High Performance Systems Programming (ECE 415)
Department of Electrical & Computer Engineering University of Thessaly

Teacher: Christos D. Antonopoulos

2nd Laboratory Exercise – Deadline 1/11/2022, 23:59

Objective: Parallelize a complete sequential application using OpenMP. Experimental evaluation on a multi-core processor.

Introduction:
As we discussed in class, a key advantage of the OpenMP programming model is that it allows easy, incremental parallelization of sequential applications. In this specific lab you are given the sequential code of an implementation of the k-means clustering algorithm. The algorithm groups n observations into k clusters. Each cluster is characterized by the coordinates of its "center" (centroid). Each of the n observations is assigned to the cluster from whose centroid it has the smallest distance. You are also given multiple different input files for different problem sizes (from simple to more realistic).

The application is amenable to parallelization. You are asked to parallelize the application with OpenMP and evaluate its performance on a multi-core system with an Intel Xeon E5-2695 processor. All development will be done on inf-mars1 (10.64.82.31) and inf-mars2 (10.64.82.32) systems while final measurements on csl-artemis (10.64.82.65). In any case, the final executable should be produced with the Intel C compiler. Use in all experiments the compiler optimization options that give the best (runtime) results for the original sequential code. Keep a copy of the original sequence code as a reference (for correctness and performance comparisons).

For the way the application runs, see the related README.

You can change the number of threads used by the OpenMP executable by setting the OMP_NUM_THREADS environment variable appropriately. E.g.:

```
export OMP_NUM_THREADS=4
```

Whenever you measure performance make sure that no one else is running at the same time.

Wanted:
The first logical step is to perform profiling and identify the points (loops) that are worth focusing your attention on. Work your way up, starting with loops that take a lot of time to run and work your way up to those with less time to run.

For each loop you find, check if it is parallelizable. If it is, add the appropriate OpenMP directives. Pay particular attention to any variables that need to be declared private. Experiment with the appropriate scheduling policy and chunk for each loop. After verifying that parallel code gives correct results (compared to sequential), consider whether parallelization is beneficial (from a performance perspective) or if any

leads to longer execution time. Try each time to run with 1, 4, 8, 16, 32, and 64 threads on csl-artemis (1, 2, 4, 8 threads on inf-mars1 and inf-mars2). Only after you have finished parallelizing and evaluating the performance of one loop is it appropriate to move on to the next.

After you've parallelized all the loops whose parallel execution leads to better performance, deal with any "overall optimizations" that can be made (for example – but not limited to – the width of the parallel regions, any implicit barriers that could be removed, etc.). Also consider any changes to the data structures and initial implementation that could offer more parallelism, better performance, etc.

As part of the experimental evaluation, provide the execution time of the original sequential program, as well as the execution time of the OpenMP program with 1, 4, 8, 16, 32, 64 threads. Repeat each experiment multiple times and report the mean and standard deviation of the readings. Try to comment on the results. Note that: a) The execution time of the 1-threaded OpenMP program may differ from that of the sequential program due to the overhead of parallelization. b) In DE times you should include I/O (input / output from / to files). c) The inf-mars processor (i7-4820K) is a 4-way multicore. Therefore, when you run from 1 to 4 threads, the operating system takes care to assign each one to a different core. csl-artemis, respectively, has 2 processors, each 16-way multicore. d) In both machines each core is 2-way SMT (or hyperthreaded according to Intel terminology). Therefore, when you run the application with more threads than cores, some of them share the same core.

Delivery:

You must deliver:

- The final code. •
- Report in which you will analyze the parallelization strategy you followed and the points you parallelized. Be sure to also mention any optimizations / parallelizations you've tried without good runtime results. Include the experimental evaluation in the report. Also, mention the compiler flags you used.

Create a .tar.gz file with the above contents and name <name1>_<AEM1>_<name2>_<AEM2>_lab2.tar.gz. Upload it on time to e-class.