TR-94-005

Efficient Algorithms for Data Distribution on Distributed Memory Multicomputers

PeiZong Lee

Efficient Algorithms for Data Distribution on Distributed Memory Multicomputers[1]

PeiZong Lee
Institute of Information Science, Academia Sinica
Taipei, Taiwan, R.O.C.
Internet: leepe@iis.sinica.edu.tw
TEL: +886 (2) 788-3799
FAX: +886 (2) 782-4814

Abstract

Data distribution has been one of the most important research topics in parallelizing compilers for distributed memory parallel computers. Previous research works either only allow programmers explicitly to specify the data distribution using language extensions and then can generate all the communication instructions by compilers, or use compiler techniques to automatically determine a static data distribution scheme for the whole target program. In this paper, we show that data re-distribution is necessary for executing a sequence of Do-loops if the communication cost due to perform this sequence of Do-loops is larger than a threshold value. Based on this observation, we propose efficient algorithms which can determine effective data distribution schema for executing a sequence of Do-loops with a general structure. Our result contributes towards automatic compilation of sequential programs to message-passing version programs running on distributed memory parallel computers. Experimental studies on a 32-node nCUBE-2 computer are also presented.

Keywords: component alignment, data distribution, distributed memory computer, dynamic programming algorithm for data distribution, parallelizing compiler.

*** A preliminary version of this technical report is accepted to be presented at the **IEEE International Conf. on Parallel and Distributed Systems**, Hsinchu, Taiwan, Dec. 19–21, 1994.

[1]This work was partially supported by the NSC under Grants NSC 82-0408-E-001-016 and NSC 84-2213-E-001-003.

1 Introduction

This paper is concerned with designing efficient algorithms for data distribution on distributed memory parallel computers. We formulate the problem: when given a sequence of s Do-loops with a general structure, we want to determine effective data distribution schema for executing this sequence of Do-loops. This problem can be classified into cases in three levels as shown in Fig. 1: (a) a sequence of s Do-loops; (b) a sequence of s Do-loops which are enclosed by an iterative loop; and (c) a sequence of s Do-loops with a general structure, and among them, some consecutive Do-loops may be enclosed by iterative loops, which, again, with adjacent Do-loops, may be enclosed by other iterative loops, and so on.

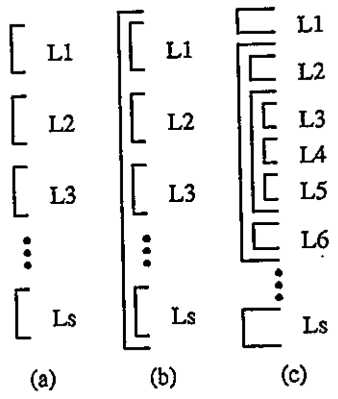


Figure 1: (a) a sequence of s Do-loops; (b) a sequence of s Do-loops which are enclosed by an iterative loop; and (c) a sequence of s Do-loops with a general structure.

This problem is quite important, because many scientific programs are comprised of a sequence of Do-loops or iterative loops, which may contain other sequences of Do-loops with a general structure. Thus, if a compiler adopts a naive data distribution scheme, this may result in excessive communication overhead when these programs are run on distributed memory parallel computers.

Data distribution has been one of the most important research topics in parallelizing compilers for distributed memory parallel computers. In theoretics, Mace first showed that a class of dynamic data layout problems for interleaved memory machines are NP-complete [24]. Anderson and Lam then presented another formulation of the dynamic data layout problem to be NP-hard [1]. Kremer also identified that the problem of dynamic data remapping (the inter-phase data layout problem) is NP-complete [18]. Li and Chen, in addition, proved that the problem of determining an optimal static alignment between the dimensions of distinct arrays is NP-complete [23].

Thus, in practice, previous parallelizing compiler research emphasizes on allowing programmers to specify the data distribution using language extensions and these can then generate all the communication instructions by compilers [2, 25, 32]. It is also possible to use compiler techniques to automatically determine sequential programs' data distribution on distributed memory systems. Li and Chen [23], Gupta and Banerjee [9] formulated the component alignment problem from the whole source program and used it to determine data distribution. As mentioned in the previous paragraph, the component alignment problem is NP-complete; nonetheless, Li and Chen have proposed an efficient heuristic algorithm based on applying the optimal matching procedure to a bipartite graph constructed from the nodes corresponding to components (dimensions) of two data arrays [23]. However, the fixed data distribution schema they derived may result in a larger communication overhead. Unlike them, in this paper we will deal with each Do-loop independently. Data distribution schema between two Do-loops may be different and may require some data communication between them. We will derive efficient dynamic programming algorithms which can determine whether data re-distribution is necessary.

In addition, there are other research works related to the compilation of programs on distributed-memory computers. Knobe, Lukas, Steele, and Natarajan provided algorithms for automatic alignment of arrays on SIMD machines [16, 17]. Chapman, Fahringer, Mehrotra, Moritsch, and Zima adopted Li and Chen's component alignment algorithm [23] for handling distributed data in Vienna Fortran [3]; in addition, they used language extension for handling dynamic data distribution [4]. Kennedy, Kremer, Meller-Crummey, and Carle proposed an automatic data layout strategy which is implemented in their D programming tools. For their strategy, they first explored several possible data layouts for each program phase, and they then defined communication cost between candidate data layouts of adjacent phases. The problem of finding dynamic data layouts for the entire program is thus reduced to a single-source shortest paths problem [20]. Kremer also developed techniques for using 0-1 integer programming for automatic data layout in the inter-phase data layout problem [19]. Other papers, which addressed the problem of determining initial data distributions or distributions for temporaries, include [5] [6] [29].

Furthermore, Hovland and Ni determined data distribution using augmented data access descriptors [11]. Kalns, Xu, and Ni suggested a cost model for determining a small set of appropriate data distribution patterns among many possible choices [15]. Kalns and Ni proposed techniques for logical processor mapping that minimizes the total amount of data that must be communicated among processors [14]. Chen and Sheu [7], Huang and Sadayappan [12], Ramanujam and Sadayappan [26, 27], and Wolf and Lam [30, 31] determined data distribution and/or degree of parallelism based on the hyperplane method. In addition, Gong, Gupta, and Melhem [8] and Hudak and Abraham [13] developed compile-time techniques for optimizing communication overhead.

With this brief introduction now completed, the rest of this paper is organized as follows. In Section 2, we review some background of distributed memory parallel computers and introduce a primitive dynamic programming algorithm for data distribution. In Section 3, we introduce another dynamic programming algorithm which is especially suitable for determining data distribution of a sequence of Do-loops enclosed by an iterative loop. In Section 4, we propose three efficient algorithms to deal with the three cases mentioned in Fig. 1. In Section 5, we present experimental studies on a 32-node nCUBE-2 computer. Finally, some concluding remarks are given in Section 6.

2 Background

In this paper, we are concerned with distributed memory systems. The abstract target machine we adopt is a q-D grid of $N_1 \times N_2 \times \cdots \times N_q$ processors, where D stands for dimensional and q is less than or equal to the deepest level of the Do-loop program. A processor on the q-D grid is represented by the tuple $(p_1, p_2, \ldots, p_q)$, where $0 \leq p_i \leq N_i - 1$ for $1 \leq i \leq q$. Such a topology can be easily embedded into almost all distributed memory machines, including configurably massively parallel computers. For example, the q-D grid can be embedded into a hypercube computer using a binary reflected Gray code.

The parallel program generated from a sequential program for a grid corresponds to the SPMD (Single Program Multiple Data) model, in which each processor executes the same program but operates on distinct data items [9, 10, 22, 28]. More precisely, a source program in general has sequential parts (which must be executed sequentially) and concurrent parts (which can be executed concurrently). Each processor will execute the sequential parts individually; while all processors will execute the concurrent parts altogether by using message-passing communication primitives. In practice, scalar variables and small data arrays used in the program are replicated on all processors in order to reduce communication costs; while large data arrays are partitioned and distributed among processors [21]. Gupta and Banerjee [9] proposed a data distribution function, which can map each array dimension to a unique dimension of the processor grid. In addition, the data distribution function can specify the method of partition to be "contiguous" or "cyclic" or "contiguous-cyclic".

2.1 A Dynamic Programming Algorithm for Data Distribution

Previously, researchers formulated the component alignment problem from the whole source program and used it to determine data distribution [9, 23]. When given a program, they first constructed a component affinity graph from the source program. It is a directed, and weighted graph, whose nodes represent dimensions (components) of arrays and whose edges specify affinity relations between nodes. Two dimensions of arrays are said to have an affinity relation if two subscripts of these two dimensions are affine functions of the same (single) index of a Do-loop. The weight with an edge is equal to the communication cost and is necessary if two dimensions of arrays are distributed along different dimensions of the processor grid. The direction of an edge specifies the direction of the data communication according to the "owner computes" rule.

The component alignment problem is defined as partitioning the node set of the component affinity graph into q disjointed subsets (q is the dimension of the abstract target grid and q may be larger than the dimension of the physical target grid) so that the total weight of edges across nodes in different subsets is minimized, with the restriction that no two nodes corresponding to the same array are in the same subset. These q disjointed subsets will be used to determine data distributions for all data arrays. Although the component alignment problem is NP-complete, Li and Chen have proposed an efficient heuristic algorithm based on applying the optimal matching procedure to a bipartite graph constructed from the nodes corresponding to components (dimensions) of two data arrays [23].

However, the fixed data distribution schema derived by previous researchers may result in a larger communication overhead. Unlike them, we will deal with each Do-loop independently. Data distribution schema between two Do-loops may be different and may require some data communication between them. In the following, we introduce a primitive dynamic programming algorithm to determine whether the data re-distribution is necessary.

Suppose that a program contains s Do-loops: L_1, L_2, ..., L_s in sequence. Let $M_{i,j}$ be the cost of computing the sequence of Do-loops L_i, L_{i+1}, ..., L_{i+j-1} using the component-alignment algorithm, and $P_{i,j}$ be the distribution scheme, for $1 \leq i \leq s$ and $1 \leq j \leq s-i+1$. Define $T_{i,j}$ to be the cost of computing the sequence of Do-loops L_1, L_2, ..., L_{i+j-1} with the restriction that it uses the distribution scheme $P_{i,j}$ to compute Do-loops L_i, L_{i+1}, ..., L_{i+j-1}. Thus, the final data distribution scheme after computing $T_{i,j}$ is $P_{i,j}$. Initially, $T_{1,j}$ is equal to $M_{1,j}$.

Algorithm 1: A dynamic programming algorithm for computing the cost of data distribution schema of executing a sequence of s Do-loops on distributed memory computers is presented.

Input: $M_{i,j}$, $P_{i,j}$, and $T_{1,i}$ $(= M_{1,i})$, where $1 \leq i \leq s$ and $1 \leq j \leq s-i+1$.

Output: The cost of executing s Do-loops on distributed memory computers.

```
1.  for $i := 2$ to $s$ do
2.      for $j := 1$ to $s-i+1$ do
3.          $T_{i,j} := \mathrm{MIN}_{1 \leq k < i}\{T_{i-k,k} + M_{i,j} + cost(P_{i-k,k}, P_{i,j})\}$ ;
4.  end_for end_for
5.  $Minimum\_Cost := \mathrm{MIN}_{1 \leq k \leq s}\{T_{s-k+1,k} + loop\_carried\_dependence(T_{s-k+1,k})\}$ .
```

Note: $cost(P_{i-k,k}, P_{i,j})$ returns the communication cost of changing data layouts from $P_{i-k,k}$ to $P_{i,j}$. $loop\_carried\_dependence(T_{s-k+1,k})$ returns the communication cost incurred by the loop-carried dependence, if a sequence of distribution schema are used for computing $T_{s-k+1,k}$. For example, if a sequence of distribution schema P_{λ_1,μ_1}, P_{λ_2,μ_2}, ..., and $P_{s-k+1,k}$ are used for computing $T_{s-k+1,k}$, then $loop\_carried\_dependence(T_{s-k+1,k})$ returns the communication cost of changing data layouts from $P_{s-k+1,k}$ to P_{λ_1,μ_1}.

