Machine Problem 0: Matrix Addition

CS 420 - Parallel Progrmg: Sci & Engrg

Due: February 8th, 11:59PM

Introduction

This assignment will introduce you to most of the infrastructure and basic abilities you will need in this course. Some tasks need to be done before we can create your turnin directories. We do not expect this assignment to be difficult or time-consuming.

Learning Goals

You will learn (and/or setup) the following (as elaborated in subsequent sections):

- Git
  - Your GitLab account hosted on Engineering at Illinois.
  - The basics of Git.
  - Submission conventions for this class.
- Computing Resources
  - Campus Cluster (CC)
    - login
    - submitting jobs
    - interactive jobs
- Matrix Addition
  - write C/C++ code.
  - perform basic benchmarking.

1 Assignment Setup

1.1 GitLab Account Creation

All of your MP submissions will be handled through the Department of Engineering GitLab server. As of this writing, we cannot force your GitLab accounts into existence. You must log in once before we can grant you access to your turnin directory.

Go to `https://gitlab.engr.illinois.edu/` , there, choose the "UOFI" tab and log in using your **NetID** and university password.

Once logged in, you may be able to begin perusing the mps for this class and your turnin repos will eventually exist at `https://gitlab.engr.illinois.edu/sp23-cs420/turnin/<NetID>/`, however, access *may* not be granted immediately.

Read Appendix A , which explains basic git usage. .

1.2 Campus Cluster Login

You will conduct all of your benchmarking on the Campus Cluster (CC). This is a standard Slurm cluster system, consisting of a login node[1] and a number compute nodes. To use such a system, you submit a *batch job* (a non-interactive script) and await the results. The system will handle scheduling all of your and other users' batch jobs to make an efficient use of available computing resources.

It is forbidden (not to mention *exceedingly bad manners*) to do any heavy computation on the login nodes. If a program is large enough, this can include compilation.

Read the CC introduction at `https://campuscluster.illinois.edu/resources/docs/start/`
Read Appendix C
Use **ssh** to log into your Campus Cluster account.

```
[self@ownmachine]$ ssh <NetID>@cc-login.campuscluster.illinois.edu
[NetID@golubh1 ~]$ _
```

Load the `git` and `intel compiler` modules. (See Appendix C.)

```
[NetID@golubh1 ~]$ module load git
[NetID@golubh1 ~]$ module load intel/18.0
```

Clone your **mp0** turnin repo to your home directory on the Campus Cluster.

```
[NetID@golubh1 ~]$ git clone <URL OF YOUR MP0 TURNIN REPO>
```

Read Appendix B

1.2.1 Submitting Jobs

Read and submit `scripts/hello_world.run` from within the cloned repo:

```
[NetID@golubh1 ~]$ cd mp0
[NetID@golubh1 mp0]$ sbatch ./scripts/hello_world.run
Submitted batch job 1234567
[NetID@golubh1 mp0]$ _
```

This will submit `hello_world.run` to the queue to run on the Campus Cluster as a job. Once it reaches its position on the queue, it will make two files, `mp0-hello.{JOB-ID}.out` and `mp0-hello.{JOB-ID}.err`. The `.out` file should have the output of the job, while the `.err` file should store all of the errors of the job to allow for debugging. For this particular script, the `.err` file should be empty and the `.out` should have

```
Hello World from golub109.campuscluster.illinois.edu
xcat
```

The particular host `golub109` can vary between runs.

2 Assignment Deliverables

2.1 Background

Addition of two $m \times n$ matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ is defined as the $m \times n$ matrix $\boldsymbol{C}$ where $c_{ij} = a_{ij} + b_{ij}$. Note that this operation is not well-defined when $\boldsymbol{A}$ and $\boldsymbol{B}$ have different dimensions.

2.2 Matrix Addition

This assignment asks you to write a few lines of C/C++ code to implement the function `matrixAdd()` in the `add.c` file. You should not have to edit any other functions or files. The code in `main.c` reads two input matrices from standard output and prints the execution time of `matrixAdd()` called on them (Do not edit this file). To compile and run the main executable, run the commands below, where `Width` and `Height` are the width and height of the desired matrices (Remove < and > when inserting the dimensions).

```
[NetID@golubh291 mp0]$ ./scripts/compile_script.sh
[NetID@golubh291 mp0]$ cd build/add
[NetID@golubh291 add]$ ./randMatrix <Width> <Height> <Width> <Height> | ./add
```

[1] Actually a handful of load-balanced login nodes.

