Milestone Report

Name:
Zixin Shen
Haoran Zhang
URL: https://jubilee101.github.io/ParallelRRT/

1. Schedule update

Week	Goal	Actual Progress & Plan
W1: 11/09-11/15	Implement Serial RRT and reproduce the parallelized version of RRT other paper shows	Implemented Serial RRT and reproduced the parallelized version of RRT other paper shows
W2: 11/16-11/22	Devise the Parallelized RRT, analyze the performance, figure out the bottleneck	Finished the visualization pipeline and evaluation pipeline, visualized the path and performance (switch the W4 to W2)
W3: 11/23-11/29	Improve our Parallelized RRT with k-d tree, compared with cuda version, and initial parallelized RRT version.	Analyzed the performance, figure out the bottleneck And according to our analysis, we devised the Parallelized RRT, got great improvement.
11/30	**Milestone Report**	**done**
W4: 11/30-12/08	Visualize the path generation and results, finalize the final report.	14.1 11/30 -12/4 Plan to improve our parallelized RRT with k-d tree, do comparison and analysis work. 14.2 12/5-12/8 finish the final report
12/09	**Poster Session**	**Poster Session**

2. Summary

In the past few weeks, we've accomplished most of the goals we originally planned. We implemented a basic serial version of the RRT algorithm which can be used to find a path from any given start point to any end point. In order to better test our implementation, we rearranged our schedule a little -- we moved the visualization part ahead so that we can visualize and verify the path we generated. We then wrote a benchmark script and measured the overall performance of the algorithm and identified some bottlenecks.

We choose two different maps for testing, map 0 is rather narrow and map 1 is a large map with more steps. Then we divided our solution into different phases to compare the time costs in different phases. We noticed that for the small map like Map 0, the process of inflating the map cost most of the time, according to Fig 3. And with the increment of the map and possible routes, the process of finding nearest weighs more. We can imagine when the map is extremely large, which means more nodes enrolled in, finding nearest will become the bottleneck of the problem according to the phase distribution in Fig 4. According to the above analysis, we decided to put emphasis on the optimization of finding nearest neighbors and inflating obstacles.

Based on the bottleneck phases identified, we developed a parallel version of the RRT algorithm using OpenMP. In fig 5 we present the improvement of each bottleneck phase after parallelization. You may find that there are some phases that did not improve as much as the overall performance. One of the reasons is that phases such as finding the nearest point have a lot of randomness in it, causing a large performance variance. We will look into deeper reasons and try to fix the problem. For Map 1, our parallel implementation gains a 6x speedup compared to the serialized version under the GHC machine with 8 CPU cores. The speedup gain for Map 0 is not as obvious as Map 1, with a 1.4x speedup. This is mainly due to the size of the map: it's too small for the parallel gain to really show and also the smaller map increase the randomization.

