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.

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

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.

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].

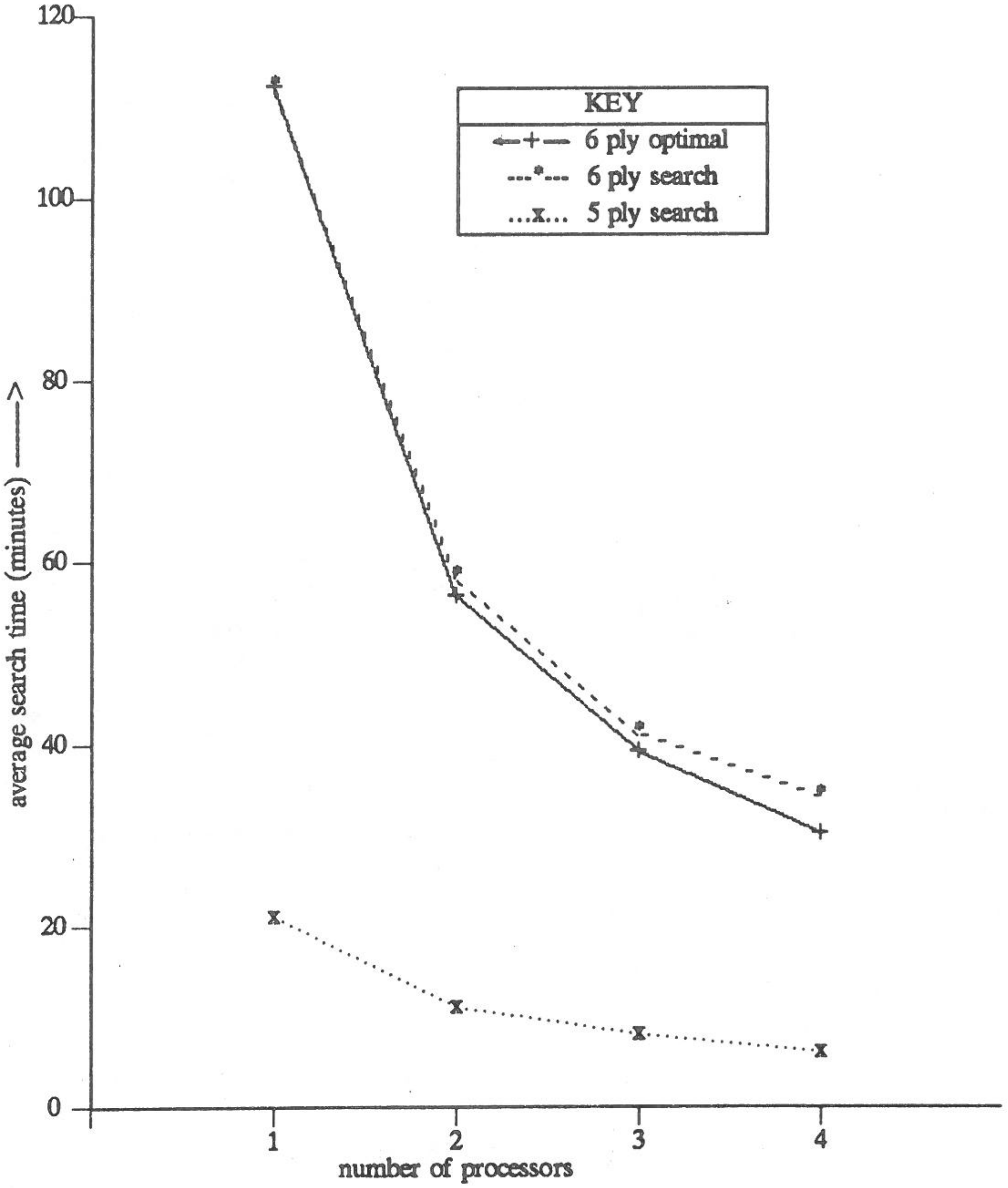

Figure 1: Average Search time vs number of processors

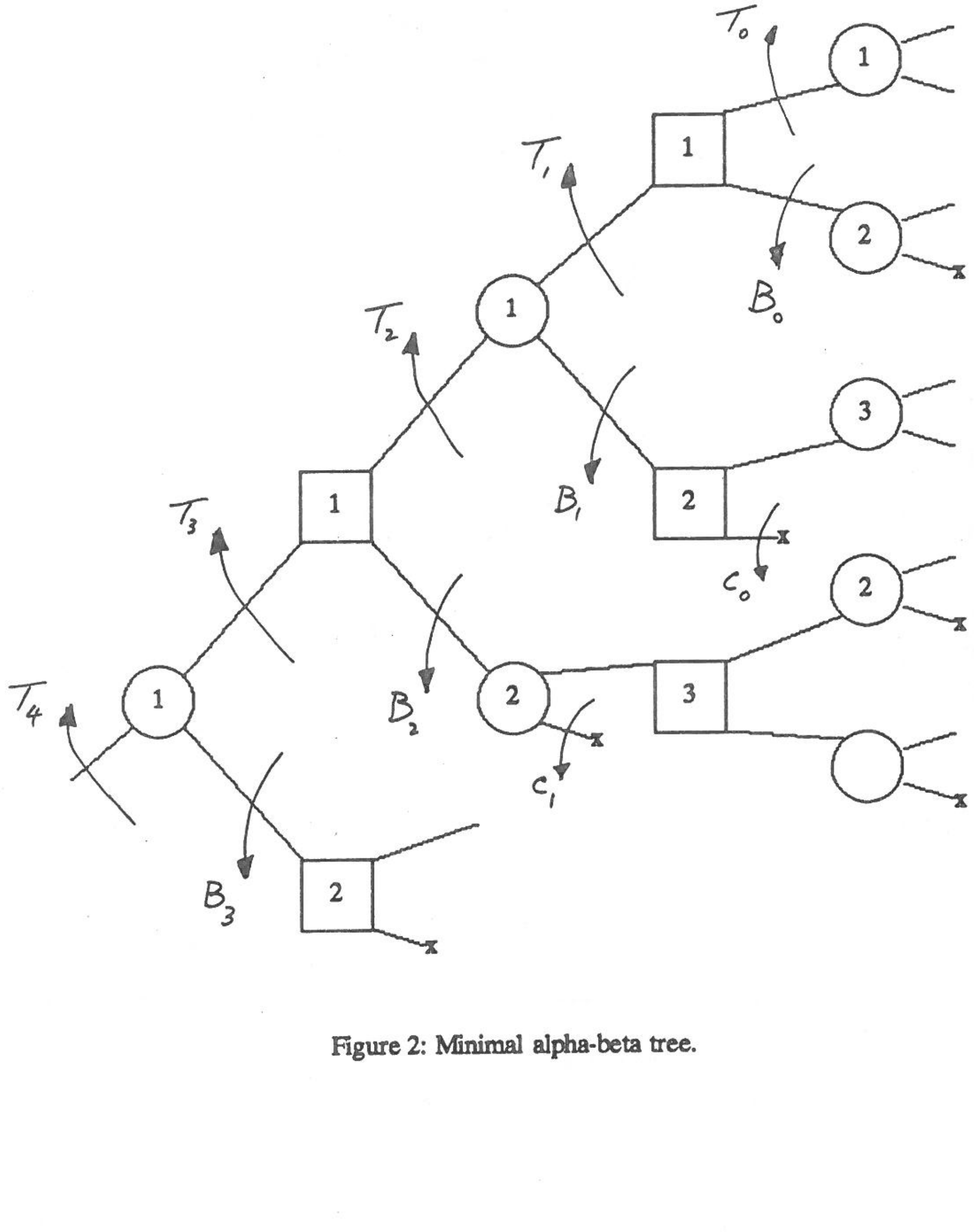

Figure 2: Minimal alpha-beta tree.

