CS406 Parallel Computing

Homework 1 - Parallelization of Breadth-First Search

ORAN CAN ÖREN

25 October 2017

1 Task Description

Our aim is to increase the performance of breadth-first search, a widely used graph traversal algorithm. We are provided a variety of graphs in compressed sparse row (CSR) format. The objective is to efficiently compute the breadth-first distance of each vertex in the graph to a given source vertex by parallel processing, using OpenMP.

2 Implementation Details and Results

I used top-down bottom-up hybrid BFS method introduced by [1]. The algorithm operates by iterations. In each iteration except the first one, the algorithm decides whether it is better to perform a top-down step or a bottom-up step. This is done by simply keeping and updating the number of edges to check from frontier and unvisited vertices as well as the number of vertices in the frontier; hence the decision procedure does not add additional complexity.

I have proceeded with the speed-up process experimentally in multiple versions where the details of which are discussed in the following subsections.

First, I attempted to parallelize the for-loops that were traversing over vertices of the graph and unvisited vertices. It turns out that this approach causes slowdowns. Performance profiling has shown that this was caused by the large amount of critical reigons inside these for-loops.

As a second attempt, instead of parallelizing the for-loops I created tasks to inspect the neighbors of vertices that are being traversed. This turned out to result in minor speedups instead of slowdowns as in the case for the first attempt.

2.1 Top-down bottom-up hybrid edge checking

Both parallelization techniques discussed in this report use the top-down bottom-up approach. The objective of this approach is to reduce the number of false edge checks (i.e. the checks on the edges that will not be included in the BFS-tree). To do so, the traditional top-down steps are replaced by bottom-up steps where it is forseen that a bottom-up step will perform fewer edge checks than a top-down step.

2.2 Sharing vertices among threads

The top-down algorithm traverses over vertices in the frontier and performs edge checks for each of these vertices. In order to share this work among threads, I parallelized this for loop. However, this parallel region includes a critical region. In this critical region, the visited vertex is added to the frontier vertex and the variables needed to compute the threshold to switch to the bottom-up step are updated.

Graphs included in this task are sparse graphs, hence the average vertex degree is very small. Therefore the overhead for such parallelism is greater than the benefits gained.

2.3 Tasks inspecting single vertices

Instead of parallelizing the for-loop that distributes vertex traversal work; I created tasks for inspecting the edges of vertices. More precisely, for the top-down step, a task is created for inspecting the edges of each vertex in the frontier. For the bottom-up step, a task is created for inspecting the edges of each non-visited vertex.

In other words, instead of assigning each thread a portion of vertices for which their neighbors are inspected; this approach creates tasks that perform traversal of neighbors of single vertices. Therefore the task granularity is finer.

2.4 Further improvement by restructuring

The performance could be improved by reordering the graph (i.e. computing an isomorphism of the vertex set) since it would increase the spatial locality of data. Algorithms such as Reverse Cuthill-Mckee and Rabbit Order should suffice to obtain a good ordering of the graph. However, due to time constraints I could not perform this additional preprocessing step.

