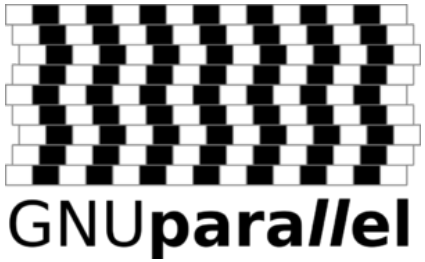
Efficient Submission of Serial Jobs

Jonathon Anderson
based on a tutorial by Aaron Holt

Tools



CURC “loadbalancer”

Batch job with one serial task

```
#!/bin/bash
#SBATCH --job-name process_file
#SBATCH --nodes 1
#SBATCH --output process_file.out
#SBATCH --time 01:30:00

python main.py input_file_1.csv
```

Batch job with multiple tasks

Serial scripts on one processor

```
#!/bin/bash
#SBATCH --nodes 1
#SBATCH --time 18:00:00

python main.py input_file_1.csv
python main.py input_file_2.csv
[...]
python main.py input_file_12.csv
```

Serial scripts on multiple processors

```
#!/bin/bash
#SBATCH --nodes 1
#SBATCH --time 01:30:00

python main.py input_file_1.csv  &
python main.py input_file_2.csv  &
[...]
python main.py input_file_12.csv  &
wait
```

Exercise 1

In a terminal, background 2 ‘sleep’ commands which pause for 30 seconds. Use ‘wait’ to wait for the commands to finish

You may use a login node or a personal computer

Useful Commands	
&	This will background a process when placed at the end of a command
sleep N	Pause for N seconds
wait	Wait for a process state change

Login

`ssh username@rc.colorado.edu`

(or `ssh user00xx@tutorial-login.rc.colorado.edu`)

Exercise 1 (solution)

```
$ sleep 30 &
[1] 18258
$ sleep 30 &
[2] 18259
$ wait
[1]-  Done                    sleep 30
[2]+  Done                    sleep 30
```

alternatively

```
$ sleep 30 & sleep 30 & wait
```

Exercise 2

Submit a slurm job which backgrounds 6 ‘sleep’ commands which pause for 10 seconds.

- Batch filename called sleep.sh
- Commands should run in parallel
- Use 1 node in default queue
- 2 minute wall time
- Output file called sleep.out
- Reservation tutorial1

Login

`ssh username@rc.colorado.edu`
(or `ssh user00xx@tutorial-login.rc.colorado.edu`)
then `ssh scompile`

Exercise 2

Submit a slurm job which backgrounds 6 ‘sleep’ commands which pause for 10 seconds.

- Batch filename called sleep.sh
- Commands should run in parallel
- Use 1 node in default queue
- 2 minute wall time
- Output file called sleep.out
- Reservation tutorial1

Useful Commands	
&	Background a process when placed at the end of a command
sleep N	Pause for N seconds
wait	Wait for a process state change
sbatch file.sh	Submit file.sh to slurm (need to load slurm with ‘ml slurm’ first)

Sbatch Options	
--nodes N	Request N nodes
--output	Specify an output filename
--reservation	Specify reservation to run in
--time	Wall time, format HH:MM:SS

Exercise 2 (solution)

sleep.sh

```
#!/bin/bash
#SBATCH --nodes 1
#SBATCH --time 0:02:00
#SBATCH --output sleep.out
#SBATCH --reservation tutorial1
sleep 10 &
sleep 10 &
sleep 10 &
sleep 10 &
sleep 10 &
sleep 10 &
wait
```

Submit with

```
$ sbatch sleep.sh
```

Bash script summary

- You don’t need a special tool
- Available almost everywhere
- Takes some bash experience to write more complex scripts
- Not great for running large numbers of tasks

GNU parallel

- A shell tool for executing tasks in parallel using one or more computers
- In it’s simplest form, a parallel replacement of a for loop
- Options to specify how many tasks should run in parallel, display output in order, limit resources and more!

GNU parallel

parallel *[options] [command [arguments]] (::: arguments | :::+ arguments | :::: argfile(s) | ::::+ argfile(s)) ...*

Two ways to print numbers 1 to 4 in parallel

```
$ parallel echo {} ::: 1 2 3 4
```

```
$ seq 1 4 | parallel echo {}
```

GNU parallel (examples)

Bash loop

```
for i in {1..10}
do
    echo $i
done
```

GNU parallel

```
seq 1 10 | parallel echo {}
```

GNU parallel (examples)

Bash loop

```
for i in {1..100}
do
    echo $i | grep 1$
done
```

GNU parallel

```
seq 1 100 | parallel 'echo {} | grep 1$'
```

Exercise 3

Convert a bash loop to GNU parallel

```
for i in {1..10}
do
    python print_input.py
file_$i.csv
done
```

Hint: use `{}` instead of `$i`: `file_{}.csv`

`print_input.py`

```
import sys
print(sys.argv)
```

Setup

```
$ module load gnu_parallel
```

Previous examples

```
for i in {1..10}
do
    echo $i
done
```

```
seq 1 10 | parallel echo {}
```

Exercise 3 (solution)

```
seq 1 10 | parallel python print_input.py file_{}.csv
```

Output

```
['print_input.py', 'file_1.csv']
['print_input.py', 'file_2.csv']
[...]
['print_input.py', 'file_10.csv']
```

