LSU INFORMATION TECHNOLOGY SERVICES

Parallel Serial Jobs Using GNU PARALLEL

Wei Feinstein
HPC User Services
LSU HPC & LONI
sys-help@loni.org
Louisiana State University
Febuary, 2017

XSEDE
Extreme Science and Engineering Discovery Environment

Overview

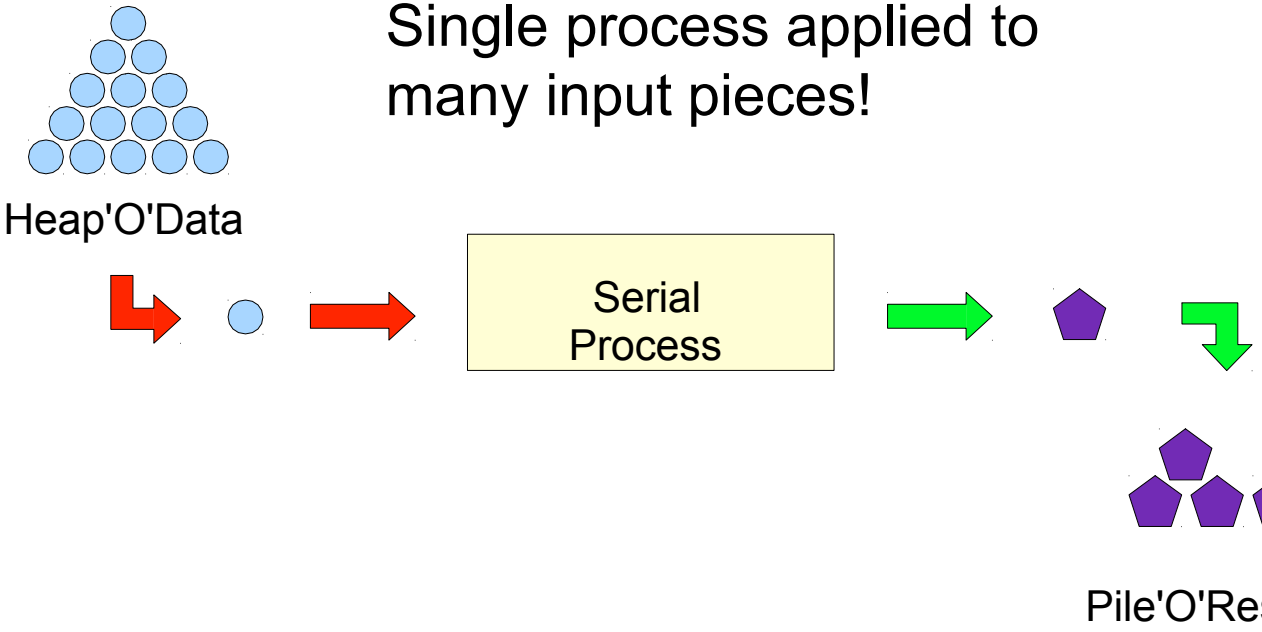
- Dilemma of running serial jobs on modern clusters
- GNU PARALLEL
  - Introduction
  - Tool flags
  - Examples

LSU/LONI HPC Environment

- Clusters, such as MIKE/SMIC/QB2, designed to run large-scale parallel tasks.
- Full nodes assigned - access to 8, 16, 20, or even 48 cores per node, depending on systems.

Q: How to handle thousands of serial (1-core) tasks without going crazy?

Problem Schematic



Generates many output

Job submission examples used by some users

Running blastp with one core

blast-1core.qsub1

```
#! /bin/bash
#PBS -A hpc_hpcadmin3
#PBS -l nodes=1:ppn=1
#PBS -l walltime=00:20:00
#PBS –q single
#PBS –N blastp-worst

export DIR=/project/$USER/distribution_workload/blast
cd $DIR;
blastp –query data/input1.faa –db db/xxx –out output/input1.out
```

blast-1core.qsub2

```
#! /bin/bash
#PBS -A hpc_hpcadmin3
#PBS -l nodes=1:ppn=1
#PBS -l walltime=00:20:00
#PBS –q single
#PBS –N blastp-worst

export DIR=/project/$USER/distribution_workload/blast
cd $DIR;
blastp –query data/input2.faa –db db/xxx –out output/input2.out
```

