Optimizing Multi-Agent Path Finding on Multi-Core Platforms

Steven Liu, Zhifei Li

Summary

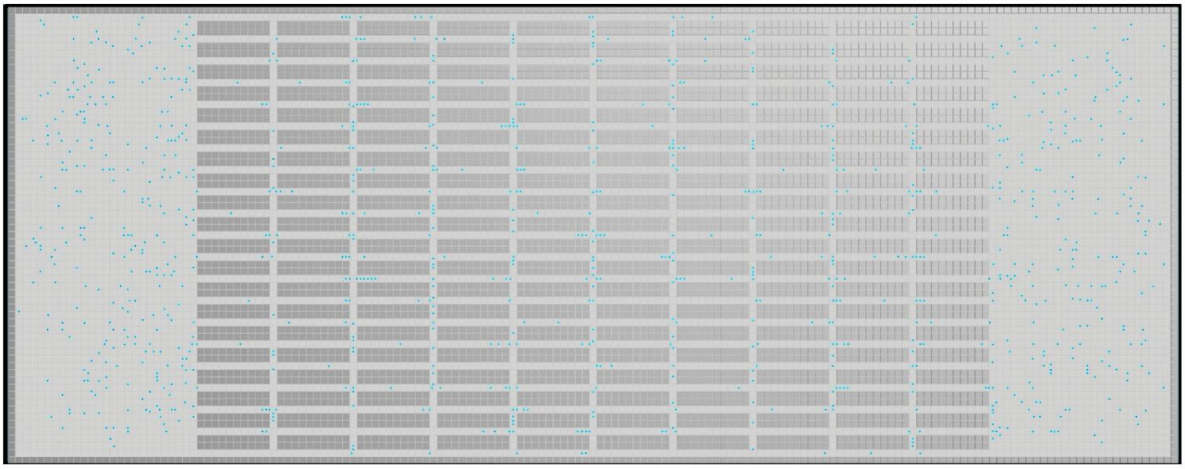
We are going to design, build and optimize the planning algorithm for Multi-Agent Path Finding Problem (MAPF) using multi-core platforms. We will also analyze the speedup and solution costs across different high level algorithm approaches.

URL

https://s-liu0411.github.io/15418_final_project/

Background

The Multi-Agent Path Finding (MAPF) problem focuses on planning collision-free paths for large groups of robots within a known environment (for instance, planning the robots' route in a warehouse). Therefore, for each robot at each time step, all other robots are considered as obstacles to avoid collisions. Furthermore, an important metric that we need to optimize is the total time for all robots to reach the final destination and we refer to it as the sum of costs (SOC).



warehouse-20-40-10-2-1
1000 Agents

Carnegie Mellon University

Another important thing for this problem is the time to solve a solution (collision free path). Since the search space is 3d (including the time frame as the third axis), there are a lot of valid solutions for a particular input. Hence, one may use an anytime algorithm to iteratively solve for a better solution in terms of SOC.

One of the advantages that we can gain by using parallelism is to significantly accelerate the exploration of the solution space and reduce the overall computation time. By distributing the workload across multiple processing cores, different parts of the search can be conducted concurrently, allowing for faster convergence towards an optimal or near-optimal solution.

Another place for parallelism to occur is to allow multiple agents to search for their routes concurrently. By independently executing pathfinding searches for each agent simultaneously, we can leverage multi-core processors to dramatically reduce overall computational time. However, since concurrent planning might introduce conflicts (collisions), we must implement efficient synchronization mechanisms and conflict-resolution strategies to maintain consistency and optimize the solution quality (sum of costs). This parallel approach effectively balances workload and computational efficiency while ensuring robust solutions.

Parallelism can also be applied to planning the priority of agents, which is essential for resolving conflicts efficiently in MAPF problems. By concurrently evaluating and assigning priorities to different agents or groups of agents, we can significantly speed up decision-making regarding agent precedence. Parallelizing this priority planning step can reduce bottlenecks caused by sequential priority evaluation, allowing more rapid determination of agent ordering and subsequently faster overall solution convergence.

