Frenchess: A Cray T3D at the 8th World Computer Chess Championship

Marc-François Baudot[1]

Jean-Christophe Weill[2]

Jean-Luc Seret[3]

Michel Gondran [4]

Abstract

This paper presents our work on parallel search algorithms at the University of Paris 8 and the Direction des Etudes et Recherches (E.D.F.).
It focuses on our parallel chess program "Frenchess" which, running on the Commissariat à l'Energie Atomique's 128 processors T3D, just finished third of a field of 24 at the 8th World Computer Chess Championship in Hong Kong (May 25 - May 30 1995).
Frenchess relies on the new ABDADA (Alpha Beta Distribué Avec Droit d'Aînesse) algorithm, an efficient and easy to implement parallel version of the alpha-beta algorithm (which was formerly known not to parallelize well). ABDADA had first been tested on a TMC CM5 and a Cray CS6400, and the adaptation to the CRAY T3D, using SHMEM, has proved very satisfying.
Even though single PE performance was disappointing, the T3D gave us excellent parallel performance, proving that our ABDADA algorithm scales very well up to at least 128 processors (with a speedup of over 70 in tournament conditions), and that it takes full advantage of the T3D's fast parallel architecture : it enabled us to build a world class chess program in just a few months' work. The ABDADA algorithm can certainly be useful for the parallelization of related problems, such as Branch and Bound problems, on this architecture.

Introduction

The Frenchess program is a joint research project on parallel search algorithms between the Direction des Etudes et Recherches (D.E.R.) of Electricite De France (E.D.F.) and the Paris 8 University, where Jean-Christophe Weill obtained his PhD Thesis on the subject of computer Chess[5][16].

Computer chess is not only a matter of prestige[6] : it is a good test-bed for parallel search techniques because search speed is of paramount importance in chess programs, and international competitions permit to judge the achievements of the contestants on a variety of hardware. Competitions are a very good motivation for the different teams working in this field to get the most out of their machines. The net result of the Frenchess project, up to now, is a new parallelization technique, ABDADA (which we will describe in the second section), that gives excellent performance for this class of algorithms, is most easy to implement, and, interestingly enough, is in its very nature fault tolerant. The ideas behind ABDADA are not limited to the

---

[1]XL, 33 rue des petits ruisseaux, 91370 Verrières-le-Buisson, France (e-mail: marc-francois.baudot@der.edf.fr)
[2]Université de Paris 8, 24 avenue du Pont Royal, 94230 Cachan (e-mail: weillj@lifl.fr)
[3]Direction des Etudes et Recherches, EDF, 1 avenue du Général de Gaulle, 92141 Clamart, France, (e-mail: jean-luc.seret@der.edf.fr)
[4]Direction des Etudes et Recherches, EDF, 1 avenue du Général de Gaulle, 92141 Clamart, France

[5]Jean-Christophe Weill and Marc-François Baudot have written several chess programs in the past few years. They have obtained the titles of Personal Computer Chess Champion and Amateur Micro-Computer Chess Champion in 1990 with the program Echec 1.9, and Amateur world Champion with Cumulus 2 in 1992.

[6]IBM's Deep Blue project is clearly aiming at the human world champion, Garri Kasparov. Thus one of our goals was to catch up with Deep Blue and then to beat them in the race to produce the best chess playing entity in the world. The results of the Hong Kong world championship show that it took us little time to nearly accomplish the former objective.

parallelization of the αβ algorithm. During this project, we have acquired a thorough understanding of the parallel machines used, a precious know-how with the contemporary emergence of massively parallel architectures.

Frenchess is a C program which uses PVM for the non speed critical sections and a few synchronisations during the search, and SHMEM where speed is important. Only integers are used, so flops will not be discussed here. It was first written for the CM5 (using CMMD and active messages on a 32 nodes CM5), then for a 32 processors Cray CS6400 (using PVM and shared memory) before being ported to the Commissariat à l'Energie Atomique's 128 PEs Cray T3D (starting in February 1995) with the 8th World Computer Chess Championship in view. This was a rather short delay, but Frenchess nevertheless tied for third place with IBM's Deep Blue (a parallel machine made of custom chess chips), which can be considered quite an achievement for a first participation in this very strong tournament. We will detail the porting to the T3D in the third section.