Need to submit N jobs given N inputs

Running blastp with one node

blast-1node.qsub

```
#! /bin/bash
#PBS -A hpc_hpcadmin3
#PBS -l nodes=1:ppn=16
#PBS -l walltime=00:20:00
#PBS –q workq
#PBS –N blastp-better

export DIR=/project/$USER/distribution_workload/blast
cd $DIR;
blastp –query data/input1.faa –db db/xxx –out output/input1.out  &
blastp –query data/input2.faa –db db/xxx –out output/input2.out  &
…
blastp –query data/input16.faa –db db/xxx –out output/input16.out  &

wait  #all the child processes to finish before terminating the parent process
```

- Only single node can be used
- Same command syntax with different inputs

Multi-Node Considerations

- The ***mother superior*** node is the only one holds all the job information, like environment variables, list of node names, etc.
- Start programs on other nodes with remote shell commands, like **ssh**.
- Assure all programs finish before script exits.

Running blastp with two nodes

blast-2nodes.qsub

```
#! /bin/bash
#PBS -A hpc_hpcadmin3
#PBS -l nodes=2:ppn=16
#PBS -l walltime=00:20:00
#PBS –q workq
#PBS –N blastp-2nodes
export DIR=/project/$USER/distribution_workload/blast
cd $DIR; mkdir output

# on compute node1(mother superior )
blastp –query data/input1.faa –db db/xxx –out output/input1.out &
…
blastp –query data/input8.faa –db db/xxx –out output/input8.out &
# start tasks on compute node2
ssh -n $HOST2 “cd $DIR; blastp –query data/input9.faa –db db/xxx –out
output/input9.out” &
…
ssh -n $HOST2 “cd $DIR; blastp –query data/input16.faa –db db/xxx –
out output/input16.out” &
wait  #all the child processes to finish before terminating the
parent process
```

What if using 10 nodes ??

Desired Solutions

- Avoid detailed scripting requirements - but allow flexibility and adaptability.
- Minimize customization and make parallelization automagically.
- Batch environment aware, particularly wallclock time constraints.

GNU Parallel

- A shell tool to execute independent jobs in parallel using one/more computers
- A job can be a single command or script
- Typical input could be a list of files, a list of parameters…
- Under the hood: the mother superior spawns ssh connections to each remote node/core, where an independent task is conducted

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

GNU Parallel installed on LSU/LONI clusters

- On SuperMike2:
  soft add +gnuparallel-20161022-gcc-4.4.6

- On QB2/SMIC:
  module load gnuparallel/20170122

```
$parallel --version
GNU parallel 20170122
Copyright (C) 2007,2008,2009,2010,2011,2012,2013,2014,2015,2016,2017
Ole Tange and Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
...
```

Serial Blast Job (run_blast.sh)

```
#!/bin/bash

export DATADIR=/project/$USER/GNU_PARALLEL/blast
blastp -query $1 -db $DATADIR/db/img_v400_PROT.00 –out $2  -outfmt 7
-max_target_seqs 10 -num_threads 1
```

```
 [wfeinste@mike421 blast]$ head input.lst
/project/wfeinste/GNU_PARALLEL/blast/data/test33.faa
/project/wfeinste/GNU_PARALLEL/blast/data/test21.faa
/project/wfeinste/GNU_PARALLEL/blast/data/test12.faa
...
```

