Parallelizing Special-Purpose Hardware and Improved Search Techniques for AntiChess Agents

Kumaresh.P, Niranjan.B, Bharanidharan.A, Harish.M, V.Uma Maheswari
Department of Computer Science and Engineering, Anna University, Chennai, India

Abstract—***The idea of creating a machine which can match or even better human intellect has been one of man's greatest fascinations. Chess agents such as Deep Blue and Fritz have given glimpses of what could be expected in future. By utilizing special-purpose hardware chips and extensive search algorithms, chess agents have taken machine intelligence to a newer level. AntiChess, also called losing chess or suicide chess, is a chess variant in which the objective of the participant is to get all his pieces captured. In our paper, we look at the prospect of improving accuracy and performance of AntiChess agents. Our proposed architecture makes use of special-purpose hardware chips in parallel with improved search techniques to do the same. These concepts could also assist decision making in competitive domains and solving real world problems.***

Index Terms—**AntiChess, Deep Blue, Quiescent Search, Singular Extensions, Special-Purpose Hardware Chips, Parallelization**

I. Introduction

It is true that powerful hardware forms the basis of a chess computer's capabilities. However, improved software based search techniques and innovations in effectively parallelizing hardware and software also play important roles. The idea of creating a machine which can match or even better human intellect has been one of man's greatest fascinations. It began with the development of a primitive chess playing agent. Since then, a lot of research has gone into developing parallel algorithms and hardware designs finding many applications in complex real world problems.

AntiChess is a variant of chess in which the goal is to either lose all of your pieces (except your king) or force your opponent to checkmate you. Though the game seems restricted and similar to chess, even the best chess grandmasters have found it a tough proposition. Like chess, AntiChess has certain features which find relevance to a number of real world problems. The rules of the game are detailed in the appendix.

II. Related Work

Quite some work has gone into design of software approaches and special-purpose hardware in the field of game-theory. Agents such as Deep Blue, Deep Thought, Rebel and Fritz have revolutionized the way man looks at machines. Deep Blue, in particular, with its judicious use of
hardware and search algorithms has set very high standards of machine intelligence.

On the other hand, not much work has been done with regard to AntiChess. Current research concentrates on customizing the existing algorithms to suit AntiChess – principal variation search, zobrist keys to name a few.

We look at the possibility of using special-purpose hardware chips, similar to those used in Deep Blue in parallel with adaptation of existing search algorithms to achieve an improvement in accuracy and speed of operation of the AntiChess agent. To the best of our knowledge, this is the first attempt at parallelizing customized hardware with improved searches in the field of AntiChess.

The design, implementation and simulation results of our proposed architecture are given in the following sections.

III. Design

A. Overview

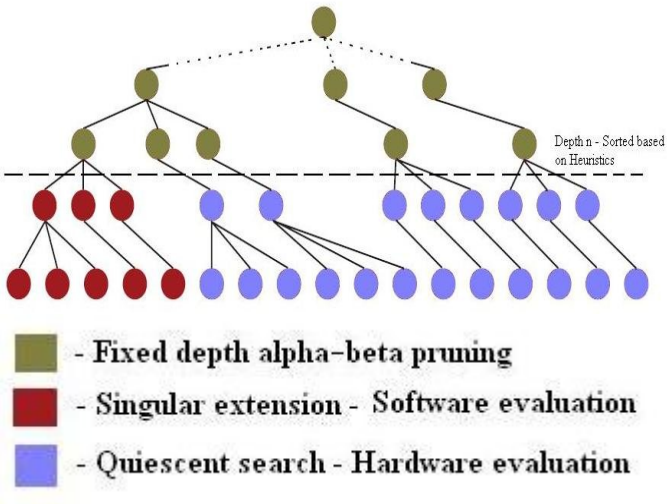
The heuristic values assigned to each piece-position pair, determines the quality of the static position evaluator in chess. But, in anti-chess, even if the heuristic evaluator judges a side clearly favorite, there could be a series of moves for the opponent which involves forced captures that could change the game altogether. It is to be noted that, in AntiChess, captures when possible *must* be made as opposed to chess, where you have a choice. As a result, we need to focus on the depth of evaluation instead of relying heavily on the evaluation function. It is here that, our proposal to use special-purpose hardware concurrently with software search, has high relevance.

The AntiChess algorithm used for static evaluation of a position varies from that of chess, only in terms of the heuristic values we assign to each piece-position pair. For instance, a queen at the centre of the board, has the highest heuristic in chess (as it controls maximum number of squares), whereas, it has the least value in anti-chess (as it would be in a position to capture the opponent pieces, which goes against the theme of AntiChess. So, the way the evaluation function operates in AntiChess is pretty similar to chess.

