Intelligent Systems and Computer Technology
D.J. Hemanth et al. (Eds.)

doi:10.3233/APC200202

Parallelizing Bidirectional A* Algorithm

Sumit Sharma [a], Shashwat Srijan [b], and Vidhya J V [b]
[a,b] *Dept of CSE, SRM Institute of Science and Technology, Chennai*

Abstract. Dijkstra's algorithm is one of the simplest shortest path finding algorithm. A star(A*) algorithm is a variation of the shortest path first Dijkstra's algorithm and is very commonly used in games using heuristics. The idea behind A* is to assign weight to each open node and then use a heuristic to calculate the cost from start to finish. A* uses heuristics and cost functions to find the most appropriate path in games and robotics. Games are needed to be fast and so we can have tradeoffs between speed and accuracy. So instead of accuracy we focus more on speed which is not needed in some of the situations like autonomous vehicles and simulation games. So, the A* algorithm may underestimate the costs but will never overestimate it. Bidirectional A*reduces the computation by calculating the shortest path from the source as well as the destination. A solution may be the General-Purpose Graphics Processing Units. It can be used to enhance the processing and execution speed of Bidirectional A* algorithm by using parallel processing and multiple threads. GPU based path finding may be approximately 45 times as fast as CPU pathfinding mechanism.

Keywords. Bidirectional A*; Parallel; Multiprocessing; GPU

1. Introduction

Game Theory is an important aspect of modern-day universe and computer systems. A major aspect of Artificial Intelligence implemented in games are the pathfinding algorithms. One of the major algorithms used for pathfinding are the A* algorithms. A* is one of the most important and essential algorithms for various pathfinding techniques. A* allows us to implement the pathfinding by identifying the location of the source nodes and the destination nodes and the calculating the heuristic function. Then the node with the maximum value of the heuristic function called as the f-cost is selected and its neighbors are expanded. For each node we identify the neighboring nodes and store them in an open array. Once the F-cost heuristic is calculated for all these nodes we identify the node with the minimum value as this represents the node that is closer to the destination. Now this node is stored in another list so that we can backtrack to this node while identifying the necessary path. The above specified algorithm is repeated for every selected node again and again till we reach a node whose neighboring node is the destination node. Once such a node is found the algorithm stops computing the heuristic function and backtracks to all the nodes which form the path thereby resulting in the shortest path from the source to the destination. The given system only implements Dijkstra's Algorithm using parallel processing and

[1] Sumit Sharma, Computer Science and Engineering, SRM Institute of Science and Technolog, Chennai; Email: sumitsh6917@gmail.com

the Bidirectional A* Search is much faster and efficient in a variety of scenarios as opposed to the simple Dijkstra's algorithm. The major trade-off between the proposed system and the existing system is that in some scenarios the proposed Bi- directional A* search may not be up to the mark as calculations may be excessive. The Bidirectional Search may fail while using Road Maps instead of Grid based systems as road maps do not allow multiple simultaneous executions for the same road point. Also, the transmission time between the GPU and the system memory is much larger than that of the system and the CPU so at times the overhead may slow down the algorithm by large amount if the grid size is extremely large. An important technique which can be used to provide such a parallelizing mechanism is given by Yuichai Zhu and his peer Jin Yang Zen in their paper titled "Massively Parallel A* search on GPU". They describe the use of priority queues to divide the algorithm into smaller parts and sending each part as a unit to the GPU simultaneously for executions. Similarly in bidirectional A* each calculation for heuristic on the source and destination node can be send as a new thread to the GPU for calculations and the result and the required path can then be obtained after the information has been transmitted back to the CPU by the GPU.

