International Journal of Advances in Electronics and Computer Science, ISSN: 2393-2835 Volume-4, Issue-2, Feb.-2017
http://iraj.in

PARALLELIZATION OF ALPHA-BETA PRUNING ALGORITHM FOR ENHANCING THE TWO PLAYER GAMES

[1]AKANKSHA KUMARI, [2]SHREYA SINGH, [3]SHAILJA DALMIA, [4]GEETHA V

[1,2,3,4]Dept. of Information Technology, National Institute of Technology Karnataka, Surathkal
E-mail: [1]akankshakmr174@gmail.com, [2]shreya.singh161195@gmail.com,
[3]shailjadalmia11@gmail.com, [4]geethav@nitk.ac.in

Abstract- The game application which requires extensive searching requires an effective and faster technique for the same. The speed of game playing also depends on the rate at which the decision making process gets executed. Alpha-Beta pruning is one of the most powerful and fundamental MinMax search improvements. The pruning helps to reduce the number of search, which further contributes to speed of the decision making at every instance of game playing. In this paper, we would like to further improve on execution time, by parallelizing the Alpha-Beta algorithm. To analyze the performance, we have considered "Stacked matrix games" for two players such as tic-tac-toe, checker board and chess. The result shows an average speed-up of 3.03 due to parallelization of Alpha-Beta pruning using Open MP.

Index terms- Alpha-Beta Pruning, MinMax Search, Game playing, Parallelization, OpenMP.

I. INTRODUCTION

The two player machine playing game needs to identify possible alternatives for various moves and then select the best move. The selection involves searching in a large tree which is normally a time consuming process. To improve on speed of game playing, the tree searching can be pruned with Alpha-Beta pruning, which is a searching algorithm that seeks to decrease the number of nodes that are evaluated by the MinMax algorithm in its search tree.
The two player machine playing games such as tic-tac-toe, checkers and chess are simple games, but these games use large trees for possible alternative solution finding [1]. The main aim is to improve the process by parallel execution of the algorithm. Since two player machine playing games are performed on single system, the parallelization of algorithm can be done at thread level. OpenMP provides a way for thread parallelism, with shared memory concept. Nowadays, as normal standalone systems also come with dual core, quad core etc., it is beneficial to run the algorithm parallel using threads. The OpenMP provides directives to compiler, for identifying the portion of code or algorithm, which can be run in parallel. For the purpose of analyzing the parallel algorithm with sequential algorithm, we have considered three games: tic-tac-toe, checkers and chess. The performance analysis of these games with respect to sequential and parallel execution of Alpha-Beta pruning technique shows better improvement with parallel execution.
The paper is structured as follows: Section II provides details on literature survey, section III explains parallelization of Alpha-Beta pruning algorithm and section IV provides details about experimental results and discussion followed by conclusion and reference.

II. LITERATURE SURVEY

A. Background

Alpha-Beta pruning [2, 3] is a search algorithm that is aimed to reduce the number of nodes that are calculated by the MinMax algorithm in its search tree. The advantage of this algorithm lies in the fact that the branches of the search tree, which do not affect the final result of the move, can be eliminated in the evaluation process. By this method, the search could be limited to the subtree which is actually useful and a more depth-focused search could be performed. It applies to sequential zero-sum two-player games with perfect information such as Chess or Checkers. The Alpha-Beta algorithm retains upper and lower bounds of values to analyse whether branches can be cut. This type of pruning can lead to considerable search reductions — essentially doubling the search depth over the original MinMax algorithm when given the same search time [4].

The MinMax estimation of a diversion tree is computed taking into account the presumption that the two players, called Max and Min, will pick their best course of action such that when the ball is in Max's court he will choose the activity that expands his addition while Min will choose the one that minimizes it on his turn. MinMax qualities are engendered from the leaves of the diversion tree to its root utilizing this principle. Alpha-Beta uses the MinMax esteem to prune a subtree when it has verification that a move won't influence the choice at the root hub. This happens when a halfway pursuit of the subtree uncovers that the rival has the chance to bring down an officially settled MinMax esteem, went down from an alternate subtree.

