes



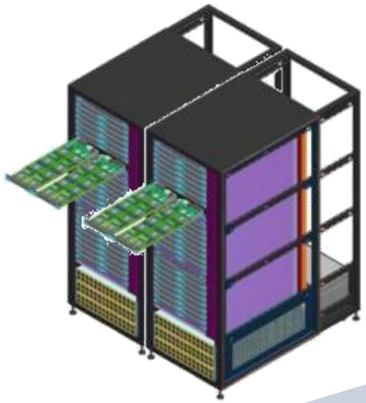
Chassis

Rank 1 Network

16 Compute Blades

No Cables

64 Compute Nodes



Group

Rank 2 Network

Passive Electrical Network

2 Cabinets

6 Chassis

384 Compute Nodes

System

Rank 3 Network

Active Optical Network

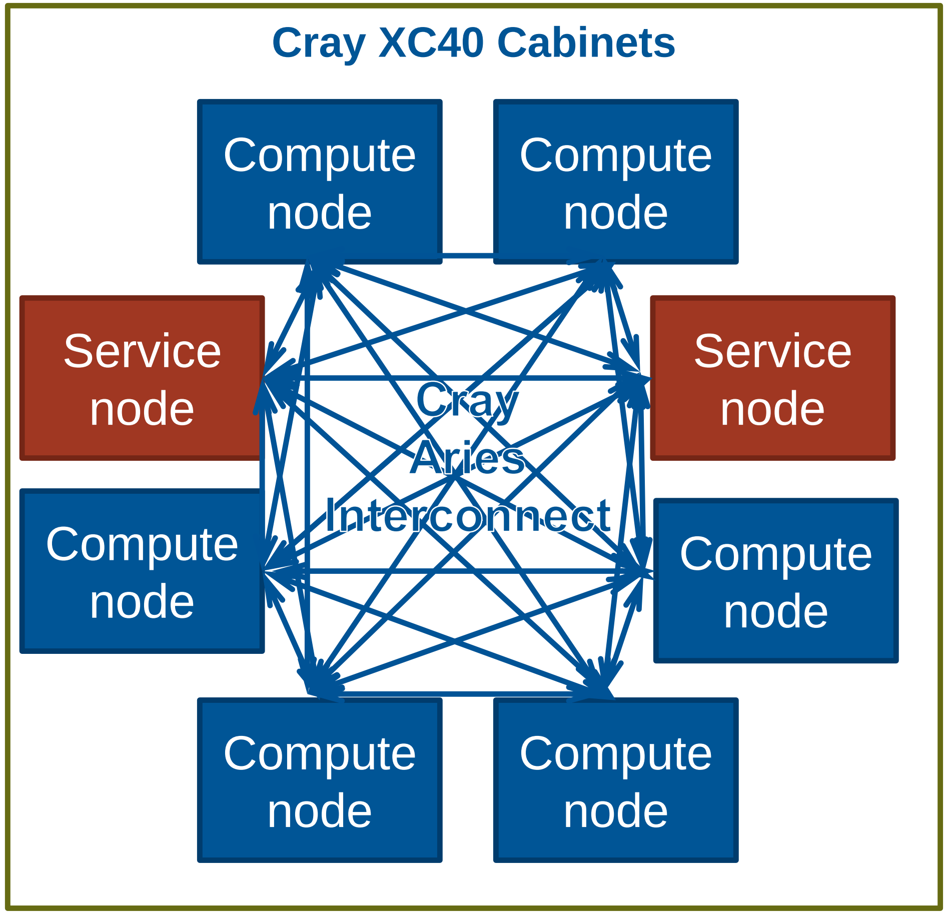
Hundreds of Cabinets

Up to 10s of thousands of nodes

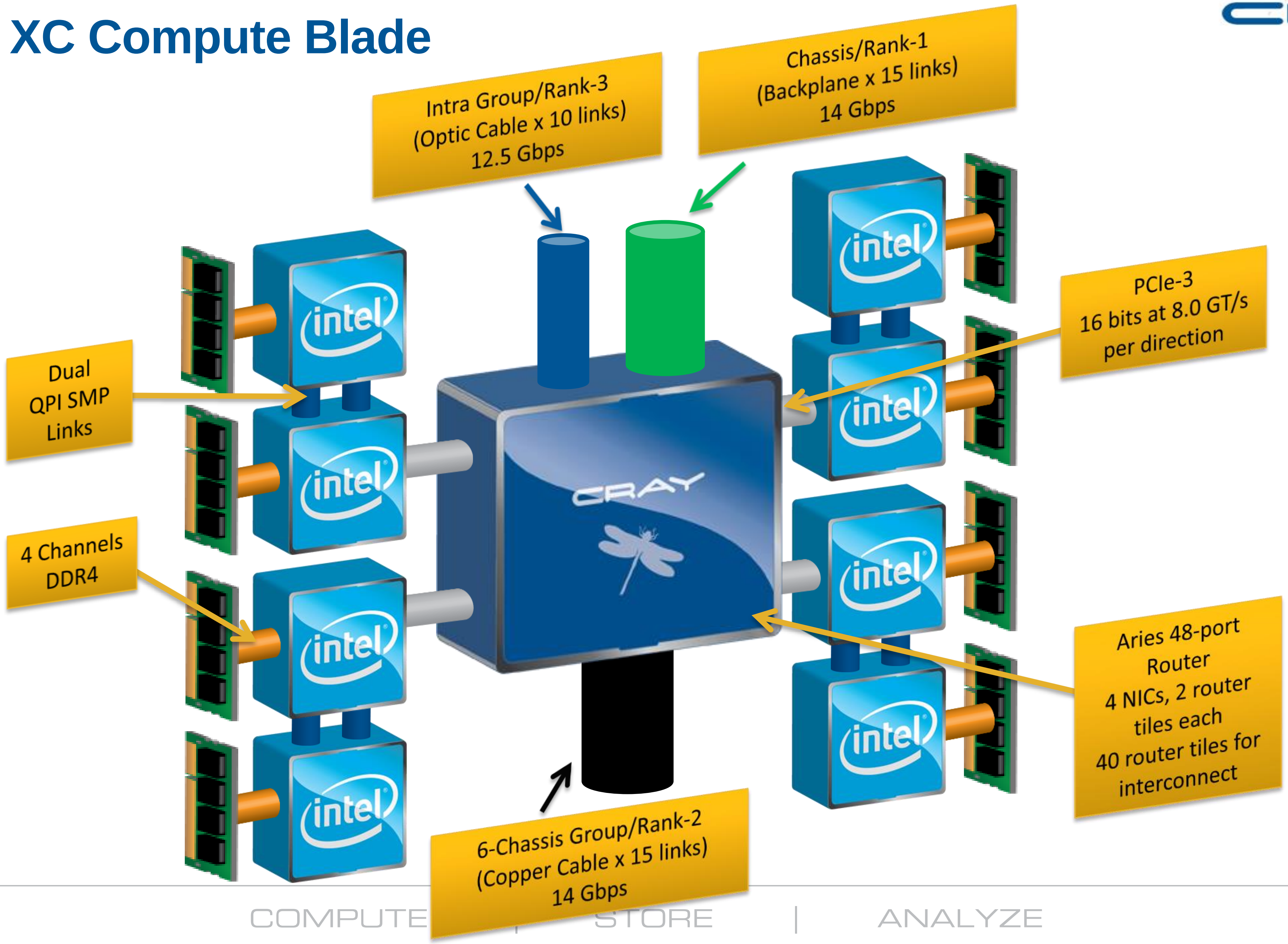
Connecting nodes together: Aries

Obviously, to function as a single supercomputer, the individual nodes must have method to communicate with each other.

