A Quick Start Guide to NCCS JAGUAR System

Go to Menu

Outline

Jaguar System Overview: General Outline

Jaguar is a Cray XT system consisting of XT4 and XT5 partitions.

Jaguar	**XT4**	**XT5**
CPU Type	2.1 GHz Quad-core AMD Opteron (Budapest)	2.6 GHz Hex-core AMD Opteron (Istanbul)
Interconnect	Cray SeaStar2 Router	Cray SeaStar2+ Router
Switching Capacity (Router's Peak Bandwidth)	45.6GB/s 6 switch ports per Cray SeaStar chip, 7.6 GB/s each	57.6GB/s 6 switch ports per Cray SeaStar2+ chip, 9.6 GB/s each
Memory type	DDR2-800 (some nodes use DDR2-667 memory)	DDR2-800
Memory Bandwidth	10.6 to 12.8 GB/sec per AMD Opteron	21.2 GB/sec to 25.6 GB/sec per compute node
Floor Space	1400 feet^2	4400 feet^2
Cooling Technology	Air	Liquid

Go to Menu

Jaguar System Overview: Summary of Resources

Jaguar is a Cray XT system consisting of XT4 and XT5 partitions

Jaguar	**XT4**	**XT5**	**Total**
Nodes per blade	4		
CPUs per node[1]	1	2	
Cores per node	4	12	
Compute nodes[2]	7,832	18,688	
AMD Opteron cores	31,328	224,256	255,584
Memory per CPU	8 GB/CPU		
System Memory	~61.2 TB	~292 TB	~353.2 TB
Disk Bandwidth	~44 GB/s	~240 GB/s	~284 GB/s
Disk Space	~750 TB	~10,000 TB	~10,750 TB
Interconnect Bandwidth	~157 TB/s	~374 TB/s	~532 TB/s
Floor Space	1400 feet^2	4400 feet^2	5800 feet^2
Ideal Performance per core[3] **(4 FLOPs/cycle times 2.1*10^9 cycles/sec)**	8.4 GFLOPS	10.4 GFLOPS	
Overall Ideal Performance	~263.16 TFLOPS	~2.33 PFLOPS	~2.60 PFLOPS

[1] In the context of Jaguar CPU is also called a socket.
[2] Note that in addition to compute nodes Jaguar also has input/output (I/O) and login service nodes.
[3] FLOPs = **FL**oating point **OP**eration**s**; FLOPS = **FL**oating point **O**perations **P**er **S**econd

Go to Menu

Jaguar System Overview: System Software

- Operating system is Cray Linux Environment (CLE) 2.1:
  - Compute Nodes – Compute Node Linux (CNL)
  - Login/Service nodes – SUSE Linux
- Compilers
  - C/C++, Fortran
- MPI implementation
  - Cray MPI based on MPICH
- High Performance Storage System (HPSS) software

Logging into Jaguar: Connection Requirements

- The only supported remote client on NCCS systems is a secure shell (SSH) client.
- The only supported authentication method is one-time passwords (OTP).

- UNIX-based operating systems generally have an SSH client built in.
- Windows users may obtain free clients online, such as PuTTY.

Any SSH client:

- must support the SSH-2 protocol (supported by all modern SSH clients).
- must allow keyboard-interactive authentication to access NCCS systems. For UNIX-based SSH clients, the following line should be in either the default ssh_config file or your $HOME/.ssh/config file:

```
    PreferredAuthentications keyboard-interactive,password
```

  The line may also contain other authentication methods, so long as keyboard-interactive is included.

Go to Menu

Logging into Jaguar: Connection Procedure

To connect to Jaguar from a UNIX-based system type the following in your terminal:

```
ssh userid@jaguar.ccs.ornl.gov
```

$\leftarrow$ Cray XT4

```
ssh userid@jaguarpf.ccs.ornl.gov
```

$\leftarrow$ Cray XT5

Enter PASSCODE: $\underbrace{\text{PIN}}_{\text{4-6 digits}} + \underbrace{\text{6 digits}}_{\text{Changes every 30 seconds}}$ from RSA® SecurID

NCCS RSA Key Fingerprints:

```
jaguar   0d:c9:db:37:55:da:41:26:55:4a:80:bb:71:55:dd:01
jaguarpf 80:58:21:03:96:47:1a:15:2c:25:d3:ca:e6:04:e8:a7
```

