Strategies and Performance Evaluation of Parallel Computation in Solving the Laplace Equation

Yoshizo Takahashi,† Yoshihiro Nobutomo,††
and Toshikazu Kawamura†††

The factors which have influence on the efficiency of parallel computation are the scheme of problem decomposition, the processor interconnection, and the OS overhead. In this paper, the scheme of problem decomposition of parallel computation in solving the Laplace equation numerically is studied from two different viewpoints, the strategy of partitioning mesh-points into blocks and the strategy for assignment of these blocks to the processors. Two matrices called the partition matrix and the distance matrix are introduced and an algorithm to give an optimum processor assignment to the blocks in the sense that it results in minimum data transmission in the whole system is presented. The performance of parallel computation in solving a Lapalce equation is evaluated by developing a software simulator of CORAL system which is a binary tree processor network. The result, in comparison with serial computation, shows that a speed-up ratio of 100 is attained with 300 processors in a binary tree.

1. Introduction

The numerical solution of partial differential equation is obtained by solving a set of difference equations with respect to the unknown variables at the mesh-points which are obtained when the whole domain is divided into small subdomains. In order to solve this set of difference equations by a parallel processing system with multiple processors, the mesh-points are partitioned into as many blocks as the number of processors, and the computation of the solution of the equations at the mesh-points belonging to one block is assigned to one processor.

The strategy for assigning the processors to the blocks affects the performance of the parallel processing. In the processor assignment it is desired that the following conditions are satisfied:

(1) The load on each processor is balanced.
(2) The synchronization among processors is small.
(3) The iteration in the numerical computation converges.
(4) The amount of data transmission among processors is small.
(5) The processors which communicate with each other are located in the nearest neighbor.

The first condition may be satisfied when the same number of mesh-points are assigned to each processor, provided the overhead of the operating system is uniformly distributed to all processors. However, in a parallel processing system with a hierarchical structure such as CORAL,[1] this assumption may not hold. For example, in CORAL which has a binary tree structure as shown in Fig. 1, two processors No. 2 and No. 3 bear the largest routing load.[2] It is, therefore, recommended to vary the size of the blocks depending on the role of the processors to which they are allocated.

The second condition may lead to the attempt of asychronous iteration in the numerical solution as realized on C.mmp.[3] It was, however, observed that, under certain conditions, the asynchronous iteration diverged even if the acceleration factor of iteration was less than 2.[4] It is also possible to make the synchronization less often if the mesh-points are distributed properly to the processors. The third condition has been studied recently by Yokoyama.[5]

In the following Chapter, the strategy of partitioning mesh-points to decrease the amount of data exchange with other processors and the number of neighboring processors with which one processor communicates is discussed. In Chapter 3 the strategy for assignment of the blocks to the processors which minimizes the interprocessor communications is studied. In Chapter 4 the effect of the partitioning on the process of numerical iteration is studied. In Chapter 5 some results obtained

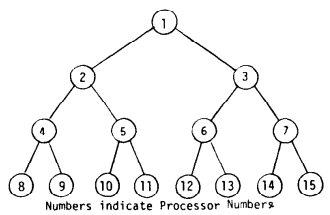

Fig. 1 Binary Tree Processor Network CORAL.

†Department of Information Science, Faculty of Engineering, Tokushima University, Tokushima 770, Japan.
††Department of Industrial Control Systems, Omika Works, Hitachi Co., Ltd., Omikacho 5-2-1, Hitachi 319-12, Japan.
†††Heavy Apparatus Engineering Laboratory, Toshiba Inc. Toshibacho 1, Fuchu 183, Japan.

from the simulation experiments of parallel computation in solving Laplace equations on CORAL with several hundreds of processors are described. Chapter 6 presents conclusions.

2. Strategy of Partitioning Mesh-points

In this Chapter the method of partitioning mesh-points which have a small amount of data exchange with other processors and a small number of neighboring processors is investigated. As an example the parallel solution of a two-dimensional Laplace equation in the rectangular domain $0<x<X$, $0<y<Y$ is considered. The equation is:

$$\left.\begin{aligned} &\frac{\partial^2 u(x,y)}{\partial x^2}+\frac{\partial^2 u(x,y)}{\partial y^2}=0 \\ &u(x,0)=f_0(x),\ u(x,Y)=f_1(x) \\ &u(0,y)=g_0(y),\ u(X,y)=g_1(y) \end{aligned}\right\} \quad (1)$$

The domain is divided into small square subdomains as indicated in Fig. 2 the side length of which is equal to

$$h=X/(M+1)=Y/(N+1) \quad (2)$$

i.e. we assume here that $X=h(M+1)$ and $Y=h(N+1)$ for simplicity. Then we obtain $(M+1)\times(N+1)$ mesh-points including those on the boundary. If we denote $u(ih, jh)$ by u_{ij}, the five-point difference approximation to Eq. (1) is written as

$$u_{ij}=(u_{i-1,j}+u_{i+1,j}+u_{i,j-1}+u_{i,j+1})/4 \quad (i=1,\cdots,M, j=1,\cdots,N). \quad (3)$$

This set of lincar equations is solved with SOR method in which the value of u_{ij} corresponding to the $(k+1)$th iteration, denoted by $u_{ij}^{(k+1)}$, is calculated by

$$u_{ij}^{(k+1)}=[u_{i-1,j}^{(k+1)}+u_{i,j-1}^{(k+1)}+u_{i+1,j}^{(k)}+u_{i,j+1}^{(k)}]\omega/4+(1-\omega)u_{ij}^{(k)} \quad (4)$$

where ω denotes an acceleration factor.[6] The above equation indicates that in order to obtain the value at one mesh-point we need the values at the four neighboring mesh-points.

Suppose that we want to partition all the mesh-points into n blocks of equal size where n is the number of processors of the system. Let the number of mesh-points belonging to one block be K, then

$$K=MN/n. \quad (5)$$

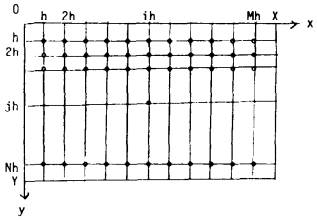

Fig. 2 Rectangular Domain represented by M×N Mesh-Points.

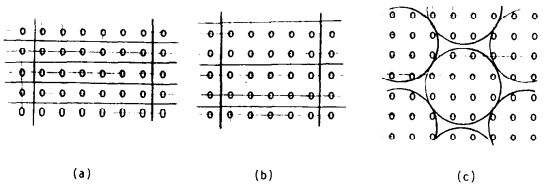

Fig. 3 Various Partitions.

There are various ways of partitioning. An example is shown in Fig. 3(a). In this example the number D of the data that the processor assigned to this block exchanges with other neighboring processors is given by

$$D=2(K+1). \quad (6)$$

When the blocks consist of two successive rows as in Fig. 3(b), then

$$D=2(K/2+2)=K+4 \quad (7)$$

which is less than that of Eq. (6). The minimum D is obtained when the shape of the block is a square of side-length of $\sqrt{K}$. In this case

$$D=4\sqrt{K}. \quad (8)$$

Since D is roughly equal to the number of the mesh-points on the boundary of the block, the minimum D is attained when the shape of the block is a circle. Let $K=12$, then the partitioning of Fig. 3(c) results in the minimum D, which is

$$D=12<4\sqrt{12}. \quad (9)$$

The result shows that circular partitioning is superior to square partitioning. Circular partitioning, however, increases the number of neighboring processors with which data exchange is required. In Fig. 3(c) the number the number of neighboring processors is 6, while that of (a) or (b) is 4. When the block is composed of complete rows or columns as indicated in Fig. 4(a), the number of neighboring processors is 2 and D is given by

$$D=2M=2K/N, \quad \text{or} \quad D=2N=2K/M \quad (10)$$

where M and N are the number of elements of the row and the column of the mesh-point matrix, respectively. The partitioning by row or by column is preferrable also when variable-sized blocks are required.

Assume that the blocks are numbered from 1 to n, where n is the number of processors as previously

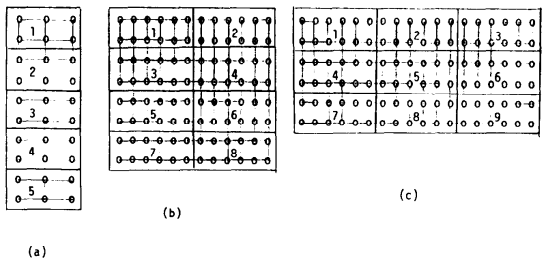

Fig. 4 Various Partitioning Strategies.

mentioned. We denote the number of the data that the processor assigned to ith block exchanges with the processor assigned to jth block by p_{ij}. We call the matrix (p_{ij}) the Partition Matrix. The partition matrix is symmetrical. For example the partition matrix of Fig. 4(a) is as follows.

$$(p_{ij})=\begin{pmatrix}0&3&0&0&0\\3&0&3&0&0\\0&3&0&3&0\\0&0&3&0&3\\0&0&0&3&0\end{pmatrix} \tag{11}$$

The partition matrix of a partitioning like Fig. 4(b) is as follows.

$$(p_{ij})=\begin{pmatrix}0&2&6&0&0&0&0&0\\2&0&0&6&0&0&0&0\\6&0&0&2&6&0&0&0\\0&6&2&0&0&6&0&0\\0&0&6&0&0&2&6&0\\0&0&0&6&2&0&0&6\\0&0&0&0&6&0&0&2\\0&0&0&0&0&6&2&0\end{pmatrix} \tag{12}$$

Finally that of the partitioning of Fig. 4(c) is given by

$$(p_{ij})=\begin{pmatrix}0&2&0&6&0&0&0&0&0\\2&0&2&0&6&0&0&0&0\\0&2&0&0&0&6&0&0&0\\6&0&0&0&2&0&6&0&0\\0&6&0&2&0&2&0&6&0\\0&0&6&0&2&0&0&0&6\\0&0&0&6&0&0&0&2&0\\0&0&0&0&6&0&2&0&2\\0&0&0&0&0&6&0&2&0\end{pmatrix} \tag{13}$$

Thus the strategy for partitioning is represented by the partition matrix. In fact, some characteristics of the partitioning are derived from the partition matrix. For example, the number D_i of the data which the processor assigned to ith block exchanges with other processors is given by

$$D_i=\sum_{i=1}^{n} p_{ij}. \tag{14}$$

3. Strategy for Processor Assignment

The next problem is how to assign processors to the blocks. Assume that the processors are connected as shown in Fig. 5(a). If the blocks of Fig. 5(b) are assigned to these processors as indicated in Fig. 5(a), all the exchanged data must pass through other processors, so that the efficiency of parallel computation would be very low. Hence a strategy of assigning the processors to the blocks which results in minimum routing overhead is required.

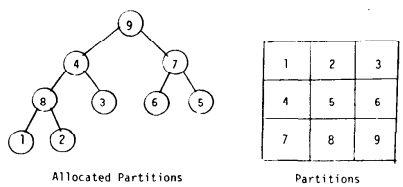

Fig. 5 An Inefficient Partition Allocation to Processors.

Consider a processor network consisting of n processors. Let the distance between two processors be the number of connections in the shortest path between them. The distance between directly connected processors, for example, is 1. The distance from the ith processor to the jth processor is denoted by d_{ij}, and we call the matrix (d_{ij}) the Distance Matrix. The distance matrix is also symmetrical. The distance matrix of the CORAL in Fig. 1, for example, is given as follows.

$$(d_{ij})=\begin{pmatrix}0&1&1&2&2&2&2&3&3&3&3&3&3&3&3\\1&0&2&1&1&3&3&2&2&2&2&4&4&4&4\\1&2&0&3&3&1&1&4&4&4&4&2&2&2&2\\2&1&3&0&2&4&4&1&1&3&3&5&5&5&5\\2&1&3&2&0&4&4&3&3&1&1&5&5&5&5\\2&3&1&4&4&0&2&5&5&5&5&1&1&3&3\\2&3&1&4&4&2&0&5&5&5&5&3&3&1&1\\3&2&4&1&3&5&5&0&2&4&4&6&6&6&6\\3&2&4&1&3&5&5&2&0&4&4&6&6&6&6\\3&2&4&3&1&5&5&4&4&0&2&6&6&6&6\\3&2&4&3&1&5&5&4&4&2&0&6&6&6&6\\3&4&2&5&5&1&3&6&6&6&6&0&2&4&4\\3&4&2&5&5&1&3&6&6&6&6&2&0&4&4\\3&4&2&5&5&3&1&6&6&6&6&4&4&0&2\\3&4&2&5&5&3&1&6&6&6&6&4&4&2&0\end{pmatrix} \tag{15}$$

Assume that the blocks 1, 2, 3, $\cdots$, n are assigned to the processors k_1, k_2, k_3, $\cdots$, k_n, that is,

block	1	2	3	$\cdots$	i	$\cdots$	n
processor	k_1	k_2	k_3	$\cdots$	k_i	$\cdots$	k_n.

The distance between two processors, which are assigned to the ith and the jth blocks respectively, is represented by $d_{k_ik_j}$. As the amount of data exchanged between these processors is p_{ij}, the total amount of data transmitted by these two processors plus the processors on the path between them is represented by $p_{ij}d_{k_ik_j}$. The total amount of data transmission in the whole system, which we denote by T, is obtained as follows.

$$T=\sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}d_{k_ik_j} \tag{16}$$

We, therefore, conclude that given the partition matrix (p_{ij}) and the distance matrix (d_{ij}), the permutation $(k_1k_2\cdots k_n)$ that results in a minimum T gives the optimum processor assignment. Although a straightforward method to obtain this permutation is not yet known, the program shown in Fig. 6 may be used to obtain it. Note that, as this program is a recursive one, the computation time increases explosively as the number of processors increases.

Here we note some of the results obtained by this program. For CORAL with five processors, the optimum

```
program allocate(input,output);
const size=20; bignum=1000000;
type row=array[1..size] of integer; matrix=array[1..size] of row;
var i,j,k,min,md,n:integer;vectr,minrow:row; p,d:matrix;
procedure sum(var m:integer);
   var i,j:integer;
   begin
      m:=0;
      for i:=1 to n do for j:=1 to n do
         m:=m+p[i,j]*d[vectr[i],vectr[j]]
   end;
procedure select(length:integer);
   var i,j,k,e:integer;
   begin
      if length=n
         then sum(md)
         else begin
            for i:=1 to n do begin
               k:=i;
               if length ≠0 then
                  for j:=1 to length do
                     if vectr[j]=i then k:=0;
               if k≠0 then begin
                  e:=length+1; vectr[e]:=k;
                  select(e);
                  if md< min then begin
                     min:=md; minrow:=vectr
                  end
               end
            end
      end
   end;
begin(*allocate*)
   read(n); writeln('PARTITION MATRIX');
   for i:=1 to n do begin
      for j:=1 to n do begin read(k);write(k:3);p[i,j]:=k;end;
      writeln;
   end; writeln('DISTANCE MATRIX');
   for i:=1 to n do begin
      for j:=1 to n do begin read(k);write(k:3);d[i,j]:=k;end;
      writeln;
   end;
   min:=bignum;
   select(0);
   writeln('MINIMUM DATA TRANSMISSION IS',min);
   write ('PARTITION'); for i:=1 to n do write(i:3);writeln;
   write('PROCESSOR'); for i:=1 to n do write(minrow[i]:3)
end.
```

Fig. 6 A Pascal Program for Optimum Processor Assignment.

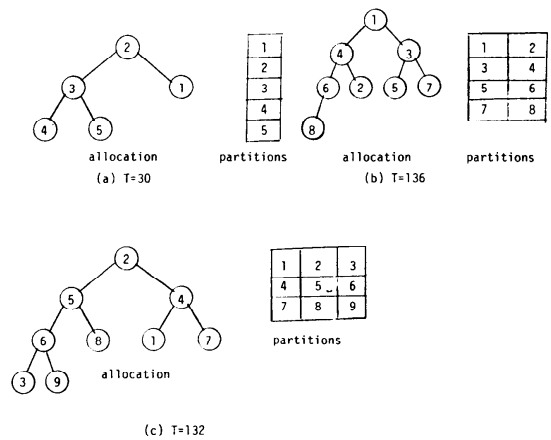

Fig. 7 Optimum Processor Assignments to the Blocks of Fig. 4.

assignment of the blocks of Fig. 4(a) is obtained as in Fig. 7(a). The result coincides with the one previously obtained by another algorithm of the authors'.[7] The optimum assignment of the blocks of Fig. 4(b) to the CORAL with eight processors is obtained as (1 5 3 2 6 4 7 8), which is shown in Fig. 7(b). Finally the optimum allocation of the blocks of Fig. 4(c) to the CORAL with nine processors is obtained as (6 1 8 3 2 4 7 5 9), which is shown in Fig. 7(c). From this figure it is observed that the neighboring blocks (1 4 2), (2 5 8), and (3 6 9) are allocated to the proximate processors. If the allocation of Fig. 5(a) is used the value of T is as large as 296 which is more than twice of that of the optimum allocation which is 132.

4. Effect of Partitioning Strategy to the Process of Iterations

In the parallel SOR method, the processors are forced to sysnchronize with each other for exchanging data. As the processor synchronization causes a decrease in the efficiency of parallel computation, we investigate a process of iteration in conjunction with the mode of partitioning which has high influence upon synchronization. To make the discussion simpler, a rectangular partitioning is assumed as shown in Fig. 8. The processor which is assigned to this block exchanges data with the other four processors which are assigned to the neighboring blocks. In the block shown in Fig. 8 the computation is performed according to the algorithm described below:

```
begin
  repeat
    for i: =i1 to i2 do
      for j: =j1 to j2 do
        begin
          if i=i1 then receive data from left;
          if i=i2 then receive data from right;
          if j=j1 then receive data from top;
          if j=j2 then receive data from bottom;
          compute u[i,j];
          if i=i1 then send u[i,j] to left;
          if i=i2 then send u[i,j] to right;
          if j=j1 then send u[i,j] to top;
          if j=j2 then send u[i,j] to bottom;
        end
  until it converges.
end.
```

Left, right, top, and bottom denote the processors assigned to the blocks in these directions respectively. We call this process a local iteration in a block. In this algorithm the computation of u_{ij} can be done only after

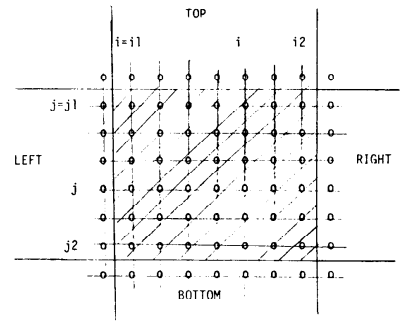

Fig. 8 A Rectangular Partition with its Neighbors.

$u_{i-1,j}$ and $u_{i,j-1}$ in the same local iteration are computed. Thus the computation in each block proceeds in a sequential manner.

We can estimate the number of local iterations that each mesh-point encounters as follows. Let $I_{ij}(k)$ denote the number of local iterations for u_{ij} in the kth iteration step, then the following relations hold.

(1) At the initial computation step, only u_{11} can be computed, so that

$$I_{11}(0)=1, \text{ and } I_{ij}(0)=0, \text{ for } i\neq 1 \text{ and } j\neq 1. \quad (17)$$

(2) When the computation of the last mesh-point of the block is finished, a new local iteration starts from the first mesh-point of the block. So that,

$$I_{i1,j1}(k+1)=I_{i2,j2}(k)+1, \text{ if } I_{i1,j1}(k)=I_{i2,j2}(k). \quad (18)$$

(3) Unless a mesh-point is on the left side, the computations proceed from left to right in one local iteration. Therefore

$$I_{ij}(k+1)=I_{i-1,j}(k), \text{ if } i\neq i1 \quad (19)$$

(4) The computation for a mesh-point on the left side is performed when all points on the right side of the previous row are finished. Therefore

$$I_{i1,j}(k+1)I=_{i2,j-1}(k), \text{ if } j\neq j1. \quad (20)$$

However, if a mesh-point is in the first row of the block, the computation is performed after the computation of the mesh-points immediately above it is finished. That is,

$$I_{i,j1}(k+1)=I_{i,j-1}(k). \quad (21)$$

Using the above recurrence relations the total number of local iterations at any mesh-point can be calculated. Figure 9(a) shows the numbers of local iterations at each mesh-point, when 2nd local iteration is performed in the last block of 15×15 matrix partitioned into 15 rows. Figure 9(b) also shows the result for another partitioning, where 5×5 mesh-points compose a block. The difference between the first and the last blocks is 1 for Fig. 9(a), whereas the difference is 3 for Fig. 9(b). When the difference is large, the efficiency of the parallel computation is poor, because the processors computing for further iterations may be doing excessive tasks which are unnecessary.

Let us now investigate an optimum partitioning strategy which minimizes difference of the number of the local iterations between the first and the last blocks. In Fig. 10 the number of computation steps which are required to proceed to the specified mesh-point in the first local iteration is indicated for a typical partitioning. Suppose that a matrix of $M\times N$ mesh-points is divided into blocks with $m\times n$ mesh-points. The number of computation steps required to proceed to the last mesh-point for the first iteration, which is denoted by S, is given by

$$S=M-m+\left(\frac{N}{n}-1\right)(mn-m+1)=M+\left(m-\frac{m-1}{n}\right)N-1. \quad (22)$$

Let $K=mn$, then S is represented as

$$S=M-\frac{N}{K}[m^2-(K+1)m]-1. \quad (24)$$

From this equation it is observed that S attains its minimum when either m or n is 1 or K and attains its maximum when $m=(K+1)/2$. The minimum and the maximum of S are as follow.

$$S_{\min}=M+N-1 \quad (25)$$

$$S_{\max}=M+\frac{(K+1)^2}{4K}N-1\sim M+\frac{NK}{4} \quad (26)$$

$S_{\min}$ may not be attained if $K<M$ or $K<N$. This concludes that the partitioning by row or by column, if possible, is superior to the partitioning by both row and column. The difference between $S_{\min}$ and $S_{\max}$ is rather remarkable.

The difference in the number of local iterations of the first and the last blocks is obtained from the following reasoning. As S computation steps are required before the last block completes the first iteration step, the number of local iterations in the first block during this period is S/K. Therefore, the difference between the

3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	2	2
3	3	3	3	3	3	3	3	3	3	3	3	2	2	2
3	3	3	3	3	3	3	3	3	3	3	2	2	2	2
3	3	3	3	3	3	3	3	3	3	2	2	2	2	2
3	3	3	3	3	3	3	3	3	2	2	2	2	2	2
3	3	3	3	3	3	3	3	2	2	2	2	2	2	2
3	3	3	3	3	3	3	2	2	2	2	2	2	2	2
3	3	3	3	3	3	2	2	2	2	2	2	2	2	2
3	3	3	3	3	2	2	2	2	2	2	2	2	2	2
3	3	3	3	2	2	2	2	2	2	2	2	2	2	2
3	3	3	2	2	2	2	2	2	2	2	2	2	2	2
3	3	2	2	2	2	2	2	2	2	2	2	2	2	2
3	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

(a)

5	5	4	4	4	4	4	4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4	4	4	4	4	3	3	3
4	4	4	4	4	4	4	3	3	3	3	3	3	3	3
4	4	4	4	4	4	3	3	3	3	3	3	3	3	3
4	4	4	4	4	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	2	2
3	3	3	3	3	3	3	3	3	3	3	2	2	2	2
3	3	3	3	3	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

(b)

Fig. 9 Iteration Steps for All Mesh-points when the Last Block Completes the 2nd Local Iteration.

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	10	11	12
7	8	9	10	11	12	13	14	15
10	11	12	13	14	15	16	17	18
11	12	13	14	15	16	17	18	19
14	15	16	17	18	19	20	21	22
17	18	19	20	21	22	23	24	25
20	21	22	23	24	25	26	27	28
21	22	23	24	25	26	27	28	29
24	25	26	27	28	29	30	31	32
27	28	29	30	31	32	33	34	35
30	31	32	33	34	35	36	37	38

M=9, N=12, m=3, n=4 K=12

Fig. 10 Number of Computation Steps Required to Proceed to Each Mesh-Point for the First Time.

iteration steps of the first and the last block is

$$I_{\text{first}} - I_{\text{last}} = S/K - 1. \quad (27)$$

5. Performance Evaluation by Computer Simulation

A software simulator of CORAL system is developed and the performance of parallel computation in solving a Laplace equation on CORAL is evaluated. The outline of the simulator developed is illustrated in Fig. 11. There are eleven subroutines and one main program. The main program controls the computation steps of up to 450 processors connected in a binary tree structure by maintaining the system tables. It also controls the clocks of all processors by the CPUTIM subroutine. After reading the parameters such as the sizes of the mesh-point matrix and of the blocks, the main program initializes the system tables and then calls the DISTRIBUTE subroutine which allocates blocks to individual processors according to a given policy. The PROCESS subroutine is then called to simulate the operation of the processor as described in the algorithm in Chapter 4. The routing algorithm used is as follows.

```
begin
  leftadrs: =ownadrs*2; rightadrs: =leftadrs+1;
  a: =destination; d: =0;
  if a=ownadrs then d: =OWN;
  while d=0 do
    if a=leftadrs
      then d: =LEFT
      else if a=rightadrs
           then d: =RIGHT
           else if a<ownadrs
                then d: =TOP
                else a: =a div 2;
  direction: =d
end.
```

Where ownadrs, leftadrs, and rightadrs are the processor numbers of the own, the left, and the right processors, respectively. OWN denotes the direction towards the own processor, that is, the message is to be received by the routing processor.

The SENDL and SENDR subroutines send informations concerning the initial conditions to the left and the right processors. The SOR subroutine is called in every computation step in order to compute u_{ij}. When data transmission is required it calls POST subroutine which prepares a message by attaching a destination address to the transmitting data and puts it in the outgoing mailbox. It is the role of the SENDD subroutine to read the destination address in the message and to move the message into the incoming mailbox of another processor in the proper direction. SENDA subroutine sends the result of computation to the processor at the root of the binary tree after the iteration converges. The output from the simulator is generated by OUTPUT subroutine. It includes information such as the average number of active processors, iteration steps of all blocks, total computation time, total routing overhead time, etc. The history of the status of all processors throughout the computation period is also obtained. Figure 12 shows a part of the simulator output, where the processor status is represented with the following codes.

W: receiving initial data
H: preparing initial data
D: sending initial data
F: computing starting values
C: computing u_{ij}
S: sending data
R: receiving data
A: sending result of computation
Z: receiving result of computation
.: waiting

A processor with a status other than waiting is considered active.

Some of the results obtained with the simulator are illustrated in Figs. 13, 14, and 15. Figure 13 shows the changes in the number of active processors as the computation steps proceed where the number of total processors is 30. As the computation starts all the processors become active quickly because the initial condition is broadcasted to all the processors. After this period, the iteration starts from the first block.

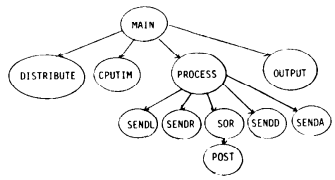

Fig. 11 Structure of the Simulator for CORAL System.

```
TIME ACTIV P1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
 0.0    1   H . . . . . . . . . . . . . . . .
 3.8    2   D W . . . . . . . . . . . . . . .
 6.4    2   H H . . . . . . . . . . . . . . .
 6.7    3   D H W . . . . . . . . . . . . . .
 9.2    4   D D W W . . . . . . . . . . . . .
 9.3    4   F D H W . . . . . . . . . . . . .
11.8    4   F H H H . . . . . . . . . . . . .
12.1    5   F H D H . W . . . . . . . . . . .
12.2    6   F D D H W W . . . . . . . . . . .
14.6    7   F D D D W W . W . . . . . . . . .
14.7    7   F D H D W H . W . . . . . . . . .
14.8    7   F F H D H H . W . . . . . . . . .
15.1    8   F F D D H H W W . . . . . . . . .
17.2    8   F F D H H H W H . . . . . . . . .
17.5    9   F F D H H D W H . . . W . . . . .
17.6   11   F F D D D D W H W W . W . . . . .
17.7   11   F F F D D D H H W W . W . . . . .
20.0   12   F F F D D D H D W W . W . . . W .
20.1   12   F F F D D H H D W W . H . . . W .
20.2   12   F F F F H H H D H H . H . . . W .
20.5   14   F F F F H D D D H H . H W W . W .
20.6   15   F F F F D D D D H H W H W W . W .
22.6   15   F F F F D D D H H H W H W W . H .

1580.8   18   . C . C . . . C . C . . . . . C C
1584.5   19   R S . C . . . C . C . . . . . C C
1585.7   21   S S R C R . . C . C . . . . . C C
1586.8   21   S S R C R . . C . C . . . . . C R
1586.9   20   . . C C S . . C . C R . . . . C R
1588.0   19   . . C C S . . R . C R . . . . C S
1588.1   19   . . C C . . . R . C S . . . . C S
1589.2   20   . . C C . . . C . C S . . . . C R
```

Fig. 12 History of Processor Status obtained by the Simulator.

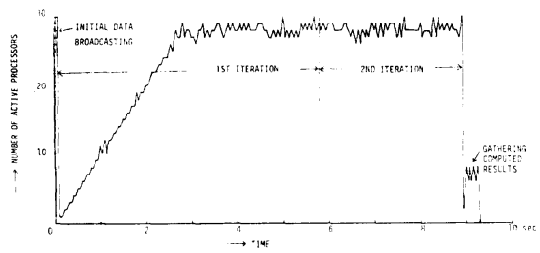

Fig. 13 Number of Active Processors versus Time for CORAL System with 30 Processors.

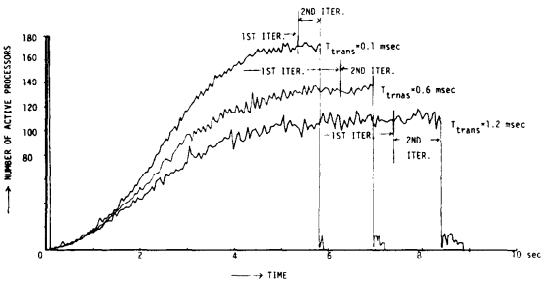

Fig. 14 Number of Active Processors versus Time for CORAL System with 180 Processors with Different Data Transmission Times.

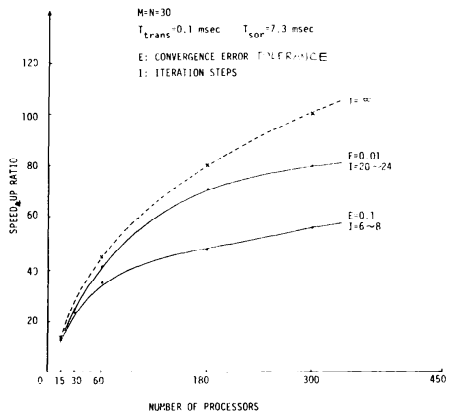

Fig. 15 Speed-up Ratios versus Number of Processors of CORAL system for Different Covergence Errors.

The number of active processors increases rather slowly until the first iteration step is completed at the last block. This is because the large number of computation steps, as represented by S in the previous chapter, are required in this initial iteration. After the first iteration is finished, the iterations proceed faster. In this figure, the computation is assumed to have converged by two iterations. A short period after the steep descent in which a few processors are still active is caused by the processors gathering results of computation from their lower processors and sending them to the upper ones.

Fig. 14 shows similar results with a total of 180 processors, and where the data transmission time is varied from 1.2 to 0.1 milisec per data. As the data transmission time is decreased, the number of active processors increases which results in shorter computation time.

In Fig. 15 the speed-up ratios, which are the ratio of computation speed with multiple of processors to that with only one processor, for different numbers of processors and also of the convergence error tolerances are indicated. As the iteration steps increase the contribution from initial idling time due to S decreases and the speed-up ratio is improved. As the convergence error tolerance is set much smaller in practical computation, the iteration steps may be several hundreds. Therefore the speed-up ratio versus number of processors in practical computation may be approximated by the dotted line in the figure, which is the speed-up ratio per one iteration in later iteration steps. From this result we can conclude that a speed-up ratio of 100 is obtainable with 300 processors of CORAL system.

6. Conclusions

The problems which arise when increasing the efficiency of parallel computation in solving the Laplace equation are discussed. Among them the strategy of partitioning mesh-points and the strategy of allocating the blocks to the processors are presented. Finally a software simulator of a binary tree processor network CORAL is developed and the parallel computation in solving the two-dimensional Laplace equation is simulated. In this simulation the transient behavior of processors and the effect of data transmission time are observed. Although the increase of the speed-up ratio tends to slow down as the number of processors increases, it was confirmed that a speed-up ratio of 100 is attainable with 300 processors. The computation time of the processors, the data transmission time, and the processor sysnchronizations are taken into account in our simulator. However, the random variations in the computation time and the overhead of the operating system were not sidered. The effects of these factors are being estimated with the use of a recently developed CORAL prototype consisting of 15 processors.[8] The estimation of the efficiency of parallel computation with more than one thousand processors would be very important but at present it is impossible with our simulator. This problem will be our target for future research.

The authors are grateful to an anonymous refree for numerous comments that helped improve the article.

References

1. TAKAHASHI, Y., WAKABAYASHI, N. and NOBUTOMO, Y. A Binary Tree Multiprocessor CORAL, *Journal. Information Processing*, 3, 4 (Feb. 1981), 230–237.
2. HOROWITZ, E. and ZORAT, A. The Binary Tree as an Interconnection Network: Application to Multiprocessor Systems and VLSI, *IEEE Trans.*, C-30, 4 (Apr. 1981), 247–253.
3. JONES, K. A. and SCHWARTZ, P. Experience Using Multiprocessor Systems—A Status Report, *Computing Surveys*, 12, 2 (June 1980), 121–165.
4. Personal communication from Yoshimura, S.

5. YOKOYAMA, M. Convergence of Parallel Processing by Iterative Method, Trans. *IPSJ*, **22**, 6 (Nov. 1981), 535–540, (Japanese).
6. TOGAWA, H. Numerical Computation of Matrices, p. 64, Baifukan, Tokyo (1971) (Japanese).
7. TAKAHASHI, Y., WAKABAYASI, N. and NOBUTOMO, Y. A Binary Tree Multiprocessor CORAL, *Tech. Rep. IECEJ*, *EC*79–60 (Jan. 1980).
8. NOBUTOMO, Y. and TAKAHASHI, Y. Performance Evaluation of CORAL Prototype: An Experimental Binary Tree Parallel Processor, *Tech. Rep. IPSJ on Comp.* Arch. 44–1 (Feb. 1982) (Japanese)

(Received January 18, 1982: revised May 31, 1982)