All nodes in the interconnected by the high speed, low latency Cray Aries Network.



XC Compute Blade



Intra Group/Rank-3
(Optic Cable x 10 links)
12.5 Gbps

Chassis/Rank-1
(Backplane x 15 links)
14 Gbps

PCIe-3
16 bits at 8.0 GT/s
per direction

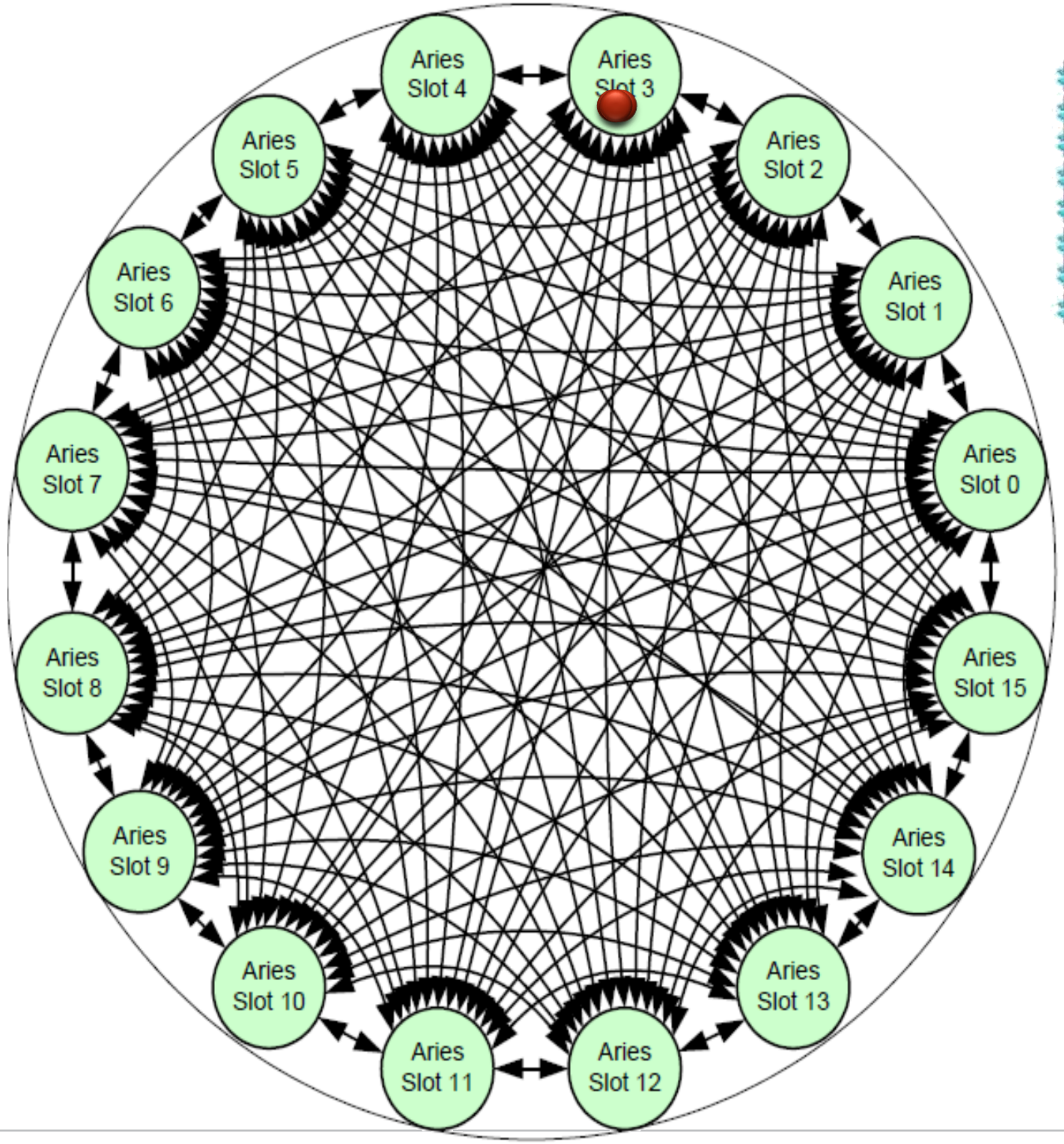
Dual
QPI SMP
Links

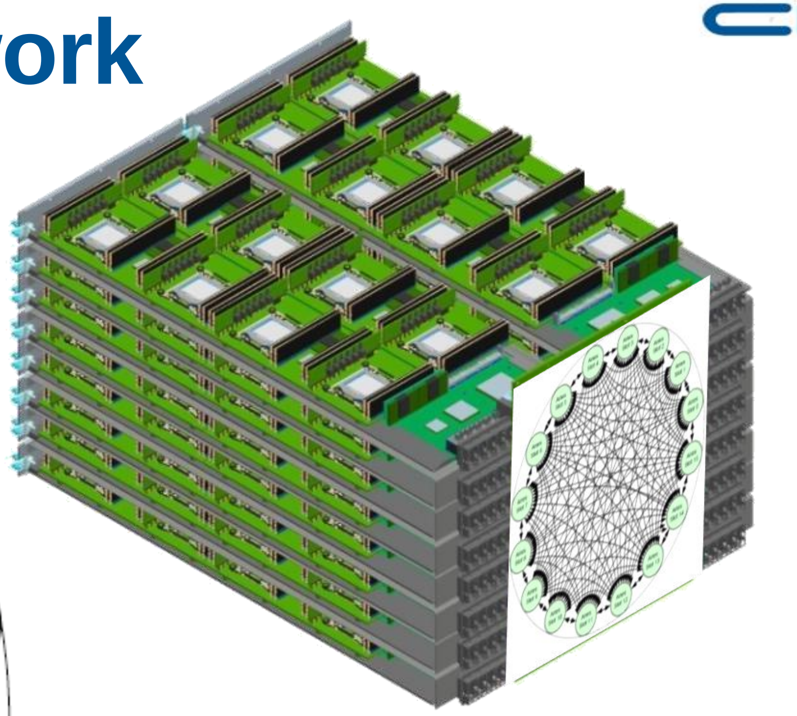
4 Channels
DDR4

Aries 48-port
Router
4 NICs, 2 router
tiles each
40 router tiles for
interconnect

6-Chassis Group/Rank-2
(Copper Cable x 15 links)
14 Gbps

Cray XC Rank1 Backplane Network





- Chassis with 16 compute blades
- 128 Sockets
- Inter-Aries communication over backplane
- Per-Packet adaptive Routing

Types of nodes:

Service nodes:

- Its purpose is managing running jobs, but you can access using an interactive session.
- It runs a full version of the CLE operating system (all libraries and tools available)
- They are shared resources, mistakes and misbehaviour can effect jobs of other users(!).

