PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT. © ICOT, 1992

Evaluation of the EM-4 Highly Parallel Computer using a Game Tree Searching Problem

Yuetsu KODAMA Shuichi SAKAI Yoshinori YAMAGUCHI

Electrotechnical Laboratory
1-1-4, Umezono, Tsukuba-shi, Ibaraki 305, Japan
kodama@etl.go.jp

Abstract

EM-4 is a highly parallel computer whose eventual target implementation has more than 1,000 processing elements(PEs). The EM-4 prototype consists of 80 PEs and has been fully operational at the Electrotechnical Laboratory since April 1990. EM-4 was designed to execute in parallel not only static or regular problems, but also dynamic and irregular problems. This paper presents an evaluation of the EM-4 prototype for dynamic and irregular problems. For this evaluation, we chose a checkers program as an example of the game tree searching problem. The game tree is dynamically expanded and its structure is irregular because the number and the depth of subtrees of each node depend heavily upon the status of the game. We examine effects of the load balancing by function distribution, data transfer, control of parallelism, and searching algorithms on the EM-4 prototype. The results show that the EM-4 is effective in dynamic load balancing, fine grain packet communication and high performance of instruction execution.

1 Introduction

Parallel computing has been effective for static or regular problems such as scientific computing and database systems. Parallel computing is, however, still an active research topic for dynamic or irregular problems.

EM-4 is a highly parallel computer which was developed at the Electrotechnical Laboratory in Japan. Its target applications include not only static or regular problems, but also dynamic or irregular problems. EM-4 provides special hardware for parallel computing: high data transfer rate, high data matching performance, dynamic load balancing, and high instruction execution performance.

In this paper, we evaluate the performance of EM-4 on a dynamic and irregular problem. The performance of EM-4 on some small programs such as recursive fibonacci is presented in [Kodama *et al.* 1991]. While the fibonacci program creates many function instances dynamically, it is not irregular because the tree of calling functions is a binary tree, the depth of each branch is similar to those of its neighbors, and the size of each node function is the same and small. We chose a game tree searching problem as a practical problem. This class of programs dynamically expands the game tree, and is irregular because the number of subtrees from each node of the game tree, the depth of subtrees, and the execution time of each node depends heavily upon the status of the game. Furthermore, the α-β searching algorithm is often used for game tree searching, because it cuts the evaluation of the current tree by using the evaluation of the previous tree. Tree cutting makes the program more dynamic and irregular.

This paper presents the evaluation of the EM-4 prototype using a checkers game program as an example of the game tree searching problem. We examine the effect of parallel computing on the EM-4 prototype. Section 2 presents an overview of the EM-4 and its prototype. Section 3 describes a game tree searching problem and a checkers game. Section 4 presents evaluation issues for load balancing, data transfer, control of parallelism, and searching algorithms for the checkers game. Section 5 gives an evaluation and examination of the strategies described in section 4. Section 6 concludes our results and discusses our future plans.

2 The EM-4 Highly Parallel Computer

EM-4 is a highly parallel computer whose eventual target implementation has more than 1,000 PEs[Yamaguchi *et al.* 1989, Sakai *et al.* 1989]. The EM-4 prototype consists of 80 PEs and has been fully operational since April 1990[Kodama *et al.* 1990].

