Scientific Workflows at scale using GNU Parallel

Ketan M. (km0@ornl.gov)
Oak Ridge National Laboratory
Tutorial presented at eScience 2023

OAK RIDGE National Laboratory

Acknowledgements

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

I acknowledge the original developer and maintainer of GNU parallel, **Ole Tange** for the gift of a wonderful tool!

A previous shorter version of this tutorial was presented as part of the Workflows Community Initiative[1] Cross-Facility Workflows Tutorials. I acknowledge **William Arndt** from NERSC / LBNL for his contributions to the tutorial.

I thank the eScience Tutorial committee and chairs for providing this opportunity.

1 https://workflows.community

Table of Contents

Part 1: Overview and Logistics

back to toc

Overview: What shall we learn

- Familiarize with GNU Parallel features
- How to utilize HPC with GNU Parallel
  - Resource Management
  - Working with multicore architectures
- Running workflow tasks asynchronously
  - Data dependencies and Parallelism

Slides and practice data for download

- Slides and practice files available:

github.com/ketancmaheshwari/escience23tut

`/data` has data used for exercises

`/src` has code shown here

About You and Me

- Basic exposure to Linux is assumed but feel free to interrupt and ask questions
  - common commands, basic understanding of files and directories, editing. eg. **cd, ls, pwd, cat**

- About Me
  - Sr. Linux Engineer at Oak Ridge National Laboratory
  - Command line and Linux terminal enthusiast

About the Slides

- 8 Parts, each part has 4-10 slides
- Lots of examples in slides
- Summary and Practice Exercises
  We try to solve them here (if time permits)
- Solve it offline if we run out of time

Part 2: Introduction to GNU Parallel

back to toc

What is GNU Parallel

- A terminal tool to parallelize shell commands
- 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
  - Works well with Resource Managers and job schedulers
- Mature (20 years old), Simple and Powerful!

Installation

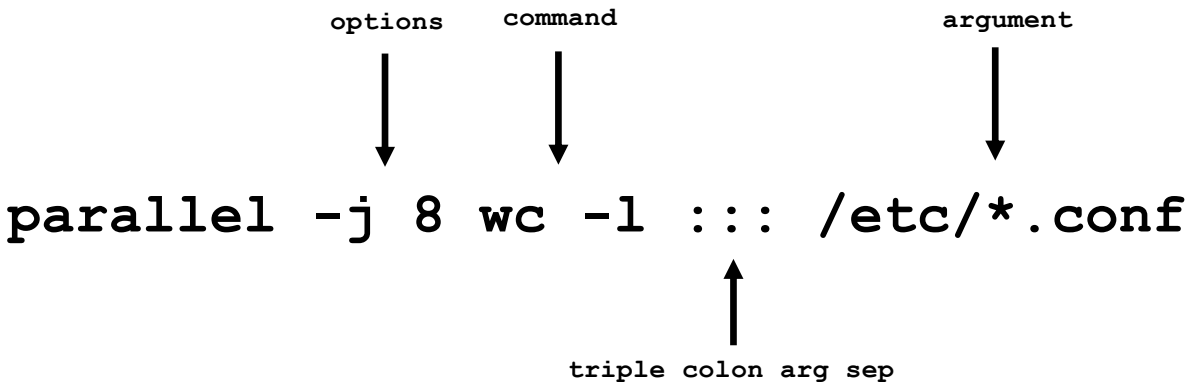
- Download the latest version:
```
curl -s https://ftp.gnu.org/gnu/parallel/parallel-latest.tar.bz2 -o parallel-latest.tar.bz2
```
- Untar and cd into it:
```
tar zxf parallel-latest.tar.bz2
cd parallel-20230822
```
- Install:
```
./configure --prefix=$HOME/parallel-install
make install    # needs libevent
```
- Set PATH and it is ready to use:
```
export PATH=$HOME/parallel-install/bin:$PATH
which parallel
```

