An Introduction to High Performance Computing
Stuart Rankin
sjr20@cam.ac.uk
Research Computing Services (http://www.hpc.cam.ac.uk/)
University Information Services (http://www.uis.cam.ac.uk/)

30th July 2019 / UIS Training

Welcome

▶ Please sign in on the attendance sheet.
▶ Keep your belongings with you.
▶ Please ask questions and let us know if you need assistance.

UIS: Research Computing Services

Your trainers for today will be:
▶ Paul Sumption
  Research Computing User Services
▶ Eleftherios Avramidis
  Research Software Engineering
You may be...

- Programmers (or not).
- UNIX power users (or not).
- Researchers wishing to run large, parallel code.
- Researchers wishing to run many, non-parallel cases.
- Researchers interested in big data, machine learning, AI.
- Researchers requiring slightly more than an ordinary workstation.
- Many different disciplines and requirements.

Plan of the Course

Part 1: Basics
Part 2: Research Computing Services HPC
Part 3: Using HPC

10:00 WELCOME
11:00-11:15 Break
12:30-13:30 LUNCH
15:30-15:45 Break
16:30 CLOSE

Prerequisites

- Basic Unix/Linux command line experience:
  Unix: Introduction to the Command Line Interface (self-paced)
  https://www.training.cam.ac.uk/ucs/Course/ucs-unixintro1
- Shell scripting experience is desirable:
  Unix: Simple Shell Scripting for Scientists
  https://www.training.cam.ac.uk/ucs/Course/ucs-scriptsci

Training accounts

- For our practical exercises we will use CSD3 accounts. You should all have applied for a CSD3 account.
- We have reserved two nodes for todays training.
- The training-cpu accounting group is only valid for today.
- Course material can be downloaded from:
  https://www.hpc.cam.ac.uk/training-courses
Security

- Cambridge IT is under constant attack by would-be intruders.
- Choose strong passwords and keep it (or private key passphrase) safe.
- Your UIS password is used for multiple systems so keep it secure!
- Keep the software on your laptops/tablets/PCs up to date — this includes home computers.
- Check out and install free anti-malware software available for work and home:
  https://help.uis.cam.ac.uk/service/security/stay-safe-online/malware
- Don’t share accounts (this is against the rules, and anyone can get their own).

Basics: Why Buy a Big Computer?

What types of big problem might require a “Big Computer”?

Compute Intensive: A single problem requiring a large amount of computation.

Memory Intensive: A single problem requiring a large amount of memory.

Data Intensive: A single problem operating on a large amount of data.

High Throughput: Many unrelated problems to be executed in bulk.

Basics: Compute Intensive Problems

- Distribute the work for a single problem across multiple CPUs to reduce the execution time as far as possible.
- Program workload must be parallelised:
  Parallel programs split into copies (processes or threads).
  Each process/thread performs a part of the work on its own CPU, concurrently with the others.
  A well-parallelised program will fully exercise as many CPUs as there are processes/threads.
- The CPUs typically need to exchange information rapidly, requiring specialized communication hardware.
- Many use cases from Physics, Chemistry, Engineering, Astronomy, Biology...
- The traditional domain of HPC and the Supercomputer.
Basics: Scaling & Amdahl’s Law

- Using more CPUs is not necessarily faster.
- Typically parallel codes have a scaling limit.
- Partly due to the system overhead of managing more copies, but also to more basic constraints;
- Amdahl’s Law (idealized):

\[ S(N) = \frac{1}{(1 - p + \frac{p}{N})} \]

where

- \( S(N) \) is the fraction by which the program has sped up relative to \( N = 1 \)
- \( p \) is the fraction of the program which can be parallelized
- \( N \) is the number of CPUs.

The Bottom Line

- Parallelisation requires effort:
  - There are libraries to help (e.g. OpenMP, MPI).
  - Aim to make both \( p \) and performance per CPU as large as possible.
- The scaling limit: eventually using more CPUs becomes detrimental instead of helpful.