B. Related Works

Alpha-Beta pruning is one of the best pruning techniques for reducing the search operations in tree [2, 3]. After its discovery, sound Alpha-Beta pruning has been extended to other game types and game tree search algorithms [5, 6, 7]. The Monte Carlo Tree

Search which is a type of simulation based best first search algorithm is extended to allow for Alpha-Beta style pruning in [8]. Recently, parallelism has been introduced in Alpha-Beta pruning to further improve its efficiency; in [9] a new parallel Alpha-Beta pruning algorithm has been presented that applies a prioritizing scheme. The algorithm was implemented on sequent symmetry shared memory multiprocessor system in [9]. Principal variation splitting (PVSplit) [10], is the consequences of some of the first attempts at parallelizing Alpha-Beta which states the rule of searching the first branch at a PV node before the search of remaining branches may begin. A more recent concept is the Young Brothers Wait Concept [11] wherein the first sibling node is searched before spawning the remaining siblings in parallel. Dynamic Tree Splitting (DTS) [12, 13, 14] is yet another concept which uses a peer-to-peer approach rather than a master-slave approach as in YBWC. In this, while multiple processors may collaborate on a node, the processor that finishes the last search is responsible for returning the node's evaluation to its parent. In [14], a new tree splitting method based on neural networks is introduced which is found to outperform YBWC and DTS on certain types of trees. Parallel implementation of the Alpha-Beta pruning algorithm on graphics processing unit (GPU) has been done in [15] where they compare the speed of parallel player with that of a standard serial one using the game of reverse with boards of different sizes. It was observed that, with larger boards, substantial speedups can be achieved on the GPU as compared to the smaller boards.

III. ALPHA-BETA PRUNING ALGORITHM

A. Sequential Alpha-Beta Pruning Algorithm

The Alpha-Beta pruning algorithm emerged as an improvement over the usual MinMax algorithm which is implemented in the Artificial Intelligence games. Both of these algorithms find out the best available move to the player and will return the exact utility regarded with that move, but the key point to be noted here is the execution time. Alpha-Beta pruning algorithms are comparatively faster than MinMax algorithms as they cut down the branches by not exploring them. The reason being, that the values/utilities calculated from these branches would not affect the final result. Since, time is not spent on exploring the other branches, these algorithms effectively cut down the execution time.

Consider a smaller version of a game tree in Figure 1. The nodes corresponding to the maximizing player are shown as squares whereas; nodes corresponding to the minimizing player are shown by circles. In the second last layer, where it's the turn of the minimizing player, the player prunes away the node 5, as the player will only explore that subtree if the parent of the subtree has a value lesser than or equal to 4. Similarly in the second layer, the minimizing player prunes away the entire subtree rooted with 8, as there's no point in exploring that subtree as it already has got a low value in 5.

The Algorithm 1 shows the sequential algorithm for Alpha-Beta pruning. The algorithm has two variables, alpha and beta, which showcase the highest score that the maximizing player is ascertained of and the lowest score that the minimizing player is ascertained of respectively. Initially, both players start with their lowest possible score. It might occur that while selecting a specific branch of a certain node the current lowest score that the minimizing player is ascertained of becomes less than the current higher score that the maximizing player is ascertained of (beta<=alpha). In this scenario, the parent node must not select this node, as it will result in making the score of the parent node worse. Hence, the other branches of the node do nothave to be explored and branches of such subtree are eliminated.