Algorithm 1 can be regarded as finding a single-source shortest paths in a weighted graph. In this weighted graph, there are two virtual nodes and $\frac{s(s+1)}{2}$ physical nodes. Two virtual nodes include one source and one sink. $\frac{s(s+1)}{2}$ physical nodes $n_{i,j}$ are numbered by i and j, where $1 \leq i \leq s$ and $1 \leq j \leq s-i+1$. Nodes' weight, edges, and edges' weight of this graph are defined as follows. (1) The weight of two virtual nodes each is zero. (2) The weight of node $n_{i,j}$ is $M_{i,j}$. (3) The source has s edges connected to nodes $n_{1,j}$, and the weight of these edges each is zero, for $1 \leq j \leq s$, respectively. (4) The sink, which also has s edges, is connected by nodes $n_{i,(s-i+1)}$, and the weight of these edges each is also zero, for $1 \leq i \leq s$, respectively. And, (5) node $n_{i,j}$ has $s-(i+j)+1$ edges connected to nodes $n_{(i+j),k}$, and the weight of these edges each is $cost(P_{i,j}, P_{(i+j),k})$, for $(i+j) \leq s$ and $1 \leq k \leq s-(i+j)+1$, respectively. Then, Algorithm 1 is equivalent to find shortest paths from source to sink such that the sum of nodes' weight and edges' weight in each of these paths are minimum. Fig. 2 shows the corresponding single-source shortest paths problem for $s = 5$.

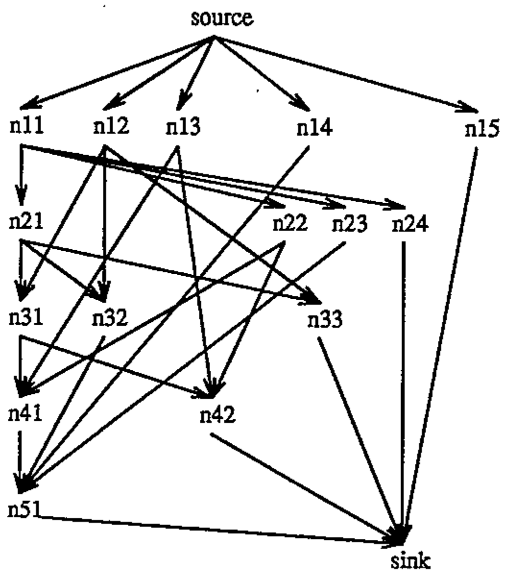


Figure 2: The corresponding single-source shortest paths problem for $s = 5$.

The data distribution scheme obtained from Algorithm 1 is at least as good as any static data distribution scheme, because the cost of any static data distribution scheme is equal to $T_{1,s}$. We now briefly analyze Algorithm 1. The time complexity of this dynamic programming algorithm is $O(s^3)$. However, before applying this dynamic programming algorithm, we need to compute $s(s+1)/2$ component alignment problems for the consecutive Do-loops $L_i, L_{i+1}, \ldots, L_{i+j-1}$, where $1 \leq i,\ j \leq i+j-1 \leq s$.

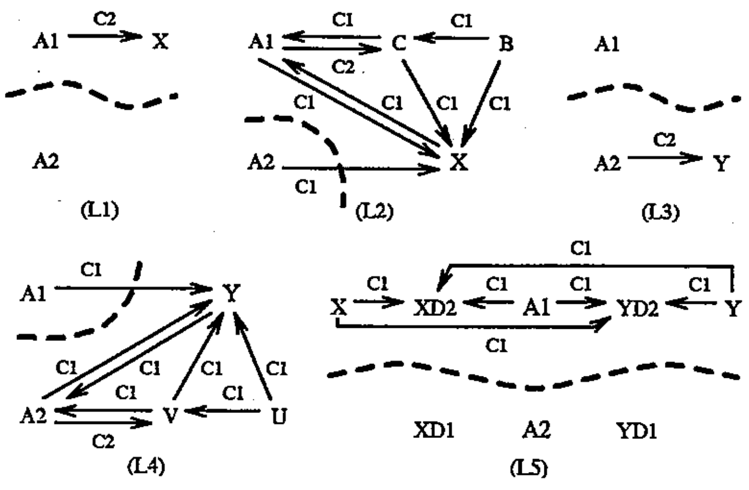
2.2 A Sample Example

In the following, we use a complete example to illustrate how to apply the above dynamic programming algorithm for determining data distribution. Suppose that the problem size is m and the number of processing elements used is N. Consider the following program which will be executed on a linear processor array.

```
 1  DO 42 n = 1, OUT_ITERATION
 2    DO 6 i = 1, m
 3      X(i) = 0.0
 4      DO 6 j = 1, m
 5        X(i) = X(i) + A(i,j)
 6    CONTINUE
 7    DO 19 k = 1, MAX_ITERATION
 8      DO 12 i = 1, m
 9        C(i) = 0.0
10        DO 12 j = 1, m
11          C(i) = C(i) + A(i,j) * B(i)
12      CONTINUE
13      DO 16 i = 1, m
14        DO 16 j = 1, m
15          A(i,j) = (A(i,j) + C(i) - X(i)) / (m * m)
16      CONTINUE
17      DO 19 i = 1, m
18        X(i) = X(i) + (B(i) - C(i)) / A(i,i)
19    CONTINUE
20    DO 24 i = 1, m
21      Y(i) = 0.0
22      DO 24 j = 1, m
23        Y(i) = Y(i) + A(j,i)
24    CONTINUE
25    DO 37 k = 1, MAX_ITERATION
26      DO 30 i = 1, m
27        V(i) = 0.0
28        DO 30 j = 1, m
29          V(i) = V(i) + A(j,i) * U(i)
30      CONTINUE
31      DO 34 i = 1, m
32        DO 34 j = 1, m
33          A(j,i) = (A(j,i) + V(i) - Y(i)) / (m * m)
34      CONTINUE
35      DO 37 i = 1, m
36        Y(i) = Y(i) + (U(i) - V(i)) / A(i,i)
37    CONTINUE
38    DO 41 i = 1, m
39      XD(n,i) = X(i) + A(i,1) * Y(i)
40      YD(n,i) = Y(i) + A(i,1) * X(i)
41    CONTINUE
42  CONTINUE
```

We let line 2 to line 6 be loop L_1; line 7 to line 19 be loop L_2; line 20 to line 24 be loop L_3; line 25 to line 37 be loop L_4; and line 38 to line 41 be loop L_5. The component affinity graphs and the corresponding component alignment of these five loops are shown in Fig. 3. The weight of an edge is defined as follows. Because the topology of our target machine is a linear array, if the corresponding array's dimensionality on the tail of an edge is 1, then the weight of that edge is defined to be ManyToManyMulticast(m/N, {N PEs}). If the corresponding array's dimensionality on the tail of an edge is 2, then the weight of that edge is defined to be ManyToManyMulticast(m^2/N, {N PEs}). ManyToManyMulticast(s, {Z PEs}) has the purpose of replicating data of size s each from a set of Z processors to each processing element of these Z processors. The communication cost of ManyToManyMulticast(s, {Z PEs}) is roughly equal to $s * Z *$ Transfer(1), where Transfer(s') means to send a message of size s' from a processor to another processor.

We now analyze the approximate computation time and communication time of these five loops



C1 = ManyToManyMulticast(m/N, {N PEs}), or m * Transfer(1)
C2 = ManyToManyMulticast(m² /N, {N PEs}), or m² * Transfer(1)

Figure 3: The component affinity graphs and the corresponding component alignment for L_1, L_2, L_3, L_4, and L_5, respectively. Transfer(s) means to send a message of size s from a processor to another processor. ManyToManyMulticast(s, {Z PEs}) has the purpose of replicating data of size s each from a set of Z processors on the specified grid dimension(s) to themselves.

depending on whether matrix A is distributed row by row or distributed column by column. Suppose that the average time of computing a floating point operation is t_f and the average time of transferring a word is t_c. Then, Table 1 shows the approximate computation time and the communication time of these five loops, where K is equal to the constant MAX_ITERATION and

$$
\begin{aligned}
C_a &= (m^2/N) * t_f \\
C_b &= K * ((5m^2 + 3m)/N) * t_f \\
C_c &= (4m/N) * t_f \\
C_d &= m * (\log N) * t_c \\
C_e &= K * (m * (\log N) + m) * t_c \\
C_f &= m * t_c.
\end{aligned}
$$

Suppose that the cost of performing a matrix transpose operation, C_T, is $(m^2/2N) * (\log N) * t_c$; in addition, C_T is very small in comparison with C_e $(= K*(m*(\log N)+m)*t_c)$, and $C_f < C_d < C_T < C_e$. Then, the communication cost $M'_{i,j}$ required for computing different sequences of consecutive Do-loops by the component-alignment algorithm is shown in Table 2. Note that, $M_{i,j}$ is the total execution time of computing the sequence of Do-loops L_i, L_{i+1}, ..., L_{i+j-1}. Since the computation of this parallel

	matrix A is distributed row by row		matrix A is distributed column by column	
	computation time	communication time	computation time	communication time
L_1	C_a	0	C_a	C_d
L_2	C_b	0	C_b	C_e
L_3	C_a	C_d	C_a	0
L_4	C_b	C_e	C_b	0
L_5	C_c	0	C_c	C_f

Table 1: Computation time and communication time of five loops.

algorithm is load-balanced, we only need to consider the communication time $M'_{i,j}$ here, because the computation time for all different data distribution schema are the same. For simplicity, we use $M_{i,j}$ to represent $M'_{i,j}$. Similarly, $T_{i,j}$ will only represent the total communication time of computing the sequence of Do-loops L_1, L_2, ..., L_{i+j-1} in this example. Thus, the expectant communication cost required for computing the sequence of s (= 5) Do-loops can be solved by Algorithm 1 as shown in Table 2.

From Table 2, we conclude that it requires in total $2C_T + C_f$ communication time for executing an iteration of the outmost loop. In addition, there is one candidate sequence of data distribution schema for an outmost iteration. That is, first, data layouts between L_1 and L_2 are not changed; next, a matrix transpose operation for matrix A is necessary before executing L_3; then, data layouts between L_3, L_4, and L_5 are not changed, although it requires transferring m words in L_5; finally, another matrix transpose operation for matrix A^T is necessary before the next iteration.

In the following, we list data distribution functions for each data array. As the iteration space is rectangular, for the sake of requiring load balance, the distribution functions for all array dimensions are determined to be contiguous. Suppose that the number of processing elements in the linear processor array is N and $1 \leq i, j \leq m$.