Go to Menu

Login Nodes

- When you login to Jaguar you will be placed on a “login node”
- Login nodes are used for basic tasks such as file editing, code compilation, data backup, and job submission
- These nodes provide a full SUSE Linux environment, complete with compilers, tools, and libraries
- The login nodes should not be used to run production jobs. Production work should be performed on the systems compute resources.
- Serial jobs (post-processing, *etc*) may be run on the compute nodes as long as they are statically linked (will be discussed later)

Go to Menu

Compute (Batch) Nodes

- All MPI/OpenMP user applications execute on batch or compute nodes
- Batch nodes provide **limited** Linux environment – Compute Node Linux (CNL)
- Compute nodes can see only the Lustre scratch directories
- Access to compute resources is managed by the PBS/TORQUE – batch system manager
- Job scheduling is handled by Moab, which interacts with PBS/TORQUE and the XT system software.

Go to Menu

File Systems: Basics

- The Network File Service (NFS) server contains user's home directories, project directories, and software directories.
- Compute nodes can only see the Lustre work space
  - The NFS-mounted home, project, and software directories are not accessible to the compute nodes.
- Shared Lustre area (SPIDER) is now available on compute nodes and is the only scratch area for the XT5.
- Executables must be executed from within the Lustre work space:
  - `/tmp/work/$USER` (XT4 and XT5)
  - `/lustre/scr144/$USER` (XT4 only)
- Batch jobs can be submitted from the home or work space. If submitted from a user's home area, a batch script should cd into the Lustre work space directory (`cd $PBS_O_WORKDIR`) prior to running the executable through aprun.
- All input must reside in the Lustre work space
- All output must also be sent to the Lustre work space

Go to Menu

File Systems: User's Directories

Each user is provided the following space resources:

- Home directory - NFS Filesystem
  `/ccs/home/$USER`

- Work directory/Scratch space - Lustre Filesystem
  `/tmp/work/$USER`

- Project directory - NFS Filesystem
  `/ccs/proj/projectid`

- HPSS storage

File Systems: Quota policy

Area	Path	Quota	Swept?	Backups?	Purge Policy
Home Directory	/ccs/home/$USER	5 GB	No	Yes	1 month post-user
Project Directory	/ccs/proj/$PROJ	50 GB	No	Yes	1 month post-project
Work Directory	/tmp/work/$USER	None	14 days	No	1 month post-user
HPSS Home	/home/$USER	2 TB 200 Files	-	-	3 months post-user
HPSS Project	/proj/$PROJ	45 TB 4500 Files	-	-	3 months post-project

Software Environment: Modules

- Software is loaded, unloaded or swapped using modules.
- Use of modules allows software, libraries, paths, etc. to be cleanly entered into and removed from your programming environment.
- Conflicts are detected and module loads that would cause conflicts are not allowed.

Software Environment: `module` command

Loading Commands

- **`module [load||unload] my_module`**
  - Loads/Unloads module *`my_module`*
  - e.g., `module load subversion`

- **`module swap module#1 module#2`**
  - Replaces *`module#1`* with *`module#2`*
  - e.g., `module swap PrgEnv-pgi PrgEnv-gnu`

Informational Commands

- **`module help my_module`**
  - Lists available commands and usage
- **`module show my_module`**
  - Displays the actions upon loading `my_module`
- **`module list`**
  - Lists all loaded modules
- **`module avail [name]`**
  - Lists all modules [beginning with *`name`*]
  - e.g., `module avail gcc`

Software Environment: Default Module List

```
username@jaguarpf-login1:/> module list
Currently Loaded Modulefiles:
  1) modules/3.1.6
  2) DefApps
  3) torque/2.4.1b1-snap.200905191614
  4) moab/5.3.6
  5) /opt/cray/xt-asyncpe/default/modulefiles/xtpe-istanbul
  6) cray/MySQL/5.0.64-1.0000.2342.16.1
  7) xtpe-target-cnl
  8) xt-service/2.2.41A
  9) xt-os/2.2.41A
 10) xt-boot/2.2.41A
 11) xt-lustre-ss/2.2.41_1.6.5
 12) cray/job/1.5.5-0.1_2.0202.18632.46.1
 13) cray/csa/3.0.0-1_2.0202.18623.63.1
 14) cray/account/1.0.0-2.0202.18612.42.3
 15) cray/projdb/1.0.0-1.0202.18638.45.1
 16) Base-opts/2.2.41A
 17) pgi/9.0.4
 18) xt-libsci/10.4.0
 19) xt-mpt/3.5.0
 20) xt-pe/2.2.41A
 21) xt-asyncpe/3.2
 22) PrgEnv-pgi/2.2.41A
```

Go to Menu

Compiling: System Compilers

The following compilers should be used to build codes on Jaguar. Use these compilers.