In the implementation, the board, which is a structure, stores the board variables and a boolean variable called turn, specifies the turn of each side. Structures of pieces, each move possible on the board, move scores and properties of each piece on the board are declared. The turn of the player is defined along with the vector of moves which are legal on that board. The Alpha-Beta algorithm makes the game tree in such a fashion that the maximizers and the minimizers are called alternatively and the value of each legal move is stored in the leaf nodes of the game tree. The minimizing function selects the minimum value and maximizing function selects the maximum value amongst the moves with respect to alpha and beta. The game tree is evaluated and best possible move is determined and rests of the capturing moves are generated by a method. The capturing moves generated in the end matches with the legal moves and each capturing move will reduce the number of moves taken by the opposite player for winning strategy.

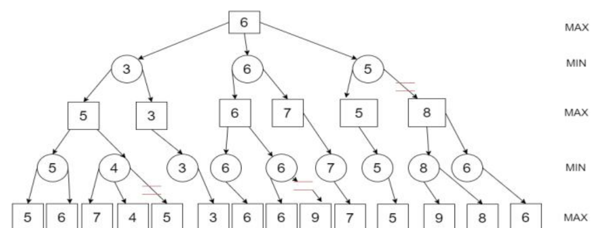

Figure 1: Example tree for Alpha-Beta Pruning

Algorithm1: Sequential Alpha-Beta Pruning

1.function alpbeta(node, depth, α, β, maxPlayer):
2. **if** height = 0 or node is a terminal node :
3. return the heuristic value of node
4. **end if**
5. **if** maxPlayer:
6. v := -∞ ;

```
7.    for each child of node:
8.      v := max(v,alpbeta(child,height - 1, α, β,false))
9.          α := max(α, v);
10.          if β ≤ α:
11.             break; (* β cut-off *)
12.          end if
13.      return v;
14.   end for
15. else
16.     v := ∞;
17.     for each child of node:
18.       v := min(v,alpbeta(child, height - 1,α, β,true))
19.          β := min(β, v);
20.          if β ≤ α:
21.             break; (* α cut-off *)
22.          end if
23.      return v;
24.     end for
25. end if
```

B. Parallel of Alpha-Beta Pruning Algorithm

In the process of parallelization, majorly the evaluation of the game tree has been parallelized. Essentially, each branch of the game tree can be evaluated in parallel. Thus, the alpha and beta values are propagated at once to each of the node at the first level and the minimizing and corresponding maximizing moves are evaluated simultaneously in all the branches. The final evaluation result of all the branches is collected by the main thread and the best move is computed with respect to the alpha and beta values of each branch. It should be noted that the pruning of branches is independent of the other branches, so the evaluation of braches comes under the category of embarrassingly parallel algorithms.

Algorithm 2 : Parallel Alpha-Beta pruning with OpenMP directives.

```
1.function alpbeta(node, height, α, β, maxPlayer):
2.   if height = 0 or node is a terminal node:
3.      return the heuristic value of node
4.   end if
5.   if maxPlayer:
6.       v := -∞;
//Begin parallel loop
7.  #pragma omp parallel
8.for each child of node:
9.   v := max(v,alpbeta(child,height - 1,α, β,false))
10.   α := max(α, v);
11.          if β ≤ α:
12.            break; (* β cut-off *)
13.       end if
14.       //End parallel loop
15.       return v;
16. end for
17. else
18.      v := ∞;
 //Begin parallel loop
19.      #pragma omp parallel
20.      for each child of node:
21.        v := min(v,alpbeta(child,height -1,α,β,true))
22.        β := min(β, v);
23.          if β ≤ α:
24.             break; (* α cut-off *)
25.          end if
26       //End parallel loop
27.       return v;
28.     end for
29.   end if
```

The partitioning of the tree for parallel computation is done on a per-child basis. Each child of the MinMax tree evaluates the minimum and maximum moves together without depending on the results obtained from other children and thus does all the move computation relatively fast. After this computation is done, the main thread collects the final result and returns accordingly as Algorithm 2 suggests.

