A Fast Parallel Alpha-Beta Algorithm for Tic TAC Toe Game

Amina Y. AlSallut [#1], Hana H. Hejazi [#2], Heba A. AbuGhali [#3]

[#] *Computer Engineering Department, Islamic University*

Gaza, Palestine

[1]aminay2005@hotmail.com

[2]h_hejaze34@hotmail.com

[3]heba_ali_7@hotmail.com

***Abstract*— In this paper, the parallel search tree algorithms, the minimax and the alpha-beta were studied and compared to a proposed parallel implementation version of the alpha-beta algorithm. In the proposed scheme, a pool master slave model is used in which the master process is given the Root node and divides the nodes among the workers. During computation, a worker which updates the values of alpha or beta – the upper and lower bounds – sends the updated values to the master. Instead of broadcasting these values to the workers by the master, a worker which still has a work to do, requests the work from the master and then the master sends the updated values to this specific worker, thus, reducing the communication overhead. The proposed algorithm among other algorithms were implemented in the frame of a tic tac toe game application. Simulation results showed the effectiveness of the proposed algorithm.**

***Keywords*— Search tree, minimax, alpha-beta, parallel model, pool master-slave.**

I. INTRODUCTION

The paper investigates the efficiency of parallel Alpha-Beta pruning algorithm for search in a game tree. The game used as a case study is a tic-tac-toe. The algorithm will be implemented and parallelized among a number of processors to improve the search performance and speedup. The suggested parallel computational model exploits tree partitioning at width for each level of the game tree, considering the branches that will be pruned; never will be visited. The updated values of alpha and beta will not be broadcasted to all the processors. Instead, we will implement the system such that these values will be communicated on demand, i.e. the processor that needs them is supposed to communicate the primary process and request the updated values, hence reducing the communication overhead. Speedup and efficiency will be estimated on the basis of experimental results. The communication/computation ratio (CCR) of the alpha-beta algorithm will also be estimated.

Search algorithms are essential part of algorithms for solving many problems in computer science with a lot of practical applications such as database systems, expert systems, robot control systems, theorem-provers. Game-playing systems have search engines at the core of the application. A number of search algorithms have been proposed to improve the search efficiency in many practical applications such as branch and bound, minimax algorithm, alpha-beta pruning, etc. A game tree in the game theory is defined as a tree with vertices denoting different game layouts and edges being the possible moves from one position to another. Tree searching is fundamental and computationally intensive problem.

II. GAME TREE SEARCH

A simple game tree for a two-player game is presented in Fig. 1 A node in the tree represents a position in the game while a branch represents a move available at a particular position. Player 1 is on move at rectangular nodes and player 2 is on move at circle nodes. For example, at the Root node, player 1 is on move and the player has two moves available: a and b. Each leaf node is assigned a score that indicate how valuable that position is. A positive score indicates that player 1 is winning, while a negative score indicates that player 2 is winning. A score of 0 indicates a draw. The magnitude of the scores also conveys important information. The higher the score, the more favorable the position is for player 1. Similarly, the lower the score, the more favorable the position is for player 2.

The value of a game tree is the score of the leaf node that is reached when both sides exercise their best options. The problem we need to solve is to find the option at the root that leads to the game tree value. For example, in Figure 1, the ideally best option for player 1 is to move toward position N because it has the highest score (8) from his viewpoint. Assume that player 1 chooses move s to start to make progress toward position N. As far as player 2 is concerned, move e will play right into player 1's hand. To prevent this from happening, a careful player 2 will choose move d instead, and so player 1 has to follow up the move and choose move k, resulting in final score of −1 [1].

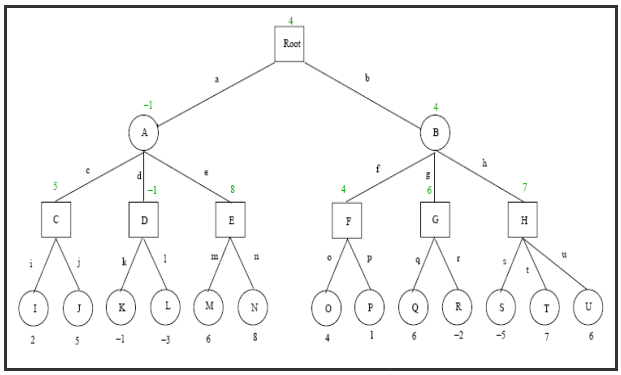


Fig. 1 A simple game tree of two players.

