Parallelizing Game Algorithms

by

Pratik Kumar Prajapati 18BCE1352
Chandan Kumar 18BCE1020
Varun Patel 18BCE1204

A project report submitted to

Dr.Balasundaram A

SCOPE

In fulfilment of the requirements for the course of

CSE4001 – PARALLEL AND DISTRIBUTED COMPUTING

In

B.TechComputer Science and Engineering

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Vandalur – Kelambakkam Road

Chennai – 600127

OCT 2020

BONAFIDE CERTIFICATE

Certified that this project report entitled "**Parallelizing Game Algorithms"** is a bonafide work of **Pratik Kumar Prajapati - 18BCE1352, Chandan Kumar – 18BCE1020, and Varun Patel – 18BCE1204,** who carried out the Project work under my supervision and guidance for **CSE4001–Parallel and Distributed Computing.**

Dr.Balasundaram A.

Assistant Professor (Sr.)

SCOPE

VIT University, Chennai

Chennai – 600 127.

ABSTRACT

Parallel processing has a major application in various Artificial Intelligence game algorithms, some of them being the Alpha-Beta Pruning Algorithm and the Minimax Algorithm.

In any search algorithm, searching best possible solution from the pool of every possibility known can lead to the construction of the whole state search space, also called the “game tree”, or simply the “state space tree”.

Hence, different game algorithms follow different approaches to effectively search the state space tree and find the best possible solution, and nowadays even more so by parallelizing these algorithms. The Minimax algorithm is one such example of a game algorithm that creates the entire state space tree and goes through its nodes to find the best solution.

A variation of the Minimax algorithm is the Alpha-Beta Pruning algorithm. Instead of searching for the whole state space, the Alpha-Beta Pruning Algorithm removes the unnecessary branches, which helps reduce the search time by a significant amount. Parallelizing Alpha Beta pruning for specific architectures like mesh (CUDA), or shared memory model (OpenMP) helps in the reduction of the computational time of the algorithm.

Keywords – Game Tree, Minimax Algorithm, Alpha-Beta Pruning Algorithm, CUDA, OpenMP

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr.Balasundaram A.,** Assistant Professor (Sr.), SCOPE, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

Pratik Kumar Prajapati

Chandan Kumar

Varun Patel

TABLE OF CONTENTS

INTRODUCTION

With the uprising of Artificial Intelligence, game playing algorithms have gone a step further, and have tried to fasten the game tree searching process by employing various methods for achieving more accurate and faster results.

In any search algorithm, searching best possible solution from the pool of every possibility known can lead to the construction of the whole state search space, also called the “game tree”, or simply the “state space tree”.

Parallel processing has a major application in these Artificial Intelligence game algorithms, some of them being the Alpha-Beta Pruning Algorithm and the Minimax Algorithm.

The Alpha-Beta pruning is a modified version of the minimax algorithm. It can be said that it is an optimization technique for the minimax algorithm. The Alpha-Beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow.

Hence by pruning these nodes, it makes the algorithm fast. The Zobrist Algorithm is another recent game algorithm, which is quite flexible, i.e. it can be mixed up with the Alpha-Beta algorithm to generate a new hybrid algorithm.

Applying parallel programming concepts can in turn improve the performance of algorithms. In this paper, we will try to implement something similar to it, i.e coming up with a parallel, hybrid, faster, and efficient game algorithm.

Hence, different game algorithms follow different approaches to effectively search the state space tree and find the best possible solution, and nowadays even more so by parallelizing these algorithms. The Minimax algorithm is one such example of a game algorithm that creates the entire state space tree and goes through its nodes to find the best solution.

A variation of the Minimax algorithm is the Alpha-Beta Pruning algorithm. Instead of searching for the whole state space, the Alpha-Beta Pruning Algorithm removes the unnecessary branches, which helps reduce the search time by a significant amount. Parallelizing Alpha Beta pruning for specific architectures like mesh or shared memory model helps in the reduction of the computational time of the algorithm.

The prime objective here, as said earlier, lies in parallelizing the Alpha-Beta Pruning Algorithm, and this can be done in a different number of ways by permuting the places where parallel looping is to be done, by controlling the number of threads and so on and so forth.

Our prime objective will be parallelizing the Minimax algorithm, the Alpha-Beta Pruning Algorithm, then also developing a hybrid of the Alpha-Beta Pruning Algorithm and the Zobrist Algorithm, and then comparing all of these on a specific game, under specific constraints, to bring them on a level playing field, and then evaluate their performance.

The Zobrist Algorithm is another game algorithm that works extremely well with the Tic-Tac-Toe game, and we feel that its true potential can be tapped when it is merged with the Alpha-Beta Pruning Algorithm or the Minimax algorithm.

With the uprising of Artificial Intelligence, game playing calculations have gone above and beyond, and have attempted to attach the game tree scanning measure by utilizing different techniques for accomplishing more exact and quicker outcomes. In any inquiry calculation, looking through most ideal arrangement from the pool of each chance known can prompt the development of the entire state search space, likewise called the game tree, or just the state space tree.

Equal preparing has a significant application in these Artificial Intelligence game calculations, some of them being the Alpha-Beta Pruning Algorithm and the Minimax Algorithm. Thus, extraordinary game calculations follow various ways to deal with adequately search the state space tree and locate the most ideal arrangement, and these days much more so by parallelizing these calculations. The Minimax calculation is one such case of a game calculation that makes the whole state space tree and experiences its hubs to locate the best arrangement.

A variety of the Minimax calculation is the Alpha-Beta Pruning calculation. Rather than looking for the entire state space, the Alpha-Beta Pruning Algorithm eliminates the pointless branches, which decreases the hunt time by a critical sum. Parallelizing Alpha Beta pruning for explicit designs like work or shared memory model aides in the decrease of the computational season of the calculation. The prime target here, as said prior, lies in parallelizing the Alpha-Beta Pruning Algorithm, and this should be possible in an alternate number of ways by permuting the spots where equal circling is to be done, by controlling the quantity of strings etc.

