High-Performance Scientific Computing: Running serial jobs in parallel

Erik Spence

SciNet HPC Consortium

12 March 2015

SciNet compute • calcul CANADA

Today's class

Today we will discuss the following topics:

- Approaches for dealing with serial jobs.
- Mini-intro to SciNet.
- GNU parallel.

SciNet compute • calcul CANADA

Why Parallel Programming?

To review:

- Your desktop has many cores, as do the nodes on SciNet's GPC (General Purpose Cluster).
- Your code would run alot faster if it could use all of those cores simultaneously, on the same problem.
- Or even better, use cores on many nodes simultaneously.
- So we need to adjust our programming accordingly: thus parallel programming.

There are several different approaches to parallel programming.

SciNet compute • calcul CANADA

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.

SciNet compute • calcul CANADA

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 SciNet (GPC).

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

SciNet compute • calcul CANADA

What are SciNet's concerns?

What concerns does *SciNet* have about you running serial jobs on GPC?

- Scheduling is done by node. Each node comes with 8 cores and ~14Gb of available memory. Use your resources efficiently.
  - Use all of the processors on the nodes you've been given continuously (load balancing), or
  - use all the memory you've been given efficiently (if 8 instances of your serial job won't fit in memory).
  - This almost certainly means having multiple subjobs running simultaneously on your nodes.
- Don't do heavy I/O.
  - Don't try to read thousands of files.
  - Don't generate thousands of files.

Using resources efficiently makes sense for you. Not crashing the filesystem is everyone's responsibility.

SciNet compute • calcul CANADA

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 Parallel[1], to manage the subjobs.

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

[1] O. Tange (2011): GNU Parallel - The Command-Line Power Tool, ;login; The USENIX Magazine, February 2011:42-47.

SciNet compute • calcul CANADA

The General Purpose Cluster

SciNet compute • calcul CANADA

The General Purpose Cluster

- 3780 nodes with 2 x 2.53GHz quad-core Intel Xeon 5500 64-bit processors.
- 30,912 cores.
- 328 TFlops.
- 16 GB RAM per node ($\sim$14GB for user jobs).
- 16 threads per node.
- Operating system: CentOS 6.
- Interconnect: InfiniBand.
- #16 on the June 2009 *TOP500* (Now at #216).
- #2 in Canada.

SciNet compute • calcul CANADA

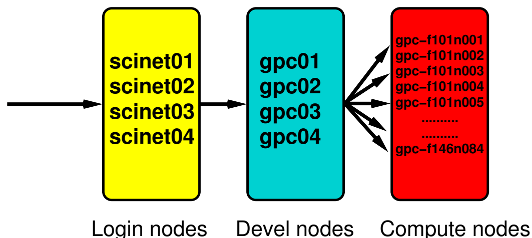
Mini-intro to SciNet - getting started

Accessing SciNet:

- You need to have an account to access SciNet.
- If you don't have an account, get one: (wiki.scinethpc.ca/wiki/index.php/Essentials)
- If you can't, for whatever reason, email us: (support@scinet.utoronto.ca).
- Before you start running on SciNet, please read the SciNet Tutorial and the GPC quick start on the wiki: (wiki.scinethpc.ca/wiki/index.php/GPC_Quickstart).
- If you ever run into trouble, start by searching the wiki. If that doesn't help, email us.

SciNet compute • calcul CANADA

Mini-intro to SciNet - accessing SciNet



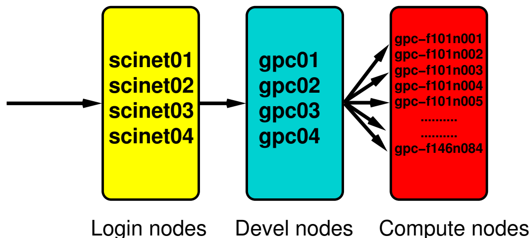
To access SciNet, first open a terminal and ssh into a login node (not part of the SciNet clusters):

```
ejspence@mycomp ~> ssh -X -l ejspence login.scinet.utoronto.ca
ejspence@scinet01-ib0 ~>
```

The login nodes are gateways. They are to be used

- for small data transfers.
- to log into one of the devel nodes.

Mini-intro to SciNet - working on SciNet



Once you have logged into a login node, ssh into a devel node. On GPC that means `gpc01 - gpc04`. These are aliases for longer node names.

```
ejspence@scinet01-ib0 ~>
ejspence@scinet01-ib0 ~> ssh -X gpc03
ejspence@gpc-f103n084-ib0 ~>
```

`gpc03` is an alias for the node named `gpc-f103n084`. All development work (editing, compiling, job submission, *etc.*) is done from the devel nodes.

SciNet compute • calcul CANADA

Mini-intro to SciNet - how to run

Once logged into a devel node, you can submit your jobs.