SahasraT hardware configuration:

- Based on Cray Linux Environment.
- Consists of
  - CPU based Cluster
    - Equipped with Intel Haswell processors
  - Accelerated based Cluster
    - Equipped with Nvidia GPUs
    - Equipped with Intel KNLs
  - 2 PB High Speed storage (Lustre file system)

Types of nodes:

Compute nodes:

- These are the nodes on which jobs are executed
- These nodes, includes GPU and KNL accelerated cards.
- It runs Compute Node Linux, a version of the OS optimised for running batch workloads
- They can only be accessed by starting jobs with aprun (in conjunction with a batch system)
- They are exclusive resources that may only be used by a single user.

System configuration: Compute (H/W)

Compute Node :

No. of Nodes	: 1376
Processor type	: Intel Haswell
No. of cores per node	: 12 cores
Clock Rate	: 2.5 GHz
Memory per Node	: 128 GB
Total Memory	: 176 TB

Accelerator Node :

Accelerator	: Intel XeonPhi 7120
No. of Nodes	: 24
No. of Cores per node	: 64 core
Clock Rate	: 1.3 GHz
Memory per node	: 96 GB
Total Peak Performance	: ~60 TFLOPS

System configuration: Compute (H/W)

GPU Node :

No. of Nodes : 44
Processor type : Nvidia tesla K 40
No. of Cores per node : 2880 cores
Memory per Node : 12GB GDDR5
CPU Cores : Ivybridge

SahasraT Access details:

- Accessed from within the IISc network
- Use sahasrat.serc.iisc.ernet.in address to login

  Eg: ssh computational_id@sahasrat.serc.iisc.ernet.in
- Use admin supply password to log in then change password – follow the institute procedure for this

Cray Software

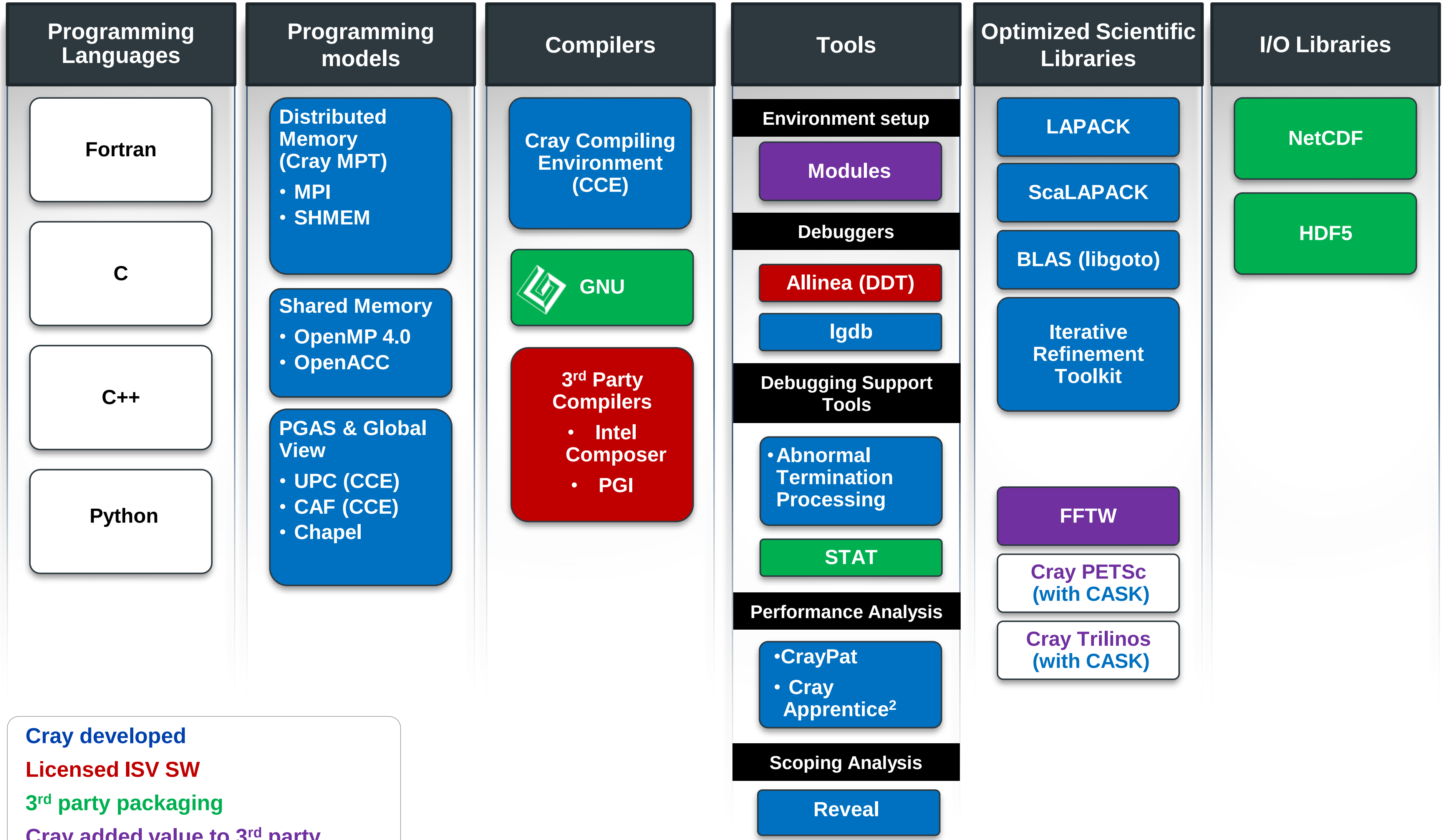
What is Cray?

