Multi-Threading

2016-17, CSCI 3150 - Assignment 3

Release: 3 Nov 2016

Deadline: 17 Nov 2016 11:59AM

Contents

1 Introduction

In this assignment, you have to solve a problem using multi-threading. You have to write two versions, one using PThread (`asg3-pthread`), one using OpenMP (`asg3-openmp`).

The program shall take two files as input. Each input file contains a list of integers. The job of the program is to output a sorted list of unique integers that are common between the two files.

1.1 Usage

To run the Pthread program:

```
./asg3-pthread [inputfile1] [inputfile2] [outputfile] [ThreadNum]
```

To run the OpenMP program:

```
./asg3-openmp [inputfile1] [inputfile2] [outputfile] [ThreadNum]
```

1.2 File Format

- `inputfile1` : The input file contains only 2 lines. The first line shows the number of integers in this file. The second line shows the integers, separated by one space character.
- `inputfile2` : Same as the above.
- `outputfile` : The output file should list the result in ascending order, line by line.

1.3 Example

`testcases/C0/input1.txt`	`10`
	`2 1 0 8 8 6 1 5 7 0`

`testcases/C0/input2.txt`	`15`
	`4 3 2 9 0 2 6 5 2 2 3 1 1 1 1`

If we execute the following:

```
./asg3-openmp testcases/C0/input1.txt testcases/C0/input2.txt res/output.txt 4
```

```
./asg3-pthread testcases/C0/input1.txt testcases/C0/input2.txt res/output.txt 4
```

Then, we expect the program executes using 4 threads and generates the following content to output.txt:

```
0
1
2
5
6
```

That is, we expect the output contains a sorted list of unique integers (in ascending order) that appear in both `input1.txt` and `input2.txt`.

2 Grading

The grading of this assignment is divided into two parts: (A) correctness (48%) and (B) scale-out (52%).

2.1 Part (A) Correctness (48%)

Its aim is to check the correctness of your program under multi-threading. For this part,

- 12 test cases, C1, C2, ..., C12, in total.
- All test cases will be tested under 4 threads.
- Each test case will feed the same input data to test your `asg3-pthread` (2 marks) and your `asg3-openmp` (2 marks).

Different test cases would have different input data, in different sizes and orders. For example, sometimes the data may contain duplicate but sometimes not. The largest integer in the test data would be $\leq 100,000,000$.

2.2 Part (B) Scale-out (52%)

A multi-threaded program is a **BAD** program if it does not scale-out — it runs slower with more threads! For example, if your program finishes the job in 60 seconds using 1 thread but finishes the job in 78 second using 2 threads, then your multi-threaded program is completely useless. This part therefore grades whether your program can scale-out. For this part,

- 3 test cases, S0, S1, S2, in total. They test your programs using very large input data.
- The first test case (test case S0) is to ensure that your program can run faster given more threads. That is, in addition to correctness, it also checks **the real time** (by the `time` tool) of your PThread and OpenMP programs to see if:

  real-time-4threads < real-time-3threads < real-time-2threads < real-time-1thread

  This test case takes **NO** marks. But passing this case is the prerequisite of executing the next two scale-out oriented test cases for you.
- Test cases S1 and S2 run your programs on large input using 4 threads and measures their real execution time in addition to their correctness. Each test case will feed the same input data to test your `asg3-pthread` ($X_{pthread}$ marks) and your `asg3-openmp` (X_{openmp} marks), where X depend on the **real execution time** of your program, its formula is:

  $$X = \frac{(N + 1 - R)}{N} * 13$$

  N is the number of students who can pass test case S1, and R is the speed-rank of your program among N students.[1] For example, if your Pthread program runs the second fastest among, say, 50 students who can pass test case S1, then your rank is #2. Your score for that case is:

  $$\frac{(50 + 1 - 2)}{50} * 13 = \frac{49}{50} * 13$$

[1]We originally want to give full marks to you as long as your program fulfills: real-time-4threads < real-time-3threads < real-time-2threads < real-time-1thread. Unfortunately, overly smart students can hack this by controlling the running time through sleep/loop. So, we have to introduce this grading scheme here. Forget about this footnote if you don't know what we are talking about.