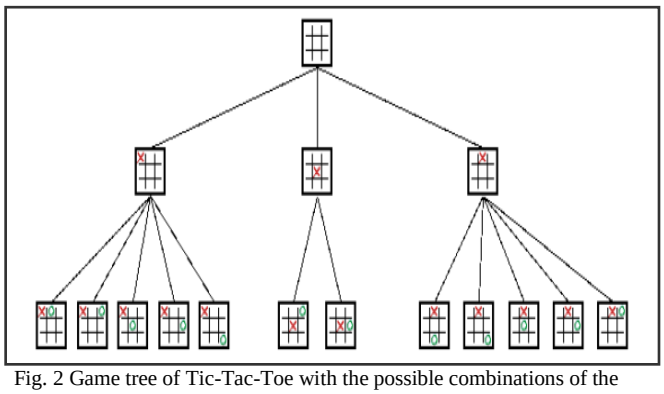


Fig. 2 Game tree of Tic-Tac-Toe with the possible combinations of the first two moves.

If at the root node, player 1 chooses move b, then player 2 has three follow-up moves available, f, g and h. If player 2 chooses move f, player 1 will follow up and choose move o, resulting in final score of 4. Similarly, final scores of 6 and 7 will be returned if player 2 chooses move g and move h respectively. Since move f results in the lowest score (4), player 2 will choose move f. Now let's look at the root node again, since move a and move b result in a score of −1 and 4 respectively, the best score that player 1 can achieve when player 2 exercises his/her best options is 4.

A. Min-Max Algorithm:

The full game tree has a root representing the initial layout of the game and vertices and edges representing all possible moves to the end of the game. All possible moves for the current player are children nodes of the root and then all moves available to the next player are children nodes of these nodes, and so forth. Each branch of the tree represents a possible move that player could make at a given point in the game. Evaluating the game at a leaf of the game tree yields the papered status of the game after that sequence of moves is made by the players. Game tree search is aimed at finding optimal strategy for the game. The algorithm assumes that the second player tries to minimize the gain of the first player, while the first player tries to maximize his gain, hence the name of the algorithm. The game tic-tac-toe is a simple game in which two players, represented as O and X, alternate in marking spaces on a 3x3 grid . The game tree of tic-tac-toe with the possible combinations of the first two moves is shown in Fig. 2 with symmetrical positions omitted for simplicity.

The min-max algorithm traverses the entire tree in a depth-first fashion, and depending on whether a node is maximizing or minimizing, the algorithm keeps track of the largest or the smallest score, respectively. When a leaf node is reached, its score is determined by an evaluation function. Figure 3 depicts the min-max algorithm.

Since Min-max explores every node in the game tree, the algorithm is not practical for a game tree with many branches or depths [1].

```
MinMax(node)
{
  if node is leaf then
     return Evaluate(node)
  if node.type == maximizing then
     score = - infty
  else score = + infty

  for (i = 1, node.number_of_branches)
  {
    value = MinMax(node.branch[i])
    if node.type == maximizing then
    {
       if value > score then
          score = value
    }
    else
    {
       if value < score then
          score = value
    }
  }
  return score
}
```