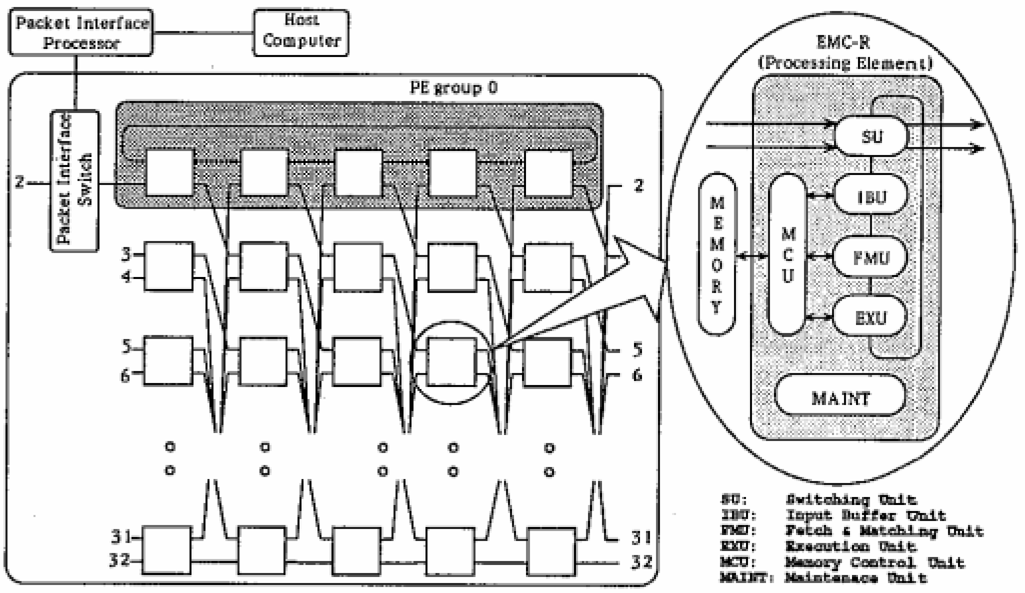


Figure 1: The organization of EM-4 Prototype

2.1 The architecture of EM-4

The organization of the EM-4 prototype is shown in Figure 1. The prototype consists of 80 PEs, and each 5 PEs are grouped and are implemented on a single PE board. The PE of the prototype is an single chip processor which is called EMC-R and is implemented in a C-MOS gate array. The PE has local memory and is connected to the other PEs through a circular omega network.

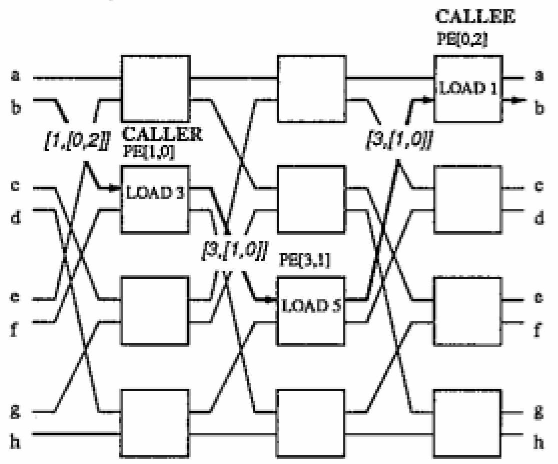
EMC-R is a RISC processor for fine grain packet-based parallel processing. EMC-R generates packets in an execution pipeline, and computation is fired by the arrival of packets. This is a dataflow mechanism, but we improved it so that it can operate on a block which consists of several instructions, executed exclusively from other instructions. This model is called the "strongly connected arc model", and the block is a strongly connected block(SCB).

When a packet arrives at a PE, the execution pipeline is fired and EMC-R executes the SCB indicated by the packet. First, EMC-R checks whether the partner of the packet has arrived. If the parter exists, it continues to execute the SCB until the end of the block. If the parter does not exist, EMC-R stores the packet data in a matching memory and waits for the next packet.

The packet size of EMC-R is two fixed words and there is only one format consisting of one address word and one data word. It can be generated in a RISC pipeline of EMC-R. During the data word is calculated in a RISC pipeline, the address word is formed in a packet generation unit when the packet output is instructed. Since the network port is only one word wide, first the address word is sent to network, and then the data word is sent. In the second clock cycle, the next instruction can be executed in parallel with data word transfer.

The circular omega network has the same structure as an omega network, except that every node of the network is connected to a PE. The network has the following features: (1) The required amount of hardware is $O(N)$, where N is the number of PEs; (2) The distance between any two PEs is $O(logN)$. The 3 by 3 packet switching unit is in a EMC-R, and a packet can be transferred to a neighboring PE independent of the instruction execution on the PE. Packets are transferred by wormhole routing, and take only $M+1$ cycles between PEs which are distance M apart if there is no network conflict.

