Many Tasks — Many Node Allocations using Slurm and GNU Parallel

GNU Parallel is the ideal tool for executing a custom program with varied parameters and utilizing the HPC.

The goal of GNU Parallel is to run the same program on a pool of different parameters. The pool of parameters can be larger than the number of core in the infrastructure.

For example, training and testing a neural network with different number of layers, running a simulator on different initial condition files, mathematical optimization ....

Useful Links

- GNU Parallel: Learn about the powerful GNU Parallel tool.
- Slurm Scheduler: Explore the Slurm scheduler documentation.
- Job Arrays: Understand the concepts and usage of Slurm job arrays.
- HPC Management of Sequential and Embarrassingly Parallel Jobs: Gain insights into managing sequential and embarrassingly parallel jobs on HPC.
- Lmod: Discover Lmod, a tool for managing the software environment on HPC clusters.
- NERSC Documentation: Read the NERSC documentation on the scalability of parallel tasks with SSH login.

Step 1: Multi-core code in one node

There command you can do from any machine in iris or aion.

Your experience code takes one input. The syntax is `${1}` in bash. In programming language, you may access the input parameter with `sys.argv[1]` in python, `argv[1]` in C/C++, ...

experiment.sh, this script emulates a task to run:

```
#!/bin/sh
echo "Running your experiment with parameter ${1} ..."
sleep 3 # emulate a long run time
echo "Experiment ${1} finished at $(date)"
```

The script below run the code on 5 inputs as input: "a", "b", "c", "d" and "e".

```
#/bin/sh
export experiment=${PWD}/experiment.sh
parallel -j 2 $experiment {} ::: a b c d e f
```

Breaking down the command: - **parallel:** The command itself, installed in your ULHPC. - **experiment**: Simulates your Bash script. Keeps aware you can run any software and is not reserved to bash scripts. - **$experiment {}**: The program to run, where `{}` is the placeholder for one parameter. - **-j 2**: Executes two parallel tasks using GNU Parallel. Each task corresponds to one parameter. - This key word **:::** indicates parameters are inlined. Note the distinction between `:::` and `::::`. `::: a b c d e f` provides different inputs to the `experiment` program. For sending files via meta characters such as `./input/*`, use `::::` instead.

The output:

```
Running your experiment with parameter a ...
Experiment a finished at Wed 15 Nov 17:07:08 CET 2023
Running your experiment with parameter b ...
Experiment b finished at Wed 15 Nov 17:07:08 CET 2023
Running your experiment with parameter c ...
Experiment c finished at Wed 15 Nov 17:07:11 CET 2023
Running your experiment with parameter d ...
Experiment d finished at Wed 15 Nov 17:07:11 CET 2023
Running your experiment with parameter e ...
Experiment e finished at Wed 15 Nov 17:07:14 CET 2023
Running your experiment with parameter f ...
Experiment f finished at Wed 15 Nov 17:07:14 CET 2023
```

Concluding remark: two jobs are effectively done simultaneously: "a" and "b" at 14:16:59, "c" and "d" finished at 14:17:02, due to the `-j 2` option. It effectively uses the multi-core capability of the CPU.

Step 2 : Multi-node Multi-core code

First we need the code we need to distribute. There the example of a bash script, but again, it can be done with any programming language. experiment.sh:

```
#!/bin/sh
echo "Running your experiment with parameter ${1} ..."
sleep 3 # emulate a long run time
echo "Experiment ${1} finished at $(date)"
```


slurm_parallel_launcher.sh:

```
#!/bin/sh -l
#SBATCH -c 30 # How many cores to use in 1 single node ?
#SBATCH -N 3 # How many nodes ?
#SBATCH -t 1
#SBATCH --export=ALL


# get host name
hosts_file="hosts.txt"
scontrol show hostname $SLURM_JOB_NODELIST > $hosts_file


# Collect public key and accept them
while read -r node; do
    ssh-keyscan "$node" >> ~/.ssh/known_hosts
done < "$hosts_file"

experiment=${PWD}/experiment.sh

# Run. The -j option controls how many experiments run in each node (they will share the
# The number of experiments is given by N*j.
parallel  --sshloginfile  $hosts_file  -j 2 $experiment {} ::: a b c d e f g h i j k l m
```

Finally, you can launch this script by doing:

```
[ppochelu@access1 ~]$ sbatch ./slurm_parallel_launcher.sh
```

