1

Efficient Algorithms for Data Distribution on Distributed Memory Multicomputers

PeiZong Lee
Institute of Information Science
Academia Sinica
Taipei, Taiwan, R.O.C.

Abstract

Data distribution has been one of the most important research topics in parallelizing compilers for distributed memory parallel computers. 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.

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

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.

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

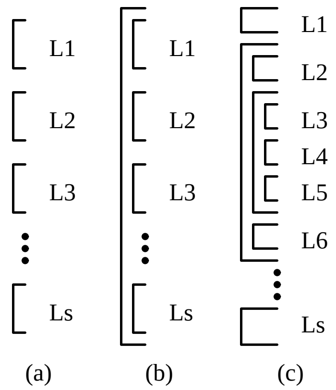

Figure 1: (a) a sequence of s Do-loops; (b) a sequence of s Do-loops which are enclosed by an iterative loop; and (c) a sequence of s Do-loops with a general structure.

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. 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. It is also possible to use compiler techniques to automatically determine sequential programs' data distribution on distributed memory systems. Li and Chen [15], Gupta and Banerjee [5] formulated the component alignment problem from the whole source program and used it to determine data distribution. However, the fixed data distribution schema they derived may result in a larger communication overhead.

In addition, there are other research works related to the compilation of programs on distributed-memory computers. Chapman, Fahringer, Mehrotra, Moritsch, and Zima adopted Li and Chen's component alignment algorithm [15] for handling distributed data in Vienna Fortran [1]; in addition, they used language extension for handling dynamic data distribution [2]. 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 [11]. Kremer also developed techniques for using 0-1 integer programming for automatic data layout in the inter-phase data layout problem [10]. Other papers, which addressed the problem of determining initial data distributions, or distributions for temporaries, or communication-free data allocation techniques, include [3, 4, 7, 8, 9].

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 [5, 6, 13, 14].

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 [5, 15]. 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 [15].

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 $1 \le i \le s$ and $1 \le j \le s-i+1$.
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. $T_{i,j} := \mathrm{MIN}_{1\le k<i}\{T_{i-k,\,k} + M_{i,\,j} + \mathit{cost}(P_{i-k,\,k},\, P_{i,\,j})\}$;
4. **end_for end_for**
5. $\mathit{Minimum\_Cost} := \mathrm{MIN}_{1\le k\le s}\{T_{s-k+1,\,k} + \mathit{loop\_carried\_dependence}(T_{s-k+1,\,k})\}$.

Note: $\mathit{cost}(P_{i-k,\,k},\, P_{i,\,j})$ returns the communication cost of changing data layouts from $P_{i-k,\,k}$ to $P_{i,\,j}$. $\mathit{loop\_carried\_dependence}(T_{s-k+1,\,k})$ 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 $P_{\alpha_1,\,\beta_1}$, $P_{\alpha_2,\,\beta_2}$, ..., and $P_{s-k+1,\,k}$ are used for computing $T_{s-k+1,\,k}$, then $\mathit{loop\_carried\_dependence}(T_{s-k+1,\,k})$ returns the communication cost of changing data layouts from $P_{s-k+1,\,k}$ to $P_{\alpha_1,\,\beta_1}$.

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}, ..., L_{i+j-1}, where $1 \le i,\ j \le i+j-1 \le s$.