B. Singular Extensions

For the AntiChess software, we used the existing alpha-beta search algorithm incorporating *singular extensions*. Alpha-beta usually implies a search and evaluation of all the possible moves to a fixed depth-say 7 or 8 ply.
An analysis of grand master chess play indicates that such expert players would search much deeper along forcing lines than a fixed-depth program could reach. Thus, we bring in the idea of s*ingular-extension*. A move is called singular if the value associated with it is much better than that of the alternative moves evaluated unto a certain depth.

Once the singular moves have been identified, the program then searches those lines much deeper than normal.



However, as stated before, there could be dynamic possibilities in AntiChess which aren't rated among the best in terms of heuristics (non-singular), but may lead to a series of forced captures (lying outside the current search depth) that could prove better than the singular variations.

It is here that we bring in special-purpose AntiChess hardware chips aided quiescent search tailor-made to identify a series of captures and check moves. These work in parallel with the software evaluation of singular moves and improve accuracy. We describe these modules below:

C. Quiescent Search

Clearly, AntiChess contains many forcing tactical sequences. If our opponent takes our knight, and we have the opportunity to capture it, then it is mandatory that we do the same. Alpha-beta search is not particularly tolerant of this kind of nuance as it also searches for non-capturing possibilities that are incidentally, against the rules of AntiChess.

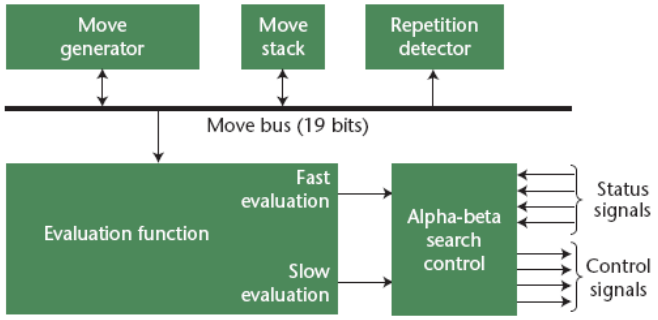
If a depth parameter is passed to the alpha-beta function, when the depth gets to zero, the search is terminated even if there is a forced capture move ahead. We deal with this using an adapted quiescent search.

When alpha-beta runs out of depth, we stop normal evaluation and call the quiescent search function which is essentially, an evaluation function that takes into account some dynamic possibilities. Our implementation of quiescent search looks only for captures and checks. Unlike chess, where captures are not compulsory, this method turns out to be very useful in AntiChess.

If it were possible to make a more accurate quiescent search with no loss in speed, our algorithm would be stronger than it is. When we were pondering about this possibility, we struck upon the idea of using specialized hardware chips to aid quiescent search, which we describe in the next section. Our current implementation increases the depth by one up until it reaches a board position with no captures and checks.

D. Special-Purpose Hardware Chip

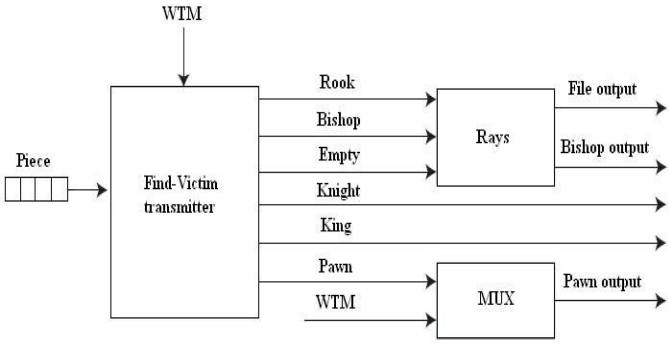
"Writing" a program in silicon offers design possibilities not available in pure software—in particular, the time complexity for competing algorithms can change dramatically. An algorithm unacceptable in a software design might work perfectly well in hardware. Either the algorithm could be trivially parallelizable in hardware without significant area penalty, or the time-scaling factor could drop from the instruction cycle time to simple gate delays.



We adapted the Deep Blue chess hardware, shown above, to meet the demands of a parallel quiescent AntiChess search. The Deep Blue chess chip is divided into four parts: the move generator, the smart-move stack, the evaluation function, and the search control. The smart-move stack further divides into a regular move stack and a repetition detector. The move generator detects dynamic moves and feeds it to the evaluation function. The operation of the evaluation function needed for AntiChess is the same as that of chess. But we came up with the following design for the move generator of AntiChess:

