# Communication-Free Data Allocation Techniques for Parallelizing Compilers on Multicomputers*

Tzung-Shi Chen and Jang-Ping Sheu

Department of Electrical Engineering, National Central University, Chung-Li 32054, Taiwan, R.O.C.
sheujp@ncuee.ncu.edu.tw

**Abstract** — In this paper, we devote our efforts to the techniques of allocating array elements of nested loops onto multicomputers in a communication-free fashion for parallelizing compilers. The arrays can be partitioned under the communication-free criteria with non-duplicate or duplicate data. In addition, the performance of the strategies with non-duplicate and duplicate array data is compared.

## 1. Introduction

For distributed memory multicomputers, the memory access time from a processor to its own local memory is much faster than the time to local memory of the other processors. An efficient parallel executing programs thus requires the goal of low communication overhead. To achieve this goal, various compiler techniques have, therefore, been developed to reduce communication traffic on multicomputers. The purpose of exploiting a large amount of parallelism in sequential programs has been the previous focus of a number of researchers [13] [14]. However, exploiting a large amount of parallelism in sequential programs may not promise that the parallelized programs for parallel execution can obtain more efficiency on multicomputers. The main reason is that those extracted parallelism may possibly cause more communication overhead during parallel execution. Under the above considerations, several researchers developed parallelizing compilers in which programmers must explicitly specify data allocation and the codes could then be generated with appropriate communication constructs [1] [6].

Achieving automatic data management in designing parallelizing compilers is, nevertheless, difficult since the data must be attentively distributed so that communication traffic is minimized in parallel execution of programs. Therefore, several researchers [3] [4] [7] [12] focus the data allocation problem on automatically allocating the data or restructuring the programs in order to improve the efficiency of usage of memory hierarchy or reduce the interprocessor communication overhead in parallel machines. For distributed memory multicomputers, large amounts of communication overhead may cause the poor performance during parallel execution of programs. Some researchers, such as King, Chou and Ni [5], Ramanaujam and Sadayappen [9], and Sheu and Tai [11], studied the problems of transforming programs into the parallel form and reducing the interprocessor communication overhead. Furthermore, Ramanaujam and Sadayappen [10] focused on analyzing the For-all loops and partitioning these loops and the corresponding data such that the partitioned programs are executed without communication overhead in the distributed memory multicomputers.

In this paper, we concentrate on automatically allocating the array elements of nested loops with uniformly generated references [3] on distributed memory multicomputers. First, we analyze the pattern of references among all arrays referenced by a nested loop, and derive the sufficient conditions for communication-free partitioning of arrays. Two communication-free partitioning strategies, non-duplicate data and duplicate data, will be proposed. Our method can obtain more parallelism than the method proposed by Ramanaujam and Sadayappen [10] in For-all loops with uniformly generated references. Finally, the performance of the data allocation with non-duplicate and duplicate data strategies is discussed.

* This work was supported by the National Science Council of the Republic of China under grant NSC 82-0408-E-008-010.

## 2. Basic Concepts and Assumptions

A normalized $n$-nested loop [14] is considered in this paper. Let $\mathbf{Z}$ and $\mathbf{R}$ denote the set of integers and the set of real numbers, respectively. The symbols $\mathbf{Z}^n$ and $\mathbf{R}^n$ represent the set of $n$-tuple of integers and the set of $n$-tuple of real numbers, respectively. The *iteration space* [14] of an $n$-nested loop is a subset of $\mathbf{Z}^n$ and is defined as $I^n = \{(I_1, I_2, \ldots, I_n) \mid l_j \leq I_j \leq u_j, \text{ for } 1 \leq j \leq n\}$. The vector $\vec{i} = (i_1, i_2, \ldots, i_n)$ in $I^n$ is represented as an iteration of the nested loop. In the nested loop, there may exist *input*, *output*, *flow dependences* or *antidependence* [8] which are referred to as *data dependence* in the following discussions. Let the linear function $h : \mathbf{Z}^n \longrightarrow \mathbf{Z}^d$ be defined as a *reference function* $h(I_1, \ldots, I_n) = (a_{1,1}I_1 + \cdots + a_{1,n}I_n, \ldots, a_{d,1}I_1 + \cdots + a_{d,n}I_n)$ and be represented by the matrix

