Lab 5: Loop-level Parallelism

Instructor: Vivek Sarkar, Co-Instructor: Mackale Joyner

Course Wiki: http://comp322.rice.edu

Staff Email: comp322-staff@mailman.rice.edu

Goals for this lab

- NOTS Setup (Section 1)
- An introduction to the shell (Section 2)
- Experimentation with Loop-level Parallelism and Chunking in Image Convolution (Section 3)

Downloads

As with previous labs, the provided template project is accessible through your private SVN repo at:

```
https://svn.rice.edu/r/comp322/turnin/S17/NETID/lab_5
```

For instructions on checking out this repo through IntelliJ or through the command-line, please see the Lab 1 handout. The below instructions will assume that you have already checked out the lab_5 folder, and that you have imported it as a Maven Project if you are using IntelliJ.

1 NOTS setup

NOTS (Night Owls Time-Sharing Service) is a Rice compute cluster designed to run large multi-node jobs over a fast interconnect. The main difference between using NOTS and using your laptop is that NOTS allows you to gain access to dedicated compute nodes to obtain reliable performance timings for your programming assignments. On your laptop, you have less control over when other processes or your power manager might start stealing cores or memory from your running parallel program.

Prior to lab, you should have completed the setup instructions from https://piazza.com/class/ixdqx0x3bjl6en?cid=117 If you have not, please do so now. These instructions ensure that we have 1) an SSH client that allows us to remotely access a terminal on NOTS, and 2) some method of uploading and downloading files to NOTS (either through SCP or SFTP). These instructions will focus on the use of SCP, but if you are familiar with SFTP (e.g. FileZilla) it should be straightforward to use that instead. You also have the option of using your SVN turnin folder to handle transferring files to and from the cluster, but the teaching staff recommends becoming familiar with tools like SCP and SFTP as they are very useful and widely used.

- Start by logging in to NOTS. From the command line on Mac/Linux:

  ```
  $ ssh ⟨your-netid⟩@nots.rice.edu
  ⟨your-netid⟩@ssh.clear.rice.edu's password:  ⟨your-password⟩
  ```

  On Windows, simply open Putty and enter `nots.rice.edu` as the hostname you want to access, then click "Open" and follow the prompts. Your password should be the same as the one you use to log in to other Rice services like CLEAR or Owlspace. Note that this login connects you to a shared login node, not a dedicated compute node.

- After you have logged in to NOTS, run the following command to setup the JDK8 and Maven path. Do not replace "jmg3" with your net ID.

  ```
  source /home/jmg3/comp322/322_setup.sh
  ```

  Note: You will have to run this command each time you log on NOTS. You should add this command to the bottom of the file ∼/.bash_profile so that it will run automatically each time you log in.

- Check your installation by running the following commands:

  ```
  which java
  ```

  You should see the following: `/opt/apps/software/Core/Java/1.8.0_45/bin/java`

  Check java installation:

  ```
  java -version
  ```

  You should see the following:

  ```
  java version "1.8.0_45" Java(TM) SE Runtime Environment (build 1.8.0_45-b14) Java
  HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
  ```

  Check maven installation:

  ```
  mvn --version
  ```

  You should see the following:

  ```
  Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T10:41:47-06:00)
  Maven home:  /home/jmg3/install/apache-maven-3.3.9 Java version:  1.8.0_45, vendor:
  Oracle Corporation Java home:  /opt/apps/software/Core/Java/1.8.0_45/jre Default
  locale:  en_US, platform encoding:  UTF-8 OS name:  "linux", version:  "3.10.0-229.el7.x86_64",
  arch:  "amd64", family:  "unix"
  ```

- When you log on to NOTS, you will be connected to a *login node* along with many other users (type `users` at the terminal if you would like to see all the other people with active sessions on the same machine). Running your performance experiments on the login node would face the same problems as on your laptops: in a multi-tenant system, it is hard to prevent interference with your performance results. Once you have an executable program, and are ready to run it on the compute nodes, you must create a job script that configures and runs your program on a dedicated compute node. The following items will walk you through editing a job script we have provided, uploading your Lab 5 project to NOTS, and then launching a compute job on NOTS using the job script.

