Introduction to HPC — Assingment #3

October 10, 2025

Assignment #3

Total: **10 points**

Due date: **October 24th**

*This assignment must be completed **individually**.*

I.e. you should work alone without any help or collaboration from others.

Please provide detailed answers to the following questions.

Please READ carefully!

- Solutions must include proper modular implementations, best practices in software development, comments and documentation.
- **Your solutions (code) MUST compile in the Teach cluster!**
  Code that does NOT compile will get a 0!

Problem #1 The following command will manipulate and convert an image into a "thumbnail" version of it:

```
convert -geometry 120 foo.jpg thumb_foo.jpg
```

`convert` is part of a highly specialized set of tools for image manipulation from the package "ImageMagick" – see https://imagemagick.org/script/convert.php.

Write the corresponding version of this command that will run with number-of-cpus jobs in parallel for all jpg files in a given directory.

Problem #2 Given a list of URLs in a file named "URLsFile.txt", list all the URLs that fail to download. Print the line number and the URL.

Do this using a sequence of shell commands or shell-script, and in a parallel efficient implementation. `wget` will be a useful command.

Problem #3 Find all the files from a list of files contained in a file named "*file_list*" that do **not** exist. Do this using a sequence of shell commands or shell-script, and in a parallel efficient implementation.

Problem #4 Let's assume that certain website stores images in the following format:

'`https://www.example.com/path/to/YYYYMMDD_##.jpg`'

where `YYYYMMDD` is the date and `##` is the number 01-24.

Write a parallel way to download images for the past 30 days.

Problem #5 NASA open data initiatives provide tiles to download available at `https://earthdata.nasa.gov`. Download tiles for our "Blue Marble" world map and create a 10240×20480 map.

For that you need to configure the URL as follows:

```
base=https://map1a.vis.earthdata.nasa.gov/wmts-geo/wmts.cgi
service="SERVICE=WMTS&REQUEST=GetTile&VERSION=1.0.0"
layer="LAYER=BlueMarble_ShadedRelief_Bathymetry"
set="STYLE=&TILEMATRIXSET=EPSG4326_500m&TILEMATRIX=5"
tile="TILEROW={1}&TILECOL={2}"
format="FORMAT=image%2Fjpeg"
url="$base?$service&$layer&$set&$tile&$format"
```

What would be the meaning of `${1}` and `${2}` in the above setup have?

Problem #6 The typical structure for a for-loop in the shell looks like this:

```
(for x in `cat list` ; do
    do_something $x
done) | process_output
```

Which using gnu-parallel can be written as:

```
cat list | parallel do_something | process_output
```

How does this generalize for nested for-loops, ie.

```
(for x in `cat xlist` ; do
    for y in `cat ylist` ; do
        do_something $x $y
    done
done) | process_output
```

Problem #7 1. Take one of the exercises from the first set of problems and modify it to accept *command-line arguments* (CLA). E.g. the Lissajous figure could take 2 CLA for the multiplicative factors of the arguments of the sin/cos functions. Verify that your updated code runs as expected passing the corresponding values for the CLA and matches the previous results you obtained.

2. Write a submission script for the teach cluser that will run this program and vary the CLA values from 1 to 5 for each function with all the possible combinations. The runs should be done efficiently using all the cores avaiable in a whole node and not wasting time between runs.

3. After you tested the previous step, and you are sure that the jobs ran succefully, now the target will be to improve the IO pattern of the job by saving the files of each run into RAMdisk. For this we will generate a second submission script based on the former one that achieves this. Recall that you should "recover" the final results before the job is finalized.

Problem #8 Reviewing the cluster's batch configuration.

Investitage the following commands in the teach cluster:

`sinfo, sinfo -s, sjstat, sdiag, sshare`

Check their `man` pages and `-help` options.

Using the aforementioned commands, answer the following questions:

1. Which queues are configured?
2. How many nodes are there in each queue?
3. What are the batch queue node and time limits?
4. What states are the nodes in (alloc, idle, etc.)?

Problem #9 *Implementing a real bioinformatics pipeline*

For this exercise we will look at a real application based in a typical bioinformatics pipeline. HMMER –http://hmmer.org/– is a tool used for searching sequence databases for sequence homologs, and for making sequence alignments. It implements methods using probabilistic models called profile hidden Markov models (profile HMMs).

HMMER v3 takes a protein sequence and compares it to a probabilistic profile that describes a protein domain. It reports when there is a statistically significant likelihood that the protein and the domain share the same evolutionary origin. This basic comparison is repeated for all combinations of many protein sequences and many domains.

1. The very first thing we will do is to install HMMER3 in the cluster. As it happens with shared systems, we do not have root privileges[1], so we need to do a local installation of HMMER3 in our user space:

   (a) Download the hmmsearch package: `hmmer.org/download.html`

   ```
   wget http://eddylab.org/software/hmmer/hmmer-3.4.tar.gz
   ```

   (b) Proceed to install the hmmsearch package – after downloading the hmmer-3.4.tar.gz, we will

   - uncompress the gzipped file,

     ```
     tar zxf hmmer.tar.gz
     ```

   - configure and install hmmer,

     ```
     cd hmmer-3.4
     ```

     Before proceeding, load the gcc/13.2.0 module, so the configuration and installation can detect this compiler!

