Introduction to High Performance Computing

Alejandro Cárdenas-Avendaño

Serial Jobs

Serial programming

Sometimes your code is serial, meaning it only runs on a single processor, and that's as far as it's going to go. There are several reasons why we might not push the code farther:

- The problem involves a parameter study; each iteration of the parameter study is very swift; each iteration is independent; but many many need to be done.
- The algorithm is inherently serial, and there's not much that can be done about it.
- You're running a commercial code, and don't have the source code to modify. You're graduating in six months, and don't have time to parallelize your code.

Sometimes you just have to let your code be serial, and run the serial processes in parallel. That's the topic of today's class.

What are our assumptions?

Let us assume the following situation: You have a serial code.

- Your code takes a set of parameters, either from a file or (preferably) from the command line.
- The code runs in a reasonably short amount of time (minutes to hours).
- You have a large parameter space you want to search, which means hundreds or thousands of combinations of values of parameters.
- You'd probably like some feedback on your jobs, things like error checking, fault tolerance, etc.
- You want to run your code on a cluster or our WorkStation.

How do we go about performing this set of calculations efficiently?

What are your options?

So how might we go about running multiple instances the code (subjobs) simultaneously?

- Write a script from scratch which launches and manages the subjobs.
- If the code is written in Python, you could use ipython notebook to manage the subjobs.
- If the code is written in R, you could use the parallel R utilities to manage the subjobs.
- Use an existing script, such as GNU Parallel1, to manage the subjobs.

We'll discuss the first and last options in this lecture.

Hyperthreading: Logical CPUs vs. cores

- Hyperthreading, a technology that leverages more of the physical hardware by pretending there are twice as many logical cores than real once.
- So the OS and scheduler see 80 logical cores.
- 80 logical cores vs. 40 real cores typically gives about a 5-10% speedup (YMMV).

Running multiple serial codes

If your subjobs all take the same amount of time, there's nothing in principle wrong with this submission script.

- Does it use all the cores? Yes.
- Will any cores be wasting time not running? Not if all the subjobs take the same amount of time.

But if your subjobs take variable amounts of time this approach isn't going to work.

```
# The first line called a hashbang or shebang. It
tells Unix that this script should be run through
the /bin/bash shell.

#!/bin/bash
g++ gslrootparams.cpp -lgsl -o root &
./root -4.0 5.0 &
./root -40.0 5.0 &

# Tell the script to wait, or all the subjobs get
killed immediately
wait

# After saving the above file, we need to give it
execute permission to make it runnable.

# $ chmod +x jobs.sh

# Execute script as
# $ ./jobs.sh

# Or simply like
# $ batch jobs.sh
```

Running multiple serial codes

Suppose I can only fit 2 subjobs simultaneously on a node.

What's wrong with this approach?

- Reinventing the wheel,
- More code to maintain/debug,
- No load balancing,
- No job control,
- No error checking,
- No fault tolerance,
- No multi-node jobs.

```
#!/bin/bash

g++ gslrootparams.cpp -lgsl -o root &

function dobytwo() {
   while [ -n "$1" ]
   do
   ./root $1 5 &
   ./root $2 5 &
   wait
   shift 2 # removes arguments from the
beginning of the argument list
   done
}

dobytwo $(seq -10 -3)
```

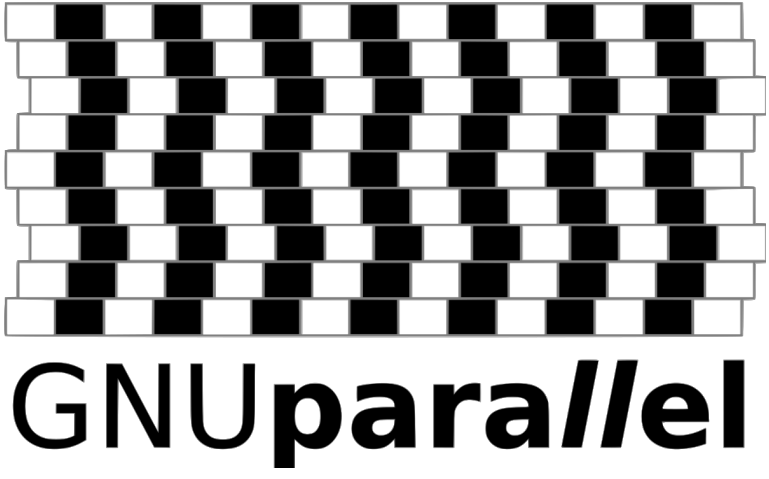
Imbalance case

```
#!/bin/bash

g++ gslrootparamsdiffTime.cpp -lgsl -o
root &

function dobytwo() {
   while [ -n "$1" ]
   do
   ./root -4.0 5 $1 &
   ./root -4.0 5 $2 &
   wait
   shift 2 # removes arguments from the
beginning of the argument list
   done
}

dobytwo $(seq 1000000 1000000 4000000)
```

GNU Parallel

```
conda install -c conda-forge parallel
```

GNU parallel solves the problem of managing blocks of subjobs of differing duration.



- Basically a perl script.
- But surprisingly versatile, especially for text input.
- Gets your many cases assigned to different cores and on different nodes without much hassle.
- Invoked using the ”parallel” command.