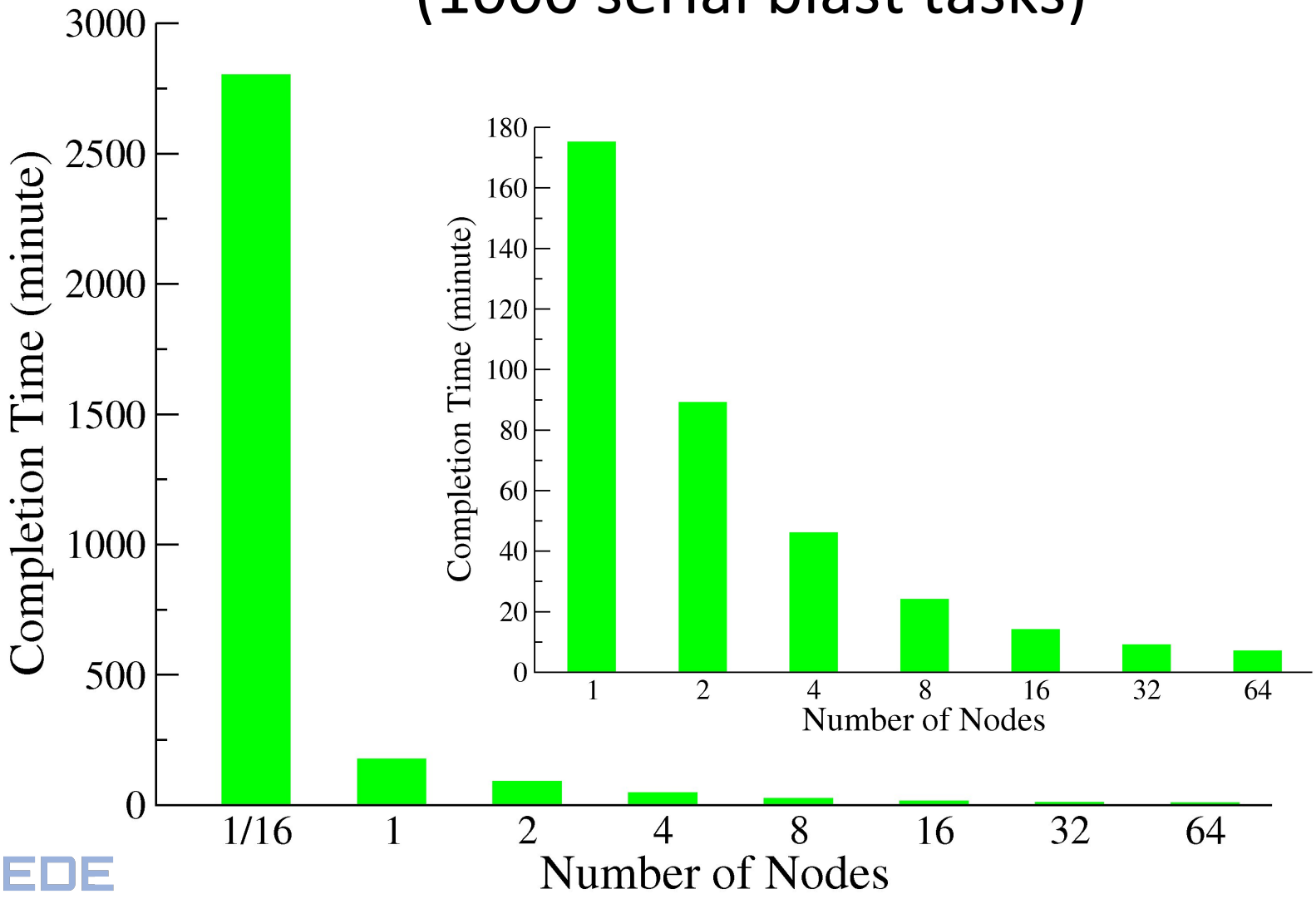
Distribute Serial tasks (blast_job.pbs)

```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -l walltime=1:00:00
#PBS -A hpc_hpcadmin3
#PBS -q workq
#PBS –N blast
#PBS –j oe

export WDIR=/project/$USER/GNU_PARALLEL/serial
cd $WDIR;
export JOBS_PER_NODE=16
# parallel command flags
PARALLEL="parallel -j $JOBS_PER_NODE --slf $PBS_NODEFILE --wd $WDIR
          --joblog logs/runtask.log --resume"
# gnu-parallel launch serial tasks
$PARALLEL –a input.lst sh run_blast.sh {} output/{/.}.out
```

```
[wfeinste@mike421 blast]$ head input.lst
/project/wfeinste/GNU_PARALLEL/blast/data/test33.faa
/project/wfeinste/GNU_PARALLEL/blast/data/test21.faa
/project/wfeinste/GNU_PARALLEL/blast/data/test12.faa
...
```