4. Analytical Model

It is difficult to develop a mathematical model which can be used to estimate the time to search a multi-branch tree, especially if several processors are to be used. In the minimax search of game trees the actual size of the tree is only known in some statistical sense, so it is customary to model first the search time for a uniform (fixed number of branches at each node) game tree of specified depth. However, since search of the minimum game tree provides a lower bound on the search time, and since that tree is regular and well defined, it is possible to derive a formula for the number of nodes in the minimal tree and hence for the search time for a designated processor configuration and search strategy. Any speedup factor which may be computed based on the use of 'f' processors to search the minimal game tree, should also be a good estimator of the speedup possible for a normal alpha-beta minimax search.

For the purposes of this analysis, the searching strategy considered is the Principal Variation Search, the processor configuration assumed is a processor tree of length one employing 'f' processors, and the tree being searched is a perfectly ordered uniform game tree of width 'w' and depth 'd'. Thus all 'f' processors traverse the first branch of each node along the principal variation and then the balance of the branches at the type 1 nodes, Figure 2, are assumed to be split equally among the processors. The experimental results presented in this report are for a system which follows this strategy.

In order to simplify presentation the following notation is used

w = No. of branches (moves) on any node (position).
d = depth of search.
f = No. of processors (fanout).
k = time to create node and generate move list.
e = time to evaluate a single move at a terminal node.
s = setup time in handling a message.
T_d = time to search a minimal tree to depth d.
t_1 = time to transmit an inter-node message of length b_1 bytes.
t_2 = time to transmit an inter-level message of length b_2 bytes.
B_d = time to do a zero window search to depth d.
N_d = Number of terminal nodes in a minimal alpha-beta tree.
E_d = Cost of evaluating the terminal nodes of an alpha-beta tree.

Note that both t_1 and t_2 are proportional not only to the communication link speed (here 4800 or 9600 baud) but also to the message length.

It is well known that for an optimal (minimal) alpha-beta tree the number of terminal nodes in a tree of depth d and width w is given by the equation

$$N_d = w^{\lceil \frac{d}{2} \rceil} + w^{\lfloor \frac{d}{2} \rfloor} - 1 \tag{1}$$

where $\lceil x \rceil$ and $\lfloor x \rfloor$ represent the first integer greater than and less than x respectively. Thus the number of terminal nodes depends on whether d is even or odd. The terminal node expression is too simple for our purpose since it does not take into account the fact that these terminal nodes are of two types, 2 and 3 in Knuth's terminology, being distinguished by the fact that all moves must be evaluated (assessed) at a type 3 node while only 1 move is evaluated at a type 2 node. Furthermore, although the dominant computations are done at the terminal nodes, the cost of an interior node is not negligible (even though it is not particularly dependent on the node type). Finally inter-processor communication is necessary at all type 1 nodes (where tree splitting occurs), and this may be quite slow in comparison to the processor cycle time.

From Figure 2 the following recurrence relationships to estimate T_d, the number of processor cycles required to search a minimal alpha-beta tree to depth d, may be derived as follows:

$$T_d = k + T_{d-1} + (w-1)B_{d-1}, \qquad \text{for } d > 0, \tag{2}$$

where $T_0 = k + we$

and $B_d = 2k + w\, B_{d-2}, \qquad \text{for } d > 1, \qquad (3)$

where $B_0 = k + e$
and $B_1 = 2k + we$

After some substitutions equation (2) becomes

$$T_d = (d+1)k + we + (w-1)\sum_{i=0}^{d-1} B_i \tag{4}$$

Note that from equation (4) the cost of evaluating the terminal nodes of an uniform alpha-beta tree of depth d is given by

$$\begin{aligned} E_d &= T_d - T_{d-1} \\ &= k + (w-1)B_{d-1} \end{aligned}$$

Thus when d is odd this is equivalent to

$$E_{2m+1} = k + (w-1)B_{2m}$$

$$= k + (w-1)\ [2k \sum_{i=0}^{m-1} w^i + (k+e)w^m]$$

$$= k + (w-1)\ [2k \sum_{i=0}^{m} w^i + ew^m - kw^m] \tag{5}$$

Equation (5) represents the cost in processor cycles of evaluating the terminal nodes of a minimal game tree. However, if we set $k=1$ and $e=0$ then this cost is also equal to the number of terminal nodes in the minimal tree, and hence reduces to equation (1) as follows:

$$E_{2m+1} = 1 + (w-1)[\frac{2(w^{m+1}-1)}{w-1} - w^m]$$

$$= 1 + 2w^{m+1} - 2 - w^{m+1} + w^m$$

$$N_{2m+1} = w^{m+1} + w^m - 1 \ = \ E_{2m+1} \tag{6}$$

which is the same as equation (1) for trees of odd depth. The corresponding equation for even-depth trees can be obtained in the same way. Thus an independent check on the validity of (4) has been provided.

While equations (3) and (4) may be used to estimate the tree traversal time for a uniprocessor they account neither for multiprocessors using tree splitting nor for interprocessor communication. When PVS is employed using a processor tree of length 1, with fan-out of 'f', the search time is

$$T_{d,f} = k + T_{d-1,f} + \lceil\frac{(w-1)}{f}\rceil B_{d-1}, \qquad \text{for } d > 0, \text{ and } f > 0, \tag{7}$$

where $T_{0,f} = k + e + \lceil\frac{(w-1)}{f}\rceil e$.

If we make the simplifying assumption that $(w-1) \bmod f = 0$, then the equivalent form of equation (6) becomes

$$N_{2m+1,f} = \frac{1}{f}[w^{m+1} + w^m - 1] + \frac{(f-1)}{f} \tag{8}$$

Since tree splitting only occurs at type 1 nodes, the formula for B_d, equation (3), still applies.

Finally, when interprocessor communication is included,

$$T_{d,f} = k + T_{d-1,f} + \lceil\frac{(w-1)}{f}\rceil B_{d-1} + (w-1)\frac{(f-1)}{f}(s+t_1) + (f-1)(s+t_2) \tag{9}$$

for d > 0, and f > 0, where T_0 is similarly modified.