- **Cray systems are designed to be High Productivity as well as High Performance Computers**

- **The Cray Programming Environment (PE) provides a simple consistent interface to users and developers.**
  - Focus on improving scalability and reducing complexity

- **The default Programming Environment provides:**
  - the highest levels of application performance
  - a rich variety of commonly used tools and libraries
  - a consistent interface to multiple compilers and libraries
  - an increased automation of routine tasks

Cray’s Supported Programming Environment



Programming Languages
- Fortran
- C
- C++
- Python

Programming models
- Distributed Memory (Cray MPT)
  - MPI
  - SHMEM
- Shared Memory
  - OpenMP 4.0
  - OpenACC
- PGAS & Global View
  - UPC (CCE)
  - CAF (CCE)
  - Chapel

Compilers
- Cray Compiling Environment (CCE)
- GNU
- 3rd Party Compilers
  - Intel Composer
  - PGI

Tools
- Environment setup
  - Modules
- Debuggers
  - Allinea (DDT)
  - lgdb
- Debugging Support Tools
  - Abnormal Termination Processing
  - STAT
- Performance Analysis
  - CrayPat
  - Cray Apprentice^2
- Scoping Analysis
  - Reveal

Optimized Scientific Libraries
- LAPACK
- ScaLAPACK
- BLAS (libgoto)
- Iterative Refinement Toolkit
- FFTW
- Cray PETSc (with CASK)
- Cray Trilinos (with CASK)

I/O Libraries
- NetCDF
- HDF5

Cray developed

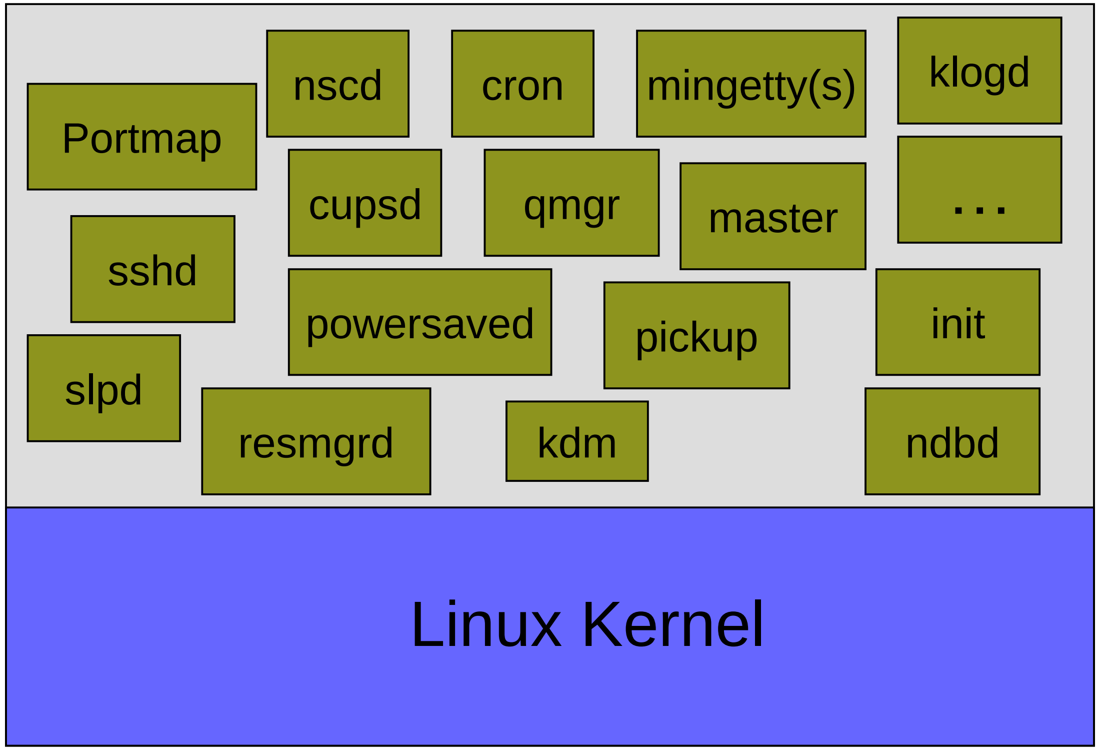
Licensed ISV SW

3rd party packaging

Cray added value to 3rd party

Trimming OS

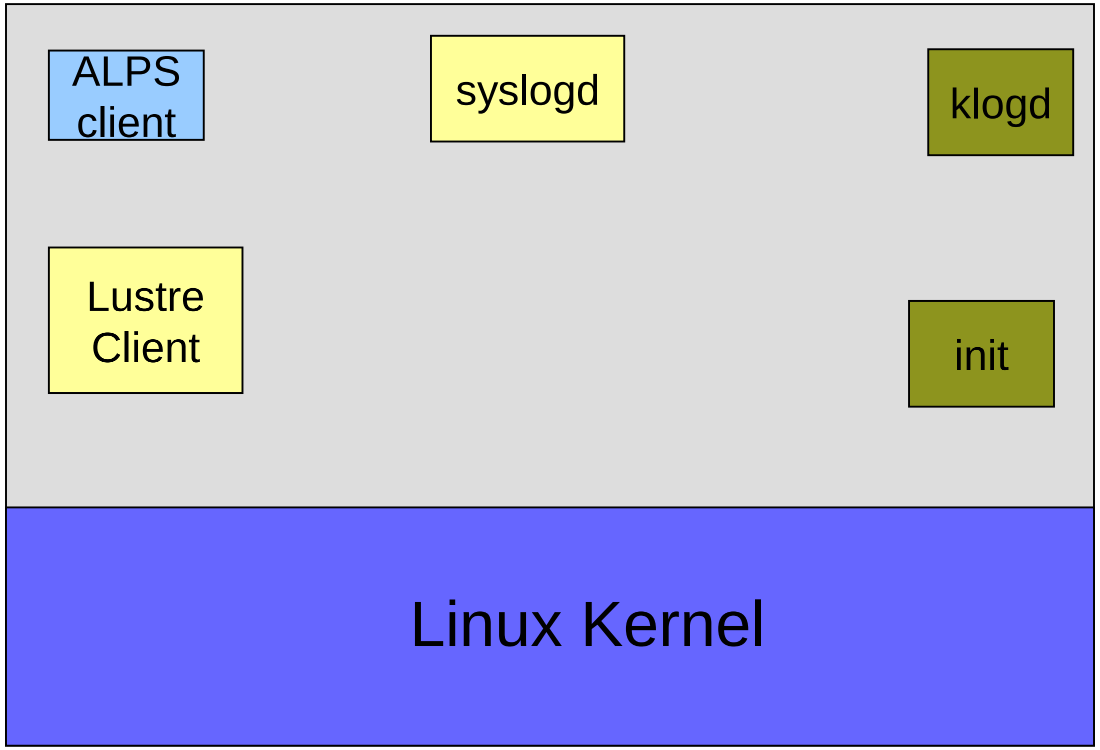
- Standard Linux Server



Portmap, nscd, cron, mingetty(s), klogd, cupsd, qmgr, master, …, sshd, powersaved, pickup, init, slpd, resmgrd, kdm, ndbd

Linux Kernel

- Linux on a Diet – *CLE*



ALPS client, syslogd, klogd, Lustre Client, init

Linux Kernel

Cray Programming Environment:

- Cray supports C, C++, Fortran, Python etc programing languages
- Cray supports GNU, Intel and other third party compilers
- Cray programming environment and cray compilers are default user environments.
- Modules application allows you to dynamically modify your user environment by using modulefiles

An introduction to modules

What are 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

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

Most important module commands

- **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/8.5.8
```

- **Metamodules bundles multiple modules**
- **Can create your own (meta)modules**

- **Module tool take care**
  - Environment variables
    - PATH, MANPATH, LD_LIBRARY_PATH, LM_LICENSE_FILE,....
  - Taking care of compiler and linker arguments of loaded products
    - Include paths, linker paths, …

More module commands