$$H = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \vdots & \vdots \\ a_{d,1} & \cdots & a_{d,n} \end{bmatrix}_{d \times n}$$

where $a_{i,j} \in \mathbf{Z}$, for $1 \leq i \leq d$ and $1 \leq j \leq n$. In the loop body, a $d$-dimensional array element $A[h(i_1, i_2, \ldots, i_n) + \bar{c}]$ may be referenced by the reference function $h$ at iteration $(i_1, i_2, \ldots, i_n)$ in $I^n$, where $\bar{c}$ is known as the constant offset vector in $\mathbf{Z}^d$ [12]. The *data space* of array $A$ is a subset of $\mathbf{Z}^d$ and is defined over the user-defined array subscript index set. For array $A$, all $s$ referenced array variables $A[H_p\vec{i} + \bar{c}_p]$, for $1 \leq p \leq s$, are called *uniformly generated references* [3] [12] if $H_1 = H_2 = \cdots = H_s$ where $H_p$ is the linear transformation function from $\mathbf{Z}^n$ to $\mathbf{Z}^d$, $\vec{i} \in I^n$, and $\bar{c}_p$ is the constant offset vector in $\mathbf{Z}^d$. Since little exploitable data dependence exists between nonuniformly generated references, we focus the data allocation to each array on the same reference function in a nested loop. The different arrays may have different reference functions.

**Example 1:** Consider a 2-nested loop $L1$.

```
for i = 1 to 4
    for j = 1 to 4
        S1 : A[2i, j] := C[i, j] * 7 ;
        S2 : B[j, i + 1] := A[2i - 2, j - 1] + C[i - 1, j - 1] ;
    end
end                                          (L1)
```

In this example, the iteration space is $I^2 = \{ (i, j) \mid 1 \leq i, j \leq 4 \}$. In loop $L1$ with three arrays $A$, $B$, and $C$, the respective reference functions are

$$H_A = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}, \; H_B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \text{and } H_C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

A flow dependence exists between the variables $A[2i, j]$ at statement $S_1$ and $A[2i - 2, j - 1]$ at statement $S_2$ with the different offset vectors $(0, 0)$ and $(-2, -1)$, respectively. For array $C$ only read by loop $L1$, an input dependence exists between the variables $C[i, j]$ at statement $S_1$ and $C[i - 1, j - 1]$ at statement $S_2$ with the different offset vectors $(0, 0)$ and $(-1, -1)$, respectively. The array variable $B[j, i + 1]$ is only generated at statement $S_2$ and its offset vector is $(0, 1)$. Loop $L1$ thus has the uniformly



Proceedings of the International Conference on Parallel Processing (ICPP'93)
0-8493-8983-6/0-7 $20.00 © 1993 IEEE

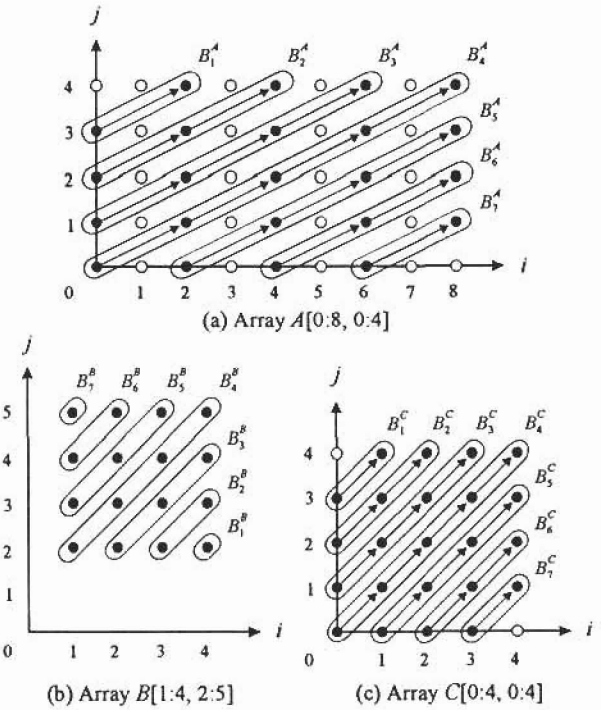



Figure 1. Partitioning arrays $A$, $B$ and $C$ of loop $L1$ into their corresponding data blocks.

generated references on arrays $A$, $B$, and $C$.