Many sources for getting help

- link to youtube videos by Ole Tange: www.pi.dk/1
- www.gnu.org/software/parallel/parallel_cheat.pdf
- Searching for “gnu parallel” on Hacker news, Reddit, Stack Exchange yields many helpful links

```
man parallel
man parallel_tutorial
parallel --help    # summary of most imp options
parallel --max-line-length-allowed # max size of cmdline
parallel --number-of-cpus && parallel --number-of-cores
```

Anatomy of a GNU Parallel Command

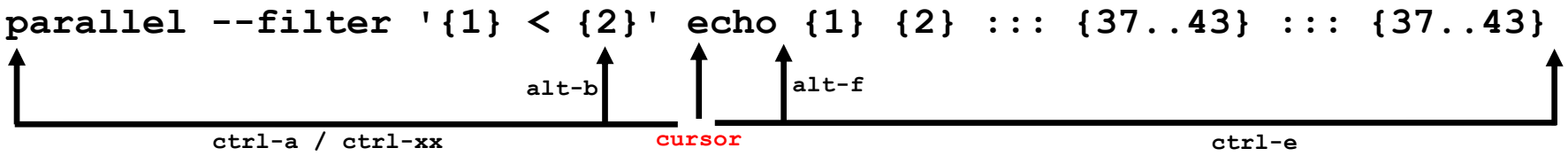


Another Form of same command


```
\ls -1 /etc/*.conf | parallel -j8 wc -l
```

Aside1: Command line Navigation

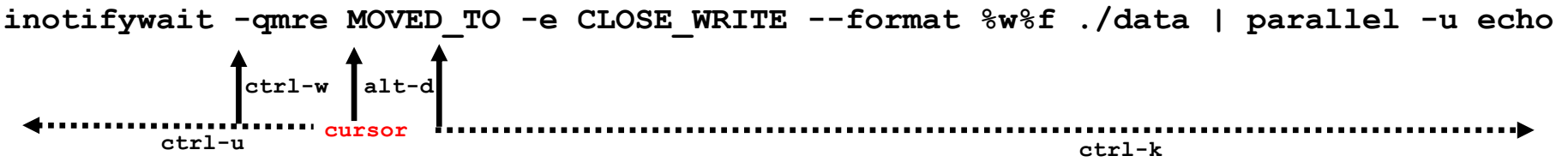
MAC users: terminal pref > profile > keyboard settings > Use option as meta key



ctrl-] <char> moves cursor to 1st occurrence of <char> to right

ctrl-alt-] <char> moves cursor to 1st occurrence of <char> to left

Aside2: command line Deletion



use ctrl-y to paste back the deleted

GNU Parallel Alternatives

- **`xargs`, `make -j`, `find + exec`,** and others are often cited as alternatives

- A comparison is made and summaries available:
  `gnu.org/software/parallel/parallel_alternatives.html`

- An insightful read, though it may or may not be unbiased! ☺

Part 3: Features and Examples - I

back to toc

Basic Syntax and Semantics

Triple colon semantic: Run <command> in parallel for each of the input parameters:

```
parallel [options] <command> ::: <args>
```

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

```
parallel [options] <command> :::: <input_file>
parallel [options] -a <input_file> <command>
```

Semantics: Run **`<command1>`** in parallel for each line of the standard output from **`<command0>`** as **arg**

```
<command0> | parallel [options] <command1>
```

Examples

- Triple colon:

```
parallel echo ::: {1..4}
parallel du -h ::: */*
```

- Quadruple colon:

```
parallel echo :::: /etc/passwd
parallel -a /etc/passwd echo # same as above
```

- Another example of the pipe form:

```
find /somedir/subdir -iname '*.txt' -print | parallel echo
"File: "
```

Examples

- With multiple `:::` all **combinations** will be generated

```
parallel echo ::: A B C ::: 1 2 3
```

Is equivalent to a nested for loop!:

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

- `--link` to map the args 1:1

```
parallel --link echo ::: A B C ::: 1 2 3
```

Examples

- Use **{[n]}** to put nth set of arguments in multiple commands / args:

```
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:
  {.} remove extension, eg. /tmp/afile.txt --> /tmp/afile
  {/} extracts just the filename, eg. /tmp/afile.txt --> afile.txt

  {#} sequence number of the job
  {%} slot number of the job

```
parallel echo "sequence {#} slot {%}" ::: {1..100}
parallel echo {2.} {1} ::: 1 2 ::: afile.txt bpic.png
```

Highly Configurable I

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

- **`--jobs/-j`** to control the job slots (limited by available cores)
  `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 %.
  Silently ignore if value is greater than cores available.

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

Highly Configurable II

- **`--timeout`**: kill a job if it takes more than a certain time (sec)

```
parallel --timeout 1000 ./runtask ::: {1..100}
```

  may be specified as a percentage value of the median runtime (<100% won't make sense)

```
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 (cwd) for commands

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

Checkpoint and Resume

`--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) to try failed jobs again.

Saving Output in Files, Variables, Databases

- Outputs may be saved in files:
```
parallel --files echo ::: A B C # will be saved in /tmp
```
- Saving output in CSV file:
```
parallel --results my.csv echo ::: A B ::: C D
```
- Saving to an SQL database:
```
DBURL=sqlite3:///mydatabase; TABLE=$DBURL/mytable
parallel --sqlandworker $TABLE echo ::: A B ::: C D
```
- Saving to shell variables:
```
env_parallel --install # activate parset, restart shell
parset myvar1,myvar2 -j2 echo ::: a b
echo $myvar1 $myvar2
```

Config Profiles I

- Specific 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:

```
$ cat ~/.parallel/savesql
--sqlandworker sqlite3://user:passwd@host:9900/mydatabase/mytable

parallel --profile savesql <analytics_process> ::: <1m args>
```

Multiple Config Profiles may be used together

- **cat ~/.parallel/benice**
  --nice 17
  --timeout 300%
- **cat ~/.parallel/dryv**
  --vv
  --dry-run
- **parallel --profile benice --profile dryv <heavy_process> ::: <args>**

Part 4: Features and Examples - II

back to toc

Resource Management

- **`--load`**: To avoid overloading systems, look at the load before starting another job.
  `parallel --load 100% echo ::: "Load less than 1 job per CPU"`
- **`--noswap`**: Check if the system is swapping and run only when not.
  `parallel --noswap echo ::: "System is not swapping now"`
- **`--memfree`**: Run only when enough memory is free.
  `parallel --memfree 1G --retries 5 echo ::: "1G is free now."`
  note: max memory is the "available" on the **`free`** command
- **`--delay <x.y>`** adds x.y sec delay in dispatching tasks to prevent overwhelming the system

Combine Data and GNU Parallel in One Script

- With the `--shebang` flag like so:

```
#!/usr/bin/parallel --shebang -r echo

data_item1
data_item2
data_item3
```

- Parallelize existing scripts with `--shebang-wrap`

```
#!/usr/bin/parallel --shebang-wrap /bin/bash
echo "Arguments $@"

chmod u+x parbash.sh
./parbash.sh 1 2 3
```

Working with Remote Systems over SSH

- General Syntax:
  `parallel -S server1,server2 commands flags ::: args`
- Example:
  `parallel -S u@vm1.org,u@vm2.org "hostname; echo {}" ::: foo bar`
  `--sshloginfile` flag allows to read the remote ssh config from a file, **`eg. .ssh/config`**
- Remote ssh hosts may be divided into groups and jobs may be selectively run on them:

```
parallel --hostgroup -S @grp1/$server1 -S @grp2/$server2 \
 echo {} ::: run_on_grp1@grp1 run_on_grp2@grp2
```

GNU parallel can transfer data to / from remote

- **`--transferfile`** to transfer files. Uses **`rsync`** to do transfer
- **`--return`** to return files from remote via **`rsync`**
- **`--cleanup`** to remove files from remote once job is done

```
echo "This is input file" > input_file
parallel -S remote_server --transferfile {} cat ::: input_file
```

```
echo "This is input file" > input_file
parallel -S $remote_server1 --transferfile {} --return {}.out cat {} ">" {}.out ::: input_file
```