Our first section will give an overview of the 8th World Computer Chess Championship and its participants.

The 8th World Computer Chess Championship

The World Computer Chess Championship is held every three years, and is open to any kind of computer. This year it had to be limited to the 24 strongest entries, and was without a doubt the strongest computer chess competition ever. Each player is given 2 hours to play the first 40 moves. The best micros are not far from the strength of a Master at these time controls, and the small number of rounds (only five!) is evidently insufficient to draw any conclusion from the final standings in a tournament of such a high level : the quality of the games has to be taken into account.

Even without considering the luck factor, rather high with such a small amount of games, micros are not at such a disadvantage as could be expected when compared to the super computers :

- The best micros are commercial projects, and they usually have one or two full time programmers, a full time chess expert to write the opening book, professional testers and feedback from customers which help fine tune the program.
- The availability of powerful micros allows for continuous automatic testing which further helps the debugging and tuning of search heuristics and evaluation functions.

On the other hand, super computers have the following problems :

- Usually a small team must take care of programming, writing the opening book, testing and tuning the program.
- The availability of these machines, except for those that rely on special hardware (but then, even though the machine might be available 24 hours a day, it's only one machine...) is very bad, as they are usually heavily used for other tasks. Debugging and fine tuning the program cannot be done on slower computers, as search extensions and evaluations behave differently depending on the speed of the computer[7].
- The program has to be specially written for machines which evolve faster than the PC architecture.
- Parallelizing the αβ algorithm is, in itself, a difficult problem [16][12][13][4][5].

One of the most famous PC programs, Chess Genius, did not do very well and finished only sixth. Less known to the public, the parallel programs were :

- **StarSocrates**, developed at the MIT Laboratory for Computer Science, runs on a 1824 nodes Intel Paragon. Each node consists of two 50MHz I860 processors. It uses a work-stealing based search [11][12]

[7]Games are played in a limited time, so the faster the computer, the greater the search depth. The best way to use the computing power is not the same if you are using a PC or a CRAY : the most useful work is not the same at every depth. Simulating the behaviour of a CRAY on a PC would just take years to give enough useful results!

and finished second. They claim a speedup of 300 to 400 for 3648 processors. Despite this, their games were not impressive, with the exception of their good win over the PC program Rebel. For instance, their last round win against Hitech was only due to Hitech's aberrant endgame play.

- **Deep Blue Prototype**, from IBM, uses 14 dedicated chess search processors in parallel, each processor searching 400000 to 500000 positions per second. It was the clear favourite, but stumbled on a position which was objectively superior but very difficult to understand for a program[8]. Then it lost a game to the PC program Fritz (which eventually won the tournament), due to a bad hole in the opening book. The other 3 games it played were all very impressive. Deep Blue finished equal third with Frenchess (Frenchess searches around 500000 positions per second on the 128 processors T3D).

- **Zugzwang**, using the YBWC algorithm [6][4][3] on a 192 processors GC-Powerplus. Zugzwang tied for 6th place. They were disappointed with single processor speed (just like we were), but it seems their communication speed could not make up for that, so we feel we were much better of with the 128 processors T3D. Even though Zugzwang is now an old program, they had a nasty bug in the first rounds.

- **Phoenix 89**, using a 20 processor Sparc 2000 server and the DPVS algorithm [14], finished equal 14th.

- **Frenchess**, on the 128 PEs T3D exhibited a high level of play, but due to bad connection problems (which plagued all the remote machines), could not use its special tournament opening books[9]. It had to use a book designed to play against humans with risky variations, and lost a game to Chess Genius due to a bad opening line. The other games were more satisfying, and in the last round it crushed its DEC Alpha based opponent with a move that surprised its opponent and the IM and IGM spectators! Shortly after, when the game was over, indicating the move was a brilliant sacrifice and not a blunder, it was called the best move of the tournament by the arbiter. The algorithm parallelized is the αβ algorithm, but the new ideas which have been introduced should also be of great interest for related subjects such as branch and bound problems. The most important of those ideas, in the Frenchess project, is the ABDADA algorithm defined by Weill, which gives very good speedups (which were thought impossible just a few years ago) and is very easy to implement.