□

**Definition 1: [Data referenced vector]**
In an $n$-nested loop $L$ with uniformly generated references, if there exist two referenced array variables $A[H\vec{i} + \bar{c}_1]$ and $A[H\vec{i} + \bar{c}_2]$ for array $A$, then the vector $\bar{r} = \bar{c}_1 - \bar{c}_2$ is called *data referenced vector* of array $A$.

□

The data referenced vector $\bar{r}$ represents the difference between two array elements $A[H\vec{i} + \bar{c}_1]$ and $A[H\vec{i} + \bar{c}_2]$ which are referenced by an iteration $\vec{i}$. Note that any data dependence in loop $L$ exists between two distinct referenced array variables $A[H\vec{i}+\bar{c}_1]$ and $A[H\vec{i}+\bar{c}_2]$, i.e., two iterations $\vec{i}_1$ and $\vec{i}_2$ can reference the same array element, if and only if $H\vec{i}_1 + \bar{c}_1 = H\vec{i}_2 + \bar{c}_2$, i.e., $H(\vec{i}_2 - \vec{i}_1) = \bar{r}$. Communication overhead is therefore not to be incurred if the iteration space is partitioned along the direction $\vec{i}_2 - \vec{i}_1$ into iteration blocks and the data space of array $A$ is partitioned along the direction $\bar{r}$ into data blocks.

Example 1 is considered here for illustrating the ideas of communication-free data allocation strategy. The arrays $A$, $B$, and $C$ of loop $L1$ have the referenced array variables $A[2i, j]$, $A[2i-2, j-1]$, $B[j, i+1]$, and $C[i, j]$, $C[i-1, j-1]$, respectively. The data referenced vectors of arrays $A$ and $C$ are $\bar{r}_1 = (2, 1)$ and $\bar{r}_2 = (1, 1)$, respectively. However, only one referenced array variable exists on array $B$; namely, no data referenced vector exists. All of the data spaces of arrays $A$, $B$, and $C$ and their data referenced vectors of each array element are shown in Figure 1(a), 1(b) and 1(c), respectively, where solid points represent that array elements are used in loop $L1$ and, however, empty points are not. At iteration $(1, 1)$, the array element $A[2, 1]$ is generated by $S_1$ and $A[0, 0]$ is used in $S_2$. Then, at iteration $(2, 2)$, the array element $A[4, 2]$ is generated by $S_1$ and $A[2, 1]$ is used in $S_2$, and so on. Restated, two iterations $\vec{i}_1 = (1, 1)$ and $\vec{i}_2 = (2, 2)$ satisfying the condition $H_A(\vec{i}_2 - \vec{i}_1) = \bar{r}_1$ can access the same array element $A[2, 1]$. The data space of array $A$ is therefore partitioned along the data referenced vector $(2, 1)$ into the data blocks $B_j^A$ for $1 \le j \le 7$, enclosing the points with lines, as shown in Figure 1(a). These used and generated array elements grouped in the same data block are then to be allocated to the same processor. Similarly, the array $C$ is also

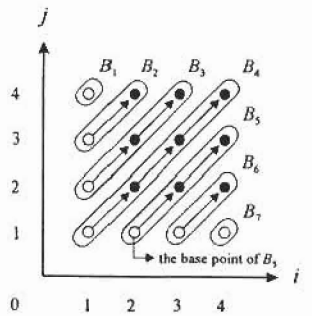



Figure 2. Partitioning the iteration space of loop $L1$ into the corresponding iteration blocks.

partitioned along data referenced vector $(1, 1)$ into their corresponding data blocks, $B_j^C$ for $1 \le j \le 7$, as shown in Figure 1(c). It is easy to show that if the iteration space is partitioned along the direction $(1, 1)$ as shown in Figure 2, no inter-block communication exists for arrays $A$ and $C$. Therefore, array $B$ must be partitioned along the direction $(1, 1)$ into the corresponding data blocks $B_j^B$, $1 \le j \le 7$, as shown in Figure 1(b), such that the partitioned iteration blocks $B_j$, $1 \le j \le 7$, can be executed in parallel without inter-block communication.

