Parallel Game-tree Search

T.A. Marsland

and

F. Popowich

Computing Science Department
University of Alberta
EDMONTON T6G 2H1
Canada

DRAFT to be

submitted to the IEEE Transactions on PAMI

January 1984

Parallel Game-tree Search

T.A. Marsland

and

F. Popowich

Computing Science Department
University of Alberta
EDMONTON T6G 2H1
Canada

ABSTRACT

The design issues affecting a parallel implementation of the alpha-beta search algorithm are discussed with emphasis on a tree decomposition scheme which is intended for use on well ordered trees. In particular, the Principal Variation splitting method has been implemented, and experimental results are presented which show how such refinements as progressive deepening, narrow window searching and the use of memory tables affect the performance of multiprocessor based chess-playing programs. When dealing with parallel processing systems, communication delays are perhaps the greatest source of lost time. Therefore, an implementation of our tree decomposition based algorithm is presented that can operate with a limited amount of communication within a network of processors. This system has almost negligible search overhead, and so the principal basis for comparison is the communication overhead, based on a new mathematical model of this component.

Keywords:

multiprocessors, concurrent programming, message passing, graph and tree search strategies, tree decomposition, alpha-beta search.

Acknowledgement

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 tree searching algorithm are adapted for use on a parallel processing system, the speed-up (reduction in search time) is notoriously less than the number of processors in the system. These losses are due to **search overhead** and **communication overhead.** The **search overhead** is algorithm dependent, since it is defined as the extra work that a parallel algorithm carries out compared to its sequential counterpart. Since an alpha-beta search uses accumulated information to determine when cutoffs are to occur, a parallel implementation may have one processor still calculating the better cutoff value that another processor could use to terminate its execution. Thus the total work done, and hence the search overhead, may be higher. In contrast, **communication overhead** results from the necessary exchange of information between processors, and is therefore dependent on the system configuration as well as the algorithm. Thus, parallel alpha-beta is potentially susceptible to search overhead losses, and compensating for this overhead may entail major communication delays.

Since the effectiveness of one sequential alpha-beta algorithm, the Principal Variation Search (PVS), has been demonstrated [1], a parallel version, dubbed PV splitting, has been devised. Previous research has not yet determined the most effective way of using N processors to search a game tree of indefinite size. When a large number of processors ($N > 1000$) is available, it is possible that the most effective organization will be totally different from the one which works best for only a few processors ($N < 10$). Several groups have considered the division of work problem for relatively small values of N, and proposals have been made based on theoretical, experimental and simulation results.

The approaches to sharing the work differ, and they include partitioning the search window between processors [2], assigning individual processors to separate subtrees [3], and managing a pool of processors [4][5]. Each has its advantages and disadvantages. For example, by dividing the search window into disjoint ranges and assigning one processor to each partition, one can guarantee that the best continuation can be found, and in less time than a single processor searching over the full window. However, since each processor must examine some minimal tree, it has been shown that the speed-up is limited to a factor of about five, no matter how many processors are applied [2]. The direct tree decomposition approach (applying one processor per subtree) [3] suffers from a similar minimal tree-search disadvantage, although the amount of work done per processor may be smaller. More seriously, it is difficult to share information between processors, yet this is vital for best performance. Finally, some methods which hold promise in terms of efficiency and information sharing properties employ a pool of processors, but here a large memory is needed, to store partially evaluated nodes as they await an available processor. Aside from the storage requirement, these methods are difficult to analyze in terms of their communication overhead [5].

We have implemented a version of PV splitting on a loosely coupled network of processors. As in window splitting [2], our interest is with the application of relatively few processors (about a dozen), but with a view to exploring the theoretical five-fold speed-up limit of the simple methods. To regulate the flow of messages, a processor tree architecture is used.

2. Principal Variation Search (PVS)

In the PVS algorithm, alternatives to the first move are assumed inferior until proven otherwise. They are examined with a minimal window of (alpha, alpha+1), based on a bound obtained from the search of the best candidate so far. Any move (sub-tree) searched with such a zero-width (minimal) window will necessarily fail. Most of these searches will "fail low" because the move is easily refuted. Whenever a better move is found, the preliminary search "fails high" and it must be repeated with more appropriate bounds, in order to determine the correct value for the new candidate. PVS takes advantage of the fact that game trees are rarely random, and that any knowledge about the application domain can be used to pre-order the move list and hence favorably bias the shape of the game tree.

An alpha-beta algorithm suitable for doing this search, since it can return values below the range of the given window [6], is shown in Figure 1. It is called *pvs* and uses application dependent functions *generate*, to form an array of successors to the given position, *empty*, to determine if no successors exist, *sizeof*, to return the size of the array, and *evaluate* to estimate the value of the subtree at a leaf node. These functions are not presented, but in our case-study form part of a chess-playing program. For simplicity some other details have been omitted, particularly use of *make* and *undo*, to update and restore the given move respectively. The mode of presentation of the programs throughout this paper is a Pascal-like pseudo code, extended with a *return* statement for function exit, although our actual implementations are done in the C language [7]. The algorithm ensures that the best available score is returned, even when all the moves have a value that is less than the alpha bound. In many respects, *pvs* is similar to SCOUT [8][9], and the Calphabeta algorithm [6], from which the notion of a minimal window search is drawn.

