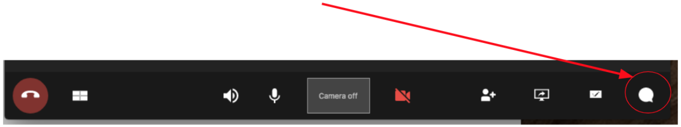
How to submit and run jobs on Compute Canada HPC systems

Alex Razoumov
alex.razoumov@westgrid.ca

✔ copy of these slides at `http://bit.ly/2D40VML`

To ask questions

- Websteam: email **info@westgrid.ca**
- Vidyo: use the GROUP CHAT to ask questions



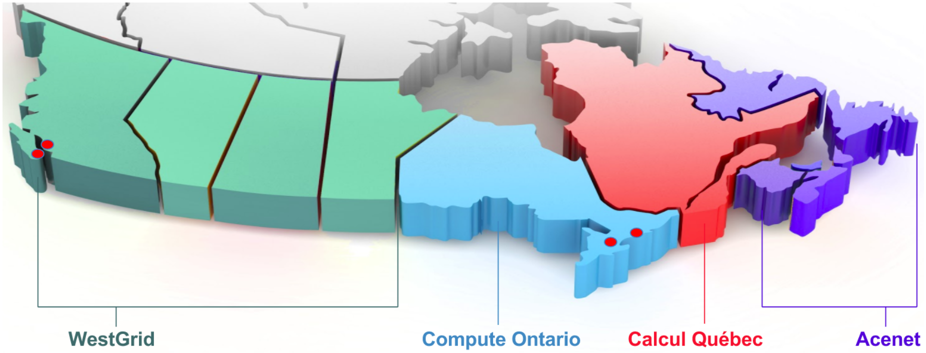
- Please mute your mic unless you have a question
- Feel free to ask questions via audio at any time

Cluster hardware overview

Compute Canada’s national systems

Since 2016, CC has been renewing its national infrastructure:

- **Arbutus** is an extension to the West Cloud (UVic), in production since Sep. 2016
- **Cedar** (SFU), **Graham** (Waterloo) are general-purpose clusters, in production since June 2017
- **Niagara** is a large parallel cluster (Toronto), in production since April 2018
- One more general-purpose system (**Béluga**) in Québec, in production in early 2019



	Cedar *cedar.computecanada.ca*	Graham *graham.computecanada.ca*
purpose	general-purpose cluster for a variety of workloads	
specs	`https://docs.computecanada.ca/wiki/Cedar`	`https://docs.computecanada.ca/wiki/Graham`
processor count	58,416 CPUs and 584 GPUs	33,472 CPUs and 320 GPUs
interconnect	100Gbit/s Intel OmniPath, non-blocking to 1024 cores	56-100Gb/s Mellanox InfiniBand, non-blocking to 1024 cores
128GB **base nodes**	**576 nodes**: 32 cores/node	**864 nodes**: 32 cores/node
256GB **large nodes**	**128 nodes**: 32 cores/node	**56 nodes**: 32 cores/node
0.5TB **bigmem500**	**24 nodes**: 32 cores/node	**24 nodes**: 32 cores/node
1.5TB **bigmem1500**	**24 nodes**: 32 cores/node	-
3TB **bigmem3000**	**4 nodes**: 32 cores/node	**3 nodes**: 64 cores/node
128GB **GPU base**	**114 nodes**: 24-cores/node, 4 NVIDIA P100 Pascal GPUs with 12GB HBM2 memory	**160 nodes**: 32-cores/node, 2 NVIDIA P100 Pascal GPUs with 12GB HBM2 memory
256GB **GPU large**	**32 nodes**: 24-cores/node, 4 NVIDIA P100 Pascal GPUs with 16GB HBM2 memory	-
192 GB Skylake base nodes	640 nodes: 48 cores/node (similar to Niagara's)	-

- All nodes have on-node SSD storage
- On Cedar use `--constraint={broadwell,skylake}` to specify CPU architecture (usually not needed)

