Efficiency of Parallel Minimax Algorithm for Game Tree Search

Plamenka Borovska, Milena Lazarova

Abstract: *The paper investigates the efficiency of parallel minimax algorithms for search in a game tree. The game used as a case study is a tic-tac-toe. The suggested parallel computational model exploits tree partitioning at width for each level of the game tree and is based on combination of the parallel algorithmic paradigms "manager-workers" and "asynchronous iterations". Performance comparison has been made for hybrid (multilevel,) flat and multithreaded parallel programming models. Speedup and efficiency as well as scalability in respect to the size of the multicomputer and its impact on the performance of the parallel system have been estimated on the basis of experimental results. The communication/computation ratio (CCR) of the parallel hybrid and flat implementations of the minimax algorithm has been estimated.*

Key words: *Combinatorial Optimization, Game Tree Search, Parallel Minimax Algorithm, Message Passing, Multhithreading, Multilevel Parallel Programming.*

INTRODUCTION

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. [1].

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. Minimax algorithm is a combinatorial optimization algorithm for search in a game tree.

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 [2, 3].

Clusters of the shared-memory architectural style have become popular nowadays as well as hyperthreading and multicore processors. Consequently, shared memory parallel programming models are emerging as a serious competitive environment to message passing. Hybrid (multi-level) parallel programming model is based on a combination of the two approaches - high-level parallelism afforded by message-passing and low-level parallelism used for loop level multithreaded parallelism.

In this paper the efficiency of parallel minimax algorithm for game tree search on multicomputer platform is investigated. The game used as a case study is a tic-tac-toe. The suggested parallel computational model is based on parallel programming paradigm "manager-worker". Hybrid (multilevel) programming model is employed and compared to the performance parameters obtained by message passing and multithreading parallel programming models. Speedup and efficiency are explored on the basis of experimental results obtained. Analysis of the experimental results and parallel performance profiling are aimed at investigation of the algorithm scalability in respect to the size of the multicomputer.

MINIMAX SEARCH 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 projected 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 (known also as noughts-and-crosses) is a simple game in which two players, represented as O and X, alternate in marking spaces on a 3x3 grid [4]. The game tree of tic-tac-toe with the possible combinations of the first two moves is shown in Fig.1 with symmetrical positions omitted for simplicity.

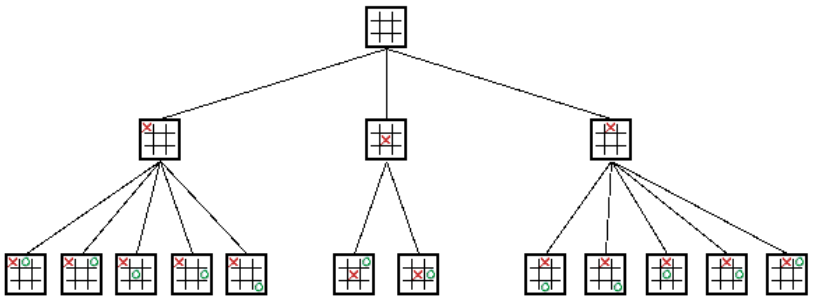


Fig.1. Game tree of Tic-Tac-Toe with the possible combinations of the first two moves

Minimax algorithm can be applied providing there are two players in the game who take turns at playing with a given number of possible moves for a given position in the game. The game is determined, i.e. the game does not employ dice rules of moves. The game is characterized by information transparency, i.e. each player knows the whole state of the game at each position. The leaves of the game tree present the final game positions where the outcome of the game is obvious. The aim of the minimax search in the game tree is to find the optimal strategy as a sequence of best possible moves of a given player taking into account possible moves of the other player up to a given depth.

A minimax search is a recursive algorithm for finding the next move of a given player. It determines all possible continuations of the game to the desired level evaluating each possible set of moves and assigning a score. The search then steps back up through the game tree and alternates between choosing the highest child score at nodes representing the first player and the lowest child score at nodes representing the second player.

A cost function value is computed for each node in the game tree and is associated with the profitability of that game layout for each of the two players. A player chooses a move resulting in a game state that maximizes the minimum value of the possible game layouts resulting from the future opponent moves. For this purpose the whole game tree is searched and a minimax cost function value is estimated for the case each of the players makes its best moves. At each game layout corresponding to a game tree node either minimum or maximum value of the sub-levels is selected in an alternating mode. Finally, one of the leaves is selected as a result, i.e. the final state of the game, which will be the result of the game if each player makes its best moves at each level.