- **`--trc`** to combine the three options (**`--transferfile, --return, and --cleanup`**)

Real-World Examples working with ssh I

- ```
  parallel -k -S
  rage1,rage4,rage7,rage8,rage9,rage10,rage11,rage12
  ${scan_cmd} ::: \
  ${scan_path}/{44..51} \
  >> scanperf.8proc.8node.out
  ```

- ```
  parallel -S rage4 --jobs 30 'nats -s rage2:4222 \
  pub migration.files.request --count 1 \
  "{\"path\": \"/lustre/crius/migagenttests/{1}/file.{2}\"}”’\
  ::: {0..63} ::: {1..3000}
  ```

Real-world examples working with ssh II

- ```
  parallel -S rage4 --jobs 30 "touch -d '-1 week'\
  /lustre/crius/{1}/file.{2}" ::: {0..63} ::: {1..3000}
  ```

- ```
  parallel -k -S \
  rage1,rage2,rage4,rage5,rage6,rage7,rage8,rage9 \
  /lustre/crius/scripts/measure_lfsfind.sh :::\ {28..35} \
  >> lfsfindperf.8proc.8node.out
  ```

PoliMOR: A Policy Engine "Made-to-Order" for Automated and Scalable Data Management in Lustre

Anjus George, Christopher Brumgard, Rick Mohr, Ketan Maheshwari, James Simmons, Sarp Oral, Jesse Hanley
National Center for Computational Sciences, Oak Ridge National Laboratory
Oak Ridge, TN, USA
{georgea,brumgardcd,mohrrf,maheshwarikc,simmonsja,oralhs,hanleyja}@ornl.gov

ABSTRACT

Modern supercomputing systems are increasingly reliant on hierarchical, multi-tiered file and storage system architectures due to cost-performance-capacity trade-offs. Within such multi-tiered systems, data management services are required to maintain healthy utilization, performance, and capacity levels. We present PoliMOR, management challenging. Data management practices must minimize wastefulness of storage resources, promote fair usage, and satisfy application I/O needs (ideally in a systematic and automated fashion). Common tasks may include purging old files to recover disk space, staging data to a tier with higher performance, or migrating data to tape. Sites could potentially handle some of these

To be presented at PDSW'23 at SC'23

The Pipe Mode to Process Large Data I

- When data is send over a Linux pipe to parallel command, it is treated as **arguments** for command to run:

```
cat data.txt | parallel echo
```

- In the pipe mode, the data is delivered to the parallel command as **stdin:**

```
cat data.txt | parallel --pipe wc -l
```

- The “--pipe” input may be controlled for block-size / number of lines and number of jobs:

```
cat data.txt |\
  parallel --pipe --block 2M -j4 --round-robin wc -l \
  # -N <num> for lines
```

The Pipe Mode to Process Large Data II [1]

- `--pipepart` may be used when using large data. Same as pipe but faster, has a few limitations [2]
- `--recend <string>` splits record at this string
- `--line-buffer` may be used to buffer output by line

```
parallel -a <file.json> --pipepart --keep-order
       --line-buffer --block 100M --recend '}\n' "jq <query>"
```

[1] thenybble.de/posts/json-analysis/
[2] www.gnu.org/software/parallel/parallel_design.html#pipepart-vs-pipe

Part 5: HPC and GNU Parallel

back to toc

A SLURM Workload Manager Primer I

- **`salloc`**
  Obtain a job allocation.
- **`sbatch`**
  Submit a batch script for later execution.
- **`srun`**
  Obtain a job allocation (as needed) and execute an application. Option we will use: **`--wait=0`** means do not terminate other tasks if one finishes
- **`squeue`**
  View information about jobs.
- **`sinfo`**
  View information about nodes and partitions.