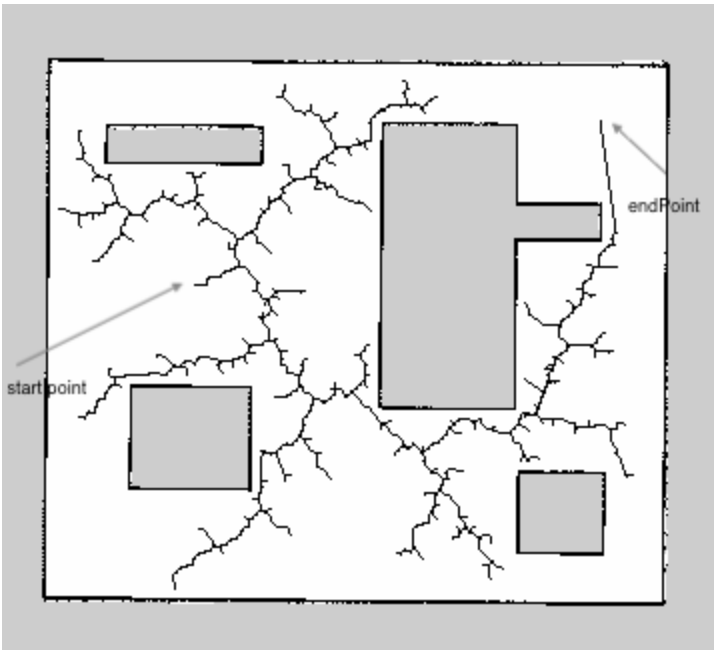

Fig 1: possible solution for Map 0

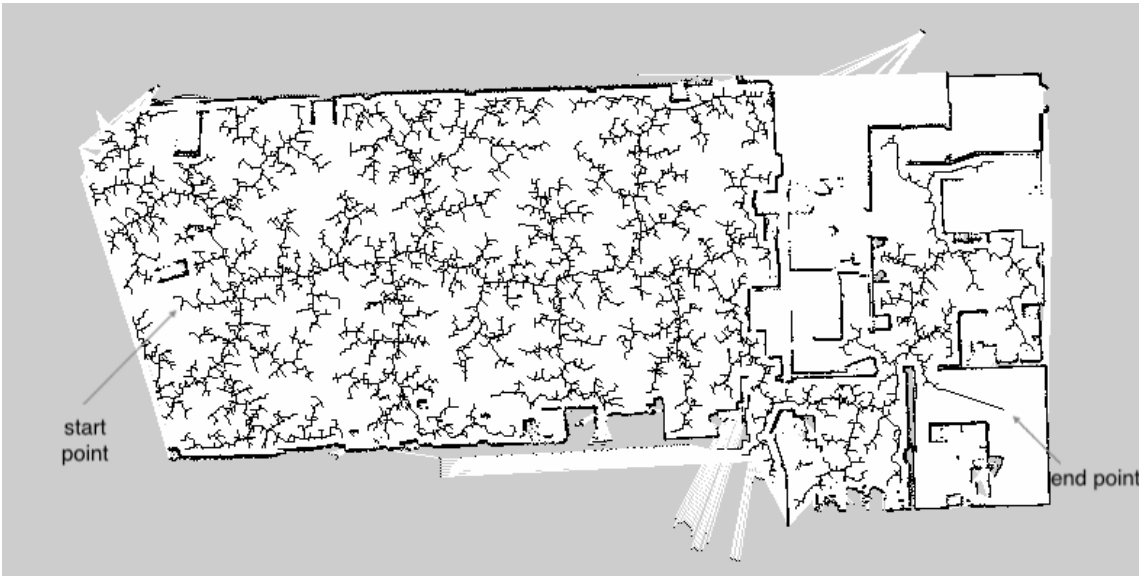

Fig 2f: possible solution for Map 1

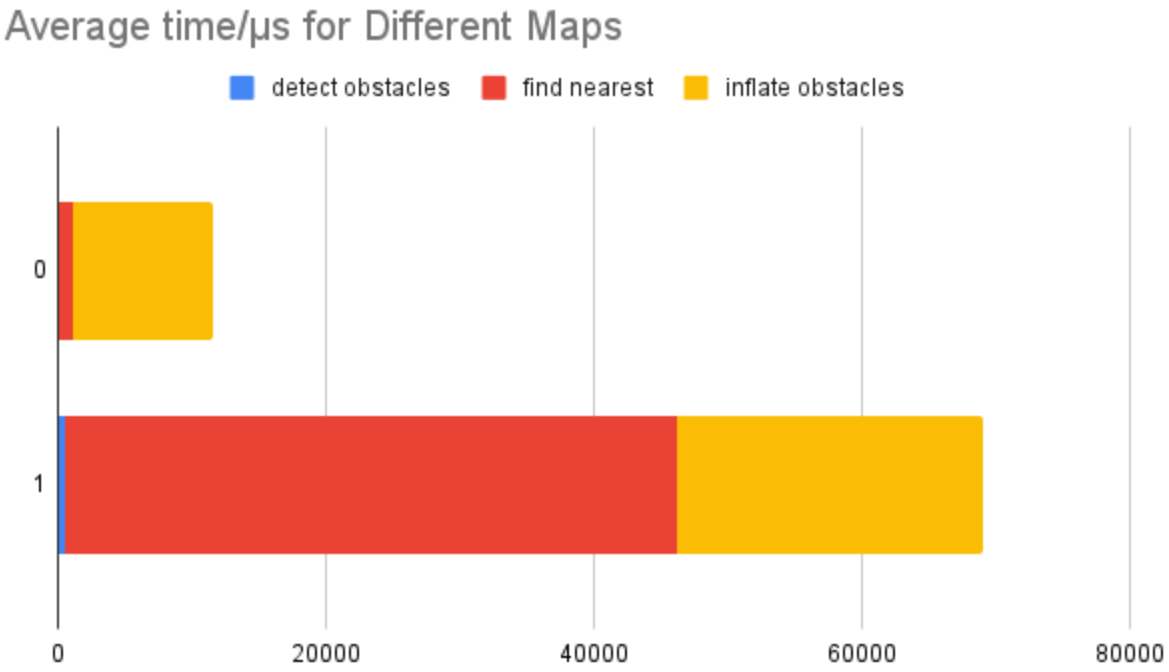

