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 move lists 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

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

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

\[
N_d = w \left\lfloor \frac{d}{2} \right\rfloor + w \left\lceil \frac{d}{2} \right\rceil - 1
\]  

(1)

where \( \lfloor x \rfloor \) and \( \lceil x \rceil \) 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 interprocessor 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:

\[
T_d = k + T_{d-1} + (w-1)B_{d-1}, \quad \text{for } d > 0,
\]

(2)

where \( T_0 = k + we \)

and \( B_d = 2k + wB_{d-2}, \quad \text{for } d > 1, \)

(3)

where \( B_0 = k + e \)

and \( B_1 = 2k + we \)

After some substitutions equation (2) becomes

\[
T_d = (d+1)k + we + (w-1)\sum_{i=0}^{d-1} B_i
\]

(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

\[
E_d = T_d - T_{d-1}
= k + (w-1)B_{d-1}
\]

Thus when \( d \) is odd this is equivalent to

\[
E_{2m+1} = k + (w-1)B_{2m}
\]
\[ = k + (w-1) \left[ 2k \sum_{i=0}^{m-1} w^i + (k+e)w^m \right] \]
\[ = k + (w-1) \left[ 2k \sum_{i=0}^{m} w^i + ew^m - kw^m \right] \quad (5) \]

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:

\[ E_{2m+1} = 1 + (w-1)\left[ \frac{2(w^{m+1}-1)}{w-1} - w^m \right] \]
\[ = 1 + 2w^{m+1} - 2 - w^{m+1} + w^m \]
\[ N_{2m+1} = w^{m+1} + w^m - 1 = E_{2m+1} \quad (6) \]

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

\[ T_{d,f} = k + T_{d-1,f} + \left[ \frac{(w-1)}{f} \right] B_{d-1}, \quad \text{for } d > 0, \text{ and } f > 0, \quad (7) \]

where \(T_{0,f} = k + e + \left[ \frac{(w-1)}{f} \right] e.\)

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

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

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

Finally, when interprocessor communication is included,

\[ T_{d,f} = k + T_{d-1,f} + \left[ \frac{(w-1)}{f} \right] B_{d-1} + (w-1)\left( \frac{(f-1)}{f} (s+t_1) + (f-1)(s+t_2) \right) \quad (9) \]

for \(d > 0, \text{ and } f > 0,\) 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

\[ B_d = 2k + wB_{d-1} + gC_{d-1}, \quad \text{for } d > 1 \]  \hspace{1cm} (10)

where $B_0 = k + (1+g)e$ and $B_1 = 2k + we + gC_0$

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

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

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

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

\[ E_{2m+1,f} = k + \left\lceil \frac{(w-1)}{f} \right\rceil B_{2m} \]  \hspace{1cm} (12)

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

\[ B_{2m} = 2k + wB_{2m-2} + gC_{2m-1} \]

\[ = 2k + wB_{2m-2} + g\left[k \sum_{i=0}^{2m-2} (1+g) + e(1+g)^{2m}\right] \]

That is,

\[ B_{2m} = A_{2m} + wB_{2m-2} \]  \hspace{1cm} (13)

where $A_{2m} = k + (k+ge)(1+g)^{2m}$
Thus
\[ B_{2m} = \sum_{i=0}^{m-1} A_{2^i(n-i)} w^i + w^m B_0 \]
\[ = k \sum_{i=0}^{m-1} w^i + (k+ge) \sum_{i=0}^{m-1} (1+g)^{2^i(n-i)} w^i + w^m (k + (1+g)e) \]
\[ = k \sum_{i=0}^{m} w^i + (k+ge)(1+g)^{2m} \sum_{i=0}^{m-1} \left[ \frac{w}{(1+g)^i} \right] + w^m ((1+g)e) \] (14)

Finally, if we make the simplifying assumption that \((w-1) \mod f = 0\), then equation (12) becomes
\[ E_{2m+1,f} = k + \frac{w-1}{f} B_{2m} \] (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)
\[ E_3 = w^3 + w^2(1+g)^3 + w((1+g)^4 - (1+g)^3) - (1+g)^4 \] (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 \(E_3\) and \(w\). Typical experimental values for \(w\) and \(E_3\) have been presented previously [MARS 83]. For our purposes it is sufficient to note that for \(w=35\), \(g=1.3\) when \(E_3=50,000\) and \(g=4.2\) when \(E_3=100,000\).
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