For Do-loops L_1 and L_2:

$$f_A(i,j) = (\lfloor \frac{i-1}{m/N} \rfloor); \quad f_X(i) = f_B(i) = f_C(i) = (\lfloor \frac{i-1}{m/N} \rfloor).$$

M_{ij}	j = 1	j = 2	j = 3	j = 4	j = 5
i = 1	0	0	C_d	$C_d + C_e$	$C_d + C_e$
i = 2	0	C_d	C_e	$C_e + C_f$	
i = 3	0	0	C_f		
i = 4	0	C_f			
i = 5	0				

$$
\begin{aligned}
&T_{11} = M_{11} = 0\\
&T_{12} = M_{12} = 0\\
&T_{13} = M_{13} = C_d\\
&T_{14} = M_{14} = C_d + C_e\\
&T_{15} = M_{15} = C_d + C_e\\
&T_{21} = T_{11} + M_{21} + (cost(P_{11}, P_{21}) = 0) = 0\\
&T_{22} = T_{11} + M_{22} + (cost(P_{11}, P_{22}) = 0) = C_d\\
&T_{23} = T_{11} + M_{23} + (cost(P_{11}, P_{23}) = C_T) = C_e + C_T\\
&T_{24} = T_{11} + M_{24} + (cost(P_{11}, P_{24}) = C_T) = C_e + C_f + C_T\\
&T_{31} = \text{MIN}\{T_{21} + M_{31} + (cost(P_{21}, P_{31}) = C_T), T_{12} + M_{31} + (cost(P_{12}, P_{31}) = C_T)\} = C_T\\
&T_{32} = \text{MIN}\{T_{21} + M_{32} + (cost(P_{21}, P_{32}) = C_T), T_{12} + M_{32} + (cost(P_{12}, P_{32}) = C_T)\} = C_T\\
&T_{33} = \text{MIN}\{T_{21} + M_{33} + (cost(P_{21}, P_{33}) = C_T), T_{12} + M_{33} + (cost(P_{12}, P_{33}) = C_T)\} = C_f + C_T\\
&T_{41} = \text{MIN}\{T_{31} + M_{41} + (cost(P_{31}, P_{41}) = 0), T_{22} + M_{41} + (cost(P_{22}, P_{41}) = C_T),\\
&\qquad\qquad T_{13} + M_{41} + (cost(P_{13}, P_{41}) = C_T)\} = C_T\\
&T_{42} = \text{MIN}\{T_{31} + M_{42} + (cost(P_{31}, P_{42}) = 0), T_{22} + M_{42} + (cost(P_{22}, P_{42}) = C_T),\\
&\qquad\qquad T_{13} + M_{42} + (cost(P_{13}, P_{42}) = C_T)\} = C_f + C_T\\
&T_{51} = \text{MIN}\{T_{41} + M_{51} + (cost(P_{41}, P_{51}) = C_f), T_{32} + M_{51} + (cost(P_{32}, P_{51}) = C_f),\\
&\qquad\qquad T_{23} + M_{51} + (cost(P_{23}, P_{51}) = C_f), T_{14} + M_{51} + (cost(P_{14}, P_{51}) = 0)\} = C_f + C_T.
\end{aligned}
$$

T_{ij}	j = 1	j = 2	j = 3	j = 4	j = 5
i = 1	0	0	C_d	$C_d + C_e$	$C_d + C_e$
i = 2	0	C_d	$C_e + C_T$	$C_e + C_f + C_T$	
i = 3	C_T	C_T	$C_f + C_T$		
i = 4	C_T	$C_f + C_T$			
i = 5	$C_f + C_T$				

$$
\begin{aligned}
Minimum\_Cost &= \text{MIN}_{1 \le k \le 5}\{T_{5-k+1,\,k} + loop\_carried\_dependence(T_{5-k+1,\,k})\}\\
&= \text{MIN}\{T_{51} + C_T, T_{42} + C_T, T_{33} + C_T, T_{24} + C_T, T_{15} + 0\}\\
&= 2C_T + C_f.
\end{aligned}
$$

Table 2: Apply Algorithm 1 to the sample program. $M_{i,j}$: the communication cost required for computing a sequence of consecutive Do-loops (L_i, L_{i+1}, $\cdots$, L_{i+j-1}) using the component-alignment algorithm. $T_{i,j}$: the communication cost required for computing a sequence of consecutive Do-loops (L_1, L_2, $\cdots$, L_{i+j-1}).

For Do-loops L_3, L_4, and L_5:

$$f_A(i,j) = f_{XD}(i,j) = f_{YD}(i,j) = (\lfloor \frac{j-1}{m/N} \rfloor); \quad f_X(i) = f_Y(i) = f_U(i) = f_V(i) = (\lfloor \frac{i-1}{m/N} \rfloor).$$

Note: the data distribution function $f_X(i) = p$ means that the entry i of the one-dimensional data array X, $X[i]$, is stored in processing element p. The data distribution function $f_A(i,j) = p$ means that the entry (i,j) of the two-dimensional data matrix A, $A[i,j]$, is stored in processing element p.

2.3 More Details about Data Distribution

This subsection maybe appears immediately after introducing Algorithm 1; however, we think that it is more suitable for presenting the sample example first. In this subsection, we describe the data distribution for each data array in $P_{i,j}$ detailedly.

As readers can see from Fig. 3, the component affinity graph and the corresponding component alignment for each Do-loop only deal with data arrays which are used in that Do-loop. Therefore, if a data array is used in L_i, L_{i+1}, ..., L_{i+j-1}, then its data distribution can be determined from the component alignment algorithm and is defined in $P_{i,j}$. However, if a data array is not used in L_i, L_{i+1}, ..., L_{i+j-1}, then, after applying the component alignment algorithm, its data distribution in $P_{i,j}$ is not defined. In the following, we use a heuristic method to assign a data distribution in $P_{i,j}$ for each data array, if this data array is not used in L_i, L_{i+1}, ..., L_{i+j-1}. This heuristic method includes two phases.

The first phase is applied during constructing the $(P_{i,j})$-table. Suppose that a data array is not used in the first $e-1$ Do-loops and it is used in the e-th Do-loop, for $e > 1$. First, we implicitly assume that its data distribution during computing the first $e-1$ Do-loops is the same as the one defined in the e-th Do-loop. Therefore, if $i+j-1 < e$, then the data distribution in $P_{i,j}$ for this data array is defined to be the same as the one defined in $P_{e,1}$. Second, if this data array is not used in a Do-loop, we also implicitly assume that its data distribution is not changed during the computation. That is, its data distribution is still the same as the one defined in the previous Do-loop. Therefore, if $i > e$ and this data array is not used in L_i, L_{i+1}, ..., L_{i+j-1}, then the data distribution in $P_{i,j}$ for this data array is defined to be the same as the one defined in $P_{(i-1),1}$. For instance, suppose that in the first three Do-loops, a data array is only used in the second Do-loop. Then its data distribution during computing the first Do-loop and the third Do-loop is the same as the one defined in the second Do-loop.

The second phase is applied after performing Algorithm 1. After performing Algorithm 1, we have found a sequence of distribution schema P_{λ_1,μ_1}, P_{λ_2,μ_2}, ..., P_{λ_ξ,μ_ξ}, for computing a sequence of s Do-loops. Suppose that a data array is first used in L_{λ_f}, L_{λ_f+1}, ..., $L_{\lambda_f+\mu_f-1}$, then its data distribution in P_{λ_f,μ_f} is determined from the component alignment algorithm. First, for $i < f$, this data array is not used in L_{λ_i}, L_{λ_i+1}, ..., $L_{\lambda_i+\mu_i-1}$, thus, we can let its data distribution in P_{λ_i,μ_i} be the same as the one defined in P_{λ_f,μ_f}. Second, for $i > f$, if this data array is not used in L_{λ_i}, L_{λ_i+1}, ..., $L_{\lambda_i+\mu_i-1}$, then its data distribution in P_{λ_i,μ_i} is defined to be the same as the one defined in $P_{\lambda_{i-1},\mu_{i-1}}$.

For instance, in the sample example, although data arrays B and C are not used in L_3, L_4, and L_5, their data distributions during computing L_3, L_4, and L_5 are the same as the ones defined in the first two Do-loops. Similarly, although data arrays XD, YD, Y, U, and V are not used in L_1 and L_2, their data distributions during computing L_1 and L_2 are the same as the ones defined in the last three Do-loops.

3 The Case When s Do-loops Are Enclosed by an Iterative Loop

Algorithm 1 works well when a program contains s Do-loops. However, if an iterative loop contains s Do-loops, we can improve the results. The fundamental rationale behind this improvement is that we find this sequence of s Do-loops appearing in a cyclic fashion. Therefore, every Do-loop can be treated as the first Do-loop. In the following, we introduce a new dynamic programming algorithm, which is based on Algorithm 1, for data distribution.

Suppose that an iterative loop contains s Do-loops: $L_1, L_2, \ldots, L_s$ in sequence. Let $M_{i,j}$ represent the total execution time of executing the sequence of j Do-loops (L_i, L_{i+1}, ..., L_{i+j-1}) if $i+j-1 \leq s$, or of j Do-loops (L_i, L_{i+1}, ..., L_s, L_1, ..., $L_{i+j-1-s}$) if $i+j-1 > s$, using a static data distribution scheme $P_{i,j}$, where $P_{i,j}$ can be obtained by applying the component-alignment algorithm, for $1 \leq i \leq s$ and $1 \leq j \leq s$. $T_{i,j}$ is defined the same as that of in Section 2.1. That is, $T_{i,j}$ is the cost of computing the sequence of Do-loops $L_1, L_2, \ldots, L_{i+j-1}$ with the restriction that it uses the distribution scheme $P_{i,j}$ to compute Do-loops L_i, L_{i+1}, ..., L_{i+j-1}. Thus, the final data distribution scheme after computing

$T_{i,j}$ is $P_{i,j}$. Initially, $T_{1,j}$ is equal to $M_{1,j}$.

Algorithm 2: A dynamic programming algorithm for computing the cost of data distribution schema of executing a sequence of s Do-loops enclosed by an iterative loop on distributed memory computers is presented.

Input: $M_{i,j}$, $P_{i,j}$, and $T_{1,i}$ $(= M_{1,i})$, where $1 \leq i \leq s$ and $1 \leq j \leq s$.

Output: The cost of executing an iterative loop which contains s Do-loops on distributed memory computers.

```
1.   for $m := 1$ to $s$ do
2.               /* Compute $s$ sequences of Do-loops. */
3.       for $i := 2$ to $s$ do
4.           for $j := 1$ to $s - i + 1$ do
5.               $T_{i,j} := \mathrm{MIN}_{1 \leq k < i}\{T_{i-k,k} + M_{i,j} + cost(P_{i-k,k}, P_{i,j})\}$ ;
6.       end_for end_for
7.       $Minimum\_Cost(m) := \mathrm{MIN}_{1 \leq k \leq s}\{T_{s-k+1,k} + loop\_carried\_dependence(T_{s-k+1,k})\}$ ;
8.               /* Shift $(M_{i,j})$-table. */
9.       for $j := 1$ to $s$ do
10.          copy $M_{1,j}$ to $TempM_j$ , copy $P_{1,j}$ to $TempP_j$ ;
11.      end_for
12.      for $i := 2$ to $s$ do
13.          for $j := 1$ to $s$ do
14.              copy $M_{i,j}$ to $M_{i-1,j}$ , copy $P_{i,j}$ to $P_{i-1,j}$ ;
15.      end_for end_for
16.      for $j := 1$ to $s$ do
17.          copy $TempM_j$ to $M_{s,j}$ , copy $TempP_j$ to $P_{s,j}$ , copy $M_{1,j}$ to $T_{1,j}$ ;
18.      end_for
19.  end_for
20.  $Final\_Minimum\_Cost := \mathrm{MIN}_{1 \leq m \leq s}\{Minimum\_Cost(m)\}$ .
```

The algorithm fragment from Line 3 to Line 7 is Algorithm 1; the algorithm fragment from Line 9 to Line 18 implements a data shift operation for $(M_{i,j})$-table. Fig. 4 shows a complete example by applying Algorithm 2 to the sample program mentioned in Section 2.2 for determining data distribution. We can see that it only requires in total $2C_T$ communication time for executing an iteration of the outmost loop. This result is better than that of Algorithm 1. From Fig. 4, we conclude that, in an outmost iteration, L_5, L_1, and L_2 use a data distribution scheme and L_3 and L_4 use another data distribution scheme. In the following, we list data distribution functions for each data array.