GNU Parallel Performance (1000 serial blast tasks)



GNU Parallel Syntax

- Reading command arguments on the command line:
  - `parallel [OPTIONS] COMMAND {} ::: TASKLIST`
- Reading command arguments from an input file:
  - `parallel [OPTIONS] COMMAND {} :::: TASKLIST.LST`
  - `parallel –a TASKLIST.LST [OPTIONS] COMMAND {}`

TASKLIST from command line

parallel [OPTIONS] COMMAND {} ::: TASKLIST

1) parallel echo {} ::: A B
```
A
B
```

2) Parallel echo {1} {2} ::: A B ::: C D
```
A C
A D
B C
B D
```

3) parallel --link echo {1} {2} ::: A B ::: C D
```
A C
B D
```

4) Parallel --link app-xxx {1} {2} ::: a1 a2 ::: b1 b2
```
app-xx a1 b1
app-xx a2 b2
```

TASKLIST From a File

- **parallel –a TASKLIST COMMAND {}**
- **parallel [OPTIONS] COMMAND {} :::: TASKLIST**

```
$cat csv-file.csv
 1,input1.txt
 2,input2.txt
```

Examples:

1) parallel –a csv-file.csv echo {}

```
1,input1.txt
2,input2.txt
```

2) Parallel --colsep '\,' –a csv-file.csv echo {1}

```
1
2
```

3) Parallel --colsep '\,' –a csv-file.csv echo {1} {2}

```
1   input1.txt
2   input2.txt
```

Manipulate Input String

```
Default string {}
parallel echo {} ::: path/input1.fas      --->path/input1.fas

Remove extension {.}
parallel echo {.} ::: path/input1.fas     --->path/input1

Remove path {/}
parallel echo {/} ::: path/input1.fas     --->input1.fas

Remove path and extension {/.}
parallel echo {/.} ::: path/input1.fas    --->input1

Change extension and path
parallel echo output/{/.}.out ::: path/input1.fas
                                    ---> output/input1.out
```

GNU Parallel Options/Flags

```
parallel –a input.lst [OPTIONS] COMMAND {}

Options:--jobs
        --slf
        --wd
        --progress
        --joblog --resume
        --timeout
        …
```

--jobs (-j)

- **--jobs N (-j N)**
  - Number of jobs/per machine (node). Run up to N jobs in parallel.
  - 0 means as many as possible. Default is 100% which will run one job per CPU core on each machine.
  - On HPC/LONI clusters, **N** is number of jobslots per node.
  - Make sure you use GNU Parallel version **>=20161022**
- **-j +/-N**

  CPU cores +/- N jobs on each node.

--slf (--sshloginfile)

--slf nodefile (--sshloginfile $PBS_NODEFILE)

used when more than one nodes is requested for a job

```
cat $PBS_NODEFILE
```

```
mike421
mike421
...
mike421
```

16 cores on mike421

```
mike429
mike429
...
mike429
```

16 cores on mike429

--wd (--workkdir)

- Default working dir on remote machines is login ~
- Can be changed by --wd
  e.g. --wd $PBS_O_WORKDIR

--joblog

- Generate a log file of each completed sub-tasks
- Used as checkpoints to resume unfinished tasks (--resume)
- Identify failed jobs

Seq	Host	Starttime	JobRuntime	Send	Receive	Exitval	Signal	Command
1	qb068	1487624896.425	256.295	0	65990	0	0	sh run_namd.sh /project/wfeinste/GNU_PARALLEL/MPI/input/ubq_ws_eq.conf 8
3	qb068	1487624896.435	256.294	0	65832	0	0	sh run_namd.sh /project/wfeinste/GNU_PARALLEL/MPI/input/ubq_ws_eq2.conf 8
2	qb058	1487624896.430	259.486	0	65849	0	0	sh run_namd.sh /project/wfeinste/GNU_PARALLEL/MPI/input/ubq_ws_eq1.conf 8
.....								

