Parallelization of A* Algorithm

Amandeep Singh
Civil Engg, IITB
3rd Year Undergraduate
180040007@iitb.ac.in
+91-9465623338

Saksham Gautam
Chemical Engg, IITB
3rd Year Undergraduate
180020088@iitb.ac.in
+91-8290667950

Saurabh Khandelwal
Chemical Engg, IITB
3rd Year Undergraduate
180040093@iitb.ac.in
+91-8830845616

Rachit Adlakha
Chemical Engg, IITB
3rd Year Undergraduate
180020070@iitb.ac.in
+91-7024561319

ABSTRACT

A* is a best-first search algorithm for finding optimal-cost paths in graphs. A* benefits significantly from parallelism because in many applications, A* is limited by memory usage. So distributed memory implementations of A* that use all of the aggregate memory on the cluster enable us to solve problems that can not be solved by serial, single-machine implementations.

We plan to develop a parallel A* algorithm suitable for distributed-memory machines. Associated with each node u is a cost g of the current best path from root to u and a cost h' that is a lower bound on the cost of the best path from u to any solution node. Two lists of nodes are utilized: OPEN and CLOSED. The parallel A* algorithm which we plan to develop can be described as Hash Distributed A* (HDA*) algorithms, since each processor will execute an almost independent sequential A* on its own OPEN and CLOSED lists.

INTRODUCTION

A* algorithm is an informed search algorithm which is used to locate shortest path from a start node to an end node. It uses heuristic functions h(n) to reduce search space like Manhattan distances or Euclidian distances. The common applications of A* algorithm can be found in motion planning problems, games, path finding in grid maps, and robotics. To apply A* in its sequential, it takes a lot of time in computing, which makes it inconvenient for domain areas that have huge search space. Parallelizing techniques the A* algorithm reduces the timings of implementation, which will be the main goal of this project. Parallelization can be done by splitting path finding problem in sub parts where search space is divided among multiple threads. This allows different numbers of available cores to work simultaneously to find the best path, which will result in less computational time. Our project work here will be beneficial for domains like motion planning problems where finding optimal solution requires a lot of time and memory consuming expansions. In this project, we have introduced a parallel version of the A* algorithm, which is HDA*, that can be implemented to make use of the available number of cores to speedup the execution of algorithm. In most of the cases, our algorithm has showed better performance than the sequential version in terms of execution time, especially in large-sized grids, i.e. for high values of n.

This project report is organized as follows: Theory Section, which will comprise of basic concepts and explainations which we are going to use throughout our computation, some basic information and steps to implement Parallel Processing, A* Algorithm, and Hash Distributed A* has been mentioned right here in this section. In the next section named "Equations" will comprise of main equations and assumptions that we will incorporate in our programming. The "Experiment Environment" will consist of the mention of the machine features on which the computation is done. And finally, the Result section will comprise of the graphical and tabular representations of the observations through rigorous computing on the basis of which we will conclude our objective in the Conclusion and Discussion section.

THEORY

This section includes basics about Parallel processing and the A* algorithm and describes features of parallel computing and how it is implemented by using parallel computers

Parallel Processing

In sequential computing, the program was divided into several instructions; then these instructions were executed in a single processor sequentially. But in parallel computing, the program is divided into parts that can be executed in parallel. Each of these parts also consist of several instructions executed separately on different processors.

Parallel computers can be classified into three types: Single Instruction Multiple Data (SIMD) type, Multiple Instruction Single Data (MISD) type, and Multiple Instruction Multiple Data (MIMD) type. In SIMD, the instruction is executed by all processors operating on different data where the pipeline CPU architecture is used. In MISD, the multiple processors with a single data stream are used and each processor operates on data using separate instructions. In MIMD, each processor deals with different instructions and works with different data.

Parallel programming models can be classified into three categories: Shared memory processor (SMP), distributed memory (cluster), and hybrid model. In SMP, all the processors can share the main memory with a global address space that allow processors to use the data. In clusters, a separate memory for each processor is used instead of having a shared memory with a global address space. And hybrid model integrates both shared and distributed memory.

A Algorithm*

A* Algorithm is one of the best technique used for path finding and graph transversals. A lot of games and web-based maps use this algorithm to find the shortest path efficiently. It is essentially a best first search algorithm that uses a heuristic to guide itself to find the shortest and optimal path between two points.

The algorithm uses two lists: An Open List that contains squares that are being considered to find the shortest path, and a Closed List that contains squares that do not have to be

considered again. A* Algorithm works by maintaining a tree of paths originating at the start node, then it extends those paths one edge at a time, and continues until its termination criterion is satisfied.

We will follow certain steps in order to implement the A* algorithm via programming – Firstly defining a list OPEN. Initially, OPEN will consist solely of a single node, the start node S. If the list is empty, return failure and exit. Then next step will be removing node n with the smallest value of f(n) from OPEN and move it to list CLOSED, if node n is a goal state, return success and exit. After then we will expand the node n, followed by checking the condition that if any successor to n is the goal node, return success and the solution by tracing the path from goal node to S. Otherwise moving on to the next step which will be applying the evaluation function f to each successor node, and checking the condition if the node has not been in either list, add it to OPEN. And lastly, if we will find the OPEN list to be empty after certain loops, then return failure or exit which will signify that we have reached the goal.

