

The Chess Monster Hydra ^{*}

Chrilly Donniger and Ulf Lorenz

Universität Paderborn,
Faculty of Computer Science, Electrical Engineering and Mathematics
Fürstenallee 11, D-33102 Paderborn

Abstract. With the help of the FPGA technology, the boarder between hard- and software has vanished. It is now possible to develop complex designs and fine grained parallel applications without the long-lasting chip design cycles. Additionally, it has become easier to write coarse grained parallel applications with the help of message passing libraries like MPI. The chess program Hydra is a high level hardware-software co-design application which profits from both worlds. We describe the design philosophy, general architecture and performance of Hydra. The time critical part of the search tree, near the leaves, is explored with the help of fine grain parallelism of FPGA cards. For nodes near the root, the search algorithm runs distributed on a cluster of conventional processors. A nice detail is that the FPGA cards allow the implementation of sophisticated chess knowledge without decreasing the computational speed.

1 Introduction

The early chess programs tried to mimic the human chess style. In the 1970s Chess 4.5 [7], however, demonstrated that emphasizing the search speed might be more fruitful. Belle[1], Cray Blitz, Hitech, and Deep Thought[4] were the top programs in the 1980s. From 1992 on, PCs dominated the world of computer chess: ChessMachine, Fritz, Shredder etc. With one well known exception: In 1997, IBM's Deep Blue[3] won the historical 6-game match against Garry Kasparov. This highlight is still of singular quality, although the playing strengths of present top programs seem to cross the borderline beyond the strongest human players. Only one big point is vacant: The final, generally accepted, victory over the humans. The previous Computer Chess World Championship in Graz showed that the race will probably be performed by only four programs: The top four of that championship (Shredder, Fritz Junior, and Brutus/Hydra) scored more than 95% of the possible points against the further 12 participants.

Typically, a game playing program consists of three parts: a move generator, which computes all possible chess moves in a given position; the evaluation procedure implements a human expert's chess knowledge about the value of a given position (these values are quite heuristic, fuzzy and limited) and the search algorithm, which organizes a forecast: At some level of branching, the by the game defined game tree (the complete one) is cut. The artificial leaves at this top are evaluated by a heuristic evaluator, because the real values are usually not known, and these values are propagated to the root of the game tree, as if they were real ones. The astonishing observation over the last 40 years in the chess game, and some other games is: the game tree acts as an error filter. The faster, and the more sophisticated the search algorithm, the better the search results!

In professional game playing programs, mostly the Negascout [6] variant of the Alphabeta algorithm [5] is used. The Alphabeta algorithm and its variants, are depth

^{*} sponsored by: PAL Computer Systems, Abu Dhabi, <http://www.hydrachess.com>

first search algorithms, which use information collected in left parts of the search tree in order to reduce searching time in right parts. The form of the search tree, as well as the efficiency with that it is examined, strongly determines the quality of the search result.

Encoding the move generator, the evaluation procedure and the search algorithm in a hardware design has the following advantages: first, the procedures can be processed in a very few cycles such that the execution speed can significantly be increased. Second, on standard PCs there occurs a tradeoff between the search speed and the implementation of chess knowledge. This tradeoff does not exist in a hardware design. Most evaluation features can be processed in parallel. Therefore, we only need additional space, but no extra time. In contrast to fixed wired hardware, FPGA serves with the advantage that debugging is easier and any improvements can be added instantly.

In this paper, we discuss the design philosophy, general architecture and performance of one of the strongest chess programs in the world. Section two starts with a hardware overview. Then in section 3, we will discuss the software architecture. Firstly the FPGA related stuff, thereafter the parallel search algorithm on the PC side. Last but not least, Section 4 deals with experimental results.

2 System

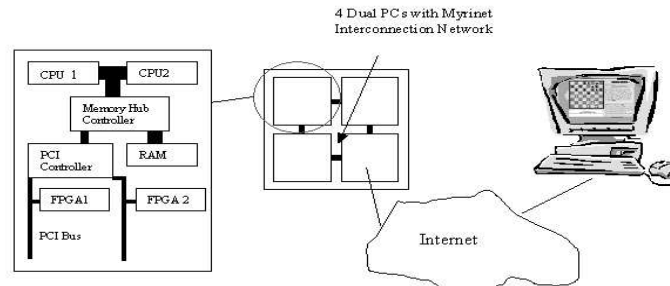


Fig. 1. The System Architecture: 4 Dual Pentium with 2 FPGA cards per node are interconnected with an end-user PC via the Internet.

Hydra uses the ChessBase/Fritz graphical user interface, running on a PC with WindowsXP. It connects with the help of a secure shell via the Internet to our Linux cluster, which itself consists of 4 Dual PC server nodes being able to handle two PCI busses simultaneously. Each PCI bus is supplied with one FPGA card. Each MPI-processes is mapped onto one of the processors and one of the FPGA cards is associated with it, as well. The server nodes themselves are interconnected by a Myrinet network.