Fig. 3 The Min-Max Algorithm

B. Alpha-Beta Algorithm:

The min-max algorithm can has been improved by proposing an efficient algorithm, alpha-beta, for sequential game tree search. The idea to cut-off unnecessary branches is to keep two scores in the search. The first one is alpha (lower bound), which keeps track of the highest score obtained at a maximizing node higher up in the tree and is used to perform pruning at minimizing nodes. Any move made from the maximizing node with score less than or equal to alpha is of no improvement and can be pruned, because there is a strategy that is known to result in a score of alpha. The second score is beta (upper bound, which keeps track of the lowest score obtained at a minimizing node higher up in the tree and is used to perform pruning at maximizing nodes. Beta can be viewed as the worst-case scenario for the opponent, because

there is a way for the opponent to force a situation no worse than beta. If the search finds a move that returns a score of beta or greater, the rest of the legal moves do not have to be searched, because there is some choice the opponent will make to prevent that move from happening. The resulting algorithm, called alpha-beta algorithm, is shown in Figure 4 [2].

```
AlphaBeta(node,alpha,beta)
{
  if node is leaf then
     return Evaluate(node)

  if node.type == maximizing then
     score = alpha
  else score = beta

  for (i=1, node.number_of_branches)
  {
     if node.type == maximizing then
     {
       value = AlphaBeta(node.branch[i],
         score,beta)
       if (value >= beta) then
          return beta
       if value > score then
          score = value
     }
     else
     {
       value = AlphaBeta(node.branch[i],
         alpha, score)
       if value <= alpha then
          return alpha
       if value < score then
          score = value
     }
  }
  return score
}
```

Fig. 4 The Alpha-Beta Algorithm

III. Parallel Tree Search Algorithms

A sequential game tree search algorithm uses single processor to search the game tree. In order to be able to search at greater search depths in reasonable time, multiple processors can be utilized for parallel computing. Clusters of the shared-memory architectural style have become popular nowadays as well as hyper threading and multi-core processors. Consequently, shared memory parallel programming models are emerging as a serious competitive environment to message passing.

A. Parallel Min-Max Algorithm

The easiest way to parallelize the minimax algorithm is to partition the search tree into sub-trees and assign them to multiple processors for searching. For that purpose the tree is partitioned at the top level and each processor investigates a single possible move.

B. Parallel Alpha-Beta Algorithm

For distributed-memory machines, principle variation splitting (PVSplit), has been a popular algorithm for searching game trees. In PVSplit, the first branch at a PV node must be searched before parallel search of the remaining branches may begin. Each tree will have to search for its own bounds (alpha and beta), and can't make use of better bounds found by other processors. To make use of these updates, if a processor finds an improvement to alpha or beta, it needs to inform other processors working on other branches so that they can make use of the tighter bounds. Passing updated alpha and beta between processors requires high communication overhead. The contribution of this paper is to implement the parallel Alpha-Beta algorithm and reduce the communication overhead as well [3].

IV. Proposed Scheme

In the proposed scheme (Fast Alpha-Beta Parallel Algorithm), a master-slave model is used in which the master process is given the Root node and the slave processes are idle. The master processor first divides the nodes according to the number of worker processors. An idle worker processor sends a message to the master requesting for work. If there are nodes available and no other processor is working on them, the master chooses one node and sends the node id and current value of alpha and beta to the worker. If a worker finds an improvement to the bounds, then the new score is transmitted to the master. Next time when another worker requests for work, the master will provide it with the updated bounds. A worker processor may also discover a pruning condition (its branch does not need to be searched) with the node it is given. In this case, the search is complete and the worker processor proceeds to request another work from the master or returns to idle state if there is no work available.

C. Performance Metrics:

Our evaluation will be based on execution time differences and comparisons between the following implemented algorithms:

- Tic TAC Toe Tree Search-Based Game using *sequential Minimax Algorithm.*
- Tic Tac Toe Tree Search-Based Game using parallel Minimax Algorithm.
- Tic Tac Toe Tree Search-Based Game using sequential Alpha-Beta Pruning Algorithm.
- Tic Tac Toe Tree Search-Based Game using parallel Alpha-Beta Pruning Algorithm (with Broadcasting technique).
- Tic Tac Toe Tree Search-Based Game using parallel Alpha-Beta Pruning Algorithm (fast) .

Of course, for the parallel algorithms, we will also estimate the communication to computation ratio.

V. Tools, Environment and Implementation

A. Tools and Environment:

- The algorithms are implemented using Visual C++ programming language.

- The parallel multiprocessors environment is simulated using the Message Passing Interface (MPI).
- The algorithms were ran on a 2.4GHz CPU, 512 MB RAM, Windows XP, Professional edition, SP2 machine.

