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

Project 3

Due: *Thursday, October 20 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 for graduate students* and can be done *in pairs for undergraduates*. You have to be able to explain all of the code that you submit, including code written by your partner in case of a pair project.

3.1 Fractal [50u/50g points]

Parallelize the FOR loop that iterates over the frames in the fractal code using a blocked assignment of iterations to threads. Follow these steps when writing the pthread code:

1. Base your code on the serial code from Project 1.
2. Make the code print "pthread" instead of "serial".
3. Include the pthread and math header files.
4. Only the master thread is allowed to print normal program output.
5. 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.
6. Make sure the number of threads is at least one.
7. Have the master thread print the number of threads.
8. Make all global variables and functions (other than main) static.
9. Create *threads - 1* worker threads and have the master thread also compute part of the fractal, i.e., make the master thread call the same function that the new threads call.
10. Start the timer just before creating the worker threads and stop the timer just after joining the worker threads.
11. Only pass the rank to the function that runs in parallel and return NULL from it. All other information is passed to and from this function via global variables.
12. Convert the void pointer argument to an integer and do not use the argument otherwise.
13. Make sure your code runs correctly for different numbers of threads, i.e., it produces the same results as the serial code.
14. Compute which frames to process based on the code for a blocked assignment given in the Chapter 9 lecture slides.
15. Adjust the "delta" computation as needed.
16. Free all dynamically allocated memory before normal program termination.
17. Print the elapsed wall-clock time of the timed code as it is done in the serial code.

The timed code section should only include creating and joining the threads, the loop computing the frames, and the delta computation. In particular, writing the BMP files, if any, and dynamic memory allocation and freeing should not be timed.

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

```
g++ -march=native -O3 -pthread fractal_pthread.cpp -o fractal_pthread
```

On Stampede, compile your pthread code as follows:

```
icc -xhost -O3 -pthread fractal_pthread.cpp -o fractal_pthread
```

Run the code using the submission script at /home1/00976/burtsche/Parallel/fractal_pthread.sub. Do not modify this script. Present the runtime for each problem size and thread count in a table. Plot the parallel efficiencies against the thread count for the four problem sizes in a single figure. Discuss the results. Explain why 32 threads result in shorter runtimes than 16 threads even though there are only 16 cores and hyperthreading is disabled. Name the source file "fractal_pthread.cpp" and submit it on TRACS.

3.2 TSP [50u/50g points]

Parallelize the FOR-i loop that iterates over the cities in the TSP code using a cyclic assignment of iterations to threads. Follow these steps when writing the pthread code:

1. Base your code on the serial code from Project 1.
2. Make the code print "pthread" instead of "serial".
3. Include the pthread header file.
4. Only the master thread is allowed to print normal program output.
5. 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.
6. Make sure the number of threads is at least one.
7. Have the master thread print the number of threads.
8. Make all global variables and functions (other than main) static.
9. Remove all but the last argument of the TwoOpt function.
10. Start and stop the timer in the main function as it is done in the serial code.
11. Create *threads - 1* worker threads in the TwoOpt function and have the master thread also compute part of the work, i.e., make it call the same function that the new threads call.
12. Only pass the rank to the function that runs in parallel and return NULL from it. All other information is passed to and from this function via global variables.
13. Convert the void pointer argument to an integer and do not use the argument otherwise.
14. Make sure your code runs correctly for different numbers of threads, i.e., it produces the same results as the serial code.
15. Compute which frames to process based on the code for a cyclic assignment given in the Chapter 9 lecture slides.
16. Each thread should compute a local solution in the parallelized loop.
17. You will need a mutex to reduce the locally best solutions found by each thread into the global solution. This reduction must happen right after the parallelized FOR-i loop while

all the threads are still running. Of course, if a thread failed to find a move than improves the tour length, it should not even try to update the globally best move.

18. Allocate the thread handles and initialize the mutex at the beginning of the TwoOpt function. Free the handles and destroy the mutex at the end of this function.
19. Print the elapsed wall-clock time of the timed code as it is done in the serial code.

Run the code using the submission script at /home1/00976/burtsche/Parallel/tsp_pthread.sub. Do not modify this script. Present the best runtime for each thread count in a table. Discuss the results. Name the source file "tsp_pthread.cpp" and submit it on TRACS.

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).
- For group projects, include a comment at the beginning of the source code and the report that lists both group members. The two group members have to submit identical files on TRACS.
- 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 13, 2016