For Do-loops L_5, L_1, and L_2:

$$f_A(i,j) = (\lfloor \frac{i-1}{m/N} \rfloor); \quad f_{XD}(i,j) = f_{YD}(i,j) = (\lfloor \frac{j-1}{m/N} \rfloor);$$

$$f_X(i) = f_Y(i) = f_B(i) = f_C(i) = (\lfloor \frac{i-1}{m/N} \rfloor).$$

For Do-loops L_3 and L_4:

$$f_A(i,j) = (\lfloor \frac{j-1}{m/N} \rfloor); \quad f_Y(i) = f_U(i) = f_V(i) = (\lfloor \frac{i-1}{m/N} \rfloor).$$

Because Algorithm 1 is a special case of Algorithm 2, it is possible to obtain a better data distribution scheme by applying Algorithm 2 than by applying Algorithm 1. We now briefly analyze Algorithm 2. The time complexity of this dynamic programming algorithm is $O(s^4)$. However, before applying this dynamic programming algorithm, we need to compute s^2 component alignment problems for the consecutive j Do-loops L_i, L_{i+1}, ..., L_{i+j-1}, if $i+j-1 \leq s$; or for the consecutive j Do-loops L_i, L_{i+1}, ..., L_s, L_1, ..., $L_{i+j-1-s}$, if $i+j-1 > s$, where $1 \leq i,\ j \leq s$. In the next section, we will propose a new technique to improve Algorithm 1 and Algorithm 2.

4 New Efficient Algorithms for Data Distribution

We now analyze the behavior of the $(M_{i,j})$-table. It is clear that $M_{i,(\gamma_i+1)} \geq M_{i,\gamma_i}$, for $1 \leq \gamma_i < s-i+1$. We define $THRESHOLD$ to be a value that is equal to three times (or four times) that of the maximal communication cost between any two distribution schema. The reason why we define $THRESHOLD$ to be this value will be clear in Theorem 2 (or Theorem 3, respectively).

We want to show that if $M_{i,(\gamma_i+1)}$ is larger than $M_{i,\beta}$ plus $M_{(i+\beta),(\gamma_i-\beta+1)}$ and plus $THRESHOLD$, for some β where $1 \leq \beta < \gamma_i+1$, then it is better to use three distribution schema $P_{i,\beta}$, $P_{(i+\beta),(\gamma_i-\beta+1)}$, and $P_{(i+\gamma_i+1),(j-\gamma_i-1)}$ to compute the sequence of Do-loops L_i, L_{i+1}, ..., L_{i+j-1}, than to use only one distribution scheme $P_{i,j}$, for $\gamma_i+1 < j \leq s-i+1$. Therefore, we need not compute $M_{i,j}$. Based on

Define: $\alpha = C_f$ $\quad \delta = C_e$ $\quad \lambda = C_f + C_e$ $\quad \xi = C_T + C_e$ $\quad \tau = 2C_T$

$\beta = C_d$ $\quad \iota = C_f + C_d$ $\quad \mu = C_d + C_T$ $\quad \pi = C_f + C_d + C_T$ $\quad \phi = 2C_T + C_f$

$\gamma = C_T$ $\quad \kappa = C_f + C_T$ $\quad \nu = C_d + C_e$ $\quad \rho = C_f + C_T + C_e$

	i\j	1	2	3	4	5
L_1	1	0	0	β	ν	ν
L_2	2	0	β	δ	λ	ν
L_3	3	0	0	α	ι	ν
L_4	4	0	α	ι	δ	ν
L_5	5	0	0	0	β	ν

M_{ij} (before m = 1)

i\j	1	2	3	4	5
1	0	0	β	ν	ν
2	0	β	ξ	ρ	
3	γ	γ	κ		
4	γ	κ			
5	κ				

T_{ij} (after m = 1)

Minimum_Cost(1)

$$= \underset{0<k<6}{\mathrm{MIN}} \{ T_{5\text{-}k+1,k} + \text{loop\_carried\_dependence}(T_{5\text{-}k+1,k}) \}$$

$$= \mathrm{MIN} \{ T_{51} + C_T ,\ T_{42} + C_T ,\ T_{33} + C_T ,\ T_{24} + C_T ,\ T_{15} + 0 \}$$

$$= 2C_T + C_f$$

	i\j	1	2	3	4	5
L_2	1	0	β	δ	λ	ν
L_3	2	0	0	α	ι	ν
L_4	3	0	α	ι	δ	ν
L_5	4	0	0	0	β	ν
L_1	5	0	0	β	ν	ν

M_{ij} (before m = 2)

i\j	1	2	3	4	5
1	0	β	δ	λ	ν
2	γ	γ	κ	π	
3	γ	κ	π		
4	κ	τ			
5	ϕ				

T_{ij} (after m = 2)

Minimum_Cost(2)

$$= \underset{0<k<6}{\mathrm{MIN}} \{ T_{5\text{-}k+1,k} + \text{loop\_carried\_dependence}(T_{5\text{-}k+1,k}) \}$$

$$= \mathrm{MIN} \{ T_{51} + 0 ,\ T_{42} + 0 ,\ T_{33} + C_T ,\ T_{24} + C_T ,\ T_{15} + 0 \}$$

$$= 2C_T$$

	i\j	1	2	3	4	5
L_3	1	0	0	α	ι	ν
L_4	2	0	α	ι	δ	ν
L_5	3	0	0	0	β	ν
L_1	4	0	0	β	ν	ν
L_2	5	0	β	δ	λ	ν

M_{ij} (before m = 3)

i\j	1	2	3	4	5
1	0	0	α	ι	ν
2	0	α	ι	ξ	
3	α	γ	γ		
4	κ	κ			
5	γ				

T_{ij} (after m = 3)

Minimum_Cost(3)

$$= \underset{0<k<6}{\mathrm{MIN}} \{ T_{5\text{-}k+1,k} + \text{loop\_carried\_dependence}(T_{5\text{-}k+1,k}) \}$$

$$= \mathrm{MIN} \{ T_{51} + C_T ,\ T_{42} + C_T ,\ T_{33} + C_T ,\ T_{24} + C_T ,\ T_{15} + 0 \}$$

$$= 2C_T$$

	i\j	1	2	3	4	5
L_4	1	0	α	ι	δ	ν
L_5	2	0	0	0	β	ν
L_1	3	0	0	β	ν	ν
L_2	4	0	β	δ	λ	ν
L_3	5	0	0	α	ι	ν

M_{ij} (before m = 4)

i\j	1	2	3	4	5
1	0	α	ι	δ	ν
2	α	γ	γ	μ	
3	κ	κ	π		
4	γ	μ			
5	τ				

T_{ij} (after m = 4)

Minimum_Cost(4)

$$= \underset{0<k<6}{\mathrm{MIN}} \{ T_{5\text{-}k+1,k} + \text{loop\_carried\_dependence}(T_{5\text{-}k+1,k}) \}$$

$$= \mathrm{MIN} \{ T_{51} + 0 ,\ T_{42} + C_T ,\ T_{33} + C_T ,\ T_{24} + C_T ,\ T_{15} + 0 \}$$

$$= 2C_T$$

	i\j	1	2	3	4	5
L_5	1	0	0	0	β	ν
L_1	2	0	0	β	ν	ν
L_2	3	0	β	δ	λ	ν
L_3	4	0	0	α	ι	ν
L_4	5	0	α	ι	δ	ν

M_{ij} (before m = 5)

i\j	1	2	3	4	5
1	0	0	0	β	ν
2	0	0	β	ν	
3	0	β	ξ		
4	γ	γ			
5	γ				

T_{ij} (after m = 5)

Minimum_Cost(5)

$$= \underset{0<k<6}{\mathrm{MIN}} \{ T_{5\text{-}k+1,k} + \text{loop\_carried\_dependence}(T_{5\text{-}k+1,k}) \}$$

$$= \mathrm{MIN} \{ T_{51} + C_T ,\ T_{42} + C_T ,\ T_{33} + C_T ,\ T_{24} + 0 ,\ T_{15} + 0 \}$$

$$= 2C_T$$

$$\text{Final\_Minimum\_Cost} = \underset{0<m<6}{\mathrm{MIN}} \{ \text{Minimum\_Cost}(m) \} = 2C_T$$

Figure 4: Apply Algorithm 2 to the sample program.

this observation, we can show that $T_{i,(\gamma_i+1)} > T_{(i+\beta),(\gamma_i-\beta+1)}$ and $T_{i,j} > T_{(i+\gamma_i+1),(j-\gamma_i-1)}$. Therefore, we need not compute $T_{i,j}$, for $\gamma_i + 1 \leq j \leq s - i + 1$.

Theorem 1 : *If $M_{i,(\gamma_i+1)} > M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + THRESHOLD$, for some β where $1 \leq \beta < \gamma_i + 1$, then $M_{i,j} > M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + M_{(i+\gamma_i+1),(j-\gamma_i-1)} + cost(P_{i,\beta}, P_{(i+\beta),(\gamma_i-\beta+1)}) + cost(P_{(i+\beta),(\gamma_i-\beta+1)}, P_{(i+\gamma_i+1),(j-\gamma_i-1)})$, for $\gamma_i + 1 < j \leq s - i + 1$.*

Proof: Consider the computation of Do-loops L_i, L_{i+1}, ..., $L_{(i+\beta-1)}$, $L_{(i+\beta)}$, ..., $L_{(i+\gamma_i)}$, $L_{(i+\gamma_i+1)}$, ..., L_{i+j-1}. Let $cost\_\delta_1$ be the cost of computing the sequence of Do-loops L_i, L_{i+1}, ..., $L_{(i+\gamma_i)}$, using the distribution scheme $P_{i,j}$. Then, $cost\_\delta_1$ is at least as large as $M_{i,(\gamma_i+1)}$. Thus, $cost\_\delta_1 > M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + THRESHOLD$. Let $cost\_\delta_2$ be the cost of computing the sequence of Do-loops $L_{(i+\gamma_i+1)}$, ..., L_{i+j-1}, using the distribution scheme $P_{i,j}$. Then, $cost\_\delta_2$ is at least as large as $M_{(i+\gamma_i+1),(j-\gamma_i-1)}$. Therefore,

$$\begin{aligned} M_{i,j} &\geq cost\_\delta_1 + cost\_\delta_2 \\ &> M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + M_{(i+\gamma_i+1),(j-\gamma_i-1)} + THRESHOLD \\ &\geq M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + M_{(i+\gamma_i+1),(j-\gamma_i-1)} + cost(P_{i,\beta}, P_{(i+\beta),(\gamma_i-\beta+1)}) \\ &\quad + cost(P_{(i+\beta),(\gamma_i-\beta+1)}, P_{(i+\gamma_i+1),(j-\gamma_i-1)}). \qquad \square \end{aligned}$$

Theorem 2 : *Suppose that $THRESHOLD$ is equal to three times that of the maximal communication cost between any two distribution schema. If $M_{i,(\gamma_i+1)} > M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + THRESHOLD$, for some β where $1 \leq \beta < \gamma_i + 1$, then $T_{i,(\gamma_i+1)} > T_{(i+\beta),(\gamma_i-\beta+1)}$ and $T_{i,j} > T_{(i+\gamma_i+1),(j-\gamma_i-1)}$, for $\gamma_i + 1 < j \leq s - i + 1$.*