	Niagara *niagara.computecanada.ca*
purpose	for large parallel jobs, ideally ≥1,000 cores with an allocation
specs	`https://docs.computecanada.ca/wiki/Niagara` and `https://docs.scinet.utoronto.ca`
processor count	60,000 CPUs and no GPUs
interconnect	EDR Infiniband (Dragonfly+, completely connected topology, dynamic routing), 1:1 to 432 nodes, effectively 2:1 beyond that
192GB **base nodes**	**1,500 nodes**: 40 cores/node

- No local disk, nodes booting off the network, small RAM filesystem
- All cores are Intel Skylake 6148 Gold (2.4 GHz, AVX512)
- Authentication via CC accounts; might need to request access in the early stages, long-term regular access for all CC account holders
- Users with an allocation: job sizes up to 1000 nodes and 24 hours max runtime
- Users without an allocation: job sizes up to 20 nodes and 12 hours max runtime
- Maximum number of jobs per user: running 50, queued 150

Accessing resources: RAS vs. RAC

- ~20% of compute cycles available via the Rapid Access Service (RAS)
  - available to all CC users via default queues
  - you can start using it as soon as you have a CC account
  - shared pool with resources allocated via “fair share” mechanism
  - will be sufficient to meet computing needs of many research groups
- ~80% of compute cycles allocated via annual Resource Allocation Competitions (RAC)
  - apply if you need >50 CPU-years or >10 GPU-years
  - only PIs can apply, allocation per research group
  - announcement in the fall of each year via email to all users
  - 2018 RAC: 469 applications, success rate (awarded vs. requested) 55.1% for CPUs, 20.5% for GPUs, 73% for storage
  - 2019 RAC submission period Sep-27 to Nov-08, start of allocations in early April 2019

File systems

Details at `https://docs.computecanada.ca/wiki/Storage_and_file_management`

filesystem	quotas	backed up?	purged?	performance	mounted on compute nodes?
$HOME	**50GB, 5e5 files per user 100GB/user on Niagara**	nightly, latest snapshot	no	medium	yes
$SCRATCH	**20TB, 1e6 files per user** except when full	no	yes	high for large files	yes
$PROJECT (long-term disk storage)	**1TB, 5e5 files per user and 10TB, 5e6 files per group** via RAC	nightly	no	medium	yes
/nearline (tape archive)	**5TB per group** via RAC	no	no	medium to low	no
/localscratch	**none**	no	maybe	very high	local

- Wide range of options from high-speed temporary storage to different kinds of long-term storage
- Checking disk usage: run `quota` command (aliased to `diskusage_report`)
- Requesting more storage: small increases via **support@computecanada.ca**, large requests via RAC
- **Niagara only**: 256TB $BBUFFER (burst buffer, NVMe SSDs) for low-latency I/O, space must be requested (10TB quota), files should be moved to $SCRATCH at the end of a job

How to log in to a cluster

Logging into the systems: SSH client + your CC account

- On Mac or Linux in terminal:

```
$ ssh yourUsername@cedar.computecanada.ca         # Cedar login node
$ ssh yourUsername@graham.computecanada.ca        # Graham login node
$ ssh yourUsername@niagara.computecanada.ca       # Niagara login node
```

- On Windows many options:
  - MobaXTerm `https://docs.computecanada.ca/wiki/Connecting_with_MobaXTerm`
  - PuTTY `https://docs.computecanada.ca/wiki/Connecting_with_PuTTY`
  - if using Chrome browser, from `https://chrome.google.com/webstore/category/extensions` install Secure Shell Extension
  - bash from the Windows Subsystem for Linux (WSL) – starting with Windows 10, need to enable developer mode and then WSL
- SSH key pairs are very handy to avoid typing passwords
  - implies secure handling of private keys, non-empty passphrases
  - `https://docs.computecanada.ca/wiki/SSH_Keys`
  - `https://docs.computecanada.ca/wiki/Using_SSH_keys_in_Linux`
  - `https://docs.computecanada.ca/wiki/Generating_SSH_keys_in_Windows`
- GUI connection: X11 forwarding (through `ssh -Y`), VNC, x2go
- Client-server workflow in selected applications, both on login and compute nodes