2.3 Bonus (maximum 10%)

Bonus will be awarded to the following students.

- (10%) Top Coder Award (TCA): the one who gets the highest mark in Part A plus Plus B.
- (5%) Top Local Coder Award: the local one who gets the highest mark in Part A plus Plus B.
- (5%) Top Non-Local Coder Award: the non-local one who gets the highest mark in Part A plus Plus B.
- (5%) Top Female Coder Award: the lady who gets the highest mark in Part A plus Plus B.
- (5%) Top Male Coder Award: the gentleman who gets the highest mark in Part A plus Plus B.

The condition is that their programs must pass all test cases in Part (A) and Part (B). A student can get at most one award. Awardees will get an email proof from Prof. Eric Lo so that they can have one more line to put in their CVs.

3 Your assignment

You are given the following files:

Name	Description
`/asg3-pthread.c` `/asg3-openmp.c`	Code skeleton for you. (**Work on it**).
`/asg3-pthread` `/asg3-openmp`	Executable but not functioning. (**Runnable on your 32-bit virtual machine**)
`/Makefile`	Makefile
`/demo-serial`	Executable, functioning, serves as the baseline to check the correctness as well as the performance. (**Runnable on your 32-bit virtual machine**)
`/testcases`	Test data of input files. All testcase use the input file under this directory
`/res`	Output directory which you should output your result to. (**Don't change to other directory**)
`/grader.sh`	We will run this script to grade your assignment (**Don't touch**).

Note: The above assumes the course's 32-bit (4 virtual CPU enabled) VM is used.

3.1 Submission

You are required to submit two source codes `asg3-pthread.c` and `asg3-openmp.c` files as one **zip** file to eLearning.

Warning: DON'T change the file names, otherwise you get 0 marks

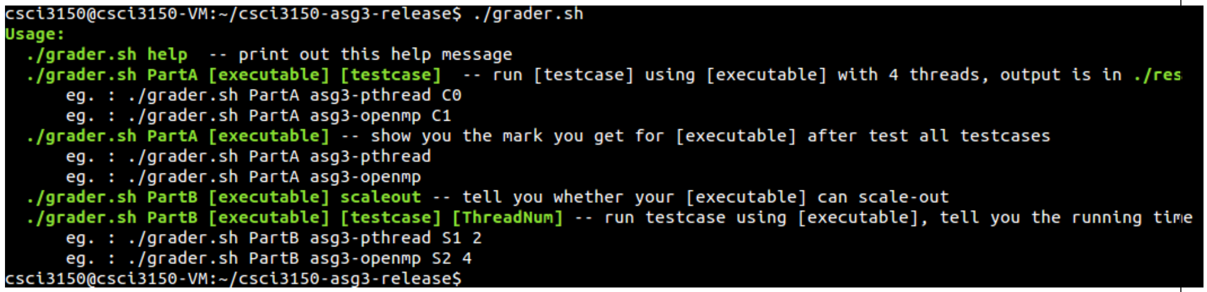
3.2 To begin

3.2.1 Usage of grader.sh

Run the following:

```
./grader.sh help
```

It will tell you the usage of `grader.sh`



```
csci3150@csci3150-VM:~/csci3150-asg3-release$ ./grader.sh
Usage:
  ./grader.sh help  -- print out this help message
  ./grader.sh PartA [executable] [testcase]  -- run [testcase] using [executable] with 4 threads, output is in ./res
      eg. : ./grader.sh PartA asg3-pthread C0
      eg. : ./grader.sh PartA asg3-openmp C1
  ./grader.sh PartA [executable] -- show you the mark you get for [executable] after test all testcases
      eg. : ./grader.sh PartA asg3-pthread
      eg. : ./grader.sh PartA asg3-openmp
  ./grader.sh PartB [executable] scaleout -- tell you whether your [executable] can scale-out
  ./grader.sh PartB [executable] [testcase] [ThreadNum] -- run testcase using [executable], tell you the running time
      eg. : ./grader.sh PartB asg3-pthread S1 2
      eg. : ./grader.sh PartB asg3-openmp S2 4
csci3150@csci3150-VM:~/csci3150-asg3-release$
```