Basics: Data Intensive Problems

- Distribute the data for a single problem across multiple CPUs to reduce the overall execution time.
- The same work may be done on each data segment.
- Rapid movement of data to and from disk is more important than inter-CPU communication.
- Big Data problems of great current interest -
  - Hadoop/MapReduce
  - Life Sciences (genomics) and elsewhere.
**Basics: High Throughput**

- Distribute independent, multiple problems across multiple CPUs to reduce the overall execution time.
- Workload is trivially (or embarrassingly) parallel:
  - Workload breaks up naturally into independent pieces.
  - Each piece is performed by a separate process/thread on a separate CPU (concurrently).
  - Little or no inter-CPU communication.
- Emphasis is on throughput over a period, rather than on performance on a single problem.
- Compute intensive capable $\Rightarrow$ high throughput capable (not conversely).

If you are using lots of R or python, you are probably high throughput, and possibly data intensive or compute intensive.

**Basics: Putting it All Together**

- Each of these types of problem requires combining many CPUs and memory modules.
- Nowadays, there can be many CPUs and memory modules inside a single commodity PC or server.
- HPC involves combining many times more than this.

**Basics: Inside a Modern Computer**

- Today’s commodity servers already aggregate both CPUs and memory to make a single system image in a single box.
- Even small computers now have multiple cores (fully functional CPUs) per socket.
- Larger computers have multiple sockets (each with their own local memory).
Basics: How to Build a Supercomputer

- A supercomputer aggregates contemporary CPUs and memory to obtain increased computing power.
- Usually today these are clusters.

1. Take some (multicore) processors plus some memory to make a node.
   - Could be an off-the-shelf server, or something more special.

2. Connect similar nodes with one or more networks. E.g.
   - Gbit Ethernet: 100 MB/sec
   - Omni-Path: 10 GB/sec
   Faster network is for inter-CPU communication across nodes.
   Slower network is for management and provisioning. Storage may use either.

3. Allocate CPUs & memory to workload
   - Clusters consist of distinct nodes (i.e. separate Linux computers), networked together and controlled centrally by a scheduler.
     - Each process/thread can see only its local node’s CPUs and memory (without help from e.g. MPI).
     - Each process/thread must fit within a single node’s memory.
   - More expensive machines logically bind nodes into a single system.
     - Logically one big node.
     - A single process can see the entire system.
     - E.g. SGI UV.

Basics: Running Applications on a Cluster

- Non-parallel (serial) code
  - For a single node as for a workstation.
  - Typically run as many copies per node as CPUs, assuming node memory is sufficient.
  - Or simply use the memory accompanying the remaining CPUs.
  - Can replicate this across multiple nodes.

- Parallel code
  - Thread parallelism works only within a node.
    E.g. pthreads, OpenMP.
  - MPI parallelism works both intra- and inter-node.
  - Some hybrid codes use both forms of parallel programming.
Basics: Summary

▶ Why have a supercomputer?
  ▶ Single problems requiring great time or big data; many problems.
▶ Most current supercomputers are clusters of separate nodes.
▶ Each node has multiple CPUs and (non-uniform, shared) memory.
▶ Parallel code may use pthreads/OpenMP/MPI within a node, or MPI across multiple nodes.
▶ Serial code uses a single CPU and the memory of one node, but may be copied across many nodes.

Part II: Research Computing Services HPC

Early History: EDSAC (1949–1958)

▶ **Electronic Delay Storage Automatic Calculator**
▶ The second general use, electronic digital (Turing complete) stored program computer
▶ 3,000 valves
▶ 650 instructions per second
▶ 2KB memory in mercury ultrasonic delay lines
▶ One program at a time!
▶ Used in meteorology, genetics, theoretical chemistry, numerical analysis, radioastronomy.
▶ “On a few occasions it worked for more than 24 hours.”
Central HPC in Cambridge