A SLURM Workload Manager Primer II

- At runtime, SLURM offers several environment variables that could be leveraged to steer executions:
  - **`$SLURM_NTASKS`**
    Same as -n, –ntasks. The number of tasks.
  - **`$SLURM_CPUS_PER_TASK`**
    Number of CPUs per task.
  - **`$SLURM_NODEID`**
    The node id of the current node. Starts from 0.
  - **`$SLURM_NNODES`**
    Total nodes allocated to current job.
  - **`$SLURM_NODELIST`**
    A list of nodes allocated to current job.

srun parallel vs parallel srun?

```
srun="srun --exclusive -N1 -n1 -c1"
parallel -j $SLURM_NTASKS "$srun ./runtask.sh {1}" ::: {1..112}
```

vs

```
srun --ntasks-per-node=1 parallel -j $cores_per_node app_invocation
```

Consensus: **`srun ... parallel ...`**

Reasons:

1. Higher overhead in invoking multiple srun allocations
2. Leveraging SLURM's srun and environment variables

Working with HPC Schedulers: SLURM, 1 node

```
#!/bin/bash

#SBATCH -J singlenode
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err
#SBATCH -t 0:10:00
#SBATCH -N 1

srun parallel --jobs 6 ./payload.sh argument_{} :::: input.txt
```

code in Github: src/singlenode

Using --dry-run to generate parallel commands

```
#SBATCH --job-name=parallel_job
#SBATCH ...


find infiles/*.txt | parallel --dry-run ./process_data {}
>commands.txt

parallel -j $SLURM_NTASKS < commands.txt

#OR

find infiles/*.txt | parallel --dry-run Rscript
R_array_test.R {} >commands.txt

parallel -j $SLURM_NTASKS < commands.txt
```

Multinode Execution in SLURM

```
#!/bin/bash

#SBATCH -J multinode
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err
#SBATCH -t 0:20:00
#SBATCH -p batch
#SBATCH -N 4

srun --no-kill --ntasks-per-node=1 --wait=0 driver.sh $1
```

Driver and Payload codes

```
# Deliver tasks depending on the nodeid
cat $1 |                                                    \
awk -v NNODE="$SLURM_NNODES" -v NODEID="$SLURM_NODEID" \
'NR % NNODE == NODEID' |                                    \
parallel ./payload.sh argument_{}
```

```
#!/bin/bash

H="$(hostname)"
echo “This is the payload script. \
$1 is the argument passed to it. Ran on machine $H.”
```

Part 6: Asynchronous Workflow Execution

back to toc

Asynchronous Execution of Workflows

proc1.sh:

```
#!/bin/bash
sleep $(shuf -i 20-60 -n 1)
shuf $1 > ./out/proc_$(basename $1)
echo "./out/proc_$(basename $1)" >> jobqueue
```

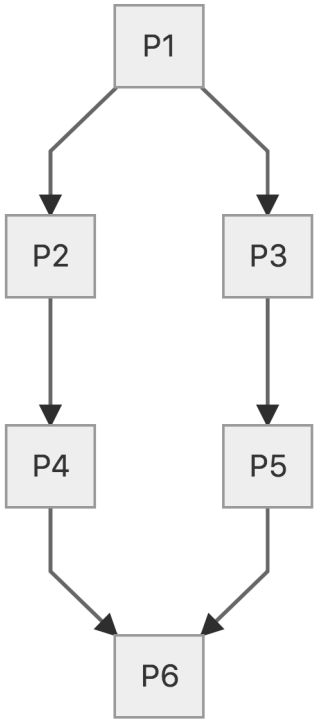
proc2.sh:

```
#!/bin/bash
shuf $1 > ./out/f_$(basename $1)
echo "Done for $1”
```

workflow.sh:

```
#!/bin/bash
parallel ./proc1.sh {} ::: ./inputs/*.txt &
>jobqueue; tail -n+0 -f jobqueue | parallel -u ./proc2.sh {}
```

A Full DAG Workflow Example



- Six processes
- Data dependencies
- May work independently with file inputs
- Branch Parallelism
- Asynchronous execution desired when multiple inputs

