FACTA UNIVERSITATIS (NIŠ)
Ser. Math. Inform. Vol. 22, No. 2 (2007), pp. 175–188

THE METHOD OF THE CHESS SEARCH ALGORITHMS PARALLELIZATION USING TWO-PROCESSOR DISTRIBUTED SYSTEM *

Vladan V. Vučković

Abstract. This paper is concerned with the realization of the parallel computer chess application. The asynchronous parallel search algorithm uses two processors connected via distributed local network. The original solution is implemented and tested in author's chess application Axon. The standard approaches of parallelism use dual and quad server PC machine, which is expensive and rare compared with classical single processor PC machines. The applied method introduces a new class of simple and efficient algorithm of the parallelization, using standard single processor units connected via local 100 Mb or 1 Gbit networks. Compared with single processor search algorithms, the parallel algorithm significantly increases performance in test suites and practical play.

1. Introduction

The theory of the computer chess has been developing for above 50 years [7],[21]. From the first chess playing machines, which has chess strength of the weak club player software and hardware components have developed rapidly. Nowadays, one can play against grandmaster strength chess engine on cheap and commonly used PC machines. Theoretically, the fundamentals of chess program have three major segments: *move generator*, *evaluation function* and *search algorithm.* Decision tree [25] that is the base of search algorithm grows *exponentially* with factors depending of position, hash tables, number of pieces on the board . . . . If we suppose that on the first level of the decision tree one has node with normal 40 exists, on the second level it will be $40^2 = 1600$ nodes, on the third 40^3 it will be 64000 etc. It is obvious that number of nodes and processing time depends exponentially of the level – depth of the search tree. In theory, that effect is called *combinational explosion.* On the other hand, the quality of computer play strongly depends on depth of the decision tree so the effect of the exponential explosion limits the computer chess

Received April 20, 2007.
2000 *Mathematics Subject Classification.* Primary 91A80; Secondary 68T99, 91A05
*The author was supported in part by AMD Adriatic.

strength. There are generally two approaches to overcome this problem: *Shannon-type-A* (full-width approach) and *Shannon-type-B* (selective search). The first one tries to solve the problem by using the simple pruning mechanisms based on Alpha-Beta technique with idea to decrease maximally the time needed to compute one single node. This approach benefits maximally of the rapidly increasing speed of the modern CPU-s and also incorporates standard (cluster) parallelization. The second approach (Type-B) is concentrate on heuristic pruning based on relatively extensive portion of knowledge directly programmed into the evaluation or move generator function. This approach is very depended on the programmer skill and the quality of the implemented pruning, so the results could be very relative. On today's level of knowledge in this area, the best combination is to use near full-width approach in main searcher, and selective searcher in q-search procedure. The algorithms could be combined: *Alpha-Beta* [14], *Null Move* [3] *and PVS (NegaScout).*

After using most power single processor techniques, the next step for the increasing of the playing strength is parallelization. This paper has intention to investigate some un-orthodox methods of parallelization, if the other techniques are well implemented. The main pruning method is *PVS/Alpha-Beta/Null Move* and it is implemented in author's *Axon* application with some technical improvements. The two-processor machines then are connected via 100 MB LAN. The method of the distributed parallelization is efficient and could be easily upgraded with multiprocessor networks.

In this paper, started with Section 2, we discuss some standard – cluster approaches in parallelism. In Section 3 the basic principles of the distributed parallelism connected with method provided by the author is presented. In Section 4, the parallelization of the most important search algorithms is defined. Sections 5 and 6 describe implementation and experiment details of the elaborated method.

2. Standard Approaches in Parallelism

In this Section, a brief overview of the most important chess parallel systems will be exposed.

