Scientific Computing in practice

Kickstart 2015 (cont.)

Ivan Degtyarenko, Janne Blomqvist, Mikko Hakala, Simo Tuomisto

School of Science, Aalto University

June 1, 2015

Aalto University
School of Science

Triton practicalities

From now on you will need your laptop

- Access & help
- Login, environment, disk space
- Code compilation
- Running jobs: serial, parallel
- GPU runs
- SLURM advances

Accessing Triton

```
$ ssh AALTO_LOGIN@triton.aalto.fi
```

- Requires valid Aalto account
- Directly reachable from:
  - Aalto net:
    - department workstations
    - wired visitor networks, wireless Aalto, Aalto Open and Eduroam at Aalto
  - CSC servers
- Outside of Aalto:
  - Must hop through Aalto shell servers; first ssh to amor.becs.hut.fi, james.ics.hut.fi, hugo.physics.aalto.fi, kosh.aalto.fi, etc.
- Aalto account is used to login, but for getting access one should contact local Triton support member first and ask to grant the access

Best practice: SSH key

On your workstation where from you want to login to Triton:

```
$ ssh-keygen
```

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub
triton.aalto.fi
```

```
$ ssh triton.aalto.fi
```

for a sake of security / convenience

SSH key must have a secure passphrase!

Triton session

```
ssh triton.aalto.fi
cd $WRKDIR
```

A few commands to get started

```
quota
account
squeue
```

Aalto University
School of Science

Frontend node: intended usage

Frontend triton.aalto.fi - just one of the computers out of others adapted for server needs (Xeon E5-2695 v2, 24x cores with 256G of memory)

- File editing
- File transfers
- Code compilation
- Job control
- Debugging
- Checking results

- No multi-CPU loads
- No multi-GB datasets into memory
- Matlab, R, IPython sessions otherwise OK

Aalto University
School of Science

Getting help

- points of search for help:
  - at wiki.aalto.fi look for Triton User Guide
    - access requires valid Aalto account
    - see also FAQ over there
  - follow triton-users@list.aalto.fi list: all users MUST BE there
  - tracker.triton.aalto.fi: for any issue; see whether it has been published already, if not, then shoot
    - Accessible from Aalto net only, from outside run proxy through your department (hugo, james, amor, … etc)
  - local support team member: for account, quota

Triton cluster report

- Ganglia monitoring tool available for all FGI resources, here is Triton's link: http://pulse.fgi.csc.fi/~tigerste/gweb/?s=by+name&c=Triton
- Shows overall cluster utilization, CPU usage, memory usage, local disk space usage, GPU utilization etc.
- Available to everyone

Triton nodes naming scheme

- Labeled after real label on each phyiscal node
  - Compute nodes: cn01, cn02, cn03 … cn488, ivy01 … ivy48
  - Same for gpu nodes: gpu001, gpu002 … gpu019
  - Fat nodes: fn01, fn02
  - Mostly administrative purpose nodes: tb01, … tb08
- Listing: cn[225-248] or gpu[001-002]
- Opterons: cn[01-112], cn[249-488]
- Xeons Westmere: cn[113-248]
- Xeons Ivy Bridge: ivy[01-48]
- This naming scheme is used by SLURM commands, at any place one needs a “nodelist”
  - `squeue -w gpu[001-019]`

Exercise: kickstart

10 minutes to proceed

Use wiki, Ganglia, command line (slurm, squeue, etc) … and Google

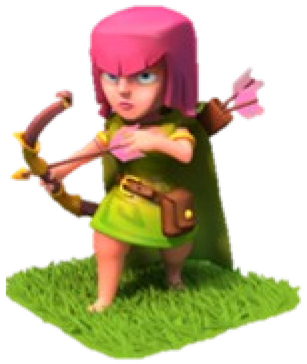
- Login to triton.aalto.fi, cd $WRKDIR (if you use the course account, create a directory for yourself)
- find out next: absolute path to your $WRKDIR, your account expiration date, your current quota at $WRKDIR and $HOME, your groups membership, for how long the system is up and how many users are logged in right now
- find out next: how big is the cluster: amount of CPUs of each type, memory on different nodes, amount of GPU cards and their type
- find out next: default version of GNU compilers (C/Fortran), default version for Python
- check out how busy is the cluster: number of pending jobs, number of running jobs
- Find out CPU type and memory amount on the frontend. Try to login to any of the compute node ('ssh cn01' for instance). Succeed? Why not?
- What is the longest run possible on the cluster? How many CPUs one can use at once and for how long? How many idling gpu nodes?

Transferring files

- Network share (NBE, CS)
  - /triton/ filesystem mounted on workstations
- SSHFS
  - Mount remote directories over SSH
- SCP/SFTP
  - Copy individual files and directories (inefficiently)

    ```
    $ scp file.txt triton:file.txt
    ```
- Rsync over SSH
  - Like `scp`, but avoids copying existing files again
  - `$ rsync -auv --no-group source/ triton:target/`

Software: modules environment