The clock of the EMC-R runs at 12.5 MHz. The RISC pipeline can execute most instruction in one clock cycle; the peak execution performance is 12.5 MIPS. It takes two clock cycles when two operand matching fails, and takes three clock cycles when the matching succeeds. The peak synchronization performance is 2.5 Msync/s. It takes two clock cycles to transfer a packet, and the peak network packet transfer performance is 18.75 Mpacket/s. EM-4 prototype consists of 80 PEs, the peak execution performance is 1 GIPS, its peak synchronization performance is 200 Msync/s, and its peak network packet transfer performance is 1.5 Gpacket/s. EMC-R achieves a high performance in both instruction execution and packet data transfer/matching.



[LD,[GA,CA]] is the MLPE packet which shows that PE[GA,CA] has the minimum load LD

Figure 2: How to Detect the Minimum Load PE

2.2 Dynamic load balancing method

To get high performance in parallel computers, high utilization of PEs, as well as high performance of PEs are necessary. If the program has simple loop structure or static data transfer structure such as in diffusion equation applicaitons, the load of the program can be estimated and the load can be statically balanced at programming or compiling time. But, if the program is dynamic or irregular structure, static load balancing is difficult and dynamic load balancing is necessary.

In the EM-4, we implemented automatic load balancing mechanisms attached to the circular omega topology. In the circular omega network, each node has two circular paths. We use a path to group the PEs, and use another path to achieve dynamic load balancing. Suppose that a PE wants to invoke a new function. This PE will send out a special MLPE(Minimum Load PE) packet. The MLPE packet always holds the minimum load value and the PE address among the PEs which it goes through. The load of each PE is evaluated by hardware in the PE mainly based on the number of packets in the input buffer. At the starting point, the MLPE packet holds its sender's load value and its PE address; when it goes through a certain SU in the circular path, the SU compares the load value of the PE connected to it, and if the value is less than of the packet, the data in the MLPE packet will be automatically rewritten to the current PE's value; otherwise the MLPE packet keeps its value and goes to the next SU. This operation is done in one clock cycles of packet transfer. When the MLPE packet returns to the starting point, it holds the least loaded PE number and its load value. Figure 2 show this. In this figure, PE[1,0] generates an MLPE packet and, after the circulation, it obtains the least loaded PE number [0,2] and its load 1.

By this method, called the circular path load balancing, each MLPE packet scans s different groups, where s is the number of network stages. When the total number of the PEs increase, coverage of PEs by this load balancing method becomes relatively small. The efficacy of this method is reported in [Kodama *et al.* 1991].

Since it takes several cycles for the MLPE packet to return, the EM-4 resolves this latency by pre-fetching: it sends a MLPE packet in advance, allocates the new function instance on the PE specified by the returned packet of MLPE, and stores the function ID in a special register of the required PE. When a function call is necessary, the stored function ID is used and another MLPE packet is sent for the next function call. In the pre-fetch strategy, the new function ID may have not yet been stored when a function call is necessary. In this case, the pre-fetch method uses one of the other distribution methods to choose the PE.

3 Game Tree Searching Problem

We choose the checkers program as an example of a game tree searching problem in order to evaluate the EM-4 on a dynamic and irregular problem. Since the rules of checkers are very simple, the program makes it easy to characterize the parallel behavior of the program.

The rule of checkers game is as follows. Each player moves one of his pieces in turn until the player who has no pieces or moves loses. Pieces can be moved to a forward diagonal area. If there is an opponent's piece in a forward diagonal area, and the next diagonal area is empty, you must jump to the empty area and remove the enemy piece. If you can jump successively, you must jump successively. If your piece arrives at the end of the enemy area, that piece can then move in all four diagonal directions.

The Min-Max searching algorithm is the simplest algorithm for the game tree searching problem. This algorithm expands the game tree by the possible moves of each player in turn. When the game tree is expanded to a certain level, each leaf is evaluated. If the stage corresponds to your turn, the maximum node is selected; if the stage is your opponent's turn, the minimum node is selected. Although the Min-Max algorithm is simple, it is not efficient because it needs to search every branch. The α-β searching algorithm[Slagle 1971] is more efficient than the Min-

Max algorithm, because this algorithm tries to cut off the evaluation of unnecessary branches.