2.1. PC based parallel chess engines. The standard method (SMP) is using dual or quad PC server machines. This type of parallel machines could be classified as firmly coupled machines (MIMD) [15]. The connection between processes (engines) is through the commonly used operative memory, under OS control. Also some modern processors have embedded parallelization techniques in processor architectures. Intel PIV encapsulates multithreading and latest AMD processors have advanced multicore architectures. The leading PC operating systems (for instance Windows XP, Linux ...) also supports multiprocessor hardware. Almost all leading professional programs (Fritz, Shredder, Juniors) have multithreading or multiprocessor abilities. Also some amateurs or private chess engines are also multiprocessor based (Zappa, Diep, Ikarus, etc.). In some applications the number of processors are 8, 16, 32 or even 128. The commonly characteristic of those machines is running on the specialized parallel hardware or PC clusters. Also, communication problems

– transferring large parts of data among processors became limiting factor. The most successful parallel chess engines, which running on Internet computer chess tournaments are dual or quad, so one could say that optimal number of processors with classic method of parallelization is 2–4.

2.2. PC based parallel chess engines with hardware accelerators. The most prominent representative of this category is *Hydra* [4]. The *Hydra* system architecture consists of N node Linux cluster. Each node encapsulates dual INTEL PIV processor with 2 FPGA (VirtexV1000E or upgraded versions) hardware accelerators connected via 2 parallel PCI slots. The PC part of searcher handles mail search tree using NegaScout search algorithm, and the last 3 plys are computed by the FPGA hardware, completely with quiescence search and all evaluations. For the parallelization, system uses *The Distributed Search Algorithm* [5] that dynamically decomposes and allocates the parts of the search tree to the corresponding processors.

2.3. Non PC based parallel chess machines. The claim for increasing of the search speed could be satisfied only with parallelization and using of the specialized hardware accelerators. Following that strategy there are some very successful and dominant parallel chess machines in the 1980's like Belle [2], Hitech, Cray Blitz [9],[10] and Deep Thought [12]. Also some other software techniques were investigated on academic level. The well-known project is MIT's parallel chess program StarSocrates followed by Cilkchess based on MTD(f) search algorithm [23].

Probably, the most outstanding parallel chess machine was developed by the IBM Company, named *IBM Deep Blue* [13]. Deep Blue Computer is implemented as a 32-node IBM RS/6000 SP high-performance computer. Each node has a single micro channel card that contains 8 individual VLSI chess processors (specially designed IC's for chess computations) [13]. The total processing of the systems utilizes 256 chess processors and can evaluate 200,000,000 positions per second. The software was coded in C under the IBM AIX system. The system utilizes the MPI (message passing) method of parallelization that is standard for the IBM SP Parallel systems. IBM reports that the Deep Blue system had a parallel efficiency of around 30%. The IBM Deep Blue machine has successfully wins the match against the World Chess Champion Garry Kasparov in 1997.

3. The Basic Method for Distributed Parallelization

The recursive Alpha-Beta algorithm [7], [14], [21], also with its upgraded versions, is highly sequential and depending very much on dynamic values generated in the search process [11], [14]. The first efforts on Alpha-Beta algorithm parallelization [5, 8, 11, 16, 20, 24] directly leaded to the main problem: the transition of the very large parts of tree "knowledge" among processors in real time [6]. The early attempt on parallelization was the *Principal Variation Splitting (PVS)* [9, 17, 18, 19, 22, 24] algorithm and its modified version *Enhanced Principal Variation Splitting (EPVS)* [8], [11]. For two processors, PVS and EPVS speedups was about 80% and 90% compared to single processor one, according to [8]. Also, the interesting approach is using two different engines, one with classic and the other with only material evaluation (*Phoenix* [24]). The problem of unused processors is solved using the *Dynamic Tree Splitting Algorithm (DTS)* [19]. According to the researches, the speedups are satisfied up to 16 processors. The fairly obvious drawback in practice is the need to use very high profile computer systems with extremely wide memory bandwidth. The most of the research work was implemented and tested on supercomputers – 4 processor Cray XMP or 8 processor Cray YMP.

