University of Manitoba

Running Jobs with SLURM: Advanced scheduling and Jobs Efficiency

May 22, 2025.

Ali Kerrache
HPC Analyst

University of Manitoba

Outline

- What are the resources needed for a particular job?
- What are the resources to optimize?
- Slurm farming using job arrays or GLOST?
- Nodes/Tasks: what to use?
- Benchmark and scalability
- Remarks and conclusions

University of Manitoba

What resources to ask for?

- ★ CPUs:
  - How many CPUs are required to run a particular job?
- ★ Memory
  - How much memory is needed to run a particular job?
- ★ Time
  - How to estimate the run time for run a particular job?
- ★ GPUs
  - Could I run my job on a GPU?

University of Manitoba

Estimating resources: CPUs

- **How to estimate the CPU resources?**
  - No direct answer: **it depends on the code**
  - **Serial code:** 1 core [--ntasks=1 --cpus-per-task=1]
  - **Threaded and OpenMP:** no more than available cores on a node [--cpus-per-task=X]
  - **MPI jobs:** can run across the nodes [--nodes=2 --ntasks-per-node=52 --mem=0].
- **Are threaded jobs very efficient?**
  - Depends on how the code is written.
  - Does not scale very well with the number of threads.
  - Run a benchmark and compare the performance and efficiency.
- **Are MPI jobs very efficient?**
  - Scale very well with the problem size.
  - Limited number of cores for small size: when using domain decomposition
  - Run a benchmark and compare the efficiency.

University of Manitoba

Estimating resources: memory

- **How to estimate the memory for my job?**
  - No direct answer: it depends on the code
  - Java applications require more memory in general
  - Hard to estimate the memory when running R, Python, Perl, …
- **To estimate the memory, run tests:**
  - Interactive job, ssh to the node and run top -u $USER {-H}
  - Start smaller and increase the memory
  - Use whole memory of the node; seff <JOBID>; then adjust for similar jobs
  - MPI jobs can aggregate more memory when increasing the number of cores
- **What are the best practices for evaluation the memory:**
  - Run tests and see how much memory is used for your jobs {seff; sacct}
  - **Do not oversubscribe the memory** since it will affect the usage and the waiting time: accounting group charged for resources reserved and not used properly.

University of Manitoba

Estimating resources: wall time

★ **How to estimate the run time for my job?**

- **No direct answer:** it depends on the job and the problem size
- See if the code can use checkpoints
- **For linear problems:** use a small set; then estimate the run time accordingly if you use more steps (extrapolate).

★ **To estimate the time, run tests:**

- Over-estimate the time for the first tests and adjust for similar jobs and problem size.

★ **What are the best practices for time used to run jobs?**

- Have a good estimation of the run time after multiple tests.
- Analyse the time used for previous successful jobs.
- Add a margin of 15 to 20 % of that time to be sure that the jobs will finish.

University of Manitoba

Program with steps: wall time

```
[~@yak ]$  seff 5080534
Job ID: 5080534
Cluster: grex
User/Group: someuser/someuser
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 01:28:33
CPU Efficiency: 99.87% of 01:28:40
core-walltime
Job Wall-clock time: 01:28:40
Memory Utilized: 274.48 MB
Memory Efficiency: 3.43% of 7.81 GB
```

- ➔ Job completed
- ➔ CPU Efficiency: 99.87%
- ➔ Wall time: 01:28:40
- ➔ Memory Utilized: 274.48 MB

Steps: 10000 (iterations)
Wall time: 1:30 to 2:00
Memory: 300 mb to 500 mb

Steps: 10000 x 10
Wall time: {1:30 to 2:00} x 10
Memory: 300 mb to 500 mb

Memory & CPU efficiencies: seff

Output from seff command for a job {OpenMP} that asked for 24 CPUs and 187 GB of memory on cedar:

Job ID: 123456789
Cluster: cedar
User/Group: someuser/someuser
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 24
CPU Utilized: 38-14:26:22
CPU Efficiency: 38.46% of 100-08:45:36 core-walltime
Job Wall-clock time: 4-04:21:54
Memory Utilized: 26.86 GB
Memory Efficiency: 14.37% of 187.00 GB

Successful job

Low CPU efficiency: 40 %
Better performance with 8 CPU

Used less memory: 15 %

billing=46,cpu=24,mem=187G,node=1

Optimization:
Better performance with 8 CPU
Memory: 4000 M per core [32 GB]

```
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem-per-cpu=4000M
```

University of Manitoba

Bundle many jobs: job array

```
#!/bin/bash
#SBATCH --account=def-someprof
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=2500M
#SBATCH --time=3-00:00:00
#SBATCH --array=0-999%10
#SBATCH --partition=genoa
# Load appropriate modules:
module load <software>/<version>
echo "Starting run at: `date`"
./my_code test${SLURM_ARRAY_TASK_ID}
echo "Program finished with exit code $? at: `date`"
```

