CSC/ECE 506: Architecture of Parallel Computers
Program 2: OpenMP programming
Due: Tuesday, February 20, 2024

This set of programming exercises will familiarize you with OpenMP and the shared-memory programming model. It demonstrates some issues that arise when using multiple threads to complete a programming task. You are going to explore how to address these issues, and measure and compare the performances from different strategies for approaching the problem.

Procedure

1. Log into login.hpc.ncsu.edu with your unity id/password via ssh in terminal (e.g., via PuTTY on Windows) `ssh unity_id@login.hpc.ncsu.edu`
2. Download the OpenMP example tar-ball by executing `wget https://www.csc2.ncsu.edu/faculty/efg/courses/506/s24/www/homework/p2/program2.zip`
3. Unzip the file.
4. `cd` into `program2/original`.
5. Check that you can compile by executing `make`.

Submission instructions (5 points)
Compress the program2 directory into a tgz file using the following command:

```
tar czf program2.tgz program2
```

and submit it to Expertiza. You may also use a zip file if you prefer.

Your .tgz file should contain—

1. The four folders and in each folder, a trap-omp.c and the Makefile.
2. A **`report.pdf`** file that includes all the explanations that you are asked to provide below.

You can transfer files from the hpc platform by issuing `scp` command from terminal or PuTTy.
Name your submission folder after your unity id. If you have a group, name it with unityA_unityB. Any failure in following the above instructions will lead to a deduction of 5 pts.

Experiments (95 points)

In this program, you are provided with a code that computes the trapezoidal approximation of

$$\int_0^x \sin(x)dx$$

using 2^{20} equal subdivisions. The correct answer should be 2.0. However, problems will occur when you run the program with multiple threads. You are asked to diagnose the cause of the problem and use OpenMP primitives to fix it and produce the correct output.

1. *(10 points)* Run the following commands, describe what you observe and explain why it happens:

```
./trap-omp
./trap-omp 2
./trap-omp 4
./trap-omp 8
./trap-omp 16
```

2. *(20 points)* Add the following preprocessor pragma directive, just before the **for** loop, which starts at line 33. Run the commands in part 1 again, describe what you observe and explain why it happens:
```
# pragma omp parallel for num_threads(threadct)
   Shared (a, n, h, integral) private(i)
```
3. *(30 points)* Try to fix `trap-omp.c` using three kinds of OpenMP primitives, viz., `reduction`, `atomic` and `critical`. Here is a reference you can turn to: https://computing.llnl.gov/tutorials/openMP/ Put your fixed `trap-omp.c` into its corresponding folder.

4. *(35 points)* For each strategy, you are asked to measure and compare the performance. The performance can be measured in the following way. Put the `omp_get_wtime` calls in the indicated places:

```
// Starting the time measurement
double start = omp_get_wtime();

// Computations to be measured
…
// Measuring the elapsed time double
end = omp_get_wtime();

// Time calculation (in seconds)
double elapsed_time = end – start;

// Print out the elapsed_time
```

Put the elapsed time into a table after running with various numbers of threads. Which one is the fastest? Explain why it produces the best performance.

Number of threads	Reduction	Atomic	Critical
2			
4			
8			
16			