*Hash Distributed A**

Hash Distributed is particularly a parallelization technique of A* as straightforward implementation of a hash based work distribution scheme on a shared memory machine. In HDA*, the closed and open lists are implemented as a distributed data structure, where each processor owns a partition of the entire search space. Hence this combination results in a simple algorithm which achieves scalability for both speed and memory usage.

The overall HDA* algorithm begins by the expansion of the start state at the head processor. Each processor P undergoes through a certain loop until an optimal solution is found - Firstly, the processor P checks if a new state has been received in its message queue. If so, P checks for this new state S in P's closed list. If the message queue is empty, then P selects a highest priority state from its local open list and expands it, resulting in newly generated states. For each of the newly generated states Si, a hash key K(Si) is computed based on the state representation, and the generated state is then sent to the processor which owns K(Si). This send is asynchronous and non-blocking. P continues its computation without waiting for a reply from the destination.

EQUATIONS

A* Algorithm extends the path that minimizes the following function-

$$f(n) = g(n) + h(n)$$

Where,

- 'n' is the last node on the path
- g(n) is the cost of the path from start node to node 'n'
- h(n) is a heuristic function that estimates cost of the cheapest path from node 'n' to the goal node

We use he following approximation heuristic to calculate h(n):

Manhattan distance - It is the sum of absolute values of differences in the destination's x and destination's y coordinates and the current cell's x and y coordinates respectively.

h(n) = abs(current_cell.x – destination.y) + abs(current_cell.y – destination.y)

EXPERIMENTS ENVIRONMENT

The experiments were conducted on a virtual machine environment. It had 8 Intel® Core™ i7-9750H CPU @ 2.60 Ghz processors. Each processor had 8 cores. The virtual machine was equipped with 5 GB of ram. The operating system used is Windows 10.

In our experiments, we executed the parallel A* and the sequential A* algorithm on seven problem sizes : 3*3, 50*50, 100*100, 200*200, 500*500, 1000*1000 and 2000*2000.

We ran each testcase 3 times and took the average of these results. We varied the number of threads from 1 to 8(doubling every time) in testing the parallel algorithm.

DISCUSSION

HDA performed better than the sequential A* (except for HDA with 8 thread). HDA with 1, 2 and 4 threads showed significant improvement in computation time. HDA with 4 threads performed the best along all input matrix sizes.

RESULTS

TABLE I. TIME TAKEN TO EXECUTE A* ALGORITHM

Matrix Size (N*N)	Time (in ms)				
	Sequential	**HDA1**	**HDA2**	**HDA4**	**HDA8**
3	13	7.67	0	0	16
50	8.33	8.67	6.67	5.67	57
100	48	47.33	2967	29	1515
200	222	149.67	89.33	112.33	439.33
500	1003	1001.67	629.67	598	3510.33
1000	4610.67	4480	2694	2007.67	16587
2000	14671	14255.67	7987.33	6882	79206.67

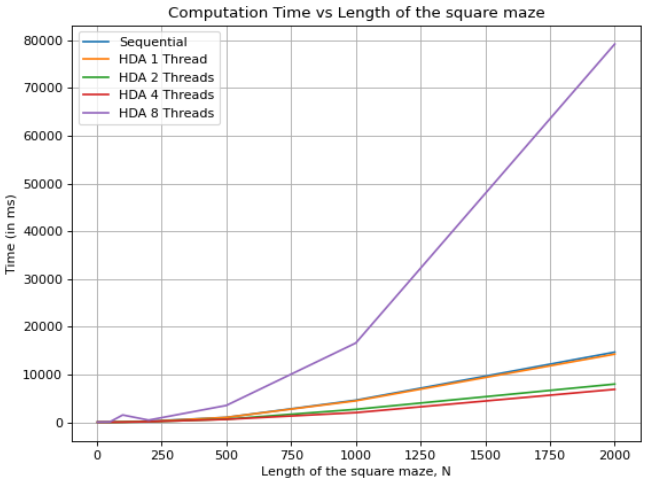

Fig. 1. Time taken to execute A* algorithm

CONCLUSION

We successfully parallelized the A* algorithm using OpenMP. The results were promising and performed better than the sequential version.

REFERENCES

[1] S. S. Z. H. A.-J. , M. B. , L. A.-J. , Mariam Arshad, Arwa Al-Issa, "Parallelizing A* Path Finding Algorithm", int. jour. eng. com. sci, vol. 6, no. 9, Sep. 2017.

[2] Kishimoto, A. et al. "Scalable, Parallel Best-First Search for Optimal Sequential Planning." ICAPS (2009).

[3] Phillips, M. & Likhachev, M. & Koenig, S.. (2014). PA∗SE: Parallel A∗ for slow expansions. Proceedings International Conference on Automated Planning and Scheduling, ICAPS. 2014. 208-216.

[4] R. Awari, "Parallelization of shortest path algorithm using OpenMP and MPI," 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2017, pp. 304-309, doi: 10.1109/I-SMAC.2017.8058360.