The ABDADA parallel minimax algorithm

The ABDADA search algorithm introduced in [16] is a parallel version of the αβ search algorithm[10][10]. For the reader interested in understanding the details of the ABDADA algorithm, we recommend first to become familiar with the αβ search algorithm[11], iterative deepening[12] and transposition tables[13] which are well known and important parts (but clearly not the only ones) of what make chess programs play so well [14][16]. However this is not necessary to understand the basic ideas.

ABDADA is based upon both the YBWC [7, 5, 6, 4, 3] and the αβ* [2] algorithms. From YBWC, it keeps the basic concept: parallel evaluation of successor positions of a game position is allowed if and only if the eldest

---

[8] This is the case of most heterogeneous material position : keeping the right pieces is often a difficult decision based on long term planning and pattern recognition. Deep Blue made the wrong decision after a well played game from both sides, and could only draw against the strong PC program WChess.

[9] ftp was just not working from Hong Kong. The reason the book had to be sent by ftp is that our opening book tools are PC programs. The opening book is specially designed against specific opponents, and next round's opponent is only known a day before the game.

[10] More precisely the negascout algorithm [16].

[11] It should be remembered that its efficiency depends on move ordering : searching the best move first is the goal of many heuristics found in chess programs.

[12] Rather than going directly for a depth n search, whose computing time cannot be known beforehand, a chess program first performs a depth 1 search, then a depth 2 search, and so on until its time control algorithm interrupts it. This has two very interesting advantages :

- Time control is easier : when the search is interrupted during a depth n search, we are sure that all the moves have been searched to at least depth n-1;
- Information is stored during depth n-1 search that will permit depth n search to order the moves efficiently. Iterative deepening is faster than searching directly at depth n.

[13] Information (i.e. score, depth of search and best move from that position) acquired during a depth i search are saved in a hash table. If a position is met twice through different paths, the second time its subtree will not be searched. Depth i+1 search, when reaching a position already searched at depth i, will first look at the move stored in the table, ensuring a better ordering of the search tree.

brother is fully evaluated. From αβ* it inherits the use of a shared transposition table to control the speculative parallel searches.

ABDADA is a particularly elegant way to parallelize the αβ search since it does not require derecursification of the αβ search, nor any division of search trees either statically or dynamically amongst processors, and no passing of sub-windows. It was proven to be easier and quicker to implement and to debug than related algorithms[16]. It is really close to the sequential algorithm, so all sequential search heuristics can be easily implemented.

One of the most efficient heuristics used in computer games is the transposition table[14] heuristic. In its most simple expression, it consists of a table of the results of positions already explored in the search[15], or of the positions seen in previous iterations when using iterative deepening . The results stored in the transposition table by a sequential chess program are the depth of the search from that position, the best move from that position, and the score so far.

The ABDADA[16] algorithm can be explained as follows:

1. Let `tt` be a shared transposition table. To the standard definition of the table we add, for each entry, a new field `nproc` which is the number of processors currently evaluating the node related to this entry;
2. All the processors begin the search simultaneously at the root of the game tree;
3. When a processor enters the evaluation of a position `P`, it increments the field `tt[P].nproc` in the transposition table;
4. When a processor leaves a position (because this position is fully evaluated or has been pruned), it decrements `tt[P].nproc` .

The analysis of a position is done in three phases:

1. The eldest son is analysed, regardless of the position of the other processors;
2. the other sons which are not currently analysed by other processors are analysed;
3. then analysis of the positions which are not totally evaluated (i.e. the corresponding entry in the transposition table is not set for the analysis depth) is done.

---

[14] Transposition tables are often called familiarly hash tables because they make use of a hash key.

[15] In Chess, it is not uncommon to reach twice the same position (i.e. the same pieces on the same squares with the same player to move) by different paths.

[16] ABDADA is the acronym of the french `"Alpha-Bêta Distribué avec Droit d'Aînesse" which can be translated as "Distributed Alpha-Beta Search with Eldest's Son Right" ("droit d'aînesse" meaning birthright).

