The Hydra Project

Hydra, currently the strongest chess program in the world, is a cutting-edge application that combines cluster computing with the fine-grain parallel FPGA world.

by Chrilly Donninger
Programmer
Nimzo Werkstatt OEG
c.donninger@wavenet.at

Ulf Lorenz
Researcher
Paderborn University
flulo@upb.de

The International Computer Games Association (ICGA) regularly organizes computer chess world championships. For quite a long time, large mainframe computers won these championships. Since 1992, however, only PC programs have been world chess champions. They have dominated the world, increasing their playing strength by about 30 ELO per year. (ELO is a statistical measure of 100 points difference corresponding to a 64% winning chance. A certain number of ELO points determines levels of expertise: Beginner = ~1,000 ELO, International Master = ~2,400, Grandmaster = ~2,500, World Champion = ~2,830.)

Today, the computer chess community is highly developed, with special machine rooms using virtual reality and closed and open tournament rooms. Anybody can play against grandmasters or strong machines through the Internet.

Hydra

The Hydra Project is internationally driven, financed by PAL Computer Systems in Abu Dhabi, United Arab Emirates. The core team comprises programmer Chrilly Donninger in Austria; researcher Ulf Lorenz from the University of Paderborn in Germany; chess grandmaster Christopher Lutz in Germany; and project manager Muhammad Nasir Ali in Abu Dhabi. FPGAs from Xilinx® are provided on PCI cards from AlphaData in the United Kingdom. The compute cluster is built by Megware in Germany, supported by the Paderborn Center for Parallel Computing. Taiwan is involved as well.

The goal of the Hydra Project is literally one of world dominance in the computer chess world: a final, widely accepted victory over human players. Indeed, we are convinced that in 2005, a computer will be the strongest chess entity, a world first.

Four programs stand out as serious contenders for the crown:

- Shredder, by Stefan Meyer-Kahlen, the dominating program over the last decade
- Fritz, by Frans Morsch, the most well-known program
- Junior, by Amir Ban and Shay Bushinsky, the current Computer Chess World Champion
- Our program, Hydra, in our opinion the strongest program at the moment

These four programs scored more than 95% of the points against their opponents in the 2003 World Championship.

Computational speed, as well as sophisticated chess knowledge, are the two most important features of chess programs. FPGAs play an important role in Hydra by harnessing the demands on speed and program sophistication. Additionally, FPGAs provide these benefits:

- Implementing more knowledge requires additional space, but nearly no additional time.
- FPGA code can be debugged and changed like software without long ASIC development cycles. This is an important feature of FPGAs, because the evolution of a chess program never ends, and the dynamic progress in computer chess enforces short development cycles. Therefore, flexibility is at least as important as speed.
- We can use a lot of fine-grain parallelism.

Technical Description

The key feature that enables computer chess programs to play as strong as – or stronger – than the best human players is their search algorithm. The programs perform a forecast: given a certain position, what can I do, what can my opponent do next, and what can I do thereafter?

Modern programs use some variant of the Alphabeta algorithm to examine the resulting game tree. This algorithm is optimal in the sense that in most cases it will examine only $O(b^{t/2})$ many leaves, instead of b^t many leaves, assuming a game tree depth of t and a uniform branching of b. With the help of upper and lower bounds, the algorithm uses information that it collects during the search process to keep the remaining search tree small. This makes it a sequential procedure that is difficult to parallelize, and naïve approaches waste resources.

Although the Alphabeta algorithm is efficient, we cannot compute true values for all positions in games like chess. The game tree is simply far too large. Therefore, we use tree search as an approximation procedure. First, we select a partial tree rooted near the top of the complete game tree for examination; usually we select it with the help of a maximum depth parameter. We then assign heuristic values (such as one side has a queen more, so that side will probably win) to the artificial leaves of the pre-selected partial tree. We propagate these values up to the root of the tree as if they were true ones (Figure 1).

The key observation over the last 40 years of computer chess data is that the game tree acts as an error filter. The larger the tree that we can examine and the more sophisticated its shape, the better its error filter property. Therefore, what we need is speed.

