Homework 3

ME 5773 High Performance Computing Due: 02/10/2024 11:59pm

This is a team assignment. The teams are assigned by the instructors. The team members will be posted on Canvas. The deliverable for this assignment is an email to the instructors with the address of the repository. This assignment will be evaluated using the most updated version of the repository **before the due date**.

1. For this assignment you are required to create a public GitHub repository. For this, you can follow the steps below:
   - 1.1. All students should create a GitHub account: Go to the link https://github.com/, click on the 'sign up' button at the top right of the page, and follow the instructions. A GitHub free account is sufficient to complete this and other assignments for this class. If you already have a GitHub account, you may continue to step 1.2.
   - 1.2. Create a public repository. The repository must be created by a single member. However, it is suggested to do so in a meeting, so that all are aware of the steps together. Use a name that is available and append 'ME5773SP24' to associate it to this course.
     - 1.2.1. Follow the instructions under section 'Create a repository' on https://docs.github.com/en/repositories/creating-and-managing-repositories/quickstart-for-repositories
     - 1.2.2. Select the following options:
       - Check 'Include a readme file'. And set '.gitignore: None'.
       - License: For this assignment you are free to choose any license or define your repository without license.
   - 1.3. Once the repository has been created, add your teammate as collaborator so that both can access and commit to the repository.
     - 1.3.1. Follow the instructions on https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-access-to-your-personal-repositories/inviting-collaborators-to-a-personal-repository
   - 1.4. All team members should clone the repository into your respective `/work/<abc123>/sources/` folders.
   - 1.5. A SSH key is required to interface with GitHub. Each student should generate this key. You may generate this with Arc. These steps follow the instructions under 'Generating a new SSH key' on https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent
     - 1.5.1. Log into Arc
     - 1.5.2. In a login Node Type
       ```
       ssh-keygen -t ed25519 -C "your_email@example.com"
       ```
       Use the same email address you used to setup your GitHub Account.
     - 1.5.3. Follow instructions. **Do not change the location of the ssh key file**.
     - 1.5.4. You will need to define a passphrase. **This passphrase should be different to your Arc's and github's passwords!**
     - 1.5.5. After this program runs, you will generate a passkey. The passkey is in the file `~/.ssh/id_ed25519.pub`

1.6. Show the contents of the file with `cat ~/.ssh/id_ed25519.pub` , and copy the text. (MobaXterm use right-click to copy. On Mac, use command+c). Do not use vi or vim for this particular step as this does not let you copy the text in your PC.

1.7. Go to your github webpage, and click your user logo (top right corner), click settings, and then select SSH and GPG keys. Click on the "new SSH key" and add the copied text. This will setup correctly your account to do push and pull requests correctly.

1.8. Email the instructors the repository address.

2. Document your repository. After you create the repository, your repository's root folder contains a README.md file. You can open this file with a text editor to modify it. This file follows the following syntax: https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

   Add a description to the readme file in your local file (within Arc). The added description should state that your repository corresponds to team assignment work for ME5773. You may add more descriptive text. After you have finished, save the file. Go to the repository's root directory. Make sure you have the latest version of your repository with the following strategy:

   `git fetch` (checks changes on the server database. Does not alter your working tree).
   `git status` (shows the current state of the files within your local repository).
   `git pull` (brings changes from the database and updates your working tree).

   Solve any conflicts before proceeding. Check any potential problems with git status. Once no conflicts are present, you can add your changes to the staging area and then commit the changes using

   ```
   git add .

   git commit -m "<message>"
   ```

   `<message>` should be a meaningful and summarized description of what modifications were made.

   Once you want to update your server repository, you need first to make sure you have the latest version locally, thus run `git fetch; git status; git pull` and finally execute:

   ```
   git push
   ```

   This updates the server repository.

3. Create a folder named assignment3. Within this folder create one file named ‘`fact.sh`’. This script should work as follows: Given an input number <N>, it outputs all factorials from 1 to N each new line. That is, if N=4 the output should be

   1! = 1
   2! = 2
   3! = 6
   4! = 10

   Examples of the intended execution are as follows:

```
[c001: abc123]$  bash fact.sh 3
1! = 1
2! = 2
3! = 6

[c001: abc123]$  bash fact.sh 7
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
```

   Hint: got to the repository bashEx (update your local version with git pull or go to link https://github.com/mauriaristi/bashEx) and look at all examples in `simpleTasks`, `variables` and `control`. In particular:

```
  bashEx/variables/vars4.sh
  bashEx/control/contr4.sh
```

   Once the program is completed, you should proceed and update your local repository from any changes on the server, and then add the modified files to the staging area. For this, move to the root folder of your repository, and execute

```
git add .
```

   Commit your changes with an appropriate message and then update your server repository. Use the same steps in point 2 of this assignment.

4. Create a bash script file named ‘twice.sh’ within your assignment3 folder. The intention for script is that given an input <N>, the execution sleeps for <2N> seconds. After sleeping, it outputs “Terminated a task that takes <2N> seconds.”.

Examples of the intended execution are as follows:

```
[c001: abc123]$  bash twice.sh 3
Terminated a task that takes 6 seconds
```

(*waits 6 seconds between entering the return key and the output to the terminal executed)

```
[c001: abc123]$  bash fact.sh 8
Terminated a task that takes 16 seconds
```

(*waits 16 seconds between entering the return key and the output to the terminal executed)

Add the local changes to the staging area, then check the latest version of the repository and commit your changes with an appropriate description. Update the server repository.

5. Create a batch file named ‘twice_job.slurm’ within the assignment3 directory. This file should execute a job that consists of six `twice.sh` script calls with the following inputs: 5, 2, 7, 6, 1, 3, respectively. Each twice.sh script call should be executed as a background process. However, the batch file should wait until all twice.sh tasks have finished. Only then, proceed to print the execution time in seconds. Each call to twice script should be distributed on a single task and on a single node and with the corresponding CPUs-per-task.

   To run the twice_job.slurm file use

   ```
   sbatch twice_job.slurm
   ```

   Run the twice_job.slurm file 5 times with the following configurations:

Run	**Number of nodes**	**Number of tasks**	**Number of cores per task**	**Compute node**
1	1	1	80	compute1
2	1	2	40	compute1
3	2	4	40	compute2
4	4	1	80	compute1
5	1	4	8	amdonly

   After each run, annotate the corresponding JobID number. Use the sacct command to request accounting information. Show information including: Job ID, State, Elapsed time, Node in which each task was run, Number of CPUs used. Track this accounting information while the job is running.

   After the job is complete, run again the sacct command requesting Job ID, State, Elapsed time, Node in which each task was run and Number of CPUs used. Export this information into a text file for the corresponding run. You can use example:

   ```
   sacct -j 123456 --format=JobId,Elapsed > run1.txt
   ```

Each run should have a corresponding file. All the text files should be kept within the folder assignment1.
Note: This command is an example, and it does not export all information required for this assignment. You should complete the `--format` flag to include all parameters requested.

Hint: Update you local version of the repository bashEx (https://github.com/mauriaristi/bashEx) and look at all examples in `slurm_batch` directory. In particular:

```
bashEx/slurm_batch/parallel_multinode_run.slurm
bashEx/slurm_batch/parallel_hello_run.slurm
```

6. [Bonus] Create a bash script file named '`k_fib.sh`' within the folder assignment3. The goal of this file is given an input <N>, compute and output in a single line the K-Fibonacci series (see https://www.geeksforgeeks.org/k-fibonacci-series/?ref=lbp)

   This file should only contain bash commands, and should not call external programs like python, matlab, or any other software. This should only use bash variables, arrays, etc.