**Created:** 1996 (as the HPCF).

**Mission:** Delivery and support of a large HPC resource for use by the University of Cambridge research community.

**Self-funding:** Paying and non-paying service levels.

**User base:** Includes external STFC & EPSRC plus industrial users.

**Plus:** Dedicated group nodes and research projects.

---

History of Performance

http://www.top500.org

<table>
<thead>
<tr>
<th>Year</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>76.8 Gflop/s</td>
</tr>
<tr>
<td>2002</td>
<td>1.4 Tflop/s</td>
</tr>
<tr>
<td>2006</td>
<td>18.27 Tflop/s</td>
</tr>
<tr>
<td>2010</td>
<td>30 Tflop/s</td>
</tr>
<tr>
<td>2012</td>
<td>183.38 Tflop/s</td>
</tr>
<tr>
<td>2013</td>
<td>183.38 CPU + 239.90 GPU Tflop/s</td>
</tr>
<tr>
<td>2018</td>
<td>2.271 CPU + 1.193 GPU Pflop/s</td>
</tr>
</tbody>
</table>

---

Darwin1 (2006–2012)

Skylake

- Each compute node:
  - 2x16 cores, Intel Skylake 2.6 GHz
  - 192 GB or 384 GB RAM
  - 100 Gb/sec Omni-Path
- 1152 compute nodes.
- 8 login nodes (login-cpu.hpc.cam.ac.uk).

Pascal

- Each compute node:
  - 4x NVIDIA P100 GPU
  - 1x12 cores, Intel Broadwell 2.2 GHz
  - 96 GB RAM
  - 100 Gb/sec (4X EDR) Infiniband
- 90 compute nodes.
- 8 login nodes (login-gpu.hpc.cam.ac.uk).

Coprocesors — GPUs etc

- CPUs are general purpose
- Some types of parallel workload fit vector processing well:
  - Single Instruction, Multiple Data (SIMD)
  - Think pixels on a screen
  - GPUs specialise in this type of work
  - Also competitor many-core architectures such as the Intel Phi
KNL (Intel Phi)

- Each compute node:
  - 64 cores, Intel Phi 7210256 CPUs
  - 96 GB RAM
  - 100 Gb/sec Omni-Path (for MPI and storage)
- 342 compute nodes
- Shared login nodes with Skylake

Cluster Storage

- Lustre cluster filesystem:
  - Very scalable, high bandwidth.
  - Multiple RAID6 back-end disk volumes.
  - Multiple object storage servers.
  - Single metadata server.
  - Tape-backed HSM on newest filesystems.
  - 12 GB/sec overall read or write.
  - Prefers big read/writes over small.

Obtaining an Account and Support

- [https://www.hpc.cam.ac.uk/applications-access-research-computing-services](https://www.hpc.cam.ac.uk/applications-access-research-computing-services)
- Email support@hpc.cam.ac.uk

Part III: Using HPC
Using HPC: Connecting to the RCS Clusters

- SSH secure protocol only.
  *Supports login, file transfer, remote desktop...*
- SSH access is allowed from anywhere.
  *Fail2Ban will ban repeatedly failing clients for 20 minutes.*
- Policies for other clusters may differ.

Connecting: Windows Clients

- putty, pscp, psftp
- WinSCP
  [http://winscp.net/eng/download.php](http://winscp.net/eng/download.php)
- TurboVNC (remote desktop, 3D optional)
- Cygwin (provides an application environment similar to Linux)
  [http://cygwin.com/install.html](http://cygwin.com/install.html)
  *Includes X server for displaying graphical applications running remotely.*
- MobaXterm
  [http://mobaxterm.mobatek.net/](http://mobaxterm.mobatek.net/)

Connecting: Linux/MacOSX/UNIX Clients

- ssh, scp, sftp, rsync
  *Installed (or installable).*
- TurboVNC (remote desktop, 3D optional)
- On MacOSX, install XQuartz to display remote graphical applications.
  [http://xquartz.macosforge.org/landing/](http://xquartz.macosforge.org/landing/)