Linux command line

- All system run Linux (CentOS 7) $\Rightarrow$ you need to know basic command line
  - for very basic introduction, `https://hpc-carpentry.github.io/hpc-shell`
  - other online tutorials , e.g. `http://bit.ly/2vH3j8v`
  - attend one of our Software Carpentry 3-hour in-person *bash* sessions
- Much typing can be avoided by using bash aliases, functions, ~/.bashrc, hitting TAB

File commands

ls	directory listing
ls -alF	pass command arguments
cd dir	change directory to dir
cd	change to home
pwd	show current directory
mkdir dir	create a directory
rm file	delete file
rm -r dir	delete directory
rm -f file	force remove file
rm -rf dir	force remove directory
cp file target	copy file to target
mv file target	rename or move file to target
ln -s file link	create symbolic link to file
touch file	create or update file

Paths

relative vs. absolute paths

meaning of **~** **.** **..**

File commands

command > file	redirect command output to file
command >> file	append command output to file
more file	page through contents of file
cat file	print all contents of file
head -n file	output the first n lines of file
tail -n file	output the last n lines of file
tail -f file	output the contents of file as it grows

Process management

top	display your currently active processes
ps	display all running processes
kill pid	kill process ID pid

File permissions

chmod -R u+rw,g-rwx,o-rwx file	set permissions

Environment variables and aliases

export VAR='value'	set a variable
echo $VAR	print a variable
alias ls='ls -aFh'	set an alias command

Searching

grep pattern files	search for pattern in files
command | grep pattern	example of a pipe
find . -name '*.txt' | wc -l	another pipe

Other tools

man command	show the manual for command
command --help	get quick help on command
df -kh .	show disk usage
du -kh .	show directory space usage

Compression and archiving

tar cvf file.tar files	create a tar file
tar xvf file.tar	extract from file.tar
gzip file	compress file
gunzip file.gz	uncompress file.gz

Loops

for i in *tex; do wc -l $i; done	loop example

Editing remote files from the command line

- Inside the terminal
  - *nano* is the easiest option for novice users
  - *emacs -nw* is available for power users
  - *vi* and *vim* are also available (for die-hard fans: basic, difficult to use)
- Remote graphical *emacs* not recommended
  - you would connect via ssh with an X11 forwarding flag (-X, -Y)
- My preferred option: local *emacs* on my laptop editing remote files via ssh with emacs's built-in package *tramp*
  - need to add to your ∼/.emacs

```
(require 'tramp)
(setq tramp-default-method "ssh")
```

  - only makes sense with a working ssh-key pair
  - your private key is your key to the cluster, so don't share it!
  - cross-platform (Windows/Linux/Mac)
- Same functionality exists in several other text editors
  - Atom (open-source, Windows/Linux/Mac)
  - Sublime Text (commercial, Windows/Linux/Mac)

Cluster software environment

Cluster software environment at a glance

- **Programming languages**: **C/C++, Fortran 90**, Python, R, Java, Matlab, **Chapel**
  - several different versions and flavours for most of these
  - for more details see `https://docs.computecanada.ca/wiki/Programming_Guide`
  - parallel programming environments in the next slide
- **Job scheduler**: Slurm open-source scheduler and resource manager
- **Popular software**: installed by staff, listed at `https://docs.computecanada.ca/wiki/Available_software`
  - lower-level, not performance sensitive packages installed via Nix package manager
  - general packages installed via EasyBuild framework
  - everything located under /cvmfs, loaded via modules (next slide)
- Other software
  - email *support@computecanada.ca* with your request, or
  - can compile in your own space (feel free to ask staff for help)

Parallel programming environment

- CPU parallel development support: OpenMP (since 1997), MPI (since 1994)
  - OpenMP is a language extension for C/C++/Fortran provided by compilers, implements shared-memory parallel programming
  - MPI is a library with implementations for C/C++/Fortran/Python/R/etc, designed for distributed-memory parallel environments, also works for CPU cores with access to common shared memory
  - industry standards for the past 20+ years
- Chapel is a open-source parallel programming language
  - ease-of-use of Python + performance of a traditional compiled language
  - combines shared- and distributed-memory models; data and task parallelism for both
  - multi-resolution: high-level parallel abstractions + low-level controls
  - in my opinion, by far the best language to learn parallel programming $\Rightarrow$ we teach it as part of HPC Carpentry, in summer schools and full-day Chapel workshops
  - experimental support for GPUs
  - relative newcomer to HPC, unfortunately still rarely used outside its small/passionate community
- GPU parallel development support: CUDA, OpenCL, OpenACC