Our prime target will be parallelizing the Minimax calculation, the Alpha-Beta Pruning Algorithm, at that point likewise building up a half breed of the Alpha-Beta Pruning Algorithm and the Zobrist Algorithm, and afterward contrasting these on a particular game, under explicit requirements, to welcome them on a level battleground, and afterward assess their presentation. The Zobrist Algorithm is another game calculation that functions admirably with the Tic-Tac-Toe game, and we feel that its actual potential can be tapped when it is converged with the Alpha-Beta Pruning Algorithm or the Minimax calculation.

The practical objective here, as said earlier, lies in parallelizing the Alpha-Beta Pruning Algorithm, and this should be conceivable in a substitute number of ways by permuting the spots where equivalent hovering is to be done, by controlling the amount of strings and so on. Our objective will be parallelizing the Minimax figuring, the Alpha-Beta Pruning Algorithm, by then in like manner developing a crossbreed of the Alpha-Beta Pruning Algorithm and the Zobrist Algorithm, and subsequently differentiating these on a specific game, under unequivocal prerequisites, to invite them on a level landmark, and a while later evaluate their introduction.

LITERATURE REVIEW

One of the first attempts at understanding parallel game algorithms [8] laid the foundation for solid qualitative and research for almost all future works on the subject of parallel game algorithms for years, and even today, many works have referred it to build on a research project in this domain.

Deployment of parallel game algorithms [7, 11, 13] opened up scope for parallelizing Alpha-Beta algorithms and extensive research has been done on implementation and parallelizing of the Alpha-Beta algorithm. Newer developments in games, such as the emergence multiplayer and games with weighted nodes have led to modifications in the Alpha-Beta algorithm to cater to these changes [1, 2, 14].

The Monte-Carlo Tree Search (MCTS) algorithm is widely believed to be the best when it comes to score-bound games [3], but it comes at the cost of missing out important nodes at times. Research work on this drawback [4] has been successful to very little extent.

Speedup improvements in the Alpha-Beta algorithm [6] have remained a topic of interest since ages, and with the recent emergence of GPU's [5, 15], it continues catching the researcher's eye from time-to-time.

A major chunk of our work will be to extrapolate the Zobrist Hashing algorithms, primarily trying to formulate its hybrid algorithm with the Minimax or Alpha-Beta Pruning algorithm. The Zobrist algorithm has shown promising results, in game tree exploration in chess playing with a hybrid with regular Alpha-Beta algorithm [27].

In the recent years, questions have been raised against the superior Alpha-Beta algorithm and its performance [18, 26], with numerous attempts at bettering it, even by a small amount if possible. The Minimax algorithm has also had its fair share of criticism on its efficiency and performance [17, 22]. With the emergence of more and more hybrid algorithms [25], the Minimax and Alpha-Beta algorithms seem to have been overshadowed by these newer and faster hybrid algorithms, leaving them in a state of limbo. The new generation researchers have shown interest in hybridising these senior algorithms, essentially to modify them to their requirements, and also to improve their performance to some extent.

Another classical game tree exploration algorithm, the Monte-Carlo Tree Search (MCTS) has also been under close observation of researchers, primarily to hybridise it with Alpha-Beta algorithm [24]. Although, work to be done in this domain is far more than we can think of, due to lack of experimental results to back-up the theoretical evidence. Attempts at formulating a hybrid of the MCTS and the Zobrist algorithm has also been made [20], with the underlying advantage being similar to that of the cache memory, which is that recently visited states can be visited more often.

In the case of the Zobrist Hashing algorithm, the use of extra memory space, essentially in the form of hash table/transposition table, has been studied under the game of chess [21, 23], with research work leaning towards both, the use and no use of hash table/transposition table for storing the current state of the game.

More recent work has been focussed on implementing the Alpha-Beta algorithm to generic Artificial algorithms [16], which serve as a template for future works to build upon.

Our primary focus, the hybridisation of the Zobrist algorithm with the Minimax algorithm, or hybrid algorithm development, essentially algorithm development, has specific areas where the Zobrist algorithm is in turn used to debug the algorithm [19] during its creation.

EXISTING SYSTEM

A. Minimax Algorithm

The classic Tic-Tac-Toe two-player game has gained significant limelight over the course of parallel game algorithm testing. The simple game provides an amply large game tree for research and analysis of game algorithms. Here, in the Minimax algorithm, the game tree (future move possibility states) are constructed, and given a score based on the possibility of winning, losing, or draw. The algorithm searches for the best possible condition (mostly winning condition, represented by "1") and returns it to the current player.

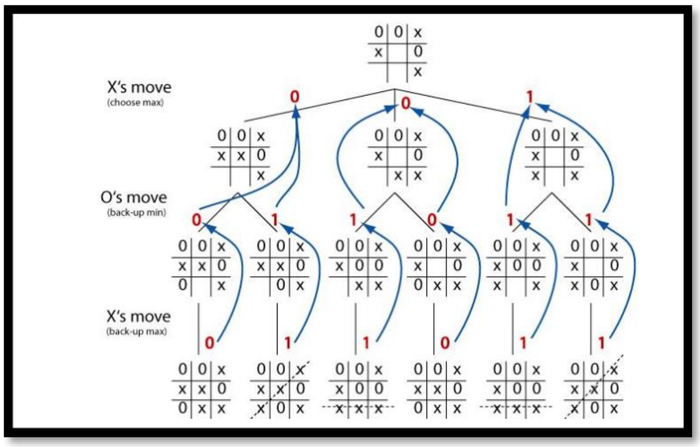


Fig. 1(a). Minimax Algorithm for Tic-Tac-Toe

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.

