Seth Hanusik, Raeshawn Bart, Robert Deal

CS6068 Final Project

Parallelization of A\* Search Algorithm

Table of Contents

[Introduction 1](#_Toc89008140)

[Design and Optimization 2](#_Toc89008141)

[Application Performance Analysis and Results 3](#_Toc89008142)

[Division of Work and Self-Assessment 4](#_Toc89008143)

[Bibliography 4](#_Toc89008144)

[Code Appendix 4](#_Toc89008145)

# Introduction

The goal of our research and application was to parallelize the A\* search algorithm. This was a good area of study because of the applications that search algorithms have in today’s computing world. Searching in addition to sorting are some of the most important algorithms when it comes to real world applications, and they are constantly trying to be improved upon to lower the runtime and complexity. A\* specifically has applications in areas like artificial intelligence because of its ability of an “informed” search.

When it comes to A\*, its complexity can vary based on the heuristic function that that is used because the better the heuristic, the less nodes that will be expanded that are not useful in finding the path to the goal. The heuristic can be classified in a few ways, the first is as an admissible heuristic, which is defined as “one that never overestimates the cost to reach the goal.” For example, that is saying that if the algorithm is estimating the cost of reaching the goal state in one move is 100, but there is a chance that going through another state is only going to cost 75, then it will not accept the solution that costs 100 because there appears to be a better solution. A stricter condition in classifying a heuristic is its consistency. A heuristic is said to be consistent “if for every node *n* and every successor *n'* of *n* generated by any action *a*, the estimated cost of reaching the goal from *n* is no greater than the step cost of getting to *n'* plus the estimated cost of reaching the goal from *n'*.” Therefore, if a heuristic is consistent, it will never even visit a state where the cost of moving to that state is greater than the estimated cost of reaching the goal state from its current state is plus the cost to move to that state. Having a consistent heuristic greatly improves the performance of the A\* algorithm. (Russel & Norvig, *Artificial Intelligence: A modern approach* 2010)  
 With that being said, the algorithm gets more efficient the less branches it has to traverse in order to get to the goal state. In the worst case, the time complexity of A\* is O(bd), where *b* is the branching factor of the heuristic used, and *d* is the depth that is needed to go to, to reach the goal state. Ideally, the branching factor of the heuristic will equal one, meaning it travels down a single path over its search.

/\* Possibly could add more information on research here, maybe get another citation to pull from specifically for parallelizing search algorithms. \*/

# Design and Optimization

In order to test the design of the algorithm, we implemented the algorithm to solve a tile game. The application starts with a start vector and goal vector in the style of a 3x6 tile table containing the numbers 0-17, where 0 represents the open position of the table.

Start

|  |  |  |
| --- | --- | --- |
| 3 | 1 | 2 |
| 6 | 4 | 5 |
| 9 | 7 | 8 |
| 10 | 0 | 11 |
| 12 | 13 | 14 |
| 15 | 16 | 17 |

Goal

|  |  |  |
| --- | --- | --- |
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |
| 9 | 10 | 11 |
| 12 | 13 | 14 |
| 15 | 16 | 17 |

The algorithm then starts with the starting vector and generates tables based on where that open position is and adds each of those new tables to a list that is sorted by priority, where the lower the priority the better. Priority (f) is calculated for each table by the number of places that are not in their correct slots (h) added to the accrued cost of the parent tables and the cost for moving that tile number (g). Cost for moving tiles is determined by the number being moved; for moving values 1 to 6, the cost is 1, for moving values 7 to 16, the cost is 3, and for moving the value 17, the cost is 15. For example, based on the starting vector above, the algorithm would generate the following four tables and their respective priorities.