Software modules

- Use appropriate modules to load centrally-installed software (might have to select the right version)

```
$ module avail <name>       # search for a module (if listed)
$ module spider <name>      # will give a little bit more info
$ module list               # show currently loaded modules
$ module load moduleName
$ module unload moduleName
$ module show moduleName    # show commands in the module
```

- All associated prerequisite modules will be automatically loaded as well
- Modules must be loaded before a job using them is submitted
  - alternatively, can load a module from the job submission script

Installed compilers

	Intel intel/2016.4 and openmpi/2.1.1 loaded by default		**GNU** module load gcc/5.4.0 (∗)		**PGI** module load pgi/17.3 (∗)	
C	icc	mpicc	gcc -O2	mpicc	pgcc	mpicc
Fortran 90	ifort	mpifort	gfortran -O2	mpifort	pgfortran	mpifort
C++	icpc	mpiCC	g++ -O2	mpiCC	pgc++	mpiCC
OpenMP flag	-qopenmp		-fopenmp		-mp	

(∗) in both cases intel/2016.4 will be unloaded and openmpi/2.1.1 reloaded automatically

- mpiXX scripts invoke the right compiler and link your code to the correct MPI library
- use `mpiXX --show` to view the commands they use to compile and link

Other essential tools

- File management with *gzip* and *tar*
- Build automation with *make*
- Version control with *git* or *mercurial*
  - normally taught as a 3-hour Software Carpentry course
- Terminal multiplexer with *screen* or *tmux*
  - share a physical terminal between several interactive shells
  - access the same interactive shells from many different terminals
  - very useful for persistent sessions, e.g., for compiling large codes
- *VNC* and *x2go* clients for remote interactive GUI work
  - on Cedar in `$HOME/.vnc/xstartup` can switch from *twm* to *mwm*/etc. as your default window manager
  - can run VNC server on compute nodes

Python

Details at `https://docs.computecanada.ca/wiki/Python`

- Initial setup:

```
module avail python       # several versions available
module load python/3.5.4
virtualenv bio     # install Python tools in your $HOME/bio
source ~/bio/bin/activate
pip install numpy
...
```

- Usual workflow:

```
source ~/bio/bin/activate    # load the environment
python
...
deactivate
```

R - details at `https://docs.computecanada.ca/wiki/R`

```
$ module spider r      # several versions available
$ module load r/3.4.3
$ R
> install.packages("sp")    # install packages from cran.r-project.org; it'll suggest
                               # installing into your personal library $HOME/R/
$ R CMD INSTALL -l $HOME/myRLibPath package.tgz    # install non-CRAN packages
```

- Running R scripts: `Rscript script.R`
- Installing and running Rmpi: see our documentation
- pbdR (*Programming with Big Data in R*): high-performance, high-level interfaces to MPI, ZeroMQ, ScaLAPACK, NetCDF4, PAPI, etc. `http://r-pbd.org`
- Launching multiple serial R calculations via *array jobs* (details in Scheduling)
  - inside the *job submission script* use something like

```
Rscript script${SLURM_ARRAY_TASK_ID}.R
```

or

```
export params=${SLURM_ARRAY_TASK_ID}
Rscript script.R
```

and then inside `script.R`:

```
s <- Sys.getenv('params')
filename <- paste('/path/to/input', s, '.csv', sep='')
```

How to move data

File transfer

In Mac/Linux terminal or in Windows MobaXterm you have several good options:

(1) use *scp* to copy individual files and directories

```
$ scp filename yourUsername@cedar.computecanada.ca:/path/to
$ scp yourUsername@cedar.computecanada.ca:/path/to/filename localPath
```

(2) use *rsync* to sync files or directories

```
$ flags='-av --progress --delete'
$ rsync $flags localPath/*pattern* yourUsername@cedar.computecanada.ca:/path/to
$ rsync $flags yourUsername@cedar.computecanada.ca:/path/to/*pattern* localPath
```

(3) use *sftp* for more interactive access

```
$ sftp yourUsername@cedar.computecanada.ca
sftp> help
```