IV. WORK DONE AND RESULT ANALYSIS

The sequential Alpha-Beta Pruning algorithm is implemented in C++. The parallelization of the algorithm is performed using OpenMP. OpenMP is a usage of multithreading, a strategy for parallelizing whereby an expert string forks a predetermined number of slave strings and the framework separates an errand among them. The strings then run simultaneously, with the runtime environment assigning strings to distinctive processors.

The segment of code that is intended to keep running in parallel is stamped likewise, with a preprocessor order that will bring about the strings to shape before the segment is executed. Each string has an id appended to it which can be acquired utilizing a method *omp_get_thread_num()*. The string id is a whole number, and the master string has an id of 0. After the execution of the parallelized code, the strings join over into the master string, which proceeds with forward to the end of the system.

As a matter of course, every string executes the parallelized segment of code freely. Work-sharing develops can be utilized to partition an assignment among the strings so that every string executes it's apportioned a portion of the code. Both undertaking parallelism and information parallelism can be accomplished utilizing OpenMP along these lines.

The runtime environment assigns strings to processors contingent upon use, machine burden and different variables. The runtime environment can dole out the quantity of strings in view of environment variables, or the code can do as such utilizing capacities. The OpenMP capacities are incorporated into a header document named omp.h in C/C++. In performance of the game playing is analyzed with respect to speedup. The speedup is calculated as shown in equation 1.

$$\text{Speedup} = \frac{\text{Time for serial execution}}{\text{Time for parallel execution}} \quad (1)$$

The results are analyzed for three different games: Tic-tac-toe, Checker and Chess.

A. Result and Analysis of Parallel Alpha-Beta Pruning Algorithm in Tic-Tac-Toe

Tic-Tac-Toe seems to be dumb, but it actually requires one to look ahead the opponent's move to ensure winning. However, one needs to consider opponent's move for their next move. For finding better solution for next move, a heuristic evaluation function is used for current board position which involves a deeper search.The serial implementation of Tic-Tac-Toe is done in C++. In the implementation, the board, a structure with 2D array of positions and number of empty positions on board at any time is declared. The number_of_rows * number_of_columns – number_of _empty_positions gives the number of moves taken by both players. The board is initialized with empty spaces and position (1, 1) is taken by 1st player and henceforth all possible moves are calculated for alternate player and he is allowed to move based on heuristic value. The sum is obtained by computing the scores of all winning strategies of the player.

The principle of MinMax is to minimize the maximum possible loss. The algorithm evaluates the leaf nodes using the heuristic evaluation function, obtaining the best possible move for computer. The computer (or the maximizing player) chooses the maximum value from the child nodes while minimizing player chooses minimum value. The algorithm continues calculating the highest and lowest values of the child until it arrives at the root node, where it selects the move with the highest value. This is the move that the player should make in order to minimize the maximum possible loss.

Algorithm 3 shows the parallel implementation of the algorithm for Tic-Tac-Toe. The parallel implementation of the same divides the search tree and spreads it out across several processors. The search tree is separated at the top level and has each processor investigate a single move. However, in Alpha-Beta cutoffs, each branch of the tree is evaluated independent of other branches of search tree. The parallel computational model of tic-tac-toe is based on combination of "master-slave" and "asynchronous iterations" programming paradigms. The master process is accountable for the below activities:

1)Distributes particular positions of the initial mark on the board for evaluation to the slave processes.

2)Gathers the best cost function values and the best moves determined by each of the slave processors at a given level of the game tree.

3)Determines the best cost function value obtained by all slave processes.

4)Broadcasts the best cost move to all slave processes.

5)Prints the results after examining the whole game tree. The slave processes are responsible for the following activities:

i)Receives the specific move to be evaluated at a given level of the game tree.

ii)Computes the cost function values for all possible moves of the other player according to Alpha-Beta algorithm.

iii)Sends the value of the best cost function and the relevant move to the master process.

iv)Receives the move to be made at the given level (broadcast by the master process).