- The job script will provide information on your job such as the number of nodes you want to use (usually just one in this course), the amount of time you want your job to be allowed to run for, the amount of memory your job will need, as well as the actual commands you want to be executed as part of your job. We have provided a script template in the Lab 5 template code at:

  ```
  lab_5/src/main/resources/myjob.slurm
  ```

  You only need to change the line marked "TODO". The change on line 8 of that file indicates an e-mail address to send job status notifications to.

- Now, we want to transfer your `lab_5` folder from your local machine up to NOTS so that we can submit the Lab 5 code to the cluster for testing. NOTE: You must place your `lab_5` folder directly inside of your `$HOME` directory, so that the absolute path to it is `/home/[net-id]/lab_5`. To transfer a folder to NOTS, you can use one of two methods:

  - Use Subversion: You can commit your local changes to SVN. Then you can checkout or update the project on your NOTS account using one of the following:

    ```
    svn checkout https://svn.rice.edu/r/comp322/turnin/S17/NETID/lab_5/
    ```

    or, if you have already checked out the SVN project on your account,

    ```
    svn update
    ```

    The `svn checkout` command must be run from your `$HOME` directory.

  - Use SCP: Use the following command on your local machine to transfer your `lab_5` folder to NOTS:

    ```
    scp -r /local/path/to/lab_5 [your-net-id]@nots.rice.edu:~/
    ```

- Now that we have edited the Lab 5 job script and uploaded the Lab 5 folder to NOTS, we can submit a test job to the cluster. To submit the job, run the following command on NOTS:

  ```
  sbatch /path/to/my/lab_5/src/main/resources/myjob.slurm
  ```

  After you have submitted the job, you should see the following:

  ```
  Submitted batch job [job number]
  ```

- To check the status of a submitted job, use the following command:

  ```
  squeue -u [your-net-id]
  ```

  If no jobs are listed, you have no running jobs.

- Because NOTS is shared by many users, it may take some time for your job to make it through the job queue and get assigned a compute node. If you would like to see an estimate of when your job will start, you can add the `--start` flag to the `squeue` command:

  ```
  squeue -u [your-net-id] --start
  ```

  Note that adding `--start` will only show jobs that are pending and have not started yet. If your job is already running, you will see it listed under `squeue -u [net-id]` but not `squeue -u [net-id] --start`.

- To cancel a submitted job, use the following command:

  ```
  scancel [job-id]
  ```

  When your job finishes running, your should see an output file titled `slurm-[job-id].out` in the same directory from which you submitted the job. This file contains the output produced by your job during its execution on a dedicated compute node. You can either transfer these output files back to your laptop for viewing, or use a tool on the cluster to open them (such as `cat`, `less`, `emacs`, or `vim`).

2 A Quick Introduction to Useful Shell Commands

As we move more into using NOTS to measure real-world performance, being comfortable with the shell will become more and more useful in 322. This section briefly introduces you to the most commonly used shell commands. There are no required "tasks" or "goals" for this section beyond improving your awareness of these commands. However, you are encouraged to play with these commands in your own terminal (on your laptop or on NOTS) to help build familiarity. We have provided a sample set of exercises at the end of this section that you can use to help learn the commands.

Linux relies heavily on an abundance of command line tools. Most input lines entered at the shell prompt have three basic elements: command, options, and arguments. The command is the name of the program you are executing. It may be followed by one or more options (or switches) that modify what the command may do. Options usually start with one or two dashes, for example, `-p` or `--print`, in order to differentiate them from arguments, which represent what the command operates on. The interested student is encouraged to view Chapter 7 in the *LinuxFoundationX: LFS101x.2 Introduction to Linux* course on edX for a more thorough introduction on the commands covered in lab today.

In order to help you organize your files, your file system contains special files called directories. A directory is a logical section of a file system used to hold files. Directories may also contain other directories. Directories are separated by a forward slash (/). The current directory, regardless of which directory it is, is represented by a single dot (.). The parent directory of the current directory (i.e., the directory one level up from the current directory) is represented by two dots (..). Your home directory is the directory you are placed in, by default, when you open a new terminal session. It has a special representation: a tilde followed by a slash (˜/).

Below are some of the most used commands, along with brief descriptions of each.