Fig 3: Average time for different maps

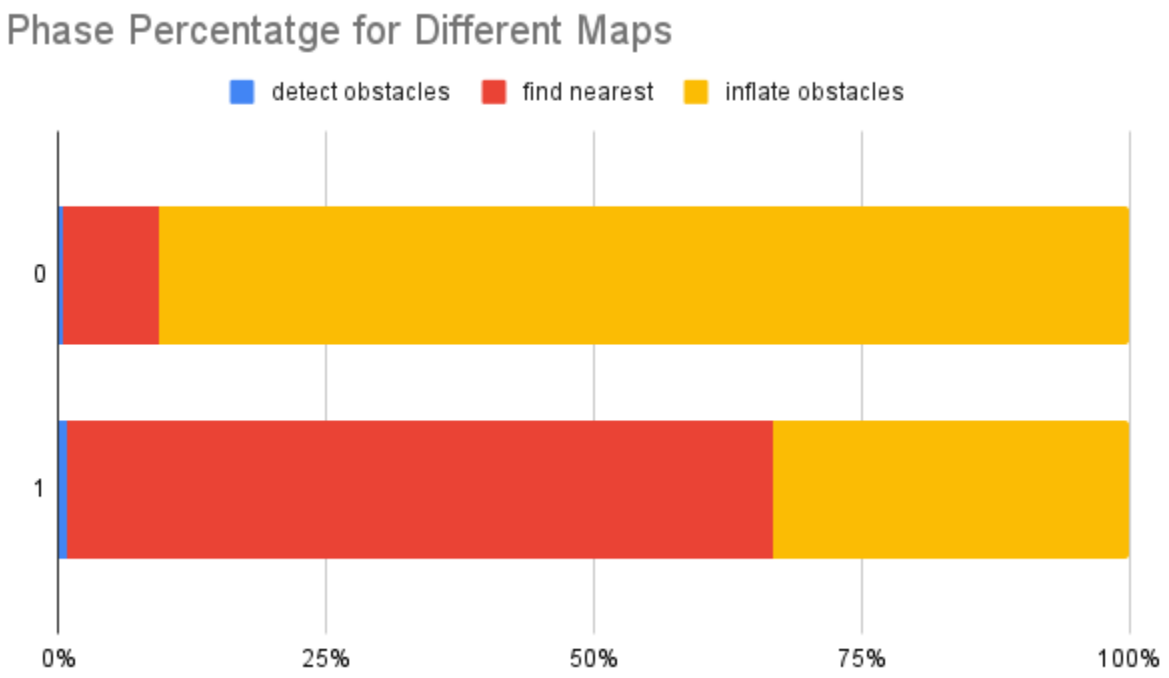

Fig 4: Phase percentation for different maps

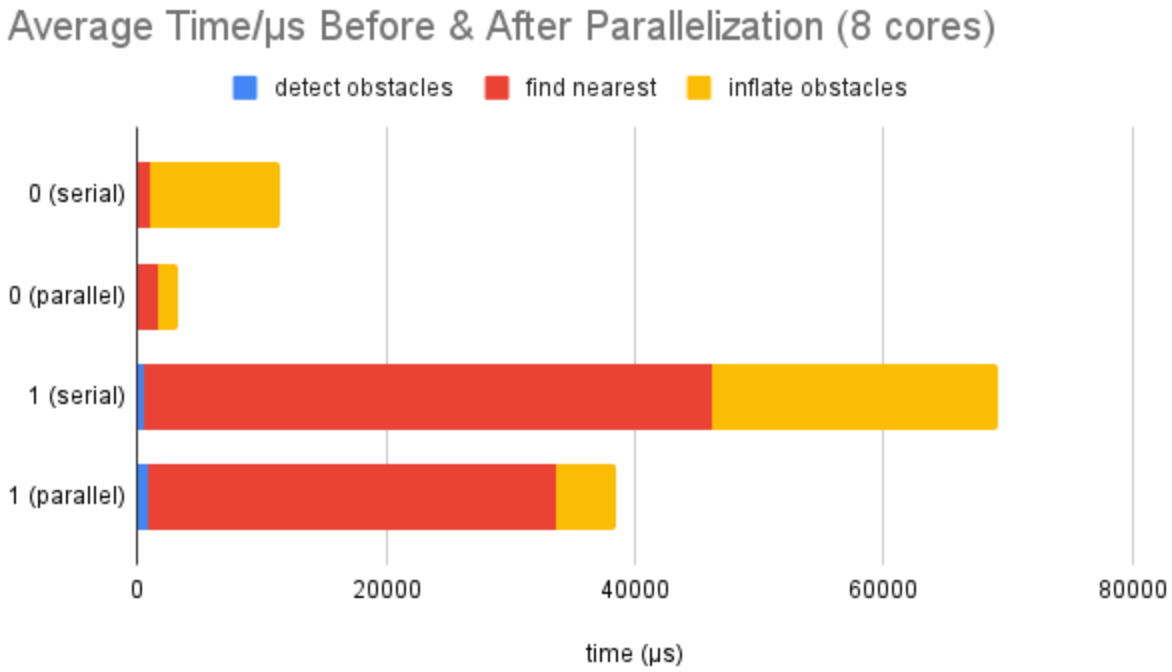