Alpha-beta algorithm is not a parallelizable algorithm by its very nature, which is a property MixMax exhibits. Particularly, if we try to parallelize the loop, we will be inept to spread the new alpha and beta values to each iteration. So, our general strategy is the following.

6)Evaluate x of the moves sequentially to get reasonable alpha/beta values that will enable us to cut out large parts of the tree.

7)Evaluate the remaining moves in parallel. This means we will evaluate some unnecessary moves, but, in practice, it's worth it.

Algorithm 3: Implementation of parallel Alpha-Beta pruning for the game Tic-Tac-Toe

```
1.PERCENTAGE_SEQUENTIAL=0.5
2.function alphabeta(position p,alpha,beta):
3.  if p is leaf:
4.     return p.evaluate();
5.  end if
6.  moves=p.getMoves();
7.   i=0;
8.  foreach i
<PERCETAGE_SEQUENTIAL*moves.length()
9.        p.applyMove(moves[i]);
10.       value := −alphabeta(p, −beta, −alpha);
11.       p.undoMove();
12.        if value > alpha:
13.            alpha = value;
14.        end if
15.        if alpha >= beta:
16.            return alpha;
17.        end if
18.  end for
19.   i = PERCENTAGE_SEQUENTIAL *
moves.length;
20.   #pragma omp parallel for
21.   for each i < moves.length :
22.       p = p.copy();
23.       value = −alphabeta(p, −beta, −alpha);
24.       if value > alpha:
25.       #pragma omp single
26.           alpha = value ;
27.       end if
28.       if alpha >= beta:
```

```
29.       #pragma omp single
30.           return alpha;
31.       end  if
32.    end for
33.   return alpha;
```

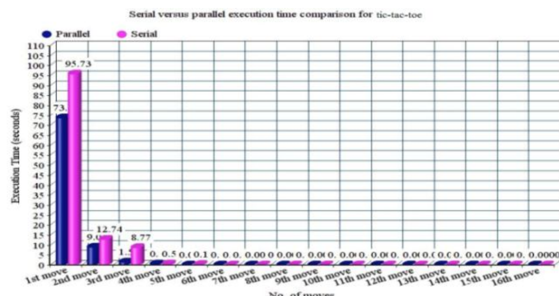

Figure 2: Serial vs. Parallel execution of Tic-Tac-Toe

The computational time for the parallel and the serial implementations of the Alpha-Beta algorithm for tic-tac-toe is shown in Figure 2. The time taken for parallel execution of the evaluate method is considerably lesser than the time taken for the serial execution of the same. In Figure 2, it can be observed that, as the Nth move increases, the serial time execution decreases more rapidly as compared to the parallel time execution. Execution time for the last few moves for parallel implementation are almost same as serial since most of the slaves are idle and not many subtrees need to be evaluated in parallel.Comparison of the results shows decrease of the communication overhead in the case of multithreaded processes.The clear depiction of parallelization of the Alpha-Beta pruning for tic-tac-toe suggests that it is best suited for four numbers of the threads. As, the number of threads is increased, the time taken to distribute the work among slaves increases and after small computation of subtree, the slave remains idle. Hence, for a 4 * 4 board size, it is well suited to implement parallelism for four threads. In almost all cases, serial takes more time than parallel as it could be depicted from Figure 2 that it is almost overlapping with parallel execution with 16 threads.
The average speedup achieved by using parallel execution is 1.8655.

B. Result and Analysis of Parallel Alpha-Beta Pruning Algorithm in Checkers

Algorithm 4 shows the parallel implementation of the algorithm for checkers board game. The ReachableBoards returns the list of all board positions that could be reached in one move by the player. In this implementation, we have the board initialization (initBrd) which goes into the evaluation and returns the alpha/beta value of it, which places the move that renders this value in <best>. The major logic of this game is evaluated by the game tree using the Alpha-Beta pruning algorithm.

