An Efficient Speedup Strategy for Constant Sum Game Computations

Alexandru-Ioan STAN
Business Information Systems Department
Babes-Bolyai University of Cluj-Napoca, Romania
alexandru.stan@econ.ubbcluj.ro

Large classes of game theoretic problems seem to defy attempts of finding polynomial-time algorithms while analyzing large amounts of data. This premise leads naturally to the possibility of using efficient parallel computing implementations when seeking exact solutions to some of these problems. Although alpha beta algorithms for more than one-player game-tree searches show moderate parallel performance, this paper sets forth an alpha beta strategy enhanced with transposition tables in order to offer satisfactory speedups on high performance servers. When the access to the transposition tables is done in low constant delay time, the achieved speedups should approach the theoretical upper bounds of the code parallelism. We tested the strategy on a well-known combinatorial game.
Keywords: *Constant Sum Games, Game Computation Parallelization Strategies*

1 Introduction

High-performance parallel computing is being increasingly used in computational science to solve large-scale algorithmically intensive problems. Recent technological advances in processor development gave rise to a large range of unexpansive chips with tens or even hundreds of cores. These multi-core chips enhanced with simultaneous multithreading, dedicated fast access memory and manifold specific cores, promise additional performance and efficiency gains [1] [2], especially in processing computational science applications.

Large classes of game theoretic problems seem to defy attempts of finding polynomial-time algorithms [3-7], requiring super-polynomial time algorithmic schemes and, thereby, being computationally intensive. Furthermore, large quantities of data can potentially be generated, analyzed and stored while exploring exponentially growing game trees.

These premises lead naturally to the possibility of applying highly parallel computational implementations when aiming to find exact solutions for many game theory problems. We propose a speedup strategy to allow constant sum games computations to take full advantage of the parallelism and the efficient scheduling strategies implemented on these architectures [8] [9]. This way, we obtain fast and accurate sub-game utility estimations by scaling up the amount of required resources and accelerating the dissemination of intermediate results.

This paper is organized as follows: section 2 is devoted to presenting the underlying game theoretic concepts used in this research and related algorithmic complexity aspects. The speedup strategy for constant sum games is expounded in the third section. A case study together with its experimental plan is detailed in section 4. Results and conclusions are presented in sections 5 and 6.

2 Overview of relevant Game Theory Concepts

The current section starts off by briefly restating the definitions of the main concepts used throughout the paper, and proceeds by expounding the underlying considerations on the algorithmic complexity of game equilibria computations.

Brief Overview of underlying game theoretic Concepts

A finite *perfect-information game* in extensive form is defined as a tuple $G = (N, A, H, Z, \chi, \rho, \sigma, u)$ where:

- N is the finite set of rational players;
- A is the set of possible actions of the players;
- H is the set of choice nodes;
- $\chi: H \rightarrow 2^A$ is the *choice function*;

DOI: 10.12948/issn14531305/18.4.2014.13

- $\rho: H \to N$ is the *player function* which indicates player to perform the move at some given choice node;
- $\sigma: H \times A \to H \cup Z$ is the successor function. This function maps pairs of choice nodes and possible actions to other successor terminal or choice nodes;
- $u = (u_1, \dots, u_{|N|})$, where $u_i: Z \to R$ is a utility function for the player i on the terminal nodes Z;
- Z is the set of terminal nodes in the game three. Each terminal node of the game tree has an N-tuple of payoffs, one for each player at the end of every possible play.

Each player has a utility function defined for every game outcome. The rational players evaluate an outcome by its expected utility.

Finite perfect-information *sequential games* are games in which the players take turns changing in defined moves the game configuration so as to achieve deterministically defined winning conditions. In these games, one player chooses his action before the others choose theirs.

In *constant sum games*, the sum of players' payoffs is the same for any outcome. Thus, gains for a subset of players are always obtained at the expense of the remaining participants. These games' equilibria originate frequently from mixed strategies associated with starkly conflicting interests. By normalizing each player's payoff, constant sum games can *always* be represented as *zero sum games* where the total payoff is equal to zero.

Constant sum games are frequently solved using the minimax theorem closely related to linear programming duality [10] and Nash equilibria. Thus, constant sum games equilibria are often find through *minmax* strategies which aim for minimizing the possible loss for a worst case (maximum loss) scenario. Generally, a *maximin* strategy is different from a minimax one. Minimax is used in constant sum games to minimize the opponent's maximum payoff. In constant sum games the two strategies are interchangeable as minimizing the individual maximum loss equates maximizing the minimum gain.