Finally, you can check the SLURM output the result of this:

```
Running your experiment with parameter e ...
Experiment e finished at Wed 15 Nov 12:21:20 CET 2023
Running your experiment with parameter b ...
Experiment b finished at Wed 15 Nov 12:21:20 CET 2023
Running your experiment with parameter c ...
Experiment c finished at Wed 15 Nov 12:21:20 CET 2023
Running your experiment with parameter f ...
Experiment f finished at Wed 15 Nov 12:21:20 CET 2023
Running your experiment with parameter a ...
Experiment a finished at Wed 15 Nov 12:21:20 CET 2023
Running your experiment with parameter d ...
Experiment d finished at Wed 15 Nov 12:21:20 CET 2023
Running your experiment with parameter g ...
Experiment g finished at Wed 15 Nov 12:21:24 CET 2023
Running your experiment with parameter h ...
Experiment h finished at Wed 15 Nov 12:21:24 CET 2023
Running your experiment with parameter i ...
Experiment i finished at Wed 15 Nov 12:21:24 CET 2023
Running your experiment with parameter j ...
[...]
```

Check the SLURM output to see the results. The order of finished tasks is not guaranteed, and six tasks are done in parallel (two tasks per node, utilizing three nodes).

Case study: Word Count on multiple files

Assuming we want to count the number of words in all files in the directory `./input`. The words will be aggregated in the `output.txt` file.

You may download the input files here:

Download 1.txt, Download 2.txt, Download 3.txt, Download 4.txt, Download 5.txt, Download 6.txt. Download 7.txt, Download 8.txt, Download 9.txt, Download 10.txt, Download 11.txt, Download 12.txt.

Launch the program using the ":::" syntax to scan all files in the `input` directory:

```
#!/bin/sh
wc -w ${1} >> ${2}
```

slurm_parallel_launcher.sh:

```
#!/bin/sh -l
#SBATCH -c 30 # How many cores to use in 1 single node ?
#SBATCH -N 3 # How many nodes ?
#SBATCH -t 1
#SBATCH --export=ALL

# get host name
hosts_file="hosts.txt"
scontrol show hostname $SLURM_JOB_NODELIST > $hosts_file

# Collect public key and accept them
while read -r node; do
    ssh-keyscan "$node" >> ~/.ssh/known_hosts
done < "$hosts_file"

experiment=${PWD}/experiment.sh
outfile=${PWD}/output.txt

parallel  --sshloginfile  $hosts_file  -j 2 $experiment {} $outfile ::: ${PWD}/input/*.tx
```

Finally, you can launch this script by doing:

```
[ppochelu@access1 ~]$ sbatch ./slurm_parallel_launcher.sh
```

After the process is scheduled and finished, the result looks like:

```
[ppochelu@access1 ~]$ cat output.txt
78 ~/input/11.txt
75 ~/input/2.txt
158 ~/input/3.txt
```

Case study: Python (Hyper)-parameter Search

Code is organized here in 2 files python files.

parameters.py for generating (hyper)-parameters:

```
stepx=0.2
minx=-2.0
maxx=2.0
stepy=0.2
miny=-2.0
maxy=2.0

parameters=[]
x=minx
y=miny
while x<maxx:
    while y<maxy:
        param=(x,y)
        parameters.append( param )
        y+=stepy
    x+=stepx
params_str=[]
for x,y in parameters:
    param_str=str(round(x,3))+","+str(round(y,3))
    params_str.append(param_str)
output="\n".join(parameters)
print(output) # the output can be re-redirected to another program
```

The Grid Search samples 40 variations in the x-dimension and 40 in the y-dimension, it made 1600 experiments to perform.

experiment.py

```
import sys
import time
param_str=sys.argv[1]
x_str, y_str = param_str.split(",")
x=float(x_str)
y=float(y_str)
score=(x-1)**2 + b*(y-x**2)**2
time.sleep(3) # simulates longer execution time
return round(score, 3), x, y
```

slurm_parallel_launcher.sh:

```
#!/bin/sh -l
#SBATCH -c 100 # How many cores to use in 1 single node ?
#SBATCH -N 3 # How many nodes ?
#SBATCH -t 5
#SBATCH --export=ALL

hosts_file="hosts.txt"
scontrol show hostname $SLURM_JOB_NODELIST > $hosts_file

while read -r node; do  # collect and accept ssh keys
    ssh-keyscan "$node" >> ~/.ssh/known_hosts
done < "$hosts_file"

PARAMS=$(python3 parameters.py) # Store in PARAMS the experiments pool
experiment_py=${PWD}/experiment.py
output=${PWD}/output.txt
parallel  --sshloginfile  $hosts_file  -j 100 python3 $experiment_py >> $output  {}  :::
```

Each node is responsible for 100 tasks at the same time. There are 3 nodes. The HPC will perform batch of 300 at the same.

Finally, you can launch this script by doing:

```
[ppochelu@access1 ~]$ sbatch ./slurm_parallel_launcher.sh
```

The running time will take approximatively (1600 tasks // 300 tasks in parallel) * 3 seconds per task = 18 seconds