Compiling and running applications on Sisu

Accessing the system

- **Just open an ssh connection to sisu.csc.fi**
  - E.g. `ssh -X username@sisu.csc.fi`
- **Alternatively, go to CSC's Scientist User Interface at http://sui.csc.fi**
  - SSH console (Java app)
  - Uploading and downloading files
  - Batch script wizard
- **Moving files to and from the system**
  - scp, e.g. `scp file username@sisu.csc.fi:~/` (to Sisu) scp `username@sisu.csc.fi:~/file .` (from Sisu)
  - Install some suitable client, e.g. FileZilla https://filezilla-project.org/

Primary file systems on Sisu

- **Home space ($HOME, Lustre)**
  - Visible to compute nodes (but perhaps read-only)
  - Has quotas and is backed up
- **Permanent application store ($USERAPPL, Lustre)**
  - For application binaries
  - Visible to compute nodes
  - Backed up
- **Work space (824TB, Lustre, /wrk or $WRKDIR)**
  - Parallel filesystem optimized for large files, high bandwidth
  - Use for job output, restart files etc.
  - Visible to login nodes and compute nodes
  - Not backed up
- **Temporary compile area ($TMPDIR, 0.5 TB)**
  - Local filesystem on login nodes
  - Compile here (not on $HOME or $WRKDIR)
- **There is no local storage on compute nodes**

Primary file systems on Sisu

- **All CSC computing servers have an access to an iRODS based “HPC archive”**
  - Need to do “module load irods” on Sisu
  - See http://datakeskus.csc.fi/fi/3.2.-archiving-data-to-the-archive-server
- **Suggested workflow**
  - Compile a binary in $TMPDIR
  - Store compiled application in $USERAPPL
  - Run in $WRKDIR
  - Post-process and analyse files in $WRKDIR
  - Store summary results in $HOME
  - If results are required long term:
    - Pack up and compress files (tar, gzip, bzip2)
    - Copy to HPC Archive

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

The module tool on the Cray XC

- **How can we get appropriate Compiler, Tools, and Libraries?**
  - The modules tool is used to handle different versions of packages, e.g.
    - `module load compiler/v1`
    - `module swap compiler/v1 compiler/v2`
    - `module load perftools`
- **Taking care of changing of PATH, MANPATH, LM_LICENSE_FILE,.... environment**
  - 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!**
- **It is also easy to setup your own modules for your own software**

Useful module commands

- **Which modules are available?**
  - `module avail`
  - Versions available of a certain module e.g. `module avail cce`
- **Which modules are currently loaded?**
  - `module list`
- **Load/unload software**
  - `module load perftools`
  - `module unload perftools`
- **Change programming environment**
  - `module swap PrgEnv-cray PrgEnv-gnu`
- **Change software version**
  - `module swap cce/8.0.2  cce/7.4.4`

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 (77/90/95/2003/2008) compiler
- These scripts will choose the required compiler version, target architecture options, scientific libraries and their include files automatically from the module environment.
- Use them exactly like you would the original compiler, e.g. To compile `prog1.f90` run

```
ftn -c 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 on Sisu
  - use `module list` to check what is currently loaded
- The Cray MPI module is loaded by default (`cray-mpich2`)
  - 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 `PrgEnvs`.
  - To change the version of the compiler in use, swap the Compiler Module. e.g. `module swap cce cce/8.1.6`

PrgEnv	Compiler Module
PrgEnv-cray	cce
PrgEnv-intel	intel
PrgEnv-gnu	gcc
PrgEnv-pgi	pgi

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 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 (1)

- **Currently static linking is default**
  - May change in the future
  - Already changed when linking for GPUs (XK6 nodes)
- **To decide how to link,**
  1. You can either set `CRAYPE_LINK_TYPE` to "`static`" or "`dynamic`" (e.g. `export CRAYPE_LINK_TYPE=dynamic`) during compilation
  2. Or pass the `-static` or `-dynamic` option to the linking compiler
- **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`)

Dynamic vs Static linking (2)

- **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 `export 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

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		-hnoomp
PrgEnv-intel	-openmp	
PrgEnv-gnu	-fopenmp	
PrgEnv-pgi	-mp	

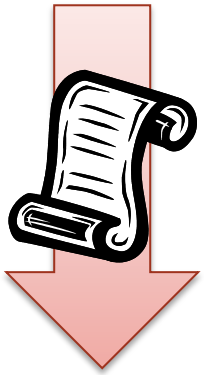
How applications run on a Cray XC

- **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 predicts how long it will run
    - Sisu uses a SLURM scheduler
  - 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 login or batch "MOM" node
  - The scheduler monitors the job and kills any that overrun their runtime prediction
- **User job scripts typically contain two types of statements**
  - Serial commands that are executed by the MOM node, e.g.
    - quick setup and post processing commands
    - e.g. (`rm`, `cd`, `mkdir` etc)
  - Parallel executables that run on compute nodes.
    - Launched using the **`aprun`** command

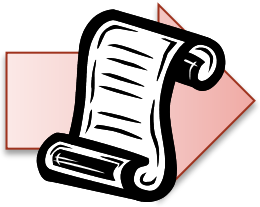
Lifecycle of a batch script

esLogin

`sbatch run.sh`



SLURM Queue Manager



SLURM MOM Node

Example Batch Job Script – `run.sh`

Scheduler Resources

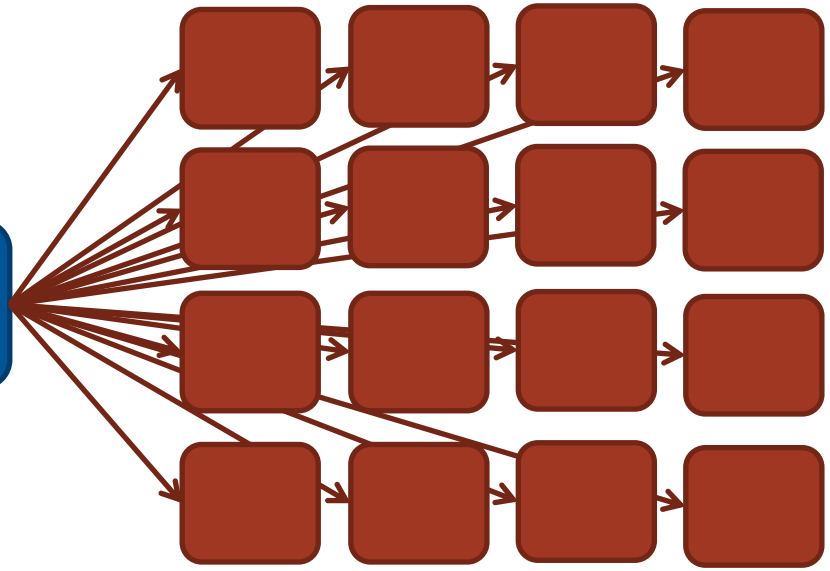
Parallel

Serial

