Investigating a Multithreaded Solution To a Rubik’s Cube Solver

By

Ben Dickinson



exploring and comparing different techniques for solving a Rubik’s cube challenge. This report will discuss the discrepancies of Multi and sequential threading in solving a complex Rubik’s cube problem.

September 2023

SIT315 Deakin University

*Abstract*

The Rubik's Cube, an iconic and challenging puzzle, has captured the fascination of puzzle enthusiasts and computer scientists alike. In this report, we explore the task of solving a Rubik's Cube using two distinct approaches: multi + distributed threading and sequential threading. The objective is to evaluate the efficiency of these two methods and determine which is better suited for the complex task of solving the Rubik's Cube.

The Rubik's Cube presents an intriguing problem due to its vast number of possible configurations. Its solution requires the execution of a series of complex algorithms, and the exploration of various search spaces. We aim to address the question of whether parallelism, specifically through multi + distributed threading, can significantly accelerate the solving process compared to the traditional sequential approach.

*Investigation and experimentation*

To study the differences between these possible solutions, we will implement a few distinct solving algorithms. The first utilizes a sequential threading technique, allowing for a single flow of processes which occur one after the other. The second, a multi-threading technique, enabling concurrent processing of cube states on a single machine. And the third employs distributed threading, allowing multiple machines to collaborate on the solving process through a networked environment. In this experiment, we aim to measure and compare the performance of these approaches in terms of solving time and scalability.

* *hypothesis*

In our study, we postulate that the sequential implementation will exhibit exponential growth in runtime as the problem size scales up. Conversely, for the parallel implementation, we anticipate a more linear increase in runtime as the problem size grows. Furthermore, in the case of the MPI + parallel implementation, we expect to observe a more pronounced increase in runtime, surpassing that of the purely parallel approach, due to the additional overhead introduced by distributed computing.

*Methods and Resources:*

***Programming Environment:***

The Rubik's Cube solving project was developed using the Microsoft Visual Studio development environment, utilizing the C++ programming language. Visual Studio provides a comprehensive integrated development environment (IDE) with features that streamline code development as well as efficient debugging applications.

***Parallelization Framework:***

The project will utilize the Microsoft Message Passing Interface (MPI) library to implement both multi-threading and distributed computing approach. MPI is a widely used library for developing parallel and distributed applications. It facilitates communication and coordination among multiple processes, making it well-suited for tasks that can benefit from parallelism across multiple computing resources.

***Multi-Threading Approach:***

For the multi-threading (Parallel) approach, the project employed C++'s native multi-threading capabilities, making use of the <thread> library. Threads were created to concurrently explore different branches of the Rubik's Cube solution space, thereby taking advantage of the multiple cores available on the CPU.

***Performance Evaluation:***

To assess the efficiency of the implemented methods, the project will conduct a systematic performance evaluation. A runtime metric will be declared to assess the efficiency of the implementation, the speedup, and scalability. These evaluations will provide insights into the advantages and limitations of each approach, helping to draw conclusions regarding their effectiveness in creating the best Rubik's Cube puzzle solver.

*Results*

In order to determine the optimal solution for solving Rubik's Cube, I devised three distinct mock-up programs: a sequential program, a parallel program, and a parallel + MPI program. These programs were designed to address various complexities of the Rubik's Cube problem, featuring different combinations of faces, moves, and sizes.

Utilizing a fundamental matrix system, where numerical values ranging from 1 to 5 corresponded to distinct cube colors, each program systematically tackled the Rubik's Cube challenge. These simulations served as a critical foundation for our comparative analysis, enabling us to gain valuable insights into the efficiency and performance of each solution approach under varying levels of complexity.

*Discussion of results*

The presented results provide a comprehensive overview of the runtime performance for the three different algorithms – Sequential, Parallel, and Parallel + MPI – as they are applied to Rubik's Cube solving tasks of increasing complexity. The graph displays the execution times in microseconds (µs) for each algorithm on the y axis, and representing the Rubik's Cube data complexities ranging from 10 to 80 units on the x axis. Example, a complexity of 40 equals a 40 faced cube, 40 x 40 cube size and 40 max moves. (just for testing purposes, this is not accurate to a real-world problem, just a metric used to test the cube’s efficiency)

REWRITE HIGHLITED PART

\*\*Sequential Algorithm:\*\*

The Sequential algorithm demonstrates an exponential increase in runtime as the complexity of the Rubik's Cube problems grows. As indicated by the data, solving a Cube with a complexity of 80 requires significantly more time (approximately 3,445,109 µs) compared to simpler configurations. This aligns with the initial hypothesis that the Sequential approach would exhibit exponential runtime growth.

\*\*Parallel Algorithm:\*\*

In contrast, the Parallel algorithm showcases a more linear increase in runtime as the problem complexity escalates. This suggests that parallelization effectively distributes the workload across multiple processing units, resulting in a more predictable and scalable performance. Solving a Cube with a complexity of 80 still requires more time than simpler problems but does not exhibit the same steep increase observed in the Sequential approach.

\*\*Parallel + MPI Algorithm:\*\*

The Parallel + MPI algorithm introduces distributed computing capabilities using MPI (Message Passing Interface). While it offers parallelism benefits, the inclusion of distributed computing introduces additional communication overhead, causing a more noticeable increase in runtime compared to the pure Parallel approach. Solving the most complex Rubik's Cube problem in this configuration (complexity 80) requires approximately 1,311,727 µs, signifying a substantial performance penalty compared to the purely parallel solution.

In summary, these results underline the advantages of parallelization in managing the increasing computational load as problem complexity rises. The Sequential algorithm's exponential runtime growth becomes a significant bottleneck for complex tasks. However, the introduction of distributed computing via MPI, while extending the problem-solving capabilities, does impose additional overhead. The choice of the most efficient algorithm depends on the specific problem complexity and available computational resources, emphasizing the trade-offs between sequential, parallel, and parallel + MPI approaches in Rubik's Cube solving.

*Documentation of best found design*

DISCUSS THE FINAL CODE SOLUTION\*

*Conclusion*

References

<https://hobbylark.com/puzzles/Rubik-Cube-Algorithms>

<https://rubiks-cube-solver.com/>

<https://chanzy3.github.io/Rubik-s-ParaCube/docs/final_report/15618_final_project.pdf>