Language	Compiler
C	**`cc`**
C++	**`CC`**
Fortran 77, 90 and 95	**`ftn`**

Note that `cc`, `CC` and `ftn` are actually the Cray XT Series wrappers for invoking the PGI, GNU, Intel, Cray, or Pathscale compilers (discussed later…)

Go to Menu

Compiling: Parallel Compiling on Jaguar

- Jaguar has two kinds of nodes:
  - Compute Nodes running the CNL OS
  - Service and login nodes running Linux
- To build a code for the compute nodes, you should use the Cray wrappers `cc`, `CC`, and `ftn`. The wrappers will call the appropriate compiler which will use the appropriate header files and link against the appropriate libraries. Use of wrappers is crucial for building the parallel codes on Cray.
- <u>We highly recommend that the `cc`, `CC`, and `ftn` wrappers be used when building for the compute nodes. Both parallel and serial codes.</u>
- To build a code for the Linux service nodes, you should call the compilers directly.
- <u>We strongly suggest that you don't call the compilers directly if you are building code to run on the compute nodes.</u>
- No long serial jobs should be run on service nodes, use compute nodes instead.

Go to Menu

Compiling: Default Compilers

- Default compiler is PGI. The list of all packages is obtained by
  - `module avail PrgEnv`
- To use the Cray wrappers with other compilers the programming environment modules need to be swapped, i.e.
  - `module swap PrgEnv-pgi PrgEnv-gnu`
  - `module swap PrgEnv-pgi PrgEnv-cray`
- To just use the GNU/Cray compilers directly load the GNU/Cray module you want:
  - `module load PrgEnv-gnu/2.1.50HD`
  - `module load PrgEnv-cray/1.0.1`
- It is possible to use the GNU compiler versions directly without loading the Cray Programming Environments, but note that the Cray wrappers will probably not work as expected if you do that.

Go to Menu

Compiling: Useful Compiler Flags (PGI)

General:

Flag	Comments
`-mp=nonuma`	Compile multithreaded code using OpenMP directives

Debugging:

Flag	Comments
`-g`	For debugging symbols; put first
`-Ktrap=fp`	Trap floating point exceptions
`-Mchkptr`	Checks for unintended dereferencing of null pointers

Optimization:

Flag	Comments
`-Minfo`	Provides info on compiler performed optimizations
`-Mneginfo`	Instructs the compiler to produce information on why certain optimizations are not performed.
`-fast`	Equivalent to **`-Mvect=sse -Mscalarsse -Mcache_align -Mflushz`**
`-fastsse`	Same as **`-fast`**
`-Mcache_align`	Makes certain that arrays are on cache line boundaries
`-Munroll=c:`*`n`*	Unrolls loops ***n*** times (e.g., ***n***=4)
`-Mipa=fast,inline`	Enables interprocedural analysis (IPA) and inlining, benefits for C++ and Fortran
`-Mconcur`	Automatic parallelization

Go to Menu

Compiling: Useful Compiler Flags (GNU)

General:

Flag	Comment
`-fopenmp`	Compile multithreaded code using OpenMP directives

Debugging:

Flag	Comment
`-g`	For debugging symbols; put first
`-finstrument-functions`	For using CrayPat
`-fbounds-check`	Enable generation of runtime checks for array subscripts

Optimization:

Flag	Comments
`-O2 -ffast -math -fomit -frame -pointer -mfpmath=sse`	Recommended first compile/run
`-mfpmath=sse`	Use scalar floating point instructions present in SSE instruction set
`-finline -functions`	Inline simple functions (turned on automatically by `-O3`)
`-funroll -loops --param max -unroll -times=`***n***	Unrolls loops ***n*** times (e.g., ***n***=4)

`pathopt2` utility can help identify compiler options that give best optimization

Compiling: Useful Compiler Flags (Pathscale)

General:

Flag	Comments
`-mp`	Compile multithreaded code using OpenMP directives (NOTE: limited support for C++ at this time)

Debugging:

Flag	Comments
`-g`	For debugging symbols; put first
`-LNO:simd_verbose=on`	Get diagnostics
`-trapuv`	Initialize variables to NaN – useful for finding uninitialized variables
`-zerouv`	Initialize variables to 0

Optimization:

Flag	Comments
`-O3 -OPT:Ofast`	Recommended first compile/run
`-OPT:Ofast`	Maximizes performance; generally safe but may impact floating point correctness. Equivalent to `–OPT:ro=2:Olimit=0:div_split=ON:alias=typed`
`-Ofast`	Equivalent to `-O3 -ipa -OPT:Ofast -fno-math-errno`
`-ipa`	Enables interprocedural analysis (IPA) and inlining
`-apo`	Enables autoparallelization

Go to Menu

Compiling: Useful Compiler Flags (Intel)

General:

Flag	Comments
`-openmp`	Compile multithreaded code using OpenMP directives

Debugging:

Flag	Comments
`-g`	Generate full debugging information in the object file.
`-debug[keyword]`	Enables or disables generation of debugging information.
`-Wuninitialized`	Determines whether a warning is issued if a variable is used before being initialized.

Optimization:

Flag	Comments
`-fast`	Maximizes speed across the entire program.
`-O[n]`	Specifies the code optimization for applications.
`-finline`	Tells the compiler to inline functions declared with __inline and perform C++ inlining.
`-ipo[n]`	Enables interprocedural optimization between files.

Go to Menu

Compiling: Useful Compiler Flags (Cray)

General:

Flag	Comments
`-h omp`	Compile multithreaded code using OpenMP directives (turned on, by default)

Debugging:

Flag	Comments
`-g`	Generate full debugging information in the object file. (Equivalent ot **`-Gn`**)
`-G level`	Enables the generation of debugging information used by symbolic debuggers such as TotalView. These options allow debugging with breakpoints.

Optimization:

Flag	Comments
`-fast`	Maximizes speed across the entire program.
`-O level`	Specifies the code optimization for applications.
`-h ipa[n]`	Allows the compiler to automatically decide which procedures to consider for inlining.
`-h unroll[n]`	Globally controls loop unrolling and changes the assertiveness of the unroll pragma.

Running Jobs: Introduction

- When you log into Jaguar, you are placed on one of the login nodes.
- Login nodes should be used for basic tasks such as file editing, code compilation, data backup, and job submission.
- The login nodes should not be used to run production jobs. Production work should be performed on the system's compute resources.
- On Jaguar, access to compute resources is managed by the PBS/TORQUE. Job scheduling and queue management is handled by Moab which interacts with PBS/TORQUE and the XT system software.
- Users either submit the job scripts for batch jobs, or submit a request for interactive job.
- The following pages provide information for getting started with the batch facilities of PBS/TORQUE with Moab as well as basic job execution.

Go to Menu

Running Jobs: Batch Scripts

- Batch scripts can be used to run a set of commands on a systems compute partition.
- The batch script is a shell script containing PBS flags and commands to be interpreted by a shell.
- Batch scripts are submitted to the batch manager, PBS, where they are parsed. Based on the parsed data, PBS places the script in the queue as a job.
- Once the job makes its way through the queue, the script will be executed on the head node of the allocated resources.

Go to Menu

Running Jobs: Example Batch Script

```
1: #!/bin/bash
2: #PBS -A XXXYYY
3: #PBS -N test
4: #PBS -j oe
5: #PBS -l walltime=1:00:00,size=192
6:
7: cd $PBS_O_WORKDIR
8: date
9: aprun -n 192 ./a.out
```

NOTE: Since users cannot share nodes, <u>size</u> requests must be
- ➢ a multiple of 4 on the XT4 or
- ➢ a multiple of 12 on the XT5.

This batch script can be broken down into the following sections:

- Shell interpreter
  - Line 1
  - Can be used to specify an interpreting shell.
- PBS commands
  - The PBS options will be read and used by PBS upon submission.
  - Lines 2–5
    - 2: The job will be charged to the XXXYYY project.
    - 3: The job will be named “test.”
    - 4: The jobs standard output and error will be combined.
    - 5: The job will request 192 cores for 1 hour.
  - Please see the PBS Options page for more options.
- Shell commands
  - Once the requested resources have been allocated, the shell commands will be executed on the allocated nodes head node.
  - Lines 6–9
    - 6: This line is left blank, so it will be ignored.
    - 7: This command will change directory into the script's submission directory. We assume here that the job was submitted from a directory in /lustre/scratch/.
    - 8: This command will run the date command.
    - 9: This command will run the executable a.out on 192 cores with a.out.