The difference between standard two-processor parallelization approach and the distributed approach is generated by the difference in hardware:

- In standard approach, the hardware is based on two processors using same motherboard and operative memory, under same OS [1], [21]. Communication is via shared central memory. Statistically these kinds of machines are only a few percentages in the full population of PC machines.
- In distributed approach, there is a network of single processor standard PC machines connected via 100 MB or 1Gb LAN. The single machines could be under different OS. The distributed applications communicate via network packages (TCPIP, UDP or others). The communication speed is a 100-1000 times slower compared to first category.

Each approach has advantages and shortcomings. The author's work is connected with the second category. The basic problem in distributed parallelization is real-time communication among processors because of limited communication channel. Also, network package communication is slow and could even be hazardous – if some package does not reach its destination. The aspiration of the work is to develop solid and stable low rate communication protocol and found the way to efficiently allocate two-processor load in real time.

The theoretical background of the implemented parallelization method is simple and it will be explain using basic Alpha—Beta search algorithm. Also, the method could be applied on other algorithms (PVS, NegaScout, Null-Move) without minimal modifications. To demonstrate the solution assume that the current position is given on the following diagram (Fig. 3.1):

The position is resumed from the grandmaster game - key move is winning Ng6! for White. Assume that this position is the root for the searching algorithm. When one analyzes position using *Axon Evaluation Tester* [25], the list of legal moves is generated (Table 3.1).

The list of moves is sorted and displayed together with their characteristically bits (weights.) For the programs that use numerical instead of heuristic move weights, the procedure is equivalent. *The basic rule for parallelization on two machines is to divide sorted root move list into two sub-lists.* The first sub-list contains move with *EVEN* indexes in table (0. H4H8, 2. H4F4, 4. E5C4, 6. F2F4, 8. H3G2 ...) and the second list includes moves with *ODD* indexes (1. H3E6, 3. H4G5, 5. E5G4, 7. A1C1, 9. E1E3 . . .). After that, each sub-list is distributed to corresponding processor. EVEN list is distributed to first processor and ODD list is

Table 3.1: Partial list of legal moves generated from Fig.3.1 position with their weights.

Num.	Move	Move Weights (Heuristic)
0.	H4H8	% . + ... a . r . p # i t
1.	**H3E6**	**% . . c . d . f a q . n p . . t**
2.	H4F4	. s . . . d . . a . . n p . i t
3.	**H4G5**	**. s . . . . . . a . . n p # i t**
4.	E5C4	. s . . . . . . a . . n p . i .
5.	**E5G4**	**. s . . . . . . a . . n . # . .**
6.	F2F4	. s . . . . . . a . . n . . i t
7.	**A1C1**	**. s . . . . . . a . . . p . i t**
8.	H3G2	. s . . . . . . a . . . p . i t
9.	**E1E3**	**. s . . . . . . a . . . p . . t**
10.	H3F5	. s . . . . . . a . . . . # . t
11.	**E1E2**	**. s . . . . . . a . . . . . . t**
12.	A1D1	. s . . . . . . a . . . . . . t
13.	**E1C1**	**. s . . . . p . i .**
14.	B2C3	. s . . . . p . i .
15.	**H3G4**	**. s . . . . # . t**
16.	G3G4	. s . . . . # . t
17.	**G1F1**	**. s . . . . t**
18.	A3A4	. s . . . . t
19.	**G1H2**	**. s . . . . t**
20.	G1G2	. s . . . . t
21.	**A1A2**	**. s . . . . t**
22.	A1B1	. s . . . . t
23.	**F2F3**	**. s . . . . t**
24.	G1H1	. s . . . .
25.	**E1B1**	**. s . . . .**
26.	B2C1	. s . . . .
27.	**E1D1**	**. s . . . .**
28.	E5D3	. s . . . .
29.	**E1F1**	**. s . . . .**
30.	E5F3	. s . . . .
31.	**H3F1**	**. s . . . .**
32.	H4H7	. . + . . d . . a . . . p # i t
33.	**H4F6**	**. . c . d . f a . r n p # i t**
34.	E5F7	. . . c . . . . a . r n p # i .
35.	**E5C6**	**. . . c . . . . a . r . p # i .**
36.	D4D5	. . . . m d . f a . . n p # . t
37.	**H4E4**	**. . . . . d . f a . . n p # i t**
38.	H4H5	. . . . . d . . a .
39.	...	....