Challenge

First of all, there are a lot of high-level approaches that we can think of. However, each one of them has its own limitations. For instance, if we have all robots to move at the same time and address the potential conflicts every time step, this will result in too much communications and synchronization overhead, and the computation for search itself is relatively cheap compared to communication and synchronization. How frequently we need to perform synchronization and communication would be a challenge.

Another way of thinking about the problem is to partition the robots into groups and address potential conflicts within the groups sequentially. However, due to the nature of the problem, the paths across the groups are not independent of each other. Resolving conflicts within a group may add new conflicts across the group. The data dependency of this problem is very hard to solve.

In addition, the workload imbalance in the problem is obvious – the computation required to search for each robot differs a lot. Therefore, we might need to come up with heuristics to guide the workload balancing.

Resources

We would probably use the lab server (128 cores) in the robotics lab that we have been working on, but it is still subject to approval and sometimes the load for the server is high since all lab members run experiments on it. Otherwise, we would first benchmark on GHC machines and possibly on PSC machines for high thread counts or heavier test cases.

In terms of the code base, we may reference some existing MAPF algorithms written in C++. But since we are trying to use different high-level approaches to make the algorithms parallelizable, we will potentially need to start from scratch and change it to parallel versions.

Here are some materials that we may reference to:

https://github.com/Jiaoyang-Li/PBS
https://jiaoyangli.me/research/mapf/

Goals and Deliverables

Plan to achieve:

1. We will come up with different implementations we have in mind (priority planning and batch processing described in background) and starts with its sequential version
2. We will come up with a parallel version of the search algorithm since it is the fundamental building block for the problem
3. We will benchmark our different algorithms using the metrics of sum of costs as well as the corresponding speedup vs the sequential version and try to achieve speedup close to existing parallel MAPF algorithms.
4. We will provide a video demo to illustrate the problem

Optional:

1. Try to work on a decentralized version of one of the implementations (distributed planning) using MPI
2. Try to combine different forms of parallelism to achieve better performance and time
3. Try to use more realistic robot model and optimize the overall simulation time by resolving unexpected conflicts introduced in the execution phase

Our project is an analysis project. We hope to investigate the tradeoff between the sum of costs across different parallel implementations and the corresponding speedup across different implementations. Furthermore, given a fixed planning time, which parallel implementation would be better? In this case, we care more about the absolute time during the planning phase and try to see which parallel version can achieve the best objective function within a certain time.

We hope to learn the advantages and disadvantages of solving the problem in parallel and gain a deeper understanding of sacrificing optimality (minimizing synchronization) and gaining better performance. By attacking the problems in different angles, we get different bottlenecks, and we hope to learn and provide quantitative analysis on how different bottlenecks weigh in the problem.

If we get time to finish the optional section, we hope to learn more about the tradeoff between different framework in distributed parallel computing and how they affect performance in real-world scenarios.

Platform choice

The problem is inherently computation heavy as the search space is extremely large. Therefore, by applying parallel computing techniques, it should be able to get better performance by dividing up the work in an efficient and balanced way. We plan to complete the project in C++ and will use threads to solve the problem because there are a lot of dependencies and different instructions for different cases. If time permits, we would also try OpenMP implementation to test which performs better.

Schedule:

Week 1 (before April 4th): Design and implement sequential versions of at least 2 different algorithms. Testing for correctness of the sequential version

Week 2 (before April 11): Change the algorithm in parallel and verify the correctness of the parallel versions

Week 3 (before April 18): Designing experiments to test and benchmark, and analyze results to try to answer the questions raised in the proposal and compare pros and cons of different approaches. Write up the milestone report

Week 4 (before April 25): write up the final report and continue to finish up experiments and analysis. If time permits, start the optional section

Week 4.5 (before April 29): Finalizing code and material for submission. Make web page and poster