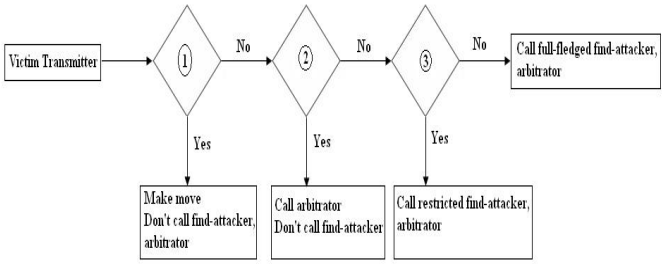
The move generator is a 8*8 combinational logic array, effectively a silicon chessboard capable of making checking, check evasion or "attacking" moves directly. Each cell in the array has four main components: a find-victim transmitter, a find-attacker transmitter, a receiver, and a distributed arbiter. Each cell contains a four-bit piece register that keeps track of the type and color of the piece on the corresponding square of the chessboard.

When enabled, the find-victim transmitter radiates appropriate attacking signals for the resident piece. If the square is vacant, incoming attack signals from a ray piece (a bishop, a rook, or a queen) pass through the cell. Third-rank squares have additional circuits to handle the two square pawn moves. At the start of a typical move generation sequence, a find-victim cycle executes, and all the moving side's pieces radiate attacking signals. The radiated attacking signals then reach the receiver, and a vote takes place to find the lowest valued victim. We reverse the priorities assigned to pieces in chess.



However, we modify the find victim transmitter as follows:

1) If there is only one victim, and only one of our pieces attacking it, we don't call the find attacker cycle and the arbitrator – but directly make the move.
2) If there are more than one victim, and only one attacking piece, we don't call the find attacker cycle, but call the arbitrator, to decide the best victim.
3) In cases where we find many victims and attackers, we call the restricted find attacker cycle and the arbitrator. By restricted find attacker cycle, we mean that, the find attacker is called only for checking if our attacking pieces are being attacked by any of the opponent's pieces.
4) In cases where there are no victims, we call the full-fledged find attacker cycle and arbitrator cycle to generate a move.



Parallel computation of 'inferior' variations as judged by the heuristic evaluator is bound to generate many exciting prospects that could make a mock of the heuristic evaluator. An example of this is given in our results.

IV. Implementation

To evaluate the proposed design, we used the existing open-source AntiChess agent provided by Source-Forge as the base to implement our enhancements.

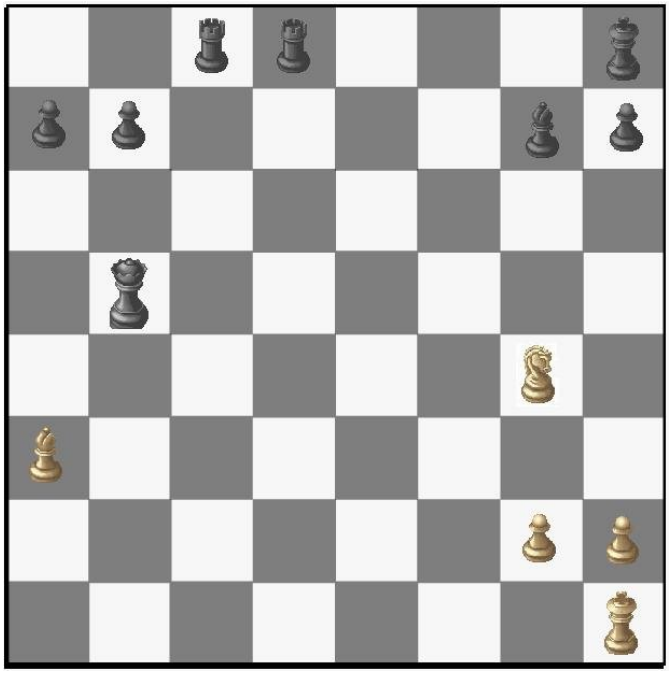
We adapted the basic alpha-beta search used in evaluation by incorporating singular extensions. We reduced the maximum depth of evaluation by 2 ply. Then, we identified singular moves as the moves having the highest heuristic (including ties). We then called a singular evaluation of these moves to use up the time we saved by reducing the evaluation depth.

In parallel, we simulated the operation of the designed hardware chips. We took into account approximate gate delays required by Deep Blue's chess hardware. Thus, we were able to determine the time taken for parallel evaluation of non-singular variations.

V. Results

We verified that the time consumed in quiescent search using the hardware chip to evaluate positions, in terms of gate delays, was much lesser than software-based quiescent search which involved machine cycles. This clearly gave a boost in performance in terms of speed of operation.

However, the significant improvement was with the efficiency in evaluation. Some of the moves which led to positions evaluated heuristically inferior, proved invaluable when we used our design, while the same move would have been discarded by the normal algorithm. We simulated the following position as arising after the 10^{th} ply ('n' in our case). While the heuristic evaluator discarded the sequence of moves that would have led to this position after 10 ply, the hardware chip simulator examined this possibility and detected a clear win for black (Black to move).



