Machine Problem 0

CS 484 - Parallel Programming

Due: January 27, 2019 @ 23:59

(Typeset on : 2019/01/13 at 22:07:21)

Introduction

This assignment will introduce you to most of the infrastructure and basic abilities you will need in this course. You should complete it immediately; 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, but it is urgent.

Please complete it immediately.

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
  - VMFarm: (login and usage)
  - Campus Cluster (CC)
    - login
    - job submission (qsub, qdel, qstat, etc.)
- Basic Benchmarking

1 Assignment Tasks

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, at `https://gitlab.engr.illinois.edu/sp19-cs484/mps/` .

Your turnin repos will eventually exist at `https://gitlab.engr.illinois.edu/sp19-cs484/turnin/<NetID>/`, however, access *may* not be granted immediately.

Read Appendix A , which explains basic git usage. .

1.2 VMFarm Login

For interactive/iterative programming, you have been granted a private VMFarm machine with a complete installation of all necessary software for this class.[1] In general, you should do your programming / correctness testing on this machine, and only move to the Campus Cluster for benchmarking.

Use **ssh** to log into our VMFarm machine. (`sp19-cs484-001.cs.illinois.edu`)

```
[self@ownmachine]$ ssh <NetID>@sp19-cs484-001.cs.illinois.edu
[NetID@sp19-cs484-001 ~]$ _
```

Clone your **mp0** turnin repo to your home directory on your VMFarm machine.

```
[...~]$git clone https://gitlab.engr.illinois.edu/sp19-cs484/turnin/<YourNetID>/mp0.git
[...~]$cd mp0
[...mp0]$ _
```

Copy the file "`/etc/cs484-identifier`" into the "`writeup`" directory of your repo then add/commit/push it. (See Appendix A)

1.3 Campus Cluster Login

You will conduct all of your benchmarking on the Campus Cluster (CC). This is a standard PBS/Torque cluster system, consisting of a login node[2] 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` module. (See Appendix C.)

```
[NetID@golubh1 ~]$ module load git
```

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

```
[NetID@golubh1 ~]$ cd mp0
[NetID@golubh1 mp0]$ qsub ./scripts/hello_world.run
1234567.cc-mgmt1.campuscluster.illinois.edu
[NetID@golubh1 mp0]$ _
```

1.3.1 Singularity on Campus Cluster

To ensure environment consistency, we expect you to benchmark your programs using a provided Singularity container. This creates an environment that is nearly identical to the environment on the VMFarm. This will also be used for consistency when grading.

We have provided useful scripts so that you will not need to interact with singularity directly.

Read and submit `scripts/hello_world_singularity.run` from within the cloned repo (on CC):

[1] Students who are familiar with Docker may choose to do their development on an identical Docker container, but must still complete this portion of the assignment on the VMFarm. Benchmarking must still be done on CC.
The container is named **uiuccs484parallelprog/cs484_student** and is available at `https://hub.docker.com/r/uiuccs484parallelprog/cs484_student` .
An explanation of how to use Docker and support (other than ensuring image correctness) will not be provided.

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

```
[NetID@golubh1 mp0]$ qsub ./scripts/hello_world_singularity.run
1234568.cc-mgmt1.campuscluster.illinois.edu
[NetID@golubh1 mp0]$ _
```

Compare `scripts/hello_world.run` and `scripts/hello_world_singularity.run` .
The line

```
#PBS -S <path_to_shell>
```

gives the path to a shell to interpret commands in. (A default is used if not given.) In this class, we will generally use this singularity shell, which we have provided.
If you end up writing your own batch scripts, remember to use this shell!

1.4 Benchmarking

Read and submit `scripts/batch_script.run` from within the cloned repo (on CC):

```
[NetID@golubh1 mp0]$ qsub ./scripts/batch_script.run
1234569.cc-mgmt1.campuscluster.illinois.edu
[NetID@golubh1 mp0]$ _
```

1.4.1 Part1

This assignment introduces you to spatial and temporal locality in memory accesses. We have provided you with a code that performs a series of irregular accesses to a contiguous array in memory ("`part1/part1.cpp`"). The code tests arrays of different sizes, but the total number of memory accesses is kept constant for all array sizes. You don't have to make any changes to this program. The program was already compiled and executed when the above batch script ran. Its output should be available in "`./writeup/part1_out.txt`".

For your writeup, do the following:

- Read the code and understand how the arrays are accessed in a irregular fashion. (You do not need to write anything for this.)
- Observe how the execution time changes as the array size grows, and briefly explain what you think is causing the variation. (**Hint:** It has to do with a hardware component that is part of a CPU. Information about the CPU was saved in "`./writeup/cpuinfo.txt`" when the benchmark ran.)

1.4.2 Part2

This section is meant to familiarize you with how compiler optimizations impact performance. We have provided you with five small functions. ("`./part2/test*.cpp`")
The batch script compiles these tests with `-O0` (without optimization), `-O3` (with the most aggressive optimizations) optimization flags, runs those programs, and saves the output in files named "`./writeup/part2_*.txt`"

In your writeup:

- Create a table of the execution times of the 10 cases (5 functions x 2 optimizations) and state if the compiler optimizations have actually increased the performance.

2 Submission Guidelines

Your writeup must be a **single PDF file** that contains the tasks outlined above.

Please save the file as "`./writeup/mp0_<NetID>.pdf`" , commit it to git, and push it to the **master** *branch of your turnin repo before the submission deadline.*

In addition to the writeup, make sure that your writeup directory contains the file "./writeup/cs484-identifier" from the VMFarm part of the assignment.

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 PBS/Torque (qsub) Basics

Some useful commands for accessing the job scheduler are:

1. `qsub <script>` : submit the job with name in place of ¡script¿ to the scheduler. Prints the new **job_id** to the terminal.
2. `qdel <job_id>` : delete the job with sequence number job_id from the scheduler.
3. `qstat -u <NetID>` : use this command to check the status of the jobs you have submitted to the scheduler.

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`