140.636 Assignment 4

Embarrassingly-Parallel processing on the JHPCE cluster

Fernando J. Pineda

Sep. 27, 2017

Assignement description

part 1 (20%) Write a fasta file splitter

Create a perl script (named fsplit) that will parse a large fasta file (approximately 42,000 sequences) into a 10 smaller files with approximately 4200 sequences each. Each file has a standard name (temp01.fna, temp02.fna, ... etc.)

Bioperl modules to make this easy. Examine the following example code that shows how to use bioperl to read records in one format and output them in another format:

```
#!/usr/bin/perl
use warnings;
use strict;
use Bio::SeqIO;

# create SeqIO object for the input stream
my $in = Bio::SeqIO->new('-file' => "B_subtilis.dat", '-format' => 'swiss');

# create SeqIO object for the input stream
my $out  = Bio::SeqIO->new('-file' => ">B_subtilis.fna", '-format' => 'fasta');

while ( my $seq = $in->next_seq() ) {
        $out->write_seq($seq);
}
```

The script creates an input stream that reads `B_subtitlis.dat` and an output stream to `B_subtilis.fna`. It then steps through each record in the intput file, reads it in unitprot format and writes it in fasta format. The code and input files are in `/users/sph140636/shared/code_examples/14_bioperl`.

You can use the basic pattern in this script to read one large fasta file and break it up into a set of smaller files.

part 2 (80%) submit an array job that splits, blasts and merges the blast results.

The script for the array job should perform the following 3 tasks:

1. deletes any temporary files that might be left over from a previous run
2. runs your fsplit script on the large fasta-formatted input file and outputs 10 smaller fasta-formatted files.
3. submits array tasks that perform the BLAST queries and appends the output to a common output file.

1. Preliminaries (setting up BLAST and homework directory)

1.1 Configuring BLAST

Skip right to step 1.2 if you already have a local BLAST installed and working.

Create a directory for your blast databases

```
mkdir ~/blastdb
```

Next, add an environment variable (`BLASTDB`) to your `.bashrc`, so that the blast application can find the blast

database, then source the `.bashrc` file to instantiate the environment variable.

```
echo 'export BLASTDB=$HOME/blast/db' >> ~/.bashrc
source ~/.bashrc
```

Load the `blast+` (version 2.2.30) module. This will give you access to all the blast binaries that are installed.

```
module load blast/2.2.30
```

1.2 Format a blast database

For the purposes of this assignment, we will create a searchable blast database from the 2012 version of the *Plasmodium falciparum* 3D7 genome. The fasta-formatted file has been preloaded and is available at `/users/sph140636/shared/code_examples/blast/Pfa3d7.fna`

cd into your blastdb directory and create a symbolic link to the fasta file. Creating a link in your directory instead of copying the file will save time and space.

```
cd $BLASTDB
ln -s /users/sph140636/shared/code_examples/blast/Pfa3d7.fna  Pfa3d7.fna
```

Next we run the `makeblastdb` command to create the BLAST index files.

```
makeblastdb -title Pfa3d7 -dbtype nucl  -in Pfa3d7.fna
```

If everything worked correctly, you should see index files in the directory with .nhr, .nin and .nsq extensions. If you do not see these files, debug your steps and try again. If everything is OK, you can delete the link to the original fasta file since it is no longer needed.

```
rm Pfa3d7.fna
```

1.3 Test on example blast queries

create a homework directory and cd into it

```
cd /users/sph140636/homework/$USER
mkdir homework4
cd homework4
```

Note that $USER expands to your username

Test your installation and BLAST database by blasting a few short sequences. Access the example sequences by linking to the fasta file

```
ln -s /users/sph140636/shared/code_examples/blast/example_queries.fasta
```

These are short sequences so you will need to use the blastn-short task which is optimized for short query sequences.

```
blastn  -task blastn-short -db Pfa3d7.fna -query example_queries.fasta
```

If you see alignments streaming by, then everything worked fine, if not you'll have to debug your steps.