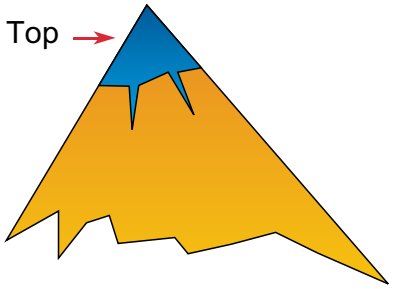


Figure 1 – Game tree search (in the blue part) leads to an approximation procedure.

The Hardware Architecture

Hydra uses the ChessBase/Fritz GUI running on a Windows XP PC. It connects to the Internet using ssh to our Linux cluster, which itself comprises eight dual PC server nodes able to handle two PCI buses simultaneously. Each PCI bus contains one FPGA accelerator card. One message passing interface (MPI) process is mapped onto each of

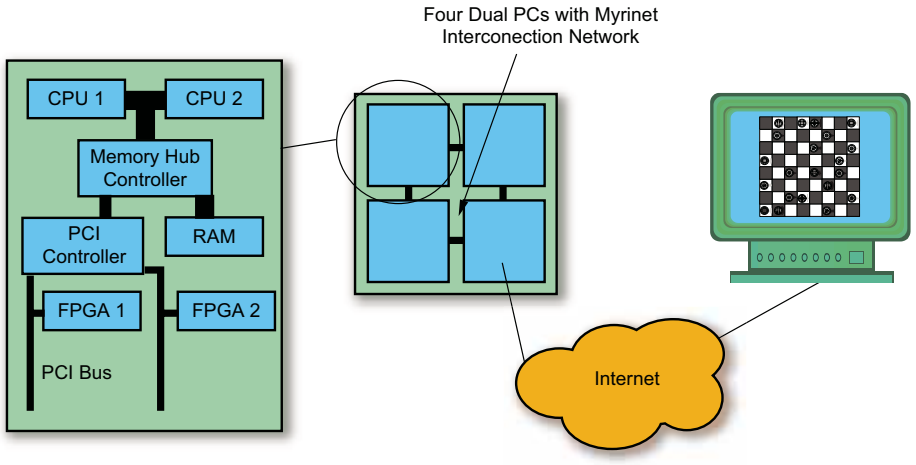


Figure 2 – A cluster of dual PCs supplied with two FPGA cards; each is connected to a GUI via the Internet.

the processors; one of the FPGAs is associated with it as well. A Myrinet network interconnects the server nodes (Figure 2).

The Software Architecture

The software is partitioned into two: the distributed search algorithm running on the Pentium nodes of the cluster and the soft co-processor on the Xilinx FPGAs.

The basic idea behind our parallelization is to decompose the search tree in order to search parts of it in parallel and to balance the load dynamically with the help of the work-stealing concept.

First, a special processor, P_0, gets the search problem and starts performing the forecast algorithm as if it would act sequentially. At the same time, the other processors send requests for work to other randomly chosen processors. When P_i (a processor 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. P_i sends back either a message that it cannot perform work, or it sends a work packet (a chess position with bounds) 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 relationship dynamically changes during computation. When P_j has finished its work (possibly with 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 processor P_i finds out that it has sent a wrong $\alpha\beta$-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," which indicates superfluous work, P_j just stops its work. We achieve speed-ups of 12 on the 16 cluster entities, which means that Hydra now examines 36 million nodes per second.

At a certain level of branching, the remaining problems are small enough that we can solve them with the help of a Configware coprocessor, benefiting from the fine-grain parallelism inside the application. We have a complete chess program on-chip, consisting of modules for the search, the evaluation, generating moves, and executing or taking back moves. At present, we use 67 block RAMs, 9,879 slices, 5,308 TBUFs, 534 flip-flops, and 18,403 LUTs. An upper bound for the number of cycles per search node is nine cycles. We estimate that a pure software solution would require at least 6,000 Pentium cycles. The longest path consists of 51 logic levels, and the design runs at 30 MHz on a Virtex™-I 1000. We have just ported the design to a Virtex XC2VP70-5 so that we can now run the program with 50 MHz.