Connecting: Login

- From Linux/MacOSX/UNIX (or Cygwin):
  `ssh -Y abc123@login-cpu.hpc.cam.ac.uk`
- From graphical clients:
  Host: [login-cpu.hpc.cam.ac.uk](http://login-cpu.hpc.cam.ac.uk)
  Username: `abc123` (your UCAM account name)
- `login-cpu.hpc` will map to a random login node
  *i.e. one of login-e-9, login-e-10, ... , login-e-16*
Connecting: First time login

- The first connection to a particular hostname produces the following:
  - The authenticity of host 'login-cpu (128.232.224.50)' can't be established.
  - ECDSA key fingerprint is SHA256:HsiY1Oe0M8tS6JwR76PeQQA/VB7r9H6TBxO9UVQFsMB.

  Are you sure you want to continue connecting (yes/no)? yes
  Warning: Permanently added 'login-cpu,128.232.224.50' (ECDSA) to the list of known hosts.

- One should always check the fingerprint before typing "yes".
- Graphical SSH clients should ask a similar question.
- Designed to detect fraudulent servers.

Connecting: File Transfer

- With graphical clients, connect as before and drag and drop.
- From Linux/MacOSX/UNIX (or Cygwin):
  - rsync -av old_directory/abc12@login-cpu.hpc.cam.ac.uk:rds/hpc-work/new_directory
  - copies contents of old_directory to˜ /rds/hpc-work/new_directory.
  - rsync -av old_directory
    abc12@login-cpu.hpc.cam.ac.uk:rds/hpc-work/new_directory
  - copies old_directory (and contents) to
    ˜ /rds/hpc-work/new_directory/old_directory.
  - Rerun to update or resume after interruption.
  - All transfers are checksummed.
  - For transfers in the opposite direction, place the remote machine as the first argument.
- Exercise 3 - File transfer.

Connecting: Remote Desktop

- First time use of TurboVNC (recommended):
  - [sjr20@login-e-1 ~]$ vncserver

    You will require a password to access your desktops.

    Password:
    Verify:
    Would you like to enter a view-only password (y/n)? n

    New 'login-e-1:99 (sjr20)' desktop is login-e-1:99

    Starting applications specified in /home/sjr20/.vnc/xstartup
    Log file is /home/sjr20/.vnc/login-e-1:99.log

    NB Choose a different password for VNC to protect your desktop from other users.
- Note the unique host and display number (login-e-1 and :99 here).
Connecting: Remote Desktop

- Remote desktop already running:

  ```
  [sjr20@login-e-1 ~]$ vncserver -list
  TigerVNC server sessions:
  X DISPLAY #   PROCESS ID
  :99          130655
  ```

- Kill it:

  ```
  [sjr20@login-e-1 ~]$ vncserver -kill :99
  Killing Xvnc process ID 130655
  ```

- Typically you only need one remote desktop.
- Keeps running until killed, or the node reboots.

Connecting: Remote Desktop

- To connect to the desktop from Linux:

  ```
  vncviewer -via abc12@login-e-1.hpc.cam.ac.uk localhost:99
  ```

- The display number :99 will be different in general and unique to each desktop.
- You will be asked firstly for your cluster login password, and secondly for your VNC password.
- Press F8 to bring up the control panel.

Using HPC: User Environment

- Scientific Linux 7.x (Red Hat Enterprise Linux 7.x rebuild)
  - bash shell
  - Gnome or XFCE4 desktop (if you want)
  - GCC, Intel, PGI compilers and other development software.
- But you don’t need to know that.
- NOT Ubuntu or Debian!
- CentOS 7 is OK.