Proof: 1. $T_{i,(\gamma_i+1)}$ is the cost of computing the sequence of Do-loops L_1, L_2, ..., L_{i-1}, L_i, L_{i+1}, ..., $L_{(i+\gamma_i)}$, with the restriction that it uses the distribution scheme $P_{i,(\gamma_i+1)}$ to compute Do-loops L_i, L_{i+1}, ..., $L_{(i+\gamma_i)}$. Then,

$$\begin{aligned} & T_{i,(\gamma_i+1)} \\ =\; & \mathrm{MIN}_{1\leq k<i}\{T_{i-k,k} + M_{i,(\gamma_i+1)} + cost(P_{i-k,k}, P_{i,(\gamma_i+1)})\} \\ >\; & \mathrm{MIN}_{1\leq k<i}\{T_{i-k,k} + M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + THRESHOLD\} \\ \geq\; & \mathrm{MIN}_{1\leq k<i}\{T_{i-k,k} + M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + cost(P_{i-k,k}, P_{i,\beta}) + cost(P_{i,\beta}, P_{(i+\beta),(\gamma_i-\beta+1)})\} \\ \geq\; & T_{(i+\beta),(\gamma_i-\beta+1)}. \end{aligned}$$

2. $T_{i,j}$ is the cost of computing the sequence of Do-loops $L_1, L_2, \ldots, L_{i-1}, L_i, L_{i+1}, \ldots, L_{i+j-1}$, with the restriction that it uses the distribution scheme $P_{i,j}$ to compute Do-loops $L_i, L_{i+1}, \ldots, L_{i+j-1}$. Let $cost\_\delta_1$ and $cost\_\delta_2$ be the same ones defined in Theorem 1. Then,

$$\begin{aligned}
T_{i,j} &= \text{MIN}_{1\le k<i}\{T_{i-k,k} + M_{i,j} + cost(P_{i-k,k}, P_{i,j})\} \\
&\ge \text{MIN}_{1\le k<i}\{T_{i-k,k} + cost\_\delta_1 + cost\_\delta_2\} \\
&> \text{MIN}_{1\le k<i}\{T_{i-k,k} + M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + M_{(i+\gamma_i+1),(j-\gamma_i-1)} + THRESHOLD\} \\
&\ge \text{MIN}_{1\le k<i}\{T_{i-k,k} + M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + M_{(i+\gamma_i+1),(j-\gamma_i-1)} + cost(P_{i-k,k}; P_{i,\beta}) \\
&\quad + cost(P_{i,\beta}, P_{(i+\beta),(\gamma_i-\beta+1)}) + cost(P_{(i+\beta),(\gamma_i-\beta+1)}, P_{(i+\gamma_i+1),(j-\gamma_i-1)})\} \\
&\ge T_{(i+\gamma_i+1),(j-\gamma_i-1)}. \qquad \square
\end{aligned}$$

Suppose that $THRESHOLD$ is equal to four times that of the maximal communication cost between any two distribution schema. Under the condition in Theorem 1 and Theorem 2, we can further prove that it is better to use several distribution schema to compute the sequence of Do-loops $L_{i-\alpha}, L_{i-\alpha+1}, \ldots, L_{i+j-1}$, than to use only one distribution scheme $P_{(i-\alpha),(j+\alpha)}$, for $1 \le \alpha < i$ and $\gamma_i + 1 \le j \le s - i + 1$. Therefore, we need not compute $M_{(i-\alpha),(j+\alpha)}$. Based on this observation, we can show that $T_{(i-\alpha),(\gamma_i+1+\alpha)} > T_{(i+\beta),(\gamma_i-\beta+1)}$ and $T_{(i-\alpha),(j+\alpha)} > T_{(i+\gamma_i+1),(j-\gamma_i-1)}$, for $1 \le \alpha < i$ and $1 \le \beta < \gamma_i + 1 < j \le s - i + 1$. Therefore, we need not compute $T_{(i-\alpha),(j+\alpha)}$, for $1 \le \alpha < i$ and $\gamma_i + 1 \le j \le s - i + 1$.

Theorem 3 : *Suppose that $THRESHOLD$ is equal to four times that of the maximal communication cost between any two distribution schema. If $M_{i,(\gamma_i+1)} > M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + THRESHOLD$, for some β where $1 \le \beta < \gamma_i + 1$, then the following four cases are true for $1 \le \alpha < i$ and $\gamma_i + 1 < j \le s - i + 1$.*

1. $M_{(i-\alpha),(\gamma_i+1+\alpha)} > M_{(i-\alpha),\alpha} + M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + cost(P_{(i-\alpha),\alpha}, P_{i,\beta}) + cost(P_{i,\beta}, P_{(i+\beta),(\gamma_i-\beta+1)}$

2. $M_{(i-\alpha),(j+\alpha)} > M_{(i-\alpha),\alpha} + M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + M_{(i+\gamma_i+1),(j-\gamma_i-1)} + cost(P_{(i-\alpha),\alpha}, P_{i,\beta}) + cost(P_{i,\beta}, P_{(i+\beta),(\gamma_i-\beta+1)}) + cost(P_{(i+\beta),(\gamma_i-\beta+1)}, P_{(i+\gamma_i+1),(j-\gamma_i-1)})$;

3. $T_{(i-\alpha),(\gamma_i+1+\alpha)} > T_{(i+\beta),(\gamma_i-\beta+1)}$;

4. $T_{(i-\alpha),(j+\alpha)} > T_{(i+\gamma_i+1),(j-\gamma_i-1)}$.

Proof: We only prove the fourth case in this presentation, other cases can be dealt with using a similar technique. $T_{(i-\alpha),(j+\alpha)}$ is the cost of computing the sequence of Do-loops L_1, L_2, ..., $L_{i-\alpha-1}$, $L_{i-\alpha}$, ..., L_{i-1}, L_i, L_{i+1}, ..., $L_{(i+\beta-1)}$, $L_{(i+\beta)}$, ..., $L_{(i+\gamma_i)}$, $L_{(i+\gamma_i+1)}$, ..., L_{i+j-1}, with the restriction that it uses the distribution scheme $P_{(i-\alpha),(j+\alpha)}$ to compute Do-loops $L_{i-\alpha}$, $L_{i-\alpha+1}$, ..., L_{i+j-1}. Let *cost_*δ_3 be the cost of computing the sequence of Do-loops $L_{i-\alpha}$, $L_{i-\alpha+1}$, ..., L_{i-1}, using the distribution scheme $P_{(i-\alpha),(j+\alpha)}$. Then, *cost_*$\delta_3$ is at least as large as $M_{(i-\alpha),\alpha}$. Let *cost_*δ_4 be the cost of computing the sequence of Do-loops L_i, L_{i+1}, ..., $L_{(i+\gamma_i)}$, using the distribution scheme $P_{(i-\alpha),(j+\alpha)}$. Then, *cost_*$\delta_4$ is at least as large as $M_{i,(\gamma_i+1)}$. Thus, *cost_*$\delta_4 > M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + THRESHOLD$. Let *cost_*$\delta_5$ be the cost of computing the sequence of Do-loops $L_{(i+\gamma_i+1)}$, ..., L_{i+j-1}, using the distribution scheme $P_{(i-\alpha),(j+\alpha)}$. Then, *cost_*$\delta_5$ is at least as large as $M_{(i+\gamma_i+1),(j-\gamma_i-1)}$. Therefore,

$$\begin{aligned}
& T_{(i-\alpha),(j+\alpha)} \\
= \; & \text{MIN}_{1\le k<(i-\alpha)}\{T_{(i-\alpha)-k,k} + M_{(i-\alpha),(j+\alpha)} + cost(P_{(i-\alpha-k),k}, P_{(i-\alpha),(j+\alpha)})\} \\
\ge \; & \text{MIN}_{1\le k<(i-\alpha)}\{T_{(i-\alpha)-k,k} + cost\_\delta_3 + cost\_\delta_4 + cost\_\delta_5\} \\
> \; & \text{MIN}_{1\le k<(i-\alpha)}\{T_{(i-\alpha)-k,k} + M_{(i-\alpha),\alpha} + M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + M_{(i+\gamma_i+1),(j-\gamma_i-1)} \\
& + \; THRESHOLD\} \\
\ge \; & \text{MIN}_{1\le k<(i-\alpha)}\{T_{(i-\alpha)-k,k} + M_{(i-\alpha),\alpha} + M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + M_{(i+\gamma_i+1),(j-\gamma_i-1)} \\
& + \; cost(P_{(i-\alpha-k),k}, P_{(i-\alpha),\alpha}) + cost(P_{(i-\alpha),\alpha}, P_{i,\beta}) + cost(P_{i,\beta}, P_{(i+\beta),(\gamma_i-\beta+1)}) \\
& + \; cost(P_{(i+\beta),(\gamma_i-\beta+1)}, P_{(i+\gamma_i+1),(j-\gamma_i-1)})\} \\
\ge \; & T_{(i+\gamma_i+1),(j-\gamma_i-1)}. \qquad \square
\end{aligned}$$

From the techniques we used to prove Theorem 2 and Theorem 3, readers can understand why $THRESHOLD$ is chosen to be equal to three times or four times, respectively, that of the maximal communication cost between any two distribution schema. Theorems 1 through 3 also suggest an ordering sequence of computing $(M_{i,j})$-table and $(T_{i,j})$-table as shown in Fig. 5. For instance, if $M_{3,2}$ is larger than $M_{3,1}$ plus $M_{4,1}$ and plus a threshold value, then from Theorem 1, we need not compute $M_{3,3}$; from Theorem 2, we need not compute $T_{3,2}$ and $T_{3,3}$; from Theorem 3-(1), we need not compute $M_{2,3}$ and $M_{1,4}$; from Theorem 3-(2), we need not compute $M_{2,4}$ and $M_{1,5}$; from Theorem 3-(3), we need not compute $T_{2,3}$ and $T_{1,4}$; and from Theorem 3-(4), we need not compute $T_{2,4}$ and $T_{1,5}$.

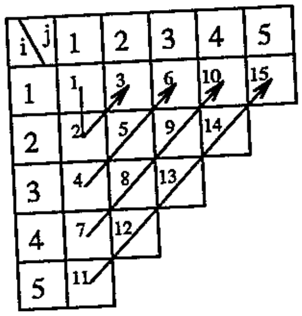


Figure 5: An ordering sequence of computing $(M_{i,j})$-table and $(T_{i,j})$-table.

4.1 The Case When a Program Fragment Contains a Sequence of s Do-loops

This section discusses the simplest case as shown in Fig. 1-(a). Based on Theorems 1 through 3, we can improve Algorithm 1. Let γ_i be the minimum integer such that $M_{i,(\gamma_i+1)} > M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + THRESHOLD$, for some β where $1 \leq \beta \leq \gamma_i \leq s-i+1$. Note that, for the boundary cases when $\gamma_i = s-i+1$ or $\beta = s-i+1$, we define dummy values $M_{i,s-i+2}$; $M_{s+1,1}$; and $M_{(i+\beta),(s-i-\beta+2)}$, so that the above assumption is satisfied. Let γ be the maximal value among γ_i, for $1 \leq i \leq s$. For example, $\gamma = \max_{1\leq i \leq s}\{\gamma_i\}$.

Algorithm 3: A new dynamic programming algorithm for computing the cost of data distribution schema of executing a sequence of s Do-loops on distributed memory computers is presented.

Input: $M_{i,j}$, $P_{i,j}$, and γ_i, where $1 \leq i \leq s$ and $1 \leq j \leq \gamma_i$; $T_{1,j}$ $(= M_{1,j})$, where $1 \leq j \leq \gamma_1$; and γ.