In constant sum games of complete information, minmax decision rules find subgame perfect Nash equilibria, i.e. offer strategy profiles representing a Nash equilibrium for any smaller game of the original game. Every finite game in extensive form has a subgame perfect equilibrium [11].

In these cases minimax provides a recursive method for finding subgame perfect equilibria through backward induction by considering the last actions of the game and calculating backwards the actions the final player should take in order to maximize her own utility. Supposing that the last player does these actions, the previous players at their turn try to maximize theirs utilities and the process continues until the initial configuration of the game is reached. The set of the remaining strategies accounts for all subgame perfect equilibria for finite-horizon extensive games of perfect information [11].

Complexity of Computing Nash Equilibria

Usually, computational problems pertain to one of the two following complexity classes: those which have a polynomial-time algorithm – the complexity class P, and those whose proofs are verifiable in polynomial time by a deterministic Turing machine – the NP complexity class. The class P is contained in NP. NP contains many other important complexity classes of problems. The hardest problems to solve are regrouped in its NP-complete subclass. Their solutions are capable of dealing with any other NP problem in polynomial time. Papadimitriou [3] showed in 1994 that computing a mixed Nash equilibrium in a game pertains to the PPAD complexity class (Polynomial Parity Arguments on Directed graphs). As a matter of fact, computing a Nash equilibrium is complete for the PPAD class of problems, that is, it is the hardest problem to solve in this complexity class. PPAD class is a subclass of TFNP [12] complexity class (Total Function Nondeterministic Polynomial) which, at its turn, is a subclass of the complexity class FNP (the function problem extension of the problem class NP). NP is a class of decision problems, while FNP is the analogous class of function problems.

DOI: 10.12948/issn14531305/18.4.2014.13

Figure 1 depicts the position of the PPAD subclass within the complexity class NP.

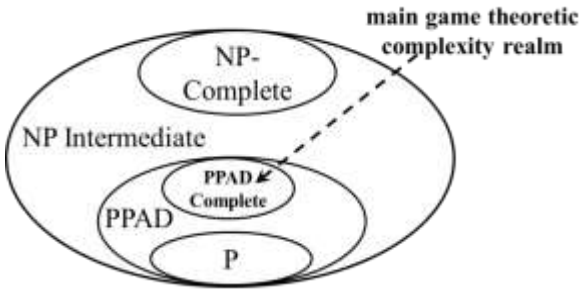

Fig. 1. The nesting of the PPAD complexity class under P ≠ NP hypotheses

Thus, computing Nash equilibria pertains, with some notable exceptions, to the NP-intermediate class and require, thereby, *computationally intensive* algorithmic schemes executed in *super-polynomial time*.

Complexity of Minimax and Alpha Beta pruning Algorithms

The minimax procedure is subject to combinatorial explosion, which makes it rather slow and ineffective when applied to high dimension problems. Therefore, it does not compute the real utility functions of the game players but rather some estimators of these. As the evaluation process is not performed in the proximity of leaf nodes, an "accurate" proxy of the utility function is used in order to decide the "best" moves to play in accordance with the confined perspective of the nearby horizon. The utility function, in this context, assesses the goodness of specific game configurations for a given player. It can take into account a large number of factors and the relations existing amongst them.

Although the minimax enhancements can significantly reduce the branching factor, the number of overall utilities to be computed on subsequent levels goes up exponentially as the branching game tree keeps growing. Thus, the minimum and maximum aggregate utilities should be computed over tremendously large sequences of game configurations.

For constant depth *d* searches in the game tree and average branching factors of *b* the maximum number of nodes to be evaluated is of order $O(b^d)$. Through alpha–beta pruning [13] branches of the game tree can be eliminated in the utility computational process. For optimal or nearly optimal move orderings (the best moves are regularly searched first), the number of leaf nodes to be evaluated is of order $O(b^{d/2})$.Therefore, in the reduced game tree the search can go twice as deep as in the ordinary minimax algorithm while using the same amount of computational resources [13]. Whereas alpha-beta pruning is extremely efficient in minimizing the search tree, there are also many other techniques stemmed from artificial intelligence which can be applied to further reduce the search space of the problem. Among those refinements we can cite alpha-beta enhancements, transposition tables, null move pruning and late move reductions. In some cases, they can further reduce the effective branch factor below the value of 3 and, more rarely, even below 2.