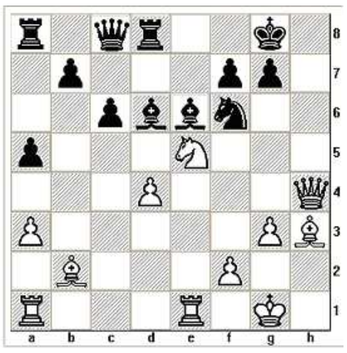

Fig. 3.1: Test position. White is on the move.

distributed to second processor. In the phase of computing, each processor runs parallel on different set of moves. It is obvious that move lists are complement, and that each processor computes only 1/2 of move in list. The position best move must be present in one of two lists. In the last example, the best move 43. E5G6 is placed in ODD list, so it will be found on second processor. The iterative deeping procedure will find the best move faster then single processor routine because of less number of passes through the root node list (22 instead of 44, in the example above). After the computation, the best move is choosing with the min-max method – as the move with better final evaluation between two candidates.

The main characteristic of the method is the static character of parallelization. The root move list is dividing just once, at the start of searching. After that, two processors work simultaneously and asynchrony, and no need to extensively change information – eventually a few bytes per second (information about the best move and evaluation). At the end of searching , generated by the master processor, the slave processor send final best move and evaluation.

4. The Parallelization of the Basic Search Algorithms

The method of the basic search algorithm parallelization will be described on standard Alpha-Beta procedure and applied to the PVS and Null-Move procedures.

4.1. Standard AlplhaBeta search algorithm. The following listing shows the source code of the single processor Alpha-Beta searcher (code is in Pseudo-C):

```
int AlphaBeta (int depth, int alpha, int beta)
{ if (depth == 0) return Evaluate();
GenerateLegalMoves();
while (MovesLeft())
{
```

```
MakeNextMove();
val = -AlphaBeta(depth-1,-beta,-alpha);
UnmakeMove();
if (val >= beta) return beta;
if (val > alpha) alpha = val;
}
return alpha; }
```

The searcher passes through the all-legal moves in move-list in each iteration. The move ordering is very important for the efficiency of the algorithm. If the best moves are on the leading positions of the list, the number of Alpha-Beta cutoffs is maximized, so the searching process is more efficient. If root node is processing, we suppose that variable *depth = max_depth*, so the instruction G*enerateLegalMoves()* must be altered with the instruction *GenerateSplitedLegalMoves(int processor)* where variable *processor* identifies processor (0 or 1 the for first or second processor). Procedure *GenerateSplitedLegalMoves* then generates legal moves with EVEN indexes if processor=0, or moves with ODD indexes if processor=1. In that way, each processor uses the same search procedures with different input parameters. The new procedure could be defined as:

```
GenerateSplitedLegalMoves (int processor);
{ int index=0;
While(AllLegalMoves())
{ if (index mod 2)=processor GenerateLegalMov;
index++;
} }
```

4.2. Parallel Alpha-Beta search algorithm. The method of parallelization of Alpha-Beta is concern with splitting the move list in two different parts, processed by each of two processors in system. The body of the search procedure differs only in one program line :

```
int ParallelAlphaBeta (int depth, int alpha, int beta, int processor, int max_depth)
{ if (depth == 0) return Evaluate();
if (depth=max_depth)
then GenerateSplitedLegalMoves ( processor);
else GenerateLegalMoves();
while (MovesLeft()) {
MakeNextMove();
val = -ParallelAlphaBeta(depth-1,-beta,-alpha, processor, max_depth);
UnmakeMove();
if (val >= beta) return beta;
if (val > alpha) alpha = val;
}
return alpha; }
```