- You have regularly named, independent datasets (test0, test1, test2, test3, …, test999) to process with a single software code
- Instead of making and submitting 1000 job scripts, a single script can be used with the **--array=0-999** option to **sbatch**
- Within the job script, $*SLURM_ARRAY_TASK_ID* can be used to pick an array element to process

  ./my_code test${SLURM_ARRAY_TASK_ID}
- When submitted, once, the script will create 1000 jobs with the index added to JobID (12345_1, … , 12345_999)
- You can use usual SLURM commands (scancel, scontrol, squeue) on either entire array or on its individual elements

University of Manitoba

Bundle many jobs: job array

- Files: n.melt-0.txt, …. In.melt-9.txt; array with 10 elements; Run a maximum of 2 at a time
- All the data in one directory: use appropriate names to avoid data overlapping

```
lmp < in.melt-${SLURM_ARRAY_TASK_ID}.txt > log_lammps_array-${SLURM_ARRAY_TASK_ID}.txt
```

- Directories: 0, …. 9; each directory has a an input file: in.melt
- Job array with 10 elements
- Run a maximum of 2 at a time
- Output in different directories: the data may have the same name.

```
cd ${SLURM_ARRAY_TASK_ID}
lmp < in.melt > log_lammps_array-${SLURM_ARRAY_TASK_ID}.txt
```

University of Manitoba

Bundle many jobs: GLOST

```
#!/bin/bash
#SBATCH --account=def-someprof
#SBATCH --ntasks=4
#SBATCH --mem-per-cpu=2500M
#SBATCH --time=1-00:00:00
#SBATCH --partition=genoa

# Load appropriate modules + glost:
module load arch/avx512  gcc/13.2.0
openmpi/4.1.6 glost/0.3.1
echo "Starting run at: `date`"
srun glost_launch list_glost_tasks.txt
echo "Program finished with exit code $? at: `date`"
```

- You have many short independent jobs (job1, job2, job3, …) to process with a single software code.
- Instead of submitting and running many jobs, a single script can be used to run these jobs as MPI job.

- List of tasks: list_glost_tasks.txt

```
job1
job2
job3
job4
–
–
job199
job200
```

University of Manitoba

Jobs optimization

Estimate the resources for your jobs:

- Number of CPUs: serial, OpenMP, MPI
- Memory: total memory or memory per core
- Run time

★ Use interactive jobs

★ Submit test jobs:

  ❖ Run a benchmark if needed {OpenMP; MPI jobs, Hybrid}

➔ Collect the stats about memory usage, wall time, .. etc.

➔ Adjust your scripts for similar jobs

How to pick a CPU partition on Grex?

Many jobs are submitted to skylake partition and using large memory: by over-subscribing the memory, many CPUs will stay idle [low usage of].

Some tips for usage optimization:

- Run tests and check the memory usage {seff}
- Adjust the memory for similar jobs
- Submit with appropriate resources {no more}.

Partitions and memory:

genoa: 27 nodes but many CPUs {5184}
serial and MPI jobs with memory per CPU around 4 GB.

skylake: 42 nodes but many CPUs {2184}
serial and MPI jobs with memory per CPU around 4 GB.

largemem: few nodes {12}, 480 CPUs
serial and MPI jobs with memory per CPU around 9 GB.

Partition	Nodes	Cores	Total	Memory	MEM/CPU
genoa	27	192	5184	750 GB	3.9 GB
largemem	12	40	480	376 GB	9.4 GB
skylake	42	52	2184	180 GB	3.5 GB

Output from: partition-list

PARTITION	CPUS(A/I/O/T)
Genoa	4225/959/0/5184
largemem	480/0/0/480
skylake	781/1455/0/2236

Skylake partition shows 781 allocated CPUs and 1455 idle CPUs. These CPUs are idle and can not run other job because all the memory was allocated to other jobs.

University of Manitoba

Thank you for your attention

Any question?

Additional slides

University of Manitoba

SLURM script: serial jobs

```
#!/bin/bash
#SBATCH --account=def-someprof
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=2500M
#SBATCH --time=1-00:00:00
#SBATCH --partition=genoa

# Load appropriate modules:
module load <dep> <software>/<version>
echo "Starting run at: `date`"
program.x [+options and arguments if any]
echo "Program finished with exit code $? at: `date`"
```

SLURM directives:

- Default: 1 core, 256mb, 3 hours
- **account**, tasks = 1, memory per core, wall time, **partition**, …
- Other: E-mail-notification, … etc.