     ```
     module load gcc/13.2.0
     ```

     Configuration:

     ```
     ./configure --prefix=$HOME/hmmer
     ```

     This previous command will configure the HMMER installation and after a long output you would see something like this:

     ```
     HMMER configuration:
     compiler:              gcc -O3    -pthread
     host:                  x86_64-pc-linux-gnu
     ```

[1] and even we have them, it is a VERY VERY BAD idea to install software as root!

```
linker:
libraries:              -lpthread
DP implementation:    sse

Now do 'make'  to build HMMER, and optionally:
'make check'  to run self tests,
'make install'  to install programs and man pages,
'(cd easel; make install)'  to install Easel tools.
```

Now is time for the actual installation:

```
make
make check
make install
(cd easel; make install)
```

2. The next step is to obtain some real data:

```
# Download the protein HMM searchable library here:
wget ftp.ebi.ac.uk/pub/databases/Pfam/current_release/Pfam-A.hmm.gz

# Download the fasta file:
wget https://ftp.uniprot.org/pub/databases/uniprot/current_release/
    knowledgebase/complete/uniprot_sprot.fasta.gz
```

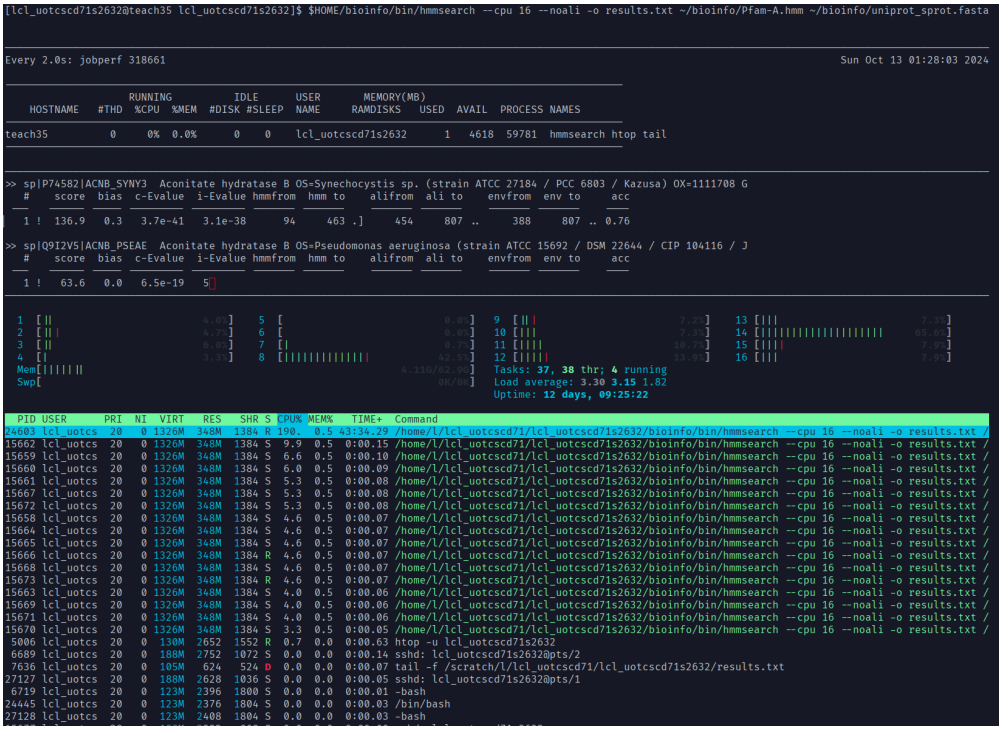
3. Preparing the data – now we need to uncompress and split some of the data files:

```
# uncompress the fasta file
gunzip uniprot_sprot.fasta.gz

# splitting the fasta file using AWK
awk -v chunksize=$(grep ">" uniprot_sprot.fasta -c) 'BEGIN{n=0; chunksize=
    int(chunksize/256)+1 }
/^>/ {
  if(n%chunksize==0){
    file=sprintf("uniprot_%d.fasta",1+(n%256));
  }
  print >> file; n++; next;
    }
{ print >> file; }' < uniprot_sprot.fasta
```

Questions

1. Take a look at what the fasta file and splits contain; inspect file sizes, file types, etc. What can you conclude?
2. Let's create now the actual processing pipeline:
   (a) Take a look at the `hmmsearch` command and valid arguments
   (b) For instance, what does the following line do?
   ```
   hmmsearch --cpu 8 --noali -o output.txt Pfam-A.hmm input.fasta
   ```
   (c) How can this be combined with GNU-parallel to make it more efficient? I.e. write a version that will use GNU-parallel in conjunction with `hmmsearch`.
   Given that `hmmsearch` has some level of parallelism (which one?), then how using GNU-parallel is going to make it more efficient?
   (d) Create a submissions script which does this in batch mode.
   Monitor and report the resources utilization of the job, e.g. using `jobperf` – in a given moment during the execution of the job.
   Compare this by looking at instantaneous statistics reported in realtime and *in-situ*, i.e. directly connected to the node where you are running the job. What can you conclude from this?
   (e) Create another submission script which does it using multiple nodes.
   Monitor and report the resources utilization of this job in similar conditions as before.
   (f) Now, let's consider what we have learned from this pipeline and program... what additional strategy would be a good idea to try next?



GNU-Parallel Resources

- useful/interesting flags: `--dry-run`, `--progress`
- https://www.gnu.org/software/parallel/

slurm scheduler

- https://slurm.schedmd.com/documentation.html
