High-Performance Scientific Computing (MATH-GA 2011/ CSCI-GA 2945)

Homework Set 1

Due: September 12, 2012 · Out: September 5, 2012

Welcome to this semester's high-performance computing ("HPC") class! Remember that this class is about *you* and how much you learn. Please let us (the instructors) know what we can do to improve your experience.

This first assignment is meant to get you familiar with the mechanics of this class, from asking questions, to compiling and running code, to turning in homework. It'll also provide some practice with C code. If you're unfamiliar with C or the tools we are using, expect to spend a fair bit of time *now* to catch up. This will ensure that you won't be lost once we get to more complicated things. If you're already a skilled hacker on the other hand, you should fly through this assignment.

After the description of the assignment, you will find a few pages that describe tools and procedures. Please make sure to read and follow those carefully.

Problem 1: Sorting with trees

Write a program to sort a sequence of numbers using a binary tree[1]. Define a `struct` to represent each node in the tree.

a) Read a number `n` from the command line.

b) Allocate an array of `n` integers. Using `rand()`[2], fill it with `n` random integers. Make sure to use `srand()` to pick a seed, for reproducibility.

c) Build a binary tree from your array of numbers.

d) Output the sorted list to `stdout`, one integer per line.

e) Make sure you free all memory that you've allocated. (Yes, we will check!)

f) As an expression of `n`, what is the worst possible runtime? What runtime do you expect typically? What determines which case you fall into? (Don't worry about constant factors or lower-order terms in your expressions.)

g) Use this timing code[3] to measure the run time you get for a range of list sizes. Discuss how your observations match with your prediction in part f).

You will turn in your solution as a '`git`[4]' repository (see below). Your submission should be in a subdirectory '`problem-1`' and should consist of at least these two files:

1. Your program, which should be called '`tree-sort.c`'.

[1] `https://en.wikipedia.org/wiki/Binary_tree`
[2] `https://www.gnu.org/software/libc/manual/html_node/ISO-Random.html`
[3] `https://gist.github.com/3614336`
[4] `https://en.wikipedia.org/wiki/Git_%28software%29`

If your implementation consists of more than this one file, you must also supply a '`Makefile`[5]' that will build your program when '`make`' is entered. (If you have never heard of `make`, ignore it for now, we will get to it.)

2. A plain text file '`discussion.txt`' containing your solution to parts f) and g) as plain text. (i.e. not Word, PDF, or some such)

   Your answer to each subproblem should be about a short paragraph.

Problem 2: Matrix multiplication

Write two codes to multiply matrices.

a) Read a number `n` from the command line. Allocate two (one-dimensional) arrays `A` and `B` of n^2 `doubles`. Fill them with random numbers between 0.5 and 2.

b) Viewing each of the one-dimensional arrays as a two-dimensional array in column-major order[6] (see Figure 1), we can now represent a matrix. Allocate a third array `C` and write a function to fill it with the product of the two matrices, according to this procedure:

```
for i = 1 to n
  for j = 1 to n
    sum = 0
    for k = 1 to n
       sum = sum + A[i,k] * B[k,j]
    end
    C[i,j] = sum
  end
end
```

(Note that you'll have to translate the two indices into one. You might want to consider using a macro for this type of access.)

c) Write a subroutine that outputs a matrix to the screen as a square array as a human would write it. Use this subroutine to check your work in part b) on a small example. Output five characters per number (plus a space), and make sure the columns of your output line up.

d) Now write a subroutine that transposes a matrix, i.e. that computes

```
for i = 1 to n
  for j = 1 to n
    D[i,j] = A[j,i]
  end
end
```

Realize that changing the matrix from column- to 'row-major' storage order is the same thing as transposition.

e) Now make a new function that also performs matrix multiplication, but accepts `A` in row-major order.

[5] `https://en.wikipedia.org/wiki/Make_%28software%29`
[6] `http://en.wikipedia.org/wiki/Row-major_order`