The Minimax Algorithm can be formulated as shown below:

```
function minimax(board, depth, isMaximizingPlayer):
if current board state is a terminal state:
    return value of the board
if isMaximizingPlayer:
    bestVal = -INFINITY
    for each move in board:
        value = minimax(board, depth+1, false)
    end for
    bestVal = max(bestVal, value)
    return bestVal
else:
    bestVal = +INFINITY
    for each move in board :
        value = minimax(board, depth+1, true)
    end for
    bestVal = min(bestVal, value)
    return bestVal
```

Consider an example:

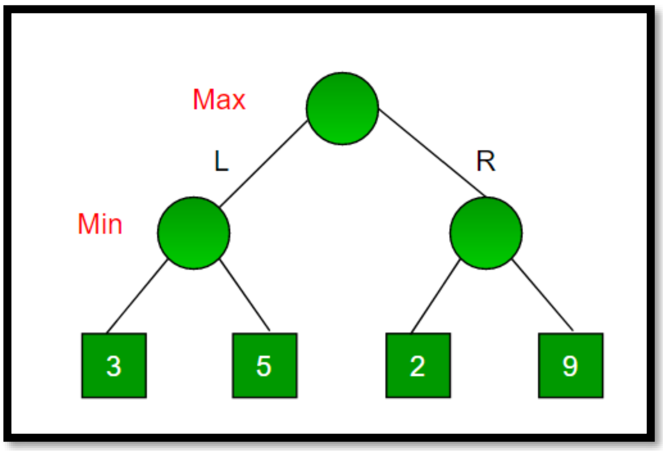


Fig. 1(b). Minimax Algorithm for a game tree

Consider a game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at next level.

The Minimax algorithm here determines which move you should make as a maximizing player, considering that your opponent also plays optimally.

Since this is a backtracking based algorithm, it tries all possible moves, then backtracks and makes a decision.

- Maximizer goes LEFT: It is now the minimizers turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will definitely choose the least among both, that is 3.
- Maximizer goes RIGHT: It is now the minimizers turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values.

Being the maximizer you would choose the larger value that is 3. Hence the optimal move for the maximizer is to go LEFT and the optimal value is 3.

Now the game tree looks like below :

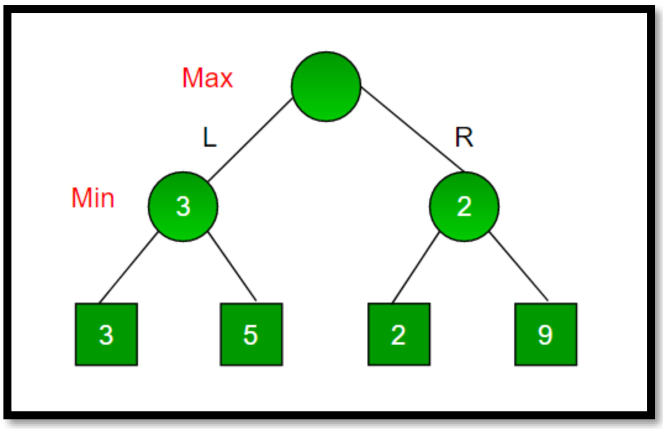


Fig. 1(c). Minimax Algorithm for a game tree

The above tree shows two possible scores when maximizer makes left and right moves.

Here, we can see that even though there is a value of 9 on the right subtree, the minimizer will never pick that. We must always assume that our opponent plays optimally.

B. Alpha-Beta Pruning Algorithm

Alpha-Beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithm. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available. It is called Alpha-Beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.

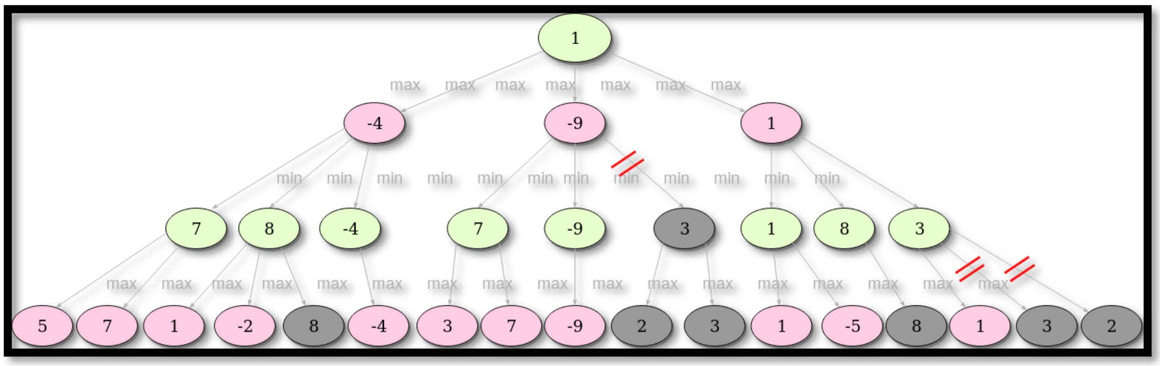


Fig. 2(a). Alpha-Beta Pruning Algorithm

Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm. As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called as Alpha-Beta Algorithm.

Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.

Algorithm for Alpha-Beta algorithm can be seen as:

```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):
    if node is a leaf node :
        return value of the node

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each child node :
            value = minimax(node, depth+1, false, alpha, beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal

    else :
        bestVal = +INFINITY
        for each child node :
            value = minimax(node, depth+1, true, alpha, beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal
```

Let's define the parameters alpha and beta.

Alpha is the best value that the maximizer currently can guarantee at that level or above.

Beta is the best value that the minimizer currently can guarantee at that level or above.