Each processor runs the same Alpha-Beta search procedure with different input parameter for processor: For instance, if one calls searcher at depth (ply) 12, the calling commands will be:

- Master : Call with command: *ParallelAlphaBeta (12,-infinite,+infinite,0,12)*
- Slave : Call with command: *ParallelAlphaBeta (12,-infinite,+infinite,1,12)*

4.3. Parallel principal variation and null-move search algorithm. With analogue to the Alpha-Beta procedure, the parallelization of the PVS and Null-Move could be performed altering the *GenerateLegalMoves()* command with **conditional command:**

```
if (depth=max_depth)
then GenerateSplitedLegalMoves ( processor);
else GenerateLegalMoves();
```

4.4. The generalization of the parallel algorithm. The described algorithm could be easily generalized to the N (N>2) processor system. Instead of parameter 2 for list dividing, one can introduce variable *number_of_processors* as the parameter. The generalized profile of the *GenerateSplitedLegalMoves* could be:

```
GenralizedGenerateSplitedLegalMoves (int processor, int number_of_processors);
{ int index=0;
While(AllLegalMoves())
{ if (index mod number_of_processors)=processor GenerateLegalMov;
index++;
} }
```

In extreme situation, if the number of processors in system is equal to the number of root list moves (for instance 44 processors for the example Fig. 1), each processor calculates only one legal move and one extra ply at the same amount of time could be achieved, compared with single processor searcher.

5. Single Processor Unit – AXON engine

In the following sections, some of the implementation details will be presented. The practical parallel system is developed around the author's single processor system *Axon.* The program, running on the Athlon 2200+ PC machine, develops +2500 Elo rating – plays with the strength of the intermaster chess player. The *Axon* single processor unit is the solid base for researching of the parallel system. Of course, the playing strength of the single processor unit could be increased using the hardware acceleration (overclocking) or the advanced processor, but with limited hardware resources the parallelization is the only way to increase calculation power. For the experiment dual processor machine *(Parallel Axon)* was developed. Structure of the parallel *Axon* chess machine is simple, using standard master-slave approach for the similar dual processor organizations. Master computer organizes all functions like the single processor unit: interface, opening tablebase management, PGN management, support functions... In the phase of calculation, the slave engine that is equipped only with endgame tablebases, resumes the half of the root moves list and simultaneously calculates the other half of the decision tree. Two machines communicate in real time via 100Mbit standard LAN network using short messages with the frequencies 2-3Hz. The information, which is transferred through the communication channel, is enough to synchronize two computers. Two best moves, derived from the both computers are compared, and the best of them is play on the chessboard against the opponent. The acceleration compared with the single processor unit is gained due to the divided move lists processed by the both computers.

5.1. Implementation details. The standard author's version of chess program *Axon* was modified, using relatively simply method of parallelization. The experimental parallel system uses two PC computers connected via local 100 MB network. The program was parallelized using *master-slave* methodology. Two identical programs compute different branches of the decision tree. Also, human operator has control only on the master machine where is possible to change position using standard *Windows GUI* interface. In the phase of computing, master generates small packages of data using distributed connection. The package contains current position, flags, game history and list of moves for the processing on the slave unit. The key problem is how to split the total list of moves, to achieve optimal ballast of the processor strength in this multiprocessor environment. The presumption is that the speed, measured by the *Axon Benchmark 4* hardware test, must be approximately equal.