There are several enhancements to alpha-beta implementations that improve their performance dramatically. Perhaps the most important is **progressive** or **iterative deepening** [10]. Rather than embarking immediately upon a maximum depth search, a series of shallower searches becoming successively deeper is used. The moves are sorted after each iteration, thus providing a more plausibly ordered move list for the next search phase, which in turn should result in a quicker search. This method is only effective when used in conjunction with a **refutation table** or a **transposition table.** For example, after each D-ply search (that is, to depth D) a sequence of moves from the root to a leaf node is stored in a **refutation table** [1]. For the selected move this table contains the best variation, while for the others it holds a D-ply sequence sufficient to refute the move. Upon a D+1 ply

```
FUNCTION pvs(p : position; alpha,beta,depth : integer) : integer;
                              { p is pointer to the current node     }
                              { alpha and beta are window bounds     }
                              { depth is the remaining search length }
                              { the value of the subtree is returned }
  VAR score, i, value : integer;
      posn : ARRAY[1..MAXWIDTH] OF position;
                                           { assert depth positive  }
BEGIN
  IF depth = 0 THEN                        { leaf, maximum depth?   }
     return(evaluate(p));
                                           { produce a pointer to   }
  posn := generate(p);                     { an array of successors }
  IF empty(posn) THEN                      { leaf, no moves?        }
     return(evaluate(p));
                                             { principal variation? }
  score := -pvs(posn[1], -beta, -alpha, depth-1);
  FOR i := 2 TO sizeof(posn) DO BEGIN
     IF (score >= beta) THEN                                 { cutoff? }
        return(score);
     alpha := max(score, alpha);
                                            { minimal window search }
     value := -pvs(posn[i], -alpha-1, -alpha, depth-1);
     IF (value > alpha) THEN
        IF (value < beta) THEN           { if "fail-high", re-search }
           score := -pvs(posn[i], -beta, -value, depth-1)
        ELSE score := value;
  END  {forloop};
  return(score);
END  {pvs};
```

Figure 1. Depth-Limited Principal Variation Search.

search, the table is used to direct each variation along a potential refutation and thus significantly reduce the search time. This knowledge table is of modest size, being linear with search depth, and is cheap to maintain. One can also implement a more comprehensive table, one which holds not only the preferred move in a position, but also bounds on the value of the associated subtree. This **transposition table** (a direct access memory table of results for subtrees already examined [11]) is more powerful since it can also be used to improve search windows, provide more cutoffs, and to extend the effective search depth [12]. Progressive deepening is an important idea because in game-tree searches most of the time is sent exploring the principal variation. Any enhancement which increases the probablity that the PV will be examined very early in the search improves the performance, since the balance of the tree is discarded more quickly.

Figure 2. Processor Tree Structure.

3. The Principal Variation Splitting Algorithm

PV splitting is based on PVS and, unlike other methods, delays tree decomposition until the first path along the Principal Variation has been searched. Thereafter individual subtrees are examined by a hierarchy of processors arranged in a tree. A processor tree consists of computers and communication lines corresponding to tree nodes and branches respectively, Figure 2. Any processor can communicate directly with only its parent and its children, but the root processor is designated the master. There is also a master/slave relationship between a parent and its children but, by supplying appropriate software, an individual processor can behave as a both parent and child.

When direct tree decomposition is used for parallel implementations of alpha-beta algorithms, some tree cut-offs may not

occur, since more moves will be examined without the benefit of a new cutoff value [2]. This becomes more evident as processor tree fanout increases. In our PV splitting algorithm, on the other hand, since the most plausible move is analyzed by all processors, not only is the first variation searched faster but the subsequent decomposition also ensures that all the processors at a given level start with a good window bound, thus providing more cutoffs and allowing the balance of the search to proceed more quickly. Consequently the search overhead is reduced.

Figure 3 illustrates a version of the PV splitting algorithm which uses the following constructs adapted from Fishburn [6].

1. *j.treesplit* represents the execution of a direct tree-partitioning algorithm by processor "j". *Treesplit* may also use the processor tree architecture, but is not presented here since it is adequately described elsewhere [11].
2. PARFOR initiates a parallel loop which conceptually creates a separate process for each iteration of the loop. The program continues as a single process when all loops are complete.
3. WHEN causes the parent to wait until the associated condition is true before proceeding with the body of the statement.
4. CRITICAL allows only one process at a time into the next block of code.
5. *terminate* kills all currently active processes in the PARFOR loop.

For simplicity, we have assumed in Figure 3 that the processor tree length is less than the depth of the search tree.

Our current implementation of PV splitting uses a processor tree of length one. Consequently, the *treesplit* and *pvsplit* routines may be simplified into one called *pmws,* Figure 4. It is this simplification which is used in our analytical model and performance studies which follow. *Pmws* stands for Parallel Minimal Window Search, since the alternate variations are searched in parallel with a zero-width window. Although some of the details concerning move updates have been omitted from the *pmws* function, we have managed to illustrate progressive deepening with function *main.* The *transmit.j* and *receive.j* functions are used to exchange information between processor j and its parent, while *j.child_pvs* represents the execution of the *child_pvs* procedure by processor "j". The major disadvantage of this method is that only a single processor is available to search any new principal variation which subsequently emerges, since no tree splitting occurs at that time. To overcome this restriction one would have to apply all the processors to the re-search of the new candidate, just as they were for the primary principal variation. Finally, in the interests of simplicity we have indicated by comments the locations where communication is necessary to update the best path to a leaf. These communications are far less frequent than those illustrated by the use of transmit/receive, Figure 4.

4. Software System

To examine overheads 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

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

```
FUNCTION pvsplit(p : position;
                 alpha, beta, depth, length : integer) : integer;
                                { length is the processor tree height }
                                              { assume length <= depth }
  VAR i, value, score, height : integer;
      j : processor;
      posn : ARRAY[1..MAXWIDTH] OF position;
BEGIN
  IF length = 0 THEN                           { end of processor tree }
     return(pvs(p, alpha, beta, depth));
                                               { produce a pointer to }
  posn := generate(p);                        { an array of successors }

  IF empty(posn) THEN                                 { no legal moves }
     return(evaluate(p));

  height := length;
  IF height >= depth THEN                       { apply processor tree }
     height := height -1;
  score := -pvsplit(posn[1], -beta, -alpha, depth-1, height);

  PARFOR i := 2 TO sizeof(posn) DO          { loop through successors }
    WHEN (a child j is idle) BEGIN
      IF score >= beta THEN BEGIN                            { cutoff? }
        terminate();
        return(score);
      END;
      CRITICAL alpha := max(alpha, score);

      value := -j.treesplit(posn[i], -beta,-alpha, depth-1,length-1);
      CRITICAL score := max(score, value);
    END;
  return(score);
END;
```

Figure 3. Principal Variation Splitting.

uni-processor version of the program.