Sequential Bash Script Representation for one set of inputs

```
#!/bin/bash
#p1.sh
if test "$#" != 2 ; then
    echo "wrong
invocation..exiting."
    exit 3
fi
if [ -f "$2" ] ; then
    rm -v "$2"
fi
cat $1 >> $2 || exit
echo "processed by p1" >> $2
echo "$2" >> ./q.p1
```

```
#!/bin/bash

p1/p1.sh inputs/in1.txt
p1/out1.txt


p2/p2.sh p1/out1.txt p2/out2.txt
p3/p3.sh p1/out1.txt p3/out3.txt
p4/p4.sh p2/out2.txt p4/out4.txt
p5/p5.sh p3/out3.txt p5/out5.txt


p6/p6.sh p4/out4.txt p5/out5.txt
outputs/out6.txt
```

All sources available in src/crystalworkflow

GNU Parallel version

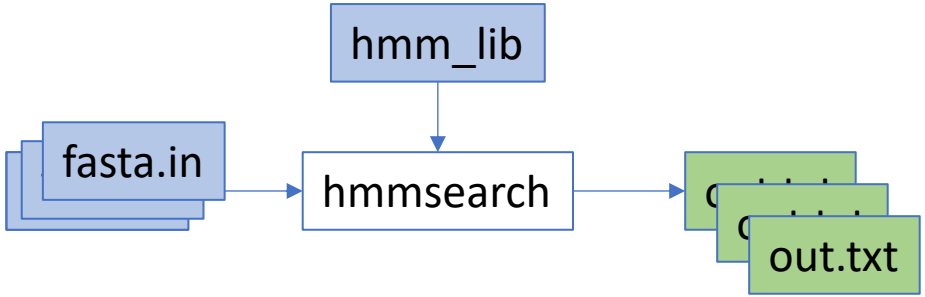
```
#!/bin/bash

mkdir -p p{1..5}/outdir outputs

parallel --link p1/p1.sh {1} {2} ::: inputs/in{1..6}.txt :::
p1/outdir/out{1..6}.txt &


touch q.p1 ; tail -n+0 -f q.p1 | parallel -u --link p2/p2.sh {1} {2} :::: - :::
p2/outdir/out{1..6}.txt &

touch q.p1 ; tail -n+0 -f q.p1 | parallel -u --link p3/p3.sh {1} {2} :::: - :::
p3/outdir/out{1..6}.txt &

touch q.p2 ; tail -n+0 -f q.p2 | parallel -u --link p4/p4.sh {1} {2} :::: - :::
p4/outdir/out{1..6}.txt &

touch q.p3 ; tail -n+0 -f q.p3 | parallel -u --link p5/p5.sh {1} {2} :::: - :::
p5/outdir/out{1..6}.txt &


(stdbuf -oL paste <(touch q.p4 ; tail -n+0 -f q.p4) <(touch q.p5 ; tail -n+0 -f
q.p5)) | parallel -u --link p6/p6.sh :::: - ::: outputs/out{1..6}.txt
```

Part 7: A Real Application

back to toc

A Real Application: Bioinformatics

- HMMER3 Takes a protein sequence and compares it to a probabilistic profile that describes a protein domain.
- It reports when there is a statistically significant likelihood that the protein and the domain share the same evolutionary origin.
- This basic comparison is repeated for all combinations of many protein sequences and many domains.



Download App Package and Data

- Download the hmmsearch package: hmmer.org/download.html
  `wget http://eddylab.org/software/hmmer/hmmer-3.4.tar.gz`
- Download the protein HMM searchable library here:
  `wget ftp.ebi.ac.uk/pub/databases/Pfam/current_release/Pfam-A.hmm.dat.gz`
- Download the fasta file:
  `wget https://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/uniprot_sprot.fasta.gz`

Install the hmmsearch package

- **tar zxf hmmer-3.4.tar.gz**
- **cd hmmer-3.4/**
- **./configure --prefix=$PWD/install**

```
... output ...
HMMER configuration:
  compiler:     gcc -O3    -pthread
  host:         x86_64-pc-linux-gnu
  linker:
  libraries:    -lpthread
  DP implementation:    sse
```