Go to <u>Menu</u>

Running Jobs: Submitting Batch Jobs - `qsub`

- To submit the batch script named `test.pbs` do:

```
qsub test.pbs
```

- All job resource management handled by Torque.
- Batch scripts can be submitted for execution using the `qsub` command.
- If successfully submitted, a PBS job ID will be returned. This ID can be used to track the job.

Running Jobs: Interactive Batch Jobs

- Batch scripts are useful for submitting a group of commands, allowing them to run through the queue, then viewing the results. It is also often useful to run a job interactively. However, users are not allowed to directly run on compute resources from the login module. Instead, users must use a batch-interactive PBS job. This is done by using the `-I` option to `qsub`.
- For interactive batch jobs, PBS options are passed through `qsub` on the command line:

```
qsub -I -A XXXYYY -q debug -V -l size=24,walltime=1:00:00
```

This request will…

`-I`	Start an interactive session
`-A`	Charge to the "XXXYYY" project
`-q debug`	Run in the debug queue
`-V`	Import the submitting users environment
`-l size=24,walltime=1:00:00`	Request 24 compute cores for one hour

Running Jobs: Altering Batch Jobs – `qdel,qhold,qrls`

- Command: `qdel`
  - Jobs in the queue in any state can be stopped and removed from the queue using the command qdel.
  - For example, to remove a job with a PBS ID of 1234, use the following command: `qdel 1234`

- Command: `qhold`
  - Jobs in the queue in a non-running state may be placed on hold using the qhold command. Jobs placed on hold will not be removed from the queue, but they will not be eligible for execution.
  - For example, to move a currently queued job with a PBS ID of 1234 to a hold state, use the following command: `qhold 1234`

- Command: `qrls`
  - Once on hold the job will not be eligible to run until it is released to return to a queued state. The qrls command can be used to remove a job from the held state.
  - For example, to release job 1234 from a held state, use the following command: `qrls 1234`

Go to Menu

Running Jobs: Monitoring Job Status - `qstat`

PBS and Moab provide multiple tools to view queue, system, and job statuses.
Command: `qstat`
Use `qstat -a` to check the status of submitted jobs:

```
nid00004:
                                                      Req'd  Req'd    Elap
Job ID Username Queue Jobname SessID NDS Tasks Memory Time   S Time
------ -------- ----- ------- ------ --- ----- ------ -----  - -----
29668   user1   batch job2    21909   1  256   --     08:00 R 02:28
29894   user2   batch run128   --     1  128   --     02:30 Q --
29895   user3   batch STDIN   15921   1  1     --     01:00 R 00:10
29896   user2   batch jobL    21988   1  2048  --     01:00 R 00:09
29897   user4   debug STDIN   22367   1  2     --     00:30 R 00:06
29898   user1   batch job1    25188   1  1     --     01:10 C 00:00
```

Job ID	PBS assigned job ID.
Username	Submitting user's user ID.
Queue	Queue into which the job has been submitted.
Jobname	PBS job name. This is given by the PBS -n option in the PBS batch script. Or, if the -n option is not used, PBS will use the name of the batch script.
SessID	Associated session ID.
NDS	PBS node count. Not accurate; will be one.
Tasks	Number of cores requested by the job's -size option.
Req'd Memory	Job's requested memory.
Req'd Time	Job's given wall time.
S	Job's current status. See the status listings below.
Elap Time	Job's time spent in a running status. If a job is not currently or has not been in a run state, the field will be blank.

Status Value	**Meaning**
E	Exiting after having run
H	Held
Q	Queued; eligible to run
R	Running
S	Suspended
T	Being moved to new location
W	Waiting for its execution time
C	Recently completed (within the last 5 minutes)

Go to Menu

Running Jobs: `showq, checkjob`

Command : `showq`

The Moab utility `showq` gives a more detailed description of the queue and displays it in the following states:

Active	These jobs are currently running.
Eligible	These jobs are currently queued awaiting resources. A user is allowed five jobs in the eligible state.
Blocked	These jobs are currently queued but are not eligible to run. Common reasons for jobs in this state are jobs on hold, the owning user currently having five jobs in the eligible state, and running jobs in the longsmall queue.