2.1 pwd

`pwd` means "print working directory". It is used to output the path of the current working directory.

2.2 mkdir

`mkdir` means "make directory" and is used to create new directories. It creates the directory(ies), only if they do not already exist. Use the `-p` flag to ensure parent directories are created as needed.

2.3 cd

`cd` means "change directory". The `cd` command is used to change the current working directory. It can be used to change into a subdirectory, move back into the parent directory, move all the way back to the root directory, or move to any given directory. When the first character of a directory name is a slash, that denotes that the directory path begins in the root directory.

2.4 ls

`ls` means "list directory". The `ls` command lists out the contents of the directory you are currently in. You can use `cd` to change into different directories and then list what's in them so I know which directory to go to next.

2.5 dirs, pushd and popd

`pushd` means "push directory". `popd` means "pop directory". Both commands are used to work with the command line directory stack. The `pushd` command saves the current working directory in memory so it can be returned to at any time, optionally changing to a new directory. The `popd` command returns to the path at the top of the directory stack. This directory stack can be viewed by the command `dirs`.

2.6 rm

`rm` means "remove". Directories and files can be removed (deleted) with the `rm` command. By default, it does not remove directories. If the `-r` (`--recursive`) option is specified, however, rm will remove any matching directories and their contents. Use the `-f` (`--force`) option to never be prompted while files are being removed. The `-v` (`--verbose`) option can be used to get rm to detail successful removal actions.

2.7 cp

`cp` means "copy a file or directory". The command has three principal modes of operation, expressed by the types of arguments presented to the program for copying a file to another file, one or more files to a directory, or for copying entire directories to another directory. The commands takes two arguments, source and destination files, which may reside in different directories. You can use `cp` to copy entire directory structures from one place to another using the `-R` option to perform a recursive copy. Using this option copies all files, and all subdirectories from the source to the destination directory. you can specify multiple files as the source, and a directory name as the destination.

2.8 mv

`mv` means “move a file or directory”. It moves one or more files or directories from one place to another, it is also used to rename files. When a filename is moved to an existing filename (in the same directory), the existing file is deleted. Use the `-f` (`--force`) option to never be prompted before overwriting existing files. The `-v` (`--verbose`) option can be used to get details on the actions and destination locations of the files being moved.

2.9 touch

The `touch` command is used to make an empty file. The `touch` command is the easiest way to create new, empty files. Touch eliminates the unnecessary steps of opening the file, saving the file, and closing the file again. It is also used to change the timestamps on existing files and directories.

2.10 less

It is used to view (but not change) the contents of a text file one screen at a time. Unlike most Unix text editors/viewers, `less` does not need to read the entire file before starting, resulting in faster load times with large files. You can open a file by passing the file name as an argument to the command. To traverse the file press the following, use the down arrow to scroll down one line while using the up arrow scrolls up one line. Using q will exit the less command.

2.11 cat

The `cat` command prints the contents of a file to screen and can be used to concatenate and list files. The name is an abbreviation of catenate, a synonym of concatenate. `cat` will concatenate (put together) the input files in the order given, and if no other commands are given, will print them on the screen as standard output. It can also be used to print the files into a new file as follows: `cat old1.txt old2.txt > newfile.txt` Typing the command cat followed by the output redirection operator and a file name on the same line, pressing ENTER to move to the next line, then typing some text and finally pressing ENTER again causes the text to be written to that file. The program is terminated and the normal command prompt is restored by pressing the CONTROL and d keys simultaneously.

2.12 find

The `find` command is a very useful and handy command to search for files from the command line. `find` will search any set of directories you specify for files that match the supplied search criteria. You can search for files by name, owner, group, type, permissions, date, and other criteria. The search is recursive in that it will search all subdirectories too. All arguments to find are optional, and there are defaults for all parts.

2.13 grep

The `grep` command is used to search text or searches the given file for lines containing a match to the given strings or words. Its name comes from the `ed` command `g/re/p` (globally search a regular expression and print), which has the same effect: doing a global search with the regular expression and printing all matching lines. By default, `grep` displays the matching lines. You can force `grep` to ignore word case with the `-i` option.