```
$ python3
-bash: python3: command not found
$ module load python3
$ python3
Python3.3.1 (default, Apr 8 2013, 16:11:51)
...
```

- **`module avail`** : compilers, libraries, interpreters and other software
- Maintained by Triton admins or FGI partners (/cvmfs)
- **`module load`** : adds installed programs and libraries to $PATH, $LD_LIBRARY_PATH, etc, sets up required environment variables for the lifetime of current shell or job

•

Aalto University
School of Science

`module` subcommands

- `module help` or `man module`
- `module list`
- `module purge`
- `module avail`
- `module whatis python`
- `module unload python`
- `module load openmpi/1.6.5-intel`
- `module swap openmpi/1.8.1-intel`
- `module show intel`

Compilers: GNU

- Default complier for Linux: `gcc -v`
- Using GNU C compiler

```
$ gcc -O2 -Wall hello.c -o hello
$ ./hello
Hello world!
```

- C++: `g++`
- Fortran: `gfortran`

Compilers: Intel

- Intel compilers C, C++, Fortran available through `module load intel`
- icc, icpc, ifort
- Example: ***`ifort hello.f90 -o hello`***
- Comes with MKL libs (math, mpi)
- Intel or GNU? both are valid, choose one that performs better while still producing correct results
- More on compiling/debugging optimizing tomorrow

Exercise: Module

10 minutes to proceed

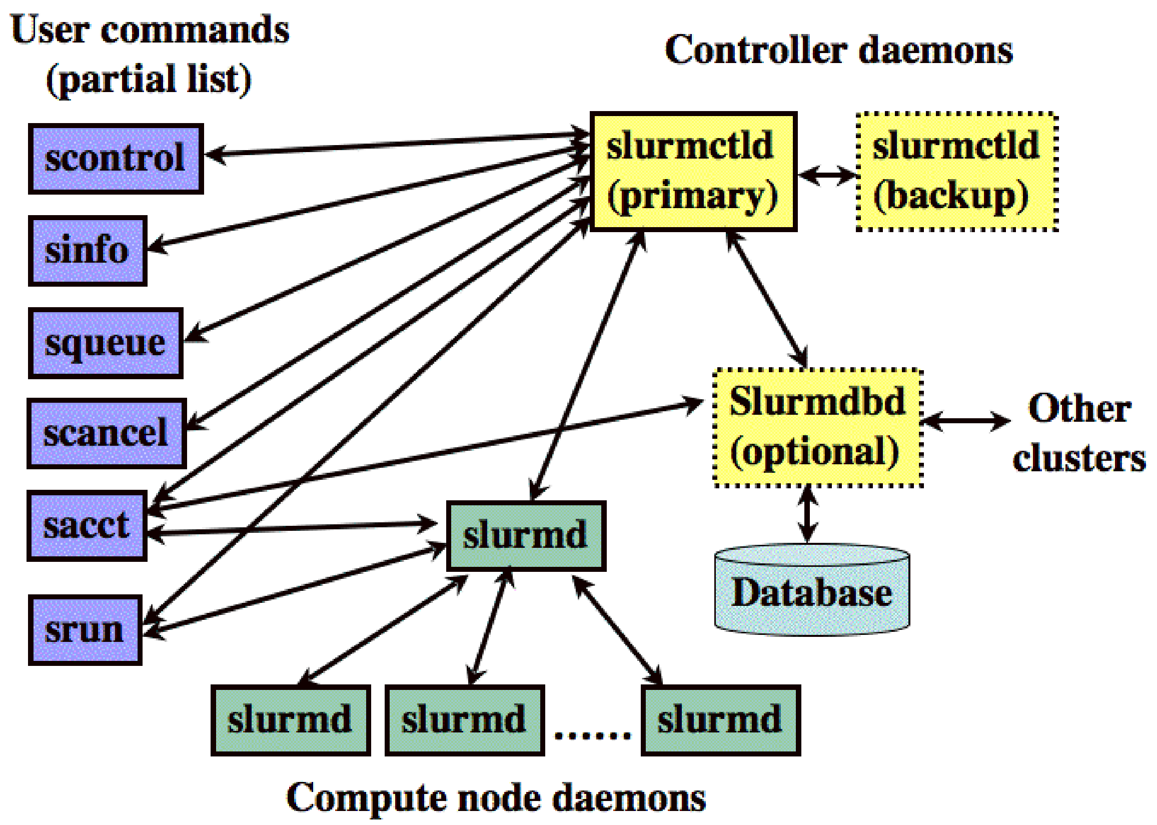
Investigate **`module`**'s subcommands (avail, load, list, purge, swap, unload, show, whatis)

- Find out Intel compilers versions available
- Load the latest Intel module, plus corresponding OpenMPI, load the latest LAMMPS. Check what you have loaded. Swap the latest OpenMPI module with the openmpi/1.6.5-intel.
- Investigate available Python versions. What environment variables it sets? Purge all your current modules loaded.
- Investigate available MATLAB versions, load the latest one.
- Add the latest Intel module to your .bashrc so that it would load automatically every time you login
- Mount your triton's work directory to your local dir on the laptop with SSHFS. (If you have Linux/Mac see sshfs, on Windows find out the way to do it)

Batch computing in HPC

- A master controller manages a set of compute nodes.
- Users submit jobs to the controller, which allocates resources from the compute nodes.
- Jobs are generally non-interactive (I/O from files, no GUI).

Batch computing system



Aalto University
School of Science

