CS 4380 / CS 5351
Parallel Programming / Parallel Processing
Fall 2016

Project 4

Due: *Thursday, November 3 at 3:30 PM*

All project files including the report have to be submitted using TRACS. Please follow the instructions at http://tracsfacts.its.txstate.edu/trainingvideos/submitassignment/submitassignment.htm. Note that files are only submitted if TRACS indicates a successful submission. See below for what files to submit. The project report has to list and discuss the results of your measurements and provide the answers to the questions. The answer to each question is limited to 50 words. This project has to be done *individually*. You have to be able to explain all of the code that you submit.

4.1 Fractal [25u/25g points]

Parallelize the *for-frame* loop of the fractal computation code from Project 1 using OpenMP. Follow these steps when implementing your code:

1. Base your code on the serial code from Project 1.
2. Include the math.h header file but do *not* include the omp.h file.
3. Make the code print "OpenMP" instead of "serial".
4. Add a new parameter at the end of the command line to specify the total number of threads and update the program usage message accordingly.
5. Make sure the number of threads is at least one.
6. Have the program print the requested number of threads.
7. Only parallelize the *for-frame* loop using a "parallel for" pragma.
8. Use "num_threads(…)" to specify how many threads to use.
9. Use "default(none)" and list all shared and private variables (excluding constants).
10. Do not specify a schedule.
11. Remove the loop-carried dependency by computing *delta* as a function of the frame number in each iteration. Note that this is somewhat different from the previous projects.
12. Verify, on small inputs, that the correct fractal is computed for different thread counts.

On the machines in the parallel lab, compile your OpenMP code as follows:

```
g++ -march=native -O3 -fopenmp fractal_omp.cpp -o fractal_omp
```

On Stampede, compile your OpenMP code as follows:

```
icc -xhost -O3 -openmp fractal_omp.cpp -o fractal_omp
```

Use the provided submission script at /home1/00976/burtsche/Parallel/fractal_omp.sub. Do not modify this script. Submit your code on TRACS. List the resulting CPU runtimes for the different problem sizes and thread counts in a table. Plot the speedups as a function of the thread count for all problem sizes in a single chart. Discuss and explain the results.

4.2 TSP [25u/25g points]

Parallelize the *for-i* loop that iterates over the cities in the TSP code using OpenMP. Follow these steps when writing the code:

1. Base your code on the serial code from Project 1.
2. Do *not* include the omp.h file.
3. Do not change the command-line parameters.
4. Make the code print "OpenMP" instead of "serial".
5. Only parallelize the *for-i* loop using a "parallel for" pragma.
6. Do not specify the number of threads.
7. Use "default(none)" and list all shared and private variables (excluding constants).
8. Specify a cyclic schedule.
9. Make the entire *if* statement inside the loop nest a critical section. Moreover, for performance reasons, make a second copy of the *if (...)* statement and only execute the critical section if the new *change* value is less than *minchange*. In other words, you run the same *if* check twice, once just outside and once inside the critical section.

On the machines in the parallel lab, compile your OpenMP code as follows:

```
g++ -march=native -O3 -fopenmp tsp_omp.cpp -o tsp_omp
```

On Stampede, compile your OpenMP code as follows:

```
icc -xhost -O3 -openmp tsp_omp.cpp -o tsp_cpu
```

Then compile the code again on Stampede as follows:

```
icc -mmic -O3 -openmp tsp_omp.cpp -o tsp_mic
```

Note the "`-mmic`" flag, which tells the compiler to generate code for the MIC (Xeon Phi) accelerator, which has 61 cores that support 4 SMT threads each, i.e., a total of 244 threads.

Use the provided submission script at /home1/00976/burtsche/Parallel/tsp_omp.sub, which runs both the CPU and the MIC executables. Do not modify this script. (The machines in the parallel lab do not have any MIC accelerators, so you cannot compile or test MIC code there.) Submit your code on TRACS. List the best resulting CPU and MIC runtime for each input in a table. Discuss and explain the results. How does the performance of the 1 MIC chip compare to the 2 CPU chips? Why is a named critical section not useful in this code? Why does duplicating the *if* statement improve performance even though it creates more code?

4.3 BST [50u/50g points]

Parallelize the *for* loop in the buildBST function of the binary search tree code from Project 1 using OpenMP. Follow these steps when implementing your code:

1. Base your code on the serial code from Project 1.
2. Include the omp.h file.
3. Make the code print "OpenMP" instead of "serial".

4. Add a new parameter at the end of the command line to specify the total number of threads and update the program usage message accordingly.
5. Make sure the number of threads is at least one.
6. Have the program print the requested number of threads.
7. Before the *for* loop, insert the statement "insert(root, hash(seed));". Then change the *for* loop to start at 1 instead of 0.
8. Parallelize this modified *for* loop using a "parallel for" pragma.
9. Use "num_threads(…)" to specify how many threads to use. For this purpose, pass the number of threads to the buildBST function.
10. Use "default(none)" and list all shared and private variables (excluding constants).
11. Do not specify a schedule.
12. Add an OpenMP lock field to the BSTnode structure.
13. Initialize the lock field wherever a new BSTnode is created.
14. When inserting a new node into the tree, only the node that will become the parent of the new node needs to be locked. To determine whether you have reached this parent node, check whether the first parameter of the recursive *insert* call is pointing to NULL. If it is, lock the current node. Once the lock has been acquired, check if the first parameter still points to NULL. If so, call *insert* and then release the lock. If not, release the lock first and then call *insert*.
15. Note that the verification code does not check for all possible errors.
16. As a partial remedy, make the verifyAndDeallocate function return the number of nodes in the (sub-)tree. If it is not equal to *n*, the main function should print an error message.
17. Verify that a reasonable solution is computed for different thread counts.

On the machines in the parallel lab, compile your OpenMP code as follows:

```
g++ -march=native -O3 -fopenmp bst_omp.cpp -o bst_omp
```

On Stampede, compile your OpenMP code also using g++ (to avoid a bug in icc):

```
g++ -march=native -O3 -fopenmp bst_omp.cpp -o bst_omp
```

Submit your code on TRACS. Run it using the submission script at /home1/00976/burtsche/Parallel/bst_omp.sub. Do not modify this script. List the runtime for each input and thread count in a table. Discuss and explain the results. Why is it necessary to run the first loop iteration (i = 0) before the parallelized *for* loop? Explain why the *hash* function is re-entrant. Why is it sufficient to only lock the parent node as opposed to all the nodes on the path from the root to the parent? Finally, explain why we have to recheck whether the first parameter points to NULL after we have acquired the lock.

Code Requirements

- Make sure your code is well commented.
- Make sure your code does not produce unwanted output such as debugging messages.
- Make sure your code's runtime does not exceed the specified maximum.
- Make sure your code is correctly indented and uses a consistent coding style.
- Make sure your code does not include unused variables, unreachable code, etc.

Code Submission

- Delete all files that you do not need anymore such as *.o and *.bmp files.
- Make sure your code complies with the above requirements before you submit it.
- Any special instructions or comments to the grader should be included in a "README" file.
- Upload all the files you need to submit onto TRACS. The report has to be in PDF. All other files, including source code, have to be plain text files (e.g., *.cpp files).
- Upload each file separately and do not compress them.
- Do not submit any unnecessary files (e.g., provided or generated files).

You can submit your file(s) as many times as you want before the deadline. Only the last submission will be graded. Be sure to submit at least once before the deadline.

October 27, 2016