The basis of both the sequential and parallel chess programs used in our tests is the Principal Variation Search. This method presumes a good ordering of the moves so that the most likely candidate is searched first, while the remaining variations are examined using a zero-width window. Progressive deepening, an iterative search with dynamic reordering of the moves, along with a refutation table, containing the main continuation for each move at the first level in the tree, are used since they have shown their merit in previous tests [1]. The use of a transposition table, containing the results from nodes seen during the search, is included in order to study the consequences of implementating such a large table in a parallel system.

Parabelle analyzes chess positions on a processor tree network. All of the processors recursively analyze the first move,

```
FUNCTION main(p : position; maxdepth : integer) : integer;
  VAR score, i, j, value, depth : integer;
      posn : ARRAY[1..MAXWIDTH] OF position;
BEGIN
  posn := generate(p);
  IF empty(posn) THEN
    return(evaluate(p))
  for depth := 1 to maxdepth DO BEGIN        { progressive deepening }
    score := -pmws(posn[1], depth-1);
    PARFOR i := 2 TO sizeof(posn) DO
      WHEN (child processor j is idle) DO BEGIN
        transmit.j(posn[i], score, depth);  { send best score to "j" }
        j.child_pvs;                       { invoke child_pvs on "j" }
        receive.j(posn[i].value);            { accept subtree value }
        CRITICAL score := max(score, posn[i].value);
      END;
    sort(posn);   { communication of refutation table update omitted }
  END;
  return(score);
END;

FUNCTION pmws(p : position; depth : integer) : integer;
                                  { note: processor tree length is 1 }
  VAR score, i, j, value : integer;
      posn : ARRAY[1..MAXWIDTH] OF position;
BEGIN
  IF depth = 0 THEN
    return(evaluate(p));
  posn := generate(p);
  IF empty(posn) THEN
    return(evaluate(p));
  score := -pmws(posn[1], depth-1);
  PARFOR i := 2 TO sizeof(posn) DO
    WHEN (child processor j is idle) DO BEGIN
      transmit.j(posn[i], score, depth);              { send to "j" }
      j.child_pvs;                { processor "j" executes child_pvs }
      receive.j(posn[i].value);
      CRITICAL score := max(score, posn[i].value);
                        { communication of better move update omitted }
    END;
  return(score);
END;

PROCEDURE child_pvs;
  VAR value, score, depth : integer; p : position;
BEGIN
  receive.m(p, score, depth);                  { receive from master }
  value := -pvs(p, -score-1, -score, depth-1);
  IF value > score THEN               { research with correct window }
    value := -pvs(p, -MAXINT, -value, depth-1);
  transmit.m(value);                                { send to master }
END;
```

Figure 4. Parallel Minimal Window Search

with the remainder being examined by individual processors using a zero-width window. There is a local refutation table for each processor that is updated after each phase of the progressive deepening search. A transposition table, which can be accessed on either a global or local basis, is also included. When a global transposition table is used, then all processors have access to the same subtree results which are stored in the master processor. Any benefits of this arrangement may be cancelled by the increase in the communication overhead of the system. On the other hand, 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.

5. Hardware Configuration

Parabelle is written in C for use on a Motorola 68000 based Multiple Instruction stream Multiple Data stream (MIMD) system [13]. The program itself requires approximately 48K bytes of space, while the the rest of each unit's 256K bytes of memory may be used for data storage. Each unit contains identical software for searching chess positions, with a master possessing an additional 128K bytes of memory to hold extra code for global table management and communication interface routines. The system used for our experiments with the *pmws* algorithm consists of four identical SUN Workstation boards [14], in a depth one processor tree configuration, Figure 5, but can be extended to handle nine processors. With a system of four processors, preliminary simulations of PV splitting have shown that a processor tree of depth one is slightly superior to one of depth two [11].

Inter-unit communication is handled at a maximum rate of 9600 baud. Communication between the processors is channeled though an eight-port Serial Communication Interface (SCI) [15]. This device currently resides in the master workstation where direct memory data transfer can occur between it and the SUN processor. Communication between the SCI and the other units is done over RS-232 data lines. Additional SCI units could be installed to allow processor trees of greater depth. Since the SCI takes care of the character by character transmission of data, the host (master) processor need only write to, and read from, the appropriate buffers to communicate with the individual processors. The SCI can also interrupt the host whenever data is available and consequently allow the host to act as a parent processor, when it is handling an interrupt, or as a child when it is in normal operating mode.

Whereas the master processor has its input buffered by the SCI, the bare SUN workstation can only buffer two characters. As a result, the master must not be allowed to transmit to the children unless it can be sure that no data will be lost. One way that this problem may be handled is by having the child transmit the Request-To-Send (RTS) signal when it desires input. The SCI will not transmit unless it receives this signal (Clear-To-Send). Fortunately this extra protocol is not necessary for the bulk of the

Figure 5. System Configuration.

messages, since they are of the form "read immediately after write", and so blocks of information from the master can always be accepted.

6. Inter-Unit Communication

When a deterministic game tree is searched, it is possible to take advantage of the fact that each processor can generate identical move lists for the current position. As a consequence, they can all recursively search the principal variation, and so explore the same first path (sequence of moves to a leaf). For the remaining moves, the master informs an idle child of the best score, and identifies the next subtree to search. In turn, the child tells the master what value it found for the subtree it was searching. This exchange is fundamental to the PV split method and is referred to as **inter-node** communication. It is brief, four bytes containing the best score and next move indicator, and occurs only during searches at PV nodes, marked with a 1 in Figure 6.

A few other inter-unit communications occur. In particular, an **Inter-level** message is sent to the master after the search of a PV node has been completed. This message is required in order to pass back the best refutation sequence and its value. The length of the message depends only on the length of the refutation line transferred, that is, the depth of the subtree searched, but the message is only sent by slaves which actually find a better line.

When using the progressive deepening refinement, **inter-iteration** communication at the root node is also required. Between iterations, the move list is first sorted by the master and then passed on to its slaves. The master refutation table is updated and similarly transferred. A large amount of information transfer, proportional to the width of the search tree, must take place at this point, but this communication delay can be tolerated since only a handful of iterations are usually performed.