3 Software Architecture

Hydra partially runs on PCs, and partially on FPGA cards. The reason is that the bottleneck of the system is the PCI bus. Therefore, on the one hand we cannot make the whole search run on the PC side. On the other hand, it is not clever to put all the search on the FPGA card, because it is much more difficult to develop a sophisticated search algorithm in hardware. We perform the last 3 plies of an n-ply search on the FPGA side, inclusively the quiescence search and all evaluations. Such a depth-3 search is initiated about 100.000 times per second per processor.

3.1 The FPGA

At the present, we use a Xilinx based VirtexV1000E card from Alphadata. We use 67 out of 96 BlockRAMs, 9,879 of 12,288 Slices, 5,308 of 12,544 TBUFs, 534 of 24,576 Flip-Flops, and 18,403 of 24,576 LUTs. We can run the FPGA with 33MHz, the longest path consists of 51 logic levels. An upper bound for the number of cycles per search node is 9 cycles. Hydra essentially contains a big piece of combinatorial logic, controlled by a finite state machine (FSM) with 54 states for the search.

The **evaluation** consists of many small features which are summed up by the help of one large adder tree. All features are simultaneously determined. Because of its structural simplicity, we refrain from further details.

A **move generator** is usually implemented in software as a quad-loop: One loop over all piece types, an inner loop over pieces of one type, a more inner loop for all directions in that the piece can move, and the most internal loop for the squares to which the piece can move under consideration of the current direction. This is quite a sequential procedure, especially, when we consider that e.g. taking moves should be sorted to the beginning of the move list. In hardware, there is a nice, fast and small move generator which works completely different. The move generator is, in principle, an 8×8 chessboard. The so called GenAggressor module and the GenVictim module, both instantiate 64 square modules, one for each square. Both determine to which neighbor square incoming signals have to be forwarded. The squares send piece-signals (if there exists a piece on them) resp. forward the signals of the far-reaching pieces. Additionally, each square can output the signal 'victim found'. Then we know that this square is the 'victim' (i.e. a to-square) of a legal move. The collection of all 'victim found' signals is the input for an arbiter (indeed a comparator tree), which selects the most attractive, not yet examined victim. The GenAggressor module takes the arbiter's output as input and sends the signal of a super-piece (a combination of all possible pieces). If e.g. the rook-move signal hits a rook of our own, we will find an 'aggressor' (i.e. a from-square of a legal move). Thus, many legal moves are generated in parallel. These moves must be sorted and we have to mask, which of them have already been examined. We sort the moves by the help of a comparator tree. The winner is determined within 6 levels of the tree. Sorting criteria are the values of attacked pieces and whether or not a move is a killer move. We refer to [3] for further details about this kind of move generator.

Figure 2 shows a simplified version of the finite state machine that controls the **search** on the FPGA card. The search works as follows. We enter the search at FS_INIT. If there is anything to do, and if a nullmove is not applicable, we come to the start of the full search. After possibly increasing the search depth (not shown in figure 2), we enter the state FS_VICTIM, where the output of GenVictim is inspected. If we find a to-square of a valid move and a futility cutoff is not possible, we will reach the state FS_AGGR, where the GenAggressor output is inspected, and if we find a from-square, we will make the next move and reach FS_DOWN. This corresponds to a recursive call of the Alphabeta algorithm with search window $[\alpha, \alpha + 1]$. If the search remaining depth is greater than 0, we start with looking for a move in the state FS_START. Otherwise we enter the quiescence search, which starts with the examination of the evaluation output. If the evaluation is not greater than alpha, we continue with a capture move, if available. If there is a piece which can be taken, we reach the QS_AGGR state, and if we additionally get a from-square by the help of the GenAggressor module, we will make a further move etc. Moves are unmade and we leave a recursion level, whenever

we reach the state QS_RETURN. A recursive algorithm like the Alphabeta algorithm

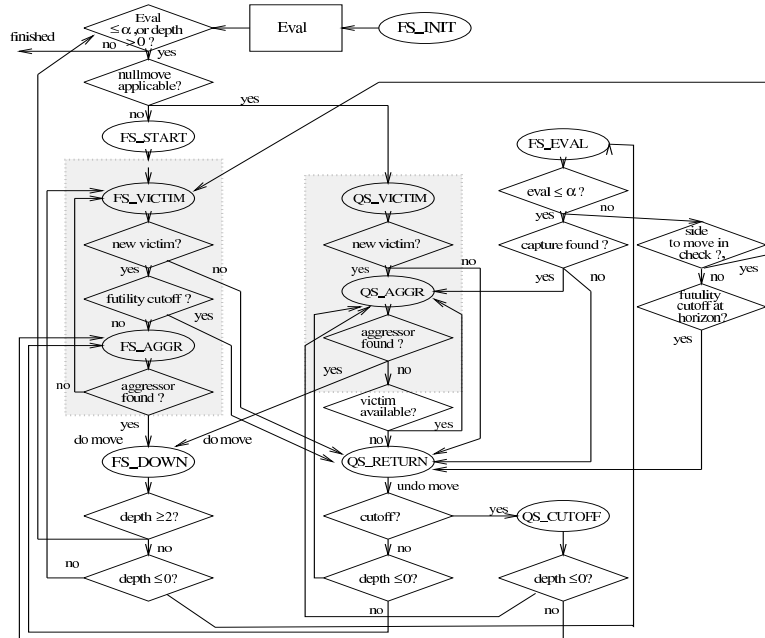


Fig. 2. Flowchart for the Finite State Machine for the Search.