--resume

--resume --joblog job.log

- Allows GNU Parallel to resume a job where it is left out due to failure, time out and etc.
- If you need to rerun a GNU Parallel job from the start, be sure to delete old job.log

--progress

Show overall computational progress

```
Computers / CPU cores / Max jobs to run
1:mike136 / 16 / 16
2:mike264 / 16 / 16

Computer:jobs running/jobs completed/%of started jobs/Average seconds to complete
mike136:16/0/50%/0.0s  mike264:16/0/50%/0.0s using input: ../blast/data/test1.faa  output/test1.out
mike136:16/1/51%/31.0s  mike264:16/0/48%/0.0s using input: ../blast/data/test2.faa  output/test2.out
mike136:16/1/50%/33.0s  mike264:16/1/50%/33.0s using input: ../blast/data/test3.faa  output/test3.out
```

--timeout secs

- A job gets terminated if the command runs longer than xxx seconds.
- Useful if you know the command should have failed if running longer than a threshold.

Options to limit resources

- To avoid overloading systems GNU parallel can look at the system load before starting another job
  e.g.: parallel **--load** 100% echo load is less than {} job per cpu ::: 1

- Check if the system is swapping
  e.g.: parallel **--noswap** echo the system is not swapping ::: now

- Using **--memfree** to check if there is enough memory free. The youngest job if the memory free falls below 50%. The killed job will put back on the queue and retried later.
  e.g. parallel --memfree 1G echo will run if more than 1 GB is ::: free

Three Examples

- **Serial job (blastp using single core)**

```
blastp –query input.fna –db db/xxx –out output/
input.out -num_threads 1
```

- **Multiple-threaded job (blastp using multiple cores)**

```
blastp –query input.fna –db db/xxx –out output/
input.out -num_threads 4
```

- **MPI jobs (namd)**

```
mpirun –np 2 `which namd2` input.conf
```

Serial Example

- **Serial job (blastp using single core)**

```
blastp –query input.fna –db db/xxx –out output/
input.out -num_threads 1
```

- Multiple-threaded job (blastp using multiple cores)

```
blastp –query input.fna –db db/xxx –out output/
input.out -num_threads 4
```

- MPI jobs (namd):

```
mpirun –np 2 `which namd2` input.conf
```

Serial Blast Job (serial/run_blast.sh)

```
#!/bin/bash

export DATADIR=/project/$USER/GNU_PARALLEL/blast

blastp -query $1 -db $DATADIR/db/img_v400_PROT.00 –out $2  -outfmt 7
-max_target_seqs 10 -num_threads  1
```

```
 [wfeinste@mike421 blast]$ head input.lst
/project/wfeinste/GNU_PARALLEL/blast/data/test33.faa
/project/wfeinste/GNU_PARALLEL/blast/data/test21.faa
/project/wfeinste/GNU_PARALLEL/blast/data/test12.faa
...
```

Distribute Serial tasks (serial/blast_job.pbs)

```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -l walltime=1:00:00
#PBS -A hpc_hpcadmin3
#PBS -q workq
#PBS –N blast
#PBS –j oe
export WDIR=/project/$USER/GNU_PARALLEL/serial
cd $WDIR;
export JOBS_PER_NODE=16
# parallel command options
PARALLEL="parallel -j $JOBS_PER_NODE --slf $PBS_NODEFILE
          --wd $WDIR --joblog logs/runtask.log --resume"
# gnu-parallel launch serial tasks
$PARALLEL –a input.lst sh run_blast.sh {} output/{/.}.out
```

```
[wfeinste@mike421 blast]$ head input.lst
/project/wfeinste/GNU_PARALLEL/blast/data/test33.faa
/project/wfeinste/GNU_PARALLEL/blast/data/test21.faa
/project/wfeinste/GNU_PARALLEL/blast/data/test12.faa
...
```

Steps to for Testing

It is VERY important to test your scripts before submitting hundreds of serial tasks