What is a 'job'?

Job = Your program + handling instructions

- Shell script that runs your program
  - Single command or complex piece of programming
- Allocation size
  - (Nodes or CPUs) x Memory x Time
- Constraints
- Submitted in a single script (.sh) file from the front node or through Grid interface

Job terminology

- Job – top level job “container”
- Array job – a set of (near identical) jobs
- Job step – Each job consists of a number of job steps, typically launched with “srun” inside the job batch script.
- Job task – Each job step consist of a number of tasks running in parallel.

The job scheduler: SLURM

- Basic units of resources:
  - Nodes / CPU cores
  - Memory
  - Time
  - GPU cards / harddrives
- Takes in cluster jobs from users
- Compares user-specified requirements to resources available on compute nodes
- Starts jobs on available machine(s)
- Individual computers and the variety of hardware configurations (mostly) abstracted away
- On Triton we have a **`/etc/slurm/job_submit.lua`** script that selects right QOS and partition based on the user-defined time/mem/CPU job requirements

SLURM (cont.)

- Tracks historical usage (walltime etc; **`sacct`**)
  - ...to enable hierarchical fair-share scheduling
- Jobs sorted in pending queue by priority (**`sprio`**)
  - Computed from user's historical usage (fair-share), job age (waiting time), and service class (QOS)
- Highest priority jobs are started on compute nodes when resources become available
- Using cluster time consumes fairshare, lowering priority of future jobs for that user & department

SLURM batch script

- To submit your program for execution with **`sbatch`**

```
#!/bin/sh
#SBATCH --time=5:00
#SBATCH --mem-per-cpu=10
srun /bin/echo hello world
```

- Submiting: `sbatch -p play hello.slrm`
- The batch script is a shell script with normal shell commands accompanied by SLURM instructions. Can be hundreds of shell scripting lines if needed.
- There is no need to define partition and QOS in general, SLURM automagically takes care about them based on the time/mem/CPU info you specify (!)

Job instructions

Job instructions are special comments in batch script, with them you are saying to SLURM your needs: how long you run, how much memory and CPUs you require, what should be the name and where should go the output etc (see **`man sbatch`** for details). For instance:

```
#SBATCH --job-name=my_name
```

```
#SBATCH --output=my_name.%j.out
```

```
#SBATCH --constraint=opteron
```

```
#SBATCH --mem-per-cpu=x
```

- Memory requirement per allocated CPU core, attempting to use more results in job termination. By default, X is in MB; with 'G' suffix in GB

```
#SBATCH —-time=d-hh:mm
```

- Job is killed after timelimit + some slack.

Important: job time limit

Always use estimated time option (!):

`--time=days-hours:minutes:seconds`

Three and half days: `--time=3-12`

One hour: `--time=1:00:00`

30 minutes: `--time=30`

One day, two hours and 27 minutes: `--time=1-2:27:00`

Otherwise: the default time limit is the partition’s default time limit

By default, the longest runtime is 5 days, the longer runs are possible but user must be explicitly added to the 'long_runs_list'

Important: job memory limit

- Always specify how much memory your job needs.
- **`--mem-per-cpu=<MB>`**
- How to figure out how much memory is needed?
  - `top` on your workstation, look for RSS column
  - `/proc/<pid>/status` or similar on your workstation, see VmHWM field.
  - `sacct -l -j <jobid> | less -S` for a completed job and `sstat -j <jobid>` for running. Check MaxRSS column. Fix your jobscript, and iterate.
    - Note: MaxRSS is sampled, might be inaccurate for very short jobs!
    - Note 2: For parallel jobs, only works properly if “srun” was used to launch every job step (NOT mpirun).

After job submission

- Job gets a unique jobID, from now on it is registered and whatever happens to the job it is already in the “history”
- Job waits in PENDING state until the resource manager finds available resources on matching node(s)
- Job script is executed on the nodes assigned to it; the requested memory and CPUs are reserved for it for a certain time
  - When job is running, user has access to the node(s) her/his job is running on, otherwise SSH access to compute nodes is restricted
- Program output saved to '.out' and '.err' files

Pending reasons: why does not run?

Look at NODELIST(REASON) field

- *`(Priority)`*: Other jobs have higher priority than your job.
- *`(Resources)`*: Your job has enough priority to run, but there aren't enough free resources.
- *`(ReqNodeNotAvail)`*: You request something that is not available. Check memory requirements per CPU, CPUs per node. Possibly time limit is the issue. Could be that due to scheduled maintenance break, all nodes are reserved and thus your -t parameter can't be larger than time left till the break.
- *`(QOSResourceLimit)`*: Your job exceeds the QOS limits. The QOS limits include wall time, number of jobs a user can have running at once, number of nodes a user can use at once, etc. This may or may no be a permanent status. If your job requests a wall time greater than what is allowed or exceeds the limit on the number of nodes a single job can use, this status will be permanent. However, your job may be in this status if you currently have jobs running and the total number of jobs running or aggregate node usage is at your limits. In this case, jobs in this state will become eligible when your existing jobs finish.
- *`(AssociationResourceLimit)`*: The job exceeds some limit set on the association. On triton, this in practice means the per-user GrpCPURunMins limit, which currently is 1.5M minutes per user. Wait a while for running jobs to proceed, so that new jobs may start. Also, shorter job time limits help.

Job priority

- SLURM assigns each job a 'priority'.
- When resources are available, launch highest priority jobs first.
  - Exception: backfill algorithm allows small jobs to run provided they don't change the estimated start time of the highest priority pending job.
- Hierarchical fair-share algorithm.
- `slurm prio` or `sprio -n -l -u $USER`
- Impact factors: job length, wait time, partition priority, fair share

Job allocation

- When job enters RUNNING state, resources on a set of nodes have been assigned to it
- The SLURM batch script is copied and executed on the first of the assigned nodes
- User code is now responsible for making use of the allocated resources: *in other words even if you got X number of CPUs, doesn't mean your program will run in parallel automatically*
- List of assigned nodes is exposed to the user program through environment variables: **`$SLURM_CPUS_ON_NODE`**, **`$SLURM_JOB_NODELIST`**, etc.
- In case of MPI, **`srun`** handle launching processes on correct nodes, but non-MPI programs will run on the first node only (!)

Cluster partitions

`slurm p`

In general SLURM takes care about submitting a job to a right partition (!), but there are cases when one needs to put a partition name explicitly: like 'play' or 'pbatch' queues