Fig 5: Comparison of the average time each phase spend before and after parallelization

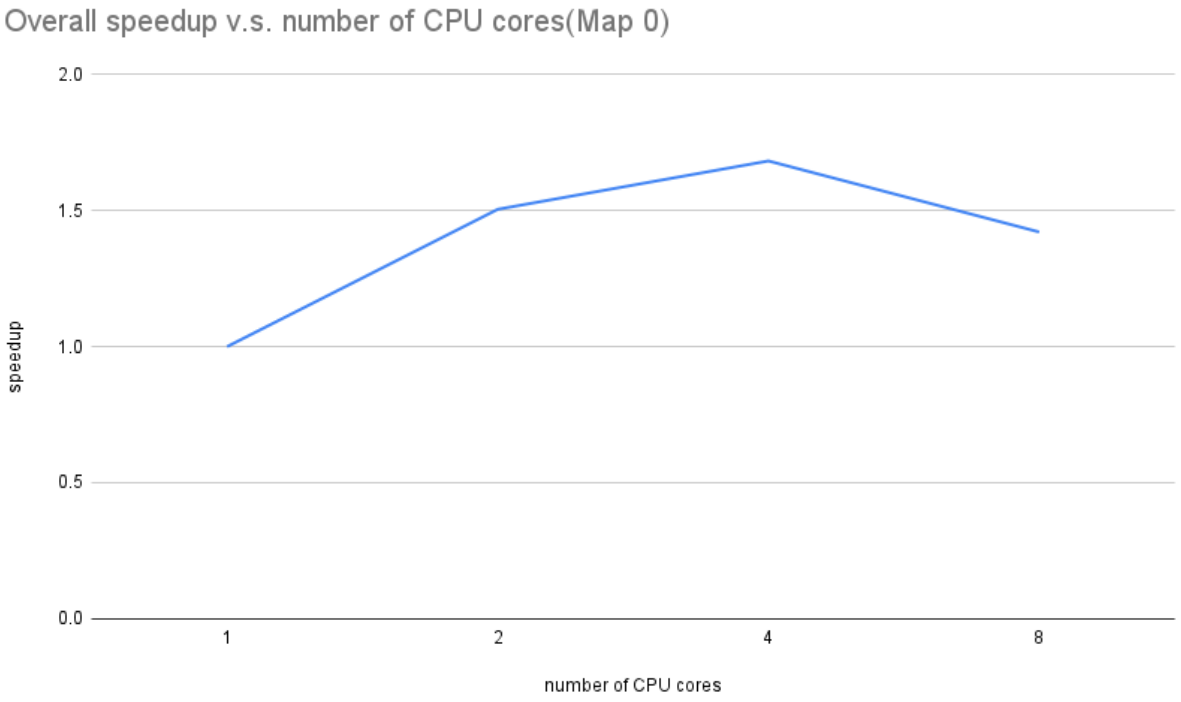

Fig 6: overall speedup v.s. Numbers of CPU cores(Map 0)