The amount of work a minimax search generates increases exponentially as a move is examined to a greater depth. To reduce computational time the search must be restricted so that no time is wasted searching moves that are obviously bad for the player. An efficient approach to reduce the number of branches to be searched by minimax search algorithm is to implement alpha-beta cutoffs in the minimax search algorithm. It follows the minimax principle of examining the cost of a player move in depth of the game tree. In case the value of the cost function of the current move being examined is worse than the best score guaranteed the algorithm will cancel further search of that branch of the tree. The alpha-beta minimax algorithm keeps track of the best moves for each player as it searches throughout the tree. Two variables are utilized for alpha-beta cutoffs:

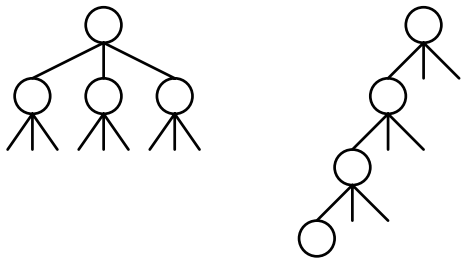
- *alpha* (low level) – it stores the highest gain obtained in a maximizing node in the upper tree levels and is used for cutting minimizing nodes;

- *beta* (high level) – it stores the lowest gain obtained in a minimizing node in upper tree levels and is used for cutting maximizing nodes.

Using alpha-beta cutoffs can cause significant speedups in implementing minimax algorithm by reducing the size of the tree that must be searched.

PARALLEL COMPUTATIONAL MODEL

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. However, if alpha-beta cutoffs are to be used, each tree will have to search for its own bounds, and can't make use of better bounds found by other processors. The two methods for game tree partitioning are illustrated in Fig.2.



(a) partitioning at width (b) partitioning at depth

Fig.2. Partitioning of the game tree for minimax search.

The case study for investigating the efficiency of parallel version of minimax algorithm is a tic-tac-toe game with board size 4×4. The execution time is measured as a time for computing the possible moves of each of the two players and choosing the best move at each level of game tree. The initial state of the game is an empty board. The goal is to evaluate the cost function value at each node of the game tree dividing the computational workload among parallel processes. We apply partitioning of the game tree at width at root, i.e. each process evaluates the cost function value associated with a given level of the game tree. In case of two parallel processes, the first will evaluate the results for positions 0, 2, 4, 6, 8, 10, 12, 14 of the initial mark on the board, and the second will be responsible for evaluating positions 1, 3, 5, 7, 9, 11, 13, 15. In case of three processes, the first process will evaluate positions 0, 3, 6, 9, 12, 15, the second process - positions 1, 4, 7, 10, 13, and the third process - positions 2, 5, 8, 11, 14.

The parallel computational model of minimax algorithm for search in a game tree is based on combination of two parallel programming paradigms: "manager-worker" and "asynchronous iterations" (Fig.3).

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).