```
#SBATCH --partition=name
```

Partition is a group of computers, for instance

- *play* meant for <15 min test runs (to see if code runs)
- Default partition '*batch*' for serial parallel runs, mix of Xeons and Opterons
- *short* has additional nodes, for <4h jobs
  - Should use it if the task can finish in time; there are no downsides
- *hugemem* is reserved for >64 GB jobs
  - One machine with 1 TB memory and 24 slightly slower CPUs
  - Separate from 'batch' only to keep it immediately available to big jobs when specifically requested
- *pbatch* is for large parallel runs on up to 32 nodes at once

`slurm p`: partition info (i.e. **`sinfo -o "%10P %.11l %.15F %10f %N"`**)

- **`NODES(A/I/O/T)`**: Number of nodes by state in the format "allocated/idle/other/total"

Aalto University
School of Science

The sandbox: 'play' partition

Three dedicated machines for up to 15 minutes test runs and debugging: ivy48, cn01 and tb008, i.e. Opteron and two Xeons, Westmere and Ivy Bridge correspondingly

```
sbatch -p play my_job.slrm
```

Checking results

- Check output files
- Check if job finished successfully with:

```
$ slurm history
```

or

```
$ slurm history 4hours
$ slurm history 3days
```

Aalto University
School of Science

Exercise: 'Hello Triton'@SLURM

15 minutes to proceed

Run and monitor jobs with SLURM

- Pick up the batch script at `/triton/scip/kickstart/hello.slrm` and submit it with `sbatch`. Look while it is pending (`slurm q, squeue -j …, scontrol show job …`). Why it is pending? Estimated start time? Cancel the job either by jobID or by name.
- Submit to 'play' queue. Failed? Modify time (require 1 minute), and memory (10M) and submit again. Check job history (`slurm history`). Check memory usage, is it possible?
- What is the longest job one can run on the default partition? Investigate `scontrol show partition` command output.
- Pick up something that eats more memory (`bash_mem.slrm`), run and check memory usage. Modify memory requirement according to your findings and run the job once again.
- Imitate job failing (for instance set time limit for `bash_mem.slrm` less that it really takes). For a sake of debugging:
  - Find out the exact node name your job was run on, job ID, start/end time
  - Real case: there is nothing in the standard output, what could be the reason?
- Modify `hello.slrm` to forward error and standard outputs to a new file named '<job_ID.out>'
- Modify `hello.slrm` so that it would notify you by email when job has ended or failed

Slurm utility

- Triton-specific wrapper by Tapio Leipälä for Slurm commands
  - `squeue, sinfo, scontrol, sacct, sstat ...`
- Informative commands only (safe to test out)
- Shows detailed information about jobs, queues, nodes, and job history
- Example: show current jobs' state:

```
$ slurm queue
JOBID  PARTITION NAME TIME START_TIME STATE
430816 batch     test 0:00 N/A        PENDING
```

Aalto University
School of Science

Slurm utility (cont.)

- `slurm p` →

```
sinfo "%10P %.11l %.15F %10f %N"
```

- `slurm q` →

```
squeue -S T,P,-S,-i -o "%18i %9P
%14j %.11M %.16S %.8T %R" -u $USER
```

- `slurm j <job_ID>` →

```
scontrol show job <job_ID>
```

Aalto University
School of Science

Exercise: **`slurm`**@SLURM

10 minutes to proceed

Investigate **`slurm`** utility

- List your own jobs, list full queue, list job history for any other user of your choice within one day
- Check out shares per user. Investigate **`/usr/local/bin/slurm`**, what is behind **`slurm shares`**?
- Copy **`slurm`** utility to your dir and modify so that **`slurm q`** would have list running jobs only with the estimated end time instead of start time

Aalto University
School of Science

Job steps (sequential)

- Within single batch script one can run several programs sequentially. From SLURM point of view they are job steps, smaller units of work started inside a job allocation.
- Every single 'srun' row in your slurm script is yet another step