If the game tree is expanded in a depth-first manner, the resources required to remember the game tree are small. This expansion makes it easy to cut off the unnecessary branches, but reduces the parallelism. If the game tree is expanded in a breadth-first manner, it results in large parallelism, so this expansion is well-suited for parallel computers. However, since the number of nodes increases exponentially as a function of the depth of the tree, the resources will be exhausted quickly if the parallelism is not controlled.

4 Execution Issues of a Checkers Game

The overheads to parallelize the checkers program are the following:

1. overhead for allocating new function instances on other PEs.
2. overhead for transferring the current status of the table to other PEs.
3. idle PEs caused by an unbalanced load.
4. decline of efficiency caused by cutting branches in the α-β search.

These overheads depend upon implementation strategy decisions. The function distribution strategy effects the function allocation overhead. Packed data transfer reduces the amount of transfer data. The idle PE ratio depends upon the load balancing strategy. The searching algorithm changes the branch cutting overhead. These overheads also depend upon the control of the parallelism and the searching strategy. Each of these decisions is described in greater detail in the following subsections.

4.1 Function distribution and load balancing

Load balancing is the most important issue in achieving high performance on parallel computers. Since the checkers program requires many function instances to expand the game tree, it distributes them among the PEs in order to balance the load across the machine.

Our checkers program can distribute function calls by one of the following two strategies:

round-robin distribution Each PE independently chooses the PE which will execute the called function in a round-robin manner.

manager distribution A centralized manager PE chooses the PE which will execute the called function.

We can also combine the two methods: that is, the manager distribution can be used until a certain level in the game tree expansion, and the round-robin distribution can be used after that level. In the round-robin distribution, the load might be unbalanced at the beginning of the program. In the manager distribution, the overhead is larger than round-robin distribution because of packet communication overhead and concentration of requests.

EM-4 dynamically distributes functions according to the load of PEs by the circular path load balancing described in section 2.2. The dynamic round-robin distribution described below is the third function distribution method that we evaluated in our checkers program.

dynamic round-robin distribution A PE is dynamically chosen by the circular path load balancing method, and in the case that the MLPE packet has not returned, a PE is chosen by the round-robin distribution method.

4.2 Data transfer

Since EM-4 is a distributed-memory parallel computer, the checkers program sends the status of the table and selected moves by packets to functions on other PEs. The status of the table is represented by a 64 word array, but each word is only 4 bits. The following two transfer methods are considered in the checkers program.

unpacked transfer use packets which have data representing a position.

packed transfer use packets which have packed data representing 8 positions.

While the unpacked transfer sends eight times more packets than the packed transfer, the packed transfer needs to pack and unpack data.

4.3 Control of parallelism

Parallelism has to be controlled to both avoid exhaustion of resources, and to provide sufficient parallelism to keep all the PEs busy. To control parallelism, throttling can limit the number of the active functions. If the number of active functions exceeds a certain amount, further requests for calling functions are buffered until other functions are finished. Throttling has the possibility of deadlock.

Another way to control parallelism is to switch from breadth-first search to depth-first search at some level of the game tree, where the level can be determined either statically or dynamically. Static switching sets the level by the depth of the game tree. Dynamic switching determines the level using the load of PEs. Breadth-first searching increases parallelism, and depth-first searching restrains parallelism.

Our checkers program uses the static switching strategy to control parallelism, because this strategy is very simple. We plan to implement the dynamic switching strategy for the checkers program in the near future.

4.4 Game tree searching algorithms

The two primary algorithms for the game tree searching problems are the Min-Max algorithm and the α-β algorithm. The Min-Max algorithm provides much parallelism in the breadth-first strategy. The α-β algorithm has high efficiency in the depth-first strategy. If the α-β algorithm is used only with the breadth-first strategy, it ignores the possibility of cutting branches, and it must search more trees than the α-β algorithm on a single processor. Since the ratio of branches cut off relative to the whole tree in the α-β algorithm increases according to the depth of the searching tree, a parallel α-β searching algorithm must be considered to increase the efficiency of branch cutting in the parallel environment.