6. Experiment and Test Results

To determine the factor of parallel efficiency of the new method, the experiment was carried out. The standard EPD test was used *(yazgac.epd).* Otherwise the EPD tests are commonly used as the benchmark tests for computer chess algorithms. They provide serious of test chess positions with key moves. The experiment consisted of 3 phases. First, test was performed on single processors on ply 7 for each position. For the experiment, the PC with *AMD Sempron 2600+* processor/ 256 Mb Ram/ 50 Mb Hash was used. After that, the same experiment was performed on two machines working parallel, also with same hardware. The goal of the experiment was to represent the acceleration of the best move searching on parallel machine compared to single-processor one. The experimental date could be systematized in Table 1. Each position from the EPD test is represented with the corresponding number of nodes generated on ply 7. Each raw in table contains data collected from the Master and Slave processor, and also control number of nodes generated by the single processor. Bolded data remarks best choice of two processors. Node saving column is calculated as:

$$\textit{Node\_saving(\%) = (single\_number\_of\_nodes-min\_number\_of\_nodes) /} \\ \textit{single\_number\_of\_nodes (\%)}$$

Time saving is in direct correspondence with node saving, because of constant NPS factor.

The results indicate that parallel system found key move faster - in less number of nodes for every test position. The percentage of node saving is very dependent of the analyzed position. For the *yazgac.epd* test suite, the savings are in range 0.58%-91.3%. The average note saving percentage for all the 33 positions is *25.5%.* The theoretical reasons for these empiric conclusions are connected with the fact that the search tree is divided in two separate branches, and the information could not be transferred between two processors.

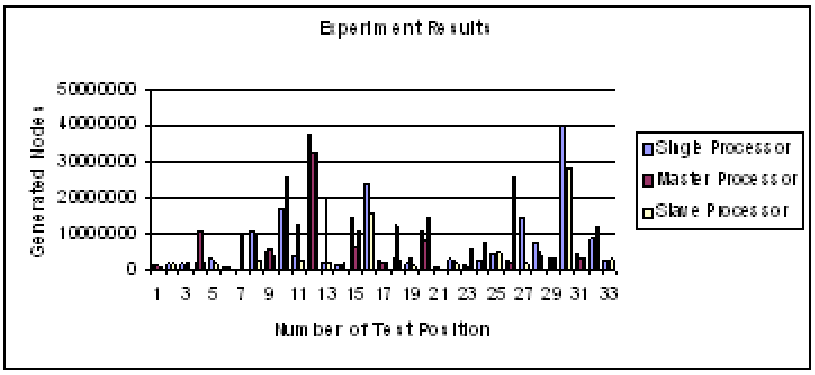

FIG. 6.1: Diagram shows result distribution with single and dual, master-slave configuration.

Normally, on single processor or SMP machines that information is commonly delivered through the memory using transposition tables or history hash tables. Each sub-tree contains less elements then full tree, which is also dependent on position. For instance, in raw 30, total tree search on key move Re4 for the single processor holds 40308515 positions. The second (slave) processor, which founds the

Table 6.1: Table shows number of positions generated with single and dual system on depth 7.