2. Related Work

From the literary survey we can observe that several tasks can easily be conducted with the help of Pathfinding algorithms. Path Finding algorithms though may be majorly used in the games industry for finding the shortest path between the source and the destination. Presently the systems which are available are majorly Serially executed pathfinding algorithms and most of the indie game developers use the serial implementations of pathfinding algorithms. The major advancements in the modern times to pathfinding were done by Yichao Zhu and his peer Jianyang Zen by implementing the Dijkstra's algorithm in the GPU thereby reducing the time taken to find the shortest path between the source and the destination. The given system only implements Dijkstra's Algorithm using parallel processing and the Bidirectional A* Search is much faster and efficient in a variety of scenarios as opposed to the simple Dijkstra's algorithm. Dijkstra's search all the possible nodes starting from the source going to the destination and displaying a path if it is available. Bidirectional A* on the other hand starts from the source to the destination and can be broken down into smaller threads which can be implemented simultaneously from both sides to find the shortest path.

3. Proposed Work

In real applications the serialized approach might not always work and we might need faster implementations of the A* algorithm. Also, there might be scenarios like a huge grid where the A* algorithm might simply just fail. So, to avoid such a scenario we can make use of General-Purpose Graphics Processing Unit to make all the calculations accurate as well as faster. Another important thing which the system proposes is that instead of implementing the simple A* algorithm the bidirectional

version of the A* algorithm will be implemented. This means that the heuristic function will be simultaneously applied to the source as well as the destination node simultaneously. The basic F-cost heuristic thus is applied at the source and the destination and a set of neighbour nodes are identified from both the source and the destination. At some point the two set of neighbouring nodes will meet and then we can backtrack using the set of selected nodes to identify all the selected paths and give the appropriate path used as required. Now, since the bidirectional A* will be implemented on the GPGPU so there is also a need to parallelize the algorithm. Since a GPU consists of a large number of cores that can simultaneously execute a large number of threads so we need to break the bidirectional A* into simpler terms.
The Major benefits of the proposed system will be the improved performance of the system while dealing with pathfinding algorithms on large grid-based scenarios as calculations will be executed in a faster manner. Also as opposed to the Dijkstra's algorithm less space will be used up as the system will only search the nodes which have a lesser heuristic value rather than searching the entire list of neighbor nodes. Since heuristic calculations are generally quite exhaustive for the CPU the GPU being functionally better at calculating can easily calculate the heuristics which can be used for searching the nodes. Since the CPU will be available as most of the pathfinding will now be done on the GPU, the CPU will now be available to carry out other important tasks including other calculations and storage operations as and when required without using up resources for pathfinding calculations.

3.1 Equations

For each node the bidirectional A* algorithm calculates the cost f(n) which tells us abut the lowest cost to travel to the nearest neighbor and the node having the overall lowest f cost is selected.
f(n) = g(n) + h(n) where,
g(n) —is the cost of path from start node to current node
h(n) —the estimated heuristic value from current node to the end node
f(n) — the lowest cost neighbor

Manhattan Distance:

```
int dx1= (int)Math.Abs((current.xIndex) - (goal.xIndex));
int dy1=(int)Math.Abs((current.yIndex) - (goal.yIndex));
if (dx1 > dy1)
   return 14*dy1 + 10*(dx1-dy1);
 else
   return 14*dx1 + 10*(dy1-dx1);
```

3.2 Map Representation

World representation in computer games is generally done with the help of grids. We consider the map as a square grid where square represents a coordinate. Map has walkable and non-walkable tiles; the obstacles are represented as non-walkable tiles. The obstacles are unwalkable and the algorithm executes and calculates values for walkable tiles.

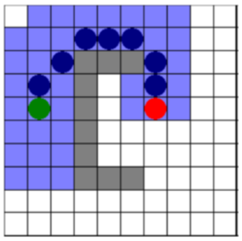
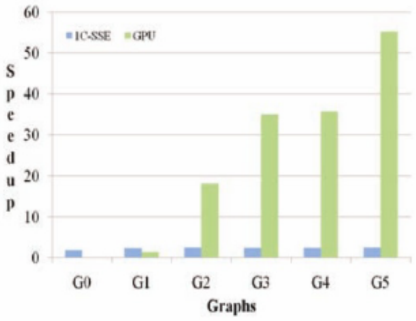


Figure 1: A* algorithm search area in light blue.