Parallel α-β searching is complicated because of the dilemma between parallelism and efficiency of branch cutting. Another important problem is the overhead of terminating functions. Since these function instances are distributed and activated in parallel, the overhead of terminating functions is more than overhead of creating functions. This difficult trade-off is simply resolved in our checkers program by changing algorithm in breadth-first strategy and depth-first strategy. In the breadth-first strategy, we select the min-max algorithm to expand the parallelism, and in the depth-first strategy, we select the α-β algorithm to achieve the efficiency of cutting branch. We call this search "serial α-β search" in this paper. This search can be easily implemented, but the efficiency of branch cutting is less than the parallel α-β search[Oki *et al.* 1989].

To get more efficiency from branch cutting, the search that uses α-β search from the leaf of breadth-first strategy is the "partial parallel α-β search". This search algorithm is illustrated in Figure 3. In this search, depth-first search is called in parallel from the leaf of breadth-first search, but the top node(which is indicated by B in the figure) of serial depth-first search gets the α-β value from the parent node (A) every time when the child node (C) return the evaluation result, and check whether the remain branch (C') can be cut off or not. The merit of this search is that we can expect enough efficiency from branch cutting and the overhead of terminating search is nothing

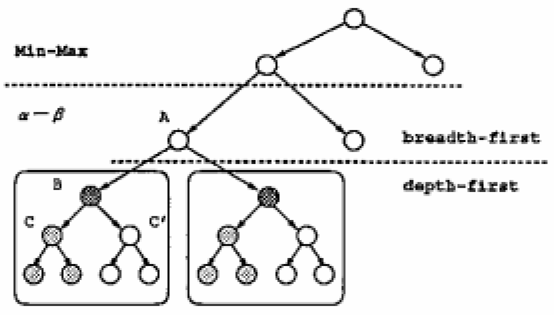


Figure 3: partial parallel search

since the child node in depth-first strategy is sequentialized.

The checkers program can use the following three searching algorithms.

Min-Max search using the Min-Max algorithm both breadth-first and depth-first.

serial α-β search using the Min-Max algorithm breadth-first, and using the α-β algorithm depth-first.

partial parallel α-β search using the Min-Max algorithm breadth-first until the last level, and using the α-β algorithm in the last level of breadth-first and then depth-first.

5 Experimental Results on the EM-4

We implemented the checkers program on the EM-4 prototype in an assembly language to evaluate the performance of the EM-4 for dynamic and irregular problems. We examine the execution issues discussed in the previous section.

5.1 Effects of function distribution and load balancing

An unbalanced workload causes idle PEs. Since the load balancing of the checkers program is performed at the function level, the function distribution strategy must be evaluated. The alternatives for the function distribution of the checkers program are the manager distribution, the round-robin distribution, the dynamic round-robin distribution, and combinations of these.

We executed the checkers program using the partial parallel α-β search using each function distribution methods. Figure 4 shows the results. We represent the speedup ratio of each distribution relative to the round-robin distribution. We executed each combination of manager distribution and round-robin distribu-