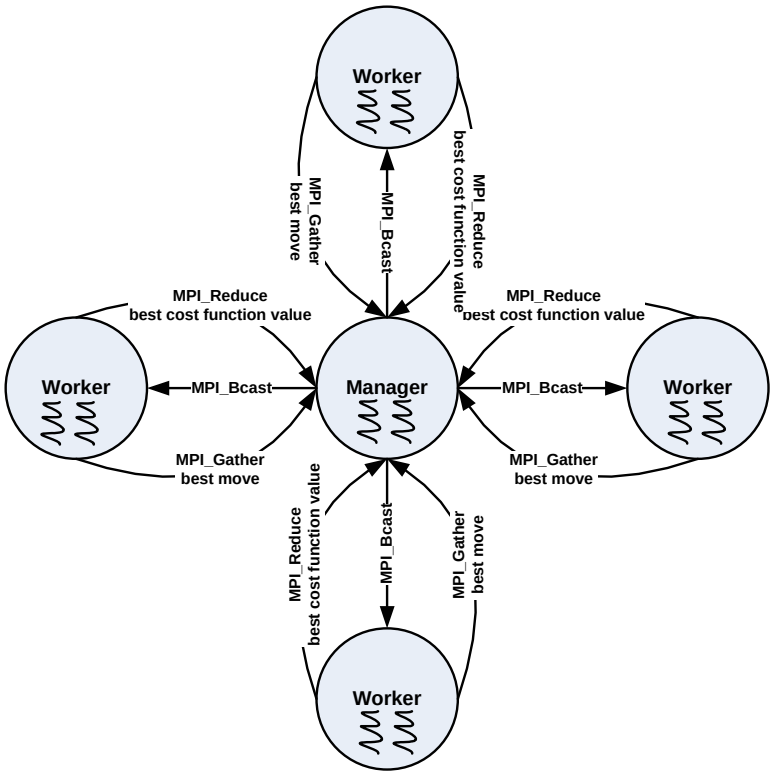


Fig.3. Parallel hybrid computational model of minimax search in a game tree.

PARALLELISM PROFILING AND PERFORMANCE ANALYSIS

Parallel minimax search algorithm and parallel minimax algorithm with alpha-beta pruning are implemented by multithreaded (OpenMP-based) programs. Both algorithms were tested on computer platforms based on dual core processor Intel Core 2 Duo. The experimental results for the speedup and efficiency of parallel minimax and parallel alpha-beta pruning applied at simulated game tree of given depth are presented in Fig.4 and Fig.5, respectively. The results show better speedup and efficiency achieved when alpha-beta cutoffs are used. Both the speedup and the utilization of parallel resources are improved as computational workload increases, i.e. tree depth level.

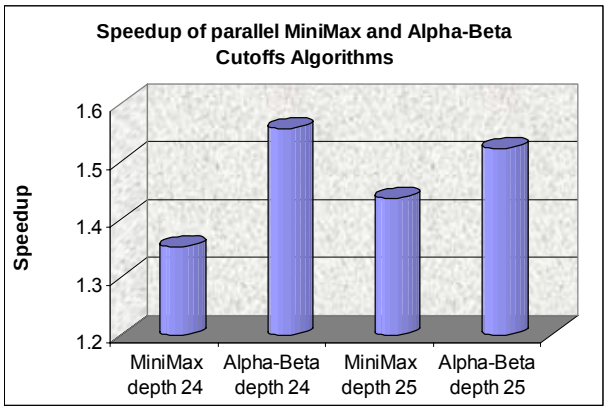


Fig.4. Speedup of the multithreaded implementations of minimax algorithm and Alpha-Beta Cutoffs

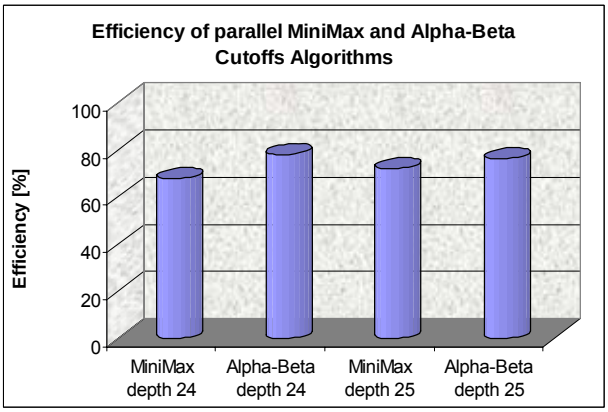


Fig.5. Efficiency of the multithreaded implementations of minimax algorithm and Alpha-Beta Cutoffs

The experimental study of the minimax search algorithm for game tree of 4×4 tic-tac-toe game is performed on multicomputer platform comprising 5 workstations (Intel Pentium IV 3.2GHz, 1GB RAM, hyperthreading) connected by Fast Ethernet 100 Mbps switch. For the implementation of the algorithm Microsoft Visual Studio 2005 and MPICH-2 are used. The experimental parallel implementation is based on multilevel (hybrid) parallel programming model (MPI+OpenMP). The experimental results for speedup and efficiency obtained for different sizes of the multicomputer are shown in fig.6 and fig.7, respectively. Obviously, the hybrid (MPI+OpenMP) implementation outperforms the flat (MPI-based) implementation in respect to the speedup and hardware resources utilization. The application scales well in respect to the size of the multicomputer with approximately proportional speedup and high efficiency of the parallel system.