This can also be expressed as the following :

```
1funct abdada(Position; α; β; depth; exclusiveP) ≡
2    var exclusive; iteration; alldone;
4     if depth= 0 then exit evaluate(Position); fi
5     Best ← -∞;
6     RetrieveAsk(Position; α; β; depth; exclusiveP);
8     Generate Move when waiting for the answer if using message passing....
10     GenMove(Position);
12     RetrieveAnswer(&α; &β; &Best);
14    The current move is not evaluated or if causing a cutoff or if we are in exclusive
15       mode and another processor is currently evaluating it.
17     if (α ≥ β) ∨ (Best= ON_EVALUATION)
18      then exit Best; fi
20     iteration ← 0;
21     alldone ← false;
23     while (iteration< 2) ^ (α< β) ^ ¬alldone do
24        iteration ← iteration + 1;
25        alldone ← true;
27        M ← FirstMove(Position);
29        while (M ≠ 0 ^ α< β) do
30           exclusive ← (iteration ≡ 1) ^ (NotFirstMove(M));
32           On the first iteration, only the leftmost son can be evaluated in parallel.
34           value ← -abdada(Position • M; -α; - max(β; Best); depth- 1; exclusive);
35           if value= -ON EVALUATION
36            then alldone ← false;
37           elsif value> Best
38              then Best ← value;
39                 if Best ≥ β then skip endsearch; fi
41           fi
42           M ← NextMove(Position) od ;
44     od
45     endsearch:
46     StoreHash(Position; α; β; Best);
47     exit Best.
```

Figure 1: The ABDADA parallel scheme on the αβ Search

The transposition table management procedures are shown in figure 2.

ABDADA has the following advantages :

- Easy and quick to implement: the main algorithm is very close to the sequential algorithm.
- Efficient for small and deep problems.
- Failure resistant: a time-out when waiting for transposition table answers ensures that the algorithm will return a correct answer even when only one processor is alive !

Its main drawback is the need for a very fast global transposition table. This is where all the implementation effort has to be made. A network of workstations using PVM will not offer sufficient communication speed for ABDADA.

Performance of the ABDADA algorithm will be discussed in the next section.

```
proc RetrieveAsk(Position; α; β; depth; exclusiveP)≡
   var entry; answer; end
   entry ← T (Position);
   answer.α← α; answer.β ← β;
   answer.score ← -∞;
   if entry = 0 then skip endprobe; fi
   if entry.height= depth^exclusiveP^entry:nproc> 0
    then answer.score ← ON EVALUATION
       skip endprobe    Only one processor allowed if exclusivity is required
   fi
   if entry.height ≥ depth
    then if entry:flag= VALID
        then answer.score ← answer.α ← answer:β ← entry.score
       elsif entry.flag= UBOUND ^ entry:score< β
         then answer.score ← answer.β ← entry.score
       elsif entry.flag= LBOUND ^ entry.score> α
         then answer:score ← answer.α ← entry.score
       fi
       if entry:depth= depth^answer.α < answer.β
        then                    Increment the counter of processors evaluating this node
          entry.nproc ← entry.nproc + 1
       fi
     else entry.depth ← depth;
       entry.flag ← UNSET;
       entry:nproc ← 1              First processor to evaluate this node
   fi
   endprobe:
   Send the answer :
   Send.ttableanswer(answer.α; answer.β; answer:score);
.
proc StoreHash(Position; α; β; score; depth)≡
   var entry; end
   entry  T (Position);
   if entry= 0 ∨ entry.height> depth then exit fi
   if entry.height= depth
    then entry.nproc ← entry.nproc- 1
     else entry.nproc ← 0
   fi
   if score ≥ β then entry.flag ← LBOUND
   elsif score ≤ α then entry.flag ← UBOUND
           else entry.flag ← VALID
   fi
   entry.score ← score;
   entry.depth ← depth;
```

Figure 2: The transposition table management for ABDADA

Porting Frenchess to the T3D

From the previous section, it is obvious that the critical part of Frenchess is the access to the global transposition table.

On the CM5, shared memory was simulated, each processor being in charge of a part of the global hash table; we had to rely on active messages to get some very good performances which CMMD could not give us otherwise.