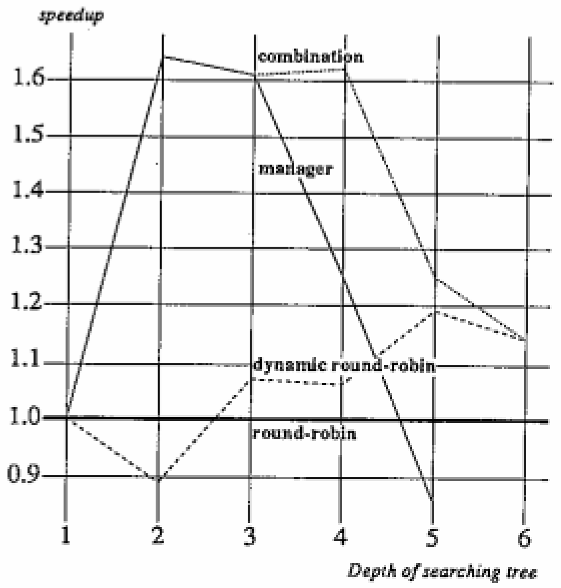


Figure 4: Effects of function distribution

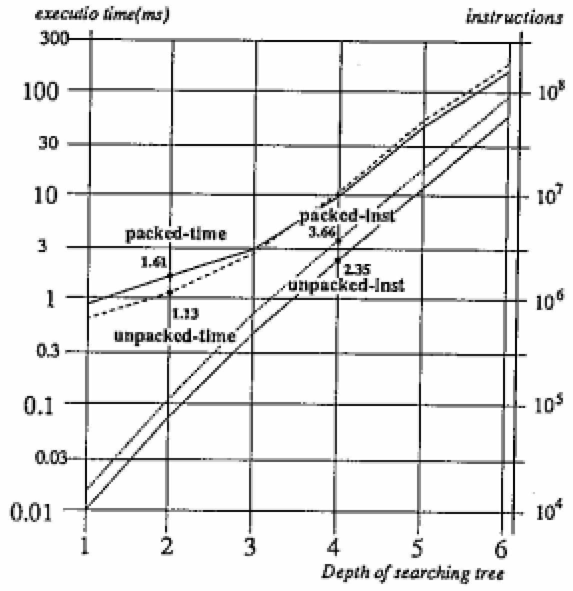


Figure 5: Comparison of data transfer

tion, and the fastest combination is shown in the figure. The combination uses the manager distribution until the third level, and thereafter uses the dynamic round-robin.

When the level of tree searching is shallow, manager distribution is better, because the manager distribution allocates functions more evenly. Since the size of each function is large relative to the whole program, the heavily loaded PE will become a bottle-neck and the program cannot achieve sufficient speed-up, even if the load is only slightly unbalanced. When the level of the search tree becomes deeper, the dynamic round-robin distribution is better, because the size of each function becomes small relative to the whole program, and a small load imbalance does not effect the execution time much. On the other hand, in the manager distribution, the requests of PE addresses for the function call concentrate on the manager PE. Because of the queue of requests, the long turnaround time of the function call makes the execution time slow. Furthermore, at the sixth level of the search tree in the manager distribution, the program cannot be executed because of overflow of the packet queue buffer.

Since the execution of the dynamic round-robin distribution is 15% faster than the round-robin distribution when the searching tree is deep, this indicates that the dynamic round-robin strategy is effective in the case that there is sufficient parallelism.

5.2 Effects of data transfer

To parallelize the program, data must be transferred between PEs, while data is only passed between memory locations in a single PE. We compared the two data transfer method, unpacked and packed. The unpacked transfer uses a packet which has data representing a position, while the packed transfer uses a packet which has packed data representing 8 positions.

Figure 5 is the results by the checker program of the partial parallel α-β search using the combination of manager and dynamic round-robin method as the function distribution. This figure shows the execution time and the total number of executed instructions of both data transfer method. Note that the execution time and the total number of the executed instructions are figured on a logarithmic scale.

In this figure, the number of executed instructions of the packed packet transfer is 50% more than the unpacked transfer for each level. The increase of the executed instructions is caused by the pack and unpack operations. When the level is shallow, the execution of the unpacked transfer is 1.5 times faster than the packed transfer. This speed-up ratio is the same as the instruction amount ratio. But when the level is deep, packed transfer is a little faster than the unpacked transfer while the instruction count of the packed transfer is larger than the unpacked transfer.

Figure 6 shows the number of active PEs and overhead PEs in both data transfer strategies. An overhead PE is a PE which is waiting for the ready of the network to send a packet or stores the packet in the memory packet buffer when the on-chip packet buffer overflows. An active PE is a PE which is neither an overhead PE nor an idle PE. At the shallow levels, the active PE ratio of both transfer strategy is low. When