2.14 man

The man command is used to format and display the system’s reference manuals. The man pages are a user manual; they provide extensive documentation about commands. Each argument given to man is normally the name of a program, utility or function. `man` is most commonly used without any options and with only one keyword. The keyword is the exact name of the command or other item for which information is desired, e.g. `man ls`.

2.15 Example Shell Exercises

1. Print the directory on NOTS where your lab_5 project has been checked out.
2. Change the directory to `src/main/resources/commands`.
3. Print the contents of the current directory, you should be able to see three .txt files.
4. Use the `less` command to view the contents of the text files.
5. Create a directory called `work` in the current directory.
6. Use the `pushd` command and change to the `work` directory.
7. Use the `pwd` command to confirm you are inside the `work` directory.
8. Print the contents of the current directory, you should see no files.
9. Create an empty file called `loremipsum.txt` using the `touch` command.
10. Use the `cat` command to copy the text from `lorem.txt` and `lorem.txt` files in the `src/main/resources/commands` directory into the `loremipsum.txt` file.
11. Use the `mv` command to move the `loremipsum.txt` file into the `src/main/resources/commands` directory.
12. Print the contents of the current directory (`work`), you should see no files if the move has been successful.
13. Now use the `cp` command to copy all the `*.txt` files from the `commands` directory into the `work` directory.
14. Use the `popd` command to switch to the `commands` directory.
15. Delete all the files whose names start with `lor*` only in the current directory.
16. Use the `pushd` command and change to the `work` directory.
17. Use the `grep` command to find the `*.txt` files which contain the text `ma`.
18. Use the `find` command to find the files whose name contain the text `sum`.
19. Use the `find` command to find the files whose name contain the text `lor` and move them to the `src/main/resources/commands` directory.

3 Image Kernels

Now, getting back to parallelism. Back on your laptop, the code provided in `Lab5.java` offers a sequential implementation of a convolution kernel which applies some transformation kernel to an image. Your task for this lab is to make this sequential implementation parallel and measure its real world speedup on the NOTS cluster.

You should start by gaining an understanding of the application you will be parallelizing today. This website provides an excellent introduction to convolutional image kernels.

Next, try running the provided tests using the provided sequential code. These tests read the image at `src/main/resources/kobe.jpg`, applies two different transformations to it, and writes the output of those transformations to `src/main/resources/kobe.1.jpg` and `src/main/resources/kobe.2.jpg`. The first transformation highlights edges in the image. The second transformation brightens the image. After running the provided tests you should be able to open `kobe.jpg`, `kobe.1.jpg`, and `kobe.2.jpg` next to each other and clearly see the results of the applied transformation.

1. You are now ready for the first part of the lab assignment, which is to *efficiently* parallelize the kernel in `Lab5.convolve`, by using the `forall` or `forallChunked` constructs in HJlib. (You are welcome to implement chunking by hand instead of using `forallChunked`.) It may help you to know that the bounds for the four loops are determined by the following values in the test case: inputWidth = 3280, inputHeight = 4928, and kernelWidth = kernelHeight = 3.

   Once you have efficiently parallelized this kernel on your laptop so that it passes the performance test, you should submit it to NOTS manually using the same process we went though above (i.e. by transferring your changed Lab 5 to the cluster using scp/sftp/SVN and then submitting using `sbatch`).

2. The second part of the lab is to parallelize the kernel using Java streams instead of HJlib's `forall` constructs. Recall that `IntStream.range(lo, hi)` is a stream *source* that generates a stream of integers in the range $lo \ldots hi$. Parallelism can be enabled by adding `.parallel()` as an *intermediate* operation, as follows: `IntStream.range(lo, hi).parallel()`. Finally, `forEach()` can be used as a *terminal* operation that specifies a lambda to be executed (in parallel) for each element of the stream.

4 Demonstrating and submitting in your lab work

For this lab, you will need to demonstrate and submit your work before leaving, as follows.

1. Show your work to an instructor or TA to get credit for this lab. They will want to see your files submitted to Subversion in your web browser and the passing unit tests on NOTS.

2. Check that all the work for today's lab is in your `lab_5` directory by opening `https://svn.rice.edu/r/comp322/turnin/S17/NETID/lab_5/` in your web browser and checking that your changes have appeared.