Left: 4 + 3 = 7

|  |  |  |
| --- | --- | --- |
| 3 | 1 | 2 |
| 6 | 4 | 5 |
| 9 | 7 | 8 |
| 0 | 10 | 11 |
| 12 | 13 | 14 |
| 15 | 16 | 17 |

Up: 6 + 3 = 9

|  |  |  |
| --- | --- | --- |
| 3 | 1 | 2 |
| 6 | 4 | 5 |
| 9 | 0 | 8 |
| 10 | 7 | 11 |
| 12 | 13 | 14 |
| 15 | 16 | 17 |

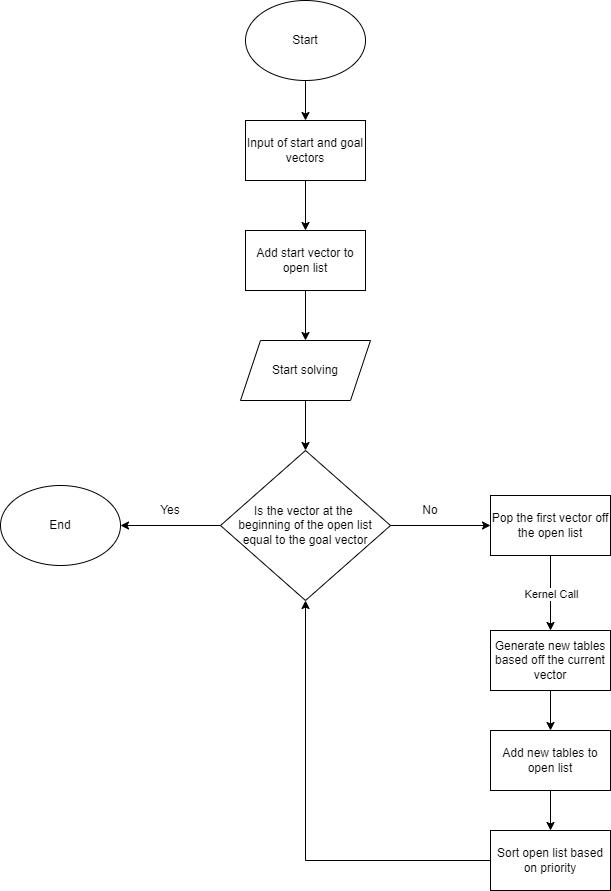
Right: 6 + 3 = 9

|  |  |  |
| --- | --- | --- |
| 3 | 1 | 2 |
| 6 | 4 | 5 |
| 9 | 7 | 8 |
| 10 | 11 | 0 |
| 12 | 13 | 14 |
| 15 | 16 | 17 |

Down: 6 + 3 = 9

|  |  |  |
| --- | --- | --- |
| 3 | 1 | 2 |
| 6 | 4 | 5 |
| 9 | 7 | 8 |
| 10 | 13 | 11 |
| 12 | 0 | 14 |
| 15 | 16 | 17 |

This process is repeated until a table is found that equals the goal vector and you can see the whole process in Figure 1 below.



Figure

As stated in Figure 1, we add parallelism in the generating of the new tables. Each table is generated by its own thread that way you can as many as four new tables in the time it takes to generate 1, doing the bulk of the computing in as little as a fourth of the time as it would take to run the algorithm serially.

# Application Performance Analysis and Results

For development, I was writing code in CUDA and executing the code on an Nvidia GTX 1660 Ti GPU.

For the application, the thrust library was implemented, specifically, the host\_vector and device\_vector types, and its sorting algorithm to sort a host\_vector. All generated tables were populated into an open list vector that was sorted based on priority and the one with the highest priority was used to generate new tables in the kernel call. The kernel call was executed with one 4 threads on one thread block and each of those threads was responsible for generating one of the possible four tables that could come from a parent table.

/\* More to be added here once we have working code and can compare the difference between our parallel and serial code \*/

# Division of Work and Self-Assessment

/\* Each member will have to add their own assessment \*/

# Bibliography

Russel, S. J., & Norvig, P. (2010). *Artificial Intelligence: A modern approach* (3rd ed.). Prentice Hall.

# Code Appendix

/\* I think once we have everything done, I will create a git repo to store all of our code and documents that I can link here so he can see both our parallel and serial code that we are comparing \*/