4.1 Average branching factor

Let us assume that a typical alpha-beta tree is approximated by a uniform alpha-beta tree. That is to say is represented by a tree in which $(1+g)$ branches are expanded at each node where a cut-off may occur, rather than only 1 for the minimal tree case. Thus while equation (9) for T_d is still correct, equation (3) becomes

$$B_d = 2k + wB_{d-2} + gC_{d-1}, \qquad \text{for } d > 1 \tag{10}$$

$$\text{where } B_0 = k + (1+g)e \qquad \text{and } B_1 = 2k + we + gC_0$$

$$\text{and } C_d = k + (1+g)C_{d-1} \tag{11}$$

$$\text{with } C_0 = k + (1+g)e$$

The only unknown here is g. If we assume that an average alpha-beta tree is approximated by a uniform tree of width w and depth d in which exactly $1+g$ branches are searched at the cut-off nodes, then g may be estimated.

Ignoring communication costs, in the multiprocessor case with fanout f the evaluation cost of the terminal nodes has been given by equation (7) as

$$E_{d,f} = T_{d,f} - T_{d-1,f} = k + \lceil \frac{(w-1)}{f} \rceil B_{d-1}$$

Thus for an odd ply tree, $d=2m+1$, one gets

$$E_{2m+1,f} = k + \lceil \frac{(w-1)}{f} \rceil B_{2m} \tag{12}$$

where B_{2m} is given by equation (10) as

$$\begin{aligned} B_{2m} &= 2k + wB_{2m-2} + gC_{2m-1} \\ &= 2k + wB_{2m-2} + g[k \sum_{i=0}^{2m-1} (1+g)^i + e(1+g)^{2m}] \end{aligned}$$

That is,

$$B_{2m} = A_{2m} + wB_{2m-2} \tag{13}$$

$$\text{where } A_{2m} = k + (k+ge)(1+g)^{2m}$$

Thus

$$\begin{aligned} B_{2m} &= \sum_{i=0}^{m-1} A_{2(m-i)}\, w^i + w^m B_0 \\ &= k \sum_{i=0}^{m-1} w^i + (k+ge) \sum_{i=0}^{m-1} (1+g)^{2(m-i)}\, w^i + w^m (k + (1+g)e) \\ &= k \sum_{i=0}^{m} w^i + (k+ge)(1+g)^{2m} \sum_{i=0}^{m-1} [\frac{w}{(1+g)^2}]^i + w^m\, ((1+g)e) \end{aligned} \tag{14}$$

Finally, if we make the simplifying assumption that $(w-1) \bmod f = 0$, then equation (12) becomes

$$E_{2m+1,f} = k + \frac{w-1}{f} B_{2m} \tag{15}$$

Note that with $f=1$, $e=0$ and $g=0$ equations (14) and (15) reduce to equation (6) as is necessary.

For 5-ply trees with width $w=35$ we know that the minimal number of terminal nodes searched is given by equation (1) as 44,000. From our experimental data we have that for our 5-ply trees, with average width $w=35$, the number of terminal nodes is between 50,000 and 100,000 nodes [MARS 83]. Thus by using equations (14) and (15), after setting $f=1$, $k=1$, $e=0$ and $m=2$, we may estimate g, the average number of additional branches that are examined at cut-off nodes. For example, from (15)

$$E_5 = w^3 + w^2(1+g)^2 + w((1+g)^4 - (1+g)^2) - (1+g)^4 \tag{16}$$

It is clear from equation (16) that one positive and one negative value for g exists, and that those values may be found iteratively by plotting g against E_5 and w. Typical experimental values for w and E_d have been presented previously [MARS 83]. For our purposes it is sufficient to note that for $w=35$, $g=1.3$ when $E_5=50{,}000$ and $g=4.2$ when $E_5=100{,}000$.

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.

Terminal node count and CPU time (5-ply) No transposition table															
	one processor			two processors				three processors				four processors			
board	nodes	secs	move	nodes	secs	speedup	move	nodes	secs	speedup	move	nodes	secs	speedup	move
A	(forced	mate)	d6d1				d6d1				d6d1				d6d1
B	39412	806	e4e5	39520	411	1.96	e4e5	39720	302	2.66	e4e5	39820	288	2.79	e4e5
C	29861	1189	e7d8	29952	658	1.81	e7d8	30458	436	2.73	e7d8	30846	371	3.20	e7d8
D	57294	816	e5e6	59526	434	1.88	e5e6	67282	428	1.90	e5e6	67316	370	2.20	e5e6
E	98258	2617	f1f5	98259	1328	1.97	f1f5	98260	907	2.88	f1f5	98261	710	3.68	f1f5
F	12591	100	g5g6	13597	62	1.61	g5g6	13614	46	2.17	g5g6	13631	41	2.44	g5g6
G	44755	1303	a3b4	44809	657	1.98	a3b4	45152	465	2.80	a3b4	45268	366	3.56	a3b4
H	4867	38	e2c3	4934	22	1.69	e2c3	4987	17	2.23	e2c3	**4872**	16	2.33	e2c3
I	77113	2607	f3g1	90465	1582	1.65	f3g1	102284	1294	2.02	f3g1	116455	1185	2.20	f3g1
J	69208	2142	d8d7	69521	1115	1.92	d8d7	69701	780	2.75	d8d7	69914	614	3.49	d8d7
K	53830	1356	g3f5	54098	686	1.98	g3f5	54354	475	2.85	g3f5	54658	374	3.62	g3f5
L	33610	819	d7f5	33834	430	1.91	d7f5	33965	298	2.74	d7f5	33812	234	3.49	d7f5
M	75173	1078	a1c1	75360	538	2.00	a1c1	75510	371	2.90	a1c1	75666	292	3.69	a1c1
N	31864	600	d1d2	34694	349	1.72	d1d2	36237	262	2.29	d1d2	37682	221	2.70	d1d2
O	22047	523	g4g7	22547	297	1.76	g4g7	22896	221	2.37	g4g7	23184	182	2.87	g4g7
P	67809	1393	g5f4	73237	856	1.63	g5f4	75741	723	1.92	g5f4	76161	664	2.10	g5f4
Q	38027	1370	d7c5	38779	750	1.83	d7c5	39540	543	2.52	d7c5	40284	441	3.10	d7c5
R	86254	2118	c8g4	86291	1058	2.00	c8g4	86327	731	2.89	c8g4	86365	564	3.75	c8g4
S	36379	862	c7c5	36506	436	1.98	c7c5	36590	307	2.81	c7c5	36759	244	3.52	c7c5
T	127897	2614	d1d2	**127714**	1334	1.96	d1d2	129125	949	2.76	d1d2	131789	770	3.39	d1d2
U	90266	2004	f5d4	92110	1029	1.95	f5d4	92169	698	2.87	f5d4	92228	541	3.70	f5d4
V	40242	1191	d7e5	40422	635	1.88	d7e5	40559	525	2.27	d7e5	40659	501	2.38	d7e5
W	173866	8070	c8f5	218013	6263	1.29	c8f5	**218012**	3975	2.03	**g7g6**	**195874**	2815	2.87	**g7g6**
X	70893	1438	b4c5	70990	737	1.95	b4c5	71087	511	2.81	b4c5	71326	436	3.29	b4c5
tot	1381516	37064		1455178	21678			1483570	15274			1482830	12254		
ave	60065	1611		63268	942	**1.84**		64503	664	**2.53**		64470	532	**3.06**	

Table 1: 5-ply search without transposition table.

Terminal node count and CPU time (5-ply) Global transposition table for all subtrees															
	one processor			two processors				three processors				four processors			
board	nodes	secs	move	nodes	secs	speedup	move	nodes	secs	speedup	move	nodes	secs	speedup	move
A	(forced	mate)	d6d1				d6d1				d6d1				d6d1
B	33950	652	e4e5	34094	522	1.25	e4e5	34101	424	1.54	e4e5	34063	480	1.36	e4e5
C	27322	1051	e7d8	27643	712	1.48	e7d8	28065	545	1.93	e7d8	28147	580	1.81	e7d8
D	52052	705	e5e6	51181	606	1.16	e5e6	51188	494	1.42	e5e6	53177	605	1.16	e5e6
E	97509	2546	f1f5	97509	1778	1.43	f1f5	97523	1374	1.85	f1f5	97534	1417	1.80	f1f5
F	11968	89	g5g6	13075	117	0.76	g5g6	13203	126	0.71	g5g6	13395	174	0.51	g5g6
G	44169	1253	a3b4	44579	889	1.41	a3b4	44849	754	1.66	a3b4	90576	1606	0.78	**f3g3**
H	4151	34	e2c3	4157	43	0.80	e2c3	4205	41	0.84	e2c3	4172	58	0.60	e2c3
I	62177	1865	f3e5	63576	1267	1.47	f3e5	63574	985	1.89	f3e5	63610	994	1.88	f3e5
J	96759	3831	d8d6	96737	2518	1.52	d8d6	96867	1846	2.08	d8d6	96692	1964	1.95	d8d6
K	52906	1315	g3f5	53580	962	1.37	g3f5	53539	759	1.73	g3f5	53666	805	1.63	g3f5
L	33364	795	d7f5	33978	608	1.31	d7f5	34013	483	1.64	d7f5	34379	532	1.49	d7f5
M	74706	1057	a1c1	74731	892	1.18	a1c1	75058	730	1.45	a1c1	75291	853	1.24	a1c1
N	30633	537	d1d2	32042	458	1.17	d1d2	33584	390	1.38	d1d2	35026	433	1.24	d1d2
O	21177	485	g4g7	21411	382	1.27	g4g7	21634	340	1.42	g4g7	21613	366	1.33	g4g7
P	51382	1059	g5e7	50820	813	1.30	g5e7	51024	709	1.49	**g5h4**	51613	797	1.33	**g5h4**
Q	36321	1269	d7c5	36446	855	1.48	d7c5	36557	664	1.91	d7c5	37151	712	1.78	d7c5
R	81373	1908	c8g4	81419	1365	1.40	c8g4	81479	1068	1.79	c8g4	81530	1117	1.71	c8g4
S	35946	818	c7c5	35991	610	1.34	c7c5	36229	491	1.66	c7c5	36321	551	1.48	c7c5
T	126903	2577	d1d2	126806	1954	1.32	d1d2	129117	1599	1.61	d1d2	130288	1820	1.42	d1d2
U	88397	1888	f5d4	90194	1426	1.32	f5d4	90302	1112	1.70	f5d4	90353	1227	1.54	f5d4
V	40226	1157	d7e5	40746	757	1.53	d7e5	40943	889	1.30	d7e5	41035	975	1.19	d7e5
W	52711	1712	e8g8	52786	1206	1.42	e8g8	52817	943	1.82	e8g8	51987	1076	1.59	e8g8
X	70931	1464	b4c5	71003	1089	1.34	b4c5	71053	851	1.72	b4c5	71039	910	1.61	b4c5
tot	1227033	30075		1234504	21838			1240924	17629			1292658	20062		
ave	53349	1307		53674	949	**1.31**		53953	766	**1.59**		56202	872	**1.41**	

Table 2: 5-ply search with global transposition table.

Terminal node count and CPU time (5-ply) Local transposition table for all subtrees															
	one processor			two processors				three processors				four processors			
board	nodes	secs	move	nodes	secs	speedup	move	nodes	secs	speedup	move	nodes	secs	speedup	move
A	(forced	mate)	d6d1				d6d1				d6d1				d6d1
B	33950	652	e4e5	34092	339	1.92	e4e5	34210	235	2.77	e4e5	34279	186	3.49	e4e5
C	27322	1051	e7d8	27970	597	1.76	e7d8	28785	401	2.62	e7d8	29238	354	2.97	e7d8
D	52052	705	e5e6	52203	360	1.96	e5e6	52902	258	2.73	e5e6	61078	333	2.11	e5e6
E	97509	2546	f1f5	97934	1308	1.95	f1f5	98012	898	2.83	f1f5	88545	667	3.82	**f1f3**
F	11968	89	g5g6	12133	50	1.79	g5g6	13255	45	1.95	g5g6	13470	42	2.11	g5g6
G	44169	1253	a3b4	44807	654	1.91	a3b4	67303	773	1.62	**f3g3**	85896	812	1.54	**f3g3**
H	4151	34	e2c3	4572	21	1.62	e2c3	4642	16	2.05	e2c3	4498	15	2.19	e2c3
I	62177	1865	f3e5	63647	978	1.91	f3e5	63686	676	2.76	f3e5	63749	555	3.36	f3e5
J	96759	3831	d8d6	96923	1953	1.96	d8d6	96913	1349	2.84	**e8e6**	96456	1109	3.45	d8d6
K	52906	1315	g3f5	53572	681	1.93	g3f5	53861	474	2.77	g3f5	53785	372	3.53	g3f5
L	33364	795	d7f5	33650	419	1.90	d7f5	33497	294	2.70	d7f5	69006	963	0.83	f7f5
M	74706	1057	a1c1	74956	533	1.98	a1c1	75234	368	2.87	a1c1	75569	290	3.64	a1c1
N	30633	537	d1d2	32079	305	1.76	d1d2	33622	224	2.39	d1d2	35064	187	2.87	d1d2
O	21177	485	g4g7	21471	277	1.75	g4g7	21633	212	2.29	g4g7	21721	172	2.82	g4g7
P	51382	1059	g5e7	51467	560	1.89	g5e7	51706	419	2.52	**g5h4**	52317	362	2.92	**g5h4**
Q	36321	1269	d7c5	37494	697	1.82	d7c5	37395	463	2.74	d7c5	38530	432	2.94	d7c5
R	81373	1908	c8g4	83836	1002	1.90	c8g4	84708	707	2.70	c8g4	85137	551	3.46	c8g4
S	35946	818	c7c5	36161	422	1.94	c7c5	36463	299	2.73	c7c5	36557	241	3.39	c7c5
T	126903	2577	d1d2	126928	1323	1.95	d1d2	128476	949	2.72	d1d2	131183	769	3.35	d1d2
U	88397	1888	f5d4	91037	992	1.90	f5d4	91386	686	2.75	f5d4	91628	531	3.55	f5d4
V	40226	1157	d7e5	40746	619	1.87	d7e5	40944	633	1.83	d7e5	41036	573	2.02	d7e5
W	52711	1712	e8g8	53014	886	1.93	e8g8	53128	638	2.68	e8g8	52344	575	2.97	e8g8
X	70931	1464	b4c5	71166	764	1.92	b4c5	71182	526	2.78	b4c5	71321	417	3.51	b4c5
tot	1227033	30075		1241858	15751			1272943	11555			1332407	10520		
ave	53349	1307		53993	684	**1.88**		55345	502	**2.55**		57930	457	**2.91**	

Table 3: 5-ply search with local transposition table.

Terminal node count and CPU time (5-ply) Local transposition table for subtrees > 2-ply															
	one processor			two processors				three processors				four processors			
board	nodes	secs	move	nodes	secs	speedup	move	nodes	secs	speedup	move	nodes	secs	speedup	move
A	(forced	mate)	d6d1				d6d1				d6d1				d6d1
B	33922	661	e4e5	34074	342	1.93	e4e5	34176	236	2.80	e4e5	34282	189	3.49	e4e5
C	29337	1162	e7d8	29464	644	1.80	e7d8	29828	426	2.73	e7d8	30024	363	3.20	e7d8
D	53613	747	e5e6	53874	383	1.95	e5e6	53924	267	2.79	e5e6	62214	343	2.18	e5e6
E	98059	2603	f1f5	98132	1321	1.97	f1f5	98239	907	2.87	f1f5	98200	709	3.67	f1f5
F	12000	95	g5g6	12075	51	1.86	g5g6	13070	45	2.07	g5g6	13110	41	2.29	g5g6
G	44242	1269	a3b4	44636	656	1.93	a3b4	58153	673	1.88	**f3g3**	59441	540	2.35	**f3g3**
H	4086	33	e2c3	4529	21	1.56	e2c3	4626	16	2.01	e2c3	4528	16	2.07	e2c3
I	62190	1882	f3e5	63718	982	1.92	f3e5	63762	679	2.77	f3e5	63830	556	3.38	f3e5
J	74334	2439	d8d5	78680	1381	1.77	**e8e7**	79713	965	2.53	**e8e7**	88430	872	2.79	**e8e7**
K	53085	1328	g3f5	53796	685	1.94	g3f5	54029	477	2.78	g3f5	53941	373	3.56	g3f5
L	33489	815	d7f5	33498	423	1.93	d7f5	33522	294	2.77	d7f5	33561	233	3.49	d7f5
M	74894	1070	a1c1	75084	536	2.00	a1c1	75139	369	2.90	a1c1	75712	295	3.62	a1c1
N	30636	566	d1d2	32081	317	1.79	d1d2	33625	228	2.48	d1d2	35068	188	3.00	d1d2
O	21184	499	g4g7	21490	282	1.77	g4g7	21669	208	2.39	g4g7	21669	173	2.87	g4g7
P	52772	1108	g5e7	52684	590	1.88	g5e7	52883	436	2.54	**g5h4**	53142	377	2.94	**g5h4**
Q	37332	1353	d7c5	37960	719	1.88	d7c5	38119	520	2.60	d7c5	38841	435	3.11	d7c5
R	81427	1940	c8g4	83740	1008	1.92	c8g4	84759	710	2.73	c8g4	85118	549	3.53	c8g4
S	36034	844	c7c5	36229	429	1.97	c7c5	36340	303	2.78	c7c5	36461	243	3.47	c7c5
T	126937	2584	d1d2	127288	1332	1.94	d1d2	128832	953	2.71	d1d2	131333	771	3.35	d1d2
U	88425	1910	f5d4	91067	999	1.91	f5d4	91427	687	2.78	f5d4	91689	531	3.59	f5d4
V	40241	1185	d7e5	40422	632	1.87	d7e5	40559	526	2.25	d7e5	40659	501	2.36	d7e5
W	52927	1745	e8g8	53137	896	1.95	e8g8	52335	697	2.50	e8g8	47575	525	3.32	e8g8
X	71062	1498	b4c5	71169	774	1.93	b4c5	71185	533	2.81	b4c5	71324	419	3.58	b4c5
tot	1212228	29345		1228827	15412			1249914	11167			1270152	9252		
ave	52705	1275		53427	670	**1.89**		54344	485	**2.59**		55224	402	**3.10**	

Table 4: 5-ply search with local, depth-limited, transposition table.

	Terminal node count and CPU time (5-ply) Global transposition table for subtrees > 2-ply														
	one processor			two processors				three processors				four processors			
board	nodes	secs	move	nodes	secs	speedup	move	nodes	secs	speedup	move	nodes	secs	speedup	move
A	(forced	mate)	d6d1				d6d1				d6d1				d6d1
B	33922	661	e4e5	33882	353	1.87	e4e5	33881	247	2.68	e4e5	33865	204	3.24	e4e5
C	29337	1162	e7d8	29391	656	1.77	e7d8	29720	439	2.65	e7d8	29818	373	3.11	e7d8
D	53613	747	e5e6	53735	400	1.87	e5e6	53785	281	2.65	e5e6	62145	376	1.98	e5e6
E	98059	2603	f1f5	98060	1350	1.93	f1f5	98061	929	2.80	f1f5	98060	750	3.47	f1f5
F	12000	95	g5g6	12017	57	1.65	g5g6	13023	52	1.80	g5g6	13040	54	1.74	g5g6
G	44242	1269	a3b4	44386	663	1.91	a3b4	57870	684	1.85	**f3g3**	59072	561	2.26	**f3g3**
H	4086	33	e2c3	4127	22	1.50	e2c3	4148	17	1.87	e2c3	4150	18	1.75	e2c3
I	62190	1882	f3e5	63646	989	1.90	f3e5	63650	686	2.74	f3e5	63692	577	3.26	f3e5
J	74334	2439	d8d5	80843	1464	1.66	d8d5	84033	1056	2.31	d8d5	84098	843	2.89	d8d5
K	53085	1328	g3f5	53497	696	1.91	g3f5	53526	486	2.73	g3f5	53466	393	3.38	g3f5
L	33489	815	d7f5	33493	435	1.87	d7f5	33513	303	2.69	d7f5	33545	255	3.19	d7f5
M	74894	1070	a1c1	74940	556	1.92	a1c1	74908	387	2.76	a1c1	75276	326	3.28	a1c1
N	30636	566	d1d2	32045	327	1.73	d1d2	33588	240	2.36	d1d2	35031	209	2.71	d1d2
O	21184	499	g4g7	21462	291	1.71	g4g7	21587	218	2.29	g4g7	21693	187	2.66	g4g7
P	52772	1108	g5e7	52610	603	1.84	g5e7	52757	452	2.45	**g5h4**	53036	403	2.75	**g5h4**
Q	37332	1353	d7c5	37714	726	1.86	d7c5	37761	525	2.58	d7c5	38276	445	3.04	d7c5
R	81427	1940	c8g4	81494	987	1.97	c8g4	81578	683	2.84	c8g4	81586	541	3.58	c8g4
S	36034	844	c7c5	36089	440	1.92	c7c5	36169	313	2.69	c7c5	36258	261	3.23	c7c5
T	126937	2584	d1d2	127015	1361	1.90	d1d2	128316	982	2.63	d1d2	129653	822	3.14	d1d2
U	88425	1910	f5d4	90266	1005	1.90	f5d4	90324	688	2.78	f5d4	90389	551	3.47	f5d4
V	40241	1185	d7e5	40421	640	1.85	d7e5	40558	534	2.22	d7e5	40658	517	2.29	d7e5
W	52927	1745	e8g8	53003	912	1.91	e8g8	52169	709	2.46	e8g8	47413	552	3.16	e8g8
X	71062	1498	b4c5	71169	792	1.89	b4c5	71185	549	2.73	b4c5	71324	450	3.33	b4c5
tot	1212228	29345		1225305	15735			1246110	11470			1255544	9680		
ave	52705	1275		53274	684	**1.84**		54178	498	**2.50**		54588	420	**2.91**	

Table 5: 5-ply search with global, depth-limited, transposition table.

Terminal node count and CPU time (6-ply)
Local transposition table for subtrees > 2-ply

	one processor			two processors				three processors				four processors			
board	nodes	min	move	nodes	min	speedup	move	nodes	min	speedup	move	nodes	min	speedup	move
A	(forced	mate)	d6d1				d6d1				d6d1				d6d1
B	132751	45	e4e5	138063	23	1.92	e4e5	140299	16	2.80	e4e5	141677	12	3.58	e4e5
C	141201	74	e7d8	146760	39	1.89	e7d8	149580	28	2.65	e7d8	153523	22	3.33	e7d8
D	131304	42	e5e6	136711	23	1.84	e5e6	143206	17	2.45	e5e6	144396	13	3.05	e5e6
E	351867	227	f1f5	362944	125	1.82	f1f5	364990	88	2.57	f1f5	365387	70	3.21	f1f5
F	42942	8	g5g6	47869	5	1.69	g5g6	52372	3	2.24	g5g6	56051	3	2.64	g5g6
G	228816	148	h5f6	239839	78	1.89	h5f6	340005	74	1.98	h5f6	404481	75	1.95	h5f6
H	14780	2	e2c3	15671	1	1.77	e2c3	16142	1	2.30	e2c3	16278	0	2.56	e2c3
I	174398	111	f3e5	178305	58	1.91	f3e5	178586	38	2.85	f3e5	178555	30	3.66	f3e5
J	436938	202	e8e6	545042	130	1.56	**e8e7**	510633	82	2.44	e8e6	509370	67	2.99	e8e6
K	249238	126	g3f5	256469	66	1.89	g3f5	270668	51	2.45	g3f5	277254	41	3.06	g3f5
L	165232	62	d7f5	170502	33	1.89	d7f5	173795	23	2.70	d7f5	176636	18	3.41	d7f5
M	189128	72	a1c1	192818	37	1.93	a1c1	200136	25	2.82	a1c1	201468	20	3.63	a1c1
N	83343	39	d1d2	86935	21	1.83	d1d2	83653	14	2.70	d1d2	78559	10	3.69	d1d2
O	61098	30	g4g7	64520	16	1.82	g4g7	66482	12	2.44	g4g7	65447	9	3.13	g4g7
P	212469	100	d2e4	187435	44	**2.24**	d2e4	219667	38	2.62	d2e4	224228	34	2.95	d2e4
Q	191081	141	d7c5	199063	75	1.87	d7c5	202848	53	2.64	d7c5	209140	46	3.08	d7c5
R	347935	191	c8g4	371298	105	1.82	c8g4	379596	71	2.67	c8g4	383867	56	3.41	c8g4
S	200645	81	c7c5	205487	43	1.87	c7c5	207509	31	2.58	c7c5	208764	25	3.18	c7c5
T	677076	315	c3b5	393672	95	**3.32**	**d1d2**	386704	66	**4.76**	c3b5	388266	52	**6.05**	c3b5
U	367754	194	f5h6	391581	103	1.88	f5h6	397914	69	2.80	f5h6	400983	54	3.59	f5h6
V	132017	86	d7e5	130688	43	1.98	d7e5	130886	39	2.19	d7e5	131032	37	2.29	d7e5
W	531720	204	e8g8	541273	126	1.62	e8g8	540947	75	2.70	e8g8	460973	63	3.20	e8g8
X	176158	78	b4c5	176184	39	1.98	b4c5	178088	27	2.85	b4c5	179959	21	3.59	b4c5
tot	5239891	2590		5179129	1339			5334706	953			5356294	790		
ave	227821	112		225179	58	**1.92**		231943	41	**2.66**		232882	34	**3.27**	

Table 6: 6-ply search with local, depth-limited, transposition table.

Appendix A: Parabelle Source Code

```c
/*
 * The following code is a condensed version of the searching routines
 * executed by the slave processors.  Moves and their values are stored
 * adjacent to each other in a stack of "shorts", and are pointed to by
 * fmp (first move pointer), mp and lmp (last move pointer)
 *
 * Other local variables are:
 *
 *   score - best score found so far
 *   value - value of current move
 *   princ - position of move in stack if the move is a new principal variation
 *   path  - position of move in the refutation table
 *   depth - current search depth
 *   pck   - a 4 byte information package consisting of a score, and either
 *           the "princ" value or the number of moves to ignore before searching
 *           the next move
 *   mask  - mask of which moves we searched
 *
 *
 * The following functions are used by the program
 *
 *   make        - makes the move that is given as the argument
 *   undo        - undoes the given move
 *   statl       - generates all the legal moves for the current position
 *                 incorporating the heuristics
 *   clear_ref   - empties the refutation table
 *   receive     - receives the specified number of bytes from the supervisor
 *                 which are referenced by the given pointer
 *   receive_ref - receives refutation table
 *   send        - sends information to the supervisor (opposite of receive)
 *   search      - returns the position of the given move in the refutation tbl
 *   enter       - enters the given move in the triangular table
 *   tenter      - enters moves that were found in the transposition table into
 *                 the triangular table
 *   valid       - determines if current board position is valid
 */

int  tdepth;      /* length of subtrees to be stored in transposition table */
int  CTSflag;     /* whether RTS/CTS handshaking should occur */
int  maxdepth     /* maximum depth of search */

short  ref[MAXWIDTH][MAXDEPTH];   /* refutation table  */
short  var[MAXDEPTH][MAXDEPTH];   /* triangular table  for storing variations */
short *lmp;

play()
{
        register short score, value, *fmp, *mp, princ;
        register int i, first, path, depth;
        short *statl(), mws(), alphabeta(), maxpos;
        char mask[100];
        struct package pck;
```

```
fmp = statl();       /* generate all legal moves from current position */


/*
 * initialize the refutation table
 */

clear_ref();
for (path = 0, mp = fmp; mp < lmp; mp += 2, path++)
        ref[path][0] = mp[1];

mp = fmp;
first  = 1;
depth  = 1;

while (depth <= maxdepth) {        /* iterative deepening  */

        if (fmp != lmp) {           /* were there any legal moves? */

                for (i = 0; i < 100; i++)
                        mask[i] = 0;        /* clear the mask */

                maxpos = princ = lmp - fmp;

                /*
                 * if this is the first iteration, we will already
                 * have a valid movelist and the refuation table will
                 * be empty.  Otherwise, we have to receive this
                 * information from the supervisor
                 */

                if (!first) {
                        receive(fmp,(lmp-fmp) * 2);
                        receive_ref(depth);
                } else
                        first = 0;

                path = search(fmp[1]);   /* find position of first move in refutation table */

                make(fmp[1]);
                score = -mws(depth-1);  /* recursively call mws */
                undo(fmp[1]);

                receive(&pck,4);
                CTSflag = 0;            /* we don't need the RTS/CTS transmission */
                score = pck.val;        /* get the initial best score */
                mp = fmp + pck.num * 2;  /* and the initial move */

                while (mp < lmp) {
                        mask[(mp - fmp) / 2] = 1;  /* we did this one */
                        mp++;
                        path = search(*mp);
        make(*mp);
        value = -alphabeta(-score-1, -score, depth-1);

        /* if the zero-width search failed high, search with larger window */

        if (value > score) {
                score = value = -alphabeta(-MAXINT, -value, depth-1);
                enter(*mp);         /* enter move in triangular table */
                princ = mp - fmp;  /* set princ to position in list  */

        }


        for (i = 1; i <= depth; i++)        /* update refutation table */
                ref[path][i] = var[1][i];

        mp[-1] = value;               /*  store new value for the move  */
        undo(*mp);
        mp++;                         /* update pointer to next move    */


        /*
         * inter-node communication:  send back the value
         * and "princ" to the supervisor and get a new
         * score along with a new move
         */

        pck.num = princ;
        pck.val = value;
        send(&pck,4);

        receive(&pck,4);
        score = pck.val;
        mp += pck.num * 2;
        princ = maxpos;
}


pck.val = DONE;
send(&pck,4);       /* tell supervisor that we are done         */
receive(&pck,4);

score = pck.val;

/* send the refutations for those moves */

for (mp = fmp+2, i = 1; mp < lmp; mp += 2, i++)
        if (mask[i]) {
                path = search(mp[1]);
                send(ref[path],(depth+1) * 2);
        }

CTSflag = 1;
```

```
                }
                depth++;
        }
        lmp = fmp;
        return(score);
}
```

```
short mws(depth)
int depth
{
        register short score, value, *fmp, *mp;
        register int i;
        struct package pck;
        short   tmove, alphabeta(), quies(), princ, maxpos;
        char    mask[100];

        tmove = 0;

        if (depth <= 0)
                return(0);

        mp = fmp = lmp;

        gen();                  /* generate all moves onto move stack */
        sort(fmp, lmp);          /* sort the moves */

        maxpos = princ = lmp - fmp;

        for (i = 0; i < 100; i++)
                mask[i] = 0;

        receive(&tmove,2);  /* receive the move from the transposition      */
        if (tmove == 0)     /* or refutation table. Does such a move exist? */
                return(0);

        make(tmove);
        score = -mws(depth-1);
        undo(tmove);

        receive(&pck,4);
        score = value = pck.val;        /* get initial score       */
        mp += pck.num * 2;              /* along with initial move */
        CTSflag = 0;

        while (mp < lmp) {              /* while there are still moves to do... */
                mask[(mp-fmp) / 2] = 1;
                mp++;
                if (*mp != tmove) {     /* do the search if it wasn't from the table */
                        tblf++;
                        make(*mp);

                        if (valid()) {  /* was this a legal move */

                                value = -alphabeta(-score-1, -score, depth-1);
                                if (value > score) {
                                        score = value = -alphabeta(-MAXINT, -value, depth-1);
                                        enter(*mp);
                                        princ = mp-fmp;
                                }
                        }
                        undo(*mp);
                        tblf--;
                }
                mp++;

                /* some more "inter-node" communication */

                pck.val = value;
                pck.num = princ;
                send(&pck,4);

                receive(&pck,4);
                score = pck.val;
                mp += pck.num * 2;
                princ = maxpos;
        }

        pck.val = DONE;
        pck.num = princ;
        send(&pck,4);

        receive(&pck,4);
        princ = pck.num;
        score = pck.val;
        CTSflag = 1;

        /* if we searched the principal variation, pass back the refutation */

        if (princ != 0  &&  mask[(princ-1) / 2]) {
                ply = maxdepth - depth;
                send(&var[ply][ply],(depth+1) * 2);
        }

        lmp = fmp;
        return(score);
}
```

```
short alphabeta(alpha, beta, depth)
short alpha, beta;
int depth;
{
        register short score, value, *fmp, *mp, bestmove;
        int  savetblf;
        short   tmove, tscore, Index, quies();
        char    tflag, tlen;

        tmove = 0;

        /*
         * the transposition table if the "transf" flag is set and if our
         * current depth is greater than or equal to the "tdepth"
         */

        if (transf && depth >= tdepth) {
                Index = index % TBL_SIZE;
                retrieve(INDEX, Index, &tlen, &tscore, &tflag, &tmove);

                if ((tmove != 0) && (tlen >= depth)) {
                        switch (tflag) {
                                case VALID:               /* enter the valid move into */
                                        tenter(tmove);    /* the refutation table and  */
                                        return(tscore);   /* return                    */

                                case LBOUND:                     /* otherwise, use the */
                                        if (tscore > alpha)      /* score to improve   */
                                                alpha = tscore;  /* the search window  */
                                        break;

                                case UBOUND:
                                        if (tscore < beta)
                                                beta = tscore;
                                        break;
                        }
                }
        }

        if (depth <= 0 ) {
                score = quies(beta);    /* perform capture search (quiescent search) */
                return(score);
        }

        /*
         * if the move is both in the refutation table and the transposition table
         * use the transposition table move
         */

        savetblf = 0;
        if (!tblf) {
                bestmove = tmove;
                tmove = ref[path][maxdepth-depth];
                if ((bestmove != 0) && (bestmove != tmove)) {
                        savetblf = 1;
                        tmove = bestmove;
                }
        }

        bestmove = 0;
        fmp = lmp;

        /* if we have a table move, search it first */

        if (tmove != 0) {
                make(tmove);
                if (savetblf) tblf++;

                if (valid()) {
                        score = -alphabeta(-beta, -alpha, depth-1);
                        enter(bestmove = tmove);
                } else tmove = 0;


                if (savetblf) tblf--;
                undo(tmove);
                if (score >= beta)
                        goto out;
        }

        mp = fmp = lmp;
        gen();
        sort(fmp, lmp);

        while (mp != lmp) {             /* go through the remaining moves */
                mp++;
                if (*mp != tmove) {
                        tblf++;
                        make(*mp);

                        if (valid()) {


                                value = -alphabeta(-beta, (alpha > v1)? -alpha: -score, depth-1);

                                if (value > score) {
                                        score = value;
                                        enter(bestmove = *mp);
                                }
                        }
                        undo(*mp);
                        tblf--;
                        if (score >= beta)
                                goto out;
                }
                mp++;
        }
out:
        /* should we store the bestmove in the transposition table? */

        if (transf && (depth >= tdepth) && (tlen <= depth) && (bestmove != 0)) {
                tflag = (score <= alpha)? LBOUND: (score >= beta)? UBOUND: VALID;
                store (INDEX, Index, length, score, tflag, bestmove);
        }
        lmp = fmp;
        return(score);
}
```

```
/*
 * This section of code corresponds to the supervisor activity for
 * inter-node communication.  An interrupt generated by data from a
 * slave will result in the "process" routine being called at this time.
 * The supervisor is know as processor 0, with the slaves going from 1 to n_ports
 * "Process" cannot be interrupted by usart interrupts.
 */

char     isdone[N_PORTS+1];        /* flags stating whether processor is done     */
short   *last[N_PORTS+1];          /* position of last move allocated             */
short    count[N_PORTS+1];         /* number of moves processor will skip         */
short    princ;                    /* position of best move                       */
short    pid                       /* current processor number                    */
short    bestid;                   /* processor that found best move              */
short    score;                    /* best score                                  */
short   *fmp;                      /* pointer to first move on stack              */
short    n_ports = N_PORTS;        /* number of external processors               */

process()
{
        register int pid, j;
        struct package pck;

        for (pid = 1; pid <= n_ports; pid++) {
                if (done_move(pid)) {                   /* If the processor has done a move... */
                        readx(pid,&pck,4,INTR);         /* ... read a package of information   */
                        if (pck.val == DONE)               /* have we run out of moves to do?      */
                                isdone[pid] = 1;
                        else {
                                fmp[last[pid] * 2] = pck.val;  /* store the value of the move */

                                /*
                                 * if a processor has found a better score, or if a move of
                                 * equal merit has been found which appears earlier in the
                                 * movelist, then it is our best move
                                 */

                                if (pck.val > score  || (pck.val == score && pck.num < princ )) {
                                                princ  = pck.num;
                                                score  = pck.val;
                                                bestid = pid;
                                }

                                /* send back the new information   */

                                pck.num = count[pid];   /* how many moves to skip */
                                pck.val = score;           /* best score              */
                                writex(pid,&pck,4);

                                /* tell everyone else we've taken another move */

                                for (j = 0; j <= n_ports; j++)
                                        count[j]++;
                                last[pid] += count[pid];   /* which move are we doing  */
                                count[pid] = 0;
                        }
                }

        }
}


enter(move)             /* enter move in triangular table of variations */
short move;
{
        int     i;

        var[ply][ply] = move;
        i = ply;
        while (i++ < depth)
                var[ply][i] = var[ply+1][i];
}
}
```