B. Implementation

- The parallel computational model of minimax algorithm for search in a game tree is based on "manager-worker" model, Fig. 5 summarizes the steps.
- The manager process is responsible for the following activities:
- distributes particular positions of the initial mark on the board for evaluation to the worker processes;
- Gathers the best cost function values and the best moves determined by each of the worker processors at a given level of the game tree (function MPI_Gather).
- determines the best cost function value obtained by all worker processes (function MPI_Reduce);
- broadcasts the best cost move to all worker processes (function MPI_Bcast);
- Prints the results after examining the whole game tree.
- The worker processes are responsible for the following activities:
- Receives the specific game move to be evaluated at a given level of the game tree;
- Computes the cost function values for all possible moves of the other player according to minimax algorithm;
- Sends the value of the best cost function and the relevant move to the master process;
- Receives the move to be made at the given level (broadcast by the master process).

And, for the parallel Alpha-Beta algorithm, the same scheme is as the minimax algorithm is applied, with the condition of that, if ,at a specific level, a processor finds that the cost function, i.e. the score of a node is greater/less than alpha/beta, that branch is pruned, and thus , no further work at that branch is to be done. Fig.6 illustrates the steps of this algorithm briefly.

While for the proposed approach, Parallel Alpha-Beta Pruning Algorithm is based on "Pool" model, the same steps as the previous algorithms are to be implemented with the following differences:

- The worker processes ask the master for an available work (node evaluation) to be done, if any.
- If there is available work, the master sends the node id and the current values of alpha and beta to the worker.

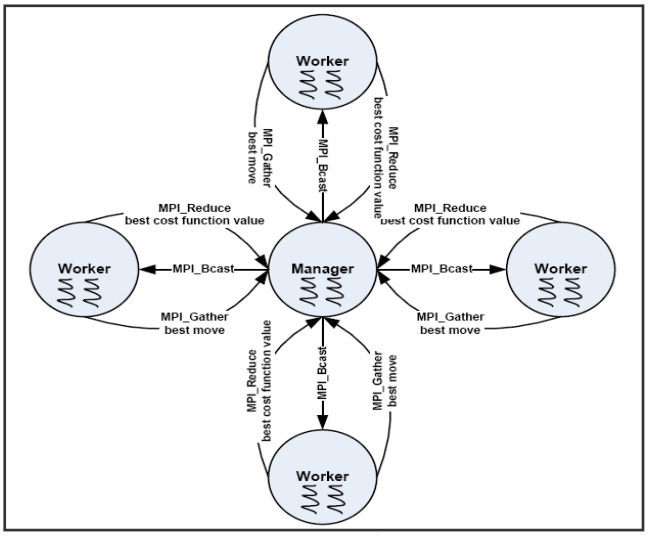


Fig. 5 Parallel computational model of minimax search in a game tree.

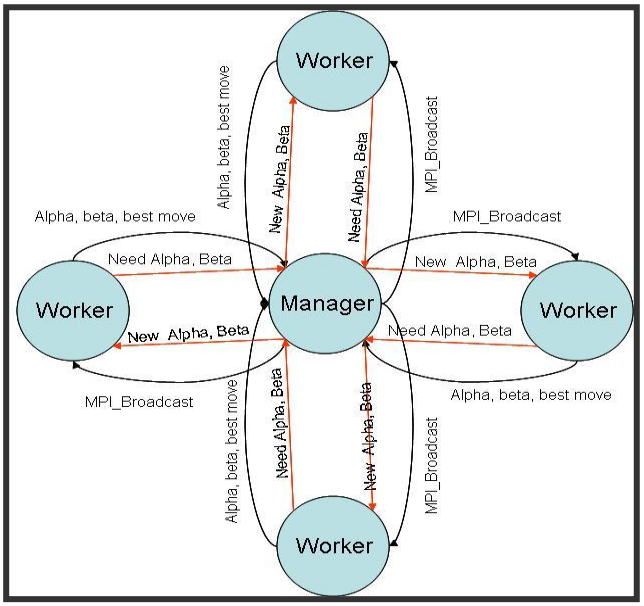


Fig. 6 Parallel computational model of alpha-beta search in a game tree

- If the worker updates the values of alpha and beta during work, it sends the value of the cost function and the updated alpha and beta to the master.
- If a worker finds that the value of the cost function will be greater/smaller than the upper/lower bounds (alpha and beta), no longer work will be done on that branch, i.e. the worker returns.