This algorithm makes the game tree in such a fashion that for each ReachableBoards(b), the value of that board is calculated. Alpha cut-off and beta cut-off is done simultaneously (as we are parallelizing that loop) depending upon the player comparison with the PLAYER_MAX. In each case of player comparison, the ReachableBoards value is compared with alpha/beta and alpha is compared with beta to return the final best player value (best move that player can make. After evaluating the game tree and determining the best possible move, the size of the board is varied, as shown in the Figure 3, to test for parallelization in different scenarios.

Algorithm 4: Implementation of parallel Alpha-Beta pruning for Checker

```
// Return an  integer that is larger for boards that are better for player MAX
1.  int BoardEval(Board b);
/* Given board b, returns a list of all the board positions that could be reached in one move by p.
 */
2. List<Board> ReachableBoards(Board b, Player p);
/* Given board initBrd, return the alphabeta value of it, and place the move that
 * renders this value in <best>.
 */
3. int BoardValue(Board initBrd, Player p, int ply, int alpha, int beta, Board &best):
4.   if ply >= MAX_PLY:              // At bottom of search tree
5.         return BoardEval(initBrd);
6.   end if
   // Else we've got to look at the descendants
7.   List<Board> boards = ReachableBoards(initBrd);
8.   best = boards[0];
9.   #pragma omp parallel[….]        //Start of parallel loop
//parallelizing each move so that the computation happens        //simultaneously
10.   for each b in boards :
11.         int val = BoardValue(b, !p, ply+1, alpha, beta);
12.         if  p == PLAYER_MAX:
13.         if  val > alpha:
14.                 alpha = val; best = b;
15.           end if
16.           if alpha >= beta:
17.                 return alpha;
18.           end if              //Alpha-cutoff
19.         else                              // MIN's turn
20.         if  val < beta:
21.                 beta = val; best = b;
22.           end if
23.         if  alpha >= beta:
24.                 return beta;
25.           end if
26.       end if                          // Beta-cutoff
27.   // End of parallel loop
```

```
28.   return (p == PLAYER_MAX ? alpha : beta);
29.   end for
```

Figure 3: Serial vs. Parallel execution of Checkers Board

The Figure 3 shows the graph of serial and parallel execution of checkers game with Alpha-Beta pruning technique. The results are analyzed for time elapsed for the evaluation of the best move in the game tree by varying the board size. The Figure 3 shows that the time taken for parallel execution of the evaluate method is considerably lesser than the time taken for the serial execution of the same. As the game board size increases, the gradient of serial time execution rises rapidly as compared to the parallel time execution.

The average speedup achieved by using parallel execution is 2.97068216.

C. *Result and Analysis of Parallel Alpha-Beta Pruning Algorithm in Chess*

The implementation of serial chess has been done in C++ and has been majorly divided into three sections namely: chess board, chess pieces and the algorithms. The chess-board part first deals with checking the validity of the moves and then orienting them upon the chess board by directing them. Associating a distinct character, value and utility to each chess piece is handled in the chess-pieces part wherein a set of classes denote proper identity to each of the chess piece. Lastly, the algorithms' section deals with building the game tree similar to Figure 1 by using recursion and consists of the Alpha-Beta pruning algorithms. It is followed by the artificial intelligence part of making the system play the game in accordance with the moves of the user.

The Alpha-Beta pruning algorithm used in the chess program maintains two variables: alpha and beta wherein alpha stands for the lower bound and beta stand for the upper bound of the utility achieved by the user whose turn is there. The algorithm recursively works and builds the game tree in a bottom-up fashion wherein at each minimizing player's turn; the minimization occurs on the maximum results achieved from the lower layers or vice-versa. The alpha and beta variables are used to determine the branches which can be pruned away as maybe the node heading the pruned branch is lower than alpha or greater than beta. Hence, it can be seen that one player's upper bound can be another player's lower bound and vice versa.