3 The computational Strategy

The strategy we propose relies mainly upon using the largely known minimax optimization of alpha beta pruning, in conjunction with low latency intermediate results repositories, i.e. transposition tables [14][15], and a good granularity in the splitting process of the computational tasks.

The speedup strategy relies mainly upon low latency transportation tables which accelerate the searching process in the game tree. Transposition tables are very useful in performing perfect information games computations where all players fully apprehend the whole state of the game, as they apply memorization to the tree search by using dynamic programming techniques. This way, we can keep track of the millions of positions analyzed up to a given point in time.

In a large number of games, it is possible to attain specific game configurations in more than one way. These different move sequences enabling players to reach one same position, are called *transpositions*. After sequences of n consecutive moves, the combinatorial limit on the number of possible transpositions may usually reach an upper bound as large as $(n!)^2$. In spite of the fact that some of these possible moves are obviously forbidden, there remains a large amount of positions that

DOI: 10.12948/issn14531305/18.4.2014.13

may be explored multiple times. We prevent this problem by using transposition tables.

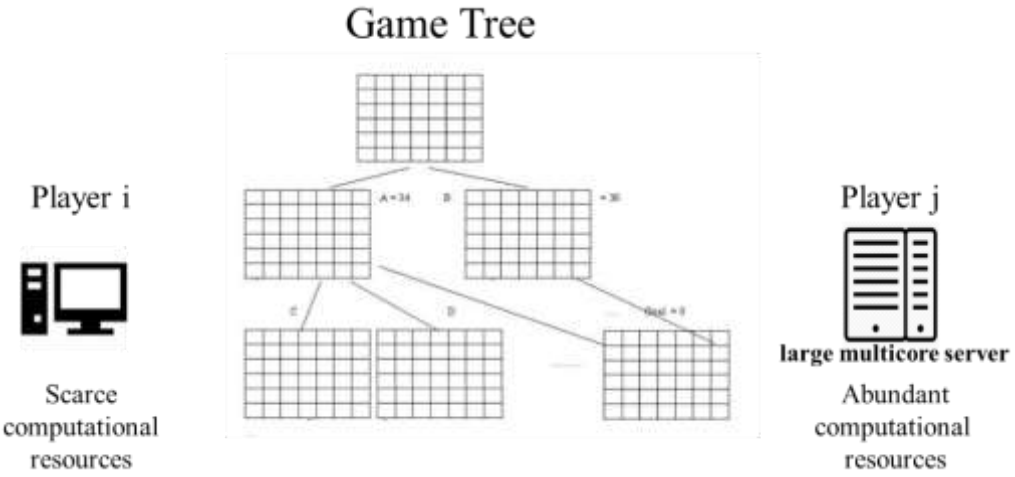

Fig. 2. Taking advantage of the abundant computational resources of large multi-core servers while exploring the game tree

Such tables are hash tables of game configurations analyzed up to a certain depth. On encountering a new position, the program checks the table to see whether the position has already been analyzed; this can be done quickly, in expected constant time. If the table contains the value that was previously assigned to this position, then this value is used directly. If not, the value is computed and the new position is entered into the hash table.

The strategy encompasses the following five aspects:

- The division of the utilities computations of the game tree configurations into a relatively high number of smaller tasks of order O(*branching factor*nb. of computing nodes*) in order to take full advantage of the parallelism and computational distribution;
- The implementation of the game tree as a set of files gradually built up by parallel processes performing time-costly intermediate sub-games computations. A shared file representation can tremendously enhance parallel computation of the tasks. In this case I/O operations (create, delete, modify, read and write) must be as fast as possible to assure the overall application performance and surmount tolerable processing latencies. We recommend a low latency file set characterized by high speed and accurate communication amongst computing workers;
- The usage of Alpha–beta pruning as optimization of the minimax algorithm in order to decrease the number of nodes to be evaluated. In our strategy the transportation table is a stored in a set of files. Such an implementation is helpful not only when seeking equivalent game configurations. In the alpha-beta implementation, the search is optimal when the best successor nodes are explored first. As the best move is not known as long as leaf leave levels are not reached, when using iterative deepening the best move found in the shallower search offers a good approximation of that and we store it in the transposition table as the best child of the node;
- Read/Write/Update operations of game tree data files containing the intermediate utility results by parallel processes performing the computational tasks. When intermediate results are computed/updated, they should be stored right away in the shared game tree. Here, synchronization methods are required in order to ensure consistency amongst the different physical supports. The data integrity is here of paramount importance and we