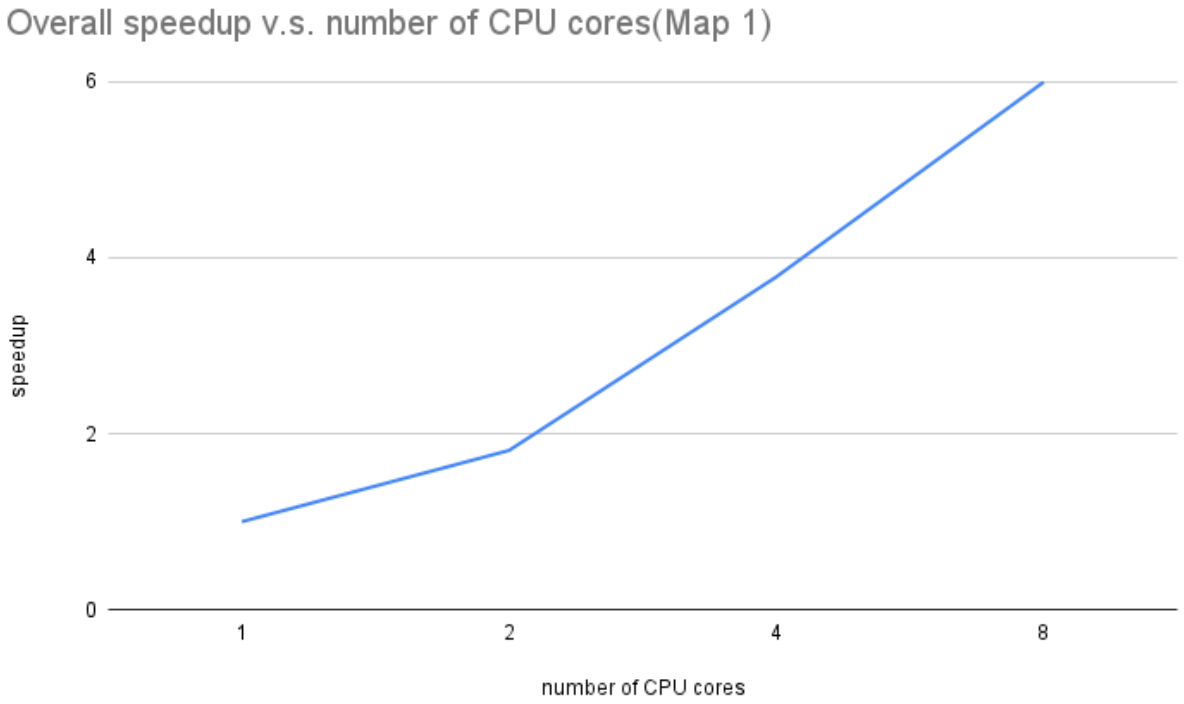

Fig 7: overall speedup v.s. Numbers of CPU cores(Map 1)

3. Goals and deliverables update

Our basic goal is to have a working version of the RRT algorithm, as well as the benchmark script to measure the performance, then implement a basic parallel version RRT and compare the performance differences. We already finished this part and get good performance.

Our ideal goal is to improve our parallel version RRT with better parallel strategy as well as using data structures such as k-d tree and compare the performance difference with version 1 and speedup with different number of cores. We will continue working on this part in the next one week.

4. What we will show in the poster session?

We will have a series of graphs that shows the gain of performance of our solution and the output path in the map.

5. Issues we met so far

1. One of the bottlenecks we identified, finding the nearest neighbor for the newly spawned point, is not gaining enough speedup with our OpenMP implementation. We changed the task

assignment mode from dynamic to static to avoid some scheduling overhead and improved the performance, but the speedup is still as much as we expected.

2. Another problem is that since RRT is a randomized algorithm to find the workable path, there exists variance between different runs which influence our evaluation of performance. We are now deciding to run our algorithms multiple times and get average evaluation to weaken such the influence, however the variances still exist.