3.3 Algorithm used

```
1.    Set static finished = true
2.    Initialise a Variable L
3.    Initialise a List to store thread Ids
4.    Set the start node and the goal Node
5.    Initialise Create_NewThread()
6.    Thread_priority = 1
7.    Initialise Calc_Heurestic()
8.    if (current node is checked by thread) then
if(current.fScores[id]<(int)L&&          current.gScores[id]+brotherThread.F
- brotherThread.Heuristic_cost_estimate(start, current) < (int)L)
9.    Search for least cost neighbour
10.   Lock the thread for synchronization
11.   if selected node executed by brother node
   then stop execution
12.   Retrace Path
```



Graph 1: performance of gpu running cuda A* search using euclidean heurestic compared to cpu simple c++ code

4. Results Discussion

We worked on the implementation of bidirectional A star search on GPU.

5. Conclusion

The A star search method has become prominent in games for discovering shortest distance between any two nodes. The modern day games have several number of agents that tend to move at an identical time in the existence of obstacles. So, it is now very significant to find shortest paths concurrently and in a speedy way. Utilizing GPU's extremely parallel multi-threaded behavior goes with this scenario perfectly. Hence, we have implemented bidirectional A star system in parallel fashion thereby breaking the algorithm and reengineering to make execute it parallelly.

The proposed system will make use of a modified algorithm for A * search and will make use of priority queues to break down the algorithm in its serial form and then using the GPU calculation capabilities to calculate the heuristic function and using it to identify the actual path.

References

[1] Y. Sazaki, H. Satria and M. Syahroyni, "Comparison of A∗ and dynamic pathfinding algorithm with dynamic pathfinding algorithm for NPC on car racing game," 2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA), Lombok, 2017, pp. 1-6.

[2] A. Silva, F. Rocha, A. Santos, G. Ramalho and V. Teichrieb, "GPU Pathfinding Optimization," 2011 Brazilian Symposium on Games and Digital Entertainment, Salvador, 2011, pp. 158-163.

[3] A. Primanita, R. Effendi and W. Hidayat, "Comparison of A∗ and Iterative Deepening A∗ algorithms for non-player character in Role Playing Game," 2017 International Conference on Electrical Engineering and Computer Science (ICECOS), Palembang, 2017, pp. 202-205.

[4] Yoon, Jaeshik, Jaehan Park, and Moonhong Baeg. "GPU-based collision detection for sampling-based motion planning." In 2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), pp. 215-218. IEEE, 2013.

[5] An, Xiaoguang, and Ling Li. "Research on Fast Collision Detection Algorithm Based on CPU Cache Technology." In 2018 International Conference on Virtual Reality and Intelligent Systems (ICVRIS), pp. 219-222. IEEE, 2018.

[6] Yngvi Bjornsson and Kari Halldorsson," Improved Heuristics for Optimal Pathfinding on Game Maps, "in American Association for Artificial Intelligence,2006

[7] Zhou, Yichao, and Jianyang Zeng. "Massively parallel A* search on a GPU." In Twenty-Ninth AAAI Conference on Artificial Intelligence. 2015.

[8] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Hoshang Kolivand, "A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games," International Journal of Computer Games Technology, vol. 2015, Article ID 736138, 11 pages,2015.

[9] Treuille, Adrien, Seth Cooper, and Zoran Popović. "Continuum crowds." In ACM Transactions on Graphics (TOG), vol. 25, no. 3, pp. 1160-1168. ACM, 2006.

[10] R. Mangharam and A. A. Saba, "Anytime Algorithms for GPU Architectures," 2011 IEEE 32nd Real-Time Systems Symposium, Vienna, 2011, pp. 47-56.doi: 10.1109/RTSS.2011.41