```
$> module list
```

- Prints actual loaded modules

```
$> module avail [-S str]
```

- Prints all module available containing the specified **str**ing

```
$> 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

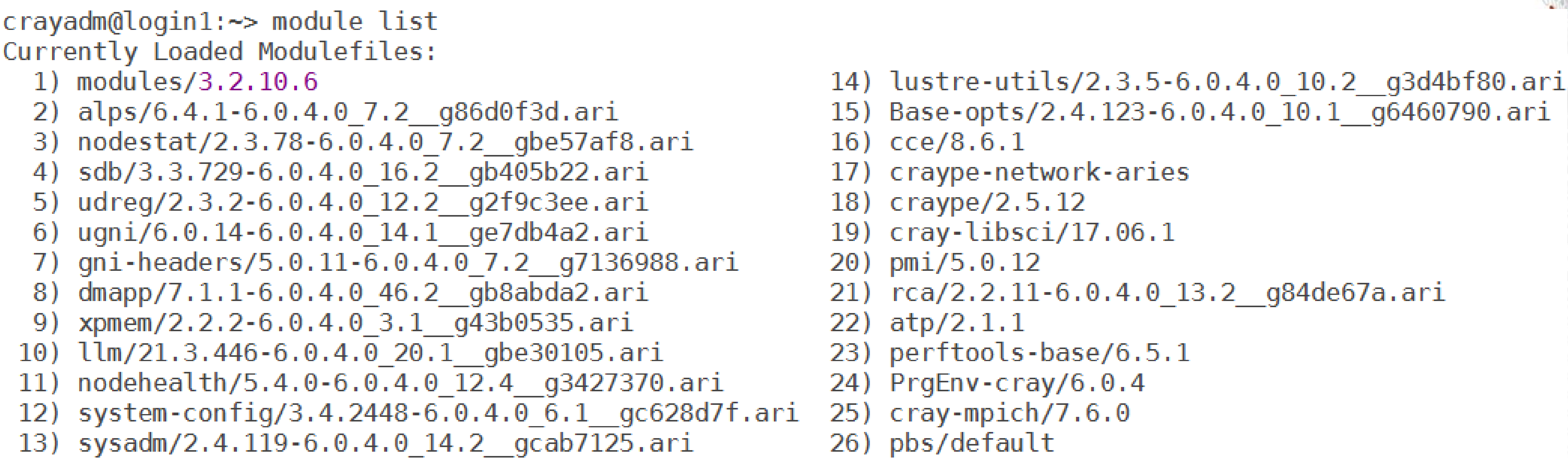
```
$> module show [mod]
```

- 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

Default module list at SahasraT



```
crayadm@login1:~> module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.6                                  14) lustre-utils/2.3.5-6.0.4.0_10.2__g3d4bf80.ari
  2) alps/6.4.1-6.0.4.0_7.2__g86d0f3d.ari              15) Base-opts/2.4.123-6.0.4.0_10.1__g6460790.ari
  3) nodestat/2.3.78-6.0.4.0_7.2__gbe57af8.ari         16) cce/8.6.1
  4) sdb/3.3.729-6.0.4.0_16.2__gb405b22.ari            17) craype-network-aries
  5) udreg/2.3.2-6.0.4.0_12.2__g2f9c3ee.ari            18) craype/2.5.12
  6) ugni/6.0.14-6.0.4.0_14.1__ge7db4a2.ari            19) cray-libsci/17.06.1
  7) gni-headers/5.0.11-6.0.4.0_7.2__g7136988.ari      20) pmi/5.0.12
  8) dmapp/7.1.1-6.0.4.0_46.2__gb8abda2.ari            21) rca/2.2.11-6.0.4.0_13.2__g84de67a.ari
  9) xpmem/2.2.2-6.0.4.0_3.1__g43b0535.ari             22) atp/2.1.1
 10) llm/21.3.446-6.0.4.0_20.1__gbe30105.ari           23) perftools-base/6.5.1
 11) nodehealth/5.4.0-6.0.4.0_12.4__g3427370.ari       24) PrgEnv-cray/6.0.4
 12) system-config/3.4.2448-6.0.4.0_6.1__gc628d7f.ari  25) cray-mpich/7.6.0
 13) sysadm/2.4.119-6.0.4.0_14.2__gcab7125.ari         26) pbs/default
```

“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@login1:~> 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/8.6.1
setenv           craype_already_loaded 1
module           swap craype/2.5.12
module           swap cray-mpich cray-mpich/7.6.0
module           load cray-libsci
module           load pmi
module           load rca
module           load atp
module           load perftools-base
setenv           CRAY_PRGENVCRAY loaded
-------------------------------------------------
```

Compile applications on the Cray XC

Things to remember before compiling

- Check loaded programming modules
- Check compiler and their versions
- If not, load relevant modules

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 current used module environment. Use the `–craype-verbose` flag to see the default options.

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

```
$> ftn -c <any_other_flags> 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++
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.

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/8.3.10`

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 …'
- **If you are compiling for the login nodes**
  - You should use the original direct compiler commands, e.g. `ifort`, `pgcc`, `crayftn`, `gcc`, … PATH will change with modules. All libraries will have to be linked in manually.
  - Conversely, you can use the compiler wrappers {`cc`,`CC`,`ftn`} and use the `-target-cpu=` option among {abudhabi, haswell, interlagos, istanbul, ivybridge, mc12, mc8, sandybridge, shanghai, x86_64. The x86_64 is the most compatible but also less specific.

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

More module commands

`$> module list`	• Prints actual loaded modules
`$> module avail [-S str]`	• Prints all module available containing the specified **str**ing
`$> module (un)load [mod_name/version]`	• Adds or remove a module to the actual loaded list • If no version specified, loading the default version
`$> 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

“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@elogin04:~> 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/8.6.3
setenv           craype_already_loaded 1
module           swap craype/2.5.13
module           swap cray-mpich cray-mpich/7.6.3
module           load cray-libsci
module           load udreg
module           load ugni
```

What module does ?