User Environment: Filesystems

- `/home/abc123`
  - 40GB quota.
  - Visible equally from all nodes.
  - Single storage server.
  - Hourly, daily, weekly snapshots copied to tape.
  - Not intended for job outputs or large/many input files.
- `/rds/user/abc123/hpc-work a.k.a. /home/abc123/rds/hpc-work`
  - Visible equally from all nodes.
  - Larger and faster (1TB initial quota).
  - Intended for job inputs and outputs.
  - Not backed up.
- Research Data Storage
  - https://www.hpc.cam.ac.uk/research-data-storage-services
Filesystems: Quotas

▶ quota

[abc123@login-e-1 ~]$ quota

Filesystem GiBytes quota limit grace files quota limit grace User/group
/home 10.6 40.0 40.0 0 ----- No ZFS File Quotas ----- U:abc123
/rda-d2 1.0 1024.0 1126.4 - 8 1048576 1048576 - G:abc123

▶ Aim to stay below the soft limit (quota).
▶ Once over the soft limit, you have 7 days grace to return below.
▶ When the grace period expires, or you reach the hard limit (limit), no more data can be written.
▶ It is important to rectify an out of quota condition ASAP.

Filesystems: Permissions

▶ Be careful and if unsure, please ask support.
  ▶ Can lead to accidental destruction of your data or account compromise.
▶ Avoid changing the permissions on your home directory.
  ▶ Files under /home are particularly security sensitive.
  ▶ Easy to break passwordless communication between nodes.

User Environment: Software

▶ Free software accompanying Red Hat Enterprise Linux is (or can be) provided.
▶ Other software (free and non-free) is available via modules.
▶ Some proprietary software may not be generally accessible.
▶ New software may be possible to provide on request.
▶ Self-installed software should be properly licensed.
▶ sudo will not work. (You should be worried if it did.)
▶ Docker-compatible containers can now be downloaded and used via singularity.
Modules load or unload additional software packages.
- Some are **required** and automatically loaded on login.
- Others are optional extras, or possible replacements for other modules.
- **Beware** unloading default modules in `/bashrc`.
- **Beware** overwriting environment variables such as `PATH` and `LD_LIBRARY_PATH` in `/bashrc`. If necessary append or prepend.

Currently loaded:

```bash
module list
Currently Loaded Modulefiles:
1) dot
2) slurm
3) turbervnc/2.0.1
4) vgl/2.5.1/64
5) singularity/current
6) rhe17/global
7) intel/compilers/2017.4
8) intel/mkl/2017.4
9) intel/impi/2017.4/intel
10) intel/libe/ldb/2017.4
11) intel/libe/tbb/2017.4
12) intel/libe/ipp/2017.4
13) intel/libe/daal/2017.4
14) intel/bundles/complib/2017.4
15) rhe17/default-peta4
```

Available:

```bash
module av
```

**What is:**

```bash
module whatis openmpi-1.10.7-gcc-5.4.0-jdc7f4f
openmpi-1.10.7-gcc-5.4.0-jdc7f4f: The Open MPI Project is an open source...
```

**Load:**

```bash
module load openmpi-1.10.7-gcc-5.4.0-jdc7f4f
```

**Unload:**

```bash
module unload openmpi-1.10.7-gcc-5.4.0-jdc7f4f
```

**Matlab**

```bash
module load matlab/r2017b
```

**Invoking matlab in batch mode:**

```bash
matlab -nodisplay -nojvm -nosplash command
```

where the file `command.m` contains your matlab code.

**The University site license contains the Parallel Computing Toolbox.**

**MATLAB Parallel Server** is also available.
User Environment: Environment Modules

- **Purge:**
  
  ```
  module purge
  ```
  
- **Defaults loaded on login (vary by cluster):**

  ```
  module show rhel7/default-peta4
  ```

  /usr/local/Cluster-Config/modulefiles/rhel7/default-peta4:

  ```
  module-whatis default user environment for Peta4 nodes with Intel MPI
  setenv OMP_NUM_THREADS 1
  module add dot slurm turbovnc vgl singularity
  module add rhel7/global
  module add intel/bundles/complib/2017.4
  ```

  ```
  module load rhel7/default-peta4
  ```