- Windows PuTTY uses `pscp` command for secure file transfer
- Number of GUI clients for *sftp* file transfer, e.g. `https://filezilla-project.org`

Globus file transfer

Details at `https://docs.computecanada.ca/wiki/Globus`

- The CC Globus Portal `https://globus.computecanada.ca` is a fast, reliable, and secure service for big data transfer (log in with your CC account)
- Easy-to-use web interface to automate file transfers between any two *endpoints*
  - an *endpoint* could be a CC system, another supercomputing facility, a campus cluster, a lab server, a personal laptop (requires Globus Connect Personal app)
  - runs in the background: initialize transfer and close the browser, it'll email status
- Uses GridFTP transfer protocol: much better performance than scp, rsync
  - achieves better use of bandwidth with multiple simultaneous TCP streams
  - some ISPs and Eduroam might not always play well with the protocol (throttling)
- Automatically restarts interrupted transfers, retries failures, checks file integrity, handles recovery from faults
- Command-line interface available as well

Quick scheduling primer and how to submit jobs

figures and some material in this section created by Kamil Marcinkowski

Why job scheduler?

- Tens of thousands of CPUs, many thousands of simultaneous jobs ⇒ need an automated solution to manage a queue of pending jobs, allocate resources to users, start/stop/monitor jobs ⇒ we use Slurm open-source scheduler/resource manager
  - efficiency and utilization: we would like all resources (CPUs, GPUs, memory, disk, bandwidth) to be all used as much as possible, and minimize gaps in scheduling between jobs
  - minimize turnaround for your jobs
- Submit jobs to the scheduler when you have a calculation to run; can specify:
  - walltime: maximum length of time your job will take to run
  - number of CPU cores, perhaps distribution across nodes
  - memory (per core or total)
  - if applicable, number of GPUs
  - Slurm partition, reservation, software licenses, ...
- Your job is automatically started by the scheduler when resources are available
  - standard output and error go to file(s)

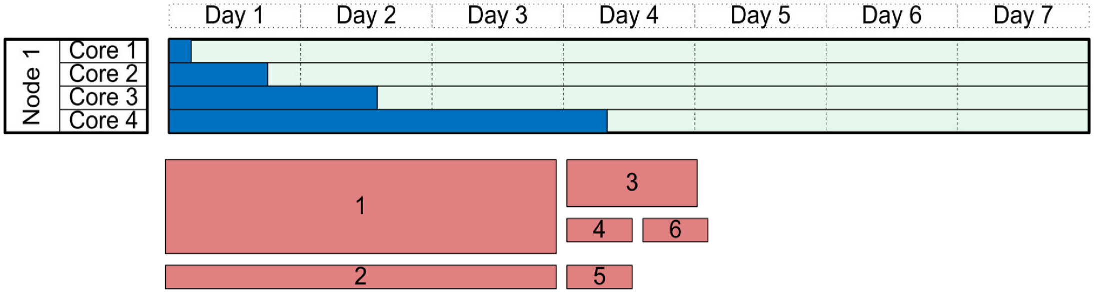
Fairshare mechanism

Allocation based on your previous usage and your *"share"* of the cluster

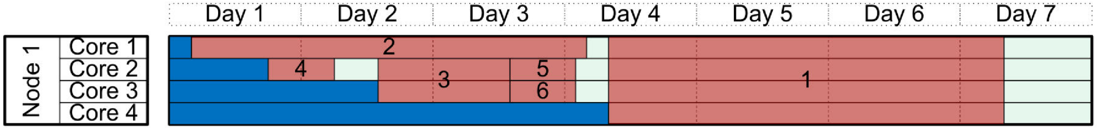
- *Priority*: one per research group (not per user!), ranges from 6 (= high, default) to 0 (= low)
- Each group has a *share target*
  - for regular queues: *share* $\propto$ the number of group members
  - in RAC: *share* $\propto$ the awarded allocation (important projects get a larger allocation)
- If a research group has used more than its *share* during a specific interval (typically 7-10 days) $\Rightarrow$ its priority will go down, and vice versa
  - the exact formula for computing priority is quite complex and includes some adjustable weights and optionally other factors, e.g., how long a job has been sitting in the queue
  - no usage during during the current fairshare interval $\Rightarrow$ recover back to level 6
- Higher priority level can be used to create short-term bursts
- Reservations (specific nodes) typically only for special events