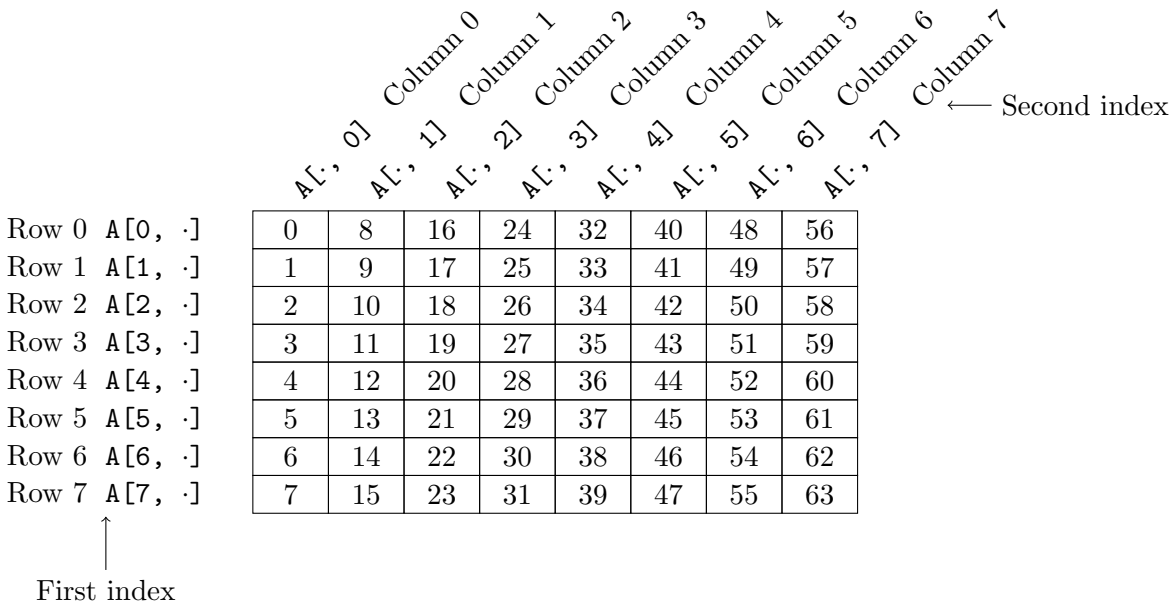

Figure 1: Column-major matrix layout. The numbers in the cells represent linear array indices.

Write some code to check that the results from this routine agree with your existing code from part b).

f) Measure the run time of your code for parts b) and e) for a few values of n. If one is faster, for what matrix size `n` does it become efficient to change the storage order, assuming `A` is always given in the 'slow' order? (Use the timing code from the first problem. You will also need to time your transposition routine.)

Also try different optimization levels by giving flags `-O0`, `-O1`, `-O2`, `-O3` to the compiler. Also try `-ffast-math`, which sets a number of potentially dangerous options, and the safer `-fassociative-math`. Read up on their meanings.

Write up your findings.

Please submit the following, in the subdirectory '`problem-2`' of your repository:

1. A program `matmul.c` solving the above assignment. `main()` in this file may *only* consist of calls to independent functions `part_a`, `part_b`, so that each part can be run in isolation by commenting out the other parts.

   You will need subroutines for at least parts b), c), and e).

   Again, please make sure that you free all the memory you allocate.

2. A plain text file `part-f.txt` with your measurements and discussion from part f).

Machines and Homework Submission

Helpful Hints

Getting Help

If you are having technical trouble, if some instructions here fail to work for you, or if things just seem confusing, don't despair: There's plenty of help available.

First of all, please make sure you are subscribed to the class mailing list. Someone might have had the same problem as you (and ideally already solved it). If you're either officially signed up in Albert or added your name to the sign-up sheet in class, you should already be on the list–you can tell by whether you've received a welcome email on Wednesday night. If you aren't subscribed, you may do so yourself at the list's web page[7]. The list has archives[8] where you can check if you've missed anything important.

If your problem concerns the NYU HPC machines (which we'll use later in the class), try checking the NYU HPC Wiki[9]. If that (and perhaps a bit of Googling around) doesn't answer your question, send email to

`hpc12@tiker.net`.

If you would like to discuss a technical issue (i.e. one that isn't directly related to you), please do not email us (the instructors) directly–we'll just tell you to ask on the list. So instead, please send a message to the mailing list, where we (and your peers) will be more than happy to assist. Thanks!

Again, welcome, and we hope you'll have an enjoyable and worthwhile experience in this class.

Note 1 *All work in this class will be UNIX-based. We've provided a virtual machine ('VM') that lets you use all the technologies we'll touch on in this class. (C99, OpenMP, MPI, OpenCL) Note that things won't go as fast in the VM as on 'real hardware', but it's more than enough for coding and debugging.*

At some later point, we will also introduce you to the usage of the shared-use clusters maintained by the NYU HPC group[10].