- **Run time environment must match compile time environment.**

User Environment: Compilers

- **Intel:** icc, icpc, ifort (recommended)

  ```
  icc -O3 -xHOST -ip code.c -o prog
  mpicc -O3 -xHOST -ip mpi_code.c -o mpi_prog
  ```

- **GCC:** gcc, g++, gfortran

  ```
  gcc -O3 -mtune=native code.c -o prog
  mpicc -cc=gcc -O3 -mtune=native mpi_code.c -o mpi_prog
  ```

- **PGI:** pgcc, pgCC, pgf90

  ```
  pgcc -O3 -tp=skylake code.c -o prog
  mpicc -cc=pgcc -O3 -tp=skylake mpi_code.c -o mpi_prog
  ```

Exercise 5: Modules and Compilers

Using HPC: Job Submission

- **Compute resources are managed by a scheduler:**
  
  SLURM/PBS/SGE/LSF/…

  - Jobs are submitted to the scheduler
    - analogous to submitting jobs to a print queue
    - a file (**submission script**) is copied and queued for processing.
Using HPC: Job Submission

- Jobs are submitted from the login node — not itself managed by the scheduler.
- Jobs may be either non-interactive (batch) or interactive.
- Batch jobs run a shell script on the first of a list of allocated nodes.
- Interactive jobs provide a command line on the first of a list of allocated nodes.

Job Submission: Using SLURM

- Prepare a shell script and submit it to SLURM:

  `[abc123@login-e-1]$ sbatch slurm_submission_script
  Submitted batch job 790299`

Job Submission: Show Queue

- Submitted job scripts are copied and stored in a queue:

  `[abc123@login-e-1]$ squeue -u abc123
  JOBID  PARTITION  NAME   USER   ST   TIME  NODES NODELIST(REASON)
  790299  skylake    Test3  abc123  PD   0:00  2 (PriorityResourcesAssocCPUMinsLimit)
  790290  skylake    Test2  abc123  R   27:56:10 2 (queue="[1,10]"

Jobs may use part or all of one or more nodes — the owner can specify --exclusive to force exclusive node access (automatic on KNL).
- Template submission scripts are available under /usr/local/Cluster-Docs/SLURM.
Job Submission: Monitor Job

▶ Examine a particular job:
[abc123@login-e-1]$ scontrol show job=790290

Job Submission: Accounting Commands

▶ How many core hours available do I have?

mybalance

<table>
<thead>
<tr>
<th>User</th>
<th>Usage</th>
<th>Account Usage</th>
<th>Account Limit Available (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sjr20</td>
<td>3</td>
<td>SUPPORT-CPU 2,929</td>
<td>22,425,600 22,422,671</td>
</tr>
<tr>
<td>sjr20</td>
<td>0</td>
<td>SUPPORT-CPU 0</td>
<td>87,600 87,600</td>
</tr>
</tbody>
</table>

▶ How many core hours does some other project or user have?

gbalance -p SUPPORT-CPU

<table>
<thead>
<tr>
<th>User</th>
<th>Usage</th>
<th>Account Usage</th>
<th>Account Limit Available (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pfb29</td>
<td>2,925</td>
<td>SUPPORT-CPU 2,929</td>
<td>22,425,600 22,422,671</td>
</tr>
<tr>
<td>sjr20</td>
<td>3</td>
<td>SUPPORT-CPU 2,929</td>
<td>22,425,600 22,422,671</td>
</tr>
</tbody>
</table>

... (Use -u for user.)