needs a stack for its procedure. The stack in Hydra is realized by six blocks of dual port block RAM. The RAM is organized as 16-bit RAM. Thus we can either write two 16-bit data into the RAM, or one 32-bit word at one point of time. A ply variable which is controlled by the search FSM controls the data flow. Different tables capture different local variables of the recursive search.

3.2 The Distributed Search Algorithm

The basic idea of our parallelization is to decompose the search tree, to search parts of the search tree in parallel and to balance the load dynamically by the help of the work stealing concept. First, a special processor P_0 gets the search problem and starts performing the Negascout algorithm as if it would act sequentially. At the same time, the other processors send requests for work to other randomly chosen processors. When a processor P_i that is already supplied with work, catches such a request, it checks, whether or not there are unexplored parts of its search tree, ready for evaluation. These unexplored parts are all rooted at the right siblings of the nodes of P_i 's search stack. Either, P_i sends back that it cannot serve with work, or it sends such a node (a chess position with bounds etc.) to the requesting processor P_j . Thus, P_i becomes a master itself, and P_j starts a sequential search on its own. The processors can be master and worker at the same time. The relationships dynamically change during the computations. When P_j has finished its work (possibly by the help of other processors), it sends an answer message to P_i . The master-worker relationship between P_i and P_j is released, and P_j becomes idle. It again starts sending requests for work into the network. When a processor P_i finds out that it has sent a wrong window to one of its workers P_j , it makes a window message follow to P_j . P_j stops its search, corrects the window and

starts its old search from the beginning. If the message contained a cutoff, P_j just stops its work. We refer to [2] for further details.

4 Results

Experiments are performed on the hardware of the Paderborn University. Every processor is a Pentium IV/2.8GHz running RedHat Linux. We measure speedups of our program with the help of the BT2630 test set.

	time(s)	SPE	work %
1	24213	1	0
2	12139	1.99	0
4	6888	3.5	3.6
8	3488	6.94	-1

Speedups on BT2630

We compare the running times and the number of used search nodes of parallel Hydra with the one processor configuration. The speedup (SPE) is the sum of the times of the sequential version divided by the sum of the times of a parallel version. The search overhead is given in percent (work %) seen from 100% of the sequential version.

Tournaments and games: Hydra's predecessor Brutus reached the third rank on the Paderborn Computer Chess Championship in February 2003 and it has won the Lippstadt FIDE Grandmaster Tournament with 9 out of 11 points, reaching a performance of 2768 ELO, which is in the range of the human top players.¹ Internal test matches show that Hydra has made further progress: 140 test games with only 4 processors against Fritz8 and Shredder8, playing on a Pentium 2.4 GHz PC, showed an advantage of more than 110 ELO points. Last but not least, Hydra won the 13th International Paderborn Computer Chess Championships, in the presence of World Champion Shredder8 and Fritz8.

5 Conclusion

We presented the professional chess program Hydra, which is one of the top chess programs in the world. It uses the FPGA technology and combines the hardware design methods of Deep Blue with fully dynamic game tree search approaches. The FPGA technology seems to serve with a good compromise, as on the one hand it is possible to make use of hardware parallelism. On the other hand we have no long development cycles. In computer chess it is essential to have the possibility to frequently change and improve the program's code.

References

1. J.H. Condon and K. Thompson. Belle chess hardware. *Advances in Computer Chess III*, M.R.B. Clarke (Editor), Pergamon Press, pages 44–54, 1982.
2. R. Feldmann, M. Mysliwicz, and B. Monien. Studying overheads in massively parallel min/max-tree evaluation. *In proc. of SPAA'94*, pages 94–104, NY, 1994.
3. F-H. Hsu. Ibm's deep blue chess grandmaster chips. *IEEE Micro*, 19(2):70–80, 1999.
4. F-H. Hsu, T. S. Anantharaman, M.S. Campbell, and A. Nowatzyk. *Computers, Chess, and Cognition*, chapter 5 Deep Thought, pages 55–78. Springer Verlag, 1990.
5. D.E. Knuth and R.W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.
6. A. Reinefeld. *Spielbaum - Suchverfahren*. Springer, 1989.
7. 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, pages 82–118, 1977.

¹ The ELO-system is a statistical measure for the strength of chess players.