1. Test your serial task
2. Test the parallel job interactively

Test Serial Script (Step 1)

```
#!/bin/bash

export DATADIR=/project/$USER/GNU_PARALLEL/blast


blastp -query $1 -db $DATADIR/db/img_v400_PROT.00 –out $2  -outfmt 7 -max_target_seqs
10 -num_threads 1
```

Step 1:

blastp –query $DATADIR/data/test1.faa -db $DATADIR/db/img_v400_PROT.
00 –out $WORKDIR/output/test1.out -outfmt 7 -max_target_seqs 10
-num_threads 1

Test GNU Parallel (Step 2)

```
#PBS …
export WDIR=/project/$USER/GNU_PARALLEL/serial
cd $WDIR;
export JOBS_PER_NODE=16
# parallel command options
PARALLEL="parallel -j $JOBS_PER_NODE --slf $PBS_NODEFILE --wd $WDIR
            --joblog logs/task.log --resume"
# gnu-parallel launch serial tasks
$PARALLEL –a input.lst sh run_blast.sh {} output/{/.}.out
```

Step 2:

- Request an interactive node

  qsub –I –A xxx –l nodes=1:ppn=16 –l walltime=2:00:00
- Interactively run above cmd line by line

Multi-thread Example

- Serial job (blastp using single core)

```
blastp –query input.fna –db db/xxx –out output/
input.out -num_threads 1
```

- **Multiple-threaded job (blastp using multiple cores)**

```
blastp –query input.fna –db db/xxx –out output/
input.out -num_threads 4
```

- MPI jobs (namd):

```
mpirun –np 2 `which namd2` input.conf
```

Multi-thread Blast Task (multi_threads/run_blast.sh)

```
#!/bin/bash
echo "using input: $1 $2 $3"
export DATADIR=/project/$USER/GNU_PARALLEL/blast

blastp -query $1 -db $DATADIR/db/img_v400_PROT.00 –out $2  -outfmt 7
-max_target_seqs 10 -num_threads $3
```

Distribute multi-thread tasks (multi_threads/blast_job.pbs)

```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -l walltime=1:00:00
#PBS -A hpc_hpcadmin3
#PBS -q workq
#PBS –N blast-mt-parallel
#PBS –j oe
export WDIR=/project/$USER/GNU_PARALLEL/multi_threads
cd $WDIR;
export JOBS_PER_NODE=8
export NTHREADS=2
# parallel command options
PARALLEL="parallel -j $JOBS_PER_NODE --slf $PBS_NODEFILE --wd
          $WDIR --joblog logs/runtask.log --resume"
# gnu-parallel launch serial tasks
$PARALLEL –a input.lst sh run_blast.sh {} output/{/.}.out $NTHREADS\
```

MPI Example

- Serial job (blastp using single core)

```
blastp –query input.fna –db db/xxx –out output/input.out -num_threads 1
```

- Multiple-threaded job (blastp using multiple cores)

```
blastp –query input.fna –db db/xxx –out output/input.out -num_threads 4
```

- **MPI jobs (namd)**

```
mpirun –np 2 `which namd2` input.conf
```

Single MPI NAMD Task (MPI/run_namd.sh)

```
#!/bin/bash
echo "$1 $2 $3"

mpirun -np $2 `which namd2`  $1 > $3
```

Distribute MPI Jobs (MPI/namd_job.pbs)