Job packing: simplified view

- Consider a cluster with 4 running jobs and 6 newly submitted jobs
- Scheduled jobs are arranged in order of their users' priority, starting from the top of the priority list (jobs from users with the highest priority)
- Consider 2D view: cores and time
- In reality a multi-dimensional rectangle to fit on a cluster partition: add memory (3rd dimension), perhaps GPUs (4th dimension), and so on, but let's ignore these for simplicity



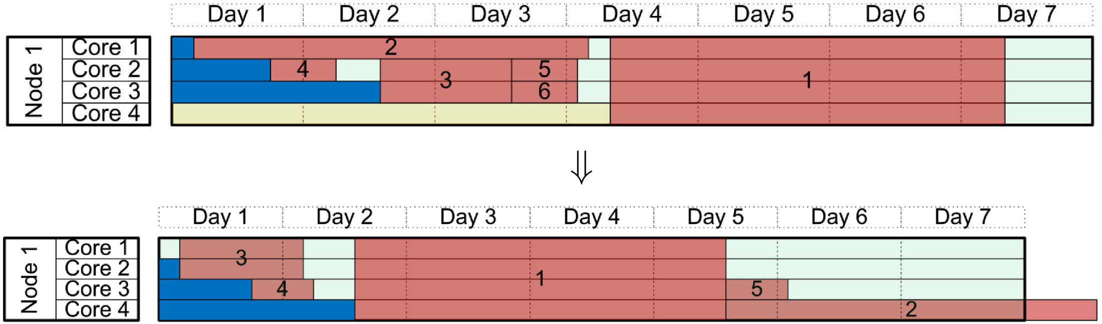
Jobs are scheduled in order of their priority. Highest-priority job may not run first!



- *Backfill*: small lower-priority jobs can run on processors reserved for larger higher-priority jobs (that are still accumulating resources), if they can complete before the higher-priority job begins

Why does my job's start time estimate keep moving into the future?

- If a running job finishes early, or a waiting job is canceled, or a new higher priority job is added to the queue $\Rightarrow$ all waiting jobs are rescheduled from scratch in the next cycle, again in the order of their priority
- This will change the estimated start time, and not always in the obvious direction ... (#2)



Job billing by core-equivalent

Goes into determining your priority

- Recall: base nodes have 128GB and 32 cores per node $\Rightarrow$ effectively 4GB per core
- Job billing is by core and memory (via core-equivalents: 4GB = 1 core), whichever is larger
  - this is fair: large-memory jobs use more resources
  - a 6GB serial job running for one full day will be billed 36 core-hours
- On GPU partitions job billing is by GPU-equivalents (the largest of GPUs / cores / memory)

Submitting a simple serial job

```
#!/bin/bash
echo This script is running on
hostname
sleep 20
```

```
$ sbatch [--account=def-razoumov-ac, other flags] simpleScript.sh
$ squeue -u yourUsername [-t RUNNING] [-t PENDING]     # list your jobs
$ scancel jobID              # cancel a scheduled or running job
$ scancel -u yourUsername   # cancel all your jobs
```

- The flag `--account=...` is needed only if you've been added to more than one CPU allocation (RAS / RAC / reservations)
  - used for "billing" purposes
  - different from your cluster account!
- No need to specify the number of cores: one core is the default
- Submitting many serial jobs is called "serial farming" (perfect for filling in the parameter space, running Monte Carlo ensembles, etc.)
- Common job states: R = running, PD = pending, CG = completing right now, F = failed

Customising your serial job

```
$ icc pi.c -o serial
$ sbatch [other flags] job_serial.sh
$ squeue -u username [-t RUNNING] [-t PENDING]       # list all current jobs
$ sacct -j jobID [--format=jobid,maxrss,elapsed]    # list resources used by
                                                     # your completed job
```

```
#!/bin/bash
#SBATCH --time=00:05:00   # walltime in d-hh:mm or hh:mm:ss format
#SBATCH --job-name="quick test"
#SBATCH --mem=100    # 100M
#SBATCH --account=def-razoumov-ac
./serial
```