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, ..., 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 $i+j-1 \le s$, 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 $1 \le i \le s$ and $1 \le j \le s$. $T_{i,j}$ 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, ..., 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 $1 \le i \le s$ and $1 \le j \le s$.
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. $T_{i,j} := \mathrm{MIN}_{1\le k<i}\{T_{i-k,\,k} + M_{i,\,j} + \mathit{cost}(P_{i-k,\,k},\, P_{i,\,j})\}$;
6.   **end_for end_for**
7. $\mathit{Minimum\_Cost}(m) := \mathrm{MIN}_{1\le k\le s}\{T_{s-k+1,\,k} + \mathit{loop\_carried\_dependence}(T_{s-k+1,\,k})\}$;
8.   /* Shuffle $M_{i,\,j}$ table. */
9.   **for** $j := 1$ to s **do**
10   **copy** $M_{1,j}$ to $TmpM_j$, **copy** $P_{1,j}$ to $TmpP_j$;
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** $TmpM_j$ to $M_{s,j}$, **copy** $TmpP_j$ to $P_{s,j}$;
18.   **copy** $M_{1,j}$ to $T_{1,j}$;
19. **end_for end_for**
20 $\mathit{Final\_Minimum\_Cost} := \mathrm{MIN}_{1\le m\le s}\{\mathit{Minimum\_Cost}(m)\}$.

The algorithm fragment from Line 3 to Line 7 is Algorithm 1; the algorithm fragment from Line 9 to Line 19 implements a data shift operation for table $M_{i,j}$. 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 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 a threshold value, 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 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 : *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 the following three cases are true for* $\gamma_i+1 < j \leq s-i+1$. (1) $M_{i,j} > M_{i,\beta} + M_{(i+\beta),(\gamma_i-\beta+1)} + M_{(i+\gamma_i+1),(j-\gamma_i-1)} + cost(P_{i,\beta}, P_{(i+\beta),(\gamma_i-\beta+1)}) + cost(P_{(i+\beta),(\gamma_i-\beta+1)}, P_{(i+\gamma_i+1),(j-\gamma_i-1)})$. (2) $T_{i,(\gamma_i+1)} > T_{(i+\beta),(\gamma_i-\beta+1)}$. *And,* (3) $T_{i,j} > T_{(i+\gamma_i+1),(j-\gamma_i-1)}$.

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, we can further prove that it is better to use several distribution schema to compute the sequence of Do-loops $L_{i-\alpha}$, $L_{i-\alpha+1}$, ..., L_{i+j-1}, than to use only one distribution scheme $P_{(i-\alpha),(j+\alpha)}$, for $1 \leq \alpha < i$ and $\gamma_i+1 \leq j \leq s-i+1$. 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 $1 \leq \alpha < i$ and $1 \leq \beta < \gamma_i+1 < j \leq s-i+1$. Therefore, we **need not** compute $T_{(i-\alpha),(j+\alpha)}$, for $1 \leq \alpha < i$ and $\gamma_i+1 \leq j \leq s-i+1$.

Theorem 2 : *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* $1 \leq \beta < \gamma_i+1$, *then the following four cases are true for* $1 \leq \alpha < i$ *and* $\gamma_i+1 < j \leq s-i+1$. (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) $T_{(i-\alpha),(\gamma_i+1+\alpha)} > T_{(i+\beta),(\gamma_i-\beta+1)}$. *And,* (4) $T_{(i-\alpha),(j+\alpha)} > T_{(i+\gamma_i+1),(j-\gamma_i-1)}$.

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 Theorem 1 and Theorem 2, 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 γ_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. $Minimum\_Cost := \mathrm{MIN}_{1 \leq k \leq \gamma}\{T_{s-k+1,k} + loop\_carried\_dependence(T_{s-k+1,k}),\ \text{if } k \leq \gamma_{s-k+1}\}$.

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 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 $s(\gamma+1)$.

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 $(i+\beta) \le s$, 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 $1 \le \beta \le \gamma_i \le s$. 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 $1 \le i \le s$. For example, $\gamma = \max_{1 \le i \le s}\{\gamma_i\}$.

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 $1 \le i \le s$ and $1 \le j \le \gamma_i$; $T_{1,j}$ $(= M_{1,j})$, where $1 \le j \le \gamma_1$; and γ.
Output: The cost of executing an iterative loop which contains s Do-loops on distributed memory computers.