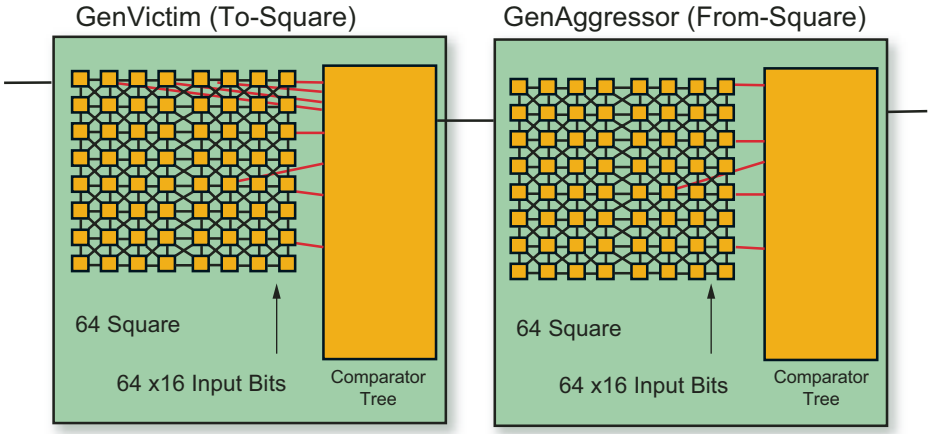
In software, a move generator is usually implemented as a quad-loop: one loop over all piece types; an inner loop over pieces of that type; a second inner loop for all directions where that piece can move; and the most internal loop for the squares to which the piece can move under consideration of the current direction. This is quite a sequential procedure, especially when we consider that piece-taking moves should be promoted to the front of the move list.

In a fine-grain parallel design, however, we have a fast, small move generator, which works very differently. In principle, the move generator consists of two 8 x 8 chess boards, as shown in Figure 3. The GenAggressor and GenVictim modules instantiate 64 square instances each. Both determine to which neighbor square incoming signals must be forwarded.

The square instances will send piece signals (if there is a piece on that square), respectively, forwarding the signals of far-reaching pieces to neighbor square instances. Additionally, each square can output the signal "victim found." Then we know that this square is a "victim" (a to-square of a legal move). The collection of all "victim found" signals is input to an arbiter (a comparator tree) that 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). For example, if a rook-move signal hits a rook of our own, we will find an "aggressor" (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 those already examined. The moves are sorted with the help of another comparator tree and the winner is determined within six levels of the tree. Sorting criteria is based on the value of attacked pieces and whether or not a move is a killer move.



1. Occupied squares send a signal in GenVictim
2. Free squares forward these signals
3. All squares receiving a signal are potential to-squares
4. Comparator tree selects most attractive to-square (taking moves).

1. Winner square generates signal of a super piece
2. Free squares forward the signals
3. Squares occupied by own pieces are potential from-squares
4. Comparator tree selects most attractive from-square

Figure 3 – The gen modules form the move generator.

Figure 4 shows the Finite State Machine for the recursive Alphabeta algorithm. On the left side of the figure you can see the states of the normal search, including the nullmove heuristic. The right side shows the states of the quiescence search. The main difficulty is controlling the timing, as some of the sub-logic (the evaluation and the move generator) may need two or three cycles instead of one. We enter the search at FS_INIT. If there is anything to do, and if nullmove is not applicable, we come to the start of the full search.

After possibly increasing the search depth (not shown in Figure 4), we enter the states FS_VICTIM and thereafter FS_AGGR, which give us the next legal move as described previously. Reaching the state FS_DOWN corresponds to a recursive call of the Alphabeta algorithm with a 1-point window (α,α + 1). If the search remaining depth is greater than zero, we look for a move in the state FS_START. Otherwise we enter the quiescence search, which starts with the evaluation inspection. In the quiescence search we only consider capture- and check-evasion moves.

The search stack (not shown) is realized by six blocks of dual-port RAM, organized as 16-bit wide RAMs. Thus, we can write two 16-bit words into the RAM, or one 32-bit word at one point of time. A depth variable controlled by the search FSM controls the data flow. Various tables capture different local variables of the recursive search.

Conclusion

We are quite optimistic that Hydra already plays better chess than anybody else. Nevertheless, we must now show this in a series of matches.

At the same time, we want to maintain the distance between Hydra and other computer players and even increase it. Therefore, in future versions of Hydra, we plan to switch to newer generations of Xilinx FPGAs, increase the number of processors further, and fine-tune the evaluation function.

For more information, visit *www.hydrachess.com* and *www.chessbase.com*.

