CS420: Parallel Programming - Fall 2017
MP0
Due Date: September 20, 5:00PM
Submission Method: SVN

September 6, 2017

1 Background

This is an introductory assignment to familiarize yourself with running programs on a computing cluster and analyzing their performance. For this assignment, you will be running code on UIUC Campus Cluster.

From a Linux/MacOs console, to access the cluster, you can use the following command:

```
ssh -YC NetID@cc-login.campuscluster.illinois.edu
```

Note: NetID should be replaced in all commands with your UIUC NetID.

From a Windows machine, you can use `PuTTY` or other SSH clients.

The cluster uses the `module` command for loading and unloading packages (`http://linux.die.net/man/1/module`). Below is a short list of commands for managing packages:

- Viewing the list of modules: `module avail`
- Viewing the loaded modules: `module list`
- Loading a module: `module load MODULE`
- Unloading a module: `module unload MODULE`

Note: MODULE is a specific package. Following examples will load vim editor and Intel toolkits (including a compiler - `icc`):

```
module load vim
module load intel/17.0
```

When logging into the cluster, you login to the *login node* of the cluster. To use the *compute nodes*, the cluster uses a job submission program called *PBS* (`http://www.upv.es/upl/U0115421.htm`). The typical usage is to write a submission script which specifies properties of the job and what programs to run. For this assignment, we provide the necessary submission scripts. Below are a few useful commands:

- Submitting a job: `qsub` SCRIPT
- Querying all user jobs: `qstat -u` NETID
- Deleting a job: `qdel` JOBID

where SCRIPT is the submission script, NETID is your Net ID, and JOBID is a numerical number representing the job as given by the provided `qstat` command. For more information on the cluster, visit `https://campuscluster.illinois.edu/user_info/doc/beginner.html`.

Below are some documentation for useful commands:

- ssh : `http://www.openbsd.org/cgi-bin/man.cgi?query=ssh&sektion=1`
- scp : `http://www.openbsd.org/cgi-bin/man.cgi/OpenBSD-current/man1/scp.1?query=scp&sec=1`
- tar : `http://www.linfo.org/tar.html`
- svn : `http://www.tutorialspoint.com/svn/`
- wget : `http://www.gnu.org/software/wget/manual/html_node/Simple-Usage.html`

Note: Cluster nodes are a shared resource, so start early to avoid issues!

2 Assignment

To begin, upload mp0.tar to your Campus Cluster account (using scp from Linux console):

```
scp mp0.tar NetID@cc-login.campuscluster.illinois.edu:/home/NetID
```

Then login to the cluster and extract the contents of `mp0.tar`:

```
tar -xvf mp0.tar
```

It should create a folder `mp0` with two sub-directories, `mp0_a` and `mp0_b`.

2.1 Part A

This program (`mp0_a`) measures the time taken for different data access patterns on a contiguous array stored in memory. You will be using this code to test your access to the machines.

This program in part A updates between 64 elements and 67 million elements of a contiguous array. Each update step is repeated several times so that there are 100 million memory accesses no matter the number of elements updated. The program measures the time spent in each update step, and prints out the average memory access time for a given number of array elements.

After you have extracted the files, do the following:

- Setup Intel compilers using `module` command.
- Run `make clean`
- Run `make`: This should create the executable `mp0_a.exe`.

To run the file, you need to submit the provided runscript:

```
qsub -q cs batch_script.run
```

When the job completes, you should find a file `out_mp0_a.txt` in your `mp0_a` directory with the output of the code.

Note: You can run it directly in the login node, but the performance will be unpredictable as the login node is shared with many users.

2.2 Part B

This assignment is meant to help you understand some useful compiler optimizations if you are not already familiar with them. The program `mp0_b.c` has five different functions, each running simple loops with array accesses.

We want you to execute the same code with different compiler optimization levels and explain the change in performance in terms of cache performance, vectorization etc.

The different optimization levels you should try are `-O0`, `-O2`, and `-O3`.

Follow the steps described in the previous section to login to your cluster account, extract the files and set up the Intel compiler. Now, do the following steps:

- cd `mp0_b`
- Run `make clean` followed by `make`. This should create the executable `mp0_b.exe` in the `mp0_b` directory
- Submit the job to execute on the cluster using qsub and the run script provided to you.

Now open `Makefile` that is provided to you and edit it using your favorite editor on the cluster. Look for `CCFLAGS = -O0`, which passes the optimization level you need for your program to the compiler; zero in this case. Change this to compile with the wanted optimization levels, comparing with the base case -O0.

In order to get some insight to the performance data, you may edit the Makefile to add more options, here are some hints:

- Read the generated *compiler report* for the binary. This can be generated by using the compiler flag `-qopt-report`
- Read the generated assembly code. This can be generated by using the compiler flag `-S`
- Looking at the Intel Compiler manual which describes what optimizations are performed.

3 Submission

If you are working on your local machine, install svn. If you are on Campus cluster, enable svn using `module load svn`.

Check out your directory in the course repository, and add a folder for mp0:

```
svn co https://subversion.engr.illinois.edu/svn/fa17-cs420/
cd NetID
mkdir mp0
svn add mp0
svn commit -m "Adding mp0 folder"
```

Create two text files (NetID_mp0_a.txt and NetID_mp0_b.txt in the directory mp0). Copy and label the different outputs you collected from your experiments.

Then, discuss your results for both Part A and Part B. For Part B, write down which loop optimization the compiler applied for each test case (e.g : vectorization) and why the optimization resulted in better performance; e.g: Vectorization uses the vector registers which can execute multiple (depending on the width of the vector register) loop iterations simultaneously provided there are no dependencies.

To submit your files, place them in the NetID/mp0 folder. Then do the following to add your files:

```
cd mp0
svn add NetID_mp0_a.txt
svn add NetID_mp0_b.txt
svn ci -m "NetID submitting mp0"
```

Please do not add and commit any other file to your svn.