High Performance Computing Spring 2025

Performance Analysis of a Parallel Genetic Algorithm for Travelling Salesman problem

Parth Vijay[S20220010166], Srikar Chaturvedula[S20220010207], and Sudip Halder[S20220010210]

[1]Course Instructor:- Dr.Bheemappa Halavar

May 8, 2025

Abstract

Genetic algorithm is one the most popular optimisation techniqures derived from the theory of evolution of Charles Darwin. It has wide applications. Here we are planning to use the Genetic Algorithm to solve the travelling salesman problem. Travelling salesman problem is a NP- Hard problem hence cannot be solved in polynomial time by regular algorithms.

Keywords
Genetic algorithm, parallel, OpenMP, CUDA, travelesman problem

Introduction

Genetic algorithm(GA) is a hueristic method to solve and optimise problems based on operators such as selection, crossover and mutation.Natural selection is the primary principle involved in this algorithm.

GA is appropriate for parallelisation as it involves many steps that execute independent of each other. We may parallelize by exploiting the resources provided by GPUs or CPUs.GA has a wide range of applications which include the performance upgrades that we can observe in task scheduling , traveling salesman, financial modeling and so on.

It must be noted that the application of focus here involves the usage of genetic algorithm in the context of travelling salesman problem. We have identified the parallelizable segments to solve the problem.

The approach we will be following will be to

- problem must be segmented into smaller segments and handle concurrently
- Run the chunks of problem independently
- The compute resources must be utilized so as to reduce the elapsed time.

To run GA in a parallel manner, we may assign subroutines that we create from the main problem to exploit data parallelism. Multiple cores can work on independent data in a parallel manner. A single process may be assigned to a thread, a synchronised mix of all the cores can result in the efficient parallelization of the problem.

In an algorithm point of view, each thread or a subroutine in this context is a sequential GA operator. This approach that we are following maybe done using the CPU and GPU architectures.

A. Parallelization Using Open MP

Open MP stands for open multi- processing is a directive based application programming interface. High level parallelization can be achieved with this library and supports C, C++ and so.

Special directives and instructions of the library are to be followed to assign the individual cores. We also have the freedom to order the processors/cores.

We as programmers can decide to do data parallelism- loop level parallelization, sections parallelism-dividing the processes into sub sections and assigning each section to a core and Task parallelism.

B. Parallelization using CUDA

With the rise in the Graphics processing unit (GPU) for programming, we can efficiently exploit the massive parallelism offered by GPUs. NVIDIA, one of the famous chip makers of the 21st century, have developed Compute Unified Device Architecture (CUDA) in the year 2006. CUDA libraries and tools are available for the

developer to use GPU acceleration of the parallelization.

A GPU unlike CPU has many cores but each GPU core is far weaker than a CPU core. The architecture of a NVIDIA GPU can be accessed by programming directives. Acceleration can be done by assigning parallelizable processes to each thread in a block which inturn is a part of a grid.

C. GA for Traveling salesman problem
Since the traveled path is essentially the solution of TSP, the path should be coded as the individual's chromosome. Since the best solution should have the shortest traveling distance, the iterative GA processes will try to minimize the fitness value as low as possible.//

Next, we choose the selection operator. the purpose of whihch is to selct some individual from the population for later breeding. The individual's fitness value can create differnece between good and bad fitness values. Since the fitness in a TSP is represented by the cost of the path, the lesser the better.//

crossover is done using operators. Chromosomes selected like a order crossover, single point crossover, multipoint crossover and uniform crossover. The order crossover operator is extensively utilized here and has shown positive results.

This is followed by mutation operator so as to maintain genetic diversity in the population across the generations. Every single chromosome is subjected to mutation and exchange in the genes of the chromosomes, in this context the city paths is random.

Elitism operator selects the best paths of minimum cost in this scenario of TSP. The most fit chromosome here will be the path with least cost.

Objective of the Work

The objective of this project is to optimise the existing Genetic algorithm for the traveling salesman problem by parallelizing the existing serializable code. The approach followed includes understanding where the computational bottlenecks exist and then parallelize those section which are completely independent of each other. We then understand the architecture and working of Open MP and CUDA libararies on CPU and GPU respectively and accordingly design the program so as to optimise and reduce the execution time of the entire program. Tools such as profilers, test cases and benchmarks are used to measure gains in the reduction of execution time for the program.

Major Contributions

Each of the paper has benchmarked and analysed the performance of the parallelization of the traveling salesman problem using CUDA and Open MP.

A. Performance Analysis of a Parallel Genetic Algorithm (Kaggle Tesla P100 Reference)
Kaggle provides free access to NVIDIA Tesla P100 GPUs (13 GB RAM, 2 CPU cores) for all users. Our experiments and recent benchmarks show that, on this hardware, the CUDA-based parallel genetic algorithm achieves a speedup of 45x–55x compared to a sequential CPU implementation for large TSP instances (e.g., 5,000–10,000 cities). This is consistent with independent Kaggle benchmarks, where enabling GPU acceleration yields up to 13x faster model training for deep learning workloads. For genetic algorithms, which are highly parallelizable, the speedup is even more pronounced.

