Alphafold on HPC

Introduction

Alphafold is a “deep learning” algorithm that takes as an input a fasta file containing a single protein amino acid sequence and outputs a directory containing various intermediate data files; performance characteristics; and five predicted protein structures in pdb format. The pdb structures are probably best reviewed in pymol.

Alphafold has been implemented on the University of Manchester HPC cluster. This document describes submitting jobs to the cluster. The first is a simple shell script that should be edited to submit the required single fasta file to alphafold.

I will also provide guidance on how to separate a large fasta file containing many protein sequences into a directory containing a single fasta file for each sequence, named for the sequence.

Local Tools

You need an terminal interface to the HPC. It is possible to access HPC through any command line terminal. On windows the easiest method is to use mobaxterm (https://ri.itservices.manchester.ac.uk/htccondor/getting-started/connecting/windows/) which incorporates both a terminal interface and a file browser and allows copying of files or directories by drag-and-drop.

You will also need a test editor. There are many available, and there is one build into mobaxterm. I also use notepad++ (https://notepad-plus-plus.org/downloads/). The mobaxterm editor will update the file on HPC on save, which is a useful feature if making small edits to a file you want to run.

It is important to note with text editors that Windows and Unix use different end-of-line (EOL) codes. If write a script from scratch on windows it is likely it will fail on the unix based HPC due to incorrect EOL. In notepad++ the EOL may be changed through the menu:
Edit ⇒ EOL Conversion ⇒ Unix (LF).
In the mobaxterm text editor it may be changed through the menu:
Encoding ⇒ Unix.

Example Scripts Manifest

This guide comes with example scripts in a zipped directory `example.zip`.

```
example-¬
        ⊢ singleFasta-¬
        |             ⊢IOR1.fasta
        |             LtestAlphafoldSingleScript.sh
        |
        L fastaManipulation-¬
                              LYeastMapK.fasta
```

Unzip the example directory and copy it to the scratch directory on the HPC.

Keep the local copy of the example directory open to review the scripts.

Single File With Script

Navigate to the `singleFasta` directory on HPC:

```
cd ~/scratch/example/singleFasta
```

Now you are in a directory containing a script `testAlphafoldSingleScript.sh` and a fasta file `IOR1.fasta`.

Description of the Components of the Script

The script contains the following code:

```
#!/bin/bash --login
#$ -cwd             # Job will run from the current directory
#$ -l v100=1        # Job will run using 1 x GPU
#$ -pe smp.pe 8


# Load the version you require
module load apps/singularity/alphafold/2.1.1

run_alphafold.sh -f `pwd`/IOR1.fasta -t 2022-04-01 -o `pwd`/output_directory -m monome
r -c full_dbs
# PLEASE NOTE: `pwd` is required so the singularity container is able to map to the CS
F filesystem
```

The meaning of which breaks down as follows:

`#!/bin/bash --login` is the shebang or hash-bang. It tells the grid engine the script is to be run in bash, logged in as you.

The `#$` lines tell the grid engine how to run the script. In this case `-cwd` to inherit the current working directory; `-l v100=1` the required resources, in this case requiring a node with a single GPU; and `-pe smp.pe 8` requiring eight cores. That last is an excess of compute since most of the work is done by the GPU, I have been running alphafold with `-pe smp.pe 2` . There are many more settings for `qsub` which may be viewed on the HPC cluster with by typing `qsub -help` on the command line.

`module load apps/singularity/alphafold/2.1.1` tells the node to load the module that is capable of running alphafold and will make the `run_alphafold.sh` script available on the comand line.

Now finally alphafold is run. There is documentation available for running alphafold (https://github.com/deepmind/alphafold#running-alphafold).

Part of that documentation provides an example run on docker:

```
python3 docker/run_docker.py \
  --fasta_paths=T1050.fasta \
  --max_template_date=2020-05-14 \
  --model_preset=monomer \
  --db_preset=reduced_dbs \
  --data_dir=$DOWNLOAD_DIR
```

These settings are duplicated in the HPC implementation.

The local documentation can be obtained by loading the module on the head node and running the script on its own:

```
module load apps/singularity/alphafold/2.1.1
run_alphafold.sh
```

Which produces documentation:

```
Please make sure all required parameters are given
Usage: /opt/apps/apps/singularity/alphafold/2.1.1/bin/run_alphafold.sh <OPTIONS>
Required Parameters:
-o <output_dir>   Path to a directory that will store the results.
-m <model_preset>  Choose preset model configuration - the monomer model, the monomer
 model with extra ensembling, monomer model with pTM head, or multimer model (default:
'monomer')
-f <fasta_path>   Path to a FASTA file containing one sequence
-t <max_template_date> Maximum template release date to consider (ISO-8601 format - i.
e. YYYY-MM-DD). Important if folding historical test sets
```

```
Optional Parameters:
-g <use_gpu>          Enable NVIDIA runtime to run with GPUs (default: true)
-n <openmm_threads>   OpenMM threads (default: all available cores)
-c <db_preset>        Choose preset MSA database configuration - smaller genetic datab
ase config (reduced_dbs) or full genetic database config (full_dbs) (default: 'full_db
s')
-p <use_precomputed_msas> Whether to read MSAs that have been written to disk. WARNIN
G: This will not check if the sequence, database or configuration have changed (defaul
t: 'false')
-l <is_prokaryote>    Optional for multimer system, not used by the single chain syste
m. A boolean specifying true where the target complex is from a prokaryote, and false
 where it is not, or where the origin is unknown. This value determine the pairing met
hod for the MSA (default: 'None')
-b <benchmark>        Run multiple JAX model evaluations to obtain a timing that exclu
des the compilation time, which should be more indicative of the time required for inf
erencing many proteins (default: 'false')
```

Which can be cross referenced against Alphafold documentation.