This method of shared memory simulation has the advantage of making simultaneous accesses to a same entry impossible (this is important to ensure that ABDADA works correctly, i.e. that the number of processors is correct for a given transposition table entry). An average speedup of 16 for 32 processors was obtained on a standard set of test positions known as the Bratko Kopec positions[16].

On the Cray CS6400, shared memory was used, mutex (mutual exclusion) locks guarantying exclusive access to each hash table entry. Precise measurement of the speedup was not possible on this machine[17] but it looked very much like what we had seen on the CM5 even though single processor speed was around four times greater.

When we had to adapt Frenchess to the T3D, we did not find anything to replace mutex locks in MPP C! Such locks existed in CRAFT or on other CRAY systems (t_lock), but were not supported on MPP architectures. So we first decided not to use SHMEM and resort to message passing until we found a solution to this problem. Another reason why you might not want to use SHMEM is that, according to the documentation, it does not co-operate very well with the cache memory.

So we first tried an all PVM version. On the other machines we had used (including networks of workstations), the ABDADA algorithm was clearly too slow with PVM. On the T3D, PVM is so much faster that ABDADA was able to give some "interesting" (i.e. speedups and not slow-downs!) speedups but still, it would not have been wise to enter a world championship with this!

Then came the time to try setting channels between processors. Unfortunately we never really succeeded in making channels work correctly twice in a row (initialisation problems, coming maybe from a bad understanding of the documentation), and the world championship was not very far away!

So we turned to SHMEM again, and people working for the CEA and a Cray engineer pointed to us that mutex locks could be simulated using the shmem_swap call. Shmem_swap has the interesting property of being an atomic operation (this is what we missed when we were looking for a replacement to mutual exclusion locks).

It enabled us to simulate shared memory efficiently as shown below :

```
/*
 * locks a hash table entry on PE node, waiting for the entry
 * to be unlocked if necessary. The important property is
 * the atomicity of shmem_swap.
 *
 * In this example, shmem_swap replaces the value of
 * hashtable[entry].lockentry on PE node  with the value
 * of IsLocked on the current PE,
 * returning the value  hashtable[entry].lockentry on PE node had
 * before the swap.
 */

lock (int entry, int node)
{
  long IsLocked;

  IsLocked = 1;
  while (IsLocked == 1) {
    IsLocked = shmem_swap ( &(hashtable[entry].lockentry), IsLocked,
                            node);
  }
}
```

[17] The only way would have been to make sure that nobody else used the same partition as Frenchess. We did not try to negociate this as we knew we would be moving to the T3D soon and the CS6400 was rather heavily used.

```
/* unlocks a lock set with lock() */

unlock (int entry, int node)
{
      long IsLocked;

      IsLocked = 0;
      shmem_swap( &(hashtable[entry].lockentry), IsLocked, node);
}
```

It should be noted that SHMEM seems to be much more reliable than PVM or channels, especially under the debugger Totalview. We feel that SHMEM documentation[1] should be made less frightening, since it enabled us to get rather easily very high performance.

One last remark : when looking for maximum performance on the T3D, one should refrain from some calls such as flush, printf or time which generate calls to the front-end. It was observed that we were using a lot of the front-end's (C90) CPU, which seemed rather curious and penalised both the T3D and the C90. We found out that our time control algorithm called time several times per second (on only one PE). When we replaced time with clock, the problem did not exist anymore.

Parallel performance on the T3D

Each processor stored 1M transposition table entries (five 64 bits words per entry).

We have measured the loss of performance on the sequential speed (measured in positions visited per second) due to the use of SHMEM. Judging from the measures we have done since we returned from the World Championship, this loss does not change noticeably with the number of processors.

The figures in the following table are an average on positions representative of the different phases of the game (opening, middle game and endgame). The time is the total time for the tests in seconds, the speed given is the average speed reached at the end of the tests on a single PE. For each test, the time was the time needed to complete a fixed number of plies, so that each tested version did exactly the same search. All the hash table entries being held on one PE (either the one doing the search or an idle PE), we compared the speed reached when:

1. The processor uses its own memory to store the transposition table, and accesses it normally (no SHMEM).
2. Same as 1 but with the locks enabled (although it will never have to wait for a lock to be released since it is the only active processor).
3. The processor accesses its own memory using SHMEM, and locks are enabled.
4. The processor accesses another processor's memory (in a partition of 2 processors) with SHMEM and locks are enabled.

