# GNU Parallel

Ketan Maheshwari, ORNL

William Arndt, NERSC

# Overview

Part-I

- Introduction
- Syntax & Semantics
- Key Features

Part-II

- Parallel on a login node
- Parallel in a single node job
- Application Example
- Parallel in a multiple-node job

# Part - I

# What is GNU Parallel

- A tool to parallelize shell commands
- Runs tasks in parallel from shell
- Easy to install, highly configurable
- Well suited to run many single-core tasks on:
  - Compute nodes leveraging multicore architectures
  - Bag of Workstations such as testbeds
- Mature (20 yrs old), simple, powerful!

# Syntax and Semantics

Basic Syntax comprise of three common forms:

- **`parallel [options / flags] <command> ::: <args>`**
  Triple colon semantic: Run <command> in parallel for each of the input parameters

- **`parallel [options / flags] <command> :::: <input_file>`**
  Quad colon semantic: Run <command> in parallel for each line in input file; **`-a`** is alternative syntax to quadruple colon

- **`<command0> | parallel [options / flags] <command1>`**
  Semantic: Run <command1> in parallel for each line of the standard output from <command0> as arg

NB: The ::: / :::: separator may be changed with **`--arg-sep`** option if needed / preferred.

# Examples

- Triple colon:
  - `parallel echo ::: {1..4}`
  - `parallel wc ::: *.txt`
- Quadruple colon:
  - `parallel echo :::: /etc/passwd`
  - `parallel -a /etc/passwd echo # same as above`
- Pipe form example:
  - `find /somedir/subdir -iname '*.txt' -print | parallel echo`

# Examples (contd)

- With multiple `:::` all input combinations will be generated
  - `parallel echo ::: A B C ::: 1 2 3`
    Equivalent to:

```
for i in A B C; do
  for j in 1 2 3; do
    Echo $i $j
  done
done
```

- Use {[n]} to put nth set of arguments in multiple commands
  - `parallel "echo counting {} ; wc -l {}" ::: *.txt`
  - `parallel "mkdir -p /tmp/dir.{1} ; fallocate -l 1K /tmp/dir.{1}/file.{2}" ::: {1..4} ::: {a..d}`

- Other patterns may be put in `{}` to treat args in special ways. See man page.

# Many sources for Getting Help

```
man parallel

man parallel_tutorial

parallel --help
```

www.pi.dk/1 # youtube tutorials from the author of GNU Parallel

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

https://hn.algolia.com/?q=gnu+parallel

```
man parallel_alternatives
```

# Parallel is Configurable

- `--keep-order/-k` will ensure the output order is preserved
  ```
  parallel -k "sleep {} ; echo {}" ::: {5..1}
  ```

- `--jobs/-j` to control the job slots
  ```
  parallel -j 2 echo ::: 5 4 3 1 2
  ```
  0 means as many jobs as possible, default is all cores on a machine. May be provided as %.

- `-N` to limit the arguments received at a time
  ```
  parallel -N3 echo ::: {A..F}
  A B C
  D E F
  ```
  Use `-N0` when command takes no arguments

- `--delay <x.y>` adds x.y secs delay in dispatching tasks to prevent overwhelming the system

# Parallel is Configurable (contd.)

- `--timeout`: kill a job if it takes more than a certain time (sec)
  - `parallel --timeout 1000 ./runtask ::: {1..100}`
    timeout value may be specified as a percentage value of the median runtime
    `parallel --timeout 200% ./runtask ::: {1..100}`

- `--progress`, `--eta`, `--bar`: show progress of a run in terms of estimated time, tasks, nodes etc.

- `--wd <dirlocation>`: provide a working directory for commands

- `--dry-run`: show what will run in standard output but do not run anything.