$\Gamma_1 := (\gamma_1 + 1)$;
for $m := 1$ to Γ_1 **do**
  /* Compute Γ_1 sequences of Do-loops. */
  for $i := 2$ to s **do**
   for $j := 1$ to $\min\{\gamma_i,\, s-i+1\}$ **do**
    $T_{i,j} := \mathrm{MIN}_{1 \le k < \min\{i,\gamma+1\}}\{T_{i-k,k} + M_{i,j}$
      $+\ cost(P_{i-k,k},\, P_{i,j}),\ \text{if } k \le \gamma_{i-k}\}$;
  end_for end_for
  $Minimum\_Cost(m) := \mathrm{MIN}_{1 \le k \le \gamma}\{T_{s-k+1,k}$
    $+\ loop\_carried\_dependence(T_{s-k+1,k}),$
      $\text{if } k \le \gamma_{s-k+1}\}$;
  /* Shuffle $M_{i,j}$ table. */
  copy γ_1 to $Tmp\Gamma$;
  for $j := 1$ to γ_1 **do**
   copy $M_{1,j}$ to $TmpM_j$, **copy** $P_{1,j}$ to $TmpP_j$;
  end_for
  for $i := 2$ to s **do**
   copy γ_i to γ_{i-1};
   for $j := 1$ to γ_i **do**
    copy $M_{i,j}$ to $M_{i-1,j}$, **copy** $P_{i,j}$ to $P_{i-1,j}$;
  end_for end_for
  copy $Tmp\Gamma$ to γ_s;
  for $j := 1$ to γ_s **do**
   copy $TmpM_j$ to $M_{s,j}$, **copy** $TmpP_j$ to $P_{s,j}$;
  end_for
  for $j := 1$ to γ_1 **do**
   copy $M_{1,j}$ to $T_{1,j}$;
end_for end_for
$Final\_Minimum\_Cost :=$
  $\mathrm{MIN}_{1 \le m \le \Gamma_1}\{Minimum\_Cost(m)\}$.

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 ν Do-loops are immediately enclosed by an iterative loop, then this iterative loop is the parent node of these ν Do-loops, and these ν Do-loops are the ν corresponding child nodes of this iterative loop. For convenience, we will say that these ν Do-loops are the *first-level* Do-loops in the iterative loop; in addition, they are *siblings* in this tree representation.

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_σ, which contains a sequence of ω 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_σ;
4. construct $(M_{i,j})$-table, $(P_{i,j})$-table, γ_i, and γ for these $\psi(s)$ first-level Do-loops, where $1 \leq i \leq \psi(s)$, $1 \leq j \leq \gamma_i \leq \psi(s)$, and $\gamma = \max_{1 \leq i \leq \psi(s)}\{\gamma_i\}$;
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_σ, which contains a sequence of ω 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_σ;
4. construct $(M_{i,j})$-table, $(P_{i,j})$-table, γ_i, and γ for these $\psi(s)$ first-level Do-loops, where $1 \leq i \leq \psi(s)$, $1 \leq j \leq \gamma_i \leq \psi(s) - i + 1$, and $\gamma = \max_{1 \leq i \leq \psi(s)}\{\gamma_i\}$;
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 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, where $s = \sum_{i=1}^{\psi(s)} s_i$.

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

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, where $s = \sum_{i=1}^{\psi(s)} s_i$.

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

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

5 Conclusion remarks

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. 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.

Because of space limitation, in this presentation, we omit a complete example and two experimental studies on a 32-node nCUBE-II computer for explaining these five algorithms. The interested readers, however, can find these elsewhere [12].

References

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

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

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

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

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

[11] U. Kremer, J. Mellor-Crummey, K. Kennedy, and A. Carle. "Automatic data layout for distributed-memory machines in the D programming environment," *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.

[12] P.-Z. Lee. "Efficient Algorithms for Data Distribution on Distributed Memory Multicomputers," Technical Report TR-94-005, Institute of Information Science, Academia Sinica, 1994.

[13] P.-Z. Lee and T. B. Tsai. "Compiling efficient programs for tightly-coupled distributed memory computers," *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.

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

[15] 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.