Experimental Algorithms:
Exercise sheet 4

Dr. Alexander van der Grinten

January 11, 2021

Notes

- Due by January 21, 2020, 9:00 AM.
- See the remarks on exercise sheets 1 and 2 (i.e., regarding the grading of exercises and on running your experiments).
- We reuse the same input instances (cit-patent and roadNet-TX) as on sheet 3. (To decrease the size of the zip files, the instances are not included in the archives on Moodle).
- Run all experiments on `gruenau1`.

Exercise 9: Multi-Source Bellman-Ford

The algorithm by Bellman-Ford can easily be adjusted to handle multiple sources: instead of operating on a single d_s vector, the algorithm can operate on a $D = (d_{s_1}, \ldots, d_{s_k})$ matrix simultaneously, i.e., it can handle a batch of k sources (namely, $s_1, \ldots, s_k$) at a time.

This exercise makes use of the `ex4-graph-mssp.zip` framework from Moodle.

(a) Adjust the GPU implementation of Bellman-Ford to support multiple sources.

(b) Adjust the CPU implementation to handle multiple sources. What is a good memory layout for the D matrix?

Optionally: can SIMD be used to accelerate the algorithm? To simplify the code, you may assume that the batch size is a multiple of 32.

(c) Compare your multi-source Bellman-Ford implementations (on the CPU and GPU) to an algorithm that runs Dijkstra from multiple sources in parallel on the CPU. How does each algorithm scale with the number of SSSPs per batch? How does it scale with the number of threads?

Hint: first determine the scalability w.r.t. the batch size. Alterwards, fix a good batch size (the `experiments.yml` file is pre-configured to 1024 here; you can change this setting as desired) and evaluate the parallel scalability.

Exercise 8: Δ-Stepping

This exercise makes use of the `ex4-graph-delta.zip` framework from Moodle.

(a) Implement the shortcutting procedure of Δ-Stepping. Note that since this adds edges to the graph, it requires re-constructing the CSR matrix (or a different representation such as adjacency arrays).

(b) Implement Δ-Stepping using OpenMP. Use your shortcutting routine to preprocess the graph. When doing this, set Δ to the minimum edge weight and keep doubling Δ until the graph size $(n + m)$ blows up by more than a constant factor α.

Let each thread be reasonsible for performing relaxations for a fixed subset of the vertices (i.e., the thread performs all relaxations of edges that *start* at vertices of this set). For example, this can be implemented by wrapping the algorithm in a top-level `#pragma omp parallel` scope.

- You can use randomized dart-throwing to "send" updated distance values to the thread that is responsible for the target vertex of a relaxation. (However, it is not required to implement sophisticated load balancing.)
- As an alternative to randomized dart-throwing, you can index the edges and have each thread write relaxations that are caused by a certain edge e into an array, with the index being determined by the edge index of e.

(c) Evaluate your parallel Δ-Stepping procedure in comparison to a sequential Dijkstra. Pick a reasonable value of α for the experiments. How does Δ-Stepping scale with the number of threads?