```
#!/bin/bash
#SBATCH --ntasks=1024
#SBATCH --ntasks-per-node=16

aprun –n 1024 –N 16 ./simulation.x
rm –r $WRKDIR/tmp
```

Cray XC Compute Nodes



Running an application on the Cray XC ALPS + aprun

- **ALPS: Application Level Placement Scheduler**
- **aprun is the ALPS application launcher**
  - It **must** be used to run application on the XC compute nodes: interactively or in a batch job
  - If `aprun` is not used, the application is launched on the MOM node (and will most likely fail).
  - `aprun` launches groups of Processing Elements (PEs) on the compute nodes
    (PE == MPI RANK == Coarray Image == UPC Thread)
  - `aprun` man page contains several useful examples
  - The 3 most important parameters to set are:

Description	Option
Total Number of PEs used by the application	-n
Number of PEs per compute node	-N
Number of threads per PE (More precise, the “stride” between 2 PEs on a node)	-d

Running applications on the Cray XC30: Some basic examples

- **Pure MPI application, two nodes**

```
$ aprun -n 32 ./a.out
```

- **Pure MPI application, using only 1 rank per node**
  - 32 MPI tasks, 32 nodes with 32*16=512 cores in total
  - Can be done to increase the available memory for the MPI tasks

```
$ aprun -N 1 -n 32 ./a.out
```

- **Hybrid MPI/OpenMP application, 2 MPI ranks (PE) per node**
  - 32 MPI tasks, 8 OpenMP threads each
  - Ensure the two MPI tasks go to different sockets with the aprun option -S (=how many MPI ranks per socket)
  - need to set `OMP_NUM_THREADS`

```
$ export OMP_NUM_THREADS=8
$ aprun -n 32 -N 2 -d $OMP_NUM_THREADS -S1 ./a.out
```

Scheduling a batch application with SLURM

- The number of required nodes can be specified in the job header
- The job is submitted by the `sbatch` command
- At the end of the exection, output and error files are returned to submission directory
- You can check the status of jobs with `squeue`
- You can delete a job with `scancel <jobid>`

Hybrid MPI + OpenMP

```
#!/bin/bash
#SBATCH --jobname hybrid
#SBATCH --ntasks=64
#SBATCH --cpus-per-task=4
#SBATCH --ntasks-per-node=4

export OMP_NUM_THREADS=4
aprun -n64 -d4 -N4 ./a.out
```

Other SLURM options

- `#SBATCH --output=std.out.%J`
  `#SBATCH --error=std.err.%J`
  File names of stdout and stderr. %J adds the job id to the filename to avoid overwriting
- `#SBATCH --ntasks=N`
  Start N PEs (tasks)
- `#BATCH --ntasks-per-node=M`
  Start M tasks per node. Total Nodes used are N/M
  (+1 if mod(N,M)!=0)
- `#SBATCH --cpus-per-task=8`
  Number of threads per tasks. Most used together with OMP_NUM_THREADS
- `#SBATCH --nodes=8`
  Reserve 8 nodes (128 physical cores) - possibly not using all of them, e.g. for having more memory per core

SLURM and aprun

SLURM	aprun	Meaning
`--ntasks=$PE`	`-n $PE`	Number of PE to start
`--cpus-per-task=$threads`	`-d $threads`	# threads/PE
`--ntasks-per-node=$N`	`-N $N`	# (PEs per node)

Shortcut: aprun's –B option will automatically use the appropriate SLURM settings for –n,-N,-d and –m, e.g.

```
aprun -B ./a.out
```

Watching a launched job on the Cray XC

- **`xtnodestat`**
  - Shows how XC nodes are allocated and corresponding `aprun` commands
- **`apstat`**
  - Shows `aprun` processes status
  - `apstat` overview
  - `apstat -a[ apid ]` info about all the applications or a specific one
  - `apstat -n` info about the status of the nodes
- **Batch `squeue` command**
  - shows batch jobs