Output: The cost of executing s Do-loops on distributed memory computers.

```
1.  for $i := 2$ to $s$ do
2.      for $j := 1$ to $\gamma_i$ do
3.          $T_{i,j} := \mathrm{MIN}_{1 \leq k < \min\{i,\gamma+1\}}\{T_{i-k,k} + M_{i,j} + cost(P_{i-k,k}, P_{i,j}),\ \text{if } k \leq \gamma_{i-k}\}$ ;
4.  end_for end_for
5.  $Minimum\_Cost := \mathrm{MIN}_{1 \leq k \leq \gamma}\{T_{s-k+1,k} + loop\_carried\_dependence(T_{s-k+1,k}),$
6.                                  $\text{if } k \leq \gamma_{s-k+1}\}$ .
```

We now analyze Algorithm 3. The time complexity of this new dynamic programming algorithm is $O((\sum_{i=2}^{s}\gamma_i)\gamma + \gamma)$, which is bounded by $O(s\gamma^2)$. In addition, before applying Algorithm 3, we only need to compute $\gamma_1 + \gamma_2 + \cdots + \gamma_s + s$ component alignment problems for the consecutive Do-loops

M_{ij}	j = 1	j = 2	j = 3	j = 4	j = 5
i = 1	0	0	C_d	save	save
i = 2	0	C_d	C_e	save	
i = 3	0	0	C_f		
i = 4	0	C_f			
i = 5	0				

$$
\begin{aligned}
T_{11} &= M_{11} = 0\\
T_{12} &= M_{12} = 0\\
T_{13} &= M_{13} = C_d\\
T_{21} &= T_{11} + M_{21} + (cost(P_{11}, P_{21}) = 0) = 0\\
T_{22} &= T_{11} + M_{22} + (cost(P_{11}, P_{22}) = 0) = C_d\\
T_{31} &= \mathrm{MIN}\{T_{21} + M_{31} + (cost(P_{21}, P_{31}) = C_T), T_{12} + M_{31} + (cost(P_{12}, P_{31}) = C_T)\} = C_T\\
T_{32} &= \mathrm{MIN}\{T_{21} + M_{32} + (cost(P_{21}, P_{32}) = C_T), T_{12} + M_{32} + (cost(P_{12}, P_{32}) = C_T)\} = C_T\\
T_{33} &= \mathrm{MIN}\{T_{21} + M_{33} + (cost(P_{21}, P_{33}) = C_T), T_{12} + M_{33} + (cost(P_{12}, P_{33}) = C_T)\} = C_f + C_T\\
T_{41} &= \mathrm{MIN}\{T_{31} + M_{41} + (cost(P_{31}, P_{41}) = 0), T_{22} + M_{41} + (cost(P_{22}, P_{41}) = C_T),\\
&\qquad T_{13} + M_{41} + (cost(P_{13}, P_{41}) = C_T)\} = C_T\\
T_{42} &= \mathrm{MIN}\{T_{31} + M_{42} + (cost(P_{31}, P_{42}) = 0), T_{22} + M_{42} + (cost(P_{22}, P_{42}) = C_T),\\
&\qquad T_{13} + M_{42} + (cost(P_{13}, P_{42}) = C_T)\} = C_f + C_T\\
T_{51} &= \mathrm{MIN}\{T_{41} + M_{51} + (cost(P_{41}, P_{51}) = C_f), T_{32} + M_{51} + (cost(P_{32}, P_{51}) = C_f)\} = C_f + C_T.
\end{aligned}
$$

T_{ij}	j = 1	j = 2	j = 3	j = 4	j = 5
i = 1	0	0	C_d	save	save
i = 2	0	C_d	save	save	
i = 3	C_T	C_T	$C_f + C_T$		
i = 4	C_T	$C_f + C_T$			
i = 5	$C_f + C_T$				

$$
\begin{aligned}
Minimum\_Cost &= \mathrm{MIN}_{1 \le k \le 3}\{T_{5-k+1,\,k} + loop\_carried\_dependence(T_{5-k+1,\,k})\}\\
&= \mathrm{MIN}\{T_{51} + C_T, T_{42} + C_T, T_{33} + C_T\}\\
&= 2C_T + C_f.
\end{aligned}
$$

Table 3: Apply Algorithm 3 to the sample program.

L_i, L_{i+1}, …, L_{i+j-1}, where $1 \le i \le s$ and $1 \le j \le \gamma_i + 1$. The total number of component alignment problems computed is thus no more than $s(\gamma + 1)$.

Table 3 shows a complete example by applying Algorithm 3 to the sample program mentioned in Section 2.2 for determining data distribution. In this example, we let $THRESHOLD$ be $4 * C_T$, we also assume that $4 * C_T$ is very small in comparison with C_e $(= K * (m * (\log N) + m) * t_c)$. Thus, $\gamma_1 = 3$; $\gamma_2 = 2$; $\gamma_3 = 3$; $\gamma_4 = 2$; $\gamma_5 = 1$; and $\gamma = 3$. We can see that the result computed from Algorithm 3 is the same as the one computed from Algorithm 1. However, the computation for $M_{1,4}$; $M_{1,5}$; $M_{2,4}$; $T_{1,4}$; $T_{1,5}$; $T_{2,3}$; and $T_{2,4}$ is saved. In addition, the computation for $T_{5,1}$ and $Minimum\_Cost$ is simplified.

4.2 The Case When s Do-loops Are Enclosed by an Iterative Loop

This section discusses the second case as shown in Fig. 1-(b). Similarly, Algorithm 2 can be improved if it adopts Algorithm 3 as part of its program instead of Algorithm 1. Let γ_i be the minimum integer such that $M_{i,(\gamma_i+1)} > M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + THRESHOLD$ if $(i+\beta) \leq s$, or $M_{i,(\gamma_i+1)} > M_{i,\beta} + M_{(i+\beta-s),(\gamma_i-\beta+1)} + THRESHOLD$ if $(i+\beta) > s$, for some β where $1 \leq \beta \leq \gamma_i \leq s$. Note that, for the boundary case when $\gamma_i = s$ we define dummy values $M_{i,s+1}$ and $M_{(i+\beta),(s-\beta+1)}$ so that the above assumption is satisfied. Let γ be the maximal value among γ_i, for $1 \leq i \leq s$. For example, $\gamma = \max_{1 \leq i \leq s}\{\gamma_i\}$.

However, unlike Algorithm 2, which computes all s sequences of rotated s Do-loops, the new version algorithm only needs to compute the first (γ_1+1) sequences of rotated s Do-loops. This is because we will use at least two distribution schema to compute the first (γ_1+1) consecutive Do-loops. However, we are only interested in whether the first distribution scheme can be combined with the last distribution scheme.

Algorithm 4: A new dynamic programming algorithm for computing the cost of data distribution schema of executing a sequence of s Do-loops enclosed by an iterative loop on distributed memory computers is presented.

Input: $M_{i,j}$, $P_{i,j}$, and γ_i, where $1 \leq i \leq s$ and $1 \leq j \leq \gamma_i$; $T_{1,j}$ $(= M_{1,j})$, where $1 \leq j \leq \gamma_1$; and γ.

Output: The cost of executing an iterative loop which contains s Do-loops on distributed memory computers.

```
1.   $\Gamma_1 := (\gamma_1 + 1)$;
2.   for $m := 1$ to $\Gamma_1$ do
3.                    /* Compute $\Gamma_1$ sequences of Do-loops. */
4.       for $i := 2$ to $s$ do
5.           for $j := 1$ to $\min\{\gamma_i, s-i+1\}$ do
6.               $T_{i,j} := \mathrm{MIN}_{1 \leq k < \min\{i,\gamma+1\}}\{T_{i-k,k} + M_{i,j} + cost(P_{i-k,k}, P_{i,j}),$ if $k \leq \gamma_{i-k}\}$ ;
7.       end_for end_for
8.       $Minimum\_Cost(m) := \mathrm{MIN}_{1 \leq k \leq \gamma}\{T_{s-k+1,k} + loop\_carried\_dependence(T_{s-k+1,k}),$
9.                                              if $k \leq \gamma_{s-k+1}\}$ ;
10.                   /* Shift $(M_{i,j})$-table. */
```

```
11.        copy $\gamma_1$ to $Temp\Gamma$;
12.        for $j := 1$ to $\gamma_1$ do
13.            copy $M_{1,j}$ to $TempM_j$ , copy $P_{1,j}$ to $TempP_j$ ;
14.        end_for
15.        for $i := 2$ to $s$ do
16.            copy $\gamma_i$ to $\gamma_{i-1}$;
17.            for $j := 1$ to $\gamma_i$ do
18.                copy $M_{i,j}$ to $M_{i-1,j}$ , copy $P_{i,j}$ to $P_{i-1,j}$ ;
19.        end_for end_for
20.        copy $Temp\Gamma$ to $\gamma_s$;
21.        for $j := 1$ to $\gamma_s$ do
22.            copy $TempM_j$ to $M_{s,j}$ , copy $TempP_j$ to $P_{s,j}$ ;
23.        end_for
24.        for $j := 1$ to $\gamma_1$ do
25.            copy $M_{1,j}$ to $T_{1,j}$ ;
26.        end_for
27.    end_for
28.    $Final\_Minimum\_Cost := \text{MIN}_{1 \le m \le \Gamma_1}\{Minimum\_Cost(m)\}$ .
```

We now analyze Algorithm 4. The time complexity of this new dynamic programming algorithm is $O(\gamma_1(\sum_{i=2}^{s}\gamma_i)\gamma)$, which is bounded by $O(s\gamma^3)$. In addition, before applying Algorithm 4, we only need to compute $\gamma_1+\gamma_2+\cdots+\gamma_s+s$ component alignment problems. The total number of component alignment problems computed is thus no more than $s(\gamma+1)$.

4.3 The Case When a Program Contains s Do-loops with a General Structure

This section discusses the most general case as shown in Fig. 1-(c). In general, some Do-loops in a sequence of Do-loops may be iterative loops, which contain other sequences of Do-loops with a general structure. In other words, some consecutive Do-loops may be enclosed by iterative loops, which, again, with adjacent Do-loops, may be enclosed by other iterative loops, and so on. This enclosure relation can be naturally represented by trees (or a forest). A Do-loop may be a simple Do-loop or an iterative loop. Suppose that an iterative loop encloses at least two Do-loops. Then, simple Do-loops are leaf nodes in the trees, iterative loops are internal nodes in the trees. If ω Do-loops are enclosed by an iterative loop, then this iterative loop is the parent node of these ω Do-loops, and these ω Do-loops are the ω corresponding child nodes of this iterative loop.

For instance, in the sample program mentioned in Section 2.2, Do-loops L_1, L_2, L_3, L_4, and L_5 are five child nodes of the outmost iterative loop. If we further elaborate the sample program, we can see that Do-loops L_2 and L_4 are two iterative loops and each of them contains three small Do-loops. L_2 contains $L_{2,1}$, which is from line 8 to line 12; $L_{2,2}$, which is from line 13 to line 16; and $L_{2,3}$, which is from line 17 to line 19. L_4 contains $L_{4,1}$, which is from line 26 to line 30; $L_{4,2}$, which is from line 31 to line 34; and $L_{4,3}$, which is from line 35 to line 37. Fig. 6 shows the family tree of the sample program. For convenience, we will say that L_1, L_2, L_3, L_4, and L_5 are the *first-level* Do-loops in the outmost iterative loop; in addition, they are *siblings* in this tree representation. Similarly, $L_{2,1}$, $L_{2,2}$, and $L_{2,3}$ are the first-level Do-loops in L_2, and they are siblings; $L_{4,1}$, $L_{4,2}$, and $L_{4,3}$ are the first-level Do-loops in L_4, and they are siblings.

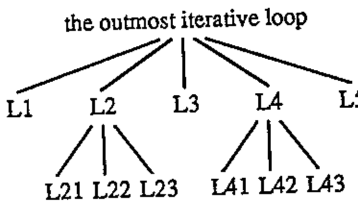


Figure 6: The family tree of the sample program.

Before introducing Algorithm 5, we first present an algorithm which computes the cost of data distribution schema of executing an iterative loop that contains a sequence of s simple Do-loops with a general structure. This basic algorithm will be used in Algorithm 5.

