HMMER
Sequence Alignment Practical

Aims

The aims of this exercise are:

- to get you used to using the command line and a text editor in a terminal window
- to gain experience running a parallel computing job on an HPC machine using the batch submission system
- to explore the parallel performance of a research software package

Overview

In this exercise you will use the HMMER software package to search a protein database for similarity-based matches to a given protein sequence (for more information about the software and what is used for, see http://hmmer.org/).

You will progress through the following steps:

1. Run HMMER on a login node;
2. Run HMMER in serial on a compute node by submitting a job to the batch system;
3. Run HMMER in parallel using multithreading, and investigate the parallel performance;

We will use MareNostrum4 for this exercise.

Instructions

Log on to MareNostrum4

Start by logging on to MareNostrum4 following the instructions already provided.

Copy protein sequence data to your directory under /gpfs/projects/

Before we can run HMMER we need to get ready the input data we want to use it for, namely:

- a database of protein sequences
- a protein sequence to query the dabase with to find similarity matches

Change location to your directory in `/gpfs/projects/` with the following command, making sure to **replace `nctXXXXX` with your own username**:

```
user@login1:~> cd /gpfs/projects/nct00/nctXXXXX
```

We have placed on MareNostrum4 a protein sequence database file containing all Swiss-Prot annotated non-human mammalian protein sequences.
Copy this database file to your directory using the **cp** command, as follows:

```
user@login1:~> cp /gpfs/projects/nct00/nct00003/uniprot_sprot_mammals.dat ./
```

Also copy into your directory a file with the sequence of the C3 protein (Uniprot ID P01024), which plays a role in the human innate immune system response:

```
user@login1:~> cp /gpfs/projects/nct00/nct00003/P01024.fasta ./
```

Examine the contents of both files so you get an idea of the kind of data they contain. For help on how to do this, see the “Examing a file in the terminal” info box below.

Examing a file in the terminal

You can skip this info box if you already know how to examine a file in the terminal

There are a few commands that are useful to display the contents of a file. A common one, **cat**, simply prints the entire contents of the file to the terminal window.

For example, try:

```
user@login1:~> cat P01024.fasta
```

This is fine for small files whose entire contents fits in the height of the terminal window, but for larger files you will only see the end of the file (as much as fits on screen). In general you can use the **less** command, which lets you interactively scroll up and down through a file, and also search inside the file.

For example, try:

```
user@login1:~> less uniprot_sprot_mammals.dat
```

While examining a file using **less** you can use the up and down arrow keys to scroll line by line, the **n** key (think "next") or **spacebar** to move a page down, and **p** (think "previous") to move a page up. To search for a particular word or phrase, type **/** followed by your search term, and press enter. If there is more than one matching search result you can move between the one highlighted and the next or previous one by pressing **n** or **p**. When you have finished viewing the file, press **q** to exit and return to the command line.

Load HMMER

HMMER has been installed on MareNostrum4 but is not managed through the module system, so we can not simply run "`module load hmmer`" and expect HMMER commands to become available. Instead you first need to specify where HMMER executable files are located, by running the following command:

```
user@login1:~> export PATH=$PATH:/gpfs/projects/nct00/nct00003/hmmer/3.3.2/bin
```

This command only needs to be run once during a terminal session. It adds the HMMER location to a list of paths stored in an environment variable called PATH for the duration of the session. It would need to be run again if you log off MareNostrum4 and log on, because that would start a new terminal session.

Run a serial phmmer query on a login node

The particular command from the HMMER package that we will use is **phmmer**. This searches a protein sequence database for similarity-based matches to a given protein sequence. Make sure you are still in your own directory under /gpfs/projects/ and try running a phmmer query that searches the mammal database for similarity matches with the C3 protein, using the following command:

```
user@login1:~> phmmer --cpu 1 -E 1e-100 P01024.fasta uniprot_sprot_mammals.dat
```

Note: the --cpu 1 option ensures phmmer runs serially, using a single core on the MareNostrum4 login node that your terminal session is running on.

The resulting output shows details about similarity matches, and a summary of search statistics at the end, including total time taken for the search (see "Elapsed:"). For more information see the HMMER User Guide at http://eddylab.org/software/hmmer/Userguide.pdf and try running the "help" command:

```
user@login1:~> phmmer -h
```

With the chosen database, C3 query sequence, the default search sensitivity of phmmer means the search takes just a second or so. This is good because we should avoid using many cores or running anything long lasting on login nodes, as that is not their intended purpose. Many people share these nodes, and using them too intensely so can slow down access for all users. This applies to most HPC systems, and if you leave something running for an extended period of time on a login node you may be asked to stop, or what you are running may be terminated automatically. To run anything intensive we should run on compute nodes.

Run a serial phmmer query on a compute node

Basic job script

To run phmmer on a compute node we will use a job script and submit this to the scheduler.

Copy the job script file `/gpfs/projects/nct00/nct00003/phmmer.slurm into` your own directory under `/gpfs/projects/`

See if you can understand what the lines in the job script that start with `#SBATCH` mean based on the lecture on batch systems and the MareNostrum4 scheduler. You may find it useful (now and later) to search in the MareNostrum4 User Guide for additional information. The new phmmer option `--noali` ("no alignments") tells phmmer to print less detailed output information, so we can easily focus on the search statistics summary.