APPENDIX B: Chess position data from the Bratko-Kopec experiment.

```
    Kb    Rb    ::    ::
 Pb Pb :: Bb :: Rw ::
    ::    Qb    :: Pb Pb
 ::    ::    Pb    ::
    :: Bw ::    ::    ::
 ::    ::    QW    ::
 Pw Pw Pw ::    Bw    ::
 ::    KW    ::    ::
JUL 31,        BLACK'S MOVE.
```

```
2K5PPP2B24Q32B5885R28-88884P33Q2PPPP1B41K1R4-
Board A:  Best move is:  Qd6d1+

2R55K21NR5P2PPPP17P+8888P71PPR3P4NPP13R1K2+
Board B:  Best move is:  d4d5

7R2BQ1PRK1P2BNNPP1P1P1P13P4-8881P62P1P3P2P1PP13BBNNP2Q1RR1K-
Board C:  Best move is:  f6f5

R1B1KB1RPPP1QPPP2P53N44P3+88882PP41P6P3PPPPRNBQKB1R+
Board D:  Best move is:  e5e6

2KR1B1RPPP3PP2N2N1Q3P1P1B8+88883P3P2N1PP2PBPQ42KR1BNR+
Board I:  Best move is:  f4f5

R1B3K12P1NQPPPP1R1P288-8883P481QN2NP1PP3PP13RR1K1-
Board J:  Best move is:  Nf6e5

R4RK14QPPP2PBB1N1P1P1P33P4+8888N1P1P3BP1P4P2Q1PPP2R1NRK1+
Board K:  Best move is:  f2f4

R3R1K1PP3PPP2P586NQ-88884P38PPQB1PPPR3R1K1-
Board L:  Best move is:  Bd7f5

R1B2RK1PPB1QP22N2N1P2P1P1P13P4-88884P3P2P1NB11PPNBPPPR2Q1RK1-
Board Q:  Best move is:  h7h5

R1B1QRK11PB3PPN1P2N2P3PP28-88882N52NP2P1PP2PPBPR1BQ1RK1-
Board R:  Best move is:  Ne5b3

4RRK1P5PP1P62QBPP28-88888P2P1PNP2PQ2PK3RR3-
Board S:  Best move is:  Re8xe4

2KRR3PPP1Q2P2N3P13P1P24N3+88883P1P21P1QP2PPB2BP1RR4K2+
Board T:  Best move is:  g3g4
```

```
 Rb :: Bb ::    Rb Kb ::
 ::    Qb    Bb    Pb Pb
 Pb ::    Pb Pb Nb    ::
 :: Pb ::    ::    ::
    ::    QW Pw ::    ::
 :: Bw Nw    Bw    ::
 Pw Pw Pw ::    :: Pw Pw
 Rw    ::    :: Rw KW
JUL 31,        WHITE'S MOVE.
```

```
R4RK1PPP3PP1BN1B33QP38+88881P6P2PPN22Q1B1PPR1B2RK1+
Board E:  Best move is:  Nc3d5 or a2a4

8P1P51P4K15P24P1P183R48+888884P3PPP2PP12R3K1+
Board F:  Best move is:  g5g6

R2Q2K12P1B1PPB1P2R23P1P24P2N+888B1P5Q2P1P24P3PP2N1PP1NK1R1R1+
Board G:  Best move is:  Nh5f6

8P3N2P4K1P13P1P24P3+8882P53P3P6P1P3KP24B3+
Board H:  Best move is:  f4f5

R3R1K1PP1B1PPP3Q44P33P4+8883P42P5P2P44BPPPR2Q1RK1+
Board M:  Best move is:  b2b4

R2Q1RK1PB2PPBP1P3NP183P4+882B58Q4P22PP2P1PP2P2PRNB2R1K+
Board N:  Best move is:  Qd1d2

5RK1P1PN2PP1P1PP1R16Q18+8888P1PP42B1PR21P2Q1PP2R3K1+
Board O:  Best move is:  Qg4xg7+

R2Q1RK1PPPN1PPP1B684P1B1+88881P1P4P1P4P4NPP1R1BQKB1R+
Board P:  Best move is:  Nd2e4

3RR1K1PB3PPP1P5Q2P1P35N2+888882PPQP2PPB2RPP3RN2K+
Board U:  Best move is:  Nf5h6

R2R2K11B2QPP1P2B1N1PN1P1P31P6-8888P71P1PPN1P1BQNBPP12R2RK1-
Board V:  Best move is:  Bb7xe4

R4RK11PP3PP2N1B3P2Q1P24P3-88883P42P5PP2BPPPR1BQK2R-
Board W:  Best move is:  f7f6

5RK13QNPPPPRNBB31PP1P33P4+88882P1PP21P1P2PPP2B2B1R2QNRNK+
Board X:  Best move is:  f2f4
```