- When all processors are done, the master determines the best value of the cost function and broadcasts it to the workers. Fig. 7 briefly describes these steps.

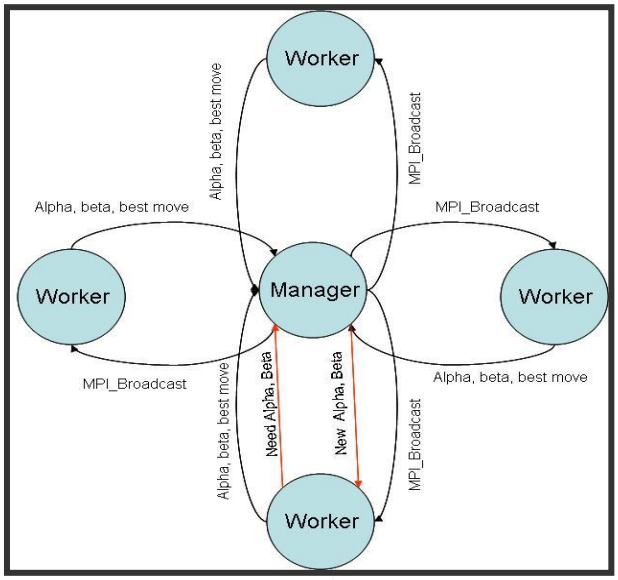


Fig. 7 Parallel computational model of the proposed alpha-beta search in a game tree

These algorithms has been implemented and used in the tic tac toe game, which has the choice of playing against the computer. In this case, the role of the algorithms takes place, i.e. to make the computer's decision of the best next move to be played. Screen shots of the implemented game are shown in the following figure 8.

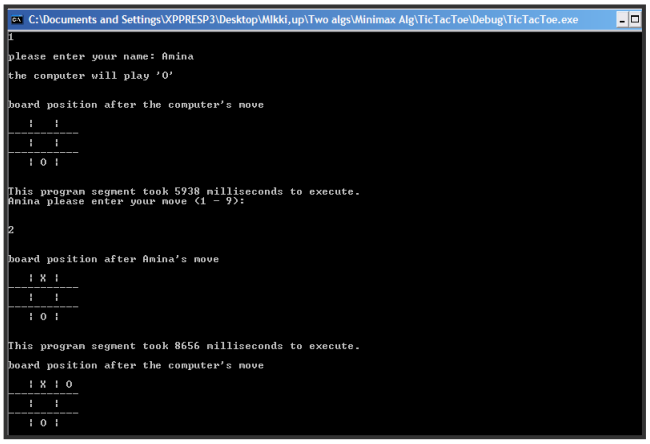


Fig. 8 A screen shot of the tic tac toe Game

C. Results and Evaluation

As described in the previous section, the following algorithms have been implemented and run:

- Sequential Minimax.
- Parallel Minimax.
- Sequential Alpha-Beta.
- Parallel Alpha-Beta (with Broadcasting).
- Parallel Alpha-Beta (fast).

First, the execution time for each of the algorithms has been measured and recorded in table 1.