- The flag `--account=...` is needed only if you've been added to more than one CPU allocation (RAS / RAC / reservations)
  - used for "billing" purposes
  - different from your cluster account!
- It is good practice to put all flags into a job script (and not the command line)
- Could specify number of other flags

Submitting array jobs

- Job arrays are a handy tool for submitting many serial jobs that have the same executable and might differ only by the input they are receiving through a file
- Job arrays are preferred as they don't require as much computation by the scheduling system to schedule, since they are evaluated as a group instead of individually
- In the example below we want to run 30 times the executable "myprogram" that requires an input file; these files are called input1.dat, input2.dat, ..., input30.dat, respectively

```
$ sbatch [other flags] job_array.sh
```

```
#!/bin/bash
#SBATCH --array=1-30          # 30 jobs
#SBATCH --job-name=myprog     # single job name for the array
#SBATCH --time=02:00:00       # maximum walltime per job
#SBATCH --mem=100             # maximum 100M per job
#SBATCH --account=def-razoumov-ac
#SBATCH --output=myprog%A%a.out   # standard output
#SBATCH --error=myprog%A%a.err   # standard error
# in the previous two lines %A" is replaced by jobID and "%a" with the array index
./myprogram input$SLURM_ARRAY_TASK_ID.dat
```

Submitting OpenMP or threaded jobs

```
$ icc -qopenmp sharedPi.c -o openmp
$ sbatch [other flags] job_openmp.sh
$ squeue -u username [-t RUNNING] [-t PENDING]      # list all current jobs
$ sacct -j jobID [--format=jobid,maxrss,elapsed]    # list resources used by
                                                    # your completed job
```

```
#!/bin/bash
#SBATCH --cpus-per-task=4   # number of cores
#SBATCH --time=0-00:05      # walltime in d-hh:mm or hh:mm:ss format
#SBATCH --mem=100           # 100M for the whole job (all threads)
#SBATCH --account=def-razoumov-ac
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK   # passed to the program
echo running on $SLURM_CPUS_PER_TASK cores
./openmp
```

- All threads are part of the same process, share same memory address space
- OpenMP is one of the easiest methods of parallel programming
- Scaling only to a single node

Submitting MPI jobs

```
$ mpicc distributedPi.c -o mpi
$ sbatch [other flags] job_mpi.sh
$ squeue -u username [-t RUNNING] [-t PENDING]     # list all current jobs
$ sacct -j jobID [--format=jobid,maxrss,elapsed]   # list resources used by
                                                   # your completed job
```

```
#!/bin/bash
#SBATCH --ntasks=4           # number of MPI processes
#SBATCH --time=0-00:05       # walltime in d-hh:mm or hh:mm:ss format
#SBATCH --mem-per-cpu=100 # in MB
#SBATCH --account=def-razoumov-ac
srun ./mpi                   # or mpirun
```

- Distributed memory: each process uses its own memory address space
- Communication via messages
- More difficult to write MPI-parallel code than OpenMP
- Can scale to much larger number of processors, across many nodes

Scheduler: submitting GPU jobs

```
#!/bin/bash
#SBATCH --nodes=3          # number of nodes
#SBATCH --gres=gpu:1       # GPUs per node
#SBATCH --mem=4000M        # memory per node
#SBATCH --time=0-05:00     # walltime in d-hh:mm or hh:mm:ss format
#SBATCH --output=%N-%j.out    # %N for node name, %j for jobID
#SBATCH --account=def-razoumov-ac
srun ./gpu_program
```

Scheduler: interactive jobs

```
$ salloc --time=1:0:0 --ntasks=2   # submit a 2-core interactive job for 1h
$ echo $SLURM_...    # can check out Slurm environment variables
$ ./serial      # this would be a waste: we have allocated 2 cores
$ srun ./mpi    # run an MPI code, could also use mpirun/mpiexec
$ exit    # terminate the job
```

- Should automatically go to one of the Slurm interactive partitions

```
$ sinfo -a | grep interac
```

- Useful for debugging or for any interactive work, e.g., GUI visualization
  - interactive CPU-based ParaView client-server visualization on Cedar and Graham `https://docs.computecanada.ca/wiki/Visualization`
  - we use `salloc` in our hands-on Chapel course
- Make sure to only run the job on the processors assigned to your job – this will happen automatically if you use srun, but not if you just ssh from the headnode

Slurm jobs and memory

It is very important to specify memory correctly!

