15618 Project Milestone Report

Haoxuan Zhu (haoxuanz), Ziyi Liu (ziyiliu)

TITLE
Parallelizing Genetic Algorithm for Traveling Salesman Problem

URL
https://hxzhu9819.github.io/pGA/

UPDATED SCHEDULE

Time	Plan
Week 1 11/07 - 11/13	Finish project proposal and start sequential implementation
Week 2 11/14 - 11/20	Finish sequential implementation and start parallelizing using CUDA on GHC machines
Week 3 11/21 - 11/27	Finish CUDA implementation and project milestone report
Week 4 **11/28 - 11/30**	DONE: brute force, sequential, and CUDA implementation, and project milestone report
12/01 - 12/04	Start OpenMP implementation - Haoxuan Zhu Find larger datasets with correct answers to verify implementation correctness - Ziyi Liu
Week 5 12/05 - 12/08	Finish OpenMP implementation and profiling on GHC machines with small number of cores - Haoxuan Zhu Profile bottlenecks in CUDA implementation to achieve better scaling with large populations and more cities - Ziyi Liu
12/09 - 12/11	Run OpenMP implementation on PSC machines with up to 128 cores to compare performance - Haoxuan Zhu Profile bottlenecks in OpenMP implementation and try different parallelization strategies to improve performance - Ziyi Liu
Week 6 **12/12 - 12/18**	(final report due) Report writeup and demo prep

WORK COMPLETED SO FAR

In the past weeks, we followed our schedule and completed sequential and CUDA implementation of the genetic algorithm for the traveling salesman problem. One problem encountered was how to check the correctness of our solution. We found a TSP dataset with real-world examples (see reference) but the dataset is too large to serve as test cases during the development phases, especially for sequential versions . To address the issue of lacking proper dataset with samples and answers online, we implemented a problem generator and a brute-force solver for verification. For sequential implementation, we tried different crossover and mutation strategies in order to converge quickly to the optimal solution. For CUDA implementation, we adopted those strategies, but due to the lack of communication among CUDA blocks, we also experimented with several ways of choosing crossover parents to reach the desired solution.

GOALS AND DELIVERABLES

We believe that we should be able to finish the goals and deliverables in our proposal. In addition, we plan to finish the “nice to haves” in our proposal as well. Specifically as follows:

- Brute-force implementation for sanity check.
- Sequential implementation that can converge quickly on relatively small samples.
- Scalable CUDA implementation.
- OpenMP implementation evaluated on both GHC and PSC machines.
- Comparison and analysis of speedups for the implementations.
- Bottleneck analysis using profiling tools for different parallelization strategies.

POSTER SESSION

For the poster session, we plan to report our comparative analysis results through graphs that illustrate the speedup of different approaches. We also plan to have a live demo showing the convergence to the optimal solution for the different implementations. We hope that people can see through the demo that the parallelized implementations converge much faster than the sequential version.

PRELIMINARY RESULTS

We haven’t run any structured experiments yet. However, based on our observations, brute-force implementation is only able to compute samples of less than 15 cities within a reasonable amount of time, whereas sequential implementation can compute even 30 cities quickly. However, it fails to converge at 30 most likely because of the limited population size. CUDA, on the other hand, can accurately compute 30 cities really quickly.

CONCERNS

One of our main concerns is the lack of large problems with answers. Our brute-force implementation can only compute the optimal solution with a small number of cities (less than 15). With more cities, the factorial becomes too large to compute. Therefore, we are unsure whether the solution given by the parallelized version is indeed optimal.

A second consideration is how to accurately compute the execution time of the CUDA implementation. For sequential implementation, we find the best route in each iteration by going through the entire population. If the route remains unchanged for some iterations, we will stop the algorithm. This gives accurate convergence time for the sequential version. In CUDA, in order to get the entire population, a cudaMemcpy from device to host needs to be performed every iteration, which increases communication costs. However, if copied back to host only after all iterations are done, or every X iterations, the convergence time will be hard to calculate and compare with that of the sequential version.

Finally, our current CUDA implementation cannot scale to a large number of populations as the number of cities increases. We suspect that this is caused by the inefficient use of memory. We will work on identifying the exact cause and fixing the issue in the coming weeks.

Reference:

1. TSP dataset: https://www.math.uwaterloo.ca/tsp/world/countries.html