First, install VirtualBox[11] and run it. Then, download the following machine image:

`http://bit.ly/hpc12-vm`

If you are *certain* that you are running a 64-bit operating system, you may instead grab this image (it will run slightly faster and also have the Intel OpenCL implementation):

`http://bit.ly/hpc12-vm-64`

[7] `http://lists.tiker.net/listinfo/hpc12`
[8] `http://lists.tiker.net/private/hpc12/`
[9] `https://wikis.nyu.edu/display/NYUHPC`
[10] `http://hpc.nyu.edu`
[11] `http://virtualbox.org`

These files are about 2 GB in size each, so it'll take a while to download. (You'll only need one of them!) If possible, grab it at NYU using a wired connection. If your browser didn't do this for you, you may have to rename the file so that it has a `.ova` file extension. If you're unable to download, ask one of the instructors–we have a USB stick with the image and VirtualBox (the software that runs it).

Once downloaded, click "File — Import Appliance..." to import the VM into VirtualBox. This takes a few minutes, after which you can delete the file you downloaded.

Note 2 *Once the VM is done importing, you should change it to use the number of cores actually present in your machine. To do so, right-click the entry 'HPC Class Linux VM', then click 'Settings...', 'System', and 'Processor'. Then slide the 'Processor(s)' slider to the number of cores you'd like to use. Avoid the red area.*

You can then start the VM by double-clicking on 'HPC Class Linux VM', and you'll be presented with a Linux desktop. You can make that desktop full-screen, if you wish.

You can of course also use your own Linux (or OS/X) machine, but then you're on your own with respect to installing the software that we'll need.

Using Unix and the command line

While the VM provides a friendly point-and-click interface, we will be doing most of our work on the command line, which you can get by double-clicking the 'Terminal' icon on the desktop (and by many other ways).

While we hope you'll eventually agree that the command line is great for doing development work, it can be a bit intimidating at first.

Once you click the icon, you're greeted with a 'prompt':

```
hpc@dizzy:~$
```

and a cursor next to it. `hpc` is your user name, `dizzy` is the name of the VM. `~` is a shorthand for your home directory, and this spot in the prompt generally shows the current working directory you're in. `$` finally is the prompt itself.

Note 3 *We'll be a bit briefer from here on out. From now on, the dollar sign "`$`" represents the command prompt.*

If you are unsure of what to type next, consider taking a look at a friendly Unix tutorial[12] (recommended). (Googling for "unix tutorial" gives plenty more options.)

You will also need to edit plain text files. There are plenty of text editors that work in the terminal (nano, emacs, vim, all installed in the VM). If you've never used any of these, the 'Text Editor' link on the desktop of the VM provides provides a simple text editor that's not scary and works a bit like Microsoft Word. You can also start that editor from the prompt:

```
$ gedit filename.c
```

[12] `http://www.ee.surrey.ac.uk/Teaching/Unix/`

(Remember that you don't have to type the `$`.)

Also, note that you will get a bash[13] shell by default. If you are used to the C shell, ask us on how to change that. (If this was mumbo-jumbo to you, forget about it.)

Using `git` for Homework and Collaboration

We expect you to use the distributed version control[14] system git[15] for developing and turning in solutions to your homework. Like compilers, debuggers, and build tools, version control systems are central to most present-day software development. By having you use `git`, we hope to be able to give you some familiarity with such tools.

Along with the tool itself, we will use a central collaboration space, `http://forge.tiker.net`, which works like the big, public development hubs Github, Google Code, or SourceForge. You will set up an account there and create a new "project" for every assignment (and your final project). The collaboration space provides the following functions in our class:

- We will grade what is on `forge` after the homework deadline. As a result, you definitely want to upload your finished assignment.
- It will allow you to collaborate on your final project with a friend.
- If you are having an issue with your current code, you may upload it and ask the instructors to view it on `forge`.
- It provides an easy conduit to get code from the HPC clusters onto another machine and back.

Note 4 *Your account on `forge` is initially limited to about 50 megabytes, with more available if needed. Please do not check in large data files before discussing your needs with Andreas.*

Your accounts on `forge` and the data you store there will be removed on February 1, 2013. Please back up your code and data before this date.

Git is preinstalled on our virtual machine.

To start, open a terminal (use the icon on the VM desktop) and inform `git` of your name and email address:

```
$ git config --global user.name "Your Name"
$ git config --global user.email you@yourdomain.example.com
```

To access the class-wide collaboration space, go to `http://forge.tiker.net`. Click "Sign in or create account". Please use your NetID as your user name. The system will confirm your email address by sending you a confirmation key. You can use the web browser in the VM to do this.

Note 5 *We have found that these confirmation emails often get flagged as spam by a variety of spam filters. Please check your spam folder if the email doesn't seem to have arrived within a few minutes.*

[13] `http://en.wikipedia.org/wiki/Bash_(Unix_shell)`
[14] `http://en.wikipedia.org/wiki/Distributed_revision_control`
[15] `http://git-scm.org`

Once you complete this verification step, fill out your name, password, and click "Enable my Account". Please do *not* use a valuable password here, in particular *not* your NYU-wide one. You are now logged into `forge.tiker.net`.