Submit and monitor the job:

- sbatch myscript.sh
- squeue -u $USER; sq; sacct -j JOB_ID

More information:

- partition-list; sinfo --format="%20P"
- Sinfo -s; sinfo -p genoa,skylake
- squeue -p genoa,skylake -t R {PD}

University of Manitoba

SLURM script: OpenMP jobs

```
#!/bin/bash
#SBATCH --account=def-someprof
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=2000M
#SBATCH --time=1-00:00:00
#SBATCH --partition=skylake
# Load appropriate modules:
module load <software>/<version>
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
echo "Starting run at: `date`"
program.x [+options and arguments if any]
echo "Program finished with exit code $? at: `date`"
```

```
#!/bin/bash
#SBATCH --account=def-someprof
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=2000M
#SBATCH --time=1-00:00:00
#SBATCH --partition=skylake
```

```
#SBATCH --cpus-per-task=N
#SBATCH --mem=<MEM>
```

Partitions:

- genoa: N up to 192
- skylake: N up to 52
- largemem: N up to 40

University of Manitoba

SLURM script: MPI jobs

```
#!/bin/bash
#SBATCH --account=def-someprof
#SBATCH --nodes=2-4
#SBATCH --ntasks=96
#SBATCH --mem-per-cpu=1200M
#SBATCH --time=2-00:00:00
#SBATCH --partition=skylake
# Load appropriate modules:
module load arch/avx512  gcc/13.2.0  openmpi/4.1.6
lammps/2024-08-29p1
echo "Starting run at: `date`"
srun lmp -in  in.lammps
echo "Program finished with exit code $? at: `date`"
```

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=192
#SBATCH --mem=0
#SBATCH --partition=genoa
```

```
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=52
#SBATCH --mem=0
#SBATCH --partition=skylake
```

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40
#SBATCH --mem=0
#SBATCH --partition=largemem
```

University of Manitoba

SLURM script: MPI+OpenMP jobs

```
#!/bin/bash
#SBATCH --account=def-someprof
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=6
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=1200M
#SBATCH --time=3-00:00:00
#SBATCH --partition=skylake

# Load appropriate modules:
module load <software>/<version>
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
echo "Starting run at: `date`"
srun program.x [+options and arguments if any]
echo "Program finished with exit code $? at: `date`"
```

```
#SBATCH --nodes=6
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=1200M
#SBATCH --partition=genoa
```

The total memory and CPUs per node should not exceed the available resources on the nodes.

```
#SBATCH --nodes=5
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=1000M
#SBATCH --partition=skylake
```

University of Manitoba

SLURM script: GPU jobs

```
#!/bin/bash
#SBATCH --account=def-someprof
#SBATCH --gpu=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=6
#SBATCH --mem-per-cpu=4000M
#SBATCH --time=0-3:00:00
#SBATCH --partition=gpu
# Load appropriate modules:
module load <software>/<version>
echo "Starting run at: `date`"
program.x [+options and arguments if any]
echo "Program finished with exit code $? at: `date`"
```

SLURM directives:

- Default: 1 core, 256mb, 3 hours
- **account**, number of tasks, memory per core, wall time, **partition**, …
- Other: E-mail-notification, … etc.

Submit and monitor the job:

- sbatch [some options] myscript.sh
- squeue -u $USER

Partition:

- partition-list; sinfo --format="%20P"
- sinfo -p <partition name>

University of Manitoba

How to get most of the scheduler?

The key is to know what resources are available on a given HPC machine, and adjust your requests accordingly.

- ★ It is up to the users to go through the documentation and run tests, …
- ★ Know what partitions are there, and what are their limits: sinfo, …
- ★ Know about the hardware (how many CPUs per node, how much memory per CPU available, …. documentation for each cluster
- ★ Know if your code is efficient for a given set of resources: benchmarks
- ★ Know time limits and estimate runtime of your jobs:
  - comes after some trials and errors [with experience].
- ★ Make sure your application obeys the SLURM resource limits.

Demonstration: MD simulation

- Serial job
- MPI job:
  - 4; 8; 16; 32; 48; 64; 96
- Job array: parameter sweep
  - data in one directory
  - data in multiple directories
- Estimation of wall wall time
- Memory efficiency
- CPU efficiency

```
module load arch/avx512  gcc/13.2.0  openmpi/4.1.6
Module load lammps/2024-08-29p1
srun lmp -in in.melt -lof log_lammps_output.txt
```