DOI: 10.12948/issn14531305/18.4.2014.13

must take advantage of effective replication and failure detection mechanisms. Any breach of data integrity can lead to incorrect computing and, therefore, problems must be quickly detected;

- A cache strategy of storing the data as the search space grows exponentially and, thereby, it may overpass the allocated memory capabilities. Thus, it can arrive that not all game positions can be stored. When the shared file physical size approaches some given threshold, rarely used positions are replaced with new ones as in classical cache mechanisms. The cache implementation might not be necessary on some configurations offering virtually unlimited storage amounts.

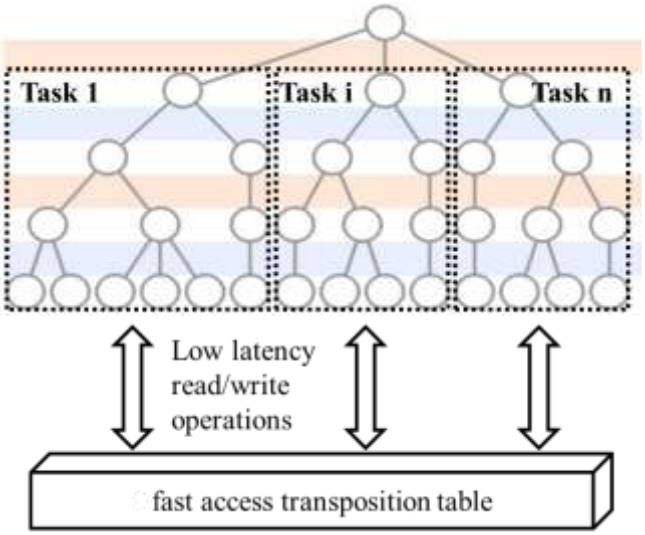

Fig. 3. Outlook of one tree node utility estimation

Using transposition tables may provoke erroneous results when past interactions in the game tree are not carefully considered. Problems could arise in games where the history of certain positions may reveal essential. For instance, in some board games such as chess, players may play two specific moves at a time only if the pieces involved had not previously moved in the game. Solutions to this type of problems generally involve adding supplemental information as part of the hashing key. Other examples are repetition draws when some positions repeat themselves during the course of the game leading to a draw outcome. Repetition of past positions can be prevented by storing history information in the nodes of the transportation table although this could be inefficient from a computational perspective.

4 Appling the computational Strategy to a well-known combinatorial Game

In this section we start off by offering a very succinct general description of the game on which we tested the effectiveness of the speedup computational strategy. Then, we present the testing strategy and some coding aspects.

General game Description

We considered testing the strategy on the *International Draughts* combinatorial game. A perfect information two-player game over a game tree, as international draughts, can be represented as an extensive form game over a specific game tree with terminal nodes having payoffs for win/draw/lose outcomes.

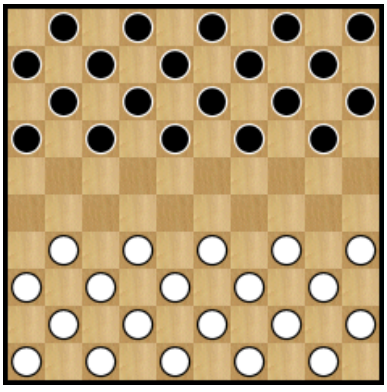

Fig. 4 Initial position

DOI: 10.12948/issn14531305/18.4.2014.13

International draughts is one of the variants of draughts, a two-player game played on a 10 by 10 board with alternating black and white squares. In conventional diagrams the board is displayed with the white pieces at the bottom and black at the top as in Figure 4.

- The two players are at opposite sides, with 20 pieces each, white for one player and black for the other.
- The game is played on the black squares of the board. Thus, there are 50 active cases. The longest diagonal joining two corners of the board and including 10 black squares, is referred to as the long diagonal.
- Before starting a game, the 20 black and 20 white pieces are arranged on the first 4 rows of each player as can be seen in Figure 4.