# Checkpointing and Resume with joblog

**`--joblog, --resume`**: Allows for monitoring progress, checkpointing and resuming an interrupted / partially failed run

```
parallel -j 16 -n 100 --joblog /tmp/job.log --resume
gen_digest {} :::: keys.txt
```

Additionally, **`--retry-failed`** (reads from log) and **`--resume-failed`** (resumes afresh) are available to try failed jobs again.

# Configuration Profiles

- Configuration profiles may be saved in files and used in combinations
  - `/etc/parallel/config` for systemwide configuration
  - `~/.parallel/config` for user-level configuration which will override systemwide

- Multiple config profiles may be created and used in combinations

```
cat ~/.parallel/benice
--nice 17
--timeout 300%

cat ~/.parallel/dryv
--vv
--dry-run

parallel --profile benice --profile dryv <heavy_process> :::
<args>
```

# Parallel works well with Remote Systems over ssh

General Syntax:

```
parallel -S server1,server2 commands flags ::: args
```

e.g.:

```
parallel -S u@vm1,u@vm2 "hostname; echo {}" ::: foo bar
```

`--sshloginfile` flag allows to read the ssh config from a file, eg. .ssh/config

Remote ssh hosts may be divided into groups and jobs may be selectively run:

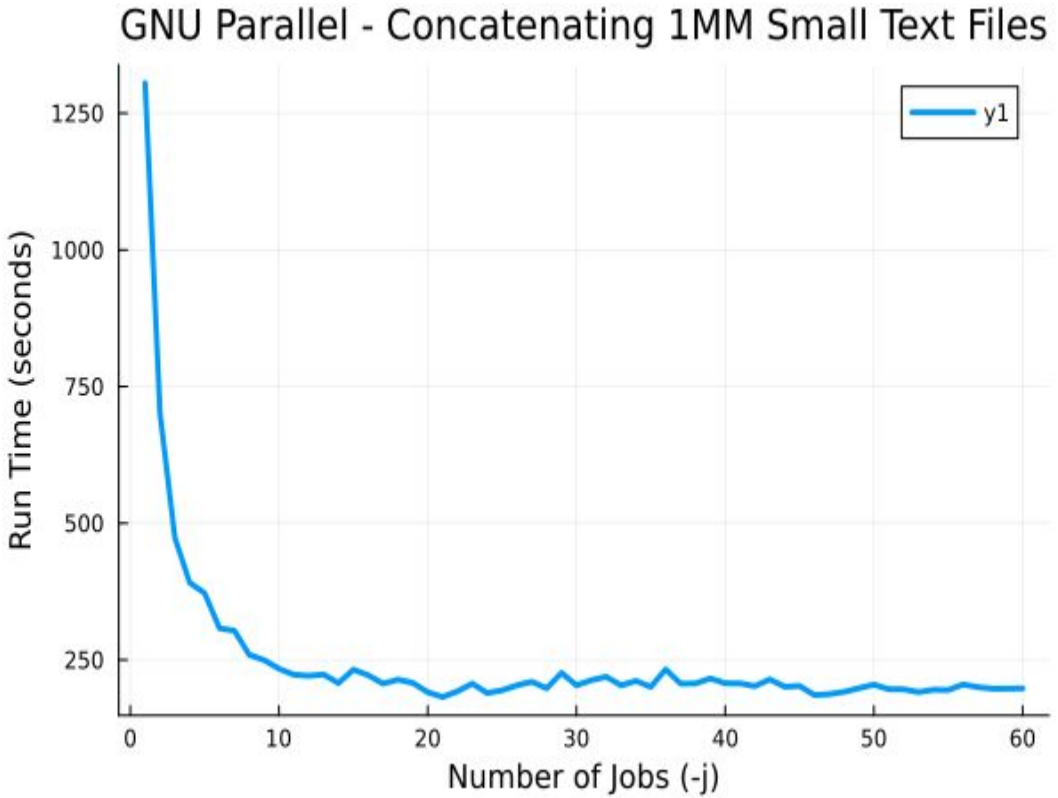
```
parallel –hostgroup -S @g1/nid1 -S @g2/nid2 echo :::
run_on_g1@g1 run_on_g2@g2
```