## 3. Communication-free Array Partitioning

### 3.1 Communication-free Array Partitioning without Duplicate Data

In this subsection, we will discuss the communication-free array partitioning without duplicate data; i.e., exactly one copy of each array element exists during execution of program.

Given an $n$-nested loop $L$, the problem is how to partition the data referenced in loop $L$ such that not only the communication overhead is not necessary but also the degree of parallelism can be extracted as large as possible. We first analyze the relations among all array variables of loop $L$ and then partition the iteration space into iteration blocks such that no inter-block communication exists. For each partitioned iteration block, all data, referenced by those iterations, must be grouped into their corresponding data block for each array. Our methods proposed in this paper can make the size of partitioned iteration blocks as small as possible so as to achieve higher degree of parallelism.

From the definition of a vector space, an $n$-dimensional vector space $V$ over $\mathbf{R}$ can be generated using exactly $n$ linearly independent vectors. Let $X$ be a set of $p$ linearly independent vectors, where $p \le n$. These $p$ vectors form a basis of a $p$-dimensional subspace, denoted by span$(X)$, of $V$ over $\mathbf{R}$. The dimension of a vector space $V$ is denoted by dim$(V)$. In the following, a formal definition of partitioning of iteration space is given.

**Definition 2: [Iteration partition]**
The *iteration partition* of an $n$-nested loop $L$ partitioned by the space $\Psi = \text{span}(\{\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_u\})$ where $\bar{t}_l \in \mathbf{R}^n$, $1 \le l \le u$, denoted as $P_\Psi(I^n)$, is to partition the iteration space $I^n$ into disjoint iteration blocks $B_1, B_2, \ldots, B_q$ where $q$ is the total number of partitioned blocks. For each iteration block $B_j$, $1 \le j \le q$, a base point $\bar{b}_j \in \mathbf{R}^n$ exists and

$$B_j = \{\vec{i} \in I^n | \vec{i} = \bar{b}_j + a_1\bar{t}_1 + a_2\bar{t}_2 + \cdots + a_u\bar{t}_u, a_l \in \mathbf{R}, 1 \le l \le u\}$$

where $I^n = \bigcup_{1 \le j \le q} B_j$.

□

**Definition 3: [Data partition]**
Given an iteration partition $P_\Psi(I^n)$, the *data partition* of array $A$ with all $s$ referenced array variables $A[H_A\vec{i} + \bar{c}_1]$, ..., $A[H_A\vec{i} + \bar{c}_s]$, denoted as $P_\Psi(A)$, is the partition of data space of array $A$ into $q$ data blocks $B_1^A, B_2^A, \ldots, B_q^A$. For each data block $B_j^A$ corresponding to one iteration block $B_j$ of $P_\Psi(I^n)$, $1 \le j \le q$,

$$B_j^A = \{A[\bar{a}] | \bar{a} = H_A\vec{i} + \bar{c}_l, \vec{i} \in B_j, 1 \le l \le s\}.$$



Proceedings of the International Conference on Parallel Processing (ICPP'93)
0-8493-8983-6/0-7 $20.00 © 1993 IEEE

□

Consider Example 1. If $\Psi = \text{span}(\{(1,1)\})$ is chosen as the space of the iteration partition $P_\Psi(I^2)$ in loop $L1$, the iteration space can be partitioned into seven iteration blocks as shown in Figure 2. The points enclosed by a line are shown in Figure 2 to form an iteration block and those dotted points represent the base points of the corresponding iteration blocks. For example, the base point $\bar{b}_5$ of iteration block $B_5 = \{\bar{i} \in I^2 | \bar{i} = \bar{b}_5 + a(1,1), 0 \le a \le 2\}$ is $(2,1)$. Based on the iteration partition $P_\Psi(I^2)$, the arrays $A$, $B$, and $C$ are partitioned into the corresponding data blocks by using the respective data partition $P_\Psi(A)$, $P_\Psi(B)$, and $P_\Psi(C)$ as shown in Figure 1.