The Algorithm 5 shows the parallel version of chess implementation. The parallelization of the Alpha-Beta algorithm concerned with chess can be applied during the iteration through all the possible moves available to the current player and then choosing the best alternative through recursion. The algorithm also has another aspect of replacing the values of alpha and beta if they find out the moves which are lower than alpha or higher than beta respectively. In this case, the parallelization for various available moves must be done independently of each other, preferably maintaining a critical section so that one thread at a time alters the values of alpha/beta and finally choosing the best possible move. Also, when a thread encounters a subtree rooted with the node which is out of bounds of alpha and beta; it can prune that branch away by not choosing to explore in that side. Since the branches can be pruned independent of each other, we can efficiently make use of the threads to reduce the computation time.

Algorithm 5: Implementation of parallel Alpha-Beta pruning for the game Chess

```
1. Function ABMaxmove( Board,
height_limit,height,a,b):
2. Vector moves;
3. best_move=NULL;
4. Best_actual_move=NULL;
5. Move=NULL;
6. Alpha=a,beta=b;
7. if height>=height_limit:
8.    return board;
9. else
10.    Move_options=board->list_all_moves();
11.    # pragma omp parallel for schedule(static)
num_threads(4)
12.    for iterator it between move_options.begin()
and  move_options.end():
13.
Move=ABMinmove(it,height_limit,limit+1,alpha,bet
a)
14.       # pragma omp critical
15.          if best_move==NULL || move
>evaluate_board(Black)>best_move-
>evaluate_board(black):
16.           Best_move=move;
17.           Best_actual_move=it;
18.           Alpha=move->evaluate_board(black);
19.          end if
20.         if beta>alpha:
21.          return best_actual_move;
22.          end if
23.    end for
24.    return best_actual_move;
25. end if
```

26. Function ABMinmove(Board, height_limit,height,a,b):
27. Vector moves;
28. best_move=NULL;
29. Best_actual_move=NULL;
30. Move=NULL;
31. Alpha=a,beta=b;
32. **if** height>=height_limit:
33. return board;
34. **else**
35. Move_options=board->list_all_moves();
36. #pragma omp parallel for schedule(static) num_threads(4)
37. **for** iterator it between move_options.begin() , move_options.end():
38. Move=ABMaxmove(it,height_limit,limit+1,alpha,beta);
39. #pragma omp critical
36. **if** best_move==NULL || move >evaluate_board(Black)<best_move->evaluate_board(white):
40. Best_move=move;
41. Best_actual_move=it;
42. Alpha=move->evaluate_board(white);
43. **end if**
44. **if** beta<alpha:
45. return best_actual_move;
46. **end if**
47. **end for**
48. return best_actual_move;
49. **end if**

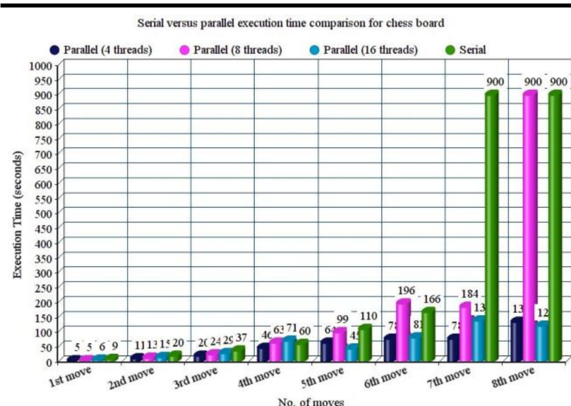

Figure 3: Serial vs. Parallel execution of Chess Board

In Figure 4, the graph shows the results of parallel and serial execution of Chess board. The user enters his/her move and waits for the system to respond with its best possible strategy. The time taken by the system has been recorded for each of its 8 chances. The y-axis indicates the time taken in seconds. The graph clearly shows the trend of the serial and parallel implementation for various numbers of threads (4, 8, 16) so as to determine the best possible parallelization scheme. It can be observed that the boundary of the area covered by the graph with four threads is quite lower than the serial implementation of the same program. The graph also indicates that, as the size of the game tree increases, and as it goes to higher depths (after depth-4) the time difference between the serial and parallel implementation also increases rapidly; i.e. for higher workloads, the parallel Alpha-Beta algorithm performs much better than its serial counterpart. The average speedup achieved for the parallel implementation of chess is 4.258; with the best result for four number of threads.