2. Parallelizing BLAST queries

Now that the preliminaries are out of the way, you can prepare the array job: We will use a file of ESTs (Expressed Sequence Tags) from *Plasmodiam vivax* and blast them against our *Plasmodium falciparum* database. Start by creating a link to the query file in your homework directory

```
ln -s /users/sph140636/shared/code_examples/blast/PlasmoDB-28_Pvivax_ESTs.fna
```

```
time blastn -db Pfa3d7.fna -query PlasmoDB-28_Pvivax_ESTs.fna > output.txt
```

The run time is 1-2 minutes depending on the node you are running on.

2. Parallel processing

2.1 Using multiple cores

Some applications (e.g. blast) are written in a way that allows them to take advantage of multiple cores on a node. Prior to submitting a job that using multiple cores, we need to tell the scheduler that we will be using multiple cores. SGE can't actually tell if you are using more cores than you specify. Using more cores than you have specified is bad practice and can degrade the performance of the cluster. It can even crash a node. So always tell SGE the number of cores you expect to use. On the flip side, don't ask for too many cores either since any cores you do not reserve will not be available to other jobs.

Let's exit our qrsh session and start a new session that allocates 4 cores to our session.

```
qrsh -pe local 4
```

By default blast will run on a single core, but it is written in such a way that it use muliple cores. We enable this capability by specifying the `num_threads` BLAST parameter.

Also don't forget to load the blast module again.

```
module load blast/2.2.30
```

The `time` command will give you timing information about a submitted command.

```
time blastn  -num_threads 4 -db Pfa3d7.fna -query PlasmoDB-28_Pvivax_ESTs.fna \
> output.txt
```

Try running this command with num_threads set to 1 and 4. The result is disappointing. Even though we use 4 cpus, it still takes about the same amount of time. We could dig deeper into why this doesn't speed things up by a factor of 4. Instead, let's just try a different strategy. We will split the query file into 10 smaller files and submit 10 independent jobs.

2.2 Embarrassingly parallel processing

How many sequences are in the query file? To count the number of sequences, we take advantage of the fact that the first line of a fasta record starts with '>', so to count the number of records we use grep to extract all the lines that start with '>' and then count the number of lines with the `wc` command.

```
grep '>' PlasmoDB-28_Pvivax_ESTs.fna  | wc -l
```

There are 41,488 ESTs in our query file. We can time how long it takes to blast this query file with the following command

The idea here is to break up the input fasta file into *N* smaller fasta files so the input files can be processed in parallel, thereby obtaining (at best) an *N*-fold speedup (in wall-clock time).

You will have to write a perl script (named `fsplit`)that splits the large query file into 10 smaller files. The perl script should take as input the approximate number of fasta records in the smaller files as well as the filename of the input file. The number of output files you get will depend on the number of records in the input file and the number records you want in eah of the smaller files. Thus for example, the command line that invokes your splitter would look like this:

```
fsplit --numseq 4200 --filename  PlasmoDB-28_Pvivax_ESTs.fna
```

And it should yield 10 files () `temp01.fna` through `temp10.fna`) with 4200 sequences each (except for the last one). If you blast one of these fasta files, you will find that it is processed roughly 10 times faster than the full file.

```
blastn -db Pfa3d7.fna -query temp1.fna > output.txt
```

No surprise here since there are about 10 times less sequences to process. If we processed each of the 10 file simultaneous, we would observe a 10x speedup (in wall-clock time). Of course it takes time to split the original file, so we do not expect a 10x speedup in this case.

2.3 Creating the script for the array job

As a starting point, here is a script that blasts just one of the temporary files and returns the result in output.txt. You will want to generalize this script by adding two features:

```
#!/bin/bash
# SGE submission options
#$ -q shared.q                # Select the queue
#$ -l h_rt=00:01:00          # Kill job if exceeds1 minute of wall clock time
#$ -cwd                       # Change to current working directory
#$ -V                         # Export environment variables into script

# If you want to use the default BLAST on the cluster, uncomment the following line
# module load blast/2.2.30

# cd into your homework directory
cd /users/sph140636/homework/$USER/homework4

# perform one blast query
blastn -db Pfa3d7.fna -query temp01.fna > output.txt
```