Exercises

a) Log in to Taito either with your training or CSC user account, either from a terminal or using NX client

b) Create a batch job script that prints the compute node on which it is running

Note. The dollar sign "**$**" at the beginning of some lines is not to be typed, but marks the command prompt. What follows is to be typed on the command prompt.

1. Simple batch job script

Using nano editor:

```
$ nano test_hostname.sh
```

```
#!/bin/bash -l
#SBATCH -J print_hostname
#SBATCH -o output.txt
#SBATCH -e errors.t
#SBATCH -t 00:01:00
#SBATCH -p test
#SBATCH -n 1
#SBATCH --mem=500
echo "This job runs on the host: "; hostname
```

CTRL+0; CTRL+X to exit (confirm save)

Submit the batch script to Taito

```
$ sbatch test_hostname.sh
```

a) Check the job status (replace <your username> with the training account or your CSC username, which ever you used to log in to Taito. You can see it also with this command **echo $USERNAME**):

```
$ squeue –u <your username>
```

b) Check out the output:

$ less output.txt (type q to quit)

$ less errors.t (type q to quit)

2. Matlab batch job

Transpose a matrix, e.g. 3X3, by running Batch job of Matlab on Taito

a) Create a matlab file, called e.g. transpos.m, which may look like this:

```
a=[1 2 3;4 5 6;7 8 9];
% non-conjugate transpose of matrix a
b2=transpose(a)
```

b) Create a batch job script, which will submit it to the queue in a row (non-interactive) mode (**$ matlab -nodesktop –nosplash)**, *e.g.* like this (everything in one line, without the "\" character):

```
$ srun -v matlab -r "transpos;exit" -nodisplay \
-nosplash -nodesktop -nojvm
```

Remember to load Matlab module before the srun command in the batch script:

```
$ module load matlab
```

You can call your batch script *e.g.* matlab_transpos.sh.

3. Copying files to hpc_archive

You can access hpc_archive only from Taito and Sisu. After installing some tools, you can access IDA, which also uses iRods technology, also from your computer.

Log in to Taito. Check the contents of your hpc_archive:

```
$ ils
```

Show the directory that you're in in hpc_archive

```
$ ipwd
```

Show the directory that you're in in taito:

```
$ pwd
```

Create a directory in hpc_archive

```
$ imkdir test
```

Move to test directory in hpc_archive

```
$ icd test
```

Confirm where you are in hpc_archive

```
$ ipwd
```

Copy (put) a file to the test directory in hpc_archive:

```
$iput <filename>
```

Confirm that the file is in hpc_archive

```
$ ils
```

Copy the file back from hpc_archive, but to a local folder called localtest

```
$ mkdir localtest
$ cd localtest
$ iget <filename>
```

3b) Make a new directory in hpc_archive home directory (not under test) called mysafe. Copy your linux-learn file there.

4. Batch job with thread parallization

Some applications can be run in parallel to speed them up. In this example you run the HMMER software to describe and analyze related or similar sequence areas both in serial and parallel to see if the jobs speed up.

HMMER uses a database that is already installed, but the protein sequences you want to study need to be copied first to be used as input:

```
$ cp /appl/bio/hmmer/example.fasta .
```

Let's first run the job with just one core. Change / add the following items in your batch script:

1) Output to **`out_%j.txt`**
2) error to **`err_%j.txt`**
3) run time 10 minutes

4) load the **`hmmer`** -module
5) run command:

```
hmmscan $HMMERDB/Pfam-A.hmm example.fasta > example_1.result
```

Submit the job with:

```
$ sbatch jobscript.sh
```

Submitting the job echoes the SLURM job id number to the screen, but that is also shown in the output and error filenames (**out_<SLURM_JOBID>.out**). Check if the job is running with

```
$ squeue -u <your username>
```

Or

```
$ squeue -j <SLURM_JOBID>
```

Once the job is finished you can check how much memory and time it used:

```
$ sacct -j <SLURM_JOBID> -o elapsed,reqmem,maxrss
```

Did you reserve a good amount of memory? (not excessively too much, but enough to not be close to memory running out and terminating the job).

Now, let's try with 4 cores. At this point we'll also switch to using the environment variable **$SLURM_CPUS_PER_TASK** to avoid mistakes and the need to change that in many places. Add this line to the batch script:

```
#SBATCH --cpus-per-task=4
```

Change the run command to:

```
hmmscan --cpu $SLURM_CPUS_PER_TASK $HMMERDB/Pfam-A.hmm \
     kaks.fasta > kaks_$SLURM_CPUS_PER_TASK.result
```

As you've asked for 4 cpus per task, the environment variable **$SLURM_CPUS_PER_TASK** evaluates to 4 when the script is run, and you only need to change the number on the #SBATCH line.

Submit the job and check with the **sacct** command how long it took to run the hmmer job and how did the memory usage change and try to answer these questions:

a) Does it make sense to use 4 cores instead of 1?

b) Was the memory reservation ok?

c) Does it make sense to use more than 4 cores?

d) How to speed up the job?