Parallel Job Orchestration with GNU Parallel

Ramses van Zon

Compute Ontario Colloquium - April 26, 2023

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Parallel Jobs

ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Motivation

Many computational research projects involve workloads of many independent, relatively short tasks.

- Genomics:
  Alignment of DNA reads
- Computational Physics/Chemistry/Material Science:
  Parameter studies of simulations
  Sampling a configuration space
- Medical imaging:
  Processing Functional MRI images
- Experimental physics:
  Matching measured gravitational wave signals with a battery of simulated waveforms.

When the number of jobs get large (think 1000s), such workloads can quickly become too large for individual workstations.

This is a common reason for researchers to move to using shared resources.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Independent jobs, that's simple, right?

- Despite being called an "embarrassingly parallel" problem, orchestrating large amounts of small computational jobs is surprisingly subtle.
- One has to ensure that these jobs are executed correctly and efficiently, and deal with cases when they are not.
- Dealing with shared resources with their own usage policies and implementations is an additional challenge.
- Many tools have been created, used for a while, and then abandoned.
- So some may write their own tool or scripts.

Here, we will focus on a tool for job orchestration that is very versatile and actively maintained, ***GNU Parallel***, and show how it helps overcome a number of challenges.

What are some of the challenges?

1. **Load balancing:**
   Balancing work across multiple cores for optimal performance if not all tasks have the same duration.
2. **Task generation:**
   Creating the commands to execute a command can be fragile and ad hoc.
3. **Fault tolerance:**
   It is essential to have a mechanism to detect and recover from failures.
4. **Scalability:**
   As the number of jobs running in parallel increases, managing them efficiently can be challenging.
5. **Monitoring and debugging:**
   Orchestrating parallel jobs requires monitoring and debugging to detect and resolve issues promptly.
6. **Data management:**
   The parallel jobs may require access to the same data, or require data movement.
7. **Dependency management:**
   Individual job may have dependencies on other jobs or resources.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Why not use SLURM?

- Most of these can **in theory** be handled by the existing SLURM job schedulers that are on the national Advanced Research Computing (ARC) systems.
- **In practice**, the latency and overhead of scheduling a single job in SLURM is often too large compared to the short duration of the job. For that reason, most ARC systems have put limits in place which may make your workflow impossible with just SLURM.
- Some automated workflow solutions that support SLURM, may nonetheless not work as they make assumptions that are in conflict with specific limitations.

Despite this, when working on the ARC systems, you will still have to use SLURM for what it is good at:

- Resource management and allocation
- Job dependencies (if you need them)
- Course grained task divisions.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

GNU Parallel

ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

GNU Parallel Features

- Command-line utility for Unix and Unix-like operating systems (Linux, MacOS)
- Versatile tool for any workflow that involves parallel processing
- Parallel execution of commands on multiple processors (cpus) or computers (nodes)
- Dynamic load balancing of the commands
- Offers job queue management, error handling, and progress monitoring
- Ideal for data processing, image manipulation, and scientific simulations

Tange, O. (2018). GNU Parallel 2018. [Online]. https://doi.org/10.5281/zenodo.1146014

https://www.gnu.org/software/parallel/parallel_tutorial.html

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

A First GNU parallel example (Niagara)

- Assume the program “mycode” has to be run with 1000 different command line arguments.
- This jobscript wants a 40-core node for 1 hour. Submit this with `sbatch`.
- Load the gnu-parallel module within your script (not necessary on GP clusters).
- The “-j 40” flag indicates you wish GNU parallel to run 40 subjobs at a time.
- Put all the commands for a given subjob onto a single line.
- Each line becomes a **sub-job**
- If you can’t fit 40 subjobs onto a node due to memory constraints, specify a different value for the “-j” flag.

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=40
#SBATCH --time=1:00:00
#SBATCH --job-name=gnuparallel40