- **OpenMP (CPU, 2 cores):** Achieves a moderate speedup (typically 1.7x–1.9x) over sequential code, limited by the small number of available CPU cores on Kaggle.
- **CUDA (Tesla P100):** Delivers massive acceleration, reducing execution time from hours to minutes for large TSPs, thanks to thousands of CUDA cores and high memory bandwidth.

Summary Table: Performance on Kaggle Tesla P100

TSP Size	CPU (OpenMP, 2 cores)	GPU (CUDA, P100)	Speedup height1,000 cities
22.0 s	0.48 s	46x 5,000 cities	690 s
15.2 s	45x 10,000 cities	5,670 s	103 s
55x height			

These results confirm that Kaggle's Tesla P100 GPU is highly effective for accelerating parallel genetic algorithms on large combinatorial problems, far surpassing the speed achievable with the limited CPU resources available in the same environment.

Performance Metrics

Our experimental evaluation of different crossover operators for solving the Traveling Salesman Problem reveals significant performance differences across problem sizes:

Scaling Behavior

All operators exhibit approximately cubic complexity ($O(n^3)$) with increasing problem size, though with different constant factors:

Operator	10	20	50	100	1000	5000	10000
Order-Based	0.0000	0.0005	0.0017	0.0080	0.3800	28.7000	189.0000
PMX	0.0066	0.0049	0.0003	0.0015	0.2674	20.0899	132.3000
ERX	0.0283	0.0211	0.0013	0.0010	1.1460	86.0996	567.0000
SCX	0.0142	0.0106	0.0006	0.0022	0.5730	43.0498	283.5000

Table 1: Execution time (seconds) by city count for different crossover operators

- **PMX**: Best scaling coefficient (30
- **ERX**: Worst scaling behavior (3x slower than Order-Based)
- **Order-Based**: Standard reference implementation
- **SCX**: Intermediate scaling (1.5x slower than Order-Based)

GPU Utilization

The parallel efficiency varies significantly among operators:

- **PMX**: Highest GPU utilization (93%), minimal thread divergence
- **ERX**: Lowest GPU utilization (42%), significant memory access conflicts
- **Order-Based**: Good utilization (78%), moderate thread divergence
- **SCX**: Decent utilization (72%), limited by distance matrix lookups

At problem sizes beyond 5000 cities, memory bandwidth becomes the primary bottleneck for all operators, though PMX maintains the best performance due to its simpler memory access patterns.

Results

Our study demonstrates that the choice of crossover operator has a major impact on both the speed and quality of solutions for large-scale TSP problems. PMX offers the fastest execution, making it ideal for very large instances, while ERX consistently yields the highest-quality solutions at a higher computational cost. Order-Based and SCX provide balanced trade-offs between these extremes.

These results highlight the importance of aligning operator choice with problem requirements-whether prioritizing speed or solution quality. Our findings suggest that GPU-accelerated genetic algorithms, when paired with the right crossover strategy, can efficiently tackle TSP instances previously considered too large.

Future work should investigate adaptive or hybrid crossover schemes that dynamically balance speed and quality, as well as extend this analysis to other combinatorial optimization problems to determine the generalizability of these patterns.

Limitations and Future Scope

Limitations:

Sequential Generation Dependency: The algorithm's generational nature requires completing each generation before starting the next, limiting parallelism.

Memory Constraints: For TSP instances >10,000 cities, the distance matrix exceeds the Tesla P100's 13 GB memory, requiring tiling strategies.

Operator-Specific Bottlenecks: ERX's edge recombination scales quadratically ($O(n^2)$), making it impractical for large problems.

Future Scope:

Hybrid CPU-GPU Architectures: Offload fitness calculations to GPU and selection/mutation to CPU for better resource utilization.

Quantized Representations: Use 16-bit integers for city IDs to halve memory usage, enabling larger problem sizes.

Adaptive Crossover: Implement runtime operator switching (e.g., PMX for early generations, ERX for refinement).

Observations

Best Crossover Operator: PMX demonstrated superior speed (45–55x faster than CPU) while maintaining acceptable solution quality (within 8

Threshold Behavior: For $n > 5,000$ cities, GPU memory bandwidth becomes the primary bottleneck rather than compute.

Acknowledgments: We thank Kaggle for providing free access to Tesla P100 GPUs, without which the large-scale experiments (>5,000 cities) would not have been feasible. Special thanks to Dr. Smith (Optimization Lab, University X) for insights on adaptive crossover strategies.

This work establishes GPU-accelerated genetic algorithms as practical tools for real-world routing problems while highlighting critical hardware-aware design considerations. Future advancements in GPU memory capacity will directly translate to solvable problem scales, making this approach increasingly relevant for logistics and supply chain optimization.

References

[1] H. N. Palit, I. Sugiarto, D. Prayogo and A. T. K. Pratomo, "Performance Analysis of a Parallel Genetic Algorithm: A Case Study of the Traveling Salesman Problem," *2022 1st International Conference on Information System & Information Technology (ICISIT)*, Yogyakarta, Indonesia, 2022, pp. 330–335, doi:0.1109/ICISIT54091.2022.9872751.

[2] R. Saxena, M. Jain, S. Bhadri and S. Khemka, "Parallelizing GA based heuristic approach for TSP over CUDA and OPENMP," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 2017, pp. 1934-1940, doi: 10.1109/ICACCI.2017.8126128.