Case	**Time** (in seconds)	**Visited Positions per second**
1	1198	5071
2	1218	5001
3	1260	4825
4	1288	4778

Of course this would be meaningless without indicating the number of SHMEM calls. In these positions, the average number of shmem_swap per second was between 2600 and 3200, shmem_get ranging from 1300 to 1600 and shmem_put were in the range 950 to 1200 per second.

On the same positions, searched to the same depths[18] with 128 processors, we get 422000 to 855000 shmem_swap per second, 166000 to 253000 shmem_get per second and 81500 to 88000 shmem_put per second for middle game positions, and as much as 1124000 shmem_swap per second for only 41000 shmem_put/s and 118000 shmem_get/s for a pawn endgame position which, using only one processor, did not single itself out particularly[19]. One should note that these figures are different from the number of "loads" and "stores'" of hash table entries. Just searching for a hash table entry may require several shmem_swap (because an entry is already locked or because of the collision handling strategy. cf. [16]) and shmem_get, and just "loading" from the hash tables will also require calls to shmem_put. With no collision and no concurrent access to a same entry the number of shmem_swaps should be exactly double that of shmem_gets (one for locking, one for unlocking), which is what we observe for one PE.

When the two PEs (the one doing the search and the one holding the transposition table) are any two PEs in a bigger partition (but the only active ones), we have not been able to measure any performance difference. Of course, this does not imply that SHMEM communication speed between any two PEs in the T3D is exactly the same, but that, as stated in the documentation, communication speed in the T3D is very little dependent on distance. Trying to improve performance using partition shape, or, in the case were only some of the processors would hold parts of the transposition table, with the distribution of these processors, would most probably have been a waste of time[20]. Therefore we think that it is easier to get the best parallel performance possible on the T3D than on the CM5.

Cases 1 and 2 show that the calls to shmem_swap are almost not penalising (but with more than one processor they might imply waiting for locks to be released). Cases 3 and 4 compared to cases 1 and 2 show that the use of SHMEM gives roughly a 5% penalty, which is, in this context, a very good result.

Of course, when using more than one PE, things might be different, but the fact that the number of visited positions per second on a PE remain fairly stable when the number of PEs increases tends to show that SHMEM still performs very well and that locks are not a performance problem. This seems confirmed by the behaviour of the program at the World Championship, but precise measurements of parallel performance require large sets of positions and long search times, and these measures are not yet completed.

So the T3D seemed well suited for the ABDADA algorithm which proved very efficient on the 32 nodes CM5, but would the ABDADA algorithm still yield good speedups with 128 processors?

We have not had time to go through all the measures done on the CM5. In fact, it would have been much longer to get significant results on the T3D since it has a more of a faster processor : a problem of reasonable size on 128 nodes would become really long on one node, and we have to make enough measures to get a meaningful average.

We have used the 24 Bratko Kopec positions, searched to a fixed depth. We do not think that this set of positions is satisfying, but they are used by most other authors and give a basis for comparison. We have also used other test positions to verify our results.

We have first adopted the practical chess programmer's point of view. We have not measured speedups with a constant transposition table size : when using 128 processors the transposition table was 128 times bigger than when using only one processor[21]. We have first measured the speedup obtained for 128 processors for a depth 9 search, compared to 1 processor. Searches were performed with a simple evaluation, no extensions except check extensions, no futility cut-offs and no null move pruning so as to give results roughly comparable to [16].

depth	3	4	5	6	7	8	9
speedup	1.36	3.07	5.75	12.9	23.3	41.2	65.9

Speedup for 128 PEs compared to 1PE with 128 times smaller transposition table.

[18] which are rather shallow for 128 processors.

[19] These blocked pawn endgames, where transposition tables dramatically improve the speed of the sequential search, are somewhat of a problem for parallel searches. At depth 13, the speedup for the 128 processors was only 17 while it was searching 200 times more nodes per second than the 1 processor version.