So the script above specifies to run alphafold with

`-f `pwd`/IOR1.fasta` which says look in the current working directory passed to the grid engine by the `#$ -cwd` setting above, for fasta file `IOR1.fasta`.

`-t 2022-04-01` which limits the pdb files used to train the model to before this date. It is included in the test script to ensure consistent results. It may be removed from production scripts.

`-o `pwd`/output_directory` sets the output directory.

`-m monomer` Indicates monomer model is to be used. Other settings are available see the alphafold documentation for details.

`-c full_dbs` sets use of the full genetic database. This may be set to `reduced_dbs` for faster, less accurate folding.

Submitting the Job to the Grid

So you should now be interacting with the HPC through a terminal.

Check you are in the correct dir by typing `pwd`. This should return something like:
`/mnt/<home-server>/<username>/scratch/example/singleFasta`
Now submit the job:

```
qsub testAlphafoldSingleScript.sh
```

That's it. You will get a message saying something like:
`Your job <a number> (" testAlphafoldSingleScript.sh ") has been submitted`

You can check the status of your job by typing `qstat`. This will produce a table of your current jobs. The important thing to note is the letter under `state`. If this is `r` the job is actually running. If the code is `qw` then the job is in the queue to be run. However if the code is `Eqw` there is something wrong with the script or the way it was submitted and you will need to solve a problem.

Monitoring Job Progress

Now the job is running the working directory will look like this:

```
example-¬
         L singleFasta-¬
                       ⊢IOR1.fasta
                       ⊢testAlphafoldSingleScript.sh
                       ⊢ld.so.cache
                       ⊢testAlphafoldSingleScript.sh.e<jobno>
                       ⊢testAlphafoldSingleScript.sh.o<jobno>
                       L output_directory-¬
                                          |...
```

If working on mobaxterm the directory view will need to be manually updated to show the change. This is the green dot with the curved white arrow in it above the file browser. Alternatively type `ls` or `ls -l` to see the dir contence in the terminal.

The files `*.e<jobno>` and `*.o<jobno>` contain the error and standard output of the job respectively. Alphafold seems to put all of its running information out to `*.e<jobno>` and if the job fails reviewing the contents of `*.e<jobno>` may offer some clue at to why. To monitor the progress of the job in real time directly on the terminal type:

`tail -fn1 testAlphafoldSingleScript.sh.e<jobno>`

Which will print each successive line of output to the terminal. For a long process like alphafold that is like watching paint dry. To exit this view type CTRL+z .

In the event that a job permanently shows status `Eqw` or you realise a mistake and would like to stop the job before it completes the comand is `qdel <jobno>`

Converting Multi-Sequence Fasta File to Multi-Single-Sequence Fasta Files

So if you have a multi sequence fasta file and you want to convert it into multiple single sequence fasta files you can use a program called `awk`.

This is probably a small enough operation that it could be run on the head node. See further down for how to submit the

So navigate to `~/scratch/example/fastaManipulation`

Now copy paste:

```
awk '/^>/ {S=sprintf("%s", $0);D=gensub(/\s.+$/,"",S);E=gensub(/>/,"","g",D);F=gensub
(/\|/,"_","g",E);print D >  F ".fasta"; next;} {printf("%s\n",$0) >> F ".fasta";nex
t;}' < YeastMapK.fasta
```

What this does is run through the file `YeastMapK.fasta` which contains 123 proteins.

`'>'` tells awk the symbol

For lines that begin with `>` (thats the `'/^>/` bit) which contain the sequence names do the operations inside the first set of `{}` which means the following:
Variable S gets the whole line.
Variable D gets S with everything from the first space removed.
Variable E gets D with the > removed from the front
Variable F gets D with the `|` replaced with `_`
Now print D to a file called F.fasta.
For subsequent lines not begining with `>` just print the whole line to the F.fasta file.
When a new line starting with `>` is encountered the process starts again and a new file is started.

The result of this for a set of protein sequences in uniprot format is a whole set of files with filenames containing the the accession numbers, and correctly formatted single protein sequence fasta files that may be submitted to alphafold.

Alternative formats of fasta file may need modifications to the cascade of pattern substitutions to form suitable file names and annotations in the newly written files.

If a fasta file is large, it would be better to submit the job to a node. There is a neat way to do a submission of a one liner to a node as follows:

```
qsub -cwd -pe smp.pe 2 -N splt <<HERE
#!/bin/bash --login
awk '/^>/ {S=sprintf("%s", $0);D=gensub(/\s.+$/,"",S);E=gensub(/>/,"","g",D);F=gensub
(/\|/,"_","g",E);print D >  F ".fasta"; next;} {printf("%s\n",$0) >> F ".fasta";nex
t;}' < YeastMapK.fasta
HERE
```

This just submits the job to a node with two cores, running the code in the current working directory with the name “splt”. Then it uses a “here doc” to read the code between `<<HERE` and `HERE` as the input script.