Task Parallelism

Warwick RSE

12/12/2022

Task parallelism

- Split up problem into separate tasks
  - Explore parameter space
  - Multiple Monte-Carlo realisations
- Run each task on a separate processor of your computer
  - Or separate computers

Task parallelism

- Scales very well
  - Only limit on scaling is number of tasks and number of processors
  - If you have one processor available per task then you can get all of your tasks done in the same time as one task
  - Slight wrinkle if each task is so quick that it takes as long to start the task as it does for it to run

GNU Parallel

- Installed on the clusters and the task farm
    - Will need the module “parallel” on the cluster systems
- Most other large systems have it installed as well and it can easily be installed on any Linux system
    - sudo apt-get install parallel
    - yum install parallel/dnf install parallel
    - brew install parallel

GNU Parallel

- Very good official tutorial at https://www.gnu.org/software/parallel/parallel_tutorial.html
- Idea is that you create a program that takes parameters through the command line and then tell parallel how to build command lines to run several copies at the same time
    - Put together program name, parameters etc.
- Has a number of job slots (usually the number of processors that you have)
    - Runs tasks in sequence on each job slot until it runs out of task

GNU Parallel

```
#!/bin/bash

parallel echo ::: A B C D E F
```

- Very simple Parallel script
  - "echo" is a command line utility that just prints its arguments

```
A
B
C
D
E
F
```

GNU Parallel

```
#!/bin/bash
parallel echo ::: A B C D E F
```

- "parallel" just runs the GNU parallel program
- "echo" here just prints the parameters that it is given but in general would be the program that you want to run in parallel (including any command line arguments that don't change)
- ":::" separates the command that you want to run from the parameters that you want to run it with
- The last bit is the list of parameters that you want to pass to your program separated by spaces (you can change the separator if you want)

GNU Parallel

```
#!/bin/bash

parallel echo ::: A B C ::: D E F ::: G H I
```

- Putting in multiple argument sources calls the program with every possible combination of the sources (Cartesian Product)

```
A D G
A D H
A D I
A E G
A E H
A E I
A F G
A F H
A F I
B D G
B D H
B D I
…
…
…
```

GNU Parallel

```
#!/bin/bash

parallel echo HELLO {} ::: A B C
```

- You can specify where to put the command line argument that Parallel generates using {}

```
HELLO A
HELLO B
HELLO C
```

GNU Parallel

```
#!/bin/bash

parallel echo {1} Says hello to {2} ::: A B C ::: D E F
```

- If you use multiple argument sources then you can choose which one to use using {number}

```
A Says hello to D
A Says hello to E
A Says hello to F
B Says hello to D
B Says hello to E
B Says hello to F
C Says hello to D
C Says hello to E
C Says hello to F
```

GNU Parallel

```
#!/bin/bash

parallel echo {1} is job number {#} ::: A B C
```

- You can get the job number for each parallel job using {#}. The job number goes up from 1 as each job is run

```
A is job number 1
B is job number 2
C is job number 3
```

GNU Parallel

```
#!/bin/bash

runfunc(){
  echo $1 has uuid `uuidgen`
}

export -f runfunc
parallel runfunc ::: A B C
```

- If you need a unique ID then you can use the uuidgen command line program to generate one
  - Your UUIDs will be different to these

```
A has uuid 536b2247-64fb-49cd-af9b-258ca52c6200
B has uuid fef2fce0-26fb-4553-9c5e-7a6c8b07cd58
C has uuid bc58cea0-8ffb-4733-9650-441f3587023d
```

GNU Parallel

- You can use Parallel to run
  - Any shell command (ls, cat, echo etc.)
  - Any program that can be run from the command line
  - bash functions if they are exported using "export -f"

GNU Parallel

- Lots of other options but you can already see how you can use Parallel to run a wide variety of jobs
- Details for our cluster
  - https://wiki.csc.warwick.ac.uk/twiki/bin/view/HPC/ClusterUserGuide#Serial_jobs
- You can easily make pretty much any modern language take command line parameters
  - Can combine with input files (specified by name) for more sophisticated control

GNU Parallel

- This covers the basics of how to get GNU parallel to start up many different tasks

- There is a lot more that you can do, see the tutorial linked earlier

- There is also often stuff that you want to do to make Parallel play nicely with a cluster

    - Good example in the Sulis documentation at https://sulis-hpc.github.io/advanced/ensemble/gnuparallel.html