[20] On the CM5, where shared memory was simulated by message passing using active messages, it was advantageous to dedicate some processors to the transposition table storage and retrieval while the others took care of the search.

[21] We would have had to use the whole machine for several hours just to measure the performance of one processor with as much memory as with 128 Pes. Doing it the other way around does not make much sense. Our problem (the world Championship) being a short term practical one, we spent machine time in a more useful way. For the same reason, we have not measured depth 10 searches with only one PE.

The speedup increases with the depth of search. A depth 9 problem is a small one for 128 processors, and the speedup is already very good. But depth 10 would take a very long time for only one processor, so the 128 processors were compared to 32 :

depth	3	4	5	6	7	8	9	10
speedup	0.98	1.03	0.87	1.00	1.32	2.03	2.29	2.95

Relative speedup of 128 processors compared to 32

When looking at these results, one must not forget that the speedup is also due to the increase in the transposition table size, which increases both ABDADA efficiency and, to a certain extent, single processor search speed (it increases the depth reached in a fixed amount of time, not the number of nodes per second).

Speedups with a constant hash table size have been measured with 128 processors compared to 32 :

depth	3	4	5	6	7	8	9	10
speedup	1.05	1.13	1.23	1.22	1.49	1.98	2.62	2.68

Relative speedup of 128 processors compared to 32 with a constant transposition table size

During a real game, where each player has 2 hours to play 40 moves, Frenchess searched most positions to depths 10, 11 or 12 (much more in simplified endgames). A conservative estimate for depth 10 speedup, in the same conditions as above, would be around 75 for 128 PEs. Thus we feel ABDADA gave us excellent performance on the 128 processors T3D.

Single PE performance

This was our big disappointment with the T3D.

Since the version of Frenchess on the CM5 was used for measuring performance, we kept it very stable over a long period of time. This was not possible with the CS6400 and T3D versions which were designed to obtain maximum performance from a chess point of view and had to be ready at very short notice (we started working on the T3D in February 1995 and the World Championship was in May 1995). Improvements in the evaluation function and to heuristics such as move ordering made each new version of the program difficult to compare with the last one from the point of view which interests us here.

Nevertheless, we have always had the possibility to switch back to the very simple evaluation and search extensions used on the CM5 version. This enabled us to make meaningful comparisons on the sequential speed of each version of Frenchess. The unit in which we measure the speed of Frenchess so as to be able to compare its performance on different architectures is the number of positions visited per second[22]. Here we give an estimate of the speed on one PE of Frenchess using a fast evaluation and simple extensions, which explains the differences between these figures and those given above.

- On a 50 MHz Sparc 10 (or a single processor of the Cray CS6400), the sequential algorithm was visiting around 20000-25000 positions/second. This was also the speed we obtained on 150 MHz DEC Alpha stations.
- When using shared memory for the transposition tables on the CS6400, this speed went down to around 12000-15000 nodes/second (we are not stating here that this decrease in speed is due only to shared memory accesses since the parallel hash table code is slightly more complex, involving mutex locks and more information than in the sequential version).
- We expected to get higher figures on the T3D, but the speed on one PE was in the 7000-9000 positions/second range!

On the other hand, we have seen that we were not losing much speed from the communications which makes up for these bad results.

---

[22]Or nodes per second. They do not mean anything when comparing two different programs. They also vary very much from one position to the other. Parallel numbers of nodes per second are even more difficult to interpret because of the amount of wasted parallel work

We think the poor single PE performance we noticed was primarily due to 2 problems:

1. cache problems (cache too small, cache invalidation?)
2. poor optimisations from the compiler. The only option we found to really improve program speed was... not using the -g (debugging) option!

We know from experience that this sort of program benefits a lot from big cache memory and advanced optimisations from the compiler, but we spent quite some time trying to improve single PE performance with only marginal results on the T3D.

Conclusion

We have shown that ABDADA still gets excellent speedups with 128 processors on the Cray T3D, and when comparing the behaviour of the T3D with that of the other machines we have used (32 nodes CM5, CS6400) for this class of parallel algorithms, we have found the T3D to be the easiest to use architecture.