# Gotchas and Limitations

Syntactic whim: need to provide args even when not required! Need to use `-N0`

```
parallel -N0
cmd_takes_no_args :::
{1..3}
```

Scaling: diminishing returns




Courtesy: randyzwitch.com/gnu-parallel-medium-data/

# GNU Parallel at NERSC, OLCF, ALCF

- On Summit, parallel is installed as core software (not a module)
  `/usr/bin/parallel`
- On Crusher (and Frontier) and Perlmutter, parallel is available as a software module
  `module load parallel`
- On Theta and Polaris, parallel is available as a software module
  `module load gnu-parallel`

# Part - II

# What if my tasks need more than one parameter?

Starting with an input file that has two parameter columns, space separated:

```
warndt@perlmutter:login32:~/parallel_demonstration> parallel echo ::: 2 3 ::: a b c > two_input.txt
warndt@perlmutter:login32:~/parallel_demonstration> cat two_input.txt
2 a
2 b
2 c
3 a
3 b
3 c
```

Access each parameter by giving parallel the separation character, and then putting a column number in the {} character:

```
warndt@perlmutter:login32:~/parallel_demonstration> parallel --colsep " " echo {1} and {2} :::: two_input.txt
2 and a
2 and b
2 and c
3 and a
3 and b
3 and c
```

# Using parallel on one Perlmutter node:

Start with a basic batch submission script and input file:

```
warndt@perlmutter:login32:~/parallel_demonstration> cat single_node.sh
#!/bin/bash
#SBATCH --qos=debug
#SBATCH --nodes=1
#SBATCH --constraint=cpu
module load parallel
srun parallel echo :::: input.txt
warndt@perlmutter:login32:~/parallel_demonstration> cat input.txt
2
3
4
```

```
warndt@perlmutter:login32:~/parallel_demonstration> sbatch single_node.sh
Submitted batch job 5257417
… waiting a bit …
warndt@perlmutter:login32:~/parallel_demonstration> cat slurm-5257417.out
2
3
4
```

# A real application example:

Start with an application:

```
$HOME/pm-hmmer-3.3.2/src/hmmsearch --cpu 8 --noali -o output.txt $SCRATCH/CR_data/Pfam-A.hmm input.fasta
```

Which we want to run on every fasta file in this directory:

```
warndt@perlmutter:login28:/pscratch/sd/w/warndt/data> ls | head -3
uniprot_100.fasta
uniprot_101.fasta
uniprot_102.fasta
warndt@perlmutter:login28:/pscratch/sd/w/warndt/data> ls | wc -l
256
```

Use a find command to create the input task file with full paths:

```
warndt@perlmutter:login28:/pscratch/sd/w/warndt/data> find $PWD -type f | grep fasta | sort > input.txt
warndt@perlmutter:login28:/pscratch/sd/w/warndt/data> head -3 input.txt
/pscratch/sd/w/warndt/data/uniprot_100.fasta
/pscratch/sd/w/warndt/data/uniprot_101.fasta
/pscratch/sd/w/warndt/data/uniprot_102.fasta
warndt@perlmutter:login28:/pscratch/sd/w/warndt/data> wc -l input.txt
256 input.txt
```

# A real application example, continued

Batch submission script:

```
warndt@perlmutter:login28:~/parallel_demonstration> cat pm.run.slurm
#!/bin/bash
#SBATCH -q regular
#SBATCH -N 1
#SBATCH -C cpu
#SBATCH -t 6:00:00
module load parallel
srun parallel --dry-run -j 64 $HOME/pm-hmmer-3.3.2/src/hmmsearch --cpu 4 --noali -o {//}/output_{/.}.txt $SCRATCH/CR_data/Pfam-A.hmm {} :::: $1
```