Next, type

```
$ ssh-keygen
Generating public/private rsa key pair.
# Just hit Enter for this prompt.
Enter file in which to save the key (/home/hpc/.ssh/id_rsa):
Created directory '/home/hpc/.ssh'.
# Pick a password here, or leave empty if you're feeling lucky.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hpc/.ssh/id_rsa.
Your public key has been saved in /home/hpc/.ssh/id_rsa.pub.
The key fingerprint is:
df:0d:72:0b:19:f6:c4:ec:10:8b:0d:45:69:84:55:0a hpc@dizzy
The key's randomart image is:
+--[ RSA 2048]----+
|       E**o.     |
|       .=o*      |
|       ..B +     |
|        . B      |
|       S + =     |
|        . = +    |
|         . o .   |
|                 |
|                 |
+-----------------+
$ cat $HOME/.ssh/id_rsa.pub
```

(Again, don't type the `$` signs...) This will print a few lines (technically, just one wrapped line) that looks like this:

```
ssh-rsa AAAABw...vM3XdIbZWwmXH/iNbFWEhZw== hpc@dizzy
```

Once you are logged into `forge`, click your name in the top left corner. Click "Update your account" on the left and then paste the line returned above into the field that says "Add public key". If you get errors, make sure that there are no explicit newlines in the key. Click "Update your Account". The key update may take up to two minutes to fully complete, so if the '`git push`' below fails, wait for two minutes, and then try again. This completes the setup steps that are only required once.

Next, create a directory for your homework submission.

```
$ mkdir hw1
$ cd hw1
$ git init
Initialized empty Git repository in /home/hpc/hw1/.git/
$ git status
On branch master
```

```
Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

The '`init`' command told `git` that you would like it to manage this directory. Now go ahead and start on your project. Once you've gotten to a point that you would like to save, say

```
$ git add file1.c file2.txt
$ git commit
```

It is easiest to `git add` one file at a time, from the directory in which the file resides.

Note 6 *You* can *add entire directories at once, but this is often a bad idea. The process typically picks up lots of files (binaries, backups, output, data...) that you did not mean to check in. If you add some of this stuff by accident, you can get rid of it by* `git rm filename`.

`git` will ask you for a commit message in a new `gedit` window. (On other machines, this might drop you into `vi` or some other horror. We'll show you how to deal with that later.)

Note 7 *You should not check in backups (ending in* `~`*), executable or binary files such as those ending in* `.o` *or ones called* `a.out`*. These can easily be rebuilt from the source code and therefore do not need to be saved.*

After your first commit, let's save your repository to the class collaboration space. To do this, navigate to `http://forge.tiker.net/` (or click on "Project List" if you're already there) and, once there, click on "Create Project". Note that this assumes you have created an account and are logged in, as described above.

You may choose a name for the new repository, as well as a "shortname" (i.e. Unix-level name).

Note 8 *To make sure we find your homework, please choose this shortname following the pattern "*`hpc12-hw1-netid123`*". If you don't, we likely won't find your submission!*

For the "Project Creation Key", enter "`nyuhpc12`" and make sure the "Private repository" box is checked. If necessary, you may also add co-owners and collaborators by their NetID. (Note that you're free to discuss homework with your peers, but you *must* write your own code. In other words, collaboration will likely only become relevant once we get to the final project.) Once you click "Create Project", your repository is created on forge. To establish the link with your local one, say

```
$ git remote add forge ssh://git@forge.tiker.net:2234/<shortname>.git
```

You may then say

```
$ git push forge master
# If you entered a password in ssh-keygen above, you'll be asked
# to enter it again now.
Counting objects: 34, done.
Delta compression using up to 8 threads.
```

```
Compressing objects: 100% (21/21), done.
Writing objects: 100% (34/34), 8.25 KiB, done.
Total 34 (delta 12), reused 30 (delta 11)
To ssh://git@forge.tiker.net:2234/hw1-ak177.git
 * [new branch]    master -> master
```

to upload your commits.

You may also navigate to `http://forge.tiker.net`, pick your project from the list, navigate to the "Source" tab and verify that the right version of the code is checked in.

From here on out, the cycle of add/commit/push just repeats. Note that you need to `git add` a file every time you've changed it. Please also take a look at the git tutorial[16]. You should at least be familiar with the usage of the commands

- `git status`
- `git add`
- `git commit`
- `git log`
- `git diff`

One piece of advice: Commit frequently–at least once for every milestone you complete along the way. This allows you to go back to a working version if you've made changes that you would like to get rid of.

[16] `http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html`