Porting of Frenchess to the T3D only began in February 1995, and in May 1995 Frenchess finished third of the World Championship, which shows that

- ABDADA is well suited to the T3D. If that did not prove to be the case with a bigger number of processors, other (but more complicated to program) recent known methods could be used to get good results.
- the T3D was easy to program with excellent parallel performance using SHMEM once the problem of the mutex lock was solved.

It seemed to us that the Cray T3D was an excellent machine as far as communications are concerned, but that its sequential performance could be easily improved by adding more cache memory, improving the compiler and maybe using a faster Alpha processor. It was our understanding, from the documentation, that software was not yet in its final state, so our hopes of sticking to the same architecture and benefiting from better sequential speed seem realistic.

Acknowledgements

We wish to thank all the people at the C.E.A., E.D.F. and CRAY who have helped us very efficiently and have made it possible for us to be ready in time for the World Championship, particularly:

D. Beaucourt, C. Caremoli, L. Colombet, J-E Duboisset, J-M. Fieni, O. Goudard, R. Napoleone, G. Roth, Y. Souffez.

References

[1]R. Barriuso and A. Knies. *SHMEM User's Guide Revision 1.08*. Cray Research Inc., April 1994.

[2]V. David. *Algorithmique parallèle sur les arbres de décision et raisonnement en temps contraint. E'tude et application au Minimax*. PhD thesis, ENSAE, 1993.

[3]R. Feldmann. *Spielbaumsuche mit massiv parallelen Systemen.* PhD thesis, Fachbereich Mathematik / Informatik - Universitat GH Paderborn, 1993.

[4]R. Feldmann, B. Monien, and P. Mysliwietz. *A fully distributed chess program.* In D.F.Beal, editor, Advances in Computer Chess 6, pages 1-27, Chichester, England, 1991. Ellis Horwood.

[5]R. Feldmann, B. Monien, P. Mysliwietz, and O. Vornberger. *Distributed game-tree search*. ICCA Journal, 2(2):65-73, 1989.

[6]R. Feldmann, B. Monien, P. Mysliwietz, and O. Vornberger. *Distributed game tree search.* In V. Kumar, Kanal L.N., and P.S. Fopalkrishman, editors, Parallel Algorithms for Machine Intelligence and Vision, pages 66-101. Springer Verlag, 1990.

[7]R. Feldmann, P. Mysliwietz, and O. Vornberger. A *local area network used as a parallel architecture.* Technical report, University of Paderborn, 1986.

[8]F.-H. Hsu. *Large Scale Parallelization of Alpha-Beta Search ; An alorithmic and Architectural Study with Computer Chess*. PhD thesis, Department of Computer Science, Carnegie Mellon University, 1990.

[9]F.-H. Hsu, T. Anantharaman, M. Campbell, and A. Nowatzyk. *Deep thought*. In T.A. Marsland and J.H. Schaeffer, editors, Computer, Chess and Cognition, pages 55-78. Springer-Verlag, 1990.

[10]D.E. Knuth and R.W. Moore. *An analysis of alpha-beta pruning*. Artificial Intelligence, 6(4):293-326, 1975.

[11]B.C. Kuszmaul. *Synchronized MIMD Computing*. PhD thesis, Department of Computer Science, Massachusetts Institute of Technology, May 1994.

[12]B.C. Kuszmaul and Joerg C.F. *Massively parallel chess*. Technical report, Department of Computer Science, Massachusetts Institute of Technology, 1995.

[13]J.H. Schaeffer. *Distributed game-tree search*. Journal of Parallel and Distributed Computing, 6(2):90-114, 1989.

[14]D.J. Slate and L.R. Atkin. *Chess 4.5 - the northwestern university chess program*. In P. Frey, editor, *Chess Skill in Man an Machine*, pages 82-118. Springer-Verlag, 1977.

[15]Thinking Machines Corporation, Cambridge, Massachusetts. *CMMD Reference Manual* V3.0, May 1993.

[16]J.-C. Weill. *Programmes d'échecs de championnat : architecture logicielle, synthèse de fonctions d'évaluation, parallélisme de recherche*. PhD thesis, Université de Paris VIII, Saint-Denis (France), January 1995.