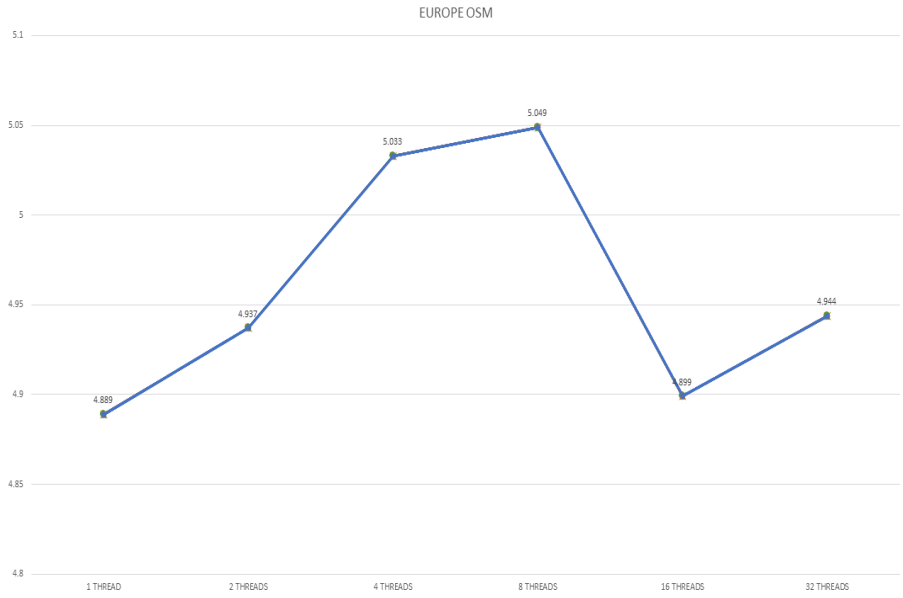
3 Performance Comparison Charts

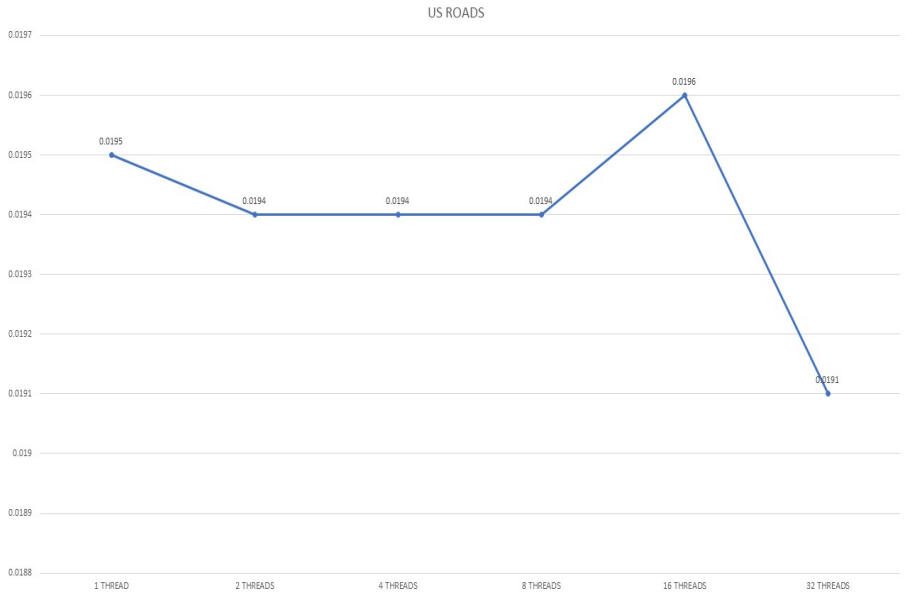
3.1 Execution time tables

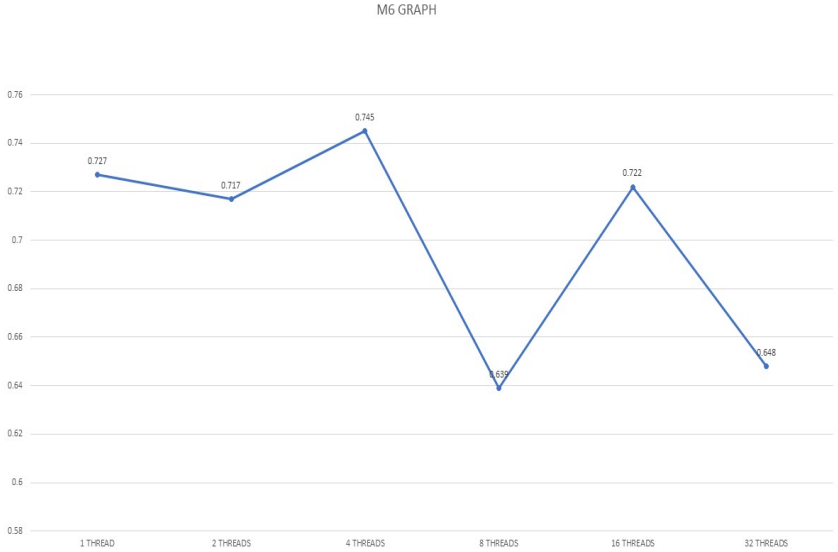
Table 1: Execution times (seconds) vs active thread count

	1 thread	**2 threads**	**4 threads**	**8 threads**	**16 threads**	**32 threads**
M6	0.727	0.717	0.745	0.639	0.722	0.648
US Roads	0.0195	0.0194	0.0194	0.0194	0.0196	0.0191
Europe OSM	4.889	4.937	5.033	5.049	4.899	4.944

3.2 Execution time plots







References

[1] Scott Beamer; Krste Asanovic; David Patterson. Direction-Optimizing Breadth-First Search.