- Your jobs do not run on login nodes, or devel nodes.
- Your jobs run on compute nodes.
- Compute nodes are reserved for your use through a queuing system.
- You can get an interactive session on a compute node by making the following request to the queuing system.

```
ejspence@gpc-f103n084-ib0 ~> qsub -X -I -q debug \
 -l nodes=1:ppn=8,walltime=2:00:00
ejspence@gpc-f109n001-ib0 ~>
```

- This gives you a dedicated compute node, in the 'debug' queue, for two hours. There is a two hour limit when you use this queue.
- But usually you don't want an interactive session, you just want to submit a job to the queue. To do this you use a job script.

SciNet compute • calcul CANADA

Mini-intro to SciNet - how to run, continued

Be aware of read/write restrictions:

- The compute nodes, where your jobs actually run, do not have write access to your /home directory, only read access.
- Compute nodes can only write to your /scratch directory.
- You can get to your scratch directory thus:

```
ejspence@gpc-f103n084-ib0 ~>
ejspence@gpc-f103n084-ib0 ~> pwd
/home/s/scinet/ejspence
ejspence@gpc-f103n084-ib0 ~> cd $SCRATCH
ejspence@gpc-f103n084-ib0 .../s/scinet/ejspence>
ejspence@gpc-f103n084-ib0 .../s/scinet/ejspence> pwd
/scratch/s/scinet/ejspence
ejspence@gpc-f103n084-ib0 .../s/scinet/ejspence>
```

SciNet compute • calcul CANADA

Submitting jobs

To submit a job to the queue we use a submission (bash) script.

- Every line that starts with "#PBS" is ignored by bash (because it starts with #) but is read by the scheduler.
  - You must specify the requested number of nodes, including the "ppn=8" part.
  - You must specify the requested walltime (48 hours max).
- When the job starts, it runs the script, starting your /home directory.
- Don't forget to set any environment variables, or load any modules, your job might need.

```
#!/bin/bash
#PBS -l nodes=1:ppn=8
#PBS -l walltime=1:00:00
#PBS -N example-job

# cd to the directory whence
# the job was submitted.
cd $PBS_O_WORKDIR

# Run the code.
./mycode

# My submission script.
```

SciNet compute • calcul CANADA

Submitting your job

```
ejspence@gpc-f103n084-ib0 ~> g++ mycode.cpp -o mycode
```

```
ejspence@gpc-f103n084-ib0 ~> cp mycode $SCRATCH/example
```

```
ejspence@gpc-f103n084-ib0 ~> cd $SCRATCH/example
```

```
ejspence@gpc-f103n084-ib0 .../scinet/ejspence/example> cat submission.sh
#!/bin/bash
#PBS -l nodes=1:ppn=8,walltime=1:00:00
#PBS -N example-job
cd $PBS_O_WORKDIR
./mycode
```

```
ejspence@gpc-f103n084-ib0 .../scinet/ejspence/example> qsub submission.sh
2961985.gpc-sched-ib0
```

```
ejspence@gpc-f103n084-ib0 .../scinet/ejspence/example> qstat -u ejspence
```

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Req'd Memory	Req'd Time	S	Elap Time
2961985.gpc-sched-ib0	ejspence	batch	example-job	–	1	8	–	1:00:00	Q	–

```
ejspence@gpc-f103n084-ib0 .../scinet/ejspence/example> ls
 example-job.e2961985  example-job.o2961985  output.txt
 mycode                submission.sh
```

```
ejspence@gpc-f103n084-ib0 .../scinet/ejspence/example>
```

et compute • calcul CANADA

If you're not sure, wiki!

I've gone over this quickly, quite intentionally.

- There are lots of details that I've skipped over.
- If you haven't taken the "Intro to SciNet" class, I recommend you do.
- If you've never run on SciNet, read the SciNet User Tutorial and the GPC quickstart guide.
- Almost certainly your question is answered on the wiki.
- If all else fails, email us (support@scinet.utoronto.ca).

`http://wiki.scinethpc.ca`

SciNet compute • calcul CANADA

Now back to serial

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.

```
#!/bin/bash
#PBS -l nodes=1:ppn=8
#PBS -l walltime=1:00:00
#PBS -N serialx8

cd $PBS_O_WORKDIR

# Run the code on 8 cores.
(cd jobdir1; ../mycode) &
(cd jobdir2; ../mycode) &
(cd jobdir3; ../mycode) &
(cd jobdir4; ../mycode) &
(cd jobdir5; ../mycode) &
(cd jobdir6; ../mycode) &
(cd jobdir7; ../mycode) &
(cd jobdir8; ../mycode) &

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

Why not roll your own?

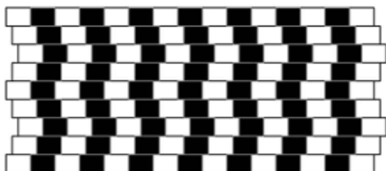
Suppose I can only fit 4 subjobs simultaneously on a node. What's wrong with this approach?

- 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
#PBS -l nodes=1:ppn=8,walltime=5:00:00
#PBS -N badserialx8

cd $PBS_O_WORKDIR

function dobyfour() {
  while[ -n "$1" ]
  do
    ./variable_time_code $1 &
    ./variable_time_code $2 &
    ./variable_time_code $3 &
    ./variable_time_code $4 &
    wait
    shift 4
  done
}

dobyfour $(seq 100)
```