TABLE I
EXECUTION TIME AND SPEEDUP FOR THE FIVE ALGORITHMS

Algorithm	Execution Time	SpeedUp
Seq Minimax	7500 ms	1
Parallel Minimax	5770 ms	1.3
Seq Alpha Beta	6250 ms	1.2
Parallel Alpha Beta	5000 ms	1.5
Parallel Alpha Beta (fast)	4410 ms	1.7

As depicted in the table, both the sequential algorithms are slow, with respect to the execution time. Obviously, the parallel version of minimax is faster than the sequential one, and the parallel alpha-beta is faster than both the sequential alpha-beta and the parallel minimax. This is can be interpreted because of the pruning technique of the parallel alpha-beta, which reduces the work and thus speeds up the algorithm. Indeed, the proposed alpha-beta algorithm implementation showed the least execution time and therefore, the fastest algorithm among the five. This is of course because of the pruning technique combined with the communication reduction between the master and the slave processes. Fig. 9 demonstrates the execution time for the three parallel algorithms. Speedup is computed in comparison with the sequential minimax algorithm. Fig. 10 shows speedup for the sequential and the parallel minimax algorithms, while fig. 11 illustrates the speedup for the sequential, parallel, and proposed parallel alpha-beta algorithms. Fig. 12 compares the speedup for all the five algorithms.

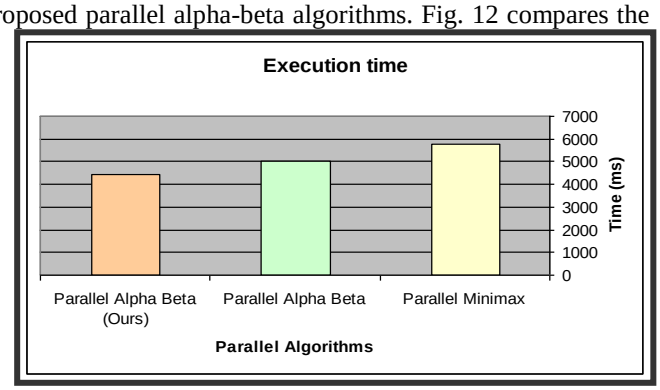


Fig. 9 The execution time for the parallel algorithms

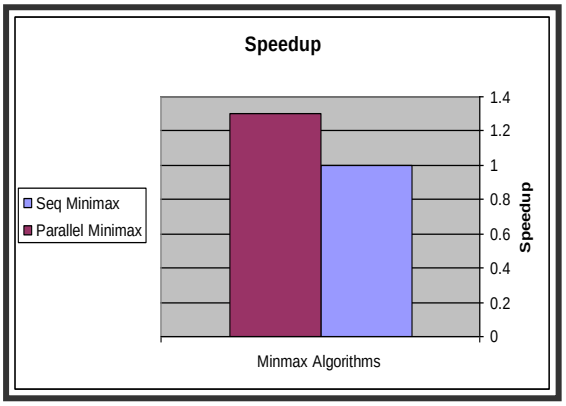


Fig. 10 Speedup for the minimax algorithms

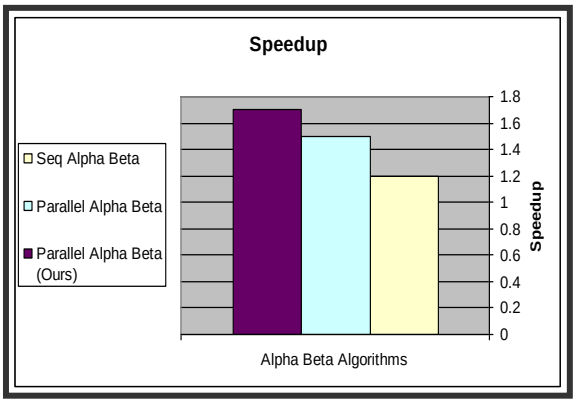


Fig. 11 Speedup for alpha-beta algorithms

Fig. 12 The speedup for the five algorithms

Second, the communication to computation ratio is estimated for each of the parallel algorithms. This ratio has been estimated for the master process and for the worker processes individually. For the worker processes, the communication to computation ratio has been estimated for one process only, because all of the other workers has identical ratios. The computation and communication time for each of the processes (the master and the worker) were measured for the part of code where the computer makes the decision of the next move to be played on the tic tac toe game board. Table 2 summarizes the communication/computation ratios for the algorithms and fig. 13 depicts the communication to computation ratios for the three parallel algorithms.