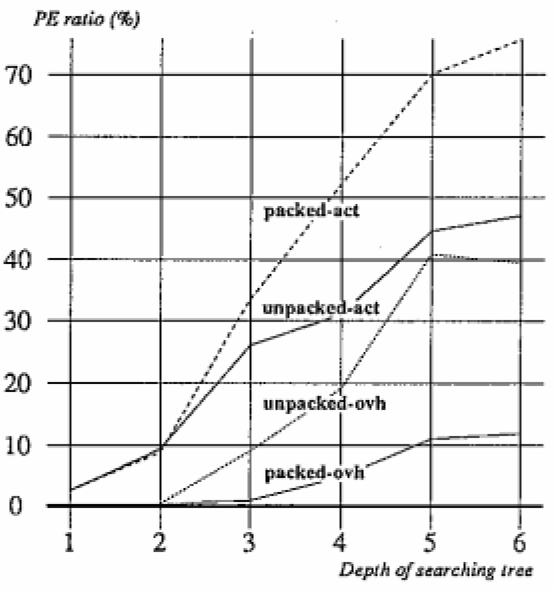


Figure 6: Examination of the active PE ratio comparing the data transfer

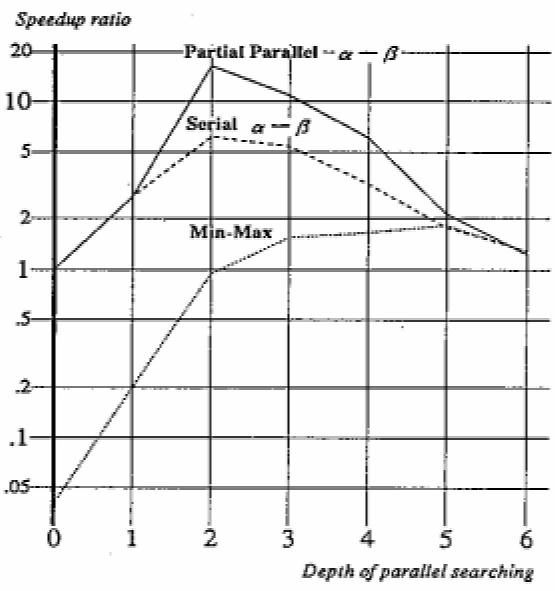


Figure 7: Effects of parallelism control

the level becomes deep, the active PE ratio of the unpacked transfer is 30% lower than the packed transfer, and the overhead PE ratio of the unpacked transfer is 30% higher than the packed transfer. This high overhead PE ratio of the unpacked transfer is the reason why it is slower than the packed transfer. Since the unpacked transfer needs to send more packets than the packed transfer, the network has many conflicts, resulting in large overhead.

Although the packed transfer shows the high ratio of the active PEs on the surface, a third of the instructions are used for packing and unpacking the packets, and the packed transfer is not so effective. Since the pipeline of the EM-4 is designed to send packets quickly, unpacked transfer is suitable for the EM-4. If there are many conflicts in the network, however, the overhead decreases the performance of sending packets. One way to reduce this overhead is to avoid the network conflicts by allocating the function locally. Since the manager and round-robin distributions does not take into account the locality between the PE which calls the function and the PE which executes the function, it increases the possibility of network conflicts. If the execution PE is selected from the neighbors of the calling PE, network conflicts do not occur as frequently. Another way to control the parallelism is by limiting the number of active functions. This is examined in detail in the next subsection.

5.3 Effects of parallelism control

While parallelism must be exploited to make the program execution faster, as mentioned before, too much parallelism causes some overhead. It is necessary to control the parallelism in order to avoid the exhaustion of resources, and to reduce the overhead of parallelization. The checkers program controls parallelism by switching the searching strategy from a breadth-first manner to a depth-first manner.

Figure 7 shows the speedup ratio to the sequential execution of the α-β search when the switchover level of the parallelism control strategy is changed. The execution uses the combination of manager and dynamic round-robin method as the function distribution strategy and the unpacked method as the data transfer strategy. Note that the X-axis represents the depth of the breadth-first searching, while these all execution search the game tree until the depth is the sixth level.

In the Min-Max search, the deeper level of parallel searching results in more parallelism, and the maximum speedup becomes 49 times. Exploiting maximum parallelism, however, does not necessarily achieve speedup. One reason is that at the sixth level, too many packets are sent and the overhead of network conflicts becomes much larger than at the shallow levels. Another reason is that excessive parallelism is just overhead such as data transfer or remote function invocation, since sufficient parallelism is exploited until the fifth level. It is sufficient to have as much parallelism as needed to activate every PE and hide the latency of remote access — excessive parallelism is not helpful.