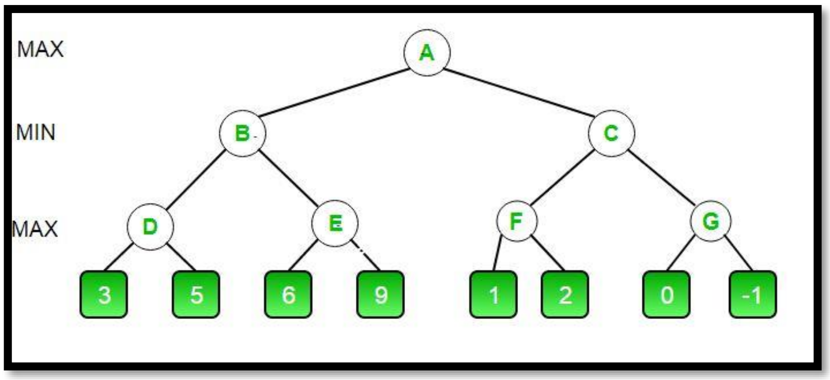


Fig. 2(b). Alpha-Beta Pruning Algorithm for a general game tree

- The initial call starts from A. The value of alpha here is -INFINITY and the value of beta is +INFINITY. These values are passed down to subsequent nodes in the tree. At A the maximizer must choose max of B and C, so A calls B first
- At B it the minimizer must choose min of D and E and hence calls D first.
- At D, it looks at its left child which is a leaf node. This node returns a value of 3. Now the value of alpha at D is max(-INF, 3) which is 3.
- To decide whether its worth looking at its right node or not, it checks the condition beta<=alpha. This is false since beta = +INF and alpha = 3. So it continues the search.
- D now looks at its right child which returns a value of 5.At D, alpha = max(3, 5) which is 5. Now the value of node D is 5
- D returns a value of 5 to B. At B, beta = min(+INF, 5) which is 5. The minimizer is now guaranteed a value of 5 or lesser. B now calls E to see if he can get a lower value than 5.
- At E the values of alpha and beta is not -INF and +INF but instead -INF and 5 respectively, because the value of beta was changed at B and that is what B passed down to E
- Now E looks at its left child which is 6. At E, alpha = max(-INF, 6) which is 6. Here the condition becomes true. beta is 5 and alpha is 6. So beta<=alpha is true. Hence it breaks and E returns 6 to B
- Note how it did not matter what the value of E‘s right child is. It could have been +INF or -INF, it still wouldn’t matter, We never even had to look at it because the minimizer was guaranteed a value of 5 or lesser. So as soon as the maximizer saw the 6 he knew the minimizer would never come this way because he can get a 5 on the left side of B. This way we dint have to look at that 9 and hence saved computation time.
- E returns a value of 6 to B. At B, beta = min(5, 6) which is 5.The value of node B is also 5.

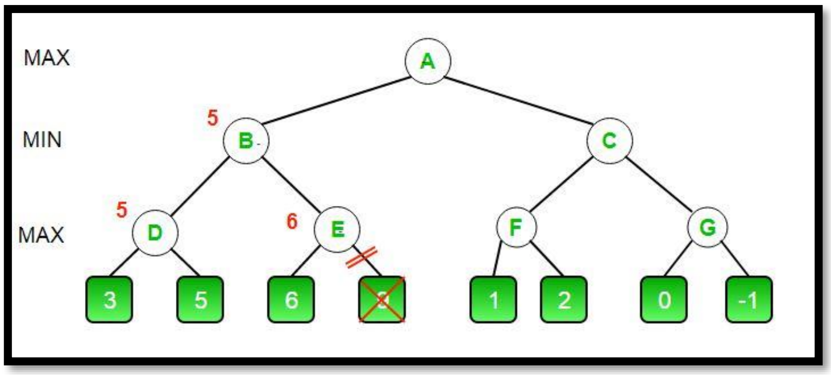


Fig. 2(c). Pruning the unnecessary nodes in the game tree using the

Alpha-Beta Pruning Algorithm

- B returns 5 to A. At A, alpha = max(-INF, 5) which is 5. Now the maximizer is guaranteed a value of 5 or greater. A now calls C to see if it can get a higher value than 5.
- At C, alpha = 5 and beta = +INF. C calls F
- At F, alpha = 5 and beta = +INF. F looks at its left child which is a 1. alpha = max(5, 1) which is still 5.
- F looks at its right child which is a 2. Hence the best value of this node is 2. Alpha still remains 5
- F returns a value of 2 to C. At C, beta = min(+INF, 2). The condition beta <= alpha becomes true as beta = 2 and alpha = 5. So it breaks and it does not even have to compute the entire sub-tree of G.
- The intuition behind this break off is that, at C the minimizer was guaranteed a value of 2 or lesser. But the maximizer was already guaranteed a value of 5 if he choose B. So why would the maximizer ever choose C and get a value less than 2? Again you can see that it did not matter what those last 2 values were. We also saved a lot of computation by skipping a whole sub tree.
- C now returns a value of 2 to A. Therefore the best value at A is max(5, 2) which is a 5.
- Hence the optimal value that the maximizer can get is 5.

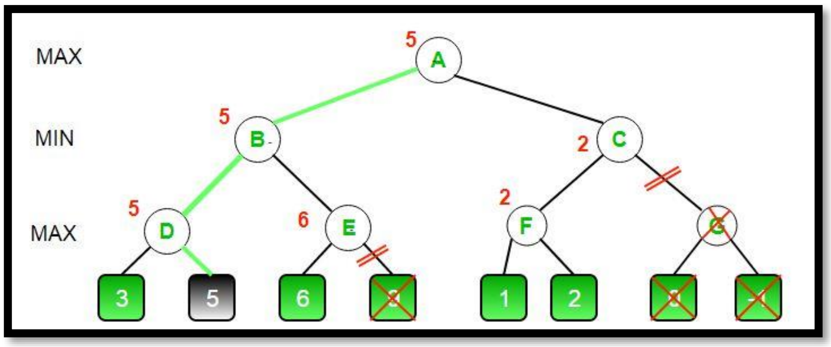


Fig. 2(d). The maximizer returns the best value for A at the current state in the game

PROPOSED SYSTEM

After closely analyzing the Minimax algorithm and the Alpha-Beta algorithm (both, serial and parallel), we noticed that the Alpha-Beta parallel algorithm was by far the best of the lot, in terms of game tree searching speed. The difference between the timings was also very impressive and significantly large.

We then looked into another algorithm, the Zobrist Algorithm, which primarily employs a hash tables (also called a “Transposition Table”) to store the position of the game. Transposition tables basically store the evaluated values of previous board states, so that if they are encountered again we simply retrieve the stored value from the transposition table.