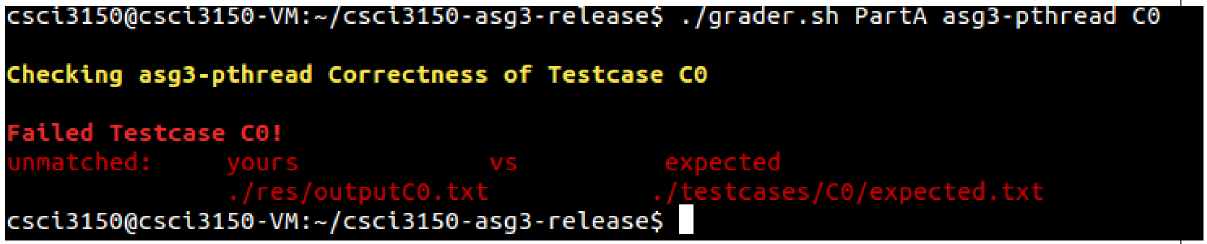
3.2.2 Running a Part A Test Case

The following example tests your programs using test case C0, whose input data are in `testcases/C0`.

```
./grader.sh PartA asg3-pthread C0
```

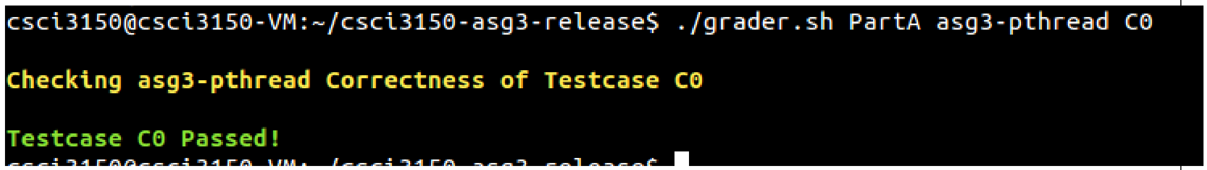
```
./grader.sh PartA asg3-openmp C0
```

Since the given `asg3-pthread` and `asg3-openmp` are not functioning initially, you shall see something like this:



```
csci3150@csci3150-VM:~/csci3150-asg3-release$ ./grader.sh PartA asg3-pthread C0

Checking asg3-pthread Correctness of Testcase C0

Failed Testcase C0!
unmatched:     yours               vs           expected
               ./res/outputC0.txt             ./testcases/C0/expected.txt
csci3150@csci3150-VM:~/csci3150-asg3-release$ 
```

After doing the assignment, you shall see something like this:



```
csci3150@csci3150-VM:~/csci3150-asg3-release$ ./grader.sh PartA asg3-pthread C0

Checking asg3-pthread Correctness of Testcase C0

Testcase C0 Passed!
```