- **make -j install** # hmmer will be installed.

gunzip and split the Fasta File

```
gunzip uniprot_sprot.fasta.gz

awk -v chunksize=$(grep ">" uniprot_sprot.fasta -c)
'BEGIN{n=0; chunksize=int(chunksize/256)+1 }
/^>/ {
       if(n%chunksize==0){
         file=sprintf("uniprot_%d.fasta",1+(n%256));
       }
       print >> file; n++; next;
     }
{ print >> file; }' < uniprot_sprot.fasta
```

Part 8: Putting it all Together

back to toc

HMMSearch Command line

```
hmmsearch --cpu 8 --noali -o output.txt \
                          Pfam-A.hmm input.fasta

ls | head -3
uniprot_100.fasta
uniprot_101.fasta
uniprot_102.fasta

find $PWD -type f | grep fasta > inputs.txt
```

First Approach: srun … parallel

```
srun --no-kill --ntasks-per-node=1 --wait=0

parallel -j 8 $HOME/hmmer-3.3/install/bin/hmmsearch

--cpu 4 --noali -o {//}/output_{/}.txt

$SCRATCH/Pfam-A.hmm {} :::: inputs.txt
```

Second Approach: Distribute Tasks

```
#driver.sh
cat inputs.txt |awk -v NNODE="$SLURM_NNODES" -v
NODEID="$SLURM_NODEID" \ 'NR % NNODE == NODEID' |
parallel ./payload.sh {}
```

```
#payload.sh [untested]
$HOME/hmmer-3.3/install/bin/hmmsearch
--cpu 4 --noali -o {//}/output_{/}.txt
$SCRATCH/Pfam-A.hmm $1
```

Summary

- GNU Parallel is an effective tool that could be useful in day-to-day tasks on the terminal as well as for larger workflows
- Many options to choose from to customize a parallel operation
- Very handy for quick prototyping
- May be well suited for some production level work

back to toc

Practice and Exercises : Titanic Data Challenge

Data: in Github, Titanic.csv

- Problem 1. What characteristics are shared by all passengers whose fare is 0?
- Problem 2. How many married women over age 50 embarked in Cherbourg? (Married women are denoted by "Mrs.")
- Problem 3. Which embarkation city had the highest-paying passengers on average?
- Problem 4. What is the most common last name among passengers? What is the average number of passengers per last name?
- Problem 5. What's the survival rate for passengers in the three different classes, i.e., what fraction of passengers in each class survived?

Practice and Exercises : MIT Datacenter Challenge

- dcc.mit.edu A 2.6T dataset pertaining to cluster operations
- **Partial data** available on Github at **`data/datacenter-challenge`**
- Although geared towards AI based models, **GNU Parallel** could help gain insight on patterns from existing data
- Scheduling Characterization
- Workload Characterization
- File System Characterization
- Error Characterization

Credits, references and resources

- gnu.org/software/parallel
- rcc.uchicago.edu/documentation/_build/html/running-jobs/srun-parallel/index.html
- ulhpc-tutorials.readthedocs.io/en/latest/sequential/basics
- docs.ycrc.yale.edu/clusters-at-yale/guides/parallel
- www.vanderbilt.edu/accre/documentation/parallel
- docs-research-it.berkeley.edu/services/high-performance-computing/user-guide/running-your-jobs/gnu-parallel
- omgenomics.com/parallel
- curc.readthedocs.io/en/latest/software/GNUParallel.html
- thenybble.de/posts/json-analysis
- stackoverflow.com/questions/tagged/gnu-parallel
- unix.stackexchange.com/questions/tagged/gnu-parallel

Thank you for your time and attention
Questions?

km0@ornl.gov

Other Possible Venues to look for challenges

- annas-blog.org/worldcat-scrape.html
- smc-datachallenge.ornl.gov
- www.reddit.com/r/DataHoarder