CONCLUSION

The machine playing games for two players are implemented in serial and parallel using Alpha-Beta pruning techniques. Three games are implemented using Alpha-Beta pruning namely checkers, tic-tac-toe and chess. The results suggest that parallel implementation of Alpha-Beta pruning can add considerable performance to computer chess, tic-tac-toe and checkers program. Significant speedups and efficiency is achieved which is best for a particular game under specific number of processors, threads and board size for the game.

There is a lot of suite for experimentation with the parallel search algorithms. In this work, we explored the performance of parallel Alpha-Beta approach over serial execution. While these game trees mimic most of the behavior of real game trees, there is no substitute for experimentation on real game trees. These game trees have served researchers as the main source of game trees and once again they can be used for neural network approach. The results described here are based on simulations. These can be extended by performing the same experiments on a real shared memory multiprocessor with search trees to achieve better speedups and performance.

REFERENCES

[1] *"ResearchShowcase @CMU".repository.cmu.edu.January,2016.*
[2] *Knuth, D. E., and Moore, R. W. 1975. An analysis of alpha-beta pruning. Artificial Intelligence 6(4):293–326.*
[3] *Russell, S. J., and Norvig, P. 2010. Artificial Intelligence — A Modern Approach (3rd international edition). PearsonEducation.*
[4] *https://skatgame.net/mburo/ps/aaai12-sbs.pdf*
[5] *Ballard, B. W. 1983. The *-minimax search procedurefor trees containing chance nodes. Artificial Intelligence 21(3):327–350.*
[6] *Sturtevant, N. R., and Korf, R. E. 2000. On pruning techniques for multi-player games. In Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of ArtificialIntelligence, AAAI/IAAI 2000, 201–207.*
[7] *Sturtevant, N. R. 2005. Leaf-value tables for pruning non-zero-sum games. In IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, 317–323. Edinburgh, Scotland, UK: ProfessionalBook Center.*
[8] *Cazenave, T., and Saffidine, A. 2011. Score bounded Monte-Carlo tree search. In van den Herik, H.; Iida, H.; and Plaat,A., eds., Computers and Games, volume 6515 of*

Lecture Notes in Computer Science. Springer Berlin / Heidelberg.93–104.

[9] *R. Hewett and K. Ganesan, Dept. of Comput. Sci. & Eng., Florida Atlantic Univ, Boca Raton, FL, USA on "Consistent Linear Speedup in parallel Alpha-Beta search"*

[10] *Marsland, T.A. and Campbell, M.S. (1982). Parallel Search of Strongly Ordered Game Trees. ACM Computing Surveys, Vol. 14, No. 4, pp. 533 – 551*

[11] *"Parallel Search".chessprogramming.wikispaces.com.January,2016.*

[12] *Hyatt, R.M. (1988). A High-Performance Parallel Algorithm to Search Depth-First Game Trees. Ph.D. Thesis, University of Alabama, Birmingham.*

[13] *Hyatt, R.M. (1997). The Dynamic Tree-Splitting Parallel Search Algorithm. ICCA Journal, Vol. 20, No. 1, pp. 3 – 19.*

[14] *Parallel Alpha-Beta Search on Shared Memory Multiprocessors by Valavan Manohararajah (Thesis)*

[15] *Parallel Alpha-Beta Algorithm on the GPU Damjan Strnad and Nikola Guid Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia*

★★★