Pass it off to the batch scheduler with the input task file:

```
warndt@perlmutter:login28:~/parallel_demonstration> sbatch pm.run.slurm /pscratch/sd/w/warndt/data/input.txt
Submitted batch job 5302714
warndt@perlmutter:login28:~/parallel_demonstration> sqs
JOBID          ST USER      NAME          NODES TIME_LIMIT       TIME  SUBMIT_TIME          QOS           START_TIME          FEATURES       NODELIST(REASON
5302714         R  warndt    pm.run.slurm  1       6:00:00       0:26  2023-02-06T11:31:23  regular_1     2023-02-06T11:31:31  cpu            nid007077
```

# A real application example, finished

After the job is finished all output files are in the data directory:

```
warndt@perlmutter:login28:~/parallel_demonstration> ls $SCRATCH/data | grep output | head -3
output_uniprot_100.txt
output_uniprot_101.txt
output_uniprot_102.txt
warndt@perlmutter:login28:~/parallel_demonstration> ls $SCRATCH/data | grep output | wc -l
256
```

# A way to use parallel with multiple node allocations

Begin by placing your task command inside payload.sh, `chmod` it to execute:

```
warndt@perlmutter:login28:~/parallel_demonstration> cat payload.sh
#/bin/bash
echo This is task $1; hostname
warndt@perlmutter:login28:~/parallel_demonstration> ls -lh payload.sh
-rwxrwx--- 1 warndt warndt 32 Feb  6 11:41 payload.sh
```

Make a copy of parallel_runner.sh:

```
warndt@perlmutter:login28:~/parallel_demonstration> cat parallel_runner.sh
#!/bin/bash
module load parallel
cat $2 | awk -v NNODE="$SLURM_NNODES" -v NODEID="$SLURM_NODEID" 'NR % NNODE == NODEID' | parallel -j $1 payload.sh {}
```

Write the batch submission script:

```
warndt@perlmutter:login28:~/parallel_demonstration> cat pm.multi.slurm
#!/bin/bash
#SBATCH --qos=regular
#SBATCH --Nodes=2
#SBATCH --constraint=cpu
#SBATCH --ntasks-per-node 1
srun --no-kill --ntasks=2 --wait=0 parallel_runner.sh $1 $2
```

Submit the script to the batch system:

```
warndt@perlmutter:login28:~/parallel_demonstration> cat input.txt
2
3
4
warndt@perlmutter:login28:~/parallel_demonstration> sbatch pm.multi.slurm 2 input.txt
Submitted batch job 5303405
warndt@perlmutter:login28:~/parallel_demonstration> sqs
JOBID          ST USER      NAME         NODES TIME_LIMIT      TIME  SUBMIT_TIME         QOS          START_TIME         FEATURES      NODELIST(REASON
5303405        PD warndt    pm.multi.slu 2         10:00        0:00  2023-02-06T12:16:18  regular_1     N/A                cpu           (Priority)
```

Once the job is allocated, see the compute nodes it is using:

```
warndt@perlmutter:login28:~/parallel_demonstration> sacct -j 5303405 -X -p -o nodelist
NodeList|
nid[005755,006153]|
```

And the output shows different nodes were used:

```
warndt@perlmutter:login28:~/parallel_demonstration> cat slurm-5303405.out
This is task 3
nid005755
This is task 2
nid006153
This is task 4
nid006153
```

# When GNU Parallel Should or Should Not be Used

- High Throughput Computing - Lots of copies of the same command, in no particular order.
  - “Write programs that do one thing and do it well.”
- Do you have any constraint about the order the tasks are run?
  - If yes, don’t use GNU Parallel.
- Are your tasks using MPI?
  - If yes, don’t use GNU Parallel. (It *could* be possible, but won’t be easier than something else)
- Do any of your tasks, or the sum of all their running time, not fit within the maximum job walltime?
  - If yes, don’t use GNU Parallel.

# Thanks for your attention. Training resumes at 9:15 with Parsl.