Num.	Single	Nodes	Master	Nodes	Slave	Nodes	**Savings**
1	Bh7	728249	**Bh7**	**717922**	bxc3	596521	1.4 %
2	Rxg7	1466745	**Rxg7**	**759523**	Qd3	1512966	48.2 %
3	Rd4	1525525	Be7	1031013	**Rd4**	**1469002**	3.7 %
4	Nc5	1564616	Nxf6	10863177	**Nc5**	**1555512**	0.58 %
5	Bd6	2597391	**Bd6**	**1388059**	Rf1	1381623	46.6 %
6	Rc1	276446	Rf4	497102	**Rc1**	**201065**	27.2 %
7	Qd8	84001	Qd3	9823036	**Qd8**	**56237**	33.1 %
8	Qc7	10527566	**Qc7**	**9189599**	Bxf7	2148993	12.7 %
9	Qxg6	4963938	Qh3	5771366	**Qxg6**	**3760483**	24.2 %
10	Nh4	16605243	**Nh4**	**15273693**	Bg5	25709788	8 %
11	Ne5	3689251	Rh8	12246408	**Ne5**	**2188589**	40.7 %
12	O-O-O	37440614	**O-O-O**	**32279825**	Qa4	32454430	13.8 %
13	Qxh3	2050870	dxc4	19446143	**Qxh3**	**1711382**	16.6 %
14	Nxb8	1115403	**Nxb8**	**1032344**	Qd3	1335819	7.4 %
15	Bxg7	14547957	**Bxg7**	**6283498**	Bg5	10789489	56.8 %
16	a4	23997961	**a4**	**22423349**	h4	15604024	6.56 %
17	f5	2159123	**f5**	**2076168**	Nxd5	1212576	3.8 %
18	Qxg5	2690807	e5	12102548	**Qxg5**	**2382803**	11.4 %
19	Nc4	1466950	Nb5	3312836	**Nc4**	**839488**	42.8 %
20	Nc3	10621431	**Nc3**	**7965793**	Ra5	13809770	25 %
21	Kc3	346605	**Kc3**	**196995**	Kc2	177727	43.2 %
22	Bxe2	2965166	Rh2	2151353	**Bxe2**	**1580737**	46.7 %
23	h7	724166	**h7**	**487909**	Bc6	5719170	32.6 %
24	Rxh3	2243812	**Rxh3**	**2138494**	Rg3	7257178	4.7 %
25	c4	4190791	**c4**	**2969132**	h4	4569494	29.2 %
26	Rxd7	2520953	**Rxd7**	**2032580**	Bg3	25238813	19.4 %
27	Qxc6	14345697	Ne1	12422980	**Qxc6**	**1243715**	91.3 %
28	Kh2	7423320	**Kh2**	**5258905**	Qxa7	3803816	29.2 %
29	Rxf4	2872671	Qxf4	527141	**Rxf4**	**2754258**	4.1 %
30	Re4	40308515	Qh5	14897082	**Re4**	**28140075**	30.2 %
31	cxb5	4005703	**cxb5**	**2622334**	Rd1	3001105	34.5 %
32	Re1	8435525	**Re1**	**5680953**	Rd1	11814379	32.5 %
33	Qe2	2447903	**Qe2**	**2128323**	Rf3	2928657	13.1 %

right solution, processed only 28140075 nodes, that is *69.8%* of full tree, and the saving factor is 30.2%. Experiment data could also be presented with diagram (Fig. 6.1):

In all test positions *min(master_nodes,slave_nodes)<single_nodes,* dual system achieves better test results compared to the single processor one.

7. Conclusion

This paper is concerned with the parallelization of the basic search algorithms. The modifications of the existent search algorithms are simple and they could be altered to single or multiprocessor mode, according to the available hardware. The applied method of parallelization differs from the standard SMP solutions because it has no need to use dual or quad machines. Instead, system uses affable single processor PC configuration, connected via standard LAN networks. The basic variant uses two-processor configuration but it will be easily expanded to N processors using generalized method described in Section 4.4. There is no need to develop special SMP code; the chess strength increasing is being achieved using available software and hardware components The software component is developed around the author's *Axon I* chess engine. The EPD test performed in this paper proves that parallel machine finds key moves faster in every case compared to the single processor one. In practical testing or playing mode dual machine reacts more rapidly then the single one, which is very important in low time control computer tournaments.

The future work will be directed in experiments with several processors, up to 16 machines LAN connected.

REFERENCES

1. S. Akl, D. Barnard and R. Doran: *The Design, Analysis and Implementation of a Parallel Alpha-Beta Algorithm*, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-4, (2), (1982) 192-203.
2. J. H. Condon and K. Thompson: *Belle Chess Hardware*, Advances in Computer Chess III, Pergamon Press, (1972) 44-54.
3. C. Donninger: *Null move and Deep Search: selective-search heuristics for obtuse chess programs* ICCA Journal, Vol. 16, No. 3, (1993) 137-143.
4. C. Donninger and U. Lorenz: *The Chess Monster Hydra*, Univesitat Paderborn, Faculty of Computer Science, Electrical Engineering and Mathematics, Paderborn, Germany, www.hydrachess.com, 2004.
5. R. Feldmann, B. Monieni, P. Mysliwietz and O. Vornberger: *Distributed game tree search*, in Parallel algorithms for machine intelligence and pattern recognition, 1990.
6. R. Finkel and J. Fishburn: *Parallelism in Alpha-Beta Search*, Artificial Intelligence (1982) 89-106.

7. P.W. Fray: *An introduction to Computer Chess. Chess Skill in Man* and *Machine*, Texts and monographs in computer science, Springer-Verlag, New York, N.Y. , 1977.

8. R. Hyatt: *A High-Performance Parallel Algorithm to Search Depth-First Game Trees*, Ph.D. Dissertation, University of Alabama at Birmingham, 1988.

9. R. Hyatt, A. Gower and H. Nelson: *Cray Blitz*, Advances in Computer Chess 4, Pergammon Press (1986) 8-18.

10. R. Hyatt, H. Nelson and A. Gower: *Cray Blitz - 1984 Chess Champion*, Telematics and Informatics (2) (4), Pergammon Press Ltd. (1986) 299-305.

11. R. Hyatt, B. Suter and H. Nelson: *A Parallel Alpha/Beta Tree Searching Algorithm*, Parallel Computing 10 (1989) 299-308.

12. F.H. Hsu, T.S. Anatharaman, M. Campbell and A. Nowatzyk: *Deep Thought*, Computers, Chess and Cognition, chapter 5, Springer Verlag (1990) 55-78.

13. F.H. Hsu: *IBM's Deep Blue Chess Grandmaster Chips*, IEEE Micro 19(2), (1999) 70-80.

14. D. Knuth and R. Moore: *An Analysis of Alpha-Beta Pruning*, Artificial Intelligence 6 (1975) 293-326.

15. B. Kuszmaul: *Synchronized MIMD Computing*, Ph.D. Thesis, MIT, 1994.

16. T. Marsland and M. Campbell: *Parallel Search of Strongly Ordered Game Trees*, ACM Computing Surveys (4) (1982) 533-551.

17. T. Marsland and M. Campbell: *Methods for Parallel Search of Game Trees*, Proceedings of the 1981 International Joint Conference on Artificial Intelligence, 1981.

18. T. Marsland, M. Campbell and A. Rivera: *Parallel Search of Game Trees*, Technical Report TR 80-7, Computing Science Department, University of Alberta, Edmonton, 1980.

19. T. Marsland, M. Olafsson and J. Schaeffer: *Multiprocessor Tree-Search Experiments*, Advances in Computer Chess 4, Pergammon Press (1986) 37-51.

20. T. Marsland and F. Popowich: *Parallel Game-tree Search*, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-7, (1985) 442-452.

21. T. Marsland and J. Schaeffer: *Computers, Chess and Cognition*, Springer-Verlag, 1990.

22. M. Newborn: *A Parallel Search Chess Program, Proceedings*, ACM Annual Conference, (1985), 272-277.

23. A. Plaat: *MDF(f) A Minmax Algorithm faster the NegaScout*, Vrije Universiteit, Mathematics and Computer Science, Amsterdam The Netherlands http://theory.lcs.mit.edu/~plaat/mtdf.html, 1997.

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

25. V. Vučković: *Decision Trees and Search Strategies in Chess Problem Solving Applications*, Proceedings of the Workshop on Computational Intelligence: Theory and Applications, University of Nis, (2001) 141-159.

Faculty of Electronic Engineering
Computer Department
P.O. Box 73
18000 Niš, Serbia
`vld@elfak.ni.ac.yu`