<table>
<thead>
<tr>
<th>board</th>
<th>one processor</th>
<th>two processors</th>
<th>three processors</th>
<th>four processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nodes</td>
<td>secs</td>
<td>move</td>
<td>nodes</td>
</tr>
<tr>
<td>A</td>
<td>(forced mate)</td>
<td>d6d1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>39412</td>
<td>806</td>
<td>e4e5</td>
<td>39520</td>
</tr>
<tr>
<td>C</td>
<td>29861</td>
<td>1189</td>
<td>e7d8</td>
<td>29952</td>
</tr>
<tr>
<td>D</td>
<td>57294</td>
<td>816</td>
<td>e5e6</td>
<td>59526</td>
</tr>
<tr>
<td>E</td>
<td>98258</td>
<td>2617</td>
<td>f1f5</td>
<td>98259</td>
</tr>
<tr>
<td>F</td>
<td>12591</td>
<td>100</td>
<td>g5g6</td>
<td>13597</td>
</tr>
<tr>
<td>G</td>
<td>44755</td>
<td>1303</td>
<td>a3b4</td>
<td>44809</td>
</tr>
<tr>
<td>H</td>
<td>4867</td>
<td>38</td>
<td>e2c3</td>
<td>4934</td>
</tr>
<tr>
<td>I</td>
<td>77113</td>
<td>2607</td>
<td>f3g1</td>
<td>90465</td>
</tr>
<tr>
<td>J</td>
<td>69208</td>
<td>2142</td>
<td>d8d7</td>
<td>69521</td>
</tr>
<tr>
<td>K</td>
<td>53830</td>
<td>1356</td>
<td>g3f5</td>
<td>54098</td>
</tr>
<tr>
<td>L</td>
<td>33510</td>
<td>819</td>
<td>d7f5</td>
<td>33834</td>
</tr>
<tr>
<td>M</td>
<td>75173</td>
<td>1078</td>
<td>a1c1</td>
<td>75360</td>
</tr>
<tr>
<td>N</td>
<td>31864</td>
<td>600</td>
<td>d1d2</td>
<td>34694</td>
</tr>
<tr>
<td>O</td>
<td>22047</td>
<td>523</td>
<td>g4g7</td>
<td>22547</td>
</tr>
<tr>
<td>P</td>
<td>67809</td>
<td>1393</td>
<td>g5f4</td>
<td>73237</td>
</tr>
<tr>
<td>Q</td>
<td>38027</td>
<td>1370</td>
<td>d7c5</td>
<td>38779</td>
</tr>
<tr>
<td>R</td>
<td>86254</td>
<td>2118</td>
<td>c8g4</td>
<td>86291</td>
</tr>
<tr>
<td>S</td>
<td>36379</td>
<td>862</td>
<td>c7c5</td>
<td>36506</td>
</tr>
<tr>
<td>T</td>
<td>127897</td>
<td>2614</td>
<td>d1d2</td>
<td>127714</td>
</tr>
<tr>
<td>U</td>
<td>50266</td>
<td>2004</td>
<td>f5d4</td>
<td>92110</td>
</tr>
<tr>
<td>V</td>
<td>40242</td>
<td>1191</td>
<td>d7e5</td>
<td>40422</td>
</tr>
<tr>
<td>W</td>
<td>173866</td>
<td>8070</td>
<td>c8f5</td>
<td>21801</td>
</tr>
<tr>
<td>X</td>
<td>70893</td>
<td>1438</td>
<td>b4c5</td>
<td>70990</td>
</tr>
<tr>
<td>tot</td>
<td>1381516</td>
<td>37064</td>
<td></td>
<td>1455178</td>
</tr>
<tr>
<td>ave</td>
<td>60065</td>
<td>1611</td>
<td></td>
<td>63268</td>
</tr>
</tbody>
</table>