To explain what is happening, `./scripts/compile_script.sh` runs a series of commands that compiles the programs and creates the executables `add` and `randMatrix`. We then go to the folder that those executables are in with `cd build/add`. `randMatrix` accepts the four parameters explained above and creates 2 random matrices of those sizes. Those matrices are then fed to the `add` program with the `|` (pipe command). The output of the program is then printed to the screen.

We have provided with you a sample test program. You can run the tests from the root folder with

```
[NetID@golubh291 mp0]$ python tests/test.py
```

It should print with either an `OK` or `FAILED` if it passes or fails the given tests respectively. If you fail any of the tests, it is extremely likely that you have an error in your code and you should go back to solve it.

You can manually verify that the output matrix is the sum of the two input matrices by creating and saving an sample input, passing the test input into the program and calculating the sum yourselves. The following commands will save the two input matrices in `myTest.txt` and the resulting sum matrix in `myOutput.txt`. You can print the files to the terminal with `cat <FILENAME>`. *Please use small test sizes (< 20) for height and width since this will be on the login nodes* (Please keep the last > before `myTest.txt` in the second line)

```
[NetID@golubh291 mp0]$ cd build/add
[NetID@golubh291 add]$ ./randMatrix <Width> <Height> <Width> <Height> > myTest.txt
[NetID@golubh291 add]$ cat myTest.txt | ./add > myOutput.txt
```

2.3 Writeup

For your writeup, do the following:

- Create a table of the execution times of `matrixAdd()` on two random input matrices of sizes 10000x10000 and 10000x10000. Run for 5 iterations.
  Hint: Edit scripts/batch_script.run to do this.

  We can do this by executing the 'add' program for the required number of times. We can then append the results to the same file or use different files.

  You can then execute the below command to submit *scripts/batch_script.run*

  ```
  [NetID@golubh291 mp0]$ bash ./scripts/submit_batch.sh
  ```

3 Submission Guidelines

Please name your writeup as "`writeup/mp0_<NetID>.pdf`*".*

Submit the following files to the **main** branch of your turnin repo before the submission deadline:

- `matrix.c`
- `writeup/mp0_<NetID>.pdf`

Also, submit the write-up to Gradescope.

Appendices

A Git Usage (Basic)

Git helps with version control of the files you work on for the MPs. Here are some basic commands that can help you get started:

1. `git clone <URL of the repository>` : Clone the repository in the directory this command is executed in.
2. `git pull` : Pull the latest changes from the remote repository.
3. `git add <name of file or directory>` : Add a file or directory so that Git can start tracking it for version control and make it a part of the repository.
4. `git commit -m <commit message>` : Commit/save your changes locally with the specified commit message describing the changes made.
5. `git push origin <branch name>` : Push your changes to the remote repository on the specified branch.

For further help regarding Git, here is a helpful resource: `https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html` . If there is any confusion, feel free to seek help from the TAs.

A.1 When things go wrong

An important use for a SCM system is to be able to fix your mistakes. But what if you make a mistake in using your SCM? Despite its foul language, this website is very useful for figuring out how to fix apparently drastic mistakes when using git:
`http://ohshitgit.com/`

B Slurm (sbatch) Basics

Some useful commands for accessing the job scheduler are:

1. `sbatch <script>` : submit the job with name in place of ¡script¿ to the scheduler. Prints the new **job_id** to the terminal.
2. `scancel <job_id>` : delete the job with sequence number job_id from the scheduler.
3. `squeue -u <NetID>` : use this command to check the status of the jobs you have submitted to the scheduler.
4. `srun --time=00:30:00 --nodes=1 --ntasks-per-node=16 --pty /bin/bash` : start an interactive job for 30 minutes on 1 node with 16 tasks.

You can find the Slurm quickstart guide here: `https://slurm.schedmd.com/quickstart.html`

C Environment Modules

The cluster has several packages for software development. You can add or delete necessary pack- ages with the module command.

Some examples are provided below:

1. `module avail` : provides a list of all available packages.
2. `module list` : provides a list of packages that are currently available in your environment.
3. `module load <package>` : add the software package to your environment.
4. `module unload <package>` : removes the software package from your environment.
5. `module initadd <package>` : the package will be loaded automatically the next time you log in.

Some useful packages that you might want to load while working:

1. `module load vim`
2. `module load git`