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

CSE-443/543: High Performance Computing

Exercise #13

Max Points: 20

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

Objective: The objective of this exercise is to:

- Explore issue of "false sharing" in caches & impacts on performance
- Understand the use of OpenMP reduction clause

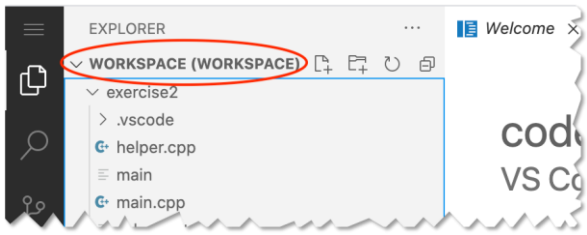
Fill in answers to all of the questions. For almost all the questions you can simply copy-paste appropriate text from the shell/output window into this document. You may discuss the questions with your TA.

Name: **Maciej Wozniak**

Task #1: 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 exercise13
$ # Copy the starter code for this exercise
$ cp /fs/ess/PMIU0184/cse443/exercises/exercise13/* exercise13
```

4. Add the newly created project to your VS-Code workspace to ease studying the source code.
5. Next, review the supplied `ex13_slurm.sh` script. This is a pretty standard job script, except for the setting of the number of OpenMP threads. Pay attention to that detail. In addition, ensure that the name of the executable in the script is consistent with the executable name for your project.

6. Compile the program using the **Release build** in VS-Code.

7. Run the supplied the program using the supplied `ex13_slurm.sh` script in the following manner:

```
$ sbatch --tasks-per-node=4 ex13_slurm.sh
```

8. While you are waiting for the job to complete review the data parallel version of `countBits` methods in the supplied `exercise13.cpp` source file, while paying attention to the following aspects:
   a. The critical section has been removed from the inner-most for-loop allowing multiple threads to effectively run in parallel.
   b. Consistent with a data parallel strategy, each thread counts number of '1' bits on a sub-set of numbers in `numList` into a per-thread entry in `bitcount[thrID]` entry.
   c. In order to create necessary entries in `bitcount` vector, a critical section and a barrier solution has been used.

9. Once the job is complete (takes < 1 minute of runtime once the job starts), view the output file to ensure the output from each of the 5 runs of the program is correct and consistent as:

```
Bit count: 632223552
Bit count: 632223552
Bit count: 632223552
Bit count: 632223552
Bit count: 632223552
```

10. Record the average statistics from the 5 runs of program below. You may use the Runtime Comparison Template for this.

Average elapsed time (sec):	52.86
Average %CPU used:	398.2%
Instructions per clock cycle	1

Task #2: Reducing "false sharing"

Estimated time to complete: 15 minutes

Background: False sharing (see: https://en.wikipedia.org/wiki/False_sharing) arises when adjacent (to each other in memory) but independent values being modified by different threads are cached in the same cache line. Consequently, each time a variable is updated, the cache has to be invalidated and refreshed to ensure caches are consistent on each core.

Exercise: The objective of this part of the exercise is to make minor modifications to the code to reduce "false sharing"

1. Observe that repeatedly modifying entries in a vector from multiple threads causes "false sharing" in the following line of code (within the `for`-loop):

```
bitcount[thrID] += popcount(numList[idx]);
```

2. Reduce the false sharing by restructuring the loop using a thread-local variable as shown below:

```
int sum = 0;  // per-thread variable to avoid "false sharing"
for (int idx = startIndex; (idx < endIndex); idx++) {
   sum += popcount(numList[idx]);
}
bitcount[thrID] = sum;  // update shared entry just once
```

3. Compile the program. Run the supplied the program using the supplied `ex13_slurm.sh` script in the following manner:

```
$ sbatch --tasks-per-node=4 ex13_slurm.sh
```

4. One the job completes, ensure the outputs are correct/consistent as shown earlier. Next, Record the average statistics from the 5 runs of program below. Use the runtime comparison template.

Average elapsed time (sec):	18.9463
T-test p-value:	0
Average %CPU used:	396.7%
Instructions per clock cycle	2.67

Important observations to note:

- Reducing false sharing will improve performance (*i.e.*, reduce elapsed time) of the program. The improvement in some cases a bit small depending on number of threads and optimization. With larger programs the improvements will be more prominent.
- Note that due to cache update issues, the CPU has to stall for cache updates. So, the instructions per clock cycle will degrade.

Task #3: Using OpenMP reduction

Estimated time to complete: 15 minutes

Background: Reduction is an operation in which data from multiple threads are combined together to produce a single value using different operations such as: sum (+), difference (-), product (*) etc. This operation is used in several different scenarios, including map-reduce operations.

Exercise: The objective of this part of the exercise is to further streamline the program using OpenMP reduction:

1. Using example from lecture slides suitably restructure the `countBits` method to use OpenMP reduction clause to compute the sum of bits from different threads. This is a relatively straightforward modification starting with getting rid of the `bitcount` vector.
2. Clean-build the program. Run the supplied the program using the supplied `ex12_slurm.sh` script in the following manner:

```
$ sbatch --tasks-per-node=4 ex13_slurm.sh
```

3. One the job completes, ensure the outputs are correct/consistent as shown earlier. Next, Record the average statistics the 5 runs of program below:

Average elapsed time (sec):	20.81418
Average %CPU used:	396%
Instructions per clock cycle	2.52

Submit files to Canvas

Upload the following files to Canvas:

1. Upload this MS-Word document (duly filled with the necessary information) saved as PDF using the naming convention **`MUid.pdf`**.
2. Upload your version of `exercise13.cpp` that you revised in this exercise.