Efficiency of Parallel Computation on the Binary-Tree Machine CORAL '83

YOSHIZO TAKAHASHI*, YOSHITAKA YAMANE**, KAZUYOSHI NISHIYAMA***,
FUMINORI YOSHITANI* and KATUHIRO INOUE*

In order to study the effectiveness of binary-tree architecture to parallel computing for general application problems, a small-scale binary-tree machine CORAL '83 equipped with a software environment has been developed. The CORAL '83 consists of a host computer and a processor tree. The latter is composed of 15 processor elements each of which has an 8085 as a cpu, 8 kB ROM, 17 kB RAM, and one parallel port for each of three directions. The software environment includes a C compiler for writing parallel programs, an initial program loader, interprocessor and host-root communication routines, and several distributed operating systems. Using this facility, a number of parallel computing programs for various problems were written and were executed to measure the efficiency of parallel computation. From them a parallel SOR computation of Laplace equation, a tree sort, the N queens problem, a computation of prime numbers with the sieve of Eratosthenes, and the FFT have been selected. The exploitation of parallel algorithms adequate to the binary-tree machine and the measured efficiencies of parallel computation for these programs are presented. This study proves that the binary-tree machine has modest efficiencies for both structured and unstructured problems.

1. Introduction

Various architectures for highly-parallel processing systems have been proposed to date. Among them the tree architecture is considered effective for some special computing problems [1][2][3]. In order to study the effectiveness of the tree architecture for more general parallel computing problems, the authors have built a small-scale binary-tree machine, CORAL '83, consisting of 15 processor elements and have developed a software environment for program writing. Using these facilities a considerable number of parallel programs for various problems were written and were executed to measure the efficiency of parallel computing. From these results the effectiveness and the limitations of the binary-tree machine were confirmed.

This paper describes the hardware and software facilities of the developed CORAL '83, presents measured efficiency of parallel computations of some of the programs, and discusses the effectiveness of the binary-tree architecture to general parallel computing problems. In Chapter 2 of this paper the general properties of a binary-tree machine are reviewed, and in Chapter 3 and Chapter 4 the hardware and software facilities of CORAL '83 are described. In Chapter 5 parallel programs for different types of problems are described and their efficiencies, measured by executing them on CORAL '83, are discussed. In Chapter 6 the effectiveness of the binary-tree machine to parallel computations is explained, and some problems on CORAL '83 that may be improved are discussed.

*Department of Information Science, Faculty of Engineering University of Tokushima
**Shikoku Kakouki Co., Ltd.
***Fuchu Works, Toshiba Corporations.

2. Properties of Binary-Tree Machine

The requirements for highly parallel processors based on MIMD architecture are generally considered as follows.

1) Simple structure which is implementable in a VLSI chip.

2) Efficiency of parallel processing, that is, the speed-up ratio or the processor utilization factor, which is good for average parallel processing problems.

3) Small amount of interprocessor communication which is required for tree-structured problems.

The last requirement arises from the fact that most cpu-limited NP complete problems have mostly tree-like structures. The following discussions reveal that the binary-tree architectures best meet these requirements.

As indicated in Fig. 1(a) the binary-tree machine is a set of processor elements connected together as a binary tree. We name a processor element at the root, a node and a leaf of the tree as the root processor, a node processor and a leaf processor. A node processor has connections to other processor elements in three directions, which are Top, Left and Right as indicated in Fig. 1(b). For convenience we define the fourth direction In for the node processor itself. The number of the processor elements n and that of arcs a of a binary-tree machine with L levels are represented as

$$n=2^{L+1}-1 \tag{1}$$

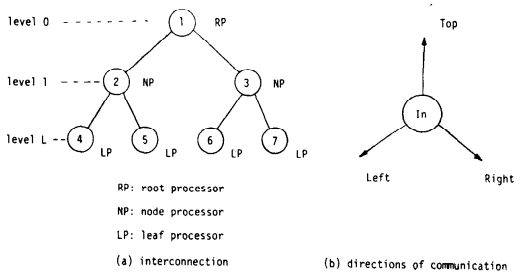

Fig. 1 Binary-Tree Machine.

$$a=n-1. \tag{2}$$

From these formula it can be shown that the number of connections per processor element is 2 in average and 3 at most. This is the reason why the structure of a processor element of the binary-tree machine is so simple.

The routing rule of the binary-tree machine is also very simple. We give a processor number to each processor element as indicated in Fig. 1(a). Let p and d be processor numbers of the routing and the destination processor elements. Then the direction of retransmitting the message is obtained by the following formula in McCarthy's conditional expression.

$$\begin{aligned}\text{direction}(d)=[&d=p &&\rightarrow \text{In},\\ &d<p &&\rightarrow \text{Top},\\ &d=2p &&\rightarrow \text{Left},\\ &d=2p+1 &&\rightarrow \text{Right},\\ &t &&\rightarrow \text{direction}(\text{d div } 2)\\ ]\end{aligned} \tag{3}$$

To attain efficient parallel processing the amount of communication between processor elements has to be kept as small as possible. We represent this amount by the processor distance which is the number of arcs that exist between two processors. In the binary-tree machine the average processor distance D is calculated as follows [2].

$$D=\frac{(L-2)2^{2L+2}+(L+4)2^{L+1}}{(2^{L+1}-1)(2^{L}-1)}\sim 2\log_2 n \tag{4}$$

The time to broadcast a datum from one processor element to all other ones is also important. We assign the processor element that makes this time minimum to the control processor which manages the distribution of program and data to, and the collection of the results from, other processor elements. The broadcast distance B is defined as the number of routings necessary to deliver a message from this processor to all other processor elements [2]. In the binary-tree machine the root processor is the control processor and the broadcast distance is calculated as

$$B=2L\sim 2\log_2 n. \tag{5}$$

These two values for the torus structure are given as

$$D=n/2 \tag{6}$$

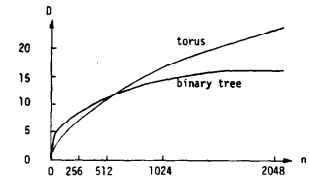

Fig. 2 Average Processor Distance of Binary-Tree and Torus Machines.

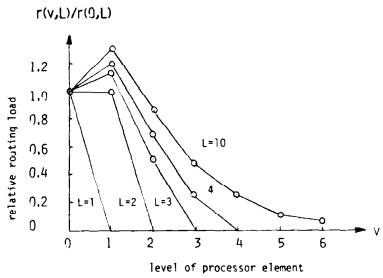

Fig. 3 Routing Load in a Binary-Tree Machine.

$$B=n/4. \tag{7}$$

Eq. (7) does not apply to some parallel processors such as Illiac IV and PAX [4] as they are provided with a broadcast-bus.

In Fig. 2 the processor distances of binary-tree and torus machines are compared. As shown in the figure, the average processor distance for the binary-tree machine is less than that for torus machines when there are more than 640 processor elements.

Two problems are known to exist in the binary-tree machine. The first one is the unbalanced routing loads among processor elements. In case a uniform communication takes place between two processor elements of all combinations, the routing load for a processor element is represented by the number of paths connecting two other processors and passing through it. Let the routing load of a processor element of the v-th level be denoted by $r(v.L)$. It is represented as

$$r(v, L)=4(2^{L}-1)(2^{L-v}-1)-3(2^{L-v}-1)^2. \tag{8}$$

Fig. 3 shows the ratio of $r(v, L)$ to $r(0, L)$ as a function of L. As is shown the routing load is largest in level 1 and decreases quickly as the level increases. However, in practical parallel computations communication between all processor elements may not be necessary. With a deliberate strategy of problem decomposition and processor allocation it is possible to balance the routing loads of all processors.

The second problem of the binary-tree machine is said to be the lack of reliability. As the root processor serves as the control processor, the failure of the root processor causes fatal system failure. When a node processor fails, all the processor elements connected underneath become unavailable. Let us assume that all

Fig. 4 CORAL '83.

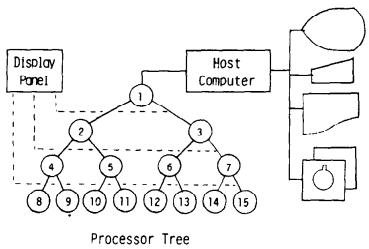

Fig. 5 Organization of CORAL '83.

processor elements have the same failure probability f. Then the ratio of the expected number of unavailable processors to the total processor elements, denoted by U, is represented as follows [5].

$$U=\sum_{v=0}^{L} f\cdot 2^{v}b(v)/n \tag{9}$$

where $b(v)$ is the number of processor elements belonging to the subtree under a level v processor element, which is

$$b(v)=2^{L-v+1}-1. \tag{10}$$

Therefore U is represented as

$$U=f[L+(L+1)/n]\sim f\log_2 n \tag{11}$$

We can conclude that the reliability of the binary-tree machine is not so bad as has been believed. This is reasonable because the processor elements that most affect the reliability are few while those that least affect it are many. Nevertheless the proposals to improve the reliability of the tree machine [6][7] are worth considering.

3. Structure of CORAL '83

The development of CORAL '83 is purposed to provide a testbed for verifying parallel algorithm, evaluating the performance of parallel computation, and developing and refining distributed operating systems. Fig. 4 shows the front view of CORAL '83. As shown in Fig. 5 it is constructed from a host computer and a processor tree connected together with a

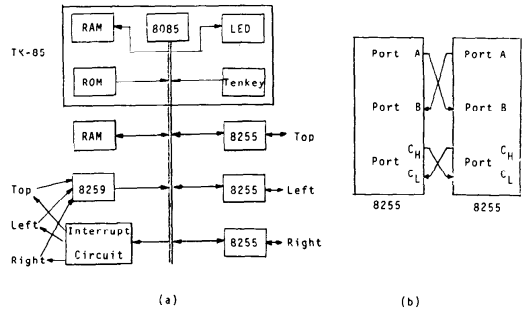

Fig. 6 Processor Element of CORAL '83.

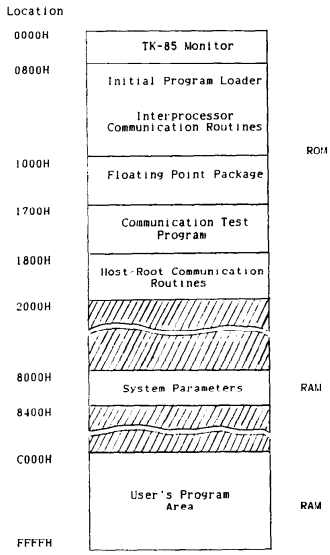

Fig. 7 Memory Map in a Processor Element.

9600 bit/sec serial communication channel. The host computer is a personal computer, if 800 Model 30, which uses a Z80 as cpu.

The processor tree is composed of 15 processor elements each of which is a single-board microcomputer, TK-85, using an 8085 as cpu, provided with 25kB of memory and parallel communication channels. The configuration of a processor element is shown in Fig. 6(a). As a means of interprocessor communication three PPIs (programmable peripheral interface), each an 8255, are provided, one for each of the three directions. As indicated in Fig. 6(b) port A of the PPI is used to send and port B is used to receive 8 bit data, and port C is divided into two 4 bit channels C_H and C_L to send and receive 4 bit control singnals for handshaking. To transmit a one-byte data between two interconnected processor elements, the following handshake procedures are used.

```
/*sender*/                /*receiver*/
begin                     begin
```

Table 1 Port Addresses and Interrupt Commands.

Direction	Port	Address	Interrupt Command
Top	A	03CH	OUT 04FH
	B	03DH	
	C	03EH	
	Control	03FH	
Left	A	02CH	OUT 05FH
	B	02DH	
	C	02EH	
	Control	02FH	
Right	A	01CH	OUT 06FH
	B	01DH	
	C	01EH	
	Control	01FH	

Note: No Top direction for root processor and no Left nor Right direction for leaf processors.

Table 2 System Parameters.

Address	System Parameters
8000H	Interrupt Vector for Top Direction
8004H	Interrupt Vector for Left Direction
8008H	Interrupt Vector for Right Direction
800CH	Interrupt Vector for Host-Root Channel
8010H	Processor Number in LLHH
8012H	Processor Level
8013H	0 when Left processor exists, FF when none
8014H	0 when Right processor exists, FF when none
8015H	number of processor elements underneath

```
    C_H:=05H;                    repeat until C_L=05H;
    repeat until C_L=05H;        C_H:=05H;
    A:=data;                     repeat until C_L=0AH;
    C_H:=0AH;                    data:=B;
    repeat until C_L=0AH         C_H:=0AH
end.                         end.
```

With these handshake programs a reliable data transmission between two processor elements is assured, while sacrificing the transmission speed which is 9 kB/sec by measurement.

Another means for interprocessor communication is the interrupt signal between interconnected processor elements. The interrupt signals are generated to and received from three directions. They are inputted to the PIC (programmable interrupt controller) 8259 with the highest priority to the Top, and the lowest one to the Right direction. The port address and the interrupt command of each direction are indicated in Table 1. There are no Left and Right directions in the leaf processors, and the Top direction of the root processor is for the serial communication channel to the host computer.

One bit of the C_H port for each direction is led to the LED on the display panel to indicate the occurrence and the direction of data transmission between processor elements. The display panel has been very helpful for debugging a parallel program and observing the efficiency of a parallel computation.

Each processor element has 8 kB of ROM and 17 kB of RAM to which the address space is allocated as indicated in Fig. 7. The lowest 8 kB is allocated to ROM in which the monitor program for the TK-85 and the system support programs such as the initial program loader, interprocessor and host-root communication routines, and communication test programs, are stored. The 1 kB RAM area starting from 8000H is used to store system parameters as described in Table 2 and also serves as the working storage for the ROM programs. Another RAM area starting from C000H is 16 kB in size and is used as the user's program area for the operating system and the application program.

4. Software Environment of CORAL '83.

4.1 Initial Program Loader

Software facilities are prepared to support developing application programs for CORAL '83. Among them are the initial program loader, the host-root and interprocessor communication routines, the distributed operating systems, and the *C* compiler. Their functions will be described.

The functions of the initial program loader are to set system parameters in the first RAM area and to deliver copies of an application program from the host computer to all processor elements. There are two initial program loaders, one on the host computer and the other on each processor element. The initial program loader on the host computer reads an application program from the disk file, and sends it to the root processor together with its loading address, size, and starting address. It then transmits a one-byte starting command to the root processor. The initial program loader on each processor element, which is executed as the reset button on the control panel is depressed, receives the application program from the Top direction and retransmits it to its Left and Right processor elements unless it is on the leaf processor. When the starting command is received from the Top direction, it retransmits it again and then jumps to the starting address of the application program.

4.2 Host-Root Communication Routines

The serial communication channel of the host computer is connected to the USART 8251 installed in the Top direction of the root processor. Serial data transmitted from the host computer is converted into a byte of data in the 8251 and when the data is complete, the root processor is interrupted to store it in a ring buffer. To transmit a byte of data to the host computer the root processor writes it into the 8251 to send bit by bit. Byte and block data host-root communication routines both for the host computer and for the root processor are provided.

Table 3 Input/Output Parameters for Interprocessor Communication Routines.

Transmission Mode		Input Parameter	Output Parameter
Byte Data	Send	B data	A error code Carry error flag
	Receive		A error code B data Carry error flag
Block Data	Send	HL data address DE data size	A error code HL next-data address DE remained-data size Carry error flag
	Receive	HL data address DE data size	A error code HL next-data address DE remained-data size Carry error flag

4.3 Interprocessor Communication Routines

As described in the previous chapter data transmission between interconnected processor elements is carried out by handshake using the *C* port of each PPI. Both byte and block data communication routines for each of the three directions are stored in ROM. The input/output parameters of these routines are shown in Table 3. The interrupt routines to initialize the PIC, to send interrupt signals to three individual directions, and to write the EOI (end of interrupt) command in the PIC are also provided.

A test program for the diagnosis of interprocessor communication is also stored in ROM. When it is started from location 1700H in a processor element, the communications to both the Left and the Right directions are tested in all processor elements under this one. When started from 1704H only the Left direction is tested, and when started from 1708H only the Right direction is tested. Results of the test can be observed on the display panel.

4.4 Distributed Operating System

Among several operating systems which have been developed for CORAL '83 [9][10], 'An' is the smallest of all allowing only one user's process. Because of the small RAM area of CORAL '83, operating systems that have extensive facilities do not afford sufficient area for the application program. This is the reason why 'An' was developed. The functions of 'An' are, to download a user's program from the host computer to generate a user's process in each one of the processor elements, and to support the routing of messages exchanged among processor elements.

There are three system processes that support interprocessor communication for the user's process. They are the Interface, the Input, and the Output processes. The relations among these and the user's processes are illustrated in Fig. 8. When a user's process wants to

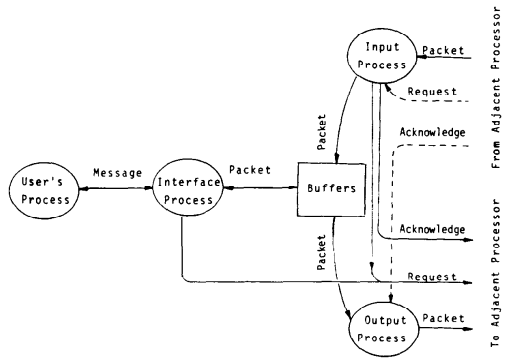

Fig. 8 Relationship among Processes of 'An' Operating System.

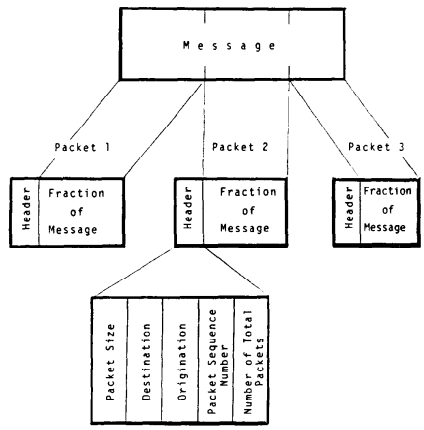

Fig. 9 Disassembling a Message into Packets.

send a message to the user's process on another processor element, it calls the following system subroutine

send (pno, adrs, size)

where pno is the processor number of the destination processor element, adrs is the location of the message, and size is the message size.

The send subroutine activates the Interface process, which disassembles this message into a sequence of packets each consisting of a header and a fraction of the message less than 256 bytes, as illustrated in Fig. 9. In the header is stored the information on destination and origination processor numbers, packet size, packet sequence, and total number of packets. The Interface process stores each packet in a buffer and links it to the tail of the buffer-link for the sending direction. Buffer-links are provided for each of the three sending directions. It then transmits a request interrupt signal to the processor in the sending direction demanding that the processor receive the packet.

In the processor element which receives this request interrupt signal from one of the directions, the Input process is activated. After preparing an empty buffer, the Input process returns an acknowledge interrupt

signal to that direction and waits until a packet arrives. When the packet arrives, the Input process stores the received packet in the buffer and, by finding the direction of retransmission by analyzing the header, links it to the tail of the buffer-link of that direction. If the direction is either Top, Left or Right, it transmits a request interrupt signal to that direction. If the direction is In, the Input process activates the Interface process which assembles the message from the packets and, when completed, forwards it to the user's process. In this processor element the user's process has already called the system subroutine, receive, and has been waiting for the arrival of the message. The call to receive is as follows:

receive (pno, adrs, size)

where pno is the processor number of the originating processor element, adrs is the address to store the message.

When the sending processor element receives the acknowledge interrupt signal from the sending direction, it activates the Output process which picks up a packet from the first buffer of the buffer-link of that direction and transmits it.

4.5 Method of Preparing an Application Program for CORAL '83

To perform a parallel computation on CORAL '83 it is necessary to prepare application programs individually for the host computer and for each processor element. However, a common program may be used for all processor elements if the processor dependent program is written in a way to check the processor number or the kind of current processor element. In this way, usually, two programs, one for the host computer and the other for the processor elements, may be prepared. We call the former the host program and the latter the CORAL program. The host program executes the initial program loader to transmit the CORAL program to all processor elements and let each process element start, sends data and then collects results of the computation from the processor tree to either display, print, or write to a file. It also serves for interactive operations of the user.

These two parallel computing programs are developed on the host computer under the *CP/M* environment. The available programming languages are 8085 asembly language and *C* language. As the cpu of the host computer, the Z80, is upward-compatible with the cpu of the processor elements, the 8085, most of the CORAL program can be debugged on the host computer. Also the *C* compiler, which operates on the host computer, generates 8085 machine code. Therefore the program written in *C* and compiled by the host computer runs on both host computer and procesor element. The only difference is in the run-time and the standard subroutines linked to the compiled program. It is also necessary to specify the program origin to C000H for CORAL programs.

The following additional standard functions of *C* are prepared for the host program.

trans(store_address, start_address, program_name)

start( )

rgetc( )	rgetw( )
rputc(byte_data)	rputw(word_data)

The trans function is used to execute the initial program loader. The functions rgetc and rgetw are used to receive byte or word data from, and the functions rputc and rputw are used to send byte or word data to the root processor.

Additional standard functions prepared for the CORAL program are as follows:

getc(direction)	getw(direction)
putc(direction, byte_data)	putw(direction, word_data)
bufout(direction)	bufin (direction, word_data)

bufinit( )

The first two functions return byte and word data received from the specified direction. The processor element executing the getc or getw function has to wait until the processor element on the specified direction executes the putc or putw function. Also the processor element executing the putc or putw function has to wait until the processor element on the specified direction executes the getc or getw function.

The bufin and bufout functions offer an interrupt-oriented buffered communication procedure without using the operating system. In every processor element ring buffers are prepared, one for each direction. They are initialized with the bufinit function. The program executing the bufin function needs not wait until the processor element on the other side executes the bufout function. When the bufin function is executed in a processor element, an interrupt signal is sent to the specified direction before executing the putw function. The interrupted processor element immediately receives the transmitted data with the getw function and stores the received data in the ring buffer of this direction. As the bufout function reads data from this buffer, the program executing this function need not wait unless the buffer is empty.

5. Parallel Computations on CORAL '83 and Evaluations

5.1 Parallel SOR Computation of Laplace Equation

A number of programs for parallel computation were written for execution on CORAL '83 to evaluate the efficiency of the parallel computation. These programs were written in assembly language in early stages but recently they have been mostly written in *C*. Some programs use the operating system while others do not. The efficiency of the parallel processing is generally

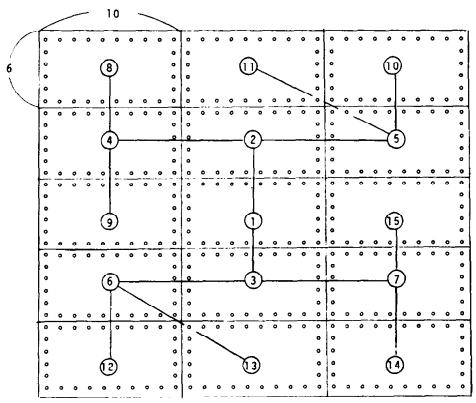

Fig. 10 Partitioning and Allocating Mesh-Points to Processor Elements.

evaluated with two measures. They are the speed-up ratio and the processor utilization factor. The speed-up ratio is the rate of decrease in the computing time by parallel processing, and the processor utilization factor is the ratio of the speed-up ratio to the number of processor elements used. By denoting the computing time, the speed-up ratio, and the processor utilization factor with n processor elements by $T(n)$, $s(n)$, and $u(n)$, the following relations exist.

$$s(n)=T(1)/T(n) \tag{12}$$

$$u(n)=s(n)/n \tag{13}$$

In this chapter the exploitations of the parallel algorithms and the efficiency of some of these programs are described. First, the parallel SOR computation of the following Laplace equation is explained.

$$\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}=0$$

$$w(x, 0)=0,\ w(x, b)=f(x)$$

$$w(0, y)=0,\ w(a, y)=g(y) \tag{14}$$

The two-dimensional Laplace equation can be solved by SOR method, in which a set of linear equations, obtained by dividing the region inside of the boundary into meshes and representing the function w with the values at each mesh-point, is solved by iteration. Assume that this region is divided into 30 by 30 meshes and the obtained mesh points are partitioned into 15 subsets each of which contains 10 by 6 mesh-points as shown in Fig. 10.

The strategy of partitioning and allocating these mesh-points to the processor elements with minimum interprocessor communication is an important problem which was studied by the authors's group [12]. One such allocation is indicated in Fig. 10, where the processor number of the processor element allocated to each partition is indicated in a circle. In this allocation processor element 2, for example, has to exchange 6 pieces of data each with processor elements 4 and 5, and 10 each with processor elements 1 and 11 in each iteration. In addition, it has to route 10 pieces of data exchanged between processor elements 6 and 9, 10 pieces of data exchanged between 5 and 15, and 6 pieces of data exchanged between 8 and 11. Therefore the data sent and received for this processor element amounts to 96 in every iteration. This is the largest number for any processor and therefore limits the computational speed of the system. This program runs under the operating system 'An'.

Table 4 Results of Parallel SOR Computation.

Performance	Measurement	Percentage
$T(1)$	1320 sec	
$T(15)$	119 〃	100 %
Net Computation Time	90 〃	75.6%
Data Transmission Time	21 〃	17.6%
Overhead Time	7 〃	5.9%
$s(15)$	11.2	
$u(15)$	0.75	

As any processor element in CORAL '83 is unable to see the status of all the other processors at the same time, there is some difficulty in detecting the convergence of computation. In this program every processor element informs its Top processor element of convergence when it is notified of convergence from its Left and Right processors and its own computation has converged. In this way the root processor can detect the convergence of the computation.

The results of measurement are shown in Table 4. The net computation time in this Table is measured by decreasing the size of transmitted data to 1 (practically 0) byte and by fixing the iterations of the original program. Consequently, this net computation time includes the computing time in each processor element and the waiting time due to interprocessor synchronization, but no interprocessor communication time. The data transmission time, on the other hand, is measured by removing the computational part from the original program leaving only the data transmission part. Therefore this data transmission time is considered as the time consumed in the interprocessor communication during the computation. The overhead time is the difference between the actual computation time and the sum of the net computing time and the data transmission time. Overhead time is considered as the time consumed in the operating system. Although the speed-up ratio and the processor utilization factor with 15 processor elements are obtained as $1320/119=11.2$ and $11.2/15=0.75$ these values could be improved to $1320/(90+7)=13.5$ and $13.5/15=0.9$ should the data transmission time be negligibly small.

5.2 Parallel Tree Sort

The principle of the parallel tree sort is as follows [13]. Let n be the number of processor elements. The data are divided into n blocks of equal size and one

Table 5 Results of Parallel Tree Sort.

n	Data Size	$T(1)$	$T(n)$	$s(n)$	$u(n)$
3	3 kB	35 sec	20 sec	1.9	0.63
7	7 〃	95 〃	36 〃	3.6	0.51
15	15 〃	295 〃	49 〃	6.0	0.40

of them is delivered to each processor element. Each processor element, after sorting the data block delivered to it, merges the data it has sorted with those sorted data sent from its Left and Right processor elements, and then sends the result to the Top direction. In this way the host computer finally receives the sorted data.

The time to sort N data elements with a single processor is represented as

$$T(1)=S(N) \tag{15}$$

where S is a function which depends on the sorting algorithm.

The time to sort the same data by parallel tree sorting is estimated as follows. As the root processor is the busiest in this computation, the computation time of this processor element is approximately equal to that of the system $T(n)$. The operation of the root processor is as follows. It receives data from the host computer, transmits all but one block of data to the lower processor elements, sorts its own data block, collects data from the lower directions to merge with its sorted data, and transmits the result to the host computer. Therefore the computation time is represented as follows.

$$T(n)=2hN+2p(n-1)N/n+mN+S(N/n) \quad (n\neq 1) \tag{16}$$

Where h and p are the times to transmit a data element between host computer and the root processor and between two interconnected processor elements respectively, m is the time to merge one data element. Therefore the speed-up ratio is represented as follows.

$$\begin{aligned} s(n)&=T(1)/T(n)\\ &=S(N)/[2hN+2p(n-1)N/n+mN+S(N/n)] \end{aligned} \tag{17}$$

Although it gives longer execution time, a better speed-up ratio is obtained when a slower sorting algorithm is used. The Shell sort, for which $S(N)$ is of order $N(\log_2 N)^2$, is used in our program.

This program uses neither operating system nor buffered interprocessor communication routines. The result of the computation is indicated in Table 5. The speed-up ratio and the processor utilization factor for 15 processor elements are 6.0 and 0.4 respectively, which are not very satisfactory.

5.3 *N* Queens Problem

The N queens problem is known as a typical NP complete problem which is usually solved with repeated backtracks. To solve this problem on CORAL '83, the search tree is broken down into many subtrees which are distributed uniformly to all processor elements to search in parallel. That is, each processor element gives sequence numbers to the possible solutions by searching first for two rows and when the remainder of its sequence number divided by n, the number of the processors used, matches with its processor number, it continues searching for further rows. The C program to do this is as follows.

```
int n, N, q[M];
main( )
{int i, j, k, *pno;
 pno=0×8010;
 for (k=0, i=0; i<N; i++)
   for (j=0; j<N; j++)
     if (i!=j && abs (i-j)!=1)
       {k++;
        if (k%n+1==*pno)
          {q[0]=i+1; q[1]=j+1:
           queen(2, q);
          }
       }
}
```

Where pno is the address of the processor number as was indicated in Table 2.

Although the number of subtrees searched by each processor element is almost the same, the computational time for each may be different because the number of solutions obtained may be different. Therefore linear speed-up may not be expected. The program uses the buffered interprocessor communication routines. The relation of the obtained speed-up ratio to the number of processor elements used is shown in Fig. 11. By using 15 processor elements the speed-up ratio of 10.9 and the processor utilization factor of 10.9/15=0.73 are obtained for a 10 queens problem.

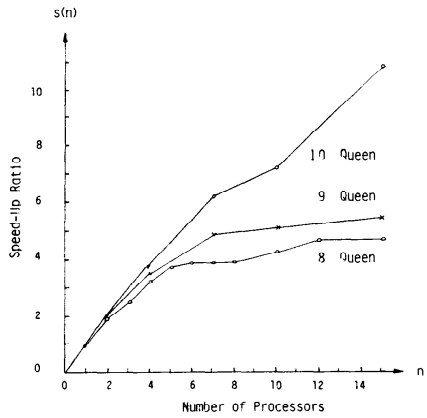

Fig. 11 Speed-up Ratio of Parallel Computing of N Queens Problem.

Table 6 Results of Parallel Computing Prime Numbers up to 40000 with the Sieve of Eratosthenes.

Performance	With Root-to-Host Data Transmission	No Root-to-Host Data Transmission
$T(1)$	940 sec	924 sec
$T(15)$	119 〃	79 〃
$s(15)$	7.9	11.7
$u(15)$	0.53	0.78

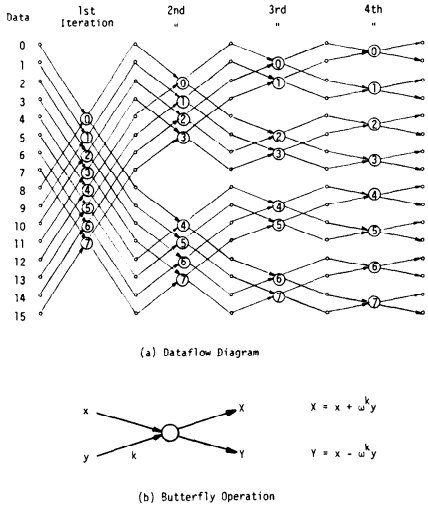

Fig. 12 Data-Flow Diagram of FFT for 16 Data and Butterfly Operation. Each circle represents a butterfly operation.

5.4 The Sieve of Eratosthenes

The sieve of Eratosthenes is a well-known parallel algorithm for computing prime numbers, in which a sequence of natural numbers is introduced into a series of processors each one of which has a different sieve and, if the incoming number is not a multiple of the sieve, sends it to the next stage. This algorithm is transformed and implemented for CORAL '83 as follows.

When all prime numbers less than an integer M are to be obtained, the necessary sieves are odd numbers from 3 to $\sqrt{M}$. The set of sieves is now divided into $L+1$ subsets, where L is the number of levels of the binary-tree machine as before. Then each subset is allocated to the processor elements of each level, which test the numbers sent from the Left and the Right directions with these sieves and send the not-a-multiple ones to the Top direction. The leaf processors also generate different series of natural numbers in addition to testing them with sieves. In this way the host computer finally receives a series of prime numbers.

One problem in this program is how to divide sieves into optimum subsets so that the computational loads of all processor elements balance. In our program a nearly optimum division is realized by allocating twice as many sieves to a processor element as to that in its upper level. That is, the largest sieve H_v and the smallest sieve L_v of processor elements of level v are determined from the following formula.

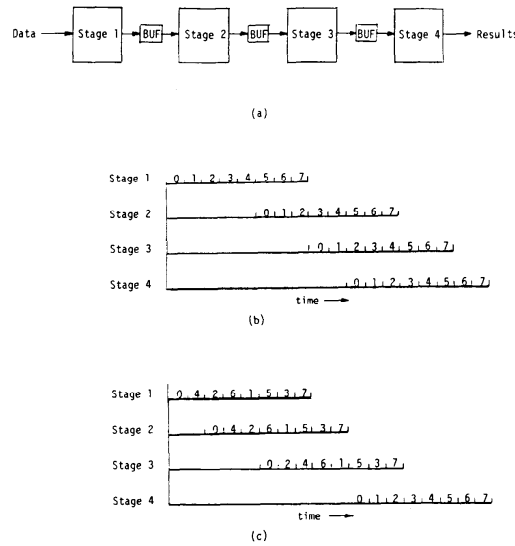

Fig. 13 Space-Time Diagram of Linear Pipeline Processing of FFT for 16 Data. Numbers denote the butterflies in Fig. 12.

$$H_v=\sqrt{M}-(2^v-1)m \tag{18}$$

$$L_v=\sqrt{M}-(2^{v+1}-1)m \tag{19}$$

where

$$m=\sqrt{M}/(2^{L+1}-1) \tag{20}$$

This program uses buffered interprocessor communication routines. The performance of the program for $M=40000$ is shown in Table 6. As the host-root communication speed limits the computational speed, the result of a program which omits transmitting results to the host computer is also shown. The speed-up ratio of $924/79=11.7$ and the processor utilization factor of $11.7/15=0.78$ are obtained.

5.5 Pipelined Computation of FFT

The FFT is a computation suited to pipelined processing and divide-and-conquer methods, both of which are properly implemented on the binary-tree machine. In the FFT for 2^n data, n iterations of computing 2^{n-1} butterflies are required. The data-flow diagram in Fig. 12(a) represents the computation of an FFT for $n=4$. The circles in this figure denote the butterfly operations as represented in Fig. 12(b) and the number in each circle identifies the individual butterfly. The computation of this FFT can be carried out with a 4-stage pipeline with buffer storages between stages as shown in Fig. 13(a), where the i-th stage is assigned to computing the butterflies of the i-th iteration. When each stage computes the butterflies in normal order the space-time

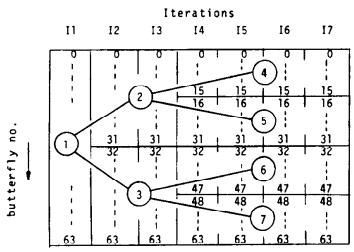

Fig. 14 Partitioning and Allocating Butterflies for Tree Pipeline Processing of FFT. I1 . . . I7 denote the iterations and the numbers represents the butterfly numbers. The circles denote the processor elements to which the partition of butterflies is allocated.

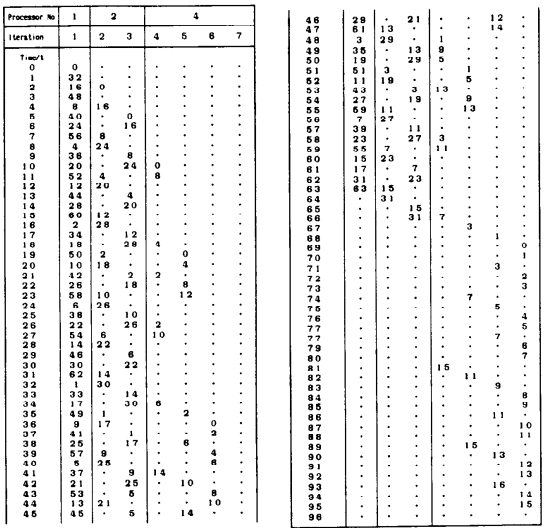

Fig. 15 Result of Simulated Tree Pipeline Processing of 128 Data FFT Indicating the Identification Numbers of Butterflies Computed in Sequence by the Processor Elements.

diagram of this pipeline is as shown in (b).

Let t be the computation time for one butterfly. The total computing time T is wirtten as

$$T=18t. \tag{21}$$

The same result is obtained when the first stage computes in bit-reversal order which is shown in (c). The computation time for an FFT for 2^n data with an n-stage pipeline is derived as follows.

$$T=[(2^{n-1}+1)+(2^{n-2}+1)+\ldots+(2^1+1)+1]t$$
$$=(2^n+n-2)t. \tag{22}$$

The processor utilization factor is then

$$u=2^{n-1}/(2^n+n-2)<0.5. \tag{23}$$

That is, in a linear pipeline computation of an FFT, the processor utilization factor can never exceed 50%.

Now the pipelined computation of an FFT on the binary-tree machine is considered. To balance the computational loads among processor elements, the butterflies are allocated to the processors of the binary tree machine in the way indicated in Fig. 14. That is, the computation of all butterflies of the first iteration is assigned to the root processor and those of the second and the third iterations are divided into two halves. The computation of one of the halves is assigned to each processor element of level 1. In general, for each of the j-th level processor elements, one 2^jth of the butterflies in the 2^jth to $(2^{j+1}-1)$th iterations are allocated. In this way each processor element needs not communicate with other processor elements except receiving data from the processor element in Top direction and sending the computed results to those in the Left and Right directions.

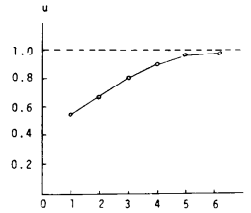

Fig. 16 Processor Utilization Factor in Tree Pipeline Processing of FFT.

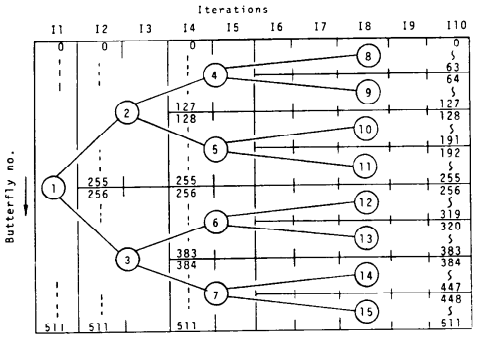

Fig. 17 Partitioning and Allocating Butterflies when Unbalance of Load is Unavoidable.

The computation in each of the iterations is performed as follows. In the first iteration the butterflies are computed in bit-reversal order. Two butterflies in the j-th iteration can be computed only after two butterflies in the $(j-1)$th iteration, for which sequence numbers are 2^{n-j} apart, are computed. The computation process is simulated and the result is shown in Fig. 15, in which the sequence of computations of butterflies of each iteration by processor elements 1, 2 and 4 is illustrated. For $n=7$ the computation time, the speed-up ratio, and the processor utilization factor are as follows.

$$T=96t \tag{24}$$

$$s=(64\times 7)t/T=4.7 \tag{25}$$

$$u=4.7/15=0.67 \tag{26}$$

Comparing these results with Eq. (23) reveals that the efficiency of the tree pipeline processing is better than that of the linear pipeline processing.

The computation time and the speed-up ratio for the FFT for 2^n data with an L level binary-tree machine, where $n=2^{L+1}-1$, can be derived as follows by studying the result in Fig. 15 in which $L=2$.

$$T=2^{2L+1-2}+1+\sum_{l=1}^{L}(2^{2^{l+1}-l-1}-2^{2^l-l-1}) \tag{27}$$

$$u=2^{2^{L+1}-2}/T \tag{28}$$

In Fig. 16 u is plotted against L. As is observed the processor utilization factor asymptotes 1 as the number of processor elements increases.

We now consider a case when there are more processor elements than the iterations of an FFT and their loads are unbalanced. For example, in computing the FFT for 2^{10} data with a 4 level binary-tree machine, the allocation shown in Fig. 17 may be applicable, where the level 2 and 3 processor elements are less loaded than others. The simulation of this allocation reveals that

$$T(15)=640t \tag{29}$$

$$s(15)=(512\times 10)t/T(15)=8.0 \tag{30}$$

$$u(15)=8.0/15=0.53 \tag{31}$$

The program for this algorithm was implemented and executed on CORAL '83 and the following results were obtained.

$$T(1)=202 \text{ sec} \tag{32}$$

$$T(15)=23.0 \text{ sec} \tag{33}$$

$$s(15)=202/23.0=8.8 \tag{34}$$

$$u(15)=8.8/15=0.59 \tag{35}$$

The greater speed-up ratio obtained in actual computation than in simulation results from the reason that the weights used in calculating butterflies in the first and the second iterations are either 1 or -1, which require no multiplication, while in simulation the computation time for all butterflies was assumed the same.

