CB2-201: Technical Aspects

Malay (malay@uab.edu)

February 16, 2015

Contents

1 There is no hyphen in bioinformatics

Paulien Hogeweg and Ben Hesper coined the term "bioinformatics" in 1970. "Computational Biology" came a little later. Although, these terms are used interchangeably, there are distinct differences. "Bioinformatics" is mostly used for cases where a biological problem is solved using computer programming as a essential tool. "Computaional Biology" on the other hand is used mostly for the theoretical aspects–model and algorithm development, etc. There are further subdivision of the field such as, computational genomics, computational neuroscience, etc. We will be using bioinformatics and compuational biology interchangeably in this course.

Remember, it's bioinformatics, not bio-informatics and it's definitely not "informatics"!

2 Processing large datasets

In bioinformatics, we frequently hand very large datasets. Most of the time it is not possible to analyze this datasets "serially". We have to process this data using parallel computing. There are generally two ways we can use parallel computing:

1. Using multiple cores in modern machine, called symmetric multiprocessing or SMP.
2. Using multiple computers on the cluster, generally called high-performance computing or HPC.

2.1 Poor man's SMP

SMP analysis of data can be done several ways. Most frequently, it is done at the programming language level, using special software libraries, like `pthreads`, `opencl`, etc. These type of parallel programming is essential when we need fine-grain parallel access to data-structure. However, fortunately, in bioinformatics most problems are "embarassingly parallel" and we can get away without special parallel programming knowledge.

The simplest way to do such parallel processing is to use the Linux shell's ability to run a process in the background:

```
gedit &
```

The `&` at the end runs `gedit` in the background and immediately returns the prompt. We can use the same technique to run two or more processes simultaneously:

```
1   echo "Before running the process"
2   (
3       count=0
4       while [ $count -le 10 ]; do
5           echo $count
6           sleep 1
7           (( count++ ))
8       done
9   ) &
10  (
11      count=100
12      while [ $count -le 110 ]; do
13          echo $count
14          sleep 1
15          (( count++ ))
16      done
17  ) &
18  wait
19  echo "Finished"
```

2.1.1 Exercise

Why there is a `wait` on the line 18? What will happen if we do not use it there?

2.2 Login to server

```
ssh username@server
```

2.3 tmux

`tmux` is a terminal multiplexer. This is the program you should start after you login to the server, before you do anything else. `tmux` will keep you program running, even if you lose connection to the server. You can also use several "windows" over on connection to the server. If you get disconnected you can log back in and run,

```
tmux attach
```

You will be back to your session. For a full list of keybindings of `tmux` look at this document:

https://gist.github.com/MohamedAlaa/2961058

2.4 High Performance Computing

Although, the above code works, it is problematic. First, the code runs regardless of the resources available in the machine and can bring it down to it's knees. Second, if one of the processes takes longer than the others to finish then other processor can sit idle. This is not an optimal solution.

To solve this situation there are software called "job shcedulers" available. One of the most popular one is "Sun Grid Engine" or SGE. This scheduler can also run jobs over a cluster of computers.

```
# Submit a job
qsub -cwd -b y -V -j y -o my.log "my job"

# For running jobs in cheaha
qsub -cwd -b y -V -j y -o my.log -l h_rt=1:00:00,vf=4G "my job"

# To check the status
qstat -u "my_user_name"
```