Zobrist Hashing is a hashing function that is widely used in 2 player board games. It is the most common hashing function used in transposition table. Transposition tables basically store the evaluated values of previous board states, so that if they are encountered again we simply retrieve the stored value from the transposition table.

The idea behind Zobrist Hashing is that for a given board state, if there is a piece on a given cell, we use the random number of that piece from the corresponding cell in the table.

If more bits are there in the random number the lesser chance of a hash collision. Therefore 64 bit numbers are commonly used as the standard and it is highly unlikely for a hash collision to occur with such large numbers. The table has to be initialized only once during the programs execution.

Also the reason why Zobrist Hashing is widely used in board games is because when a player makes a move, it is not necessary to recalculate the hash value from scratch. Due to the nature of XOR operation we can simply use few XOR operations to recalculate the hash value.

The Zobrist Hashing algorithm can be understood as:

```
table[#ofBoardCells][#ofPieces]
        // Returns Zobrist hash function for current configuration of board.
        function findhash(board):
        hash = 0;
        for each cell on the board:
                if cell is not empty:
                        piece = board[cell];
                        hash ^= table[cell][piece];
                end if
        end for
        return hash
```

So, for a given board state, if there is a piece on a given cell, we use the random number generated of that piece from the corresponding cell in the transposition table.

The fact at hand here is that transposition table does need extra memory space for storing the hash values of the different board states, but it does however make it faster to search these board state values.

Using this method of transposition tables, for storing the hash values of the different board states, and the already existing Minimax algorithm, we came up with an idea for an alternative algorithm, which can be called a "hybrid" of these two methods (Minimax-Zobrist hybrid algorithm).

The approach we employed here was that when a player plays a game of tic-tac-toe, and for the simplicity of understanding, let us assume the player wins, if this is the case, then and only then the Zobrist Hashing will be used to store the different move-combinations of the game in the transposition table. Hence, when the player plays the game next time, and if the player and the opponent repeat the moves which led to the player's victory in the previous game, the algorithm will retrieve those moves and predict if the player would win this time around or not.

So, essentially when the player is playing a new game, they will find out if the moves they are making will lead them to a winning condition or not.

The main time evaluation of this hybrid game algorithm lies in saving the winning game moves in the transposition table, and then checking if the new game has the similar moves. Hence the time evaluation will be carried out over these two code blocks/snippets.

EXPERIMENTAL SETUP

A. Software Requirements

- System with 8GB RAM

B. Hardware Requirements

- Linux environment (preferred), or,
- Linux Virtual Machine (with minimum 4 processors and OpenMP installation).
- GNU compiler for compiling the programs.

B. Executing Experiment

- Serial programs to be executed using normal gcc compiler on Linux (gcc file_name.c), and
- Parallel programs to be executed using appropriate openmp compiling statements on Linux environment (gcc -fopenmp file_name.c).
- Fixed no. of threads (2) to be used.

EXPERIMENTAL RESULTS AND DISCUSSION

The Alpha-Beta (parallel) algorithm shows a clear advantage for 3x3 tic-tac-toe game over the Minimax (parallel) and Alpha-Beta (serial) algorithms.

However, the Alpha-Beta (serial) algorithm lagged behind the Minimax (parallel) algorithm by a couple hundredth of a second, in average case scenario, showing that the parallel version of the inferior Minimax algorithm is faster in searching the game tree than the serial version of the superior Alpha-Beta algorithm.

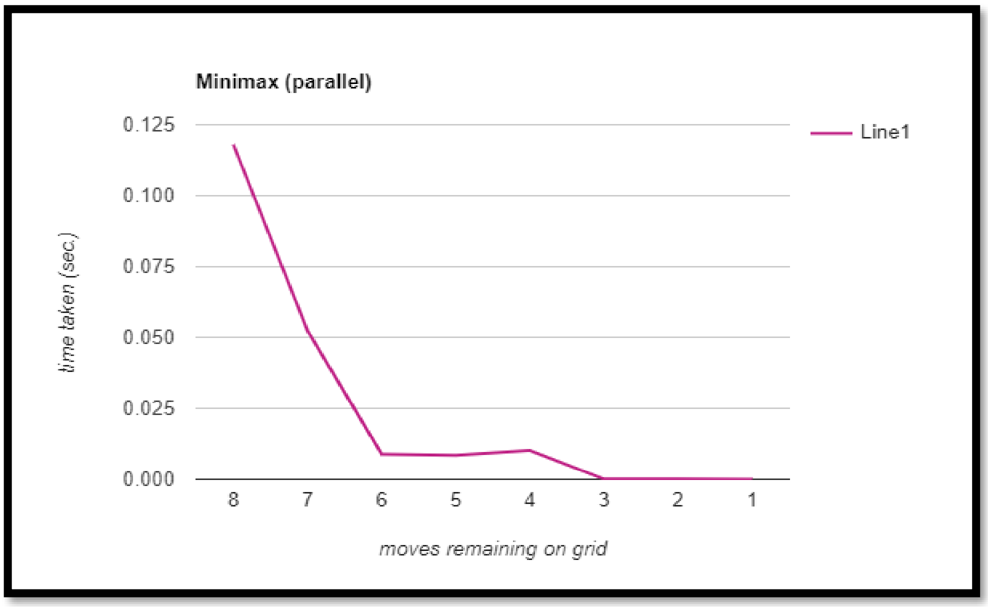
The hybrid algorithm proposed here also shows comparable performance as compared to the fastest of them all, the Alpha-Beta algorithm (parallel).