If a transposition table is used, all the processors must be able to search and update this large direct access hash table. In one of our experiments, the transposition table was managed by the master processor. The children access this global table via **at-node** communication. Whenever a child processor is about to execute *pvs* on a node within its assigned search tree, a 6 byte request message (consisting of a hash access key) is sent to the master to see if a table entry exists for that node. If the access is successful, the master passes back 6 bytes consisting of the move found and its score, along with an indication of the score's reliability [11]. With this information, the child can narrow its search window or even avoid searching the subtree altogether. After completing its search, the child transmits to the master a 12 byte update message consisting of the two information packets just described. Unlike the other forms of communication, **at-node** messages can be very frequent.

Finally, there are **interactive** messages, to allow the startup of the individual processors, the setting of system configuration variables, and other forms of external communication. Input to the master processor is echoed to all of the children. **Interactive** communication involves tasks such as updating the board configuration, obtaining the opponent's move, and even setting the maximum search depth. Typically, these are messages to the user's console and do not affect the performance of the

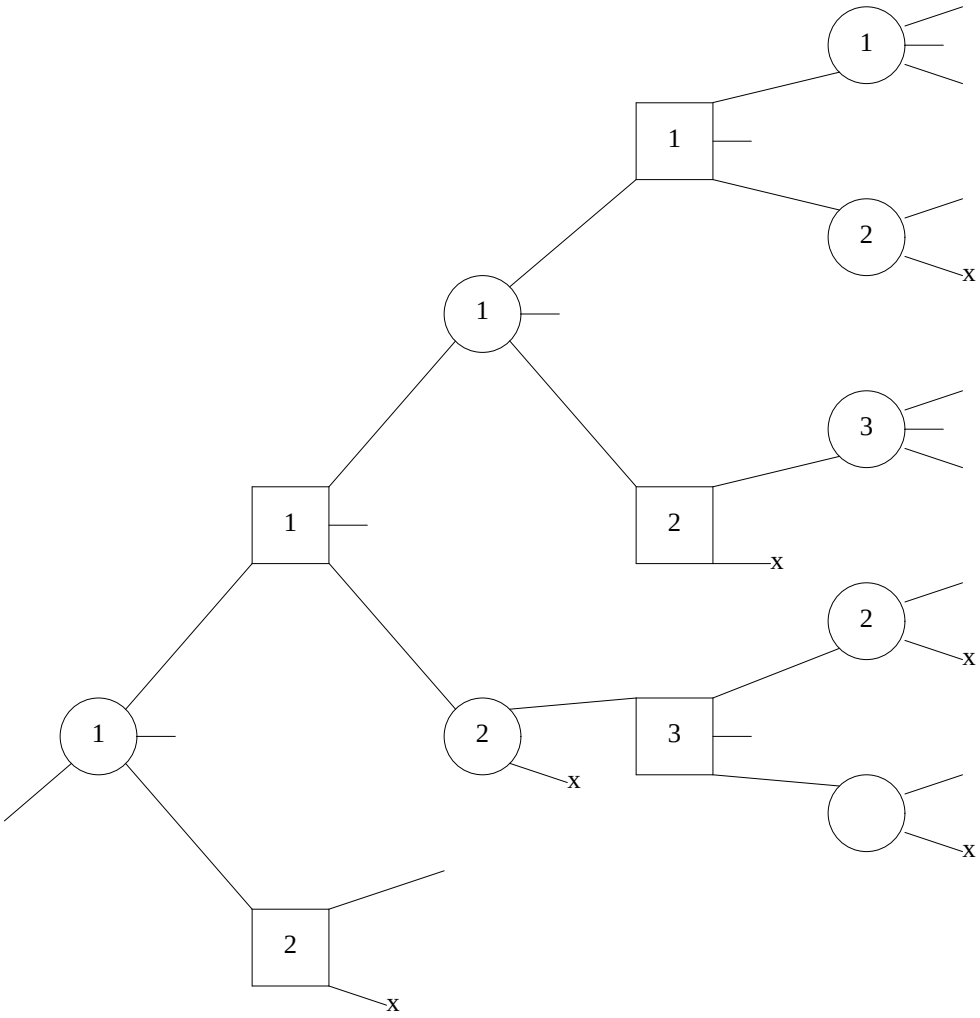

Figure 6. Properties of an alpha-beta search tree

system.

The different types of communication, along with their length are summarized in decreasing order of frequency in Table 1. The **at-node** communication is optional in the sense that use of a global transposition table is not mandatory. In fact, since the child may be idle when it is waiting for a response to its request, it may be prudent to suppress this communication and perhaps implement a transposition table local to each individual processor. Certainly our experimental results, Tables 2 and 3, indicate this to be the case. Another possibility would be for the master to interrupt the child if and only if the position is found in the

transposition table. Thus the child would not have to wait for a negative response and so could proceed immediately with its search, discarding its work only if a successful retrieval occurs.

Type	Frequency	Length
at-node	before and after each node in the tree (optional)	6 or 12 bytes
inter-node	before and after each successor of a PV node	4 bytes
inter-level	after each PV node	2 bytes for each ply from leaf
inter-iteration	before and after each progressive deepening phase	$4w + 2iw$ bytes, w is tree width at the root, i the iteration number
interactive	occur in response to user requests	varies

Table 1. Types of Inter-processor Communication.

7. 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 PV nodes, Figure 6, 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

d = depth of search.
e = time to evaluate a single move at a leaf node.
f = number of processors (fanout).
g = average number of "cut-off" branches searched.
k = time to create node and generate move list.
s = setup time in handling a message.
w = number of branches (moves) on any node (position).
p_f = ratio of d to $d-1$ ply search time using f processors.
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.
C_d = time to search a "cut-off" branch to depth d.
E_d = cost of evaluating the leaf nodes of an alpha-beta tree.
N_d = number of leaf nodes in a minimal alpha-beta tree.
T_d = time to search a minimal tree to depth d.

Note that both t_1 and t_2 are proportional not only to the communication link speed but also to the message length.

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

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

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

From Figure 6 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:

$$T_d = k + T_{d-1} + (w-1)B_{d-1} \qquad (for\ d > 0)$$
where $T_0 = k + we$
and $B_d = 2k + w\,B_{d-2} \qquad (for\ d > 1)$ (2)
where $B_0 = k + e$
and $B_1 = 2k + we$

After some substitutions equation (2) becomes

$$T_d = (d+1)k + we + (w-1)\sum_{i=0}^{d-1} B_i \qquad (3)$$

Note that from equation (3) the cost of evaluating the leaf nodes of a 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 and $m > 0$, this is equivalent to

$$\begin{aligned} E_{2m+1} &= k + (w-1)B_{2m} \\ &= k + (w-1)\left[2k \sum_{i=0}^{m-1} w^i + (k+e)w^m\right] \\ &= k + (w-1)\left[2k \sum_{i=0}^{m} w^i + ew^m - kw^m\right] \end{aligned} \tag{4}$$

Equation (4) represents the cost, in processor cycles, of evaluating the leaf 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 leaf nodes in the minimal tree, and hence reduces to equation (1) as follows:

$$\begin{aligned} E_{2m+1} &= 1 + (w-1)\left[\frac{2(w^{m+1}-1)}{w-1} - w^m\right] \\ &= 1 + 2w^{m+1} - 2 - w^{m+1} + w^m \\ N_{2m+1} &= w^{m+1} + w^m - 1 \quad = \quad E_{2m+1} \end{aligned} \tag{5}$$

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 (3) has been provided.

While equations (2) and (3) may be used to estimate the tree traversal time for a uniprocessor they account neither for multiprocessors using tree splitting nor for inter-unit communication. When a processor tree of length 1, with fan-out of $'f'$, executes PV splitting the search time is

$$T_{d,f} = k + T_{d-1,f} + \left\lceil \frac{(w-1)}{f} \right\rceil B_{d-1} \qquad (\text{for } d > 0 \text{ and } f > 0) \tag{6}$$

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

As in (4) we can now calculate (for $d = 2m+1$)

$$E_{d,f} = T_{d,f} - T_{d-1,f} = k + \left\lceil \frac{(w-1)}{f} \right\rceil B_{2m}$$

and so with the simplifying assumption that $(w-1) \bmod f = 0$, as well as $k=1$ and $e=0$, then the equivalent form of equation (5) becomes

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

where $N_{2m+1,\,f}$ is the average number of leaf nodes in the minimal tree examined per processor.

Since tree splitting only occurs at PV nodes, the formula for B_d, equation (2), still applies.

Finally, when **inter-node** and **inter-level** communications are included,

$$T_{d,\,f} = k + T_{d-1,\,f} + \lceil \frac{(w-1)}{f} \rceil B_{d-1} + (w-1)\frac{(f-1)}{f}(s+t_1) + (f-1)(s+t_2) \qquad (7)$$

for $d > 0$ and $f > 0$, where $T_{0,\,f}$ is similarly modified.

7.1. 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 (7) for T_d is still correct, for $d > 1$ equation (2) becomes

$$B_d = 2k + wB_{d-2} + gC_{d-1} \qquad (8)$$

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

and $C_d = k + (1+g)C_{d-1}$

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 leaf nodes can be determined from equation (6) as

$$E_{d,\,f} = T_{d,\,f} - T_{d-1,\,f} = k + \lceil \frac{(w-1)}{f} \rceil B_{d-1} \qquad (9)$$

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

$$E_{2m+1,\,f} = k + \lceil \frac{(w-1)}{f} \rceil B_{2m} \qquad (10)$$

where B_{2m} is given by equation (8) as

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

which is equivalent to,

$$B_{2m} = A_{2m} + wB_{2m-2} \tag{11}$$

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

Thus

$$\begin{aligned} B_{2m} &= \sum_{i=0}^{m-1} A_{2(m-i)}\ w^i + w^m B_0 \\ &= k \sum_{i=0}^{m-1} w^i + (k+ge) \sum_{i=0}^{m-1} (1+g)^{2(m-i)}\ w^i + w^m (k + (1+g)e) \\ &= k \sum_{i=0}^{m} w^i + (k+ge)(1+g)^{2m} \sum_{i=0}^{m-1} [\frac{w}{(1+g)^2}]^i + w^m ((1+g)e) \end{aligned} \tag{12}$$

Finally, if we make the simplifying assumption that $(w-1)$ mod $f = 0$, then equation (10) becomes

$$E_{2m+1,\ f} = k + \frac{w-1}{f} B_{2m} \tag{13}$$

Note that with $f=1,\ e=0$ and $g=0$ equations (12) and (13) reduce to equation (5) as is necessary.

For 5-ply trees with width $w=35$ we know that the minimal number of leaf 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 leaf nodes is between 50,000 and 100,000 nodes [1]. Thus by using equations (12) and (13), 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 (13)

$$E_5 = w^3 + w^2(1+g)^2 + w((1+g)^4 - (1+g)^2) - (1+g)^4 \tag{14}$$

It is clear from equation (14) that one positive and one negative value for g exists, and that those values may be found iteratively or by plotting g against E_5 and w. Typical experimental values for w and E_d have been presented previously [1]. 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. Thus one expects a cut-off to occur quite soon. If it does not, then perhaps a new principal variation is in the offing, and a re-search may begin before the narrow window search completes.

7.2. Overheads

It is also possible to develop simple formulae for search and communication overheads. Search overhead has already been defined as the extra work done by f processors over that done by a single processor. Thus

$$search\ overhead = \frac{Cost\ of\ leaf\ nodes\ evaluated\ by\ f\ processors}{Cost\ of\ leaf\ nodes\ evaluated\ by\ one\ processor} - 1 \tag{15}$$

Thus for an alpha beta tree,

$$search\ overhead = \frac{E_{d,\,f}}{E_{d,\,1}} - 1 \tag{16}$$

where $E_{d,\,f}$ is given by equation (10) or (13). For our results in Tables 2, 3 and 4, however, we use a search overhead estimate based on the leaf node count. For a minimal tree this would be

$$Node-search\ overhead = \frac{N_{d,\,f}}{N_{d,\,1}} - 1$$

On the other hand, the time overhead can be measured quite accurately as

$$Time\ overhead = \frac{f * Time\ taken\ by\ f\ processors}{Time\ taken\ by\ one\ processor} - 1 \tag{17}$$

which in turn becomes

$$Time\ overhead = \frac{f\ *\ T_{d,\,f}}{T_{d,\,1}} - 1 \tag{18}$$

where $T_{d,\,f}$ is given by (10).

Communication overhead is algorithm and system dependent and is perhaps best expressed as

$$Communication\ overhead = Time\ overhead - Search\ overhead.$$

Thus from (16) and since $E_{d,\,f} = T_{d,\,f} - T_{d-1,\,f}$

$$Communication\ overhead = \frac{f\ *\ T_{d,\,f}}{T_{d,\,1}} - \frac{T_{d,\,f} - T_{d-1,\,f}}{T_{d,\,1} - T_{d-1,\,1}} \tag{19}$$

But, when progressive deepening is used, we can estimate $T_{d,\,f}$ by

$$T_{d,\,f} = p_f\ *\ T_{d-1,\,f}$$

For the uniprocessor case, computer chess experimenters estimate p_1 to be 5.5. A study of Figure 7 supports this estimate and, based of the ratio between the 6-ply and 5-ply searches, shows that p_f ranges from 5.3 to 6.8 as f varies from one to four processors. Therefore equation (19) is approximated by

$$Communication\ overhead = \left\lceil f - \frac{(p_f - 1)\ *\ p_1}{(p_1 - 1)\ *\ p_f} \right\rceil * \frac{T_{d,\,f}}{T_{d,\,1}} \tag{20}$$

which in turn is approximated by

$$Communication\ overhead = (f-1)\ *\ \frac{T_{d,\,f}}{T_{d,\,1}} \tag{21}$$

It follows therefore that our assumptions are equivalent to saying that the cost of a tree search is dominated by the cost of evaluating the leaf nodes.

8. Performance

The test results, as based on *Parabelle's* performance on a standard sequence of 24 chess positions [16], indicate that PV splitting has low search overhead. For comparison we present the data from a series of five ply searches, with and without transposition tables, performed by a system consisting of from one to four processors. This search depth limit was chosen because longer trees may cause overloading of the transposition table, thus obscuring the results.

By contemporary standards, the program for our application is weak at computer-chess. Our performance results are not designed to show how well the program plays chess, but rather to assess relative performance of the various components of the system. 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 extensibility of our implementation to speed. Our aim is to explore an 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.

8.1. Without Transposition Table

By disabling the transposition table, it is easier to observe characteristics of the parallel searching algorithm that may be obscured when the table is in use. Table 2 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 *speedup* being the average of the ratio of the time required by the uni-processor program to that of the multiprocessor system. All the averages, which appear on the bottom row of the table, are calculated by taking the mean of the values for the 24 test positions.

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 a few exceptions this is true for all the test positions. To understand why one or two tests do not display this characteristic, one must first reconsider the behavior of the parallel algorithm as compared to its sequential counterpart.

If the first move is not best, new candidates will arise and these will have to be re-searched with the correct window as they are recognized. When this is being done in parallel, it is possible that more of these wide 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

	No Transposition Table				
fanout f	leaf nodes	time secs	average speedup	% search overhead	% time overhead
1	60065	1611	1.0	0	0
2	63268	942	1.84	5.3	16.9
3	64503	664	2.53	7.4	26.6
4	64470	532	3.06	7.3	32.1

Table 2. Basic performance results without transposition tables.

scores for the list of moves, rather than the approximations returned by a uni-processor using a minimal window search. This can subsequently alter the search since a different move ordering will result after the move list is sorted between iterations. This reordering is the reason for the change in the move selected by systems of different processor sizes and for the occasional decrease in leaf nodes searched by the larger systems [17]. For example, Table 2 shows an apparent anomaly in the reduction of leaf nodes searched when four processors are used. This was due to the biasing effect by a single test position on the average number of nodes searched. Due to the size of that search, any trends present had a large effect on the average, as illustrated by 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 search 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, the child that is searching this variation may be the only processor working while the others are waiting for it to finish.

In our system, communication overhead cannot be measured directly because the message lengths are too short (only a few bytes) and the clock timer interval too long (1/60 secs). However, it can be estimated as the difference between the time overhead, equation (18), and the search overhead, (16). Much of the idle time results from the transmission of refutations and move lists after an iteration is completed. This **inter-iteration** communication occurs at 9600 baud for systems of up to three processors, but at half that speed with the four processor configuration. However, the largest part of the communication overhead is attibutable to the short **at-node** messages, which arise when the (optional) global transposition table is used.

8.2. Global or Local Transposition Table

The system performance with either a global or a local transposition table was tested using both a complete and a partial storage system. 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 a leaf. In the complete system, on the other hand, a table lookup is made upon the initial visit to any tree node. The global transposition table can hold 8192 entries for each side, while local tables are half that size.

When the complete storage of tree nodes is employed, there is a dramatic decrease in the number of leaf nodes visited. The results in Table 3 also show that with the global table fewer leaf nodes are visited than when local tables are used. This is to be expected since a global table is updated by all processors, and so will provide more cutoffs. However, the global table's effect on the leaf node count is at the expense of an increase in communication overhead. In fact the communication delays destroy the program's performance. With a four processor configuration the speedup is actually less than that for three processors, and this is attributed to the increased message traffic and reduced communication rate. 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 leaf nodes than its global counterpart. It is not at all clear from our present set of experimental results just what the effect of using a very high performance communication line would be. The components of the communication delay are access time and transfer time. Reducing the latter may have only limited effect. Our new system, in which the processors are interconnected via an Ethernet, will allow us to address this issue and also to study to properties of our analytical model where formulae for search overhead and communication overhead were presented.

In a parallel search system using a transposition table, there is no guarantee that the same cutoffs will always occur, especially if the number of processors changes. For example, with a global table, the order of storage and retrieval will vary for systems of different sizes, and this can even result in different moves being selected when several moves have the same best score [17]. 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 processors search which subtrees. Consequently, the leaf 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 can be partially alleviated by only storing the values of subtrees whose length is greater than two 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 approach. Nevertheless, the results in Table 3 show that use of a local transposition table produces a better speedup than the glo-

fanout f	(a) Global Transposition Table									
	all subtrees stored					only subtrees of length > 2 ply				
	leaf nodes	time secs	average speedup	%search o/head	% time o/head	leaf nodes	time secs	average speedup	%search o/head	% time o/head
1	53349	1307	1.0	0	0	52705	1275	1.0	0	0
2	53674	949	1.31	0.6	45.2	53274	684	1.84	1.1	7.3
3	53953	766	1.59	1.1	75.8	54178	498	2.5	2.8	17.2
4	56202	872	1.41	5.3	166.0	54558	420	2.91	3.5	31.8

fanout f	(b) Local Transposition Table									
	all subtrees stored					only subtrees of length > 2 ply				
	leaf nodes	time secs	average speedup	%search o/head	% time o/head	leaf nodes	time secs	average speedup	%search o/head	% time o/head
1	53349	1307	1.0	0	0	52705	1275	1.0	0	0
2	53993	684	1.88	1.2	4.6	53427	670	1.89	1.4	5.1
3	55345	502	2.55	3.7	15.2	54344	485	2.59	3.1	14.1
4	57930	457	2.91	8.6	39.8	55224	402	3.1	4.8	26.1

Table 3. Comparison of partial and full storage of subtree results

bal table case, even though the former configuration examines more leaf nodes. In both instances, the performance, based on leaf node count and search time, is better when using the partial rather than the complete storage system. In the latter case, more store requests arise and these must replace older entries in the table, making a later retrieval of them impossible. 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 and the results appear in Table 4. 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 improved 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. Thus the multiprocessor system made better use of the transposition table, leading to some good cutoffs that were not available to the uni-processor program. These speedups compare quite favorably to the 2.34 achieved with treesplitting on Arachne [18] using three processors, and the 2.4 obtained on an early version of a five processor system, OSTRICH/P [19], in an application similar to ours. The curves of Figure 7 show the reduction in search times as the number of processors increase. The results from the two to six ply

Terminal node count and CPU time (6-ply) Local transposition table for subtrees > 2-ply												
	one processor			two processors			three processors			four processors		
board	leaf nodes	time mins	move	leaf nodes	time mins	average speedup	leaf nodes	time mins	average speedup	leaf nodes	time mins	average speedup
A	(forced	mate)	d6d1									
B	132751	45	e4e5	138063	23	1.92	140299	16	2.80	141677	12	3.58
C	141201	74	e7d8	146760	39	1.89	149580	28	2.65	153523	22	3.33
D	131304	42	e5e6	136711	23	1.84	143206	17	2.45	144396	13	3.05
E	351867	227	f1f5	362944	125	1.82	364990	88	2.57	365387	70	3.21
F	42942	8	g5g6	47869	5	1.69	52372	3	2.24	56051	3	2.64
G	228816	148	h5f6	239839	78	1.89	340005	74	1.98	404481	75	1.95
H	14780	2	e2c3	15671	1	1.77	16142	1	2.30	16278	0	2.56
I	174398	111	f3e5	178305	58	1.91	178586	38	2.85	178555	30	3.66
J	436938	202	e8e6	545042	130	1.56	510633	82	2.44	509370	67	2.99
K	249238	126	g3f5	256469	66	1.89	270668	51	2.45	277254	41	3.06
L	165232	62	d7f5	170502	33	1.89	173795	23	2.70	176636	18	3.41
M	189128	72	a1c1	192818	37	1.93	200136	25	2.82	201468	20	3.63
N	83343	39	d1d2	86935	21	1.83	83653	14	2.70	78559	10	3.69
O	61098	30	g4g7	64520	16	1.82	66482	12	2.44	65447	9	3.13
P	212469	100	d2e4	**187435**	44	**2.24**	219667	38	2.62	224228	34	2.95
Q	191081	141	d7c5	199063	75	1.87	202848	53	2.64	209140	46	3.08
R	347935	191	c8g4	371298	105	1.82	379596	71	2.67	383867	56	3.41
S	200645	81	c7c5	205487	43	1.87	207509	31	2.58	208764	25	3.18
T	677076	315	c3b5	**393672**	95	**3.32**	**386704**	66	**4.76**	**388266**	52	**6.05**
U	367754	194	f5h6	391581	103	1.88	397914	69	2.80	400983	54	3.59
V	132017	86	d7e5	**130688**	43	1.98	**130886**	39	2.19	**131032**	37	2.29
W	531720	204	e8g8	541273	126	1.62	540947	75	2.70	**460973**	63	3.20
X	176158	78	b4c5	176184	39	1.98	178088	27	2.85	179959	21	3.59
total	5239891	2590		5179129	1339		5334706	953		5356294	790	
mean	227821	112		225179	58	1.92	231943	41	2.66	232882	34	3.27

Table 4. 6-ply search with local, depth-limited, transposition table.

searches can be compared to the curve for the optimal speedup (that is, N processors use 1/N the time of one processor). 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 game tree searches [4][5], although their simulation results were for comparatively narrow trees and could not adequately account for communication losses.

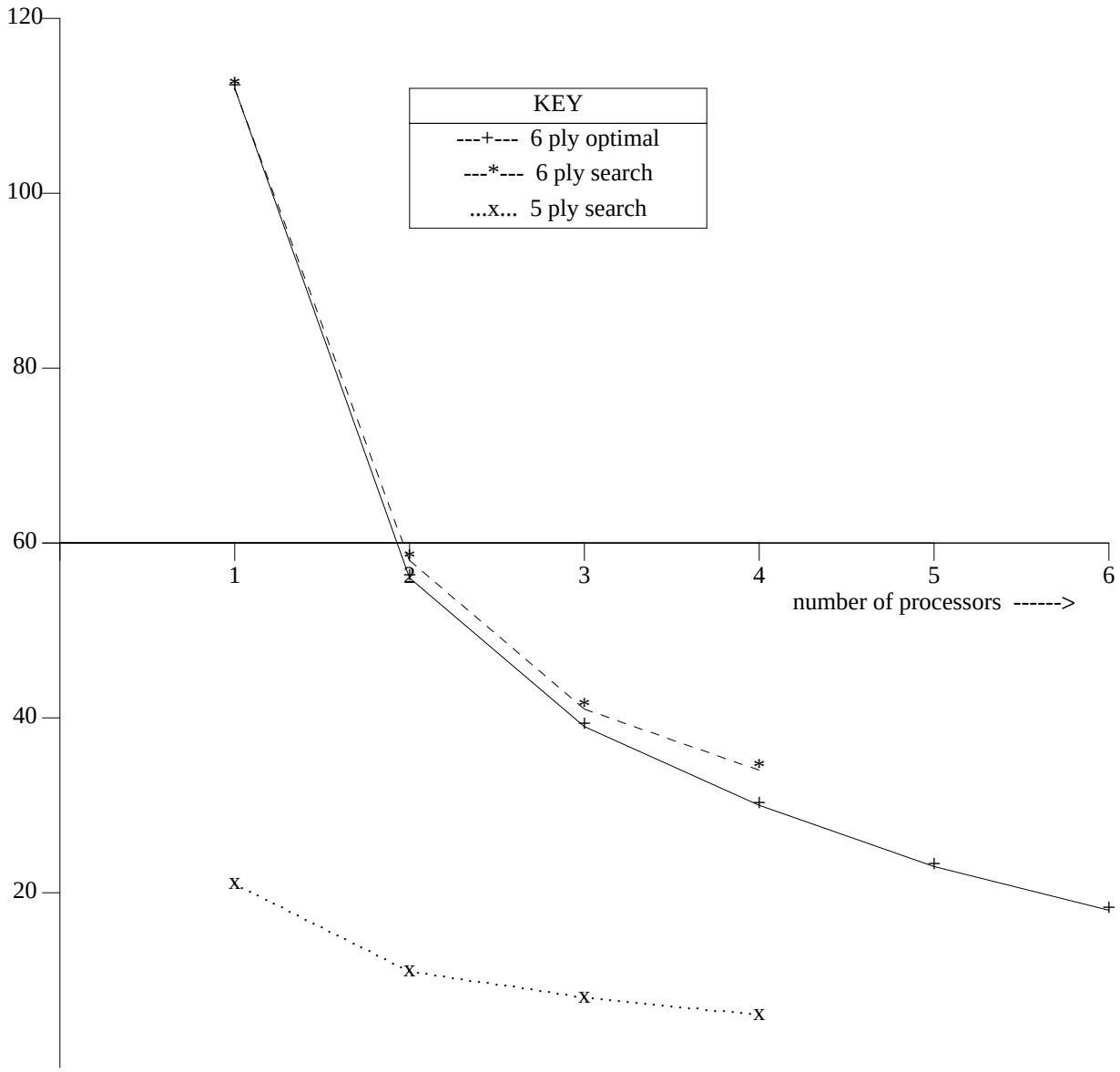

Figure 7. Search Time vs Processors

9. Summary

We have presented the outline of a multi-processor based system for use in minimax game tree searches. The use of memory tables has been examined along with the problems involved in their local and global implementations. The system development has also addressed the issues of processor management and inter-unit communication, which can be related to other parallel sys-

tems. Although we have designed our parallel processing system for a specific application, it is expressed in general terms so that the ideas employed may be suitable for any minimax tree search application.

Our experiments with the *Parabelle* system indicate that the PVS method is promising for use in multiple processor tree searching systems, and the inclusion of local, depth limited transposition tables appears to be an effective enhancement to the basic system. Although the communication problems associated with the global transposition table could be alleviated with faster communication speeds, the savings from visiting slightly fewer leaf nodes are not likely to compensate for the wait time due to communication delays. Also, if the new alpha bound were made available to all processors as soon as it was determined, rather than when a child processor has finished its subtree search, more cutoffs could be obtained. 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 deferring new principal variation searches until there is more than one possible new candidate [20]. 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

[4] 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.

[2] G. Baudet, "The Design and Analysis of Algorithms for Asynchronous Multiprocessors", Ph.D. thesis, Computer Science Dept., Carnegie-mellon Univ. Pittsburgh, April 1978.

[13] B.A. Bowen and R.J.A. Buhr, *The Logical Design of Multiple Microprocessor Systems*, Prentice-Hall, 1980, 65-71.

[16] I. Bratko and D. Kopec, "A Test for Comparison of Human and Computer Performance in Chess". In M.R.B. Clarke, editor, *Advances in Computer Chess 3*, Pergamon Press, 1982, 31-56.

[9] M. Campbell and T.A. Marsland, "A Comparison of Minimax Tree Search Algorithms", *Artificial Intelligence* **20** (1983) 347-367.

[15] Central Data Corporation, "Intelligent Octal Serial Interface", 1981.

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

[6] J. Fishburn, "Analysis of Speedup in Distributed Algorithms", Ph.D. thesis, TR 421, Computer Science Dept., Univ. Wisconsin, Madison, May 1981.

[3] J. Fishburn and R. Finkel, "Parallel Alpha-Beta Search on Arachne", TR 394, Computer Science Dept., Univ. Wisconsin, Madison, July 1980.

[10] J. Gillogly, "Performance Analysis of the Technology Chess Program", Ph.D. dissertation, Computer Science Dept., Carnegie-mellon Univ., Pittsburgh, March 1978.

[7] B.W. Kernighan and D.M. Ritchie, *The C Programming Language*, Prentice Hall, 1978.

[5] 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.

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

[1] T.A. Marsland, "Relative Efficiency of Alpha-Beta Implementations", *IJCAI Conference Proceedings*, Karlsruhe, August 1983, 763-766.

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

[8] J. Pearl, "Asymptotic Properties of Minimax Trees and Game Searching Procedures", *Artificial Intelligence* **14** (1980), 113-138.

[17] F. Popowich and T.A. Marsland, "Parabelle: Experience with a Parallel Chess Program", TR 83-7, Computing Science Dept., Univ. of Alberta, Edmonton, August 1983.

[12] D. Slate and L. Atkin, "CHESS, 4.5 — The Northwestern University Chess Program". In P. Frey, editor, *Chess Skill in Man and Machine*, Springer Verlag, New York, 1977, 82-118.

[14] SUN Microsystems Inc., "SUN-1 System Reference Manual", July 1982.

[20] K. Thompson, Private Communication, Oct. 1981.