To submit your job, use the **sbatch** command:

```
user@login1:~> sbatch phmmer.slurm
```

Slurm tells you the unique numerical ID of your job when it confirms your batch job has been submitted. Check the status of your job using the **squeue** command, and see if you can understand what the current status of your job is. Once the scheduler has started the job its status (ST) column should say `R` ("running").

When we ran phmmer interactively on the login node its output went directly to the terminal window for us to see. Now that phmmer runs on a compute node Slurm manages where the output goes. By default it ends up in a file labelled `slurm-######.out`, were `######` is the job's unique numerical ID. However this file remains empty until the job finishes, which generally takes a long time and which may not happen at all if something goes wrong with the job. If we want to track progress by monitoring the output while the job is being executed we can redirect it to a named file using the **>** symbol, as follows:

```
user@login1:~> phmmer --cpu 1 --noali -E 1e-100 P01024.fasta uniprot_sprot_mammals.dat > results
```

Edit the job script in your directory to redirect phmmer output to a named file (see the "Editing a file in the terminal" info box below for help).
Now submit your job again and confirm the output file is as expected.

Editing a file in the terminal

You can skip this info box if you already know how to use a basic text editor in the terminal

It is important to be able to use an in-terminal text editor to at least be able to perform basic editing of job scripts on an HPC machine. Certain text editors are available on almost all HPC systems including on MareNostrum4, namely Emacs, Vim, and nano. We mention nano here as a very basic option - you should feel free to use whichever editor you prefer.

Nano can be launched with the command **nano** and, optionally, the name of an existing file that you wish to edit or a new file to create.
For example, try:

```
user@login1:~> nano phmmer.slurm
```

The terminal will change to show you are inside the nano text editor. Typing will insert text as you would expect, backspace will delete text, and navigation is using arrow keys. Menu options appear at the bottom, with **^** indicating use of the Ctrl key. For example, you can save (or "write out" to) a file using the key combination **Ctrl-o** (hold Ctrl and press o), which asks you to confirm the filename (press Enter). To exit nano and return to the command line use Ctrl-x (if you have unsaved changes nano will ask if you want to "Save modified buffer").

Run a parallel phmmer query using multiple threads (--cpu)

Matching the C3 protein against the mammals protein database with default sensitivity takes just a second or so even on one core. Try adding the phmmer option `--max` to your job script. This increases sensitivity to potential matches, but you will find your job takes longer to finish.

To speed up execution, run phmmer in parallel by setting its `--cpu` option and the Slurm `--cpus-per-task` option to some number between 1 and 48 (**always the same number for both options**). The maximum comes from the number of cores on a single node, which is 48 (in the form of 2 x 24-core Intel processors). This will cause phmmer to run in parallel using that number of threads, which are executed on that same number of cores.

Elapsed wall time vs CPU time

If you examine the phmmer output file for some search query you will see that at the end it contains a line with timing information of the following form:

```
# CPU time: 103.91u 0.11s 00:01:44.02 Elapsed: 00:01:44.09
```

The final value (00:01:44.09 in the above) gives the elapsed time, that is, how long phmmer ran for. We also call this the **wall clock time**, or **wall time**. It is the most intuitive measure of execution time, namely the duration - as read off from a clock hanging on the wall - that someone needs to wait for until the application finishes.

CPU time is different from **wall time.** CPU time can be thought of as the total number of core-seconds of work done by an application in the same way that the effort of multiple people working together on a job can be measured in total number of person-hours. It is a cumulative measure of time summing up contributions from all cores (all people).

How are the two related in practice? In the above example a single core runs the application for ~104 seconds, which is equal to the elapsed wallclock time. The timing for 4 threads might however give a different result, for example:

```
# CPU time: 113.48u 0.16s 00:01:53.64 Elapsed: 00:00:29.86
```

Here 4 cores accumulate a total CPU time of ~113 seconds (more than when the application was run on a single core), or an average of ~28 seconds per core, but because they are all working in parallel phmmer finishes in just under 30 seconds, almost 3.5 times faster than in the single core case, which took 104 seconds.

Parallel performance

Investigate how phmmer's execution time changes as more cores are used, record the reported elapsed times as in Table 1 below. For convenience and to avoid errors, instead of changing both `--cpus-per-task` and `-cpu` manually you can change the former and always set the latter equal to `$SLURM_CPUS_PER_TASK`, so that it automatically obtains the same value from this Slurm environment variable.

Table 1 - Times taken by multithreaded parallel phmmer protein search on a single compute node:

# Cores	Elapsed run time (seconds)
1	
2	
4	
8	
16	
24	
32	
40	
48	

Once you have completed Table 1 you can compute the speedups of the multithreaded to complete Table 2. The speedup is the ratio of runtime on 1 core to the runtime on *N* cores – this will make more sense after tomorrow's lecture on measuring parallel performance.

Table 2 - Speedups for multithreaded parallel phmmer protein search on a single compute node:

# Cores	Observed speedup
1	
2	
4	
8	
16	
24	
32	
40	
48	