1. O. Tange, “GNU Parallel - The Command-Line Power Tool” ;login: 36 (1), 42-47 (2011)
2. http://www.gnu.org/software/parallel/parallel_tutorial.html

GNU parallel example

Notes about our example:

- The “-j” flag indicates you wish GNU parallel to run N subjobs at a time.
- If you can’t fit N subjobs onto a node due to memory constraints, specify a different value for the ”-j” flag.
- Put all the commands for a given subjob onto a single line.

```
#!/bin/bash

g++ gslrootparams.cpp -lgsl -o root &

parallel -j 2 <<EOF
./root -4.0 5.0; echo "job 1 done"
./root -4.0 5.0; echo "job 2 done"
./root -4.0 5.0; echo "job 3 done"
./root -4.0 5.0; echo "job 4 done"
EOF
```

GNU Parallel, continued

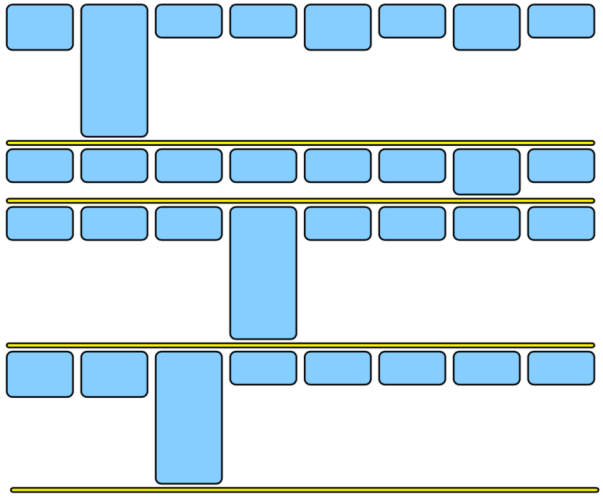
What does GNU parallel do?

- GNU parallel assigns subjobs to the processors.
  - As subjobs finish it assigns new subjobs to the free processors.
  - It continues to do assign subjobs until all subjobs in the subjob list are assigned.
- Consequently there is built-in load balancing!
- You can use more than N subjobs per node (hyperthreading).
- You can use GNU parallel across multiple nodes as well.
- It can also log a record of each subjob, including information about subjob duration, exit status, etc.

If you're running blocks of serial subjobs, just use GNU parallel!

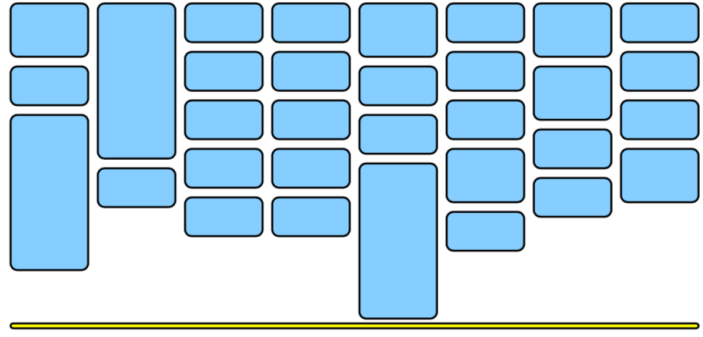
Backfilling

Without GNU parallel:



17 hours
42% utilization

With GNU parallel:



10 hours
72% utilization

GNU parallel example 2

Sometimes it's easier to just create a list that holds all of the subjob commands.

subjobs

~/Dropbox/Konrad/Docencia/HPC/Lectures/Lecture VII/Codes/subjobs

```
1  ./root -4.0 5.0; echo "job 1 done"
2  ./root -4.0 5.0; echo "job 2 done"
3  ./root -4.0 5.0; echo "job 3 done"
4  ./root -4.0 5.0; echo "job 4 done"
```

Use the --no-run-if-empty flag to indicate that empty lines in the subjob list file should be skipped.

```
#!/bin/bash

g++ gslrootparams.cpp -lgsl -o root &

parallel -j 2 --no-run-if-empty < subjobs
```

GNU parallel syntax

Some commonly used arguments for GNU parallel:

- --jobs NUM, sets the number of simultaneous subjobs. By default parallel uses the number of virtual cores (24 on our WorkStation). Same as -j N.
- --joblog LOGFILE, causes parallel to output a record for each completed subjob.
- The records contain information about subjob duration, exit status, and other goodies.
- --resume, when combined with --joblog, continues a full GNU parallel job that was killed prematurely.
- --pipe, splits stdin into chunks given to the stdin of each subjob.

If you're running blocks of serial subjobs, just use GNU parallel!

GNU Parallel, more options

GNU parallel has a tonne of optional arguments. We've barely scratched the surface.

- There are specialized ways of passing in combinations of arguments to functions.
- There are ways to modify arguments to functions on the fly.
- There are specialized ways of formatting output.
- Review the man page for parallel, or review the program's webpage, for a full list of options.

If you're running blocks of serial subjobs, just use GNU parallel!

Summary on Serial Jobs

- Be aware of the features of your code, and the details of the hardware where you will run it.
- If you need to run serial jobs on a cluster with multicore architecture, be sure to run them in batches, so as to use your nodes efficiently.
- Unless your jobs all take the same amount of time, don’t try to write your own serial-job management code.
- Use GNU-Parallel to manage your serial jobs.

References

- SciNet Education (https://support.scinet.utoronto.ca/education/help/help_about.php)
- International HPC Summer School 2018 Lectures
- ARCHER UK National Supercomputing Service Training Courses (http://www.archer.ac.uk/training/)
- Muna, D & Price-Whelan, A. SciCoder Workshop. scicoder.org
- Peter Norvig, “What to demand from a Scientific Computing Language”, Mathematical Sciences Research Institute, 2010.