GNU 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

SciNet compute • calcul CANADA

GNU parallel example

Notes about our example:

- Load the gnu-parallel module within your script.
- The "-j 8" flag indicates you wish GNU parallel to run 8 subjobs at a time.
- If you can't fit 8 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
#PBS -l nodes=1:ppn=8,walltime=1:00:00
#PBS -N gnu-parallelx8

cd $PBS_O_WORKDIR
module load gnu-parallel

# Run the code on 8 cores.
parallel -j 8 <<EOF
cd jobdir1; ../mycode; echo "job 1 done"
cd jobdir2; ../mycode; echo "job 2 done"
cd jobdir3; ../mycode; echo "job 3 done"
⋮
cd jobdir26; ../mycode; echo "job 26 done"
cd jobdir27; ../mycode; echo "job 27 done"
cd jobdir28; ../mycode; echo "job 28 done"
EOF
```

SCINet compute • calcul CANADA

GNU Parallel, continued

What does GNU parallel do?

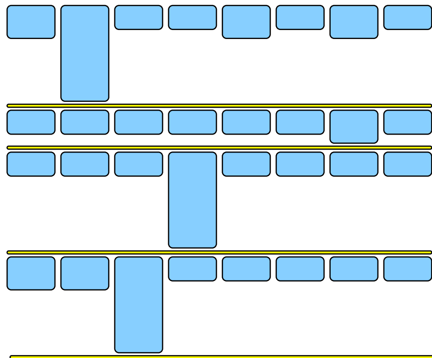
- GNU parallel assigns subjobs to the processors.
  - As subjobs finish it assigns new subjobs to the free processors.
  - It continues to do assign subjobs until all subjobs in the subjob list are assigned.
- Consequently there is built-in load balancing!
- You can use more than 8 subjobs per node (hyperthreading).
- 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!

SciNet compute • calcul CANADA

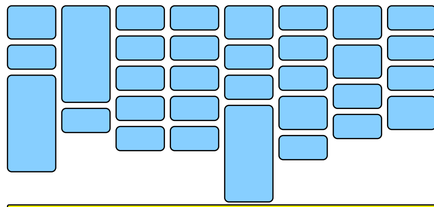
Backfilling

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.

```
ejspence@gpc-f103n084-ib0 ~>
ejspence@gpc-f103n084-ib0 ~> cat subjobs
cd jobdir1; ../mycode; echo "job 1 done"
cd jobdir2; ../mycode; echo "job 2 done"
cd jobdir3; ../mycode; echo "job 3 done"
⋮
cd jobdir26; ../mycode; echo "job 26 done"
cd jobdir27; ../mycode; echo "job 27 done"
cd jobdir28; ../mycode; echo "job 28 done"
ejspence@gpc-f103n084-ib0 ~>
```

```
#!/bin/bash
#PBS -l nodes=1:ppn=8
#PBS -l walltime=1:00:00
#PBS -N gnu-parallelx8

cd $PBS_O_WORKDIR
module load gnu-parallel

# Run the code on 8 cores.
parallel -j 8 \
  --no-run-if-empty < subjobs
```

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

SciNet compute • calcul CANADA

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 (16 on GPC nodes). 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.

GNU parallel multi-node example

Notes about using multiple nodes:

- By default parallel only knows about the head node. Tell it about the other nodes using `--sshloginfile $PBS_NODEFILE`.
- The non-head nodes also don't know where they should be working, thus `--workdir $PWD`.
- This setup will run 40 jobs simultaneously, and only makes sense if, say, several hundred jobs are in your subjobs file.
- Remember that the fewer nodes you request, the more likely your job is to run.

```
#!/bin/bash
#PBS -l nodes=5:ppn=8
#PBS -l walltime=1:00:00
#PBS -N multi-node

cd $PBS_O_WORKDIR
module load gnu-parallel

# Run the code on 5 nodes, 8 cores
# each (40 subjobs).
parallel --jobs 8 \
  --sshloginfile $PBS_NODEFILE \
  --joblog progress.log \
  --workdir $PWD < subjoblist
```

SciNet compute • calcul CANADA

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.

`wiki.scinethpc.ca/wiki/index.php/User_Serial`

SciNet compute • calcul CANADA

Summary

Today's take-home message.

- If you need to run serial jobs on GPC, 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.

SciNet compute • calcul CANADA