The results obtained can be tabularized as follows:

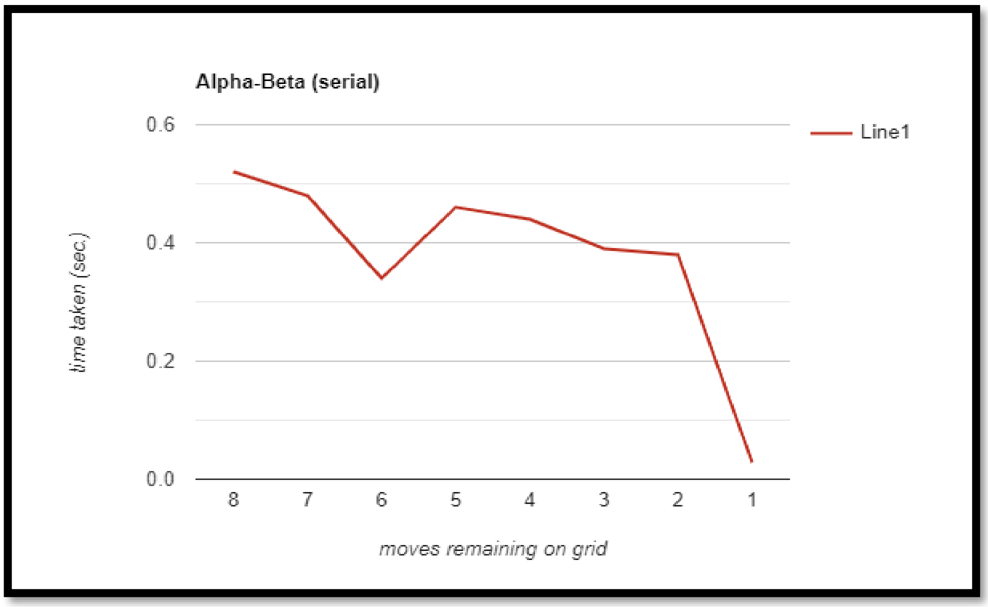
Moves remaining	Alpha-beta parallel	Minimax parallel	Alphabeta serial
8	0.00012	0.117923	0.52
7	0.000133	0.052329	0.48
6	0.00016	0.008885	0.34
5	0.000114	0.008424	0.46
4	0.000086	0.010153	0.44
3	0.000047	0.000095	0.39
2	0.000045	0.000093	0.38
1	0.000032	0.000054	0.029

Table 1. Time taken (in sec.) by different algorithms for different states of tic-tac-toe game

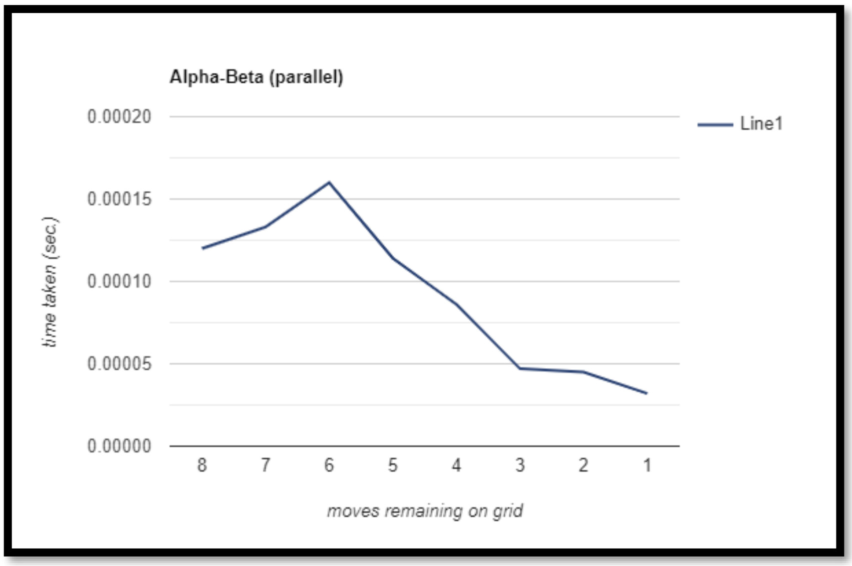
Performance Comparison Graphs



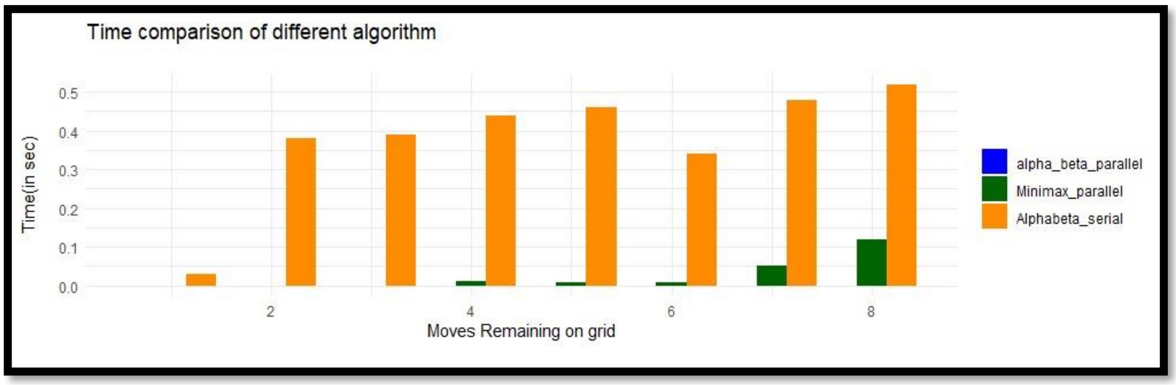
Graph 1. Minimax Algorithm (parallel)