Algorithm 5a: An algorithm for computing the cost of data distribution schema of executing an iterative loop which contains a sequence of s simple Do-loops with a general structure on distributed memory computers is constructed.

Input: A program fragment of an iterative loop which contains a sequence of s simple Do-loops with a general structure.

Output: The cost of executing this iterative loop on distributed memory computers.

```
1.  suppose that there are $\psi(s)$ first-level Do-loops in this input iterative loop;
```

```
2.  scan these $\psi(s)$ first-level Do-loops one by one, while there exists an iterative loop, $L_\sigma$, which
        contains a sequence of $\omega$ simple Do-loops: $L_{\sigma 1}$, $L_{\sigma 2}$, ..., $L_{\sigma\omega}$, with a general structure, do
3.      recursively apply Algorithm 5a to the iterative loop $L_\sigma$;
4.  construct $(M_{i,j})$-table, $(P_{i,j})$-table, $\gamma_i$, and $\gamma$ for these $\psi(s)$ first-level Do-loops,
        where $1 \le i \le \psi(s)$, $1 \le j \le \gamma_i \le \psi(s)$, and $\gamma = \max_{1\le i\le\psi(s)}\{\gamma_i\}$;
5.  apply Algorithm 4 to these $\psi(s)$ first-level Do-loops which are enclosed by an iterative loop,
        and multiply the number of iterations to the resulting sequence of distribution schema
        as their weight.
```

We now briefly illustrate Algorithm 5a. The first four steps in Algorithm 5a are quite straightforward, in the following, we only explain the fifth step. We notice that, after applying Algorithm 4 to an iterative loop, if the resulting sequence of distribution schema contains more than one distribution scheme, then these distribution schema cannot be combined with any other distribution scheme in the sequel. For convenience, we use a dummy distribution scheme to represent the resulting distribution schema in the sequel. However, if there is only one distribution scheme obtained from Algorithm 4, this distribution scheme may be combined with schema obtained from adjacent Do-loops.

The following Algorithm 5 is very similar to Algorithm 5a, except that Algorithm 5 deals with a program fragment which is not necessary to contain only one iterative loop.

Algorithm 5: An algorithm for computing the cost of data distribution schema of executing a program fragment which contains a sequence of s simple Do-loops with a general structure on distributed memory computers is constructed.

Input: A program fragment which contains a sequence of s simple Do-loops with a general structure.

Output: The cost of executing this sequence of Do-loops on distributed memory computers.

```
1.  suppose that there are $\psi(s)$ first-level Do-loops in this input program fragment;
2.  scan these $\psi(s)$ first-level Do-loops one by one, while there exists an iterative loop, $L_\sigma$, which
        contains a sequence of $\omega$ simple Do-loops: $L_{\sigma 1}$, $L_{\sigma 2}$, ..., $L_{\sigma\omega}$, with a general structure, do
3.      apply Algorithm 5a to the iterative loop $L_\sigma$;
4.  construct $(M_{i,j})$-table, $(P_{i,j})$-table, $\gamma_i$, and $\gamma$ for these $\psi(s)$ first-level Do-loops,
        where $1 \le i \le \psi(s)$, $1 \le j \le \gamma_i \le \psi(s) - i + 1$, and $\gamma = \max_{1\le i\le\psi(s)}\{\gamma_i\}$;
5.  apply Algorithm 3 to these $\psi(s)$ first-level Do-loops, which are not enclosed by any iterative loop,
        for finding the resulting sequence of distribution schema.
```

We now use the sample program again to go through Algorithm 5. First, because the sample program contains only one iterative loop, Algorithm 5 calls Algorithm 5a to handle this iterative loop. Then, because the outmost iterative loop contains five Do-loops L_1; L_2; L_3; L_4; and L_5, it (Algorithm 5a) scans these five Do-loops one by one. Since L_2 and L_4 are iterative loops, it recursively applies itself to these two loops. When dealing with L_2, because $L_{2,1}$, $L_{2,2}$, and $L_{2,3}$ are simple Do-loops, it constructs $(M_{i,j})$-table, $(P_{i,j})$-table, γ_i, and γ for these three Do-loops, where $1 \leq i \leq 3, 1 \leq j \leq \gamma_i = 3$, and $\gamma = 3$. After that, it can apply Algorithm 4 to L_2, and obtains a single data distribution scheme which illustrates that matrix A is distributed row by row as mentioned in Table 1. Similarly, when dealing with L_4, because $L_{4,1}$, $L_{4,2}$, and $L_{4,3}$ are simple Do-loops, it constructs $(M_{i,j})$-table, $(P_{i,j})$-table, γ_i, and γ for these three Do-loops, where $1 \leq i \leq 3$, $1 \leq j \leq \gamma_i = 3$, and $\gamma = 3$. After that, it can apply Algorithm 4 to L_4, and obtains a single data distribution scheme which illustrates that matrix A is distributed column by column also as mentioned in Table 1.

After handling L_2 and L_4 (both of which are iterative loops), Algorithm 5a constructs $(M_{i,j})$-table, $(P_{i,j})$-table, γ_i, and γ for the five Do-loops L_1; L_2; L_3; L_4; and L_5, where $1 \leq i \leq 5$; $1 \leq j \leq \gamma_i \leq 5$; $\gamma_1 = 3$; $\gamma_2 = 2$; $\gamma_3 = 4$; $\gamma_4 = 3$; $\gamma_5 = 4$; and $\gamma = 4$. It then applies Algorithm 4 to these five first-level Do-loops, and obtains a sequence of two data distribution schema as shown in Section 3. Then, it returns to Algorithm 5. Because there is only one outmost iterative loop in the sample program and whose data distribution schema have been obtained, Step 4 and Step 5 in Algorithm 5 are not applied in this case.

We now analyze the time complexity of Algorithm 5. First, Algorithm 3 can be regarded as a special case of Algorithm 4. Suppose that a sequence of s simple Do-loops with a general structure are enclosed by $\psi(s)$ disjointed first-level iterative loops or simple Do-loops, and in addition, the i-th iterative loop contains s_i simple Do-loops with a general structure. Then, $s = \sum_{i=1}^{\psi(s)} s_i$. We now first analyze how many component alignment problems are required to be computed in Algorithm 5. From Section 4.2, if an iterative loop contains s simple Do-loops, it requires computing $O(s\gamma)$ component alignment problems. Therefore, from line 4 of Algorithm 5a and Algorithm 5, we can formulate the recursive formula of the number of component alignment problems that are required to be computed in Algorithm 5 as follows.

$$\begin{cases} C(1) = 1; \\ C(s) = \sum_{i=1}^{\psi(s)} C(s_i) + O(\psi(s)\gamma), \text{ where } s = \sum_{i=1}^{\psi(s)} s_i. \end{cases}$$

This recursion formula is similar to the one that counts the number of nodes in an arbitrary tree in which each internal node has at least two child nodes, and is bounded by the order of the number of its leaf nodes. Therefore, using a similar technique, we can show that $C(s)$ is bounded by $O(s\gamma)$; in addition, the constant factor is less than 2.

We now analyze other computation time required for Algorithm 5. First, from Section 4.2, if an iterative loop contains s simple Do-loops, Algorithm 4 can deal with this iterative loop within $O(s\gamma^3)$ time units. Therefore, from line 5 of Algorithm 5a and Algorithm 5, the recursion formula of other computation time $T(s)$ can be formulated as follows.

$$\begin{cases} T(1) = 1; \\ T(s) = \sum_{i=1}^{\psi(s)} T(s_i) + O(\psi(s)\gamma^3), \text{ where } s = \sum_{i=1}^{\psi(s)} s_i. \end{cases}$$

Similar to computing $C(s)$, we can show that $T(s)$ is bounded by $O(s\gamma^3)$.

5 Experimental Studies

In this section we present experimental studies and show why it is important to determine whether data re-distribution is necessary. The target machine we used is a 32-node nCUBE-2 computer currently in Academia Sinica. In this computer, each node has 4 Mega bytes memory, runs at a modest clock rate of 20 MHz, and is rated at 7.5 MIPS (Mega Instructions Per Second), and 3.5 MFLOPS (Mega FLOating-point operations Per Second) in single precision arithmetic.

5.1 The Sample Program

Table 4 lists experimental results of implementing the sample application in Section 2.2 with various problem sizes. In this experimental study, we implement two versions of parallel programs: (1) based on a dynamic data distribution scheme; (2) based on a static data distribution scheme. We let the constant OUT_ITERATION = 10, and the constant MAX_ITERATION = $20 * \log m$, where m is the problem size. Experimental results show that to use the proposed dynamic data distribution scheme is better than to use a static data distribution scheme. Note that, the computation time of these two parallel algorithms are not exactly the same, because the second algorithm which implements several message-passing data communication operations during the computation requires more indexing operations than the first algorithm which is based on the original sequential computation.

matrix size	#PE = 2	#PE = 4	#PE = 8	#PE = 16	#PE = 32
$2^5 \times 2^5$	11.1 (0.1)	5.6 (0.1)	2.8 (0.1)	1.6 (0.3)	1.2 (0.5)
	12.0 (0.8)	7.3 (1.6)	5.9 (2.5)	6.1 (3.8)	8.2 (5.9)
$2^6 \times 2^6$	51.7 (0.3)	25.9 (0.2)	13.1 (0.2)	6.7 (0.3)	3.7 (0.6)
	53.3 (1.3)	28.9 (2.4)	17.8 (3.8)	13.6 (5.7)	14.0 (8.5)
$2^7 \times 2^7$	238.2 (1.2)	119.3 (0.8)	59.8 (0.5)	30.1 (0.5)	15.5 (0.7)
	241.3 (2.0)	124.8 (4.0)	67.8 (6.3)	41.6 (9.2)	31.5 (13.2)
$2^8 \times 2^8$	1083 (4.9)	542.1 (2.9)	271.3 (1.7)	136.0 (1.2)	68.5 (1.1)
	1091 (3.9)	553.2 (7.6)	287.0 (11.8)	156.5 (16.3)	96.4 (22.7)
$2^9 \times 2^9$	4869 (20.3)	2435 (11.3)	1218 (6.8)	610.0 (3.9)	305.6 (2.6)
	4890 (7.8)	2462 (15.9)	1251 (23.4)	651.3 (32.1)	358.0 (42.4)
$2^{10} \times 2^{10}$	****	10841 (45.2)	5427 (29.8)	2714 (15.7)	1358 (9.0)
		10911 (32.4)	5499 (50.6)	2801 (67.0)	1464 (85.2)
$2^{11} \times 2^{11}$	****	****	****	12129 (68.5)	6072 (46.8)
				12322 (149.0)	6291 (190.4)

Table 4: The simulation time, "execution time (communication time)", for solving the sample program is expressed in units of seconds: (1) based on a dynamic data distribution scheme; (2) based on a static data distribution scheme. (The net computation time) = (execution time) − (communication time). "****" means "not implement" because of the memory limitation.

5.2 Two-dimensional Fast Fourier Transform (2-D FFT)

When given a data matrix whose entries are complex numbers, the 2-D FFT can be computed by a conventional row-column method. In this method, first, we perform a 1-D FFT for each row; then, we perform a 1-D FFT for each column. In this experimental study, we implement a 2-D FFT and then immediately following by an inverse 2-D FFT. Therefore, the input data matrix will be equal to the output data matrix. This program contains four loops.

L_1: loop 1 performs a 1-D FFT for each row;
L_2: loop 2 evaluates a 1-D FFT for each column;
L_3: loop 3 calculates an inverse 1-D FFT for each column; and
L_4: loop 4 computes an inverse 1-D FFT for each row.

