In his exalted name

Parallel Programming

Fall 2015

Assignment No. 2

Total Points: 100

Due Date: Friday, December 4, 2015

Problem Statement

In this assignment you are supposed to develop a C/C++ program that implements a parallel version of a hybrid algorithm. A hybrid algorithm is an algorithm that combines two or more other algorithms that solve the same problem, either choosing one (depending on the data), or switching between them over the course of the algorithm. This is generally done to combine desired features of each, so that the overall algorithm is better than the individual components.

Timsort is a hybrid stable sorting algorithm, derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data. It was invented by Tim Peters in 2002 for use in the Python programming language.

The insertion sort, which is inefficient on large data, but very efficient on small data (say, five to ten elements), is used as the final step, after applying another algorithm, which is in this case, merge sort. Merge sort is asymptotically optimal on large data, but the overhead becomes significant if applying it to small data, hence the use of a different algorithm at the end of the recursion.

You are supposed to implement a *parallelized pseudo-timsort* algorithm using *OpenMP*, by combining the two mentioned sorting algorithms provided in the source code along this document. Tune your algorithm and report your speedups.

Deliverables

Executable files should get two command-line parameters as their input arguments. Size of the vector and number of preferable threads. Note that if you have embedded a dynamic thread generation mechanism in your program, then supplying the number of threads parameter won't be compulsory.

E.g. The command below means that the vector size is 100 (filled with random numbers in [0, 100)) and the preferable number of threads is 4.

```
$ ./pseudo_timsort 100 4
```

Report files have a brief description of parallelization methods you used, the results and your analysis of them. The result section has time and speed-up tables for input sizes 100KB, 1MB, 10MB, 100MB, and 1GB and different thread numbers.

Ex. Speedup table (numbers are fake!)

Average time of 5 runs

Input size	**1MB**	**10MB**	**100MB**	**1GB**
#Threads				
2	**1.81**	**1.85**	**1.93**	**1.94**
4	**3.70**	**3.68**	**3.67**	**3.71**
8	**6.21**	**6.92**	**7.21**	**7.76**

If you are unable to run your program with the requested inputs, due to your machine limitations, please mention it in your report and try the program with input sizes possible for your case.

Submission

Upload your source code, executable file(s) along with your report PDF in an archive file to our course webpage, named in the following format:

[Parallel Programming] [HW2] <First Name> <Last Name> - <Student ID>

Ex. [Parallel Programming] [HW2] Ahmad Siavashi – 94131100

Bonus

Any innovative work will be considered for extra points.

The deadline is next Friday, 11:55 PM. There is a delay penalty of -5% per day.

Good Luck ☺