```
crayadm@login1:~> module show cce
-------------------------------------------------------------------
/opt/cray/pe/modulefiles/cce/8.6.1:

conflict         cce
setenv           GCC_X86_64 /opt/gcc/6.1.0/snos
setenv           CRAY_BINUTILS_ROOT_X86_64 /opt/cray/pe/cce/8.6.1/binutils/x86_64/x86_64-pc-linux-gnu/../
setenv           CRAY_BINUTILS_BIN_X86_64 /opt/cray/pe/cce/8.6.1/binutils/x86_64/x86_64-pc-linux-gnu/bin
setenv           LINKER_X86_64 /opt/cray/pe/cce/8.6.1/binutils/x86_64/x86_64-pc-linux-gnu/bin/ld
setenv           ASSEMBLER_X86_64 /opt/cray/pe/cce/8.6.1/binutils/x86_64/x86_64-pc-linux-gnu/bin/as
setenv           FTN_X86_64 /opt/cray/pe/cce/8.6.1/cce/x86_64
setenv           CC_X86_64 /opt/cray/pe/cce/8.6.1/cce/x86_64
setenv           CRAY_CXX_IPA_LIBS_X86_64 /opt/cray/pe/cce/8.6.1/cce/x86_64/lib/libcray-c++-rts.a
setenv           CRAYLIBS_X86_64 /opt/cray/pe/cce/8.6.1/cce/x86_64/lib
prepend-path     INCLUDE_PATH_X86_64 /opt/cray/pe/cce/8.6.1/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/8.6.1/binutils/cross/x86_64-aarch64/aarch64-unknowun-linux-gnu/../
setenv           CRAY_BINUTILS_BIN_AARCH64 /opt/cray/pe/cce/8.6.1/binutils/cross/x86_64-aarch64/aarch64-unknowun-linux-gnu/bin
setenv           LINKER_AARCH64 /opt/cray/pe/cce/8.6.1/binutils/cross/x86_64-aarch64/aarch64-unknowun-linux-gnu/bin/ld
setenv           ASSEMBLER_AARCH64 /opt/cray/pe/cce/8.6.1/binutils/cross/x86_64-aarch64/aarch64-unknowun-linux-gnu/bin/as
setenv           CRAY_CXX_IPA_LIBS_AARCH64 /opt/cray/pe/cce/8.6.1/cce/aarch64/lib/libcray-c++-rts.a
setenv           CRAYLIBS_AARCH64 /opt/cray/pe/cce/8.6.1/cce/aarch64/lib
prepend-path     INCLUDE_PATH_AARCH64 /opt/cray/pe/cce/8.6.1/cce/aarch64/include/craylibs
setenv           CRAYLMD_LICENSE_FILE /opt/cray/pe/cce/cce.lic
setenv           CRAY_BINUTILS_ROOT /opt/cray/pe/cce/8.6.1/binutils/x86_64/x86_64-pc-linux-gnu/../
setenv           CRAY_BINUTILS_VERSION /opt/cray/pe/cce/8.6.1
setenv           CRAY_BINUTILS_BIN /opt/cray/pe/cce/8.6.1/binutils/x86_64/x86_64-pc-linux-gnu/bin
setenv           CRAY_CCE_SHARE /opt/cray/pe/cce/8.6.1/cce/x86_64/share
setenv           CRAY_CXX_IPA_LIBS /opt/cray/pe/cce/8.6.1/cce/x86_64/lib/libcray-c++-rts.a
setenv           CRAY_FTN_VERSION 8.6.1
setenv           CRAY_CC_VERSION 8.6.1
setenv           PE_LEVEL 8.6
prepend-path     FORTRAN_SYSTEM_MODULE_NAMES ftn_lib_definitions
prepend-path     MANPATH /opt/cray/pe/cce/8.6.1/man
prepend-path     NLSPATH /opt/cray/pe/cce/8.6.1/cce/x86_64/share/nls/En/%N.cat
prepend-path     CRAY_LD_LIBRARY_PATH /opt/cray/pe/cce/8.6.1/cce/x86_64/lib
prepend-path     PATH /opt/cray/pe/cce/8.6.1/binutils/x86_64/x86_64-pc-linux-gnu/bin:/opt/cray/pe/cce/8.6.1/binutils/cross/x86_64-aarch64/aarch64-unknowun-linux-gnu/.
./bin:/opt/cray/pe/cce/8.6.1/utils/x86_64/bin
append-path      MANPATH /usr/share/man
-------------------------------------------------------------------
```

Targeting different node types

- **Compiling for the CPU nodes**
  - module load craype-haswell
    (enables the haswell specific instructions. Default is x86_64)

```
% module load PrgEnv-Cray or PrgEnv-gnu or PrgEnv-intel
% module load craype-haswell
% module load <application related modules>
```

Then compile application

Targeting different node types

- **Compiling for the CPU nodes**
  - module load craype-haswell
    (enables the haswell specific instructions. Default is x86_64)

```
% module load PrgEnv-Cray or PrgEnv-gnu or PrgEnv-intel
% module load craype-haswell
% module load <application related modules>
```

Then compile application

Targeting different node types

- **Compiling for KNL nodes**

While compiling application for KNL,

- Load cray-mic-knl
  % module load craype-mic-knl

- Based on PrgEnv, use below flags and compile application

  “-xMIC-AVX512” for Intel Compilers
  “-hcpu=mic-knl” for Cray compilers
  “-march=knl” for GNU compilers

Targeting different node types

- **Compiling for the GPU nodes**
  - module load craype-accel-nvidia35 or craype-accel-nvidia60

    Here, craype-accel-nvidia60 is for Pascal
    craype-accel-nvidia35 for Kepler
  - “module display craype-accel-nvidia35” tells you that this module also loads cudatoolkit and cray-libsci-acc

Eg :