Table 1: 5-ply search without transposition table.
<table>
<thead>
<tr>
<th>board</th>
<th>one processor</th>
<th>two processors</th>
<th>three processors</th>
<th>four processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nodes secs move</td>
<td>nodes secs speedup move</td>
<td>nodes secs speedup move</td>
<td>nodes secs speedup move</td>
</tr>
<tr>
<td>A</td>
<td>33950 652 e4e5</td>
<td>34094 522 1.25 e4e5</td>
<td>34101 424 1.54 e4e5</td>
<td>34063 480 1.36 e4e5</td>
</tr>
<tr>
<td>B</td>
<td>27322 1051 e7d8</td>
<td>27643 712 1.48 e7d8</td>
<td>28065 545 1.93 e7d8</td>
<td>28147 580 1.81 e7d8</td>
</tr>
<tr>
<td>C</td>
<td>52052 705 e5e6</td>
<td>51181 606 1.16 e5e6</td>
<td>51188 494 1.42 e5e6</td>
<td>53177 605 1.16 e5e6</td>
</tr>
<tr>
<td>D</td>
<td>97509 2546 f1f5</td>
<td>97509 1778 1.43 f1f5</td>
<td>97523 1374 1.85 f1f5</td>
<td>97524 1417 1.80 f1f5</td>
</tr>
<tr>
<td>E</td>
<td>11968 89 g5g6</td>
<td>12075 117 0.76 g5g6</td>
<td>12303 126 0.71 g5g6</td>
<td>13395 174 0.51 g5g6</td>
</tr>
<tr>
<td>F</td>
<td>44169 1253 a3b4</td>
<td>44579 889 1.41 a3b4</td>
<td>44849 754 1.66 a3b4</td>
<td>90576 1606 0.78 f3g3</td>
</tr>
<tr>
<td>G</td>
<td>4151 34 e2c3</td>
<td>4157 43 0.80 e2c3</td>
<td>4205 41 0.84 e2c3</td>
<td>4172 58 0.60 e2c3</td>
</tr>
<tr>
<td>H</td>
<td>62177 1865 f3e5</td>
<td>63576 1267 1.47 f3e5</td>
<td>63574 985 1.89 f3e5</td>
<td>63610 994 1.88 f3e5</td>
</tr>
<tr>
<td>I</td>
<td>96759 3831 d8d6</td>
<td>96737 2518 1.52 d8d6</td>
<td>96867 1846 2.08 d8d6</td>
<td>96683 1964 1.95 d8d6</td>
</tr>
<tr>
<td>J</td>
<td>52905 1315 g3f5</td>
<td>53580 962 1.37 g3f5</td>
<td>53559 759 1.73 g3f5</td>
<td>53666 805 1.63 g3f5</td>
</tr>
<tr>
<td>K</td>
<td>33634 795 d7f5</td>
<td>33978 608 1.31 d7f5</td>
<td>34013 483 1.64 d7f5</td>
<td>34379 552 1.49 d7f5</td>
</tr>
<tr>
<td>L</td>
<td>74706 1051 a1c1</td>
<td>74731 892 1.18 a1c1</td>
<td>75058 730 1.45 a1c1</td>
<td>75291 853 1.24 a1c1</td>
</tr>
<tr>
<td>M</td>
<td>30633 537 d1d2</td>
<td>32042 458 1.17 d1d2</td>
<td>33584 390 1.38 d1d2</td>
<td>35026 433 1.24 d1d2</td>
</tr>
<tr>
<td>N</td>
<td>21177 485 g4g7</td>
<td>21411 382 1.27 g4g7</td>
<td>21634 340 1.42 g4g7</td>
<td>21613 366 1.33 g4g7</td>
</tr>
<tr>
<td>O</td>
<td>51382 1059 g5e7</td>
<td>50820 813 1.30 g5e7</td>
<td>51024 709 1.49 g5e7</td>
<td>51613 797 1.33 g5e7</td>
</tr>
<tr>
<td>P</td>
<td>36321 1269 d7c5</td>
<td>36446 895 1.48 d7c5</td>
<td>36557 664 1.91 d7c5</td>
<td>37151 712 1.78 d7c5</td>
</tr>
<tr>
<td>Q</td>
<td>81373 1908 c8g4</td>
<td>81419 1365 1.40 c8g4</td>
<td>81479 1068 1.79 c8g4</td>
<td>81530 1117 1.71 c8g4</td>
</tr>
<tr>
<td>R</td>
<td>35946 818 c7c5</td>
<td>35991 610 1.34 c7c5</td>
<td>36229 491 1.66 c7c5</td>
<td>36321 551 1.48 c7c5</td>
</tr>
<tr>
<td>S</td>
<td>126803 2577 d1d2</td>
<td>126806 1954 1.32 d1d2</td>
<td>129117 1589 1.61 d1d2</td>
<td>130288 1820 1.42 d1d2</td>
</tr>
<tr>
<td>T</td>
<td>88397 1888 f5d4</td>
<td>90194 1426 1.32 f5d4</td>
<td>90302 1112 1.70 f5d4</td>
<td>90353 1227 1.54 f5d4</td>
</tr>
<tr>
<td>U</td>
<td>40226 1157 d7e5</td>
<td>40746 757 1.53 d7e5</td>
<td>40943 889 1.30 d7e5</td>
<td>41035 975 1.19 d7e5</td>
</tr>
<tr>
<td>V</td>
<td>32711 1712 e8g8</td>
<td>52786 1206 1.42 e8g8</td>
<td>52817 943 1.82 e8g8</td>
<td>51987 1076 1.59 e8g8</td>
</tr>
<tr>
<td>W</td>
<td>70931 1464 b4c5</td>
<td>71003 1089 1.34 b4c5</td>
<td>71053 851 1.72 b4c5</td>
<td>71039 910 1.61 b4c5</td>
</tr>
<tr>
<td>X</td>
<td>1227033 30075</td>
<td>1234504 21838</td>
<td>1240924 17629</td>
<td>1292658 20062</td>
</tr>
<tr>
<td>ave</td>
<td>53349 1307</td>
<td>53674 949 1.31</td>
<td>53953 766 1.59</td>
<td>56202 872 1.41</td>
</tr>
</tbody>
</table>

Table 2: 5-ply search with global transposition table.
<table>
<thead>
<tr>
<th>board</th>
<th>one processor</th>
<th>two processors</th>
<th>three processors</th>
<th>four processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nodes secs move</td>
<td>nodes secs speedup move</td>
<td>nodes secs speedup move</td>
<td>nodes secs speedup move</td>
</tr>
<tr>
<td>A</td>
<td>(forced mate) d6d1</td>
<td>34092 339 1.92 e4e5</td>
<td>d6d1</td>
<td>34210 235 2.77 e4e5</td>
</tr>
<tr>
<td>B</td>
<td>33950 652 e4e5</td>
<td>27970 597 1.76 e7d8</td>
<td>28785 401 2.62 e7d8</td>
<td>34279 186 3.49 e4e5</td>
</tr>
<tr>
<td>C</td>
<td>27322 1051 e7d8</td>
<td>52203 360 1.96 e5e6</td>
<td>52902 258 2.73 e5e6</td>
<td>29238 354 2.97 e7d8</td>
</tr>
<tr>
<td>D</td>
<td>52052 705 e5e6</td>
<td>97509 2546 f1f5</td>
<td>98012 898 2.83 f1f5</td>
<td>61078 333 2.11 e5e6</td>
</tr>
<tr>
<td>E</td>
<td>97509 2546 f1f5</td>
<td>11968 89 g5g6</td>
<td>13255 45 1.95 g5g6</td>
<td>88545 667 3.82 f1f5</td>
</tr>
<tr>
<td>F</td>
<td>11968 89 g5g6</td>
<td>44169 1253 a3b4</td>
<td>67303 773 1.62 f3g3</td>
<td>13470 42 2.11 g5g6</td>
</tr>
<tr>
<td>G</td>
<td>44169 1253 a3b4</td>
<td>4572 21 1.62 e2c3</td>
<td>4642 16 2.05 e2c3</td>
<td>85896 812 1.54 f3g3</td>
</tr>
<tr>
<td>H</td>
<td>4151 34 e2c3</td>
<td>63647 978 1.91 f3e5</td>
<td>63686 676 2.76 f3e5</td>
<td>4498 15 2.19 e2c3</td>
</tr>
<tr>
<td>I</td>
<td>62177 1865 f3e5</td>
<td>96913 1349 2.84 e8e6</td>
<td>96456 1109 3.45 d8d6</td>
<td>63749 555 3.36 f3e5</td>
</tr>
<tr>
<td>J</td>
<td>96759 3831 d8d6</td>
<td>53861 474 2.77 g3f5</td>
<td>53785 372 3.53 g3f5</td>
<td>96456 1109 3.45 d8d6</td>
</tr>
<tr>
<td>K</td>
<td>52906 1315 g3f5</td>
<td>33364 795 d7f5</td>
<td>33497 294 2.70 d7f5</td>
<td>69006 963 0.83 f7f5</td>
</tr>
<tr>
<td>L</td>
<td>33364 795 d7f5</td>
<td>74706 1057 a1c1</td>
<td>75234 368 2.87 a1c1</td>
<td>75569 290 3.64 a1c1</td>
</tr>
<tr>
<td>M</td>
<td>74706 1057 a1c1</td>
<td>32079 305 1.76 d1d2</td>
<td>33622 224 2.39 d1d2</td>
<td>35064 187 2.87 d1d2</td>
</tr>
<tr>
<td>N</td>
<td>30633 537 d1d2</td>
<td>21471 277 1.75 g4g7</td>
<td>21633 212 2.29 g4g7</td>
<td>21721 172 2.82 g4g7</td>
</tr>
<tr>
<td>O</td>
<td>21177 485 g4g7</td>
<td>51382 1059 g5e7</td>
<td>51706 419 2.52 g5h6</td>
<td>52317 362 2.92 g5h6</td>
</tr>
<tr>
<td>P</td>
<td>51382 1059 g5e7</td>
<td>3621 1269 d7c5</td>
<td>37395 463 2.74 d7c5</td>
<td>38530 432 2.94 d7c5</td>
</tr>
<tr>
<td>Q</td>
<td>3621 1269 d7c5</td>
<td>83836 1002 1.90 c8g4</td>
<td>84708 707 2.70 c8g4</td>
<td>85137 551 3.46 c8g4</td>
</tr>
<tr>
<td>R</td>
<td>83836 1002 1.90 c8g4</td>
<td>36161 422 1.94 c7c5</td>
<td>36463 299 2.73 c7c5</td>
<td>36557 241 3.39 c7c5</td>
</tr>
<tr>
<td>S</td>
<td>36161 422 1.94 c7c5</td>
<td>126928 1323 1.95 d1d2</td>
<td>128476 949 2.72 d1d2</td>
<td>131183 769 3.35 d1d2</td>
</tr>
<tr>
<td>T</td>
<td>126928 1323 1.95 d1d2</td>
<td>91037 992 1.90 f5d4</td>
<td>91386 686 2.75 f5d4</td>
<td>91628 531 3.55 f5d4</td>
</tr>
<tr>
<td>U</td>
<td>91037 992 1.90 f5d4</td>
<td>40746 619 1.87 d7e5</td>
<td>40944 633 1.83 d7e5</td>
<td>41036 573 2.02 d7e5</td>
</tr>
<tr>
<td>V</td>
<td>40746 619 1.87 d7e5</td>
<td>52711 1712 e8g8</td>
<td>53128 638 2.68 e8g8</td>
<td>52344 575 2.97 e8g8</td>
</tr>
<tr>
<td>W</td>
<td>52711 1712 e8g8</td>
<td>70931 1464 b4c5</td>
<td>71182 526 2.78 b4c5</td>
<td>71324 487 3.51 b4c5</td>
</tr>
<tr>
<td>X</td>
<td>70931 1464 b4c5</td>
<td>71166 764 1.92 b4c5</td>
<td>1272943 11555 57930 457 2.91</td>
<td></td>
</tr>
<tr>
<td>tot</td>
<td>1227033 30075</td>
<td>1241858 15751</td>
<td>1332407 10520</td>
<td>-22-</td>
</tr>
<tr>
<td>board</td>
<td>one processor</td>
<td>two processors</td>
<td>three processors</td>
<td>four processors</td>
</tr>
<tr>
<td>-------</td>
<td>---------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>nodes secs move</td>
<td>nodes secs speedup move</td>
<td>nodes secs speedup move</td>
<td>nodes secs speedup move</td>
</tr>
<tr>
<td>A</td>
<td>d6d1</td>
<td>d6d1</td>
<td>d6d1</td>
<td>d6d1</td>
</tr>
<tr>
<td>B</td>
<td>e4e5</td>
<td>e4e5</td>
<td>e4e5</td>
<td>e4e5</td>
</tr>
<tr>
<td>C</td>
<td>e7d8</td>
<td>e7d8</td>
<td>e7d8</td>
<td>e7d8</td>
</tr>
<tr>
<td>D</td>
<td>e5e6</td>
<td>e5e6</td>
<td>e5e6</td>
<td>e5e6</td>
</tr>
<tr>
<td>E</td>
<td>f1f5</td>
<td>f1f5</td>
<td>f1f5</td>
<td>f1f5</td>
</tr>
<tr>
<td>F</td>
<td>g5g6</td>
<td>g5g6</td>
<td>g5g6</td>
<td>g5g6</td>
</tr>
<tr>
<td>G</td>
<td>a3b4</td>
<td>a3b4</td>
<td>a3b4</td>
<td>a3b4</td>
</tr>
<tr>
<td>H</td>
<td>e2c3</td>
<td>e2c3</td>
<td>e2c3</td>
<td>e2c3</td>
</tr>
<tr>
<td>I</td>
<td>f3e5</td>
<td>f3e5</td>
<td>f3e5</td>
<td>f3e5</td>
</tr>
<tr>
<td>J</td>
<td>e8e7</td>
<td>e8e7</td>
<td>e8e7</td>
<td>e8e7</td>
</tr>
<tr>
<td>K</td>
<td>g3f5</td>
<td>g3f5</td>
<td>g3f5</td>
<td>g3f5</td>
</tr>
<tr>
<td>L</td>
<td>d7f5</td>
<td>d7f5</td>
<td>d7f5</td>
<td>d7f5</td>
</tr>
<tr>
<td>M</td>
<td>a1c1</td>
<td>a1c1</td>
<td>a1c1</td>
<td>a1c1</td>
</tr>
<tr>
<td>N</td>
<td>d1d2</td>
<td>d1d2</td>
<td>d1d2</td>
<td>d1d2</td>
</tr>
<tr>
<td>O</td>
<td>g4g7</td>
<td>g4g7</td>
<td>g4g7</td>
<td>g4g7</td>
</tr>
<tr>
<td>P</td>
<td>g5e7</td>
<td>g5e7</td>
<td>g5e7</td>
<td>g5e7</td>
</tr>
<tr>
<td>Q</td>
<td>d7c5</td>
<td>d7c5</td>
<td>d7c5</td>
<td>d7c5</td>
</tr>
<tr>
<td>R</td>
<td>c8g4</td>
<td>c8g4</td>
<td>c8g4</td>
<td>c8g4</td>
</tr>
<tr>
<td>S</td>
<td>c7c5</td>
<td>c7c5</td>
<td>c7c5</td>
<td>c7c5</td>
</tr>
<tr>
<td>T</td>
<td>d1d2</td>
<td>d1d2</td>
<td>d1d2</td>
<td>d1d2</td>
</tr>
<tr>
<td>U</td>
<td>f5d4</td>
<td>f5d4</td>
<td>f5d4</td>
<td>f5d4</td>
</tr>
<tr>
<td>V</td>
<td>d7e5</td>
<td>d7e5</td>
<td>d7e5</td>
<td>d7e5</td>
</tr>
<tr>
<td>W</td>
<td>e8g8</td>
<td>e8g8</td>
<td>e8g8</td>
<td>e8g8</td>
</tr>
<tr>
<td>X</td>
<td>b4c5</td>
<td>b4c5</td>
<td>b4c5</td>
<td>b4c5</td>
</tr>
<tr>
<td>tot</td>
<td>122288 15412</td>
<td>1249914 11167</td>
<td>1270152 9252</td>
<td>1270152 9252</td>
</tr>
<tr>
<td>ave</td>
<td>52705 1275</td>
<td>53427 670</td>
<td>54344 485</td>
<td>55224 402</td>
</tr>
</tbody>
</table>

Table 4: 5-ply search with local, depth-limited, transposition table.
<table>
<thead>
<tr>
<th></th>
<th>one processor</th>
<th>two processors</th>
<th>three processors</th>
<th>four processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>board</td>
<td>nodes</td>
<td>secs</td>
<td>move</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>(forced mate)</td>
<td>d6d1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>33922</td>
<td>661</td>
<td>e4e5</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>29337</td>
<td>1152</td>
<td>e7d8</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>53613</td>
<td>747</td>
<td>e5e6</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>98059</td>
<td>2603</td>
<td>f1f5</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>12000</td>
<td>95</td>
<td>g5g6</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>44242</td>
<td>1269</td>
<td>a3b4</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>4086</td>
<td>33</td>
<td>e2c3</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>61210</td>
<td>1882</td>
<td>f3e5</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>74334</td>
<td>2439</td>
<td>d8d5</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>53085</td>
<td>1328</td>
<td>g3f5</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>73489</td>
<td>815</td>
<td>d7f5</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>74994</td>
<td>1070</td>
<td>a1c1</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>30636</td>
<td>566</td>
<td>d1d2</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>21184</td>
<td>499</td>
<td>g4g7</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>52772</td>
<td>1108</td>
<td>g5e7</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>37332</td>
<td>1353</td>
<td>d7c5</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>81427</td>
<td>1940</td>
<td>c8g4</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>36034</td>
<td>844</td>
<td>c7c5</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>126937</td>
<td>2504</td>
<td>d1d2</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>88425</td>
<td>1910</td>
<td>f5d4</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>40241</td>
<td>1185</td>
<td>d7e5</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>59297</td>
<td>1745</td>
<td>e8g8</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>71062</td>
<td>1498</td>
<td>b4c5</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>71185</td>
<td>549</td>
<td>e4e5</td>
<td></td>
</tr>
<tr>
<td>tot</td>
<td>121228</td>
<td>29345</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ave</td>
<td>52705</td>
<td>1275</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>54178</td>
<td>498</td>
<td>2.50</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: 5-ply search with global, depth-limited, transposition table.
<table>
<thead>
<tr>
<th></th>
<th>one processor</th>
<th>two processors</th>
<th>three processors</th>
<th>four processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nodes</td>
<td>min</td>
<td>move</td>
<td>nodes</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td>138063</td>
</tr>
<tr>
<td>B</td>
<td>132751</td>
<td>45</td>
<td>e4e5</td>
<td>136711</td>
</tr>
<tr>
<td>C</td>
<td>141201</td>
<td>74</td>
<td>e7d8</td>
<td>136711</td>
</tr>
<tr>
<td>D</td>
<td>131304</td>
<td>42</td>
<td>e5e6</td>
<td>136944</td>
</tr>
<tr>
<td>E</td>
<td>351867</td>
<td>227</td>
<td>f1f5</td>
<td>47869</td>
</tr>
<tr>
<td>F</td>
<td>42942</td>
<td>8</td>
<td>g5g6</td>
<td>239839</td>
</tr>
<tr>
<td>G</td>
<td>228815</td>
<td>148</td>
<td>h5f6</td>
<td>15671</td>
</tr>
<tr>
<td>H</td>
<td>14780</td>
<td>2</td>
<td>e2c3</td>
<td>178305</td>
</tr>
<tr>
<td>I</td>
<td>174398</td>
<td>111</td>
<td>f3e5</td>
<td>545042</td>
</tr>
<tr>
<td>J</td>
<td>436938</td>
<td>202</td>
<td>e8e6</td>
<td>255469</td>
</tr>
<tr>
<td>K</td>
<td>249238</td>
<td>126</td>
<td>g3f5</td>
<td>170502</td>
</tr>
<tr>
<td>L</td>
<td>165223</td>
<td>62</td>
<td>d7f5</td>
<td>192818</td>
</tr>
<tr>
<td>M</td>
<td>189128</td>
<td>72</td>
<td>a1c1</td>
<td>89935</td>
</tr>
<tr>
<td>N</td>
<td>83343</td>
<td>39</td>
<td>d1d2</td>
<td>64520</td>
</tr>
<tr>
<td>O</td>
<td>212469</td>
<td>100</td>
<td>d2e4</td>
<td>187435</td>
</tr>
<tr>
<td>P</td>
<td>191081</td>
<td>141</td>
<td>d7c5</td>
<td>190063</td>
</tr>
<tr>
<td>Q</td>
<td>347935</td>
<td>191</td>
<td>c8g4</td>
<td>371298</td>
</tr>
<tr>
<td>R</td>
<td>200645</td>
<td>81</td>
<td>c7c5</td>
<td>205487</td>
</tr>
<tr>
<td>S</td>
<td>677076</td>
<td>315</td>
<td>c3b5</td>
<td>393672</td>
</tr>
<tr>
<td>T</td>
<td>367754</td>
<td>194</td>
<td>f5h6</td>
<td>391851</td>
</tr>
<tr>
<td>U</td>
<td>130017</td>
<td>86</td>
<td>d7e5</td>
<td>130688</td>
</tr>
<tr>
<td>V</td>
<td>531720</td>
<td>204</td>
<td>e8g8</td>
<td>541273</td>
</tr>
<tr>
<td>W</td>
<td>176158</td>
<td>78</td>
<td>b4c5</td>
<td>176184</td>
</tr>
<tr>
<td>tot</td>
<td>5239891</td>
<td>2590</td>
<td></td>
<td>5179129</td>
</tr>
<tr>
<td>ave</td>
<td>227821</td>
<td>112</td>
<td></td>
<td>225179</td>
</tr>
</tbody>
</table>

Table 8: 6-ply search with local, depth-limited, transposition table.
/*
 * 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 *stati(), mws(), alphabeta(), maxpos;
    char mask[100];
    struct package pck;
}
Appendix A: Parabelle Source Code

fmp = stat1();  /* generate all legal moves from current position */

/ *
 * initialize the refutation table
 */
clear_ref();
for (path = 0, mp = fmp; mp < Imp; mp += 2, path++)
    ref[path][0] = mp[1];

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

while (depth <= maxdepth) {  /* iterative deepening */
    if (fmp != Imp) {  /* were there any legal moves? */
        for (i = 0; i < 100; i++)
            mask[i] = 0;  /* clear the mask */

        maxpos = princ = Imp - fmp;

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

        if (lfirst) {
            receive(fmp,(Imp-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 < Imp) {
            mask[(mp - fmp) / 2] = 1;  /* we did this one */
            mp++;
            path = search(*mp);

-27-
Appendix A: Parabelle Source Code

make(*np);
value = -alphabet(-score-1, -score, depth-1):

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

if (value > score) {
    score = value = -alphabet(-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;
Appendix A: Parabelle Source Code

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

    tmove = 0;
    if (depth <= 0)
        return(0);
    mp = fmp = 1mp;

    gen();          /* generate all moves onto move stack */
    sort(fmp, 1mp); /* sort the moves */
    maxpos = princ = 1mp - 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 < 1mp) { /* 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 */
            tbl++;  
            make(*mp);
            if (valid()) { /* was this a legal move */
                value = -alphabet(-score-1, -score, depth-1);
                if (value > score) {
                    score = value = -alphabet(-MAXINT, -value, depth-1);
                    enter(*mp);
                    princ = mp-fmp;
                }
            }
        }
    }
```
Appendix A: Parabelle Source Code

```c
}
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);
}
imp = fmap;
return(score);
```
Appendix A: Parabelle Source Code

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 (ttblf) {
        bestmove = tmove;
        tmove = ref[path][maxdepth-depth];
Appendix A: Parabelle Source Code

```c
if ((bestmove != 0) && (bestmove != tmove)) {
    savetblf = 1;
    tmove = bestmove;
}

bestmove = 0;
mp = 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++;
}
```
Appendix A: Parabelle Source Code

```c
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);
    }
    imp = fmp;
    return(score);
```
 Appendix A: Parabelle Source Code

/*
 * 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 uart 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)
                isdone[pid] = 1;
            else {
                fmp[pid] = 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]++;
        }
    }
}
Appendix A: Parabelle Source Code

    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 :
Pw Pw Pw : : : : Bw :
JUL 31, BLACK'S MOVE.

2K5PPP2B24Q32B5885R28-88884P3Q2PPPP1B41K1R4-
Board A: Best move is: Qd6d4+

2R55K21NR5P2PPP2P17P+8888P71P3P3P44NPP13R1K2+
Board B: Best move is: d4d5

7R2BQ1PRK1P2BNNPP1P1P1P13P4-8888P62P1P3P2PP1P13BNNP2Q1RR1K-
Board C: Best move is: f6f5

R1B1KB1PPP10PPP2P53N44P3+88888P41P6P3PPPPrN8NQK1R1+
Board D: Best move is: e5e6

2KR1B1RPPP3P2P2N21Q3P1P1B8+88883P3P21P1P2PPQ42KR18NR+
Board I: Best move is: f4f5

R1B3K12P1N0QPPP1R1P28B-88883P481Q2N1P1PP3P13R1K1-
Board J: Best move is: Nf6e6

R4RK1QPPP2PBB1N1P1P1P33P4+88881N1P3B1P4P4Q2Q1PPPPR218RK1+
Board K: Best move is: f2f4

R3R1K1PPP3P2P536N0-88884P38PPQ81P3PR3R1K1-
Board L: Best move is: Bsd7f5

R1B2RK1PP81QP22N2P1P1P1P13P4-88884P3P21PN1B1PPN8PPPR2Q1RR1K-
Board Q: Best move is: h7h8

R1B1Q0R1K1P83PN1P2N2P3P28-88882N52N2P21P2PPBPR1BQ1RK1-
Board R: Best move is: Ne8b3

4R3K1P5PP1P62QP2P28-88888P21PP2PQ2P2PK33P3-
Board S: Best move is: Re8xe4

2KRR3PPP1Q2P3N3P13P1P24N3+88883P1P21P1QP2PPB2BP1RR4K2+
Board T: Best move is: g3g4

Rb : : : : : : :
Pb : : : : : : :
Pb Pb : : Bb : : Rw : : :
: : Qb : : : : Pb Pb :
Pw Pw Pw : : : : Bw :
JUL 31, WHITE'S MOVE.

R4RK1PPP3P1BN1B330P34+88881P62PPN2Q1B1PR1B2R1K1+
Board E: Best move is: Nc3d5 or a2a4

8P1P51P4K15P24P1P183R48+88884P3PPP2PP12R3K1+
Board F: Best move is: g5g6

R2Q2K12P1B1PPB1P2R2P1P24P2N+88881P5Q2P1P24P3PP2N1PP1N1K1R1R1+
Board G: Best move is: Nh5f6

8P3N2P4K1P13P1P24P3+88882P53P36P1P3K2P4B3+
Board H: Best move is: f4f5

R3R1K1PP1B1PP3Q44P33P4+88883P42P5P2P44BPPR2Q1KR1+
Board M: Best move is: b2b4

R2Q1RK1PB2PPBP1P3NP183P4+88885Q4P42PP2P1P2PPR8B2R1K+
Board N: Best move is: Qd1d2

5RK1P1P2PP1P1PP1R16Q1B+88881P42B1P2R12Q1P2P2R3K1+
Board O: Best move is: Qg4xg7+

R2Q1RK1PPP1N1P1B684P1B1+88881P41P4P4NPP1R1BQKB1R1+
Board P: Best move is: Nd2e4

3R1K1PB3PP1P5Q2P1P3SN2+88882PQPQ82PP2RP2PP3R2N2K+
Board Q: Best move is: Nf5h5

R2R2K11B2QPP1P821NP1N1P1P31P6-88887P1P1PPN1P1BQNP12R2R1K1-
Board V: Best move is: Bb7xe4

R4RK11PPP2N1B3P2Q1P24P3-88883P42P5PP2PPR1BQK2R-
Board W: Best move is: f7f6

5RK130NPRPPRN8B31P1P33P4+88882P1P21P1P2PPB2BP12Q5RNK+
Board X: Best move is: f2f4