Without entering into much detail, the moves and captures abide by the following rules:

- There are two types of pieces: ordinary pieces and crowned pieces (or kings);
- The first move is always played by the white;
- Opponents make moves alternately;
- An ordinary piece must move forward, diagonally one square on an empty square in the next row;
- When it reaches the last row, the piece becomes a king. For this, the piece is crowned by placing over a second piece of the same color;
- A king must wait until the opponent has played at least once before taking action. A king moves backward or forward on successive free squares on the diagonal it occupies;
- Opposing pieces must be captured whenever a piece (crowned or not) jumps over them;
- The opposite pieces must be captured even though this would be disadvantageous;
- The game is won whenever either player has no pieces left;
- A game is a draw if the two opponents do not have the possibility to win the game or board configurations repeat themselves.

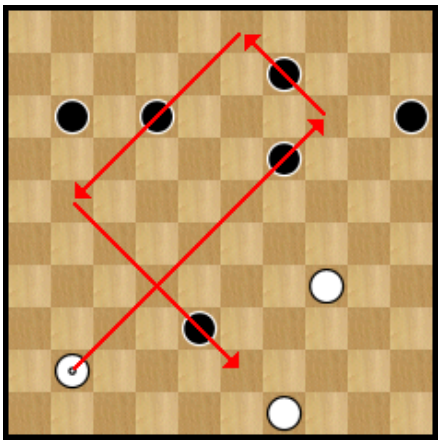

Fig. 5. Example of capturing of several black pieces by a white king

A detailed list of the International Draughts game's rules can be found in [16].

The evaluation Functions

In the Draughts game case the utility function estimator is generally computed as a weighted average function. The different weighed variables are: the number of black and white pieces, the number of black and white crowned pieces and the disparity of pieces (the number of black pieces minus the number of white pieces, the number of black kings minus the number of white kings).

In the implementation of the game-playing alpha-beta pruning algorithm, we used three different evaluation functions to estimate the utility (or the goodness) of specific board configurations for each player. The evaluation functions were typically designed so as to increase the speed of exploring the subsequent configurations tree. The functions were static in the sense that they analyze only board configurations from a static perspective and do not explore possible subsequent moves.

Board Encoding

For a given configuration we represented each Draughts board using 6 double precision 64 bit-boards. The first 2 double precision bit-boards were used to indicate whether each cell is or not empty. The third and the forth bit-boards were used to indicate the type of the piece (crowned or not crowned). The fifth and the sixth bit-boards were used to give the color

DOI: 10.12948/issn14531305/18.4.2014.13

of the existing pieces (black or white).

The testing Approach

Our testing method made use of two non-human computing players playing against each other. One of the players employed a sequential algorithm while the other player had access to parallel computational resources and implemented a solution in conformity with the recommended strategy.

Two performance case studies were employed:

- In the first case, both players had to find a move in a given amount of time. Here, we counted the number of tree nodes analyzed by each player during the same time period;
- In the second case, the players had to find the best move by exploring the tree up to some maximum depth. Here, we compared the time amounts taken by the players to perform the equivalent complexity computations.

In order to assess the importance of the I/O delay time to access the transposition table, we also artificially delayed each I/O operation and evaluated the performance loss under high, medium and slow latency conditions.

5 Some experimental Results

This section goes through several execution aspects observed while running the combinatorial game. Some aspects are specific to this particular game, whereas others may apply to any constant sum game implementation.

Observed average Branching Factor

In our setting, there were in average around 7 to 11 moves per position. Thus, adding an additional level to the game tree roughly expanded each leaf by the average branching factor of the game tree. The alpha-beta pruning optimization reduced branching by factors ranging between 20 and 25 percent. This decreased the number of nodes to be evaluated and improved the overall computational speedup without any loss in result accuracy. We first explored the best moves configurations in order to ensure good effectiveness in the pruning scheme.

Speedup Measurements

In this context, the speedup metric is the improvement factor as mode computational contribute to decrease the running time. We measured the speedup factor of the algorithmic strategy by timing the execution for different parallelism levels (i.e. number of computing nodes, processes or threads).

The speedup has the following power function form with a sub-unitary positive exponent: $speedup(p) = p^q$. Here p stands for the parallelism level and q for the discounting exponent. The exponent can be estimated by a log-log regression on the speedup data series, i.e. $\log(speedup) = q\log(p) + \varepsilon$. For low depth searches the q exponent was around 0.8, while for high depth searches it was at 0.7 or below. This values approach theoretical maximum speedups levels for the draughts game given the effective branching factor.