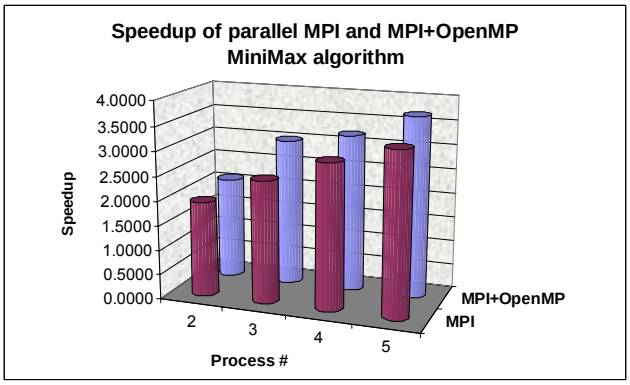


Fig.6. Speedup of the parallel flat and hybrid implementations of minimax algorithm

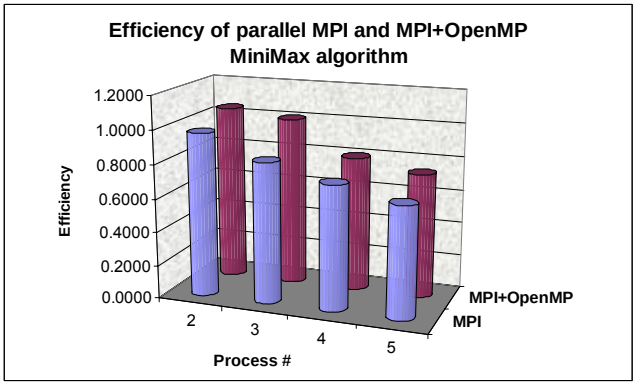


Fig.7. Efficiency of the parallel flat and hybrid implementations of minimax algorithm

The Gantt's chart and the histogram of the communication transactions for 5 processors are shown in Fig.8 and Fig.9, respectively. The most time consuming computations are the first three moves, i.e. the uppermost levels of the game tree. For the lower levels computational time is almost equal for each level and is less than the communication overhead introduced for data exchange between the processes.

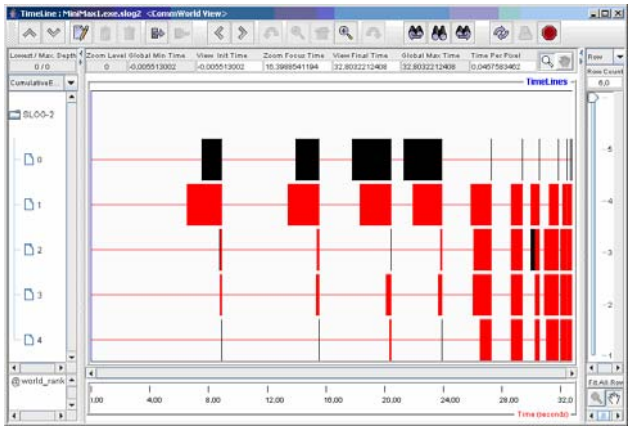


Fig.8. Gantt's chart of the hybrid implementation of minimax algorithm

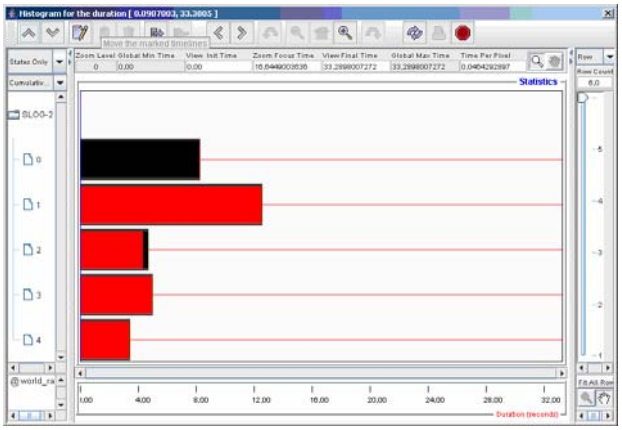


Fig.9. Histogram of communication transactions of the hybrid implementation of minimax algorithm

The computational time and the time for communications of each of the parallel worker processes for the hybrid and the flat parallel implementations of the minimax algorithm are shown in fig.10 and Fig.11, respectively. Comparison of the results shows decrease of the communication overhead in the case of multithreaded processes.