GNU parallel, useful options

View what commands parallel will run without executing them

```
$ seq 10 | parallel --dry-run echo {}
```

Limit number of tasks running at one time:

```
$ seq 10 | parallel -j 2 echo {}
```

Wait until enough memory is available to start next task:

```
$ seq 10 | parallel --memfree 2G echo {}
```

GNU parallel with Slurm

submit_gnu_parallel.sh

```
#!/bin/bash
#SBATCH --job-name gnu_parallel
#SBATCH --nodes 1
#SBATCH --output gnu_parallel.out
#SBATCH --reservation tutorial1
#SBATCH --time 01:00:00

seq 10 | parallel python print_input.py file_{}.csv
```

print_input.py

```
import sys

# print arguments
print(sys.argv)

# process data here
```

GNU parallel summary

- Great for replacing and speeding up simple loops
- Control how your tasks are run
- Can run on multiple computers as well (may take some effort to get working with slurm)
- Lots of examples and documentation online
- Useful tool outside of compute nodes too
- Not always available

CURC loadbalancer

- Submitting hundreds of slurm jobs is inefficient
- Balances serial applications using MPI (without needing knowledge of MPI!)
- Schedules tasks across multiple nodes from one job
  - Choose how many tasks will run at a time
  - Starts tasks in order (no control over output order)
  - Replaces finished tasks with new ones
  - Straightforward input format

CURC loadbalancer input file

One task per line

Each task may run multiple commands, each command separated by a semicolon

```
for i in {1..100}
do
    echo “sleep 2; echo process $i” >> cmd_file
done
```

CURC loadbalancer input file (example)

cmd_file

```
sleep 2; echo process 1
sleep 2; echo process 2
sleep 2; echo process 3
[...]
sleep 2; echo process 98
sleep 2; echo process 99
sleep 2; echo process 100
```

CURC loadbalancer with Slurm

```
$ sbatch submit_lb.sh
```

submit_lb.sh

```
#!/bin/bash
#SBATCH --nodes 2
#SBATCH --ntasks-per-node 24
#SBATCH --output output.out
module load intel impi
module load loadbalance
mpirun lb cmd_file
```

Exercise 4

Generate a loadbalancer input file

- Should be 50 lines long
- No copy/paste coding! Use a loop

Useful Commands	
`echo`	Background a process when placed at the end of a command
`>>`	Append to an output file
`"..."`	Wait for a process state change

`cmd_file`

```
hostname; sleep 2; echo process 1; python print_input.py file_1.csv
[...]
```

Exercise 4 (solution)

```
for i in {1..50}
do
     echo “hostname; sleep 2; echo process $i; python print_input.py $i” >> cmd_file
done
```

Exercise 5

Submit a loadbalancer batch job

- Limit your job runtime to 2 minutes
- tutorial1 reservation
- Output file `loadbalance.out`
- 1 node
- 24 tasks per node
- Call your batch script `submit_lb.sh`
- Hint: You need the `intel`, `impi`, and `loadbalance` modules
- Hint: ‘`mpirun lb cmd_file`’ will be used in your batch script

Exercise 5 (solution)

submit_lb.sh

```
#!/bin/bash
#SBATCH --nodes 1
#SBATCH --ntasks-per-node 24
#SBATCH --output loadbalance.out
#SBATCH --time 00:02:00
#SBATCH --reservation tutorial1

module load intel impi
module load loadbalance
mpirun lb cmd_file
```

Submit with

```
$ sbatch submit_lb.sh
```

Output from multiple nodes

Ran on 5 nodes with 5 tasks per node.

Input file with ‘`sleep 2; echo process $i`’

```
process 1
process 2
process 4
process 3
process 5
process 6
process 8
process 7
process 9
```

Load Balancer Summary

- No mpi knowledge required
- Saves time by reducing scheduling overhead
- Runs on multiple nodes
- Input file can be created in your favorite language
- Non-standard (but it is on github)
  - https://github.com/ResearchComputing/lb

Summary

1) Save yourself some time waiting in the queue by specifying a wall time on your jobs (--time)
2) Efficiently use resources to speed computation and allow more users to use the supercomputer at once. Run as many tasks as you can per node (up to the number of cores on that node)

References

Bash scripting

https://www.rc.colorado.edu/blog/reducejanuswaittimes

GNU parallel

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

Examples: https://www.gnu.org/software/parallel/man.html

O. Tange (2011): GNU Parallel - The Command-Line Power Tool,
;login: The USENIX Magazine, February 2011:42-47.

CURC loadbalancer

https://www.rc.colorado.edu/support/examples-and-tutorials/load-balancer.html

Additional exercises

Use GNU Parallel to parallelize additional loops

Exercise 6

```
for color in red green blue
do
    for size in S M L XL XXL
    do
        echo $color $size
    done
done
```

Exercise 7

```
(
  for color in red green blue
  do
    for size in S M L XL XXL
    do
        echo $color $size
    done
  done
) | sort
```

Additional exercises (solutions)

Exercise 6

```
parallel echo {1} {2} ::: red green blue ::: S M L XL XXL
```

Exercise 7

```
parallel echo {1} {2} ::: red green blue ::: S M L XL XXL | sort
```