5.6 Discussion

In the preceding sections the efficiencies of parallel computing of three types of problem have been studied. The processor utilization factors for 15 processor elements range from 0.4 to 0.78. The first type of problem is the near-neighbor problem for which the lattice machine is believed to outperform the binary-tree machine. The parallel SOR computation is this type of problem. As was reported in [14] the processor utilization obtained with 128 processors PAX-128 for computing 32 by 32 mesh-points is 0.83, while our result with 15 processor elements for 30 by 30 mesh-points is 0.7 which is less than but not very much behind the lattice machine. In this comparison, the numbers of mesh-points allocated to one processor element are 8 for PAX and 60 for CORAL. The processor utilization may be smaller if the same number of mesh-points is allocated to the processor elements of CORAL.

The second type of problem is the unstructured one. The N queens problem solved in our program falls in this category. In this program the tree-structure inherent in this problem was not utilized, but all processor elements are made to operate almost independently of each other. The processor utilization factor for this problem is fairly good.

The tree sort, the sieve of Eratostenes, and the FFT belong to the third type of problem, which is tree-structured. This type of problems appears to fit the binary-tree machine. However, the processor utilization factor for the parallel tree sort is as small as 0.4, while those of the other programs are satisfactory. The reason is that, in a parallel tree sort the data to merge in upper processor elements are much greater than those in lower ones, while in other programs the loads are well balanced in all processor elements.

From the above discussions we can conclude that the overhead due to interprocessor communications is reasonably small in the binary-tree machine, and if a successful load balancing is accomplished, a satisfactory performance is obtained.

6. Conclusion

Having developed a binary-tree machine CORAL '83 and performed several parallel computations with it, the following conclusions are obtained.

1) The binary-tree architecture adapts well to the parallel computation of a wide range of problems.

2) Although the binary-tree machine generally adapts to the problems with tree structure, a careful load balancing among processor elements is necessary.

3) The amount of interprocessor communication in the binary-tree machine does not degrade the efficiency of parallel computation by much.

As CORAL '83 is a prototype machine, several problems, that have to be improved to make it competitive with conventional computers, are observed. They are as follows.

1) The slow host-root communication speed reduces the efficiency of problems which need a great deal of input or output data.

2) The rather slow interprocessor communication sometimes limits the efficiency of the parallel processing.

3) The small size of the local memory of each processor element does not afford a sufficient area for the application program when the operating system is used.

4) The computing speed of the processor element itself is poor.

To dissolve these shortcomings we are now undertaking a project to build a new binary-tree machine composed of 63 16-bit microprocessors connected with a DMA channel [15]. When this is complete, many real-scale problems will be solved much more efficiently than

are those presented in this article.

The authors are grateful to Prof. Yeng Jia Fung of Benjing Technical Institute of Architecture whose suggestive discussions with us, during his stay at our laboratory from 1983 to 1984, helped the progress of this study.

References

1. HOROWITZ, E. and ZORAT, A. The Binary Tree as an Interconnection Network: Application to Multiprocessor System and VLSI, *IEEE Trans.*, **C-30**, 4 (1981), 247-253.
2. TAKAHASHI, Y. Processor Interconnection Systems for Parallel Processors, *J. IPSJ*, **23**, 3(1982), 201-209 (in Japanese).
3. TAKAHASHI, Y., WAKABAYASHI, N. and NOBUTOMO, Y. A Binary Tree Multiprocessor:CORAL, *J. Inf. Process.*, **3**, 4 (1981), 232-237.
4. HOSHINO, T. et al. PACS: A Parallel Microprocessor Array for Scientific Calculations, *ACM Trans. Comput. Syst.*, **1**, 3 (Aug. 1983), 195-221.
5. TAKAHASHI, Y. Fault Tolerance in Processor Networks, *Proc. 25th Annual Convention IPSJ* (1982), 1F-6.
6. DESPAIN, A. and PATTERSON, D. Xtree: A Structured Multiprocessor Computer Architecture, *Proc. 5th Symp. Comput. Arch.* (April 1978), 144-151.
7. RAGHAVENDRA, C. S., AVIZIENIS, A. and ERCEGOVAC, M. D. Fault Tolerance in Binary Tree Architectures, *IEEE Trans.*, **C-33**, 6 (June 1984), 568-572.
8. TAKAHASHI, Y. A Distributed Operating System for a Binary Tree Multiprocessor, *Proc. 14th IBM Comput. Sci. Symp.* (Oct. 1980), Operating System Engineering, Lecture Notes in *Computer Science*, **143**, 270-286.
9. NOBUTOMO, Y. and TAKAHASHI, Y. Performance Evaluation of Binary Tree Multiprocessor CORAL Prototype, *Trans. Comput. Arch. IPSJ*, **44**-1 (Feb. 1982) (in Japanese).
10. YENG, J. F. and TAKAHASHI, Y. BTOS: A Distributed Operating System for Binary Tree Multiprocessor CORAL, *Proc. 29th Annual Convention IPSJ* (1984), **7E-7** (in Japanese).
11. NISHIYAMA, Y. and TAKAHASHI, Y. Parallel Processing of a Functional Programming Language FP on Binary Tree Multiprocessor CORAL, *Proc. 27th Annual Convention IPSJ* (1983), **6N-4** (in Japanese).
12. TAKAHASHI, Y., NOBUTOMO, Y. and KAWAMURA T. Strategies and Performance Evaluation in Solving Laplace Equation, *J. Inf. Process.*, **5**, 4 (1982), 239-246.
13. MEADS C. and CONWAY, L. Introduction to VLSI systems, *Addison-Welsley*, (1980).
14. SHIRAKAWA, T., KAGEYAMA, T., ABE, H. and HOSHINO, T. Processor Array PAX-128, *Trans. IECE*, **J67-D**, **8** (Aug. 1984), 853-860 (in Japanese).
15. FUJIMOTO, K., KUWAHARA, A. and TAKAHASHI, Y. Design of Prototype Binary Tree Machine CORAL with MC68000, *Proc. 29th Annual Convention IPSJ* (1984), **5B-2** (in Japanese).

(Received July 22, 1985; revised November 22, 1985)