Command : `checkjob`

The Moab utility `checkjob` can be used to view details of a job in the queue.

For example, if job 736 is a job currently in the queue in a blocked state, the following can be used to view why the job is in a blocked state:

`checkjob 736` The return may contain a line similar to the following:

```
BlockMsg: job 736 violates idle HARD MAXJOB limit of 2 for
      user (Req: 1 In Use: 2)
```

This line indicates the job is in the blocked state because the owning user has reached the limit of two jobs currently in the eligible state.

Go to Menu

Running Jobs: `showstart, showbf, xtprocadmin`

Command : `showstart`

The Moab utility `showstart` gives an estimate of when the job will start.

```
showstart 100315
job 100315 requires 16384 procs for 00:40:00
Estimated Rsv based start in 15:26:41 on Fri Sep 26
23:41:12
Estimated Rsv based completion in 16:06:41 on Sat Sep 27
00:21:12
```

Since the start time may change dramatically as new jobs with higher priority are submitted, so you need to periodically rerun the command.

Command : `showbf`

This command can be used by any user to find out how many processors are available for immediate use on the system. It is anticipated that users will use this information to submit jobs that meet these criteria and thus obtain quick job turnaround times. As such, it is primarily useful for small jobs. This command incorporates down time, reservations, and node state information in determining the available backfill window.

Go to Menu

Running Jobs: Job Execution - `aprun`

- By default, commands will be executed on the job's associated service node.
- The `aprun` command is used to execute a job on one or more compute nodes.
- The XT's layout should be kept in mind when running a job using `aprun`. The XT5 partition currently contains two hex-core processors (a total of 12 cores) per compute node. While the XT4 partition currently contains one quad-core processor (a total of 4 cores) per compute node.
- The `PBS size` option requests compute cores.

Go to **Menu**

Running Jobs: Basic `aprun` options

Option	Description
`-D`	Debug (shows the layout aprun will use)
`-n`	Number of MPI tasks. Note: If you do not specify the number of tasks to aprun, the system will default to 1.
`-N`	Number of tasks per Node. (XT5: 1 – 12) and (XT4: 1 – 4) NOTE: Recall that the XT5 has two Opterons per compute node. On the XT5, to place one task per quad-core Opteron, use -S 1 (not -N 1 as on the XT4). On the XT4, because there is only one Opteron per node, the -S 1 and -N1 will result in the same layout.
`-m`	Memory required per task. Default: 4-core, 8-GB Cray XT4 nodes (8 GB / 4 CPUs = 2 GB) XT4: A maximum of 2GB per core; 2.1GB will allocate two cores for the task
`-d`	Number of threads per MPI task. Note: As of CLE 2.1, this option is very important. If you specify OMP_NUM_THREADS but do not give a -d option, aprun will allocate your threads to a single core. You must use OMP_NUM_THREADS to specify the number of threads per MPI task, and you must use -d to tell aprun how to place those threads.
`-S`	Number of PEs to allocate per NUMA node.
`-ss`	Strict memory containment per NUMA node.

Go to Menu

Running Jobs: XT5 example

`aprun -n 24 ./a.out` will run a.out across 24 cores. This requires two compute nodes. The MPI task layout would be as follows:

Compute Node 0												
Opteron 0							**Opteron 1**					
Core 0	**Core 1**	**Core 2**	**Core 3**	**Core 4**	**Core 5**		**Core 0**	**Core 1**	**Core 2**	**Core 3**	**Core 4**	**Core 5**
0	1	2	3	4	5		6	7	8	9	10	11

Compute Node 1												
Opteron 0							**Opteron 1**					
Core 0	**Core 1**	**Core 2**	**Core 3**	**Core 4**	**Core 5**		**Core 0**	**Core 1**	**Core 2**	**Core 3**	**Core 4**	**Core 5**
12	13	14	15	16	17		18	19	20	21	22	23