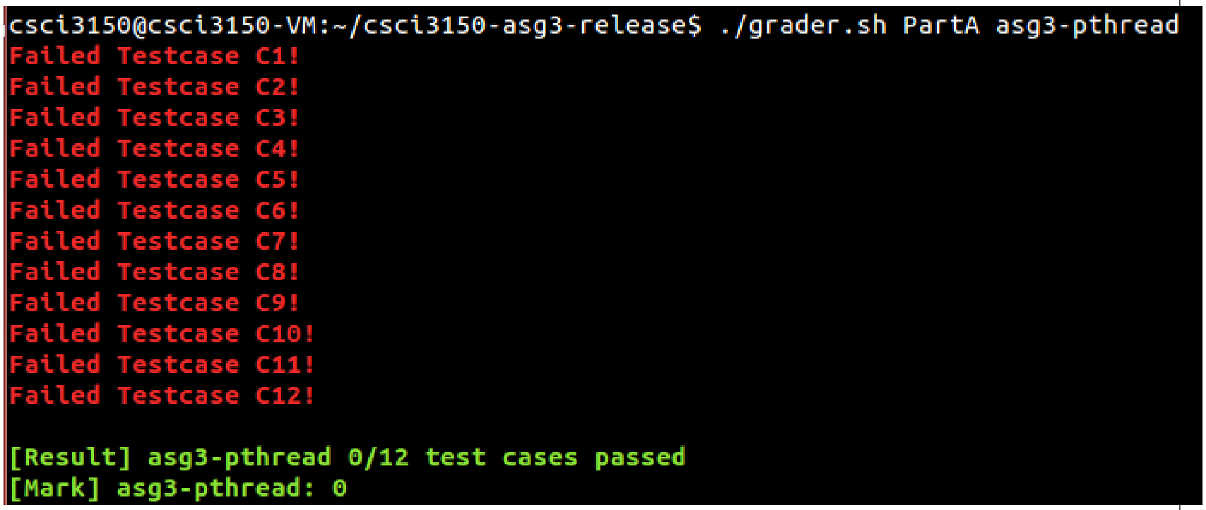
3.2.3 Running all Part A Test Cases

The following command executes all Part A test cases (C1 to C12) to test your program:

```
./grader.sh PartA asg3-pthread
```

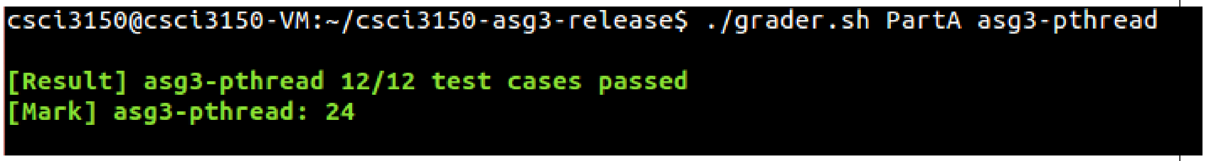
```
./grader.sh PartA asg3-openmp
```

Since the given `asg3-pthread` and `asg3-openmp` are not functioning initially, you shall see something like this:

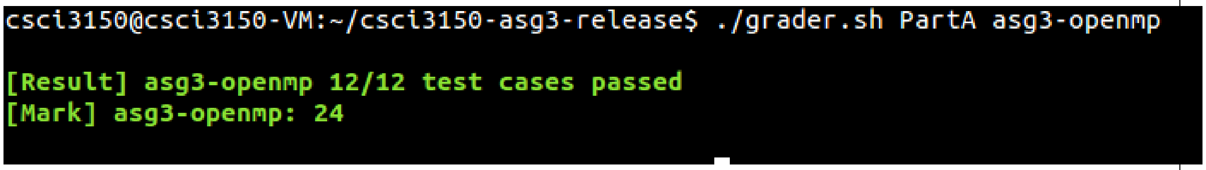


```
csci3150@csci3150-VM:~/csci3150-asg3-release$ ./grader.sh PartA asg3-pthread
Failed Testcase C1!
Failed Testcase C2!
Failed Testcase C3!
Failed Testcase C4!
Failed Testcase C5!
Failed Testcase C6!
Failed Testcase C7!
Failed Testcase C8!
Failed Testcase C9!
Failed Testcase C10!
Failed Testcase C11!
Failed Testcase C12!

[Result] asg3-pthread 0/12 test cases passed
[Mark] asg3-pthread: 0
```