▶ List all jobs charged to a project/user between certain times:

gstatement -p SUPPORT-CPU -u xyz10 -s "2018-04-01-00:00:00" -e "2018-04-30-00:00:00"

<table>
<thead>
<tr>
<th>JobID</th>
<th>User</th>
<th>Account</th>
<th>JobName</th>
<th>Partition</th>
<th>End</th>
<th>ExitCode</th>
<th>State</th>
<th>CompHrs</th>
</tr>
</thead>
<tbody>
<tr>
<td>263</td>
<td>xyz10</td>
<td>support-c+</td>
<td>_interact+</td>
<td>skylake</td>
<td>2018-04-18T19:44:40</td>
<td>0:0</td>
<td>TIMEOUT</td>
<td>1.0</td>
</tr>
<tr>
<td>264</td>
<td>xyz10</td>
<td>support-c+</td>
<td>_interact+</td>
<td>skylake</td>
<td>2018-04-18T19:48:07</td>
<td>0:0</td>
<td>CANCELLED</td>
<td>0.1</td>
</tr>
<tr>
<td>275</td>
<td>xyz10</td>
<td>support-c+</td>
<td>_interact+</td>
<td>skylake</td>
<td>Unknown</td>
<td>0:0</td>
<td>RUNNING</td>
<td>0.3</td>
</tr>
</tbody>
</table>

... (Use -s and -e instead of -e and -s.)

Job Submission: Cancel Job

▶ Cancel a particular job:
[abc123@login-e-1]$ scancel 790290

Job Submission: Scripts

▶ SLURM

In /usr/local/Cluster-Docs/SLURM, see examples: slurm_submit.peta4-skylake, slurm_submit.wilkes2.

#!/bin/bash

# Name of the job:
#SBATCH -J myjob

# Which project should be charged:
#SBATCH -A CHANGEME

# How many whole nodes should be allocated?
#SBATCH --nodes=1

# How many tasks will there be in total? (<= nodes*32)
#SBATCH --ntasks=116

# How much wallclock time will be required?
#SBATCH --time=02:00:00

# Select partition:
#SBATCH -p skylake

... (Use -t for tasks.)

▶ #SBATCH lines are structured comments — correspond to sbatch command line options.

▶ The above job will be given 1 cpu16 cpus on 1 node for 2 hours (by default there is 1 task per node, and 1 cpu per task).
Serial jobs requiring large memory, or OpenMP codes.

```bash
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=32 # Whole node
#SBATCH --mem=5990
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS=32 # For OpenMP across 32 cores
$application $options
...
```

Serial jobs requiring large memory, or OpenMP codes.

```bash
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=16 # Half node
#SBATCH --mem=5990
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS=16 # For OpenMP across 16 cores
$application $options
...
```
Job Submission: Single Node Jobs

- Serial jobs requiring large memory, or OpenMP codes.

```bash
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=32 # Whole node
#SBATCH --mem=5990
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS=16 # For OpenMP across 16 cores (using all memory)
$application $options
...
```

Job Submission: MPI Jobs

- Parallel job across multiple nodes.

```bash
#!/bin/bash
...
#SBATCH --nodes=4
#SBATCH --ntasks=128 # i.e. 32x4 MPI tasks in total.
#SBATCH --cpus-per-task=2
...
mpirun -np 128 $application $options
...
```

- SLURM-aware MPI launches remote tasks via SLURM (doesn’t need a list of nodes).

Job Submission: Hybrid Jobs

- Parallel jobs using both MPI and OpenMP.

```bash
#!/bin/bash
...
#SBATCH --nodes=4
#SBATCH --ntasks=64 # i.e. 16x4 MPI tasks in total.
#SBATCH --cpus-per-task=2
...
mpirun -ppn 16 -np 64 $application $options
...
```

- SLURM-aware MPI launches remote tasks via SLURM (doesn’t need a list of nodes).

- This job uses 128 CPUs (each MPI task splits into 2 OpenMP threads).
Job Submission: High Throughput Jobs

- Multiple serial jobs across multiple nodes.
- Use `srun` to launch tasks (job steps) within a job.

```bash
#!/bin/bash
...
#SBATCH --nodes=2
...
cd directory_for_job1
srun --exclusive -N 1 -n 1 $application $options > output 2> err &
...

#SBATCH --nodes=2
...
cd directory_for_job2
srun --exclusive -N 1 -n 1 $application $options > output 2> err &
...

#SBATCH --nodes=2
...

#SBATCH --nodes=2
...
...

for job64
srun --exclusive -N 1 -n 1 $application $options > output 2> err &
wait
```

- Exercise 6–8 - Submitting Jobs.