```
module        load PrgEnv-gnu/6.0.4
module        load gcc/4.9.3 or gcc/5.3.0
module        load craype-ivybridge
module        load craype-accel-nvidia35 ( we have Kepler 40)
```

Summary

- **Four compiler environments available on the XC:**
    - Cray (PrgEnv-cray is the default)
    - Intel (PrgEnv-intel)
    - GNU (PrgEnv-gnu)
    - PGI (PrgEnv-pgi)
  - All of them accessed through the wrappers ftn, cc and CC – just do module swap to change a compiler or a version.
- **There is no universally fastest compiler**
  - Performance strongly depends on the application (even input)
  - We try however to excel with the Cray Compiler Environment
  - If you see a case where some other compiler yields better performance, let us know!
- **Compiler flags do matter**
  - be ready to spend some effort for finding the best ones for your application.
  - More information is given at the end of this presentation.

Run applications on XC

How to run application on a XC 40 ?

- **Two ways to run applications :**
  - Interactive mode
    - Log in to service node
    - Less response time
    - Prompt the user for input as data or commands
    - Best suited for Short tasks, those which require frequent user interaction
  - Batch mode
    - Submitted to a job scheduler
    - Best for longer running processes
    - Avoids idling the computing resources

How to run application on a XC 40?

Most Cray XCs are batch systems

- Users submit batch job scripts to a scheduler from a login node (e.g. PBS, MOAB, SLURM) for execution at some point in the future.
  Each job requires resources and a prediction how long it will run.
- The scheduler (running on an external server) chooses which jobs to run and allocates appropriate resources
- The batch system will then execute the user’s job script on an a different node as the login node.
- The scheduler monitors the job and kills any that overrun their runtime prediction.
- The batch script contains one or more parallel job runs executed via aprun

- The main Cray system uses the workload manager and the Application Level Placement Scheduler (ALPS)
- In your daily work you will mainly encounter the following commands:

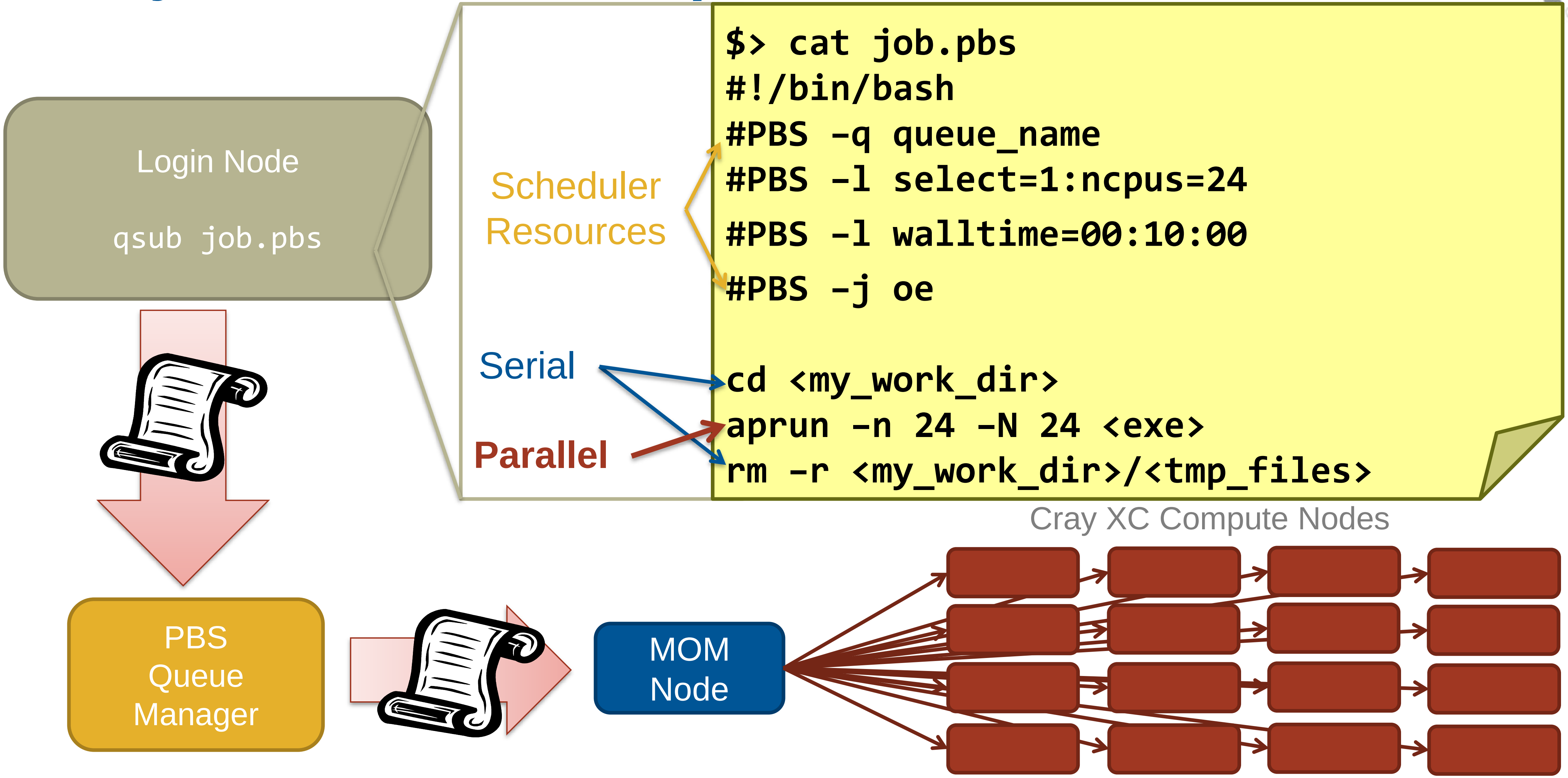
```
qsub  – Submit a batch script to PBS.
aprun – Run parallel jobs within the script.
qdel  – Signal jobs under the control of PBS
qstat – information about running jobs
```

- Plenty of 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.
- Nodes are used exclusively.

Running a job on HPC system :

- Prepare job submission script
- Script file defines the commands and cluster resources used for the job
- Log in to “External Log-in node”
- The **qsub** command is used to submit a job to the PBS queue
- PBS queue used to allocate resources.

Lifecycle of a batch script



```
$> cat job.pbs
#!/bin/bash
#PBS -q queue_name
#PBS -l select=1:ncpus=24
#PBS -l walltime=00:10:00
#PBS -j oe

cd <my_work_dir>
aprun -n 24 -N 24 <exe>
rm -r <my_work_dir>/<tmp_files>
```

Requesting Resources

- Job requirements as #PBS comments in the headers of the batch script
- Common options:

Option	Description
`-l nodes=<nnodes>:ppn=24`	Requests X full nodes (only full nodes are available on HazelHen)
`-l walltime <HH:MM:SS>`	Maximum wall time job will occupy
`-N <job_name>`	Name of the job
`-A <code>`	Account to run job under (for controlling budgets)
`-j oe`	collect both `stderr` and `stdout` to a single file specified by the `–o` option or the default file for `stdout`.

`–o <my_output_file_name>`
`–e <my_error_file_name>`
	Redirects stdout and stderr to two separate files. If not specified, the script output will be written to files of the form <script_name>.e<JOBID> and <script_name>.o<JOBID>.

`-q <queue>`	Submit job to a specific queues

These can be overridden or supplemented by adding arguments to the `qsub` command line, e.g.

```
$> qsub –l select=20:ncpus=24 run.pbs
```

Running an application using ALPS + aprun

- `aprun` is the ALPS application launcher
  - Runs applications on the XC compute nodes.
    `aprun` launches groups of Processing Elements (PEs) on the compute nodes
    (PE == (MPI RANK || Coarray Image || UPC Thread || ..))
  - Cannot get more resources for aprun than requested via WLM.
  - The most important parameters (manpage for more examples)

Option	Description
-n	Total Number of PEs used by the application
-N	Number of PEs per compute node
-d	“stride” between 2 PEs on a node, usually used for: Number of threads per PE
-S	Pes per numa node (can have effects for memory bandwidth)
-j	-j 2 enables hyperthreading

  - Applications started without aprun, are executed on mom nodes and can affect other users jobs

Cray XC Basic MPI-Jobs Examples

Single node, Single task
Run a job on one task on one node with full memory.

```
…
#PBS -l select=1:ncpus=24
…
aprun -n 1 ./<exe>
```

Single node, Multiple Ranks
Run a pure MPI job with 24 Ranks or less on one node.

```
…
#PBS -l select=1:ncpus=24
…
aprun -n 24 ./<exe>
```

Multiple nodes, Multiple Ranks
Run a pure MPI job on 4 nodes with 24 MPI ranks or less on each node.