```
srun -n 1 myprogram
srun -n 1 myprogram2
srun -n 1 myprogram3
```

- Running job can be tracked by

`$ slurm steps <jobid>` or `sstat -a -j <jobid>`

- Steps will be marked as *<jobid>.1*, *<jobid>.2*, etc
- After completion, job accounting remembers exit status and resource usage of each individual job step

```
$ slurm history
```

Aalto University
School of Science

Array jobs

- The way to submit thousands of tasks at once: they will be handled by SLURM as a single job. So called "embarrassingly parallel" jobs, i.e. fully independent runs that use the same binary but different datasets as an input
- All tasks have same limits: time, memory size etc. Supported for batch jobs only (i.e. not for interactive sessions)
- On Triton one can launch array job that will have maximum 9999 jobs: `--array=0-9999` or `--array=0-100:5` or `--array=1,3,5,7`
- Array of jobs, from SLURM point of view, is a set of individual jobs with its own array index each. Individual jobs are assigned an ID like jobID_index (for instance 1639672_1), where jobID is the first job ID of the array
- Additional environment variables that can be used in .slrm scripts:
  - `$SLURM_ARRAY_JOB_ID` the first job ID of the array
  - `$SLURM_ARRAY_TASK_ID` the job array index value
- SLURM allows to handle jobs either as a whole or by individual jobs:
  - `scancel 1639672` to cancel all array jobs at once
  - `scancel 1639672_1` to cancel only fisrt job of the array
  - `scancel 1639672_[1-3]` to cancel first three jobs only
- `squeue -r` to see all individual jobs, otherwise handled as a single record

Array job example

Submits 200 jobs, each has its own directory and the corresponding `input_file` prepared in advance

```
#!/bin/sh
#SBATCH --time=5
#SBATCH --mem-per-cpu=100
#SBATCH --array=1-200
cd run_$SLURM_ARRAY_TASK_ID
srun $WRKDIR/my_program input_file
```

Job dependencies

`--dependency=<dependency_list>` allows to postpone the job start before some other condition is met. Where **`<dependency_list>`** is of the form `<type:job_id[:job_id],type:job_id[:job_id]]>`

- Example `sbatch -dependency=after:63452 job.slrm` runs after job 63452 has been started
- Other types of dependencies:
  - `afterany:job_id[:jobid...]` begin execution after the specified jobs have terminated
  - `afternotok:`... after the specified jobs have terminated in some failed state (non-zero exit code, timed out, etc).
  - `afterok:`... after the specified jobs have successfully executed (exit code of zero)
  - `expand:job_id` resources allocated to this job should be used to expand the specified job. The job to expand must share the same QOS (Quality of Service) and partition.
  - `singleton` begin execution after any previously launched jobs sharing the same job name and user have terminated

Constraints

Limit job for running on the nodes with some specified feature(s)

```
#SBATCH --constraint=[xeon|xeonib]
```

or

```
#SBATCH --constraint=tesla2090
```

Available: opteron,xeon,xeonib,tesla*

Useful for multi-node parallel jobs, gpu jobs

Exclusive access to nodes

```
#SBATCH --exclusive
```

- Resource manager guarantees no other job runs on the same node(s)
- Terrible idea for <12 core jobs in general
- May be necessary for timing cache-sensitive code
- Don't use it unless you need it; constraints only make job less likely to find available resources
- Makes sense for parallel jobs: moving data over the network is orders of magnitude slower than interprocess communication inside a single host
- Resources are wasted if `--ntasks` is not a multiple of 12 (or 20 for ivy[*] nodes)

Exercise: array jobs@SLURM

15 minutes to proceed

Array jobs, job steps, dependencies, constraints with SLURM

- Run any two jobs but so that the second one would start only if the first one has finished successfully.
- Find out the difference between ***`--constraint=xeon|xeonib|opteron`*** and ***`--constraint=[xeon|xeonib|opteron]`***
- Create on your own or copy **`/triton/scip/kickstart/array.slrm`** template for the array jobs. Run as is on the play queue. Check output.
- Modify **`array.slrm`** to run 10 jobs on 5 different opterons only, two processes per node at maximum
- Modify **`array.slrm`** so that each individual job would have a name with the job index in it, and standard output would go to the corresponding 'run_*' directory

Aalto University
School of Science

Running in parallel on Triton

- Means using more that one CPU core to solve the problem
- Either across the nodes with MPI or within one node as multithreaded (i.e. OpenMP)
- Using `--ntasks=#` (or just `-n #`) allocate a # number of CPU cores, though alone does not guarantee job reservation on *one* computer, though SLURM will try hard to place them on the same node(s)
- To get them to run on the same node(s) specify number of nodes (a multiple of 12 or 20 if you go for xeonib nodes only):