The benchmark is obtained by performing successive searches on 30 valid game configurations. The test configurations were generated at random. Thereafter, we cumulated the overall search time for all of these searches, for different search depths. In order to get good estimators for the average search time in each case, we repeated the operation many times.

DOI: 10.12948/issn14531305/18.4.2014.13

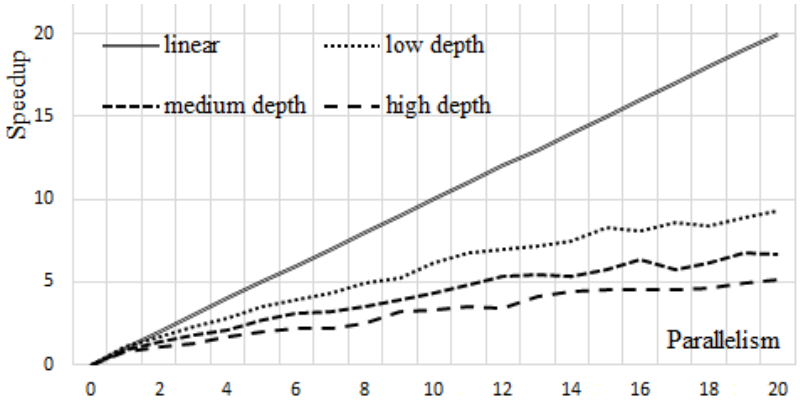

Fig. 6. Speedup for different search levels

The curves in Figure 6 indicate the level of average search time for different levels of parallelism. The principal observation is that the overall speedup is roughly a increasing function of the search time and search depth. This is a rather natural result as deeper search trees are characterized by increasing parallelism and thereby more speedup.

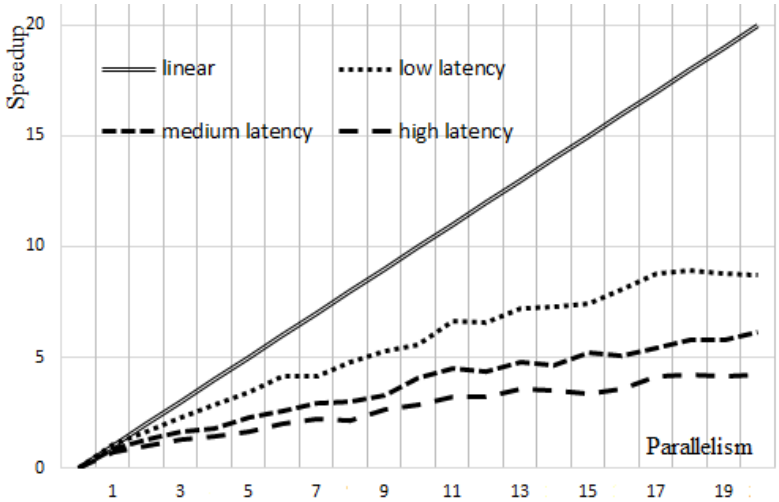

Fig. 7. Impact of I/O latency on accessing the transposition table

The measured speedups curves for different latency levels are depicted in Figure 7. The value of the speedup exponent ranged between 0.65 and 0.8. The curves in Figure 7 indicate that the speedup is roughly a decreasing function of the latency time. We can also see that there are no high amplitude "humps" in the curves. The lack of "hump" shapes in the low latency curve also suggests that within the game tree the parallelism hierarchy is well controlled, so we are not suboptimal in the task allocation. This is somehow an indication that more flexible node allocation schemes will not perform much better.

6 Conclusions

Many problems pertaining to the game theory realm appear not to possess exact polynomial-time solutions and, therefore, are computationally intensive. Furthermore, they may generate and analyze large quantities of data. Under these circumstances, applying parallel computing implementations, when seeking exact solutions, can be an interesting and effective computational alternative.

We proposed in this paper an alpha beta strategy enhanced with low latency access transposition tables in order to offer satisfactory speedups in high-performance parallel systems. We tested the strategy on a well-known combinatorial game: the International

DOI: 10.12948/issn14531305/18.4.2014.13

Draughts. In order to further increase the speed of the computations we reduced the frequency and the overall size of I/O operations on the transposition table. When the access to the transposition tables is done in low constant delay time, the achieved speedup performance vastly improves the sequential running time approaching the theoretical upper bounds of the code parallelism.

