Parabelle: Experience With a Parallel Chess Program

F. Popowich
and
T.A. Marsland

Technical Report TR83-7

August 1983

Parabelle: Experience With a Parallel Chess Program

F. Popowich
and
T.A. Marsland

Computing Science Department
University of Alberta
EDMONTON T6G 2H1
Canada

08-31-83
TR83-7

Abstract

In a recent report, a parallel version of an alpha-beta search algorithm was presented. The efficiency of this Principal Variation Splitting method has been explored by implementing it into a working chess program, by measuring the resulting performance, and by examining some of the problems associated with parallelism. The results of these tests are presented here, along with discussion of some possible solutions to general difficulties with parallel tree-searching problems.

Keywords: multiprocessors, concurrent programming, message sending, graph and tree search strategies, tree decomposition, alpha-beta search, computer chess

Acknowledgements

The assistance of Steve Sutphen and Jan Rus in configuring and maintaining the system hardware was invaluable, as was the help of Mui-Hua Lim in typesetting the equations and figures. Also, the work of Marius Olafsson in verifying the theoretical model and doing the computations to estimate the average branching factor is recognized. Financial support for this project was provided by the Canadian National Science and Engineering Research Council grants A5556 and A7902.

1. Introduction

When sequential versions of the alpha-beta searching algorithm are adapted for use on a parallel processing system, the speedup in search time is notoriously less than the number of processors in the system. This can be accounted for by taking **search overhead** and **communication overhead** into consideration [MARS82]. The **search overhead** is algorithm dependent since it is defined as the extra work that a parallel algorithm must carry out compared to its sequential counterpart. On the other hand, **communication overhead** results from the necessary exchange of information between processors and is therefore dependent on the system configuration, as well as the algorithm. Since an alpha-beta search uses accumulated information to determine when cutoffs are to occur, a parallel implementation may have one processor missing a cutoff because another one is still calculating the better cutoff value. Thus, parallel alpha-beta is very susceptible to search overhead losses.

To examine these and other problems (such as memory table management) associated with a parallel searching algorithm, we have designed *Parabelle*, a chess program which is based on *Tinkerbelle†*. The results from *Parabelle* are compared to a uni-processor version of the program.

2. Description of the System

The basis of both the sequential and parallel chess programs used in our tests is the Principal Variation Search [MARS83a]. This method presumes a strong ordering of the moves so that the most likely candidate is searched first, while the remaining variations are examined using a zero-width window. **Iterative deepening,** a progressive deepening of the search with dynamic reordering of the moves, along with a **refutation table,** consisting of the refutation lines for each move at the first level in the tree, are used since they have shown their merit in previous tests [MARS83]. During the first and last iterations, a capture search of up to eight ply in depth is performed. The use of a **transposition table,** containing positions seen during the search, is included in order to study different possible implementations of such a large table in a parallel system.

---

† A chess program, developed by K. Thompson [BTL], which participated at the US Computer Chess Championship, ACM National Conference, San Diego, 1975.

Parabelle uses a Principal Variation Splitting method to analyze chess positions on a processor tree network. All of the processors recursively analyze the first move, with the remaining moves being examined using a zero-width window by individual processors. There is a local refutation table for each processor that is updated after each phase of the iterative deepening search. A transposition table, which can be accessed on either a global or local basis, is also included. If a global transposition table is used, then all processors have access to the same positions which are stored in the supervisor processor. Any benefits of this arrangement may be cancelled by the increase in the communication overhead of the system. However, if a local transposition table for each processor is used, then this communication is not necessary, but the table will contain fewer entries since it will not have results from positions seen by other processors.

Parabelle is written in C for use on a Motorola 68000 based Multiple Instruction stream Multiple Data stream (MIMD) system [BOWE80]. Inter-unit communication is handled via communications lines at a maximum rate of 9600 baud. Each unit contains identical software for searching chess positions, with a supervisor processor containing extra code for global table management and communication interface routines. The program itself requires approximately 48K bytes of memory, while the the rest of each unit's 256K bytes of memory is used for data storage. The current system can handle up to nine processors in a depth one processor tree configuration. A sample of the program can be seen in Appendix A.

3. Performance

The program was tested on a sequence of 24 chess positions which appear in Appendix B [BRAT82]. A five ply search, with and without transposition tables, was performed on a system consisting of from one to four processors. This depth was chosen because greater depths result in the overloading of the transposition table.

By contemporary standards, the program for our application is a very weak chess player. Our performance results are not designed to show how well the program plays chess, but rather the relative performance of the various algorithms. While the chess moves selected are displayed in our tables, these are only present to show how the principal variation changes from one search depth to another, or from one processor configuration to another. At present, the quality of the choice is not our primary concern. Also, the

chess program we are using is rather slow. This is partly because it is written wholly in a portable version of C, and also because we preferred maintainability to speed. Our aim is to find the most efficient way of employing many processors in the search of game trees. We assume that any efficiency improvements in the application program itself will be reflected equally in the various algorithms.

3.1. Without Transposition Table

By disabling the transposition table, it is easier to observe certain characteristics of the parallel searching algorithm that may be obscured when the table is in use. Table 1 summarizes the results obtained on five ply searches without a transposition table. The *nodes* column corresponds to the number of leaf nodes searched by all the processors combined. The *secs* field of the table is the real time required, truncated to seconds, for the system to search the tree, with the *speedup* being the ratio of the time required by the uni-processor program to that of the multiprocessor system. The averages, which appear on the bottom row of the table, are calculated by taking the mean of a column of values.

The search overhead of the system is reflected in the general increase in the number of leaf nodes examined as more processors are used. With the exception of positions **H, T,** and **W,** this can be observed for every board in Table 1. To understand why these particular boards do not display this characteristic behavior, one must first examine more closely the behavior of the parallel algorithm as compared to its sequential counterpart.

If the first move is not the best, then a normal window search will be performed on new candidate variations as they are recognized. When this is being done in parallel, it is possible that more of these wider window searches will be carried out, since many processors may simultaneously have a move that is better than the first one. Consequently, there will be more true scores for the list of moves, rather than the approximations returned by the minimal window search. This can alter the search since a different move ordering will result when the movelist is sorted between iterations. Moreover, moves possessing identical scores may be re-ordered because of the nature of the heap sort that is used [MARS83a]. This accounts for the change in the move selected by systems of different processor sizes, such as position W in Table 1, and for the decrease in leaf nodes searched for larger systems, as was seen in **H, T,** and **W.** Although the move

selected may be different, it always has the same score. One should also note the biasing effect of position W on the average number of nodes searched. Due to the size of this search, any trends present have a large effect on the average, as can be seen in the decrease in the average node count from the three to four processor case.

Another factor affecting the speedup is the required synchronization of processors after the searching of all the moves at a node where splitting has occurred. The problem is especially evident for searches where the principal variation changes. When a new candidate variation is found late in the search, as in searches of **D** and **P** from Table 1, the slave that is searching this variation may be the only processor working while the others are waiting for it to finish.

Communication overhead can be observed indirectly in certain board positions where there is little search overhead, even though it is not measured directly. The inability to obtain direct measurements is due to a combination of the short message time and the relatively large timer clock intervals (1/60 sec). The effect of this overhead is particularly noticeable in cases like **F** and **H,** where the entire search requires very little time. Much of the idle time results from the transmission of refutations and movelists after an iteration is completed. This **inter-iteration** communication [MARS83a] occurs at 9600 baud for systems of up to three processors, but at only 4800 baud for the four processor configuration. However, the largest part of the communication consists of short **inter-node** messages [MARS83a], which are used to communicate positions to be searched and scores obtained.

3.2. With Transposition Table

The system performance with a transposition table was tested using both a complete and a partial storage system with a local and a global table. The former arrangement involves storing all nodes visited and performing a table lookup before evaluation of any tree node. In the latter case, only those positions that have been searched to a depth more than two ply are saved and, similarly, a table lookup is performed only when the node is more than two ply from the leaf node. The global transposition table can hold 8192 entries for each side, while local tables are limited to 4096 elements.

When the complete transposition table system is implemented, there is a dramatic decrease in the number of leaf nodes visited. The results in Table 2 show that the global table results in fewer terminal nodes visited than the local tables, Table 3, which can be expected since a global table will provide more cutoffs by having all processors access it. However, the global table's effect on the terminal node count is at the expense of a large increase in communication overhead. The net speedup is only marginal, with the slow communication speeds destroying the program's performance. With a four processor configuration the speedup is actually less than that for three processors, due to the reduction of the communication rate from 9600 to 4800 baud. This rate change was required to prevent the overloading of the communications interface unit. Fortunately, the local transposition table does not require this communication, thus it provides superior elapsed time performance, even though it examines more terminal nodes than its global counterpart.

When a transposition table is used with a parallel system, there is no guarantee that the same cutoffs will occur as in systems composed of different numbers of processors. With a global table, the order of storage and retrieval will vary for systems of different sizes, which can even result in different best moves being selected. When implementing a local table, the positions seen by other processors will not be able to produce a cutoff for the processor in question. Varying cutoffs will then result for differing system sizes depending on which processor searches which subtrees. Consequently, the terminal node counts may even decrease when the number of processors increases, although this is not generally true.

The problems of table overloading and the high frequency of table access associated with the complete storage table (which is used for all subtrees) can be partially alleviated by not storing the values of subtrees whose length is less than three ply. Use of this technique decreases the communication overhead for the global table, resulting in a system which compares more favorably with the local table. Once again, the results from using a local transposition table, Table 4, show a better speedup than those of the global table in Table 5, even though the former configuration examines more terminal nodes. The performance, with respect to terminal node count and search time, is actually better when using the partial rather than the complete storage system. In the latter case, more new entries exist and these must be stored on top of older

ones already in the table. A larger table should alleviate this problem.

Based on the five ply results, it appears that the addition of a local, depth limited transposition table produces the best search times, and correspondingly, the best speedup for the number of processors. With this in mind, a series of six ply tests were performed with the results appearing in Table 6. For the five ply studies, a speedup of 1.89, 2.59 and 3.10, with a standard deviation of 0.10, 0.29 and 0.52 was obtained for the two through four processor systems respectively. The six ply results illustrated speedups of 1.92, 2.66 and 3.27 with larger standard deviations of 0.33, 0.51, and 0.75. In positions **P** and **T,** a speedup greater than the number of processors is achieved, which partly explains the increase in the six ply average. However, this is more an indication of the fact that for the single processor case the transposition table was overloaded. Some good cutoffs occurred in the multi-processor systems that were not available to the uni-processor program. One should also note an increase in the error margin as the number of processors increases. These speedups compare quite favorably to the 2.34 achieved with treesplitting on Arachne [FINK82] using three processors, and the 2.4 obtained on five processor OSTRICH/P [NEWB82] by using a method similar to ours. The curves of Figure 1 show the reduction in search times as the number of processors increase. The results from the six ply searches can be compared to the curve for the optimal speedup of *n* for *n* processors. Even though the largest system tested consisted of only four processors, one can see the leveling of both of these curves as the number of processors increases. The leveling out mirrors the results obtained using other parallel algorithms on minimax tree searches [AKL82][LIND83].

Figure 1. Search Time vs. Processors.

4. Analytical Model

5. Conclusions

Our experimentation with the *Parabelle* system seems to indicate that the PVS method is promising for use in multiple processor tree searching systems. The use of local, depth limited transposition tables also appears to be an effective enhancement to the basic system. The communication problems associated with the global transposition table could be alleviated with faster communication speeds, but the corresponding reduction in the number of nodes that must be visited would only be marginal. More cutoffs could be obtained if the new alpha bound were made available to all processors as soon as it was determined, rather than when a slave processor has finished its subtree search. A further reduction in processor idle time, and thus a corresponding improvement in performance, would be obtained by the allocation of more than one processor for searching new principal variations. By this means, the better cutoff value associated with the new variation will be used earlier in the search of all the subsequent moves in the list. Another possibility for improvement lies in the deferring of new principal variation searches until there is more than one possible new candidate [THOM81]. Thus, if only one such move were found, the wider re-search would not be necessary since one would know that it is the best move, although the true score would not be available.

References

AKL82 S. Akl, D. Barnard and R. Doran, "Design, Analysis, and Implementation of a Parallel Tree Search Algorithm", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-4,** 2 (1982), 192-203.

BOWE80 B.A. Bowen and R.J.A. Buhr, *The Logical Design of Multiple Microprocessor Systems*, Prentice-Hall, (1980), 65-71.

BRAT82 I. Bratko and D. Kopec, "A Test for Comparison of Human and Computer Performance in Chess", *Advances in Computer Chess 3*, M.R.B. Clarke (ed.), Pergamon Press, 1982.

FINK82 R. Finkel and J. Fishburn, "Parallelism in Alpha-Beta Search", *Artificial Intelligence*, **19,** 89-106, (1982).

LIND83 G. Lindstrom, "The Key Node Method: A Highly-Parallel Alpha-Beta Algorithm", UUCS 83-101, Dept. of Computer Science, Univ. of Utah, Salt Lake City, March 1983.

MARS82 T.A. Marsland and M. Campbell, "Parallel Search of Strongly Ordered Game Trees", *Computing Surveys*, **14,** 4 (1982), 533-551.

MARS83 T.A. Marsland, "Relative Efficiency of Alpha-Beta Implementations", IJCAI Proceedings, Karlsruhe, August 1983, 763-766.

MARS83a T.A. Marsland and F. Popowich, "Multiprocessor Tree-searching System Design", TR83-6, Computing Science Dept., Univ. of Alberta, Edmonton, July 1983.

NEWB82 M. Newborn, "OSTRICH/P - A Parallel Search Chess Program", SOCS - 82.3, School of Computer Science, McGill Univ., Montreal, March 1982.

THOM81 K. Thompson, Private Communication, Oct. 1981.