```
#SBATCH --ntasks=24

#SBATCH --nodes=2
```

Aalto University
School of Science

Multithreaded job

```
#!/bin/sh
#SBATCH -time=2:00:00
#SBATCH --mem=1G
#SBATCH --cpus-per-task=12
export OMP_PROC_BIND=true
srun openmp_program
```

We will be talking more about it on Day #3

MPI job

```
#!/bin/sh
#SBATCH --time=30
#SBATCH --mem-per-cpu=1G
#SBATCH --nodes=4
#SBATCH --ntasks=48
module load openmpi
srun --mpi=pmi2 mpi_program
```

MPI flavors on Triton

- Several versions available through `module`
  - OpenMPI
  - MVAPICH2
- Try both, your software may benefit from a particular flavor
- Compiled against particular compiler suit: GCC or Intel, Intel also provides its own
- Parallel math libs: BLACS and ScaLAPACK
- More about MPI tomorrow

Scalability

Refers to ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors. Factors that contribute to scalability include:

- Hardware: particularly memory-cpu bandwidths and network communications
- Application algorithm
- Parallel overhead related
- Characteristics of your specific application and coding

In general, ideal scalability factor is 2, and good is about 1.5, that is your program runs 1.5 times faster when you double the number of CPU cores

--gres option

- `#SBATCH --gres=`*`name:X`* instructs SLURM to find out a node for you with the specific resources, where 'name' is the resource name and 'X' is a number
- Available resources (in addition to CPUs and memory) on Triton:
  - gpu
  - spindle (aka # of harddrives)

```
$ sinfo -o '%.48N %.9F %.15G'
                                        NODELIST NODES(A/I            GRES
               cn[129-224],ivy[01-48],tb[003,005-008] 141/2/6/1       spindle:2
          cn[01-18,20-64,68-69,71-112,225-362,365-488] 232/108/2       spindle:1
                                        cn[113-128] 16/0/0/16       spindle:4
                                              fn02   1/0/0/1       spindle:6
                                       gpu[002-011]  9/0/1/10 spindle:2,gpu:2
                                       gpu[012-019]   7/0/1/8 spindle:1,gpu:2
```

Using GPUs

- gpu001 node for playing/compiling/debugging, others gpu[002-019] for production runs; gpu[001-011] and gpu[017-019] are Teslas 2090 and 2070 with 6G of video memory, others gpu[012-016] are Tesla 2050 with 3G of memory
- gpu[] nodes are the same Xeon nodes with the GPU cards installed and dedicated to GPU computing only – see 'gpu' and 'gpushort' partitions; can run for up to 30 days
- **`module load cuda`**
- **`sbatch gpu_job.slrm`**

```
…
## require GPUs, could be 1 or 2
#SBATCH --gres=gpu:1
## optional, require Tesla 2090 only
#SBATCH --constraint=tesla2090
module load cuda
srun --gres=gpu:1  my_gpu_program
```

- See Triton User Guide for more examples and details

Interactive login

- We strongly recommend to adapt your workflow for normal batch jobs, but in case you need to run some interactively, there are a few ways to do it
- Request one hour on a node:

```
sinteractive -t 1:00
```

  - You will receive a normal shell allocated for you with one CPU core and 1G of memory.
  - `sinteractive` is not a slurm command but a bash script at `/usr/local/bin/sinteractive`
  - Runs on the dedicated partition 'interactive'; from SLURM point of view a normal job in RUNNING stage, always has a name `_interactive`
  - To finish the job, just exit the shell

Interactive runs

- `salloc` … does the same as sbatch but interactively, i.e. allocates resources, runs user's program and opens the session that accepts user's input. When program is complete, resource allocation is over.
- `salloc --time=120:00 --mem-per-cpu=2500 srun my_program`
- On other hand, one can just open a session and then use **`srun`** to start the program, every single srun considered by SLURM as yet another job step

```
triton$ salloc --time=120:00 --mem-per-cpu=2500 --nodes 1 --ntasks 4
salloc: Pending job allocation 929194
salloc: job 929194 queued and waiting for resources
salloc: job 929194 has been allocated resources
salloc: Granted job allocation 929194
triton$ srun hostname
cn01
triton$ srun /path/to/executable --arguments
...
```

- Usage of **`srun`** is obligatory (!), otherwise you will run on the front-end
- Most of the options valid for sbatch can be used with salloc as well

Aalto University
School of Science

Exercise: SLURM's advances

20 minutes to proceed, use wiki to solve

SLURM advances: running multithreaded and MPI job, interactive jobs/logins, using GPUs

- Start `sinteractive` session with no options, find out sessions's end time, what is the default time?
- 'GPU computing' page at Triton User Guide has an example of running deviceQuery utility on a gpu node. Try it with salloc.
- Copy multithreaded hello_omp and hello_omp.slrm. Make sure that you require 5 minutes, 50M of memory and run 4 threads only. Run with sbatch.
- Copy MPI version of hello_mpi, make you own (consult slides) .slrm file, adapt and run with salloc as is. Library missing? Load openmpi/1.8.5-java first and try again.
- Try compiling with GCC and running hello_[mpi|omp].c on your own. For really advanced participants, do the same with the Intel compiler.

Matlab jobs

Aalto University
School of Science

Matlab “environments” on Triton

Interactive use on `fn01` (currently 1 machine)

```
triton$ ssh -XY fn01
fn01$ module load matlab; matlab &
```

- 24 CPUs and 1TB memory shared by a number of users

Noninteractive Slurm job (unlimited jobs)

- Matlab started from a user-submitted job script
- Licensing terms changed in 2013; use of Matlab in cluster and grid environments is no longer prohibited

Matlab Distributed Computing Server (128 workers)

- Special license that extends Parallel Computing Toolbox
- Mainly useful for actual parallel applications
- Tight integration between the interactive Matlab GUI and cluster

Choosing the Matlab environment

Slurm job + Matlab script

- Unconstrained license
- Batch job without GUI
- Simple and efficient way to run serial jobs
- Best use case a series of single-threaded jobs

Parallel Toolbox + MDCS

- Licensed for only 128 cores
- MDCS allows large parallel jobs across multiple nodes
- Jobs controlled from interactive Matlab GUI
- Supports data-parallel SPMD paradigm
- Parallel Toolbox brings some programming overhead
- Learning curve

Matlab on Triton

Primarily for serial runs use Slurm directly. A batch job without graphical window.

1) Prepare a M-file for the job

2) Prepare slurm submission script (example on right)

3) Submit

```
#!/bin/bash

#SBATCH -t 04:00:00

#SBATCH —mem-per-cpu=1000


module load matlab

cd /triton/becs/scratch/myuname/

# execute run.m

srun matlab -nojvm -r run
```

Aalto University
School of Science

Matlab DEMO

Matlab Demo

Actual research case from Brain and mind group

- Analyzing fMRI data from actual measurements. Taking into account physiology and head movement and doing final visualization for further analysis.
- Using existing Matlab package available online

  (https://git.becs.aalto.fi/bml/bramila)
- Need to write integration for the packages. Manage
  - Input data
  - Setup proper Matlab environment (code paths)
  - Submit job(s) to the queue
  - Run things on /local, collect results to /triton

Matlab Excercise

Small existing matlab code, that solves ODE and plots the results

- Split the computation and visualization
- Write the computation as a Matlab function
- Handle input / output. Run on /local and save results to a file at /local.
- Run for Mu values of 1..9 in parallel (play queue)
- Visualize the final results