After doing the assignment, you shall see something like this:



```
csci3150@csci3150-VM:~/csci3150-asg3-release$ ./grader.sh PartA asg3-pthread

[Result] asg3-pthread 12/12 test cases passed
[Mark] asg3-pthread: 24
```



```
csci3150@csci3150-VM:~/csci3150-asg3-release$ ./grader.sh PartA asg3-openmp

[Result] asg3-openmp 12/12 test cases passed
[Mark] asg3-openmp: 24
```

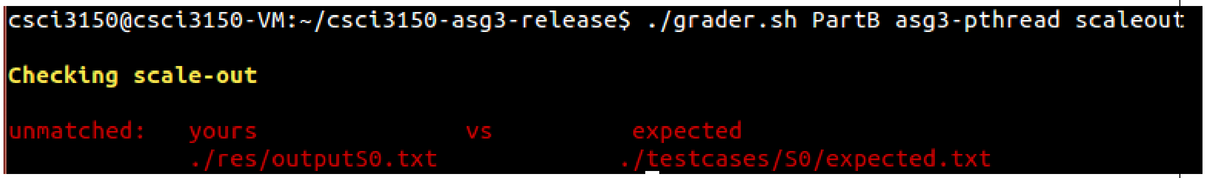
3.2.4 Checking whether your program can scale out

The following checks whether your program can scale out by comparing the running times of using 1, 2, 3, and 4 threads.

```
./grader.sh PartB asg3-pthread scaleout
```

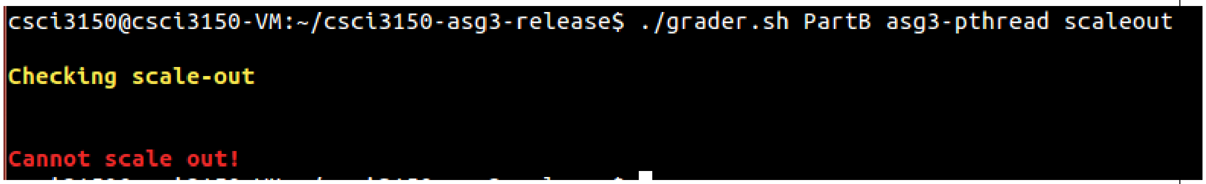
```
./grader.sh PartB asg3-open scaleout
```

Before you do the assignment, you shall see something like this:



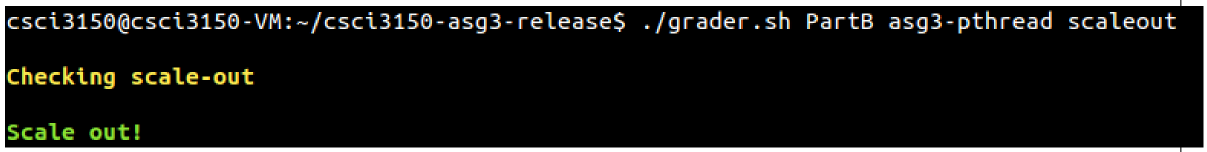
```
csci3150@csci3150-VM:~/csci3150-asg3-release$ ./grader.sh PartB asg3-pthread scaleout

Checking scale-out

unmatched:   yours                vs           expected
             ./res/outputS0.txt              ./testcases/S0/expected.txt
```

After doing the assignment but if your program cannot scale-out, you shall see something like this:



```
csci3150@csci3150-VM:~/csci3150-asg3-release$ ./grader.sh PartB asg3-pthread scaleout

Checking scale-out


Cannot scale out!
```

After doing the assignment and if your program can scale-out, you shall see something like this:



```
csci3150@csci3150-VM:~/csci3150-asg3-release$ ./grader.sh PartB asg3-pthread scaleout

Checking scale-out

Scale out!
```