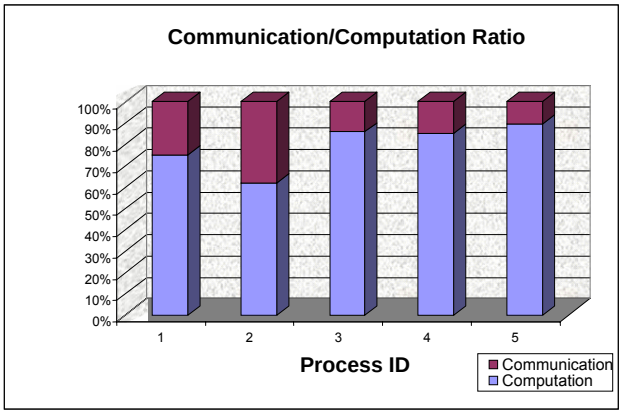


Fig.10. Communication/Computation Ratio (CCR) of the parallel hybrid implementations of the minimax algorithm

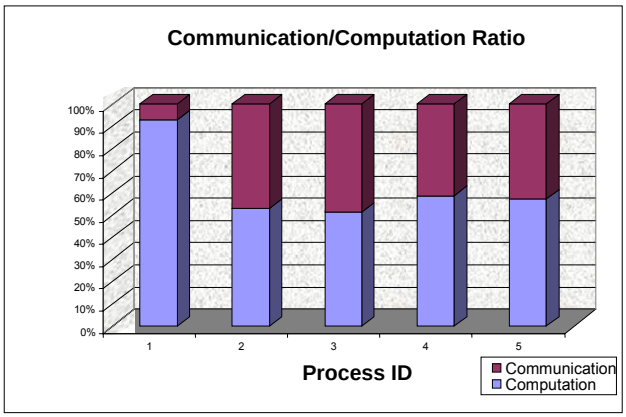


Fig.11. Communication/Computation Ratio (CCR) of the parallel flat implementations of the minimax algorithm

CONCLUSIONS AND FUTURE WORK

The paper investigates the efficiency of parallel minimax algorithms for search in a game tree for the case study of tic-tac-toe game. The suggested parallel computational model exploits tree partitioning at root of each level in the game tree and is based on a combination of the parallel algorithmic paradigm "manager-workers" and the paradigm of asynchronous iterations. Experimental study is based on multithreaded, flat (MPI-based) and hybrid (MPI+OpenMP) parallel program implementations. Performance comparison shows that the hybrid parallel programming model better utilizes parallel hardware resources of the target multicomputer platform. Scalability in respect to the size of the multicomputer and its impact on the performance of the parallel system has been estimated on the basis of experimental results. The communication/computation ratio (CCR) of the parallel hybrid and flat implementations of the minimax algorithm has been estimated. Future work should involve examination of a class of games similar to tic-tac-toe but for spaces of higher dimensionality and games played for "m in a row" on k×n board.

REFERENCES

[1] Eppstein D., Z. Galil. Parallel Algorithmic Techniques for Combinatorial Computation. Annual Review of Computer Science. Vol. 3, No.3, pp.233÷283, 1988.

[2] Brockington M. A Taxonomy of Parallel Game-Tree Search Algorithms. International Computer Chess Association Journal. Vol.19, No.3, pp.162÷174, 1996.

[3] Marsland T., F. Popowich. Parallel Game Tree Search. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 7 No. 4, pp. 442-452, 1985.

[4] Browne, C. Connection Games: Variations on a Theme, A K Peters Ltd., 2005.

[5] Chen Y., Y. Zhang, C. Dulong. Performance Analysis of Two Parallel Game-Tree Search Applications. International Workshop on State-of-the-art in Scientific and Parallel Computing, Umeå, Sweden, June, 2006.

[6] Brockington M., J. Schaeffer. APHID: Asynchronous Parallel Game-Tree Search. Journal of Parallel and Distributed Computing, vol. 60, pages 247÷273, 2000.

ABOUT THE AUTHORS

Assoc. Prof. Plamenka Borovska, PhD, Head of Computer Systems Department, Technical University of Sofia, Phone: +359 2 965 2524, E-mail: pborovska@tu-sofia.bg.

Assist. Prof. Milena Lazarova, PhD, Department of Computer Systems, Technical University of Sofia, Phone: +359 2 965 3285, E-mail: milaz@tu-sofia.bg.