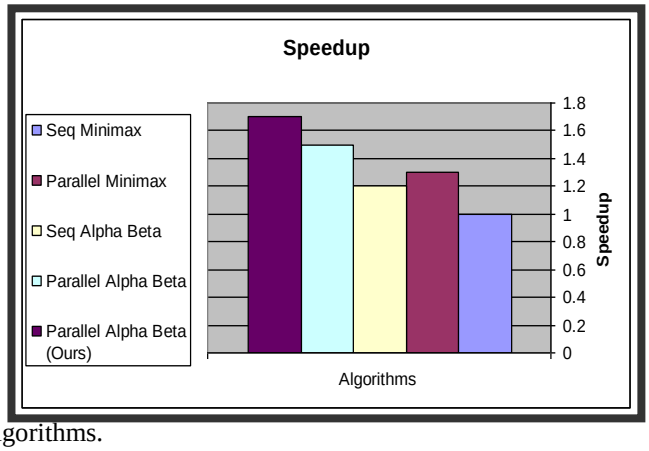


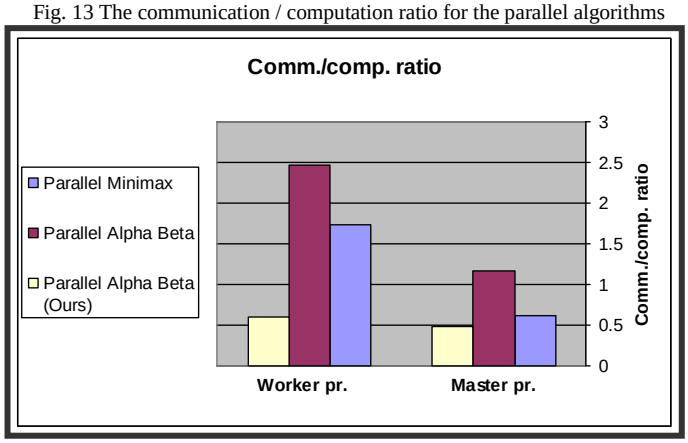
Table 2

The communication to computation ratios for the parallel algorithms

Algorithm	Execution Time	Comp. Time		Comm. Time		Comm/Comp	
		Master Pr.	worker Pr.	Master Pr.	worker Pr.	Master Pr.	worker Pr.
Parallel Minimax	5770 ms	1833	975	1123	1685	0.612657	1.728205
Parallel Alpha Beta	5000 ms	1100	758	1278	1864	1.161818	2.459103
Parallel Alpha Beta (Ours)	4410 ms	1689	1193	814	714	0.481942	0.598491

Obviously, the ratio increases from the minimax to the alpha-beta algorithm, because of the increase of the communication overhead in the alpha-beta algorithm with broadcasting the updated alpha and beta values to all the processors, even if one or more of these processes don't actually need the updated values due to a pruning condition met.

Fig. 13 The communication / computation ratio for the parallel algorithms



On the other hand, the proposed alpha-beta parallel algorithm showed a tremendous reduction in the communication to computation ratio. This is due to the limited communications between the master and worker processes as described before.

VI. Conclusion:

Many game tree search algorithms have been proposed in the literature. Moreover, many parallel versions of those algorithms have been proposed as well. This paper proposed an implementation model of the alpha-beta parallel pruning algorithm, trying to reduce the communication overhead, and thus fasten the algorithm execution, while preserving the quality of the solution. An application game which is the tic tac toe has been implemented using this algorithm. Computer simulations showed the tremendous overhead reduction. This algorithm can be used in many related applications, especially those that need faster execution time.

References

1. Plamenka Borovska, Milena Lazarova,"Efficiency of Parallel Minimax Algorithm for Game Tree Search", ACM International Conference Proceeding Series 285.14 , 2007
2. Brian Greskamp,"Parallelizing a Simple Chess Program", ECE412, 2003
3. Valavan Manohararajah, "Parallel Alpha-Beta Search on Shared Memory Multiprocessors", Masters Thesis, 2001
4. www.netlib.org/utk/lsi/pcwLSI/text/node350.html,Netlib Repository at UTK and ORNL
5. Kevin Steele, "Parallel Alpha-Beta Pruning of Game Decision Trees",1999