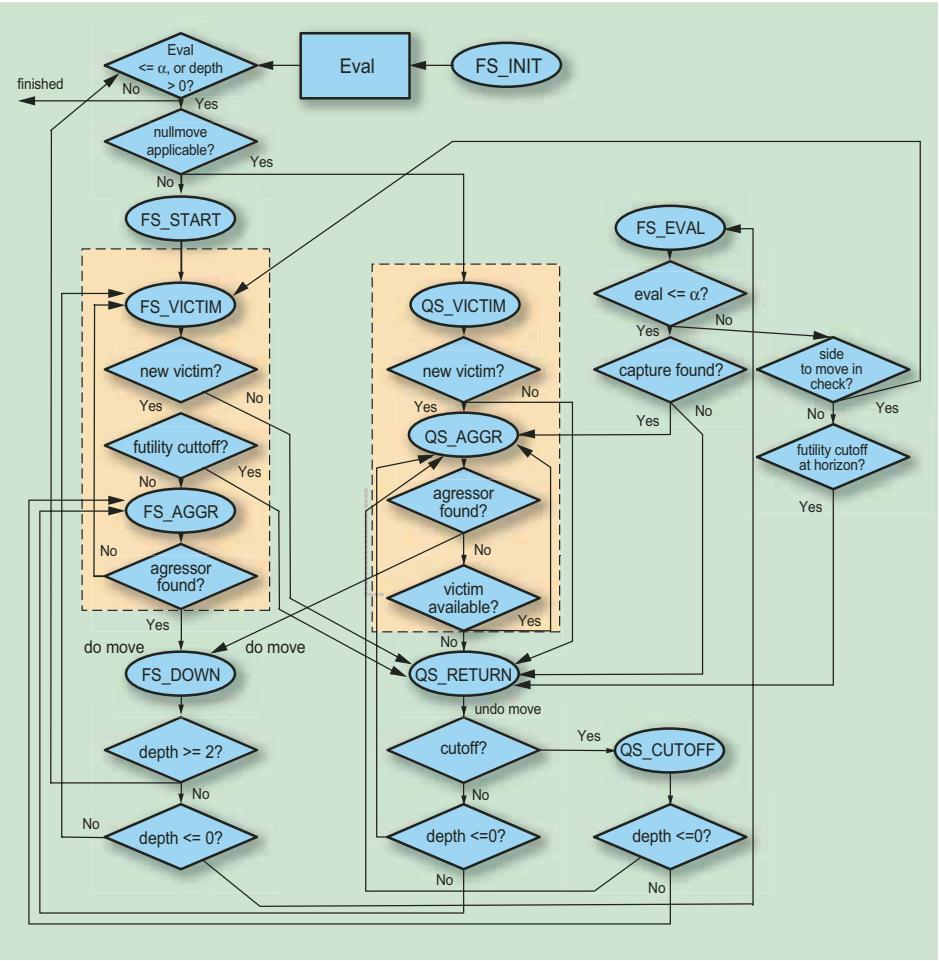


Figure 4 – Simplified flow chart for the 56-state FSM that operates the Alphabeta algorithm

History of Modern Computer Chess

- **1940-1970:** Programmers attempt to mimic human chess style, but resulting programs are weak
- **1970s:** Chess 4.5 is the first "strong" program, emphasizing tree search. It is the first program to win a tournament against humans (the Minnesota Open 1977)
- **1983:** Belle becomes National Master with 2,100 ELO
- **1988:** Hitec wins for the first time against a Grandmaster
- **1988:** IBM's Deep Thought plays on Grandmaster level
- **1992:** The ChessMachine, a conventional PC program by Ed Schröder, becomes World Champion.
- **1997:** IBM's Deep Blue beats Kasparov in a six-game match
- **2003:** Hydra's predecessor wins a human Grandmaster Tournament with 9 out of 11 points, reaching 2,768 ELO
- **February 2004:** Hydra achieves #1 rank in International Paderborn Computer Chess Championship
- **April 2004:** Hydra reaches 2,920 ELO on ChessBase chess server
- **August 2004:** Hydra scores 5.5:2.5 against Shredder and 3.5:0.5 against a 2,650 ELO Grandmaster
- **October 2004:** Hydra is crowned Machine World Team Champion against the human team, performing 2,950 ELO again