**Example 2:** Consider a 2-nested loop $L2$.

```
for i = 1 to 4
    for j = 1 to 4
        S1: A[i + j, i + j] := B[2i, j] * A[i + j - 1, i + j] ;
        S2: A[i + j - 1, i + j - 1] := B[2i - 1, j - 1]/3 ;
    end
end                                          (L2)
```

In loop $L2$, the respective reference functions of arrays $A$ and $B$ are

$$H_A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \text{ and } H_B = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}.$$

The data referenced vectors $\bar{r}_1$, between $A[i+j, i+j]$ and $A[i+j-1, i+j-1]$, $\bar{r}_2$, between $A[i+j-1, i+j-1]$ and $A[i+j-1, i+j]$, and $\bar{r}_3$, between $A[i+j-1, i+j]$ and $A[i+j, i+j]$, of array $A$ are $(1,1)$, $(0,-1)$, and $(-1,0)$, respectively. The data referenced vector $\bar{r}_4$ of array $B$ is $(1,1)$. Consider the equation $H_A\bar{t}_2 = \bar{r}_2$. Two iterations $\bar{i}_1$ and $\bar{i}_2$ can access the same element of array $A$ if the equation $\bar{i}_2 - \bar{i}_1 = \bar{t}_2$ is satisfied. Because no solution exists in the equation $H_A\bar{t}_2 = \bar{r}_2$, no data dependence exists between $A[i+j-1, i+j-1]$ and $A[i+j-1, i+j]$. However, solving the equation $H_B\bar{t}_4 = \bar{r}_4$ can exactly obtain a solution $\bar{t}_4 = (\frac{1}{2}, 1)$. It is impossible for the data dependence vector $\bar{t}_4$ between two iterations since $\bar{t}_4$ does not belong to $\mathbf{Z}^2$. Also no data dependence exists on array $B$. Let the symbol $0^d \in \mathbf{Z}^d$ be denoted as a zero-vector where each component is equal to 0. Consider the equation $H\bar{t} = \bar{r}$. In the special case, when $\bar{r} = 0^d$, the set of solutions $\bar{t}$ of equation $H\bar{t} = 0^d$ is $\text{Ker}(H)$, the null space of $H$. The vector $\bar{t}$ indicates the difference of two iterations accessing the same element of a certain array variable. For example, $\text{Ker}(H_A)$ is $\text{span}(\{(1,-1)\})$ in loop $L2$. On variable $A[i+j, i+j]$, the array element $A[4,4]$, referenced by the iteration $(1,3)$, can be referenced again by iterations $(1,3) + \text{span}(\{(1,-1)\})$, i.e., $(2,2)$ and $(3,1)$, of loop $L2$.

□

In the following, how to choose the better space to partition the iteration space and data spaces without duplicate data is discussed.

**Definition 4: [Reference space]**

In an $n$-nested loop $L$, if a reference function $H_A$ and $s$ variables $A[H_A\bar{i} + \bar{c}_1]$, ..., $A[H_A\bar{i} + \bar{c}_s]$ for array $A$ exist, and the data referenced vectors are $\bar{r}_p = \bar{c}_j - \bar{c}_k$ for all $1 \le j < k \le s$ and $1 \le p \le \frac{s(s-1)}{2}$, then the *reference space* of array $A$ is

$$\Psi_A = \text{span}(\beta \cup \{\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_{\frac{s(s-1)}{2}}\})$$

where $\beta$ is the basis of $\text{Ker}(H_A)$ and $\bar{t}_j \in \mathbf{R}^n$, $1 \le j \le \frac{s(s-1)}{2}$, must satisfy the following conditions

(1) $\bar{t}_j$ is a particular solution of equation $H_A\bar{t} = \bar{r}_j$ and

(2) a solution $\bar{t}' \in \bar{t}_j + \text{Ker}(H_A)$ exists such that $\bar{t}' \in \mathbf{Z}^n$ and $\bar{t}' = \bar{i}_2 - \bar{i}_1$ where $\bar{i}_1, \bar{i}_2 \in I^n$.

□

The *reference space* used here is similar to the *group-temporal reuse vector space* previously defined by Wolf and Lam [12]. The *reference space* represents the relations of all data references between iterations. For array $A$, no data dependence exists between iteration blocks when the iteration space $I^n$ is partitioned with the *reference space* $\Psi_A$. This is because all data dependences are considered in $\Psi_A$ such that data accesses do not need between iteration blocks. In each iteration block, iterations according to the *lexicographical order* [14] are executed so as to preserve the dependency in loop. In loop $L2$, the *reference space* $\Psi_A$ of array $A$ is $\text{span}(\{(1,-1), (\frac{1}{2}, \frac{1}{2})\})$ because $\text{Ker}(H_A) = \text{span}(\{(1,-1)\})$ and only a particular solution $\bar{t}_1 = (\frac{1}{2}, \frac{1}{2})$ of equation $H_A\bar{t} = \bar{r}_1$ exists which satisfies the conditions (1) and (2) in Definition 4. The *reference space* $\Psi_B$ of array $B$ is $\text{span}(\phi)$ because $\text{Ker}(H_B) = \{0^2\}$ and the only solution $\bar{t}_4 = (\frac{1}{2}, 1) \notin \mathbf{Z}^2$ not satisfying the condition (2) in Definition 4.

**Theorem 1:**

Given an $n$-nested loop $L$ with $k$ array variables, let the *reference space* $\Psi_{A_j}$ be $\text{span}(X_j)$ of each array $A_j$ for $1 \le j \le k$. If $\Psi = \text{span}(X_1 \cup X_2 \cup \cdots \cup X_k)$, then $\Psi$ is the *partitioning space* for communication-free partitioning of arrays $A_j$ for $1 \le j \le k$ without duplicate data by using the iteration partition $P_\Psi(I^n)$.

**Proof:** The proof of this theorem can refer to [2].

□

By Theorem 1, when $\dim(\Psi) < n$, this means that the iteration partition $P_\Psi(I^n)$ exists more parallelism in loop $L$. By Definition 2, the smaller the value of $\dim(\Psi)$ is, the higher the degree of parallelism has. In general, when $\dim(\Psi) < n-1$, our method can exploit more parallelism than Ramanaujam and Sadayappen's method [10] in For-all loops with uniformly generated references. This is because Ramanaujam and Sadayappen's method only uses $(n-1)$-dimensional hyperplanes to partition the arrays in For-all loops. Consider loop $L1$. The *reference spaces* are $\Psi_A = \Psi_C = \text{span}(\{(1,1)\})$, and $\Psi_B = \{0^2\}$ for respective arrays $A$, $C$, and $B$. Therefore, by Theorem 1 the *partitioning space* is $\Psi = \text{span}(\{(1,1)\} \cup \{(1,1)\} \cup \phi)$ for communication-free iteration partition $P_\Psi(I^2)$ of loop $L1$. Due to $\dim(\Psi) = 1 (< 2)$, large amounts of parallelism exists in loop $L1$. The overall results of partitioned data and iteration blocks in loop $L1$ have been respectively shown in Figure 1 and Figure 2. Since loop $L1$ is not a For-all loop, Ramanaujam and Sadayappen's method can not solve it.

### 3.2 Communication-free Array Partitioning with Duplicate Data

In this subsection, we consider the communication-free array partitioning with duplicate data; i.e., there may exist more than one copy of an array element allocated onto local memory of processors. Due to communication overhead being most time-consuming in parallel executing programs, it is worthwhile to duplicate referenced data onto processors such that high degree of parallelism can be exploited and meanwhile the computations should be correctly performed in a communication-free fashion. Duplicate data strategy, in comparison with non-duplicate one, may extract more parallelism of programs based on communication-free array partitioning. In the following definition, two kinds of arrays are classified.