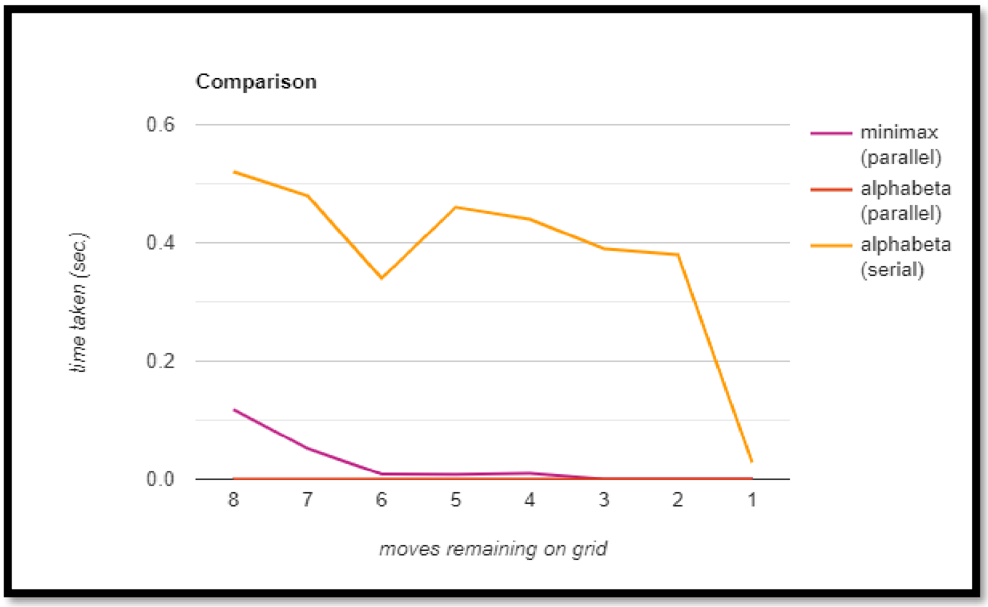
Graph 2. Alpha-Beta Algorithm (serial)



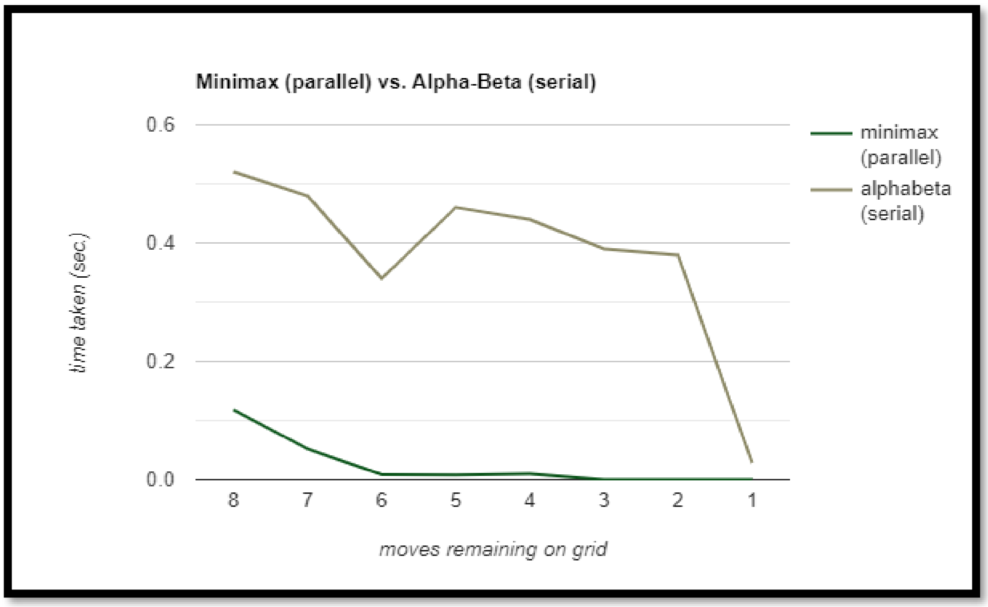
Graph 3. Alpha-Beta Algorithm (parallel)



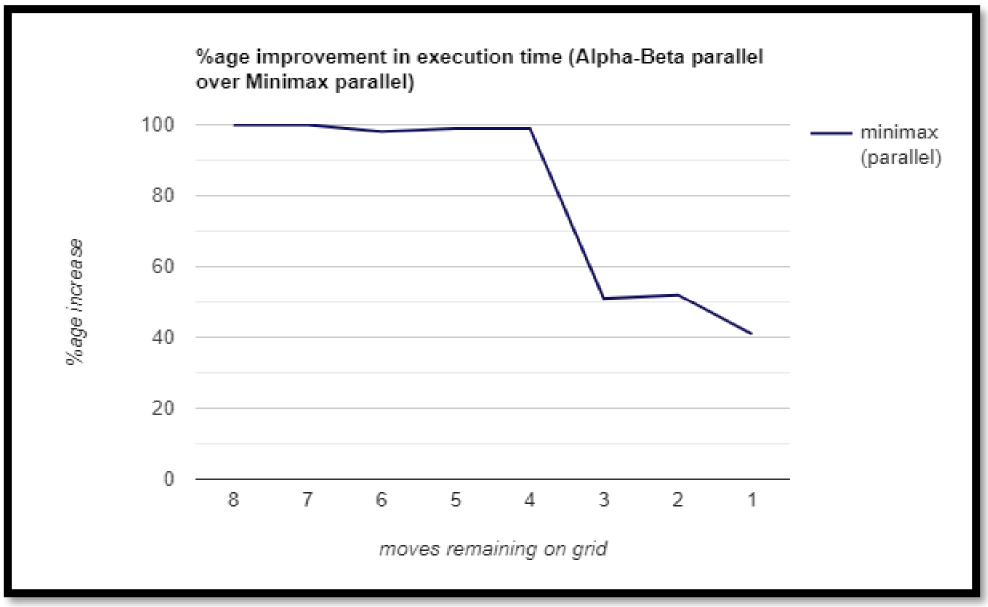
Graph 4. Overall Algorithm Comparison



Graph 5. Overall Algorithm Comparison



Graph 6. Minimax (parallel) vs. Alpha-Beta (serial)



Graph 7. %age improvement in execution time (Alpha-Beta parallel over Minimax parallel)

CONCLUSION AND FUTURE WORK

With the outburst of game programming and parallel computing, the game tree traversal has become an important factor in differentiating the average and faster algorithms.

After this experiment, we can see the impact parallel algorithms have on the searching of large size game trees for selecting the best possible winning move.

Parallelising of the hybrid algorithm can be further improved by instilling more threads and therefore creating more options for parallel task division.

The algorithms employed here are on a specific system with some fixed constraints, so as to gauge them on the same level. Hence, their performance will be interesting to see when these constraints are shuffled around.