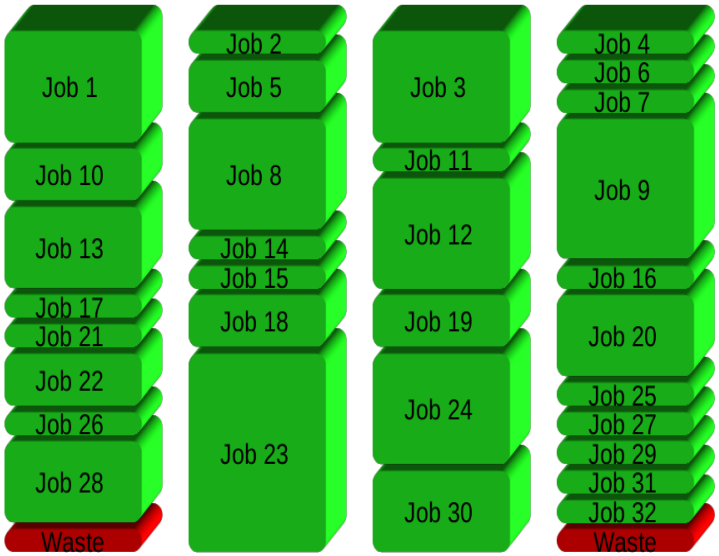
```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -l walltime=1:00:00
#PBS -A hpc_hpcadmin3
#PBS -q workq
#PBS –N blast
#PBS –j oe
export WDIR=/project/$USER/GNU_PARALLEL/MPI
cd $WDIR;
export JOBS_PER_NODE=2
export NPROCS=8        # launch 8 MPI processes/task
# parallel command options
PARALLEL="parallel -j $JOBS_PER_NODE --slf $PBS_NODEFILE --wd $WDIR
          --joblog logs/runtasks.log --resume"
# gnu-parallel launch serial tasks
$PARALLEL –a input.lst sh run_namd.sh {} $NPROCS output/{/.}.log
```

```
[wfeinste@mike421 blast]$ cat input.lst
/project/wfeinste/GNU_PARALLEL/MPI/input/ubq_ws_eq.conf
/project/wfeinste/GNU_PARALLEL/MPI/input/ubq_ws_eq1.conf
/project/wfeinste/GNU_PARALLEL/MPI/input/ubq_ws_eq2.conf
/project/wfeinste/GNU_PARALLEL/MPI/input/ubq_ws_eq3.conf
….
```

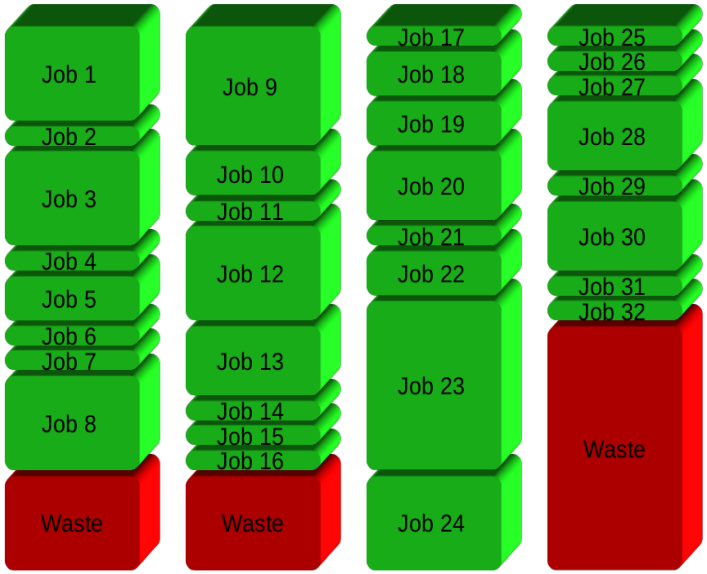
Load Balancing

- GNU Parallel starts next job once the previous finishes
- CPUs are kept busy to the max level

With Load Balancing



No Load Balancing



Conclusions

- GNU Parallel is an effective tool and easy to use
- Parallel independent jobs, often with different parameters
  - Serial tasks (1-core)
  - Multi-thread tasks (multiple cores)
  - MPI jobs
- Be wise on how many nodes to request
  - Too many ssh connections from the mother superior
  - Too many CPUs may become idle to wait for the last a few tasks to finish
- Thorough testing before start any production runs

Future Trainings

1. February 22, 2017: Parallel Serial Jobs Using GNU Parallel
2. March 8, 2017: Introduction to R
3. March 15, 2017: Introduction to Python
4. March 22, 2017: Parallel computing with R
5. March 29, 2017: Intermidiate Python Programming
6. April 5, 2017: Machine Learning in HPC Environments

http://www.hpc.lsu.edu/training/tutorials.php#upcoming