```
…
#PBS -l select=1:ncpus=24


…
aprun -n 96 -N 24 ./<exe>
```

Hyperthreads on the XC with ALPS

- **Intel Hyper-Threading is a method of improving the throughput of a CPU by allowing two independent program threads to share the execution resources of one CPU**
  - When one thread stalls the processor can execute read instructions from a second thread instead of sitting idle
  - Because only the thread context state and a few other resources are replicated (unlike replicating entire processor cores), the throughput improvement depends on whether the shared execution resources are a bottleneck
  - Typically much less than 2x with two hyperthreads
  - With `aprun`, hyper-threading is controlled with `-j`
    - `-j 1` = no hyper-threading (default)
      (a node is treated to contain 24 cores)
    - `-j 2` = hyper-threading enabled
      (a node is treated to contain 48 cores)
  - Try it, if it does not help, turn it off.

```
…
#PBS –l select=1:ncpus=24
…
aprun –n 48 –j2 ./<exe>
```

XC Hybrid MPI/OpenMP Jobs (Example)

Pure OpenMP Job
Using 4 threads on one a single node

```
…
#PBS -l select=1:ncpus=24
…
export OMP_NUM_THREADS=4
echo "OMP_NUM_THREADS: $OMP_NUM_THREADS"
aprun -n 1 -d $OMP_NUM_THREADS ./<omp_exe>
```

Hybrid MPI/OpenMP job on 3 nodes with 12 MPI ranks per node, 4 threads for each rank, using Hyperthreads.

```
…
#PBS -l select=3:ncpus=24
…
export OMP_NUM_THREADS=4
echo "OMP_NUM_THREADS: $OMP_NUM_THREADS"
aprun -n 36 -N 12 -d $OMP_NUM_THREADS -j 2 ./<hybrid_exe>
```

Monitoring your Job

- After submitting your job, you can monitor its status

Command	Description
`$> qsub <batch_script> <JOBID>`	Start your job with from the shell with qsub. The <JOBID> is printed.
`$> qstat -u $USER`	Prints status of all your jobs. Always check that the reported resources are what you expect.
`$> showq –u $USER`	information of active, eligible, blocked and completed jobs
`$> checkjob <JOBID>`	Detailed job state information and diagnostic output
`$> qdel <JOBID>`	Only if you think that your job is not running properly after inspecting your output files, you can cancel it with qdel.

Interactive Sessions

request an **interactive session**.

•use qsub option –I

•typically used for small jobs which have to be run frequently for testing or for debugging sessions with STAT, ATP, DDT etc. and usually used with small amount of nodes.

```
eslogin08$> qsub -I -l nodes=2,walltime=00:19:00
qsub: waiting for job 123456.XXX-batch.YYY.com to start
...
qsub: job 123456.XXX-batch.YYY.de ready
Welcome to XXX (Cray XC40) at XXX.
Directory: /home/userxyz
Fri Feb 07 08:15:00 CEST 2015
mom15$> aprun –n 24 –N 12 … <my_application>
```

Once the Job is executed by PBS, the user receives a shell prompt where commands like `aprun` can be executed directly. An entire batch script could be executed with `source <bath_scipt>`.

(!) interactive sessions are executed on MOM nodes. Every compute intense calculation has to be executed with `aprun`.

Environment variables

- Job specific environmental variables are available

Environment Variable	Description
PBS_O_WORKDIR	Directory where qsub has been executed
PBS_JOBID	Job ID
PBS_JOBNAME	Job name as specified by the user
PBS_NODEFILE	List of allocated nodes.

- E.g. using the maximum allocated resources

```
#!/bin/bash
#PBS -N xthi
#PBS -l nodes=3:ppn=24
#PBS -l walltime=00:05:00
...
NS=$( qstat -f ${PBS_JOBID} | awk '/Resource_List.nodect/{ print $3 }' )
NRANK=$[ ${NS} * 24 ]

aprun -n ${NRANK} -N 24 -d ${OMP_NUM_THREADS} -j1 ./a.out
```

Queues on SERC System

```
crayadm@login1:~> qstat -q

server: sdb

Queue            Memory CPU Time Walltime Node   Run   Que   Lm  State
---------------- ------ -------- -------- ---- ----- ----- ----  -----
large              --      --    24:00:00  --      0     0   --   E R
medium             --      --    24:00:00  --      8    17   --   E R
small72            --      --    72:00:00  --     15    16   --   E R
small              --      --    24:00:00  --     20    38   --   E R
gpu                --      --    24:00:00    4    30    20   --   E R
mgpu               --      --    24:00:00   24     1     3   --   E R
knl                --      --    24:00:00  --      2     0   --   E R
idqueue            --      --    02:00:00  --      9    22   --   E R
                                               ----- -----
                                                  86   136
```

Queues on SERC System

Batch Strategies and Queues :

Queue name: Batch
Queue type: Route
Max_queued_by_each_user: 2
Route destinations: idqueue, small, small72, medium, large, gpu, knl

==============================

Queue Name: idqueue
Queue Type: Execution
Job type: CPU MPI based/ openmp based
Max_job_queued_per_user: 2
Core ranges: 24 – 256 ~ 10 nodes
Max_walltime: 2hrs
Max_user_job_run: 1
Total_job_runs: 32

Queues on SERC System

Queue Name: small
Queue Type: Execution
Max_job_queued_per_user: 3
Job type: CPU MPI based/openmp based
Core ranges: 24 – 1032
Max_walltime: 24hrs
Max_user_job_run: 2
Total_job_runs: 20

==================================

Queue Name: small72
Queue Type: Execution
Max_job_queued_per_user: 1
Job type: CPU MPI based/openmp based
Core ranges: 24 – 1032
Max_walltime: 72hrs
Max_user_job_run: 1
Total_job_runs: 15

Queue Name: medium
Queue Type: Execution
Max_job_queued_per_user: 1
Job type: CPU MPI based/openmp based
Core ranges: 1033 - 8208
Max_walltime: 72hrs
Max_user_job_run: 1
Total_job_runs: 10

==================================

Queue Name: large
Queue Type: Execution
Max_job_queued_per_user: 1
Job type: CPU MPI based/openmp based
Core ranges: 8209 - 22800
Max_walltime: 24hrs
Max_user_job_run: 1
Total_job_runs: 4

Queues on SERC System

Queue Name: gpu
Queue Type: Execution
Job Type: Cuda based code/Opencl code/ GPU applications
Max_job_queued_per_user: 5
Core ranges: 1 – 48
Min no. of accelerators (Nvidia): 1
Max no. of accelerators (Nvidia): 4
Max_walltime: 24hrs
Max_user_job_run: 3
Total_job_runs: 30

=====================================

Queue Name: knl
Queue Type: Execution
Job Type: intel-xeon phi coprocessor job
Max_job_queued_per_user: 3
Core ranges: 1 - 480
Max_walltime: 24hrs
Max_user_job_run: 2

Limitations of SahasraT:

- Resources are shared between users
- User will get 1.5GB of /home area
- 10 TB of high speed storage (Lustre Storage)

  Location : /mnt/lustre/<user>
- Third party applications’ licenses are to be provided by users

Questions?

Thank You

Email : iisc_support@cray.com