The following will place tasks in a round robin fashion.

```
> setenv MPICH_RANK_REORDER_METHOD 0
> aprun -n 24 a.out
```

```
Rank 0,  Node 0, Opteron 0, Core 0
Rank 1,  Node 1, Opteron 0, Core 0
Rank 2,  Node 0, Opteron 0, Core 1
Rank 3,  Node 1, Opteron 0, Core 1
Rank 4,  Node 0, Opteron 0, Core 2
Rank 5,  Node 1, Opteron 0, Core 2
Rank 6,  Node 0, Opteron 0, Core 3
Rank 7,  Node 1, Opteron 0, Core 3
Rank 8,  Node 0, Opteron 0, Core 4
Rank 9,  Node 1, Opteron 0, Core 4
Rank 10, Node 0, Opteron 0, Core 5
Rank 11, Node 1, Opteron 0, Core 5
Rank 12, Node 0, Opteron 1, Core 0
Rank 13, Node 1, Opteron 1, Core 0
Rank 14, Node 0, Opteron 1, Core 1
Rank 15, Node 1, Opteron 1, Core 1
Rank 16, Node 0, Opteron 1, Core 2
Rank 17, Node 1, Opteron 1, Core 2
Rank 18, Node 0, Opteron 1, Core 3
Rank 19, Node 1, Opteron 1, Core 3
Rank 20, Node 0, Opteron 1, Core 4
Rank 21, Node 1, Opteron 1, Core 4
Rank 22, Node 0, Opteron 1, Core 5
Rank 23, Node 1, Opteron 1, Core 5
```

Go to Menu

Running Jobs: XT4 example

`aprun -n8 a.out` will run the MPI executable a.out on a total of eight cores, four cores on two compute nodes. The MPI tasks will be allocated in the following sequential fashion:

Compute Node 0			
Opteron 0			
Core 0	**Core 1**	**Core 2**	**Core 3**
0	1	2	3

Compute Node 1			
Opteron 0			
Core 0	**Core 1**	**Core 2**	**Core 3**
0	1	2	3

The following will place tasks in a round robin fashion.

```
> setenv MPICH_RANK_REORDER_METHOD 0
> aprun -n 8 a.out
Rank 0,  Node 0, Opteron 0, Core 0
Rank 1,  Node 1, Opteron 0, Core 0
Rank 2,  Node 0, Opteron 0, Core 1
Rank 3,  Node 1, Opteron 0, Core 1
Rank 4,  Node 0, Opteron 0, Core 2
Rank 5,  Node 1, Opteron 0, Core 2
Rank 6,  Node 0, Opteron 0, Core 3
Rank 7,  Node 1, Opteron 0, Core 3
```

Running Jobs: Threads

- The system supports threaded programming within a compute node.
- On the XT5, threads may span both Opterons within a single compute node, but cannot span compute nodes.
- Users have a great deal of flexibility in thread placement. Several examples are shown below.
- Note: Under CNL 2.1, threaded codes must use the

  `aprun -d depth` option

  The -d option specifies the number of threads per task. Without the option all threads will be started on the same core. Under previous CNL versions the option was not required. The number of cores used is calculated by multiplying the value of -d by the value of -n.
- Focus of this discussion will be OpenMP threads

Go to Menu

Running Jobs: Threads – XT5 Example

- Example: Launch 4 MPI tasks, each with 6 threads. Place 1 MPI task per Opteron (this requests 2 compute nodes and requires a size request of 24):

```
export OMP_NUM_THREADS=6
> aprun -n4 -d6 -S1 a.out
Rank 0, Thread 0, Node 0, Opteron 0, Core 0 <-- MASTER
Rank 0, Thread 1, Node 0, Opteron 0, Core 1 <-- slave
Rank 0, Thread 2, Node 0, Opteron 0, Core 2 <-- slave
Rank 0, Thread 3, Node 0, Opteron 0, Core 3 <-- slave
Rank 0, Thread 4, Node 0, Opteron 0, Core 4 <-- slave
Rank 0, Thread 5, Node 0, Opteron 0, Core 5 <-- slave
Rank 1, Thread 0, Node 0, Opteron 1, Core 0 <-- MASTER
Rank 1, Thread 1, Node 0, Opteron 1, Core 1 <-- slave
Rank 1, Thread 2, Node 0, Opteron 1, Core 2 <-- slave
Rank 1, Thread 3, Node 0, Opteron 1, Core 3 <-- slave
Rank 1, Thread 4, Node 0, Opteron 1, Core 4 <-- slave
Rank 1, Thread 5, Node 0, Opteron 1, Core 5 <-- slave
Rank 2, Thread 0, Node 1, Opteron 0, Core 0 <-- MASTER
Rank 2, Thread 1, Node 1, Opteron 0, Core 1 <-- slave
Rank 2, Thread 2, Node 1, Opteron 0, Core 2 <-- slave
Rank 2, Thread 3, Node 1, Opteron 0, Core 3 <-- slave
Rank 2, Thread 4, Node 1, Opteron 0, Core 4 <-- slave
Rank 2, Thread 5, Node 1, Opteron 0, Core 5 <-- slave
Rank 3, Thread 0, Node 1, Opteron 1, Core 0 <-- MASTER
Rank 3, Thread 1, Node 1, Opteron 1, Core 1 <-- slave
Rank 3, Thread 2, Node 1, Opteron 1, Core 2 <-- slave
Rank 3, Thread 3, Node 1, Opteron 1, Core 3 <-- slave
Rank 3, Thread 4, Node 1, Opteron 1, Core 4 <-- slave
Rank 3, Thread 5, Node 1, Opteron 1, Core 5 <-- slave
```