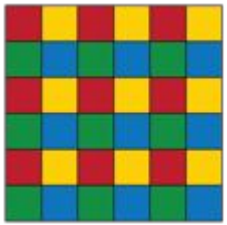
in.melt

```
# 3d Lennard-Jones melt
units           lj
atom_style      atomic
lattice         fcc 0.8442
region          box block 0 50 0 50 0 50
create_box      1 box
create_atoms    1 box
mass            1 1.0
velocity        all create 3.0 87287
pair_style      lj/cut 2.5
pair_coeff      1 1 1.0 1.0 2.5
neighbor        0.3 bin
neigh_modify    every 20 delay 0 check no
fix             1 all nve
thermo          250
run             10000
write_data  config.end_melt
```

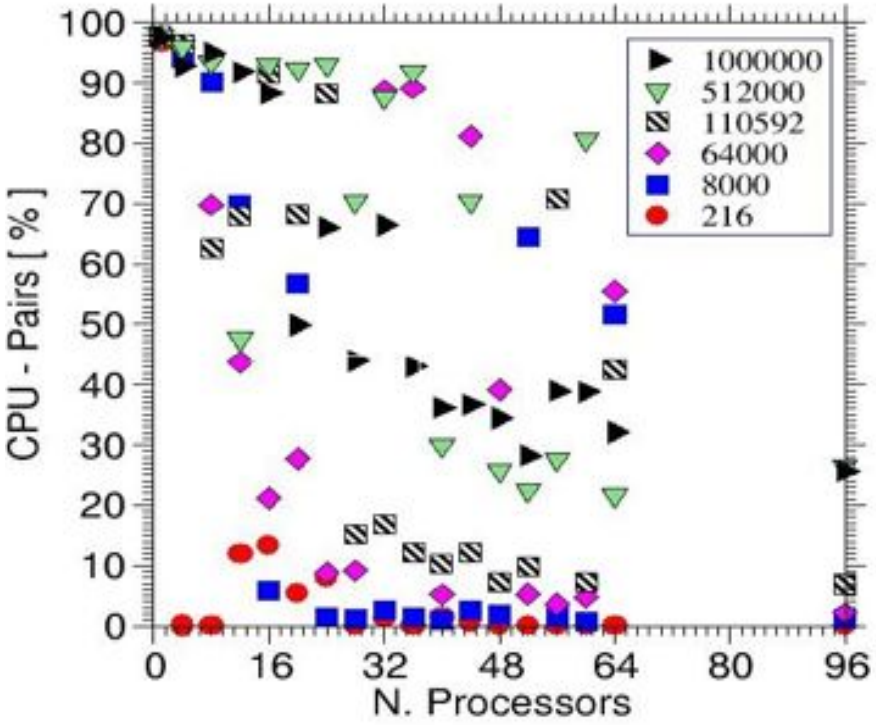
University of Manitoba

Performance tests: MD

Domain decomposition



- ★ Size, shape of the system.
- ★ Number of processors.
- ★ size of the small units.
- ★ correlation between the communications and the number of small units.
- ★ Reduce the number of cells to reduce communications.



University of Manitoba

Time breakdown

Loop time of 5316.35 on 1 procs for 10000 steps with 500000 atoms

Performance: 812.587 tau/day, 1.881 timesteps/s
99.8% CPU use with 1 MPI tasks x no OpenMP threads

MPI task timing breakdown:

Section	min time	avg time	max time	%varavg	%total
Pair	4617.5	4617.5	4617.5	0.0	86.86
Neigh	517.75	517.75	517.75	0.0	9.74
Comm	35.556	35.556	35.556	0.0	0.67
Output	0.11397	0.11397	0.11397	0.0	0.00
Modify	119.55	119.55	119.55	0.0	2.25
Other		25.83			0.49

University of Manitoba

MPI job

CPUs	CPU Efficiency	CPU time	Run time [s]	Performance tau/day	Pair Interactions	Communication time [%]
1	99.87%	5317	5317	812.587	86.86	-
4	99.76%	6200	1550	2787	84.50	3.13
8	99.09%	6312	789	5479	78.13	10.77
16	99.21%	7360	460	9388	67.74	21.35
32	98.32%	5984	187	23136	76.97	10.32
48	97.56%	5904	123	35220	74.85	12.63
64	95.17%	5888	92	47175	73.62	14.52
96	95.03%	5760	60	71874	73.24	15.16

University of Manitoba

Summary about HPC workflow

- Account and active role:
  - CCDB
- Have a look to the documentation:
  - Hardware, available tools, …
  - policies?
  - login nodes
  - storage, …
- Tools to connect and transfer files
- Access to storage: home, scratch, project
- Access to a program to use:
  - Install the program or ask for it.
  - Use the existing modules

- Test jobs:
  - Login node
  - Interactive job via salloc
- Write a job script:
  - Slurm directives
  - Modules
  - Command line to run the code
- Monitor jobs:
  - sacct; seff, optimize jobs
- Analyze data:
  - Post processing
  - Visualization