---

Job Submission: Interactive

- Compute nodes are accessible via SSH while you have a job running on them.
- Alternatively, submit an interactive job:

```bash
dsintr -A TRAINING-CPU -N1 -n8 -t 1:0:0
```

- Within the window (screen session):
  - Launches a shell on the first node (when the job starts).
  - Graphical applications should display correctly (if they did from the login node).
  - Create new shells with `ctrl-a c`, navigate with `ctrl-a n` and `ctrl-a p`.
  - `ssh` or `srun` can be used to start processes on any nodes in the job.
  - SLURM-aware MPI will do this automatically.

---

Job Submission: Array Jobs

- [http://slurm.schedmd.com/job_array.html](http://slurm.schedmd.com/job_array.html)
- Used for submitting and managing large sets of similar jobs.
- Each job in the array has the same initial options.
- **SLURM**

```
[abc123@login-e-1]$ sbatch --array=1-7:21,3,5,7 -A TRAINING-CPU submit_script
Submitted batch job 791609
```

```
[abc123@login-e-1]$ squeue -u abc123
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
791609_1 skylake hpl abc123 R 0:06 1 cpu-a-6
791609_2 skylake hpl abc123 R 0:06 1 cpu-a-16
791609_3 skylake hpl abc123 R 0:06 1 cpu-a-7
791609_4 skylake hpl abc123 R 0:06 1 cpu-a-7
791609_5 skylake hpl abc123 R 0:06 1 cpu-a-7
791609_6 skylake hpl abc123 R 0:06 1 cpu-a-7
791609_7 skylake hpl abc123 R 0:06 1 cpu-a-7
```

- i.e. `$(SLURM_ARRAY_JOB_ID).$(SLURM_ARRAY_TASK_ID)`

- **SLURM**

```
SLURM_ARRAY_JOB_ID = SLURM_JOBID for the first element.
```

---

Job Submission: Array Jobs (ctd)

- Updates can be applied to specific array elements using `$(SLURM_ARRAY_JOB_ID).$(SLURM_ARRAY_TASK_ID)`
- Alternatively operate on the entire array via `$(SLURM_ARRAY_JOB_ID)`.
- Some commands still require the SLURM_JOB_ID (sacct, sreport, sshare, sstat and a few others).
- Exercise 9 - Array Jobs.
Scheduling

▶ SLURM scheduling is multifactor:
  ▶ QoS — payer or non-payer?
  ▶ Age — how long has the job waited?
    Don’t cancel jobs that seem to wait too long.
  ▶ Fair Share — how much recent usage?
    Payers with little recent usage receive boost.
  ▶ sprio -j jobid
▶ Backfilling
  ▶ Promote lower priority jobs into gaps left by higher priority jobs.
  ▶ Demands that the higher priority jobs not be delayed.
  ▶ Relies on reasonably accurate wall time requests for this to work.
  ▶ Jobs of default length will not backfill readily.

Wait Times

▶ 36 hour job walltimes are permitted.
▶ This sets the timescale at busy times (without backfilling).
▶ Use backfilling when possible.
▶ Short (1 hour or less) jobs have higher throughput.

Checkpointing

▶ Insurance against failures during long jobs.
▶ Restart from checkpoints to work around finite job length.
▶ Application native methods are best. Failing that, one can try
  DMTCP:
  http://dmtcp.sourceforge.net/index.html

Job Submission: Scheduling Top Dos & Don’ts

▶ Do . . .
  ▶ Give reasonably accurate wall times (allows backfilling).
  ▶ Check your balance occasionally (mybalance).
  ▶ Test on a small scale first.
  ▶ Implement checkpointing if possible (reduces resource wastage).
▶ Don’t . . .
  ▶ Request more than you need
    — you will wait longer and use more credits.
  ▶ Cancel jobs unnecessarily
    — priority increases over time.