The serial α-β search executes fastest at the second level, and when the level is deeper the performance decreases. This is because parallel searching uses breadth-first search, and much information that could be used to cut subtrees is discarded to parallelize the program. As parallel searching gets deeper, more information is discarded. As a result, it reduces the efficiency of cutting excessive branches, and increases the number of trees to be evaluated. The partial parallel α-β is same as the serial α-β search.

5.4 Effects of searching algorithms

Figure 7 also shows the effects of searching algorithms. The execution of the Min-Max search on 80 PEs is 49 times faster than the Min-Max search on a single PE, but only 1.8 times faster than the α-β search on one PE. This shows that the Min-Max search is suitable for parallel execution, but that it is difficult to compensate for the difference of efficiency between the Min-Max search and the α-β search by parallel execution.

The α-β search is a very serial algorithm, but can achieve 16 times speedup via partial parallel α-β search, while the serial α-β search can achieve 6 times speedup. This is because the partial parallel α-β search uses the information of cutting trees at the last level of parallel searching, and the efficiency of cutting trees in the partial parallel α-β search is higher than the serial α-β search.

6 Conclusion and Future plans

To evaluate the highly parallel computer EM-4 on dynamic and irregular programs, we execute the game tree searching problem of checkers on the EM-4 prototype, which consists of 80 PEs. The effects of the strategies for load balancing, data transfer, parallelism control and searching algorithm are examined.

Our checkers program achieves 49 times speedup in the Min-Max search and 16 times speedup in the α-β search on 80 PEs system. In this execution, the combination of the manager distribution until the third level and the dynamic round-robin distribution thereafter is used as the function distribution method for load balancing, the unpacked transfer is used as the data transfer strategy, and the static switching from the breadth-first to the depth-first at the fifth level in the Min-Max search and at the second level in the α-β search is used to control parallelism.

In this evaluation, we demonstrated that the EM-4 is effective for dynamic load balancing, fine grain packet communication and high performance of instruction execution.

In the near future, we plan to implement a dynamic switching strategy which controls parallelism according to the load of neighboring PEs. We will also implement the full parallel α-β search, compare it with partial parallel α-β search, and make clear the advantages and disadvantages of each method in the EM-4 for the parallel game tree searching.

Furthermore, we are designing a higher performance parallel computer EM-5. This computer will reduce the overheads which are found in these evaluations such as network conflicts.

Acknowledgments

We wish to thank Dr. Toshitsugu Yuba, Director of the Computer Science Division, Mr. Toshio Shimada, Chief of the Computer Architecture Section for supporting this research, and the staff of the Computer Architecture Section for their fruitful discussions. Special thanks are due to Dr. Mitsuhisa Sato of the Computer Architecture Section and Mr. Andrew Shaw of MIT for their suggestions and careful reading.

References

[Slagle 1971] James R. Slagle, Artificial Intelligence: The Heuristic Programming Approach, McGraw-Hill Inc., (1971).

[Oki *et al.* 1989] H. Oki, K. Taki, S. Sei and S. Huruichi, The parallel execution and evaluation of a go problem on the multi PSI, Proc. of the Joint Symp. on Parallel Processing '89, (1989), 351-357.(in Japanese)

[Yamaguchi *et al.* 1989] Y. Yamaguchi, S. Sakai, K. Hiraki, Y. Kodama and T. Yuba, An Architectural Design of a Highly Parallel Dataflow Machine, Proc. of IFIP 89, (1989), 1155-1160.

[Sakai *et al.* 1989] S. Sakai, Y. Yamaguchi, K. Hiraki, Y. Kodama and T. Yuba, An Architecture of a Dataflow Single Chip Processor, Proc. of ISCA 89, (1989), 46-53.

[Kodama *et al.* 1990] Y. Kodama, S. Sakai and Y. Yamaguchi, A Prototype of a Highly Parallel Dataflow Machine EM-4 and its Preliminary Evaluation, Proc. of InfoJapan 90, (1990), 291-298.

[Kodama *et al.* 1991] Y. Kodama, S. Sakai and Y. Yamaguchi, Load balancing by Function Distribution on the EM-4 Prototype, to appear in Supercomputing '91, (1991).