DUE DATE: **By the end of your lab session**

sCSE-443/543: High Performance Computing

Exercise #15

Max Points: 20

You should save/rename this document using the naming convention `MUid_Exercise15.docx` (example: `raodm_Exercise15.docx`).

Objective: The objective of this exercise is to:

- Use OpenMP sections to parallelize a task-parallel application
- Use OpenMP critical sections to avoid race conditions.

Submission: Save this MS-Word document using the naming convention *`MUid`*`_Exercise15.docx` prior to proceeding with this exercise. Upload the following at the end of the lab exercise:

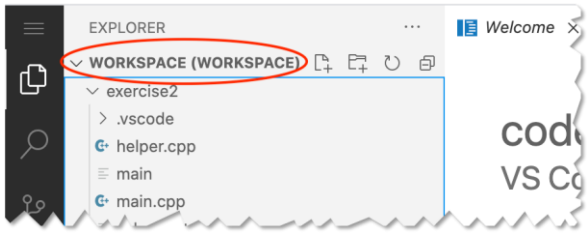
1. This document saved as a PDF file named with the convention *`MUid`*`_Exercise15.pdf`.
2. Program completed in part #3 of this exercise and named with the convention `Exercise15.cpp`.

You may discuss the questions with your instructor.

Part #0: Setting up VS-Code project

Estimated time to complete: 5 minutes

1. Log into OSC's OnDemand portal via https://ondemand.osc.edu/. Login with your OSC id and password.
2. Startup a VS-Code server and connect to VS-Code. Ensure you switch to your workspace. Your VS-Code window should appear as shown in the adjacent screenshot.



3. Next, create a new VS-Code project in the following manner:
   a. Start a new terminal in VS-Code
   b. In the VS-Code terminal use the following commands:

```
$ # First change to your workspace directory
$ cd ~/cse443
$ # Use ls to check if workspace.code-workspace file is in pwd
$ # Next copy the basic template for a C++ project
$ cp -r /fs/ess/PMIU0184/cse443/templates/openmp exercise15
$ # Copy the starter code for this exercise
$ cp /fs/ess/PMIU0184/cse443/exercises/exercise15/* exercise15
```

Task #1: Create reference output

Estimated time to complete: 5 minutes

Exercise: The objective of this part of the exercise is to create a reference output file that will be used to detect potential errors when the program is multithreaded.

1. Create a reference output from the single-threaded starter code. You will use the output for functional testing in the next steps. You can just run it on the login node as it only takes ~4 seconds to run.

```
$ export OMP_NUM_THREADS=1
$ ./exercise15 > reference_output.txt
$ chmod a-w reference_output.txt
```

Part #2: Multithread a task parallel method to run using OpenMP [10 points]

Estimated time to complete: 15 minutes

Background: OpenMP provides a convenient mechanism to organize code into parallel sections, using the `#pragma omp parallel sections` directive. Each section is processed by a single OpenMP thread. Several sections are executed concurrently from multiple threads, depending on the number of threads available to OpenMP. If there are more threads than sections, then some of the threads. On the other hand, if there are more sections than threads, then sections are executed in batches and the order in which sections are executed is not guaranteed.

Exercise: The objective of this part of the exercise is to parallelize just the `compute()` method into 2 two parallel sections:

1. One section must call the `isPrime` method
2. The other section to call the `isAngular` method.

For this part of the exercise simply use only OpenMP sections. Do not use critical sections yet. Yes, the program will experience race conditions due to concurrent operations on `CountMap& tracker` and the objective is to refresh your memory on how race conditions manifest.

Observations: Once you have parallelized the `compute` method using 2 OpenMP sections, run the program and compare the outputs as show below (easiest to run it from the terminal tab in `VS-Code`):

```
$ export OMP_NUM_THREADS=2
$ ./exercise15 | diff reference_output.txt -
```

The above diff command compares output from your program against the reference output and will show lines that are different. Ideally, there should be no difference once you have parallelized a program. However, in this case you will observe differences due to aforementioned race condition in the current implementation. Record the differences observed (just copy-paste output from diff) from two different runs of your program (**try to see if you are able to observe two different types of differences**) into the space below.

Incorrect Output #1

```
634c634,635
< 4271:10
---
> 4271:1
> 4271:9
951c952,953
< 5981:10
---
> 5981:1
> 5981:9
1092c1094,1095
< 6761:10
---
> 6761:1
> 6761:9
1215c1218,1219
< 7451:10
---
> 7451:1
> 7451:9
1336c1340,1341
< 8081:10
---
> 8081:1
> 8081:9
1438c1443,1444
< 8641:10
---
> 8641:1
> 8641:9
1618c1624,1625
< 9631:10
---
> 9631:1
> 9631:9
1684c1691
< Number of entries: 1683
---
> Number of entries: 1690
```

Incorrect Output #2

```
634c634,635
< 4271:10
---
> 4271:1
> 4271:9
951c952,953
< 5981:10
---
> 5981:1
> 5981:9
1092c1094,1095
< 6761:10
---
> 6761:1
> 6761:9
1215c1218,1219
< 7451:10
---
> 7451:1
> 7451:9
1336c1340,1341
< 8081:10
---
> 8081:1
> 8081:9
1438c1443,1444
< 8641:10
---
> 8641:1
> 8641:9
1618c1624,1625
< 9631:10
---
> 9631:1
> 9631:9
1684c1691
< Number of entries: 1683
---
> Number of entries: 1690
```

Part #3: Resolve Race Conditions [10 points]

Estimated time to complete: 15 minutes

Background: Race conditions occur when multiple threads modify a shared value. The value can either be entries in a data structure or the data structure as a whole. In order to eliminate race condition, OpenMP programs must be coded to access shared variables in critical sections marked by the `#pragma omp critical` directive.

Exercise: Modify the code from previous and place all operations on the shared hash map tracker with a critical block in each parallel section. If you name the critical block, then ensure that you use the same name in both parallel sections.

Observations: Once you have resolved the race conditions, run the program and compare the outputs as show below (it is easiest to run it from the terminal tab in `VS-Code`):

```
$ export OMP_NUM_THREADS=2
$ ./exercise15 | diff reference_output.txt -
```

Unlike in the previous task, the outputs in this task should be an exact match and the `diff` command should not generate any differences. Run the program two to three times to ensure that no differences arise (indicating you have correctly resolved all race conditions in the `convert` method)

Part #4: Submit files to Canvas

Upload just the following files:

1. This MS-Word document saved as a PDF file and named with the convention *`MUid_Exercise15`*`.pdf`.
2. Program developed in Part #3 named with the convention `Exercise15.cpp`.

Part #5: Exam Question Practice

Estimated time to complete: 10 minutes

Background: OpenMP permits critical sections to be named. The named sections can be used consistently throughout the program to selectively arbitrate access to shared variables to increase concurrency. However, care must be taken to ensure multithreading-safe (MT-Safe) access.

Exercise: Modify the two critical sections in your solution to use two different section names and you will observe that the race conditions reappear. Now, think about answers to the following questions (so you can possibly supply it in the exam):

1. Why did the race condition reappear when two different names were used for the two different critical sections?

   It is because two separate threads are trying to work on the same variable

2. When would be useful to two use critical sections with different names?

   If they are running independent tasks on different data or copies of the same data