# load modules needed
module load gcc
module load gnu-parallel

# Run the code on 40 cores (-j 40)
# Run in random order (--shuf)
# Skip empty commands (--no-run-if-empty)
parallel -j 40 --shuf --no-run-if-empty <<EOF
./mycode 1; echo "job 1 done"
./mycode 2; echo "job 2 done"
./mycode 3; echo "job 3 done"
...
./mycode 999; echo "job 999 done"
./mycode 1000; echo "job 1000 done"
EOF
```

A First GNU Parallel Example, continued

What does GNU parallel do here?

- GNU parallel assigns 40 subjobs to the cores on the node.
- As subjobs finish it assigns new subjobs to the free cores.
- It continues to assign subjobs until all subjobs in the subjob list are assigned.
- To prevent load imbalance due to correlations between the cmdline argument and the tasks' durection, we added "–shuf".
- Consequently there is built-in load balancing!

If you're running blocks of serial subjobs, just use GNU parallel!

This is only the begining, not the end!

You can use GNU parallel across multiple nodes as well, and it can log a record of each subjob, including information about subjob duration, exit status, etc...

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

A First GNU parallel example on Graham

- Assume the program "mycode" has to be run with 1000 different command line arguments.
- This jobscript wants a 32-core node for 1 hour. Submit this with `sbatch`.
- No need to load a gnu-parallel module.
- The "-j 32" flag indicates you wish GNU parallel to run 32 subjobs at a time.
- Put all the commands for a given subjob onto a single line.
- Each line becomes a **sub-job**
- If you can't fit 32 subjobs onto a node due to memory constraints, specify a different value for the "-j" flag.

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=32
#SBATCH --time=1:00:00
#SBATCH --job-name=gnu-parallelx32
#SBATCH --mem=0

# load modules needed
module load gcc

# Run the code on 32 cores (-j 32)
# Run in random order (--shuf)
# Skip empty commands (--no-run-if-empty)
parallel -j 32 --shuf --no-run-if-empty <<EOF
./mycode 1; echo "job 1 done"
./mycode 2; echo "job 2 done"
./mycode 3; echo "job 3 done"
...
./mycode 999; echo "job 999 done"
./mycode 1000; echo "job 1000 done"
EOF
```

GNU parallel on your own Linux/Mac computer

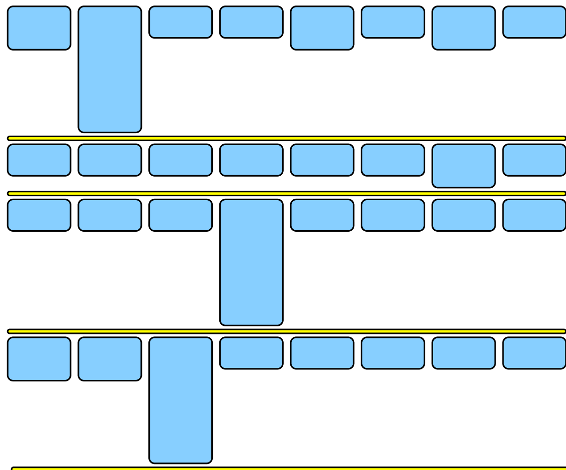
- Assume the program "mycode" has to be run with 1000 different command line arguments.
- Linux: `apt/dnf/yum install parallel`, or download from https://ftpmirror.gnu.org
- Mac: probably `sudo port install parallel` or `brew install parallel`
- The "-j" was omitted from the parallel command; parallel will then use all cores on your computer.

```
# Run the code all cores
# Run in random order (--shuf)
# Skip empty commands (--no-run-if-empty)
parallel --shuf --no-run-if-empty <<EOF
./mycode 1; echo "job 1 done"
./mycode 2; echo "job 2 done"
./mycode 3; echo "job 3 done"
...
./mycode 999; echo "job 999 done"
./mycode 1000; echo "job 1000 done"
EOF
```

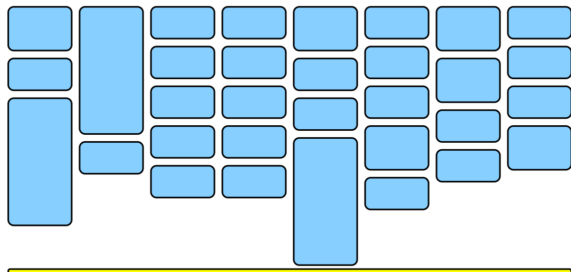
- Put all the commands for a given subjob onto a single line.
- Each line becomes a **sub-job**
- If you can't fit 4 subjobs onto a node due to memory constraints, specify a different value with the "-j" flag.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

What are the gains of load balancing?



17 hours
42% utilization



10 hours
72% utilization

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Load balancing: Done!

ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

GNU parallel, modified

Sometimes it's easiest to just create a file with a list that holds all of the subjob commands.

```
nia-login01:scratch$ cat subjobs
./mycode 1; echo "job 1 done"
./mycode 2; echo "job 2 done"
./mycode 3; echo "job 3 done"
...
./mycode 999; echo "job 999 done"
./mycode 1000; echo "job 1000 done"
nia-login01:scratch$
```

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=40
#SBATCH --time=1:00:00
#SBATCH --job-name=gnuparallel40

# load modules needed...
module load gcc
module load gnu-parallel

# Run the code on 40 cores.
parallel -j 40 --no-run-if-empty --shuf < subjobs
```

An accidental empty would yield a failing subjob. That's why we use `--no-run-if-empty` flag.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Issues with this file-based approach

- Each of the commands in that file is essentially the same, and it is tedious to maintain and error prone.
- You could write a script to generate that file.
- But there's a better way with GNU Parallel, using its replacement string features.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

GNU Parallel replacement strings

Input for GNU Parallel can be read from the command line:

```
$ parallel echo ::: A B C
```

- The command is `echo` (the colon triplet `:::` signifies the end of the command).
- After those colons come the various arguments to be substituted at the end of the command.

The output is then (order may differ as the jobs are run in parallel):

```
A
B
C
```

If the substitution should not happen at the end, you can use `{}` in the command. E.g.

```
$ parallel echo {} is a letter ::: A B C
```

gives

```
A is a letter
B is a letter
C is a letter
```

SciNet ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

GNU Parallel: a little help from seq

To use the replacement strings in our example, we would need something like:

```
parallel --shuf './mycode {}; echo "job {} done"' ::: 1 2 3 4 5 ... 1000
```

- We have to quote the command so that the linux shell does not interpret the special symbols like ";". (This is also usefull to make e.g. input/output redirection work)
- Any options like "--shuf" must be given before any argument lists.
- That is a very long line to type and maintain.

Let's use seq command instead to print a sequence of numbers.

```
$ seq 4
1
2
3
4
```

```
$ seq 2 4
2
3
4
```

```
$ seq 2 0.5 4
2.0
2.5
3.0
3.5
4.0
```

```
$ echo $(seq 2 0.5 4)
2.0 2.5 3.0 3.5 4.0
```

SciNet ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Our example, on Niagara, using replacement strings

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=40
#SBATCH --time=1:00:00
#SBATCH --job-name=gnuparallel40

# load modules needed...
module load gcc
module load gnu-parallel

# Run the code on 40 cores.
parallel -j 40 --shuf './mycode {}; echo "job {} done"' ::: $(seq 1000)
```

ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Another example: GNU Parallel with a list of files

Example:

```
$ find . -name '*.csv' | parallel -j 4 grep -l JFK
```

This accomplishes the following:

- The linux command `find` lists files with the extension `csv` in the current directory and its subdirectory.
- parallel, which is the GNU parallel command, divides up the list of filenames up.
- For each filename, it executes the linux `grep` command for each filename to find files containing `JFK`.
- It runs 4 subjobs at the same time, because of the `-j4` parameter. Without this parameter, GNU parallel uses as many cores as there are.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

GNU Parallel combining input lists

- GNU parallel can take several sets of parameters, and run them in every combination.
- For instance:

```
$ parallel echo ::: 1 2 ::: A B ::: ! ?
```

yields:

```
1 A !
1 A ?
1 B !
1 B ?
2 A !
2 A ?
2 B !
2 B ?
```

```
$ parallel echo {3} {1} {2} ::: 1 2 ::: A B ::: ! ?
```

yields:

```
! 1 A
? 1 A
! 1 B
? 1 B
! 2 A
? 2 A
! 2 B
? 2 B
```

SciNet ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

More substitution strings

`{.}`	input without extension
`{/}`	basename of input
`{//}`	directory name of input
`{/.}`	basename of input without extension
`{#}`	sequence number of (sub)job
`{%}`	(sub)job slot number

(prefix with a number when using multiple input lists)

Task generation: Done!

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

GNU Parallel using a database

Using Sqlite

- GNU parallel can create entries for each subjob in a database.
- Then it can run those, filling in the job particularities.

```
$ parallel --sqlmaster sqlite3:///db.sq/tbl  echo ::: 1 2 ::: A B ::: ! ?
```

This stores the jobs in the file `db.sq`, in table `tbl`.

```
$ parallel --sqlworker sqlite3:///db.sq/tbl
```

This executes the jobs in the database.

Keeps track of runtime, completion, parameters.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

GNU Parallel's restart capability

- `--joblog LOGFILE`, causes parallel to output a record for each completed subjob.
- The records contain information about subjob duration, exit status, and other goodies.
- `--resume`, when combined with `--joblog`, continues a full GNU parallel job that was killed prematurely.

  For this to work the original GNU parallel job must have had a `--joblog` option.

Monitoring: Done!

Fault Tolerance: Done!

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Conclusion

Conclusion

- Be aware of the features of your code, and the details of the hardware where you will run it.
- If you need to run serial jobs on a cluster with multicore architecture, be sure to run them in parallel.
- Unless your jobs all take the same amount of time, don't try to write your own serial-job management code.
- Use GNU Parallel to manage your serial jobs!

Thank you for your attention!

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO