The Realization of the Distributed Computer Chess System

Vladan Vučković

Abstract - **This paper is concerned with the implementation of the asynchronous parallel search algorithm using 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 author's solution introduces a new class of simple and efficient algorithm of the parallelisation, using standard single processor units connected via local 100Mb or 1Gbit networks. Compared with single processor search algorithms, the parallel algorithm significantly increases performance in test suites and practical play.**

Keywords - **Computer chess, search algorithms, parallel computing, local area networks.**

I. INTRODUCTION

The theory of computer chess is complex connecting many sub-domains like theory of games, decision trees, and theory of programming, operation research, and optimization. The nature of the computer chess could be explained very simply: namely, decision tree that is the base of machine chess-playing 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 30 exists, on the second level it will be 302=900 nodes, on the third 303 it will be 27000 etc. It is obvious that number of nodes, also with processing time depends exponentionaly 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 strength.

There are generally two approaches to overcome this problem: *Shannon-type-A* (full-width approach) and *Shannon-type-B* (selective search) **[1]**. The first one tries to solve the problem by using the simple pruning mechanisms based on Alfa-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) parallelisation (IBM Deep Blue). The second approach (Type-B) is concentrate on heuristic pruning based on relatively extensive portion of knowledge, direct 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, Null Move and PVS (NegaScout).

This paper has intention to investigate the other possibilities of computer chess strength increasing using parallelism if the other techniques are well implemented. The main pruning method is Alfa-Beta and it is implemented in author's *Axon* application with some technical improvements. The results of tests prove that the implementation of the pruning technique is able to cut the large parts of the tree improving the computer playing strength notably.

II. STANDARD PARALLEL CHESS MACHINES

The standard research parallel chess machines are developed on several universities, as the stand alone applications or a part of some general parallel algorithm design:

- Chess on Massively Parallel Systems at University of Paderborn,
- Parallel Computing Works at CalTech,
- CilkChess Parallel Chess Program at MIT **[2]**,
- International Computer Chess Association.

The main characteristics of two leading parallel chess architectures will be presented in this section.

2.1 DEEP BLUE

Probably, the most important parallel chess machine was developed by the IBM Company, named *IBM Deep Blue* **[3]**. 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). 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 parallalization that is standard for the IBM SP Parallel systems. This method can work well for this application since the data required for processing is relatively small and can be easily (and cheaply) replicated among the processors. Primarily communication between the processors is limited to delegate which processors examine which portions of the tree of possible moves (each level represents one player's move, so the root would have all the possible

Vladan Vučković is with the Faculty of Electronic Engineering, University of Niš, 18000 Niš, Serbia and Montenegro, Email: vld@elfak.ni.ac.yu

moves one player could make, the next level would have the moves in response by the other player). Software determines a set of tactics/moves to explore and then determines likely outcomes and the goodness of them. Since chess is a complex game and the number of moves (and the moves that follow that, etc) is incredibly enormous, true brute force is not used. Highly unlikely moves are eliminated from further consideration (such as putting the king in checkmate). The original algorithm used is based on the alpha-beta algorithm. The system passes messages between all the processors and each processor works on a set of possible moves. Since each processor is examining a different set of moves, it is likely that parallel processing is applicable.

The following graph was compiled from information on the IBM website, but since the information was scattered, it should only be used as an approximation of true performance.

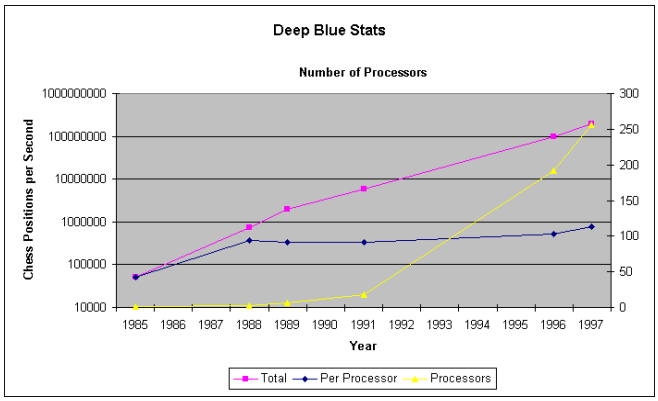

Fig. 1. The influence of parallelization to the CPP factor

The graph displays the number of chess positions per second that are computed. This is analogous to determining the goodness of a position and with some unlisted constant could be converted to operations per second metric. The line representing the positions per second per chess processor is calculated by dividing the total equally over the number of chess processors. This is just an average number, and may not be accurate. It is possible that one or more of the chess processors (but not all) are doing more work than others.

Obviously the number of calculations per second increases with time due to advances in chip technology and algorithm enhancements, but the overall enhancement of the system out performs Moore's law applied to one processor, leading to the conclusion that parallel computing was successful. It also appears that after 1988 there was a slight decrease in position per second, this is probably due to the overhead of parallel computation, but still allowed the overall performance of the machine to increase. Unfortunately data regarding the raw power of each processor running alone was not available to evaluate a true speedup performance. IBM estimates that for a single computer to analyze the same number of positions per second would need to run at about 1 Terahertz. IBM reports that the current Deep Blue system has a parallel efficiency of around 30%.

2.2 CILKCHESS

Cilkchess (developed on MIT university) is a complete rewrite of the previous chess program *(Socrates).* The parallel search algorithm uses a form of Jamboree-search to control parallel search overhead: at each node the first successor is searched serially. If it does not cause a cutoff, the remaining successors are searched in parallel. The transposition table is stored in 32 gigabytes of shared memory. Entries are not locked. (Although this goes against common parallel programming practice, it certainly is fast and seems to work well in practice.) In the late middle game, Cilkchess typically looks more than 15 ply (half moves) ahead and performs 5-11 million make-moves per second on a 256-processor SGI Origin 2000. The evaluation function is built for speed, uses only knowledge, which is known to work, few extensions, and null-move based forward pruning. In the latest version, the weights of the evaluation function are tuned using a temporal-coherence learning algorithm.

III. IMPLEMENTATION

The standard author's version of chess program *Axon* was modified, using relatively simply method of parallelization. The experimental parallel system use two PC computers, which are connected via local 100 Mb network The program was parallelized using *master-slave* methodology. Two identical programs use identical hardware are identical, but they 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 interface. In the phase of computing, master send small pack of date using distributed connection. The packet 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. To demonstrate our solution assume that the current position given on the following diagram (Figure 2):

Fig. 2. AXON I Parallel version of program

The position is resumed from the grandmaster game - key move is winning Ng6!. When we analyze this position using *Axon Evaluation Tester* **[4]**, the following list of legal moves was generated:

0. H4H8 -> %.+.....a.r.p#it (<--)
1. H3E6 -> %..c.d.faq.np..t (<--)
2. H4F4 -> .s...d..a..np.it (1)
3. H4G5 -> .s.....a..np#it (2)
4. E5C4 -> .s.....a..np.i. (<--)
5. E5G4 -> .s.....a..n.#.. (1)
6. F2F4 -> .s.....a..n..it (<--)
7. A1C1 -> .s.....a...p.it (<--)
8. H3G2 -> .s.....a...p.it (1)
9. E1E3 -> .s.....a...p..t (<--)
10. H3F5 -> .s.....a....#.t (2)
11. E1E2 -> .s.....a.....t (1)
12. A1D1 -> .s.....a.....t (1)
13. E1C1 -> .s.....p.i. (2)
14. B2C3 -> .s.....p.i. (<--)
15. H3G4 -> .s.....#.t (3)
16. G3G4 -> .s.....#.t (<--)
17. G1F1 -> .s.....t (<--)
18. A3A4 -> .s.....t (<--)
19. G1H2 -> .s.....t (1)
20. G1G2 -> .s.....t (2)
21. A1A2 -> .s.....t (2)
22. A1B1 -> .s.....t (3)
23. F2F3 -> .s.....t (1)
24. G1H1 -> .s..... (3)
25. E1B1 -> .s..... (3)
26. B2C1 -> .s..... (1)
27. E1D1 -> .s..... (4)
28. E5D3 -> .s..... (2)
29. E1F1 -> .s..... (5)
30. E5F3 -> .s..... (3)
31. H3F1 -> .s..... (4)
32. H4H7 -> ..+..d..a...p#it (3)
33. H4F6 -> ...c.d.fa.rnp#it (4)
34. E5F7 -> ...c....a.rnp#i. (4)
35. E5C6 -> ...c....a.r.p#i. (5)
36. D4D5 -> ....md.fa..np#.t (<--)
37. H4E4 -> .....d.fa..np#it (5)
38. H4H5 -> .....d..a..np#.t (6)
39. H4H6 -> ....fa..np#it (7)
40. H4G4 -> ....f...np#it (8)
41. E5D7 -> ....a..n.#.. (6)
42. E1E4 -> ....a....#.t (6)
43. E5G6 -> ....#.. (7)

The list of moves is sorted and displayed together with their characteristically bits (weights.) *The basic rule for parallelization on two machines is to divide this list in two sub-lists*. The first sub-list contains move with **EVEN** indexes (0. H4H8, 2. H4F4, 4. E5C4 ...) and the second list incloses moves with **ODD** indexes (1. H3E6, 3. H4G5, 5. E5G4 …). After that, each sub-list is distributed to corresponding processor. In the phase of computing, each processor works on different set of moves, so the parallelization is achieved.

IV. 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 contents serious of test chess positions also with key moves.

The experiment consists of 3 phases. First, test will be performing on single processors on depth 7 each position. For the experiment, the PC with *AMD Sempron 2600+* processor/ 256 Mb Ram/ 50 Mb Hash will be used. After that, the same experiment will be performed on two machines working parallel, also with same hardware. The goal of the experiment will be to represent the acceleration of the best move searching on parallel machine compared to single-processor one. The experimental date could be systematized in the following table (Table 1). This table indicates that parallel system always found key moves faster (in less number of positions). The percentage of acceleration and the processor, which computes key move could not be, determine exactly, it is very dependent of the analyzed position. The theoretical reasons for these empiric conclusions are connected with the fact that the search tree is divided in two branches. 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 found the right solution, processed only 28140075 nodes, that is 69.8% of full tree.

TABLE I
TABLE SHOWS NUMBER OF POSITIONS GENERATED WITH SINGLE AND DUAL SYSTEM ON DEPTH 7.

	Single		**Master**		**Slave**	
1	Bh7	728249	**Bh7**	717922	bxc3	596521
2	Rxg7	1466745	**Rxg7**	759523	Qd3	1512966
3	Rd4	1469002	Be7	1031013	**Rd4**	1525525
4	Nc5	1555512	Nxf6	10863177	Ng5	1564616
5	Bd6	1388059	**Bd6**	2597391	Rf1	1381623
6	Rc1	276446	Rf4	497102	**Rc1**	201065
7	Qd8	84001	Qd3	9823036	**Qd8**	56237

8	Qc7	10527566	**Qc7**	9189599	Bxf7	2148993
9	Qxg6	4963938	Qh3	5771366	**Qxg6**	3760483
10	Nh4	16605243	**Nh4**	15273693	Bg5	25709788
11	Ne5	3689251	Rh8	12246408	**Ne5**	2188589
12	O-O-O	37440614	**O-O-O**	32279825	Qa4	32454430
13	Qxh3	2050870	dxc4	19446143	**Qxh3**	1711382
14	Nxb8	1115403	**Nxb8**	1032344	Qd3	1335819
15	Bxg7	14547957	**Bxg7**	6283498	Bg5	10789489
16	a4	23997961	**a4**	22423349	h4	15604024
17	f5	2159123	**f5**	2076168	Nxd5	1212576
18	Qxg5	2690807	e5	12102548	**Qxg5**	2382803
19	Nc4	1466950	Nb5	3312836	**Nc4**	839488
20	Nc3	10621431	**Nc3**	7965793	Ra5	13809770
21	Kc3	346605	**Kc3**	196995	Kc2	177727
22	Bxe2	2965166	Rh2	2151353	**Bxe2**	1580737
23	h7	724166	**h7**	487909	Bc6	5719170
24	Rxh3	2243812	**Rxh3**	2138494	Rg3	7257178
25	c4	4190791	**c4**	2969132	h4	4569494
26	Rxd7	2520953	**Rxd7**	2032580	Bg3	25238813
27	Qxc6	14345697	Ne1	12422980	**Qxc6**	1243715
28	Kh2	7423320	**Kh2**	5258905	Qxa7	3803816
29	Rxf4	2872671	Qxf4	527141	**Rxf4**	2754258
30	Re4	40308515	Qh5	14897082	**Re4**	28140075
31	cxb5	4005703	**cxb5**	2622334	Rd1	3001105
32	Re1	8435525	**Re1**	5680953	Rd1	11814379
33	Qe2	2447903	**Qe2**	2128323	Rf3	2928657

V. CONCLUSION

The distributed computer chess system presented in this paper runs on two PCs with AMD Sempron 2600+ processors connected via 100Mb local area network. The software component is developed around the Axon I chess engine. The EPD test performed in this paper proves that parallel machine finds key moves faster in every case compared to single processor one. This conclusion may be generalized, so in practical testing or playing mode dual machine reacts more rapidly then the single one. In the future, the author intends to expand number of processors and to accommodate software for the efficient multi-processor work.

REFERENCIES

[1] Peter W. Fray *Chess Skill in Man and Machine*, texts and monographs in computer science, Springer-Verlag, 1977,1978, New York, USA

[2] Cilkchess at supertech.lcs.mit.edu/chess/

[3] Deep Blue at www.research.ibm.com/deepblue

[4] Vuckovic, Vladan "Decision Trees and Search Strategies in Chess Problem Solving Applications", *Proceedings of a Workshop on Computational Intelligence and Information Technologies*, pg. 141-159, February 27. 2001.