Here, white is considered clearly advantageous in terms of material (in AntiChess). However, with black to play, this sequence of moves was found by the hardware chip simulator.

1. ... Bh6 2. N*h6 Qb2 3. B*b2+ Rc3 4. B*c3+ Rd4
5. B*d4++ mates black and hence black turns out to be the winner as he has successfully lured white into mating him.

VI. Potential Applications

The research work done for AntiChess helps us model many real world situations before actually taking a decision. Most of these real world situations involve a number of forced moves. This kind of modeling using the optimizations for AntiChess helps us in taking optimal decisions in an environment with a lot of forced constraints unlike chess because the optimizations for chess consider a wide range of alternatives whereas AntiChess takes into account the inherent constraints of the problem.

We need to model stock markets before actually deciding on investing heavily in some company. In this case, there may be many forcing constraints such as a stock market crash or a fall in the prices of shares of the company because of competitors' actions. Even though the competitor takes a series of actions which makes the prices of our shares fall, an algorithm similar to our AntiChess application will be able to better provide us with a actions that we need to take in order to force the competitor on the back foot.

In business strategy evaluation, there are many cases where we can force an opponent to adopt a strategy. For example, in the telecommunications industry, a player entering newly into the market may keep his prices lower than the market average, as a result of which the player forces other players to reduce their prices too. Similar to this we can make a series of well planned moves which forces the opponent to make certain moves. Once we establish ourselves, we can then adopt innovative strategies to increase our turn over.

VII. Conclusion and Future Work

We have proposed and evaluated the parallel use of singular search algorithms and special-purpose hardware in improving the performance of the anti-chess agent. Specifically, our contributions are relevant in choosing alternatives in constrained environments which may arise in a competitive scenario.

We are working on extending the algorithm and design to support multiple competitors (4 player AntiChess, for instance) which increases the complexity exponentially.

Appendix

Rules of the 2-Player AntiChess game:

A move is defined by the following rules:

1. "White" moves first. The players alternate in making one move at a time until the game is completed.
2. A move is the transfer by a player of one of his pieces from one square to another square, which is either vacant or occupied by an opponent's piece.
3. No piece except the knight may cross a square occupied by another piece. That is, only the knight may jump over other pieces.
4. A piece played to a square occupied by an opponent's piece captures that piece as part of the same move. The captured piece is immediately removed from the board.

A player's moves are limited by the following fact: A player is forced to capture an opponent's piece whenever possible. If a player can take several of the opponent's pieces, he/she is free to choose which piece to take. This limitation does not exist in regular chess. All the pieces move exactly as they do in standard chess. The rules of AntiChess also allow the special moves of castling and en passant.

There are two additional moves that are native to AntiChess:

When a pawn moves to the last row on the opposing side, it turns into a queen. (In standard chess, such a pawn can be turned into any piece of the player's choice.)

Once per game, each player may switch the location of two of its pieces. This switch is considered a move.

The switch is subject to the same rules as all the other moves, e.g. you may not switch into a check. Each player is only allowed to perform this move once per game.

References

[1] F.-h. Hsu, "Chess Hardware in Deep Blue", IEEE Computing in science and engineering, January 2006, Vol.8 No.1, pp.50-60

[2] T.S. Anathraman, M.S. Campbell, and F.-h. Hsu, "Singular Extensions: Adding Selectivity to Brute Force Searching," *Artificial Intelligence*, No. 1, 1990, pp. 99-109.

[3] D.J. Slate and L.R. Atkin, "Chess 4.5—The Northwestern University Chess Program," *Chess Skill in Man and Machine*, P.W.Frey, ed.Springer-Verlag, 1977, pp. 82–118.

[4] AntiChess Project developer's Forum and Documentation - Sourceforge.net

[5] S. F.-h. Hsu, "*Large Scale Parallelization of Alpha-Beta Search: An Algorithmic and Architectural Study with Computer Chess"*, Tech. Report CMU-CS-90-108, Carnegie Mellon Univ., Pittsburgh, Pa., 1990.

[6] R. W. F.-h. Hsu, "A Two-Million Moves/s CMOS Single Chip Chess Move Generator," *IEEE J. of Solid-State Circuits*, No. 5, 1987, pp. 841-846.

[7] S. P F.-H. Hsu, "IBM's Deep Blue Chess Grandmaster Chips," *IEEE Micro*, vol. 19, no. 2, 1999, pp. 70–81.

[8] Steven Walczak, "Knowledge-Based Search in Competitive Domains", IEEE Transactions on Knowledge and Data Engineering, May 2003, pp.734-743