We hope that by enlarging the game tree (say for 5x5 tic-tac-toe), we can get some very good results with the algorithms employing parallelism. As the quantity of nodes to be searched will increase, so will the time difference between parallel and serial algorithms increase to search the game tree. With the upheaval of game programming and equal figuring, the game tree crossing has become a significant factor in separating the normal and quicker calculations.

After this examination, we can see the effect equal calculations have on the looking of enormous size game trees for choosing the most ideal winning move. Parallelising of the cross breed calculation can be additionally improved by ingraining more strings and accordingly making more choices for equal assignment division. The calculations utilized here are on a particular framework with some fixed limitations, to check them on a similar level. Thus, their presentation will be intriguing to see when these limitations are rearranged around.

REFERENCES

[1] Saffidine, A., Finnsson, H. and Buro, M., 2012, July. Alpha-Beta Pruning for Games with Simultaneous Moves. In AAAI.

[2] Sturtevant, N.R. and Korf, R.E., 2000. On pruning techniques for multi-player games. AAAI/IAAI, 49, pp.201-207.

[3] Cazenave, T. and Saffidine, A., 2010, September. Score bounded Monte-Carlo tree search. In International Conference on Computers and Games (pp. 93-104). Springer, Berlin, Heidelberg.

[4] Sturtevant, N.R., 2005, July. Leaf-value tables for pruning non-zero-sum games. In International Joint Conference on Artificial Intelligence (Vol. 19, p. 317). LAWRENCE ERLBAUM ASSOCIATES LTD.

[5] Strnad, D. and Guid, N., 2011. Parallel alpha-beta algorithm on the GPU. Journal of computing and information technology, 19(4), pp.269-274.

[6] Hewett, R. and Ganesan, K., 1992, January. Consistent linear speedup in parallel alpha-beta search. In 1992 Fourth International Conference on Computing and Information (pp. 237-238). IEEE Computer Society.

[7] Akl, S.G., Barnard, D.T. and Doran, R.J., 1982. Design, analysis, and implementation of a parallel tree search algorithm. IEEE Transactions on Pattern Analysis and Machine Intelligence, (2), pp.192-203.

[8] Knuth, D.E. and Moore, R.W., 1975. An analysis of alpha-beta pruning. Artificial intelligence, 6(4), pp.293-326.

[9] Ballard, B.W., 1983. The*-minimax search procedure for trees containing chance nodes. Artificial Intelligence, 21(3), pp.327-350.

[10] Wahib, M., Munawar, A., Munetomo, M. and Akama, K., 2011, June. Optimization of parallel genetic algorithms for nVidia GPUs. In 2011 IEEE Congress of Evolutionary Computation (CEC) (pp. 803-811). IEEE.

[11] Althofer, I., 1993. A parallel game tree search algorithm with a linear speedup. Journal of Algorithms, 15(2), pp.175-198.

[12] Chen, Y., Tan, Y., Zhang, Y. and Dulong, C., 2006, June. Performance analysis of two parallel game-tree search applications. In International Workshop on Applied Parallel Computing (pp. 1105-1114). Springer, Berlin, Heidelberg.

[13] Hyatt, R.M., 1997. The dynamic tree-splitting parallel search algorithm. ICGA Journal, 20(1), pp.3-19.

[14] Korf, R.E., 1991. Multi-player alpha-beta pruning. Artificial Intelligence, 48(1), pp.99-111.

[15] Rocki, K. and Suda, R., 2009, September. Parallel minimax tree searching on GPU. In International Conference on Parallel Processing and Applied Mathematics (pp. 449-456). Springer, Berlin, Heidelberg.

[16] Huang, B., 2015, January. Pruning Game Tree by Rollouts. In AAAI (pp. 1165-1173).

[17] Korf, R.E. and Chickering, D.M., 1996. Best-first minimax search. Artificial intelligence, 84(1-2), pp.299-337.

[18] Stockman, G.C., 1979. A minimax algorithm better than alpha-beta?. Artificial Intelligence, 12(2), pp.179-196.

[19] Cheng, L., Yajie, W., Fei, L. and Xiaoyan, W., 2012, May. Debugging computer game program using Zobrist hashing. In 2012 24th Chinese Control and Decision Conference (CCDC) (pp. 1514-1517). IEEE.

[20] Mason, D.R., Hudson, T.S. and Sutton, A.P., 2005. Fast recall of state-history in kinetic Monte Carlo simulations utilizing the Zobrist key. Computer physics communications, 165(1), pp.37-48.

[21] Vučković, V., 2016. An algorithm for the detection of move repetition without the use of hash-keys. Yugoslav Journal of Operations Research, 17(2).

[22] Campbell, M.S. and Marsland, T.A., 1983. A comparison of minimax tree search algorithms. Artificial Intelligence, 20(4), pp.347-367.

[23] Huizhan, L., Chenjun, X., Hongye, L. and Jiao, W., 2012, May. Hash table in Chinese chess. In 2012 24th Chinese Control and Decision Conference (CCDC) (pp. 3286-3291). IEEE.

[24] Baier, H., 2016. A Rollout-Based Search Algorithm Unifying MCTS and Alpha-Beta. In Computer Games (pp. 57-70). Springer, Cham.

[25] Preux, P. and Talbi, E.G., 1999. Towards hybrid evolutionary algorithms. International transactions in operational research, 6(6), pp.557-570.

[26] Darwish, N.M., 1983. A quantitative analysis of the alpha-beta pruning algorithm. Artificial intelligence, 21(4), pp.405-433.

[27] Liu, Y., Zang, H. and Fu, P., 2012. A Hybrid Game Tree Search Algorithm for Playing Chess. Journal of Computational Information Systems, 14, pp.5803-5811.

APPENDIX - 1

Codes (hosted on GitHub):

https://github.com/varunpatel-git/parallelizing-game-algorithms/tree/main/codes

APPENDIX - 2

Output screenshots:

https://github.com/varunpatel-git/parallelizing-game-algorithms/tree/main/screenshots