- If you don't ask for enough, and your job uses more, your job will be killed
- If you ask for too much, it will take a much longer time to schedule a job, and you will be wasting resources
- If you ask for more memory than is available on the cluster your job will never run; the scheduling system will not stop you from submitting such a job or even warn you
  - always ask for slightly less than total memory on a node as some memory is used for the OS, and your job will not start until enough memory is available
- Can use either `#SBATCH --mem=4000` or `#SBATCH --mem-per-cpu=2000`
- What's the best way to find your code's memory usage?

Slurm jobs and memory (cont.)

- Second best way: use Slurm command to estimate your completed code's memory usage

```
$ sacct -j jobID [--format=jobid,maxrss,elapsed]
                        # list resources used by a completed job
```

- Use the measured value with a bit of a cushion, maybe 15-20%
- Be aware of the discrete polling nature of Slurm's measurements
  - sampling at equal time intervals might not always catch spikes in memory usage
  - sometimes you'll see that your running process is killed by the Linux kernel (via kernel's *cgroups* `https://en.wikipedia.org/wiki/Cgroups`) since it has exceeded its memory limit but Slurm did not poll the process at the right time to see the spike in usage that caused the kernel to kill the process, and reports lower memory usage
  - sometimes `sacct` output in the memory field will be empty, as Slurm has not had time to poll the job (job ran too fast)

Why is my job *not* running?

In no particular order:

(1) Other jobs have greater priority

(2) There is a problem with the job / resources are not available

- resources do not exist on the cluster?
- did not allow for OS memory (whole-node runs)?

(3) The job is blocked

- disk quota / other policy violation?
- dependency not met?

(4) There is a problem with the scheduling system or cluster

- most frequent: someone just submitted 10,000 jobs via a script, then cancelled them, then resubmitted them, rendering Slurm unresponsive for a while, even if resources are available for your job right now

Best practices

Best practices: computing

- Production runs: only on compute nodes via the scheduler
  - do not run anything intensive on login nodes or directly on compute nodes
- Only request resources (memory, running time) needed
  - with a bit of a cushion, maybe 115-120% of the measured values
  - use Slurm command to estimate your completed code's memory usage
- Test before scaling, especially parallel scaling
- For faster turnaround, request whole nodes (`--nodes=...`) and short runtimes (`--time=...`)
  - recall: by-node jobs can run on the "entire cluster" partitions, as opposed to smaller partitions for longer and "by-core" jobs
- Do not run unoptimized codes (use compilation flags -O2 or -O3 if needed)
- Be smart in your programming language choice, use precompiled libraries

Best practices: file systems

Filesystems in CC are a shared resource and should be used responsibly!

- Do not store millions of small files
  - organize your code's output
  - use tar or even better dar (`http://dar.linux.free.fr`, supports indexing, differential archives, encryption)
- Do not store large data as ASCII (anything bigger than a few MB): waste of disk space and bandwidth
  - use a binary format
  - use scientific data formats (NetCDF, HDF5, etc.): portability, headers, binary, compression, parallel
  - compress your files
- Use the right filesystem
- Learn and use parallel I/O
- If searching inside a file, might want to read it first
- Have a backup plan
- Regularly clean up your data in $SCRATCH, $PROJECT, possibly archive elsewhere

Summary

Documentation and getting help

- Official documentation `https://docs.computecanada.ca/wiki`
- WestGrid training materials `https://westgrid.github.io/trainingMaterials`
- Compute Canada *Getting started* videos `http://bit.ly/2sxGO33`
- Compute Canada YouTube channel `http://bit.ly/2ws0JDC`
- Email **support@computecanada.ca** (goes to the ticketing system)
  - try to include your full name, CC username, institution, the cluster name, copy and paste as much detail as you can (error messages, jobID, job script, software version)
- Please get to know your local support
  - difficult problems are best dealt face-to-face
  - might be best for new users