Table 5 shows the approximate computation time and communication time of these four loops depending on whether the input data matrix A is distributed row by row or distributed column by column. A static data distribution scheme by distributing data either row by row or column by column will incur $2C_h$ communication overhead due to requiring several "bit-reverse shuffle-exchange" and "butterfly-pattern" data communication, where $C_h = 2 * (m^2/N) * (\log N) * t_c$. However, because $C_h > C_T$, where C_T is the cost of performing a matrix transpose operation, thus, by applying Algorithm 5 (or Algorithm 1 or Algorithm 3), we can show that data re-distribution is required between L_1 and L_2, and between L_3 and L_4. Table 6 lists the experimental results of implementing this 2-D FFT program based on both a dynamic data distribution scheme and a static data distribution scheme. Experimental results also show that to use the above mentioned dynamic data distribution scheme is better than to use a static data distribution scheme.

	matrix A is distributed row by row		matrix A is distributed column by column	
	computation time	communication time	computation time	communication time
L_1	C_g	0	C_g	C_h
L_2	C_g	C_h	C_g	0
L_3	C_g	C_h	C_g	0
L_4	C_g	0	C_g	C_h

Table 5: Computation time and communication time of four loops. $C_g = c * (m^2 * (\log m)/N) * t_f$, where c is a constant; $C_h = 2 * (m^2/N) * (\log N) * t_c$.

6 Conclusions

We have presented five heuristic algorithms for data distribution on distributed memory multicomputers. In addition, data distribution schema obtained from these five algorithms are at least as good as any static data distribution schema. First, we proposed a primitive dynamic programming algorithm for data distribution which is suitable for the case when a program contains s Do-loops. However, when dealing with the case when a sequence of s Do-loops are enclosed by an iterative loop, the resulting data distribution schema derived from this algorithm cannot be satisfied. We then generalized the first algorithm to deal with this more general case. The fundamental rationale behind this generalized algorithm is that we found this sequence of s Do-loops appearing in a cyclic fashion. Therefore, every Do-loop can be treated as the first Do-loop.

matrix size	#PE = 2	#PE = 4	#PE = 8	#PE = 16	#PE = 32
$2^5 \times 2^5$	0.164 (0.017) 0.253 (0.088)	0.085 (0.012) 0.254 (0.165)	0.048 (0.012) 0.294 (0.241)	0.034 (0.016) 0.353 (0.319)	0.038 (0.029) 0.266 (0.243)
$2^6 \times 2^6$	0.749 (0.063) 0.950 (0.205)	0.382 (0.039) 0.745 (0.351)	0.199 (0.027) 0.712 (0.499)	0.111 (0.025) 0.775 (0.651)	0.077 (0.034) 0.885 (0.809)
$2^7 \times 2^7$	3.395 (0.246) 3.885 (0.523)	1.719 (0.144) 2.520 (0.782)	0.875 (0.088) 1.965 (1.055)	0.454 (0.060) 1.833 (1.338)	0.252 (0.054) 1.918 (1.637)
$2^8 \times 2^8$	15.205 (0.974) 16.524 (1.500)	7.677 (0.563) 9.573 (1.890)	3.884 (0.326) 6.284 (2.323)	1.974 (0.195) 4.893 (2.823)	1.020 (0.130) 4.472 (3.356)
$2^9 \times 2^9$	67.332 (3.840) 71.533 (4.933)	33.982 (2.230) 39.060 (5.154)	17.156 (1.279) 22.883 (5.564)	8.668 (0.730) 15.103 (6.197)	4.398 (0.427) 11.700 (7.051)
$2^{10} \times 2^{10}$	****	****	75.161 (5.070) 90.658 (15.071)	37.900 (2.854) 53.256 (14.753)	19.134 (1.610) 35.267 (15.490)
$2^{11} \times 2^{11}$	****	****	****	****	83.016 (6.314) 121.486 (36.713)

Table 6: The simulation time, "execution time (communication time)", for solving the 2-D FFT program is expressed in units of seconds: (1) based on a dynamic data distribution scheme; (2) based on a static data distribution scheme. "****" means "not implement" because of the memory limitation.

After that, we showed that data re-distribution is necessary for executing a sequence of Do-loops if the communication cost due to perform this sequence of Do-loops is larger than a threshold value. Based on this observation, we derived three new efficient algorithms for data distribution. These algorithms can be used to deal with the most general case when a sequence of Do-loops contains some iterative loops, and of which themselves may contain further, smaller Do-loops with a general structure.

Suppose that we must use at least two distribution schema to compute any $(\gamma + 1)$ consecutive Do-loops. Then, we can find the sequence of distribution schema for executing s consecutive simple Do-loops (as shown in Fig. 1-(a)) in $O(s\gamma^2)$ time units; for executing an iterative loop which contains s simple Do-loops (as shown in Fig. 1-(b)) in $O(s\gamma^3)$ time units; and for executing a sequence of s simple Do-loops with a general structure (as shown in Fig. 1-(c)) also in $O(s\gamma^3)$ time units. In addition, while applying these algorithms, we only need to compute at most $s(\gamma + 1)$ component alignment problems each with a reasonable problem size for the first two cases, and to compute at most $2s(\gamma+1)$ component alignment problems each with a reasonable problem size for the third case. In practice, our method can be used in parallelizing compilers to automatically determine data distribution for distributed memory systems.

References

[1] J. M. Anderson and M. S. Lam. Global optimizations for parallelism and locality on scalable parallel machines. In *Proc. of ACM-SIGPLAN PLDI*, pages 112–125, Albuquerque, N.M., June 1993.

[2] D. Callahan and K. Kennedy. Compiling programs for distributed-memory multiprocessors. *The Journal of Supercomputing*, 2:151–169, 1988.

[3] B. Chapman, T. Fahringer, and H. Zima. Automatic support for data distribution on distributed memory multiprocessor systems. In *Lecture Notes in Computer Science 768, Sixth International Workshop on Languages and Compilers for Parallel Computing*, pages 184–199, Portland, Oregon, August 1993.

[4] B. Chapman, P. Mehrotra, H. Moritsch, and H. Zima. Dynamic data distributions in Vienna Fortran. In *Proc. of Supercomputing '93*, pages 284–293, Portland, Oregon, November 1993.

[5] S. Chatterjee, J. R. Gilbert, and R. Schreiber. Mobile and replicated alignment of arrays in data-parallel programs. In *Proc. of Supercomputing '93*, November 1993.

[6] S. Chatterjee, J. R. Gilbert, R. Schreiber, and S. H. Teng. Automatic array alignment in data-parallel programs. In *Proc. of ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, Charleston, SC, January 1993.

[7] T. Chen and J. Sheu. Communication-free data allocation techniques for parallelizing compilers on multicomputers. *IEEE Trans. Parallel Distributed Syst.*, 5(9):924–938, September 1994.

[8] C. Gong, R. Gupta, and R. Melhem. Compilation techniques for optimizing communication on distributed-memory systems. In *Proc. of International Conf. on Parallel Processing*, pages II–39–46, St. Charles, IL, Aug. 1993.

[9] M. Gupta and P. Banerjee. Demonstration of automatic data partitioning techniques for parallelizing compilers on multicomputers. *IEEE Trans. Parallel Distributed Syst.*, 3(2):179–193, Mar. 1992.

[10] S. Hiranandani, K. Kennedy, and C-W. Tseng. Compiling Fortran D for MIMD distributed-memory machines. *Communications of the ACM*, 35(8):66–80, Aug. 1992.

[11] P. D. Hovland and L. M. Ni. A model for automatic data partitioning. In *Proc. of International Conf. on Parallel Processing*, pages II–251–259, St. Charles, IL, Aug. 1993.

[12] C. H. Huang and P. Sadayappan. Communication-free hyperplane partitioning of nested loops. *Journal of Parallel and Distributed Computing*, 19:90–102, 1993.

[13] D. E. Hudak and S. G. Abraham. *Compiling Parallel Loops for High Performance Computers.* Kluwer Academic Publishers, Norwell, Massachusetts, 1993.

[14] E. T. Kalns and L. M. Ni. Processor mapping techniques toward efficient data redistribution. Technical Report MSU-CPS-ACS-86, Department of Computer Science, Michigan State University, January 1994.

[15] E. T. Kalns, H. Xu, and L. M. Ni. Evaluation of data distribution patterns in distributed-memory machines. In *Proc. of International Conf. on Parallel Processing*, pages II–175–183, St. Charles, IL, Aug. 1993.

[16] K. Knobe, J. D. Lukas, and G. L. Steele Jr. Data optimization: Allocation of arrays to reduce communication on SIMD machines. *Journal of Parallel and Distributed Computing*, 8(2):102–118, Feb. 1990.

[17] K. Knobe and V. Natarajan. Automatic data allocation to minimize communication on SIMD machines. *The Journal of Supercomputing*, 7:387–415, 1993.

[18] U. Kremer. Np-completeness of dynamic remapping. In *Proc. of the Fourth Workshop on Compilers for Parallel Computers*, Delft, The Netherlands, December 1993.

[19] U. Kremer. Automatic data layout using 0–1 integer programming. In *Proc. of International Conf. on Parallel Architectures and Compilation Techniques*, Montréal, Canada, August 1994.

[20] U. Kremer, J. Mellor-Crummey, K. Kennedy, and A. Carle. Automatic data layout for distributed-memory machines in the D programming environment. In *Automatic Parallelization — New Approaches to Code Generation, Data Distribution, and Performance Prediction*, pages 136–152, Vieweg Advanced Studies in Computer Science, Verlag Vieweg, Wiesbaden, Germany, 1993.

[21] P.-Z. Lee and T. B. Tsai. Compiling efficient programs for tightly-coupled distributed memory computers. In *Proc. of International Conf. on Parallel Processing*, pages II–161–165, St. Charles, IL, August 1993, also Technical Report TR-93-004, Institute of Information Science, Academia Sinica.

[22] J. Li and M. Chen. Compiling communication-efficient problems for massively parallel machines. *IEEE Trans. Parallel Distributed Syst.*, 2(3):361–376, July 1991.

[23] J. Li and M. Chen. The data alignment phase in compiling programs for distributed-memory machines. *Journal of Parallel and Distributed Computing*, 13:213–221, 1991.

[24] M. Mace. *Memory Storage Patterns in Parallel Processing.* Kluwer Academic Publishers, Boston, MA, 1987.

[25] P. Mehrotra and J. Van Rosendale. Programming distributed memory architectures using Kali. In A. Nicolau, D. Gelernter, T. Gross, and D. Padua, editors, *Advances in Languages and Compilers for Parallel Computing*, pages 364–384. Pitman/MIT-Press, 1991.

[26] J. Ramanujam and P. Sadayappan. Compile-time techniques for data distribution in distributed memory machines. *IEEE Trans. Parallel Distributed Syst.*, 2(4):472–482, Oct. 1991.

[27] J. Ramanujam and P. Sadayappan. Tiling multidimensional iteration spaces for multicomputers. *Journal of Parallel and Distributed Computing*, 16:108–120, 1992.

[28] P. S. Tseng. *A Systolic Array Parallelizing Compiler.* Kluwer Academic Publishers, Boston, MA, 1990.

[29] S. Wholey. Automatic data mapping for distributed-memory parallel computers. In *Proc. of International Conf. on Supercomputing*, July 1992.

[30] M. E. Wolf and M. S. Lam. A data locality optimizing algorithm. In *Proc. of ACM SIGPLAN '91 Conf. on Programming Language Design and Implementation*, pages 30–44, Toronto, Ontario, Canada, June 1991.

[31] M. E. Wolf and M. S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. Parallel Distributed Syst.*, 2(4):452–471, Oct. 1991.

[32] H. P. Zima, H-J. Bast, and M. Gerndt. SUPERB: A tool for semi-automatic MIMD/SIMD parallelization. *Parallel Computing*, 6:1–18, 1988.