Acknowledgement This work was co-financed from the ESF through Sectoral Operational Programme Human Resources Development 2007-2013, project POSDRU/159/1.5/S/134197 „Performance and excellence in doctoral and postdoctoral research in Romanian economics science domain" and UEFISCI, under project PN-II-PT-PCCA-2013-4-1644.

References

[1] J. E. Savage and M. Zubair, "A unified model for multicore architectures", in *Proceedings of the 1st international forum on Next-generation multicore/manycore technologies (IFMT '08)*, ACM, New York, NY, USA, Article 9 , 12 pages, 2008.

[2] V. Pankratius, C. Schaefer, A. Jannesari, and W. F. Tichy, "Software engineering for multicore systems: an experience report", in *Proceedings of the 1st international workshop on Multicore software engineering (IWMSE '08)*, ACM, New York, NY, USA, pp.53-60, 2008.

[3] C. Papadimitriou, "On the complexity of the parity argument and other inefficient proofs of existence", *Journal of Computer and System Sciences*, Vol. 48, no. 3, pp.498–532, 1994.

[4] C. Daskalakis, P. Goldberg and C. Papadimitriou "The complexity of computing a Nash equilibrium", in *Proc. of the 38th Annual ACM Symposium on the Theory of Computing (STOC)*, ACM Press, 2006, pp. 71–78.

[5] A. Fabrikant, C.H. Papadimitriou and K. Talwar, "The complexity of pure Nash equilibria", in *Proc. of the 36th Annual ACM Symposium on the Theory of Computing (STOC)*, ACM Press, 2004, pp. 604–612.

[6] G. Gottlob, G. Greco and F. Scarcello, "Pure Nash equilibria: Hard and easy games", *Journal of Artificial Intelligence Research*, Vol. 24, pp. 195–220, 2005.

[7] G. Schoenebeck and S. Vadhan, "The computational complexity of Nash equilibria in concisely represented games", in *Proceedings of the 7th ACM Conference on Electronic Commerce (ACM–EC)*, ACM Press, pp. 270–279, 2006.

[8] R. Nobre, P. Pinto, T. Carvalho, J. M. P. Cardoso, and P. C. Diniz, "On Expressing Strategies for Directive-Driven Multicore Programing Models", in *Proceedings of Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM '14)*, ACM, New York, NY, USA, pp.7-14, 2014.

[9] G. Imre and G. Mezei, "Parallel graph transformations on multicore systems", in *Proceedings of the 2012 international conference on Multicore Software Engineering, Performance, and Tools* (MSEPT'12), Victor Pankratius and Michael Philippsen (Eds.), Springer-Verlag, Berlin, Heidelberg, pp.86-89, 2012.

[10] K. Binmore, *Playing for real: a text on game theory*. Oxford University Press US, 2007, chapters 1 & 7.

[11] M.J. Osborne, *An Introduction to Game Theory*. Oxford University Press, USA, 2004.

[12] K. Daskalakis, *The complexity of Nash equilibria*. ProQuest, UMI Dissertation Publishing, September, 2011.

[13] S. J., Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (3rd ed.).* Upper Saddle River, New Jersey: Pearson Education, Inc., 2010, pp. 167 ISBN 0-13-604259-7.

[14] A. Kishimoto and J. Schaeffer, "Distributed game-tree search using transposition table driven work scheduling", in *IEEE*

DOI: 10.12948/issn14531305/18.4.2014.13

Proceedings of the 9th International Conference on Parallel Processing(ICPADS), pp. 323-330, 2002.

[15]] A. Kishimoto and J. Schaeffer, "Transposition Table Driven Work Scheduling in Distributed Game-Tree Search", in *Proc. of Fifteenth Canadian Conference on Artificial Intelligence (AI'2002)*, vol. 2338 of *Lecture Notes in Artificial Intelligence* (*LNAI*), pp. 56-68, Springer, 2002.

[16] Official FMJD rules for competitions. Available: http://www.fmjd.org.

Alexandru-Ioan STAN is a teaching assistant at the Faculty of Economics and Business Administration, within the Business Information Systems department at Babes-Bolyai University of Cluj-Napoca. He holds a Ph.D. from the same university. His research is oriented towards the applications of computational finance. He is also interested in parallel and distributed computing.

DOI: 10.12948/issn14531305/18.4.2014.13