Go to Menu

Running Jobs: Threads – XT4 Example

- Example: Launch 2 MPI tasks, each with 4 threads (this requests 2 compute nodes and requires a size request of 8):

```
export OMP_NUM_THREADS=4
> aprun -n2 -d4 a.out
Rank 0, Thread 0, Node 0, Opteron 0, Core 0 <-- MASTER
Rank 0, Thread 1, Node 0, Opteron 0, Core 1 <-- slave
Rank 0, Thread 2, Node 0, Opteron 0, Core 2 <-- slave
Rank 0, Thread 3, Node 0, Opteron 0, Core 3 <-- slave
Rank 1, Thread 0, Node 1, Opteron 0, Core 0 <-- MASTER
Rank 1, Thread 1, Node 1, Opteron 0, Core 1 <-- slave
Rank 1, Thread 2, Node 1, Opteron 0, Core 2 <-- slave
Rank 1, Thread 3, Node 1, Opteron 0, Core 3 <-- slave
```

Go to Menu

Third-Party Software

- NCCS has installed many third-party software packages, libraries, etc., and created module files for them
  - Third-party applications (e.g., MATLAB, GAMESS)
  - Latest versions or old versions not supported by vendor (e.g., fftw/3.1.2)
  - Suboptimal versions to do proof-of-concept work (e.g., blas/ref)
  - Debug versions (e.g., petsc/2.3.3-debug)
- NCCS modules available via `module load` command, installed in `/sw/xt/` directory

Go to Menu

Debugging and Profiling

The following tools are availably on Jaguar for debugging and profiling:

Debugging	Profiling and Analysis
DDT, TotalView	**Cray PAT, Apprentice2, PAPI, TAU etc.**

Always check the compatibility of the compiler options you want to use.

Resources for Users: Getting Started

- About Jaguar

  http://www.nccs.gov/computing-resources/jaguar/

- Quad Core AMD Opteron Processor Overview

  http://www.nccs.gov/wp-content/uploads/2008/04/amd_craywkshp_apr2008.pdf

- PGI Compilers for XT5

  http://www.nccs.gov/wp-content/uploads/2008/04/compilers.ppt

- NCCS Training & Education – archives of NCCS workshops and seminar series, HPC/parallel computing references

  http://www.nccs.gov/user-support/training-education/

- 2009 Cray XT5 Quad-core Workshop

  http://www.nccs.gov/user-support/training-education/workshops/2008-cray-xt5-quad-core-workshop/

Go to Menu

Resources for Users: Advanced Topics

- Debugging Applications Using TotalView

  http://www.nccs.gov/user-support/general-support/software/totalview

- Debugging Applications Using DDT

  http://www.nccs.gov/computing-resources/jaguar/software/?software=ddt

- Using Cray Performance Tools - CrayPat

  http://www.nccs.gov/computing-resources/jaguar/debugging-optimization/cray-pat/

- I/O Tips for Cray XT4

  http://www.nccs.gov/computing-resources/jaguar/debugging-optimization/io-tips/

- NCCS Software

  http://www.nccs.gov/computing-resources/jaguar/software/

Go to Menu

Resources for Users: More Information

- NCCS website

  http://www.nccs.gov/

- Cray Documentation

  http://docs.cray.com/

- Contact us

  help@nccs.gov