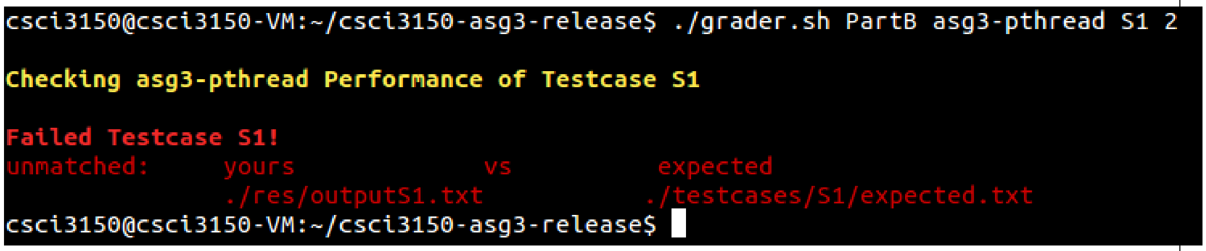
3.3 Running your program with test cases S1 and S2

The following example command executes test cases S1 using 2 threads:

```
./grader.sh PartB asg3-pthread S1 2
```

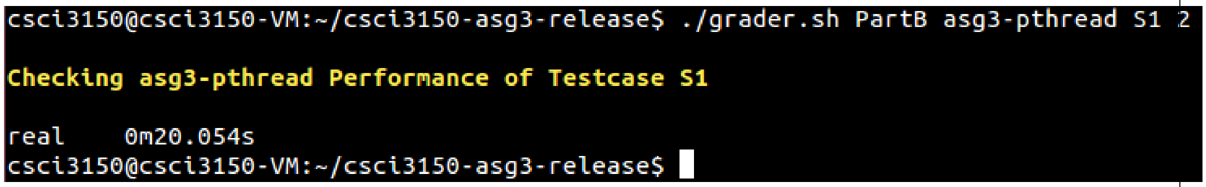
```
./grader.sh PartB asg3-openmp S1 2
```

Before you do the assignment, you shall see something like this:



```
csci3150@csci3150-VM:~/csci3150-asg3-release$ ./grader.sh PartB asg3-pthread S1 2

Checking asg3-pthread Performance of Testcase S1

Failed Testcase S1!
unmatched:     yours                vs          expected
               ./res/outputS1.txt             ./testcases/S1/expected.txt
csci3150@csci3150-VM:~/csci3150-asg3-release$
```

After you do the assignment, you shall see something like this:



```
csci3150@csci3150-VM:~/csci3150-asg3-release$ ./grader.sh PartB asg3-pthread S1 2

Checking asg3-pthread Performance of Testcase S1

real    0m20.054s
csci3150@csci3150-VM:~/csci3150-asg3-release$
```

3.4 Your job

Your job is to start from the given `asg3-pthread.c` and `asg3-openmp.c` and pass as many test cases as possible (Part A). In addition, your program shall finish execution as soon as possible (Part B).

3.5 Notes and Tips

- **Hardcoding won't work. We will use another similar set of test data and expected output when grading**. The good news is that, if you pass all the test cases during the assignment, you should also pass all the test cases during grading (unless you do hardcoding).

- The grading will be fully automated. The grading platform is our course's VM: Ubuntu 14.04 32-bit, 4 cores, 1GB memory. Once we start the grading processing, we reserve the right to deduct marks from you for any request that requires TA's extra manual effort.
- Use the synchronization primitives (e.g., condition variable) judiciously in order to get a good balance between correctness and performance. For example, deliberately define the critical section to be the whole program can make your program correct (Part A) but cannot scale-out (Part B).[2]

3.6 Late Submission Policy

We follow the late submission policy specified in the course outline.

4 Questions

If you have doubts about the assignment, you are encouraged to ask questions on our Facebook group.

5 Academic Honesty

We follow the University guide on academic honesty against any plagiarism.

[2]This is the key reason that we have to set the weighting of Part B be higher than that of Part A. If Part A takes, say, 80% of scores, then an overly smart student would simply write a single-threaded (non PThread/OpenMP) program to easily pass all the correctness test cases. Ignore this footnote if you don't know what we are talking about.