**Definition 5: [Fully and partially duplicable arrays]**

If any flow dependence does not exist on an array $A$, then the array $A$ is called *fully duplicable array*; otherwise, the array $A$ is called *partially duplicable array*.

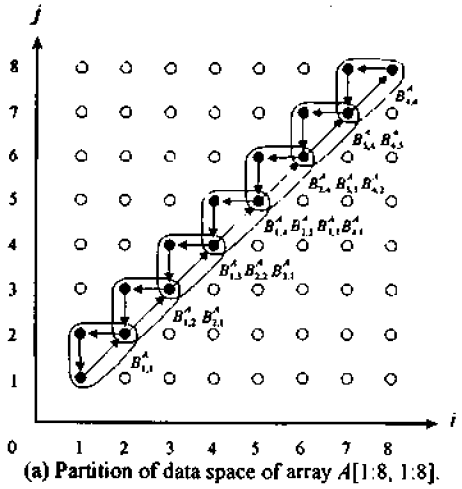
□

For the two kinds of arrays, how to choose the better space to partition the iteration space and arrays with duplicate data such that no inter-block communication exists is discussed as follows.

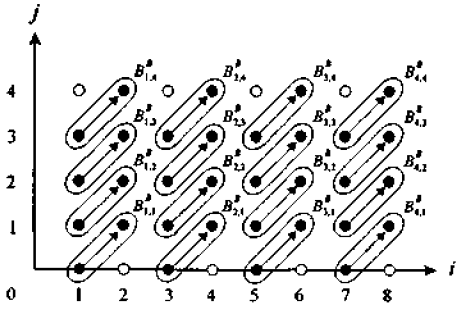
First, we examine the fully duplicable arrays. Because no flow dependence exists on array $A$, any iteration will not use the elements of array $A$ generated by other iterations; therefore, the data can be arbitrarily distributed onto various processors with duplicating the elements of array $A$ and the semantic of original loop can be retained. Therefore, the *reference space* $\Psi_A$ can be reduced into $\text{span}(\phi)$ denoted as the *reduced reference space* $\Psi^r_A$. That is, $\Psi^r_A$ is the subspace of $\Psi_A$. Next, the partially duplicable arrays are to be examined. Assume there exist $p$ flow dependences on a partially duplicable array $A$ in loop $L$. The *reference space* $\Psi_A$ of array $A$ can be reduced into the *reduced reference space* $\Psi^r_A = (\beta \cup \{\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_p\})$ where $\beta$ is the basis of $\text{Ker}(H_A)$ and $\bar{t}_j$, $1 \le j \le p$, which lead to flow dependences are particular solutions satisfying the conditions (1) and (2) in Definition 4. The reducible reason for the *reference space* is that only the flow dependences can actually cause the data transfer between execution of iterations. That is, only flow dependence is necessary to be considered during execution of



Proceedings of the International Conference on Parallel Processing (ICPP'93)
0-8493-8983-6/0-7 $20.00 © 1993 IEEE



(a) Partition of data space of array $A[1{:}8, 1{:}8]$.



(b) Partition of data space of array $B[1{:}8, 0{:}4]$.

Figure 3. Partition of arrays $A$ and $B$ in loop $L2$ using the data partition $P_{\Psi^r}(A)$ and $P_{\Psi^r}(B)$, respectively.

programs; however, input, output dependences and antidependence merely determine the precedence of executing iterations so that they can not make any data transfer.

While partitioning the iteration space, data references which occur among all arrays in a nested loop must be considered. Given an $n$-nested loop $L$ with $k$ array variables, assume the *reduced reference space* $\Psi^r_{A_j} = \text{span}(X^r_j)$ of each either fully or partially duplicable array $A_j$ exists, $1 \le j \le k$. Then, $\Psi^r = \text{span}(X^r_1 \cup X^r_2 \cup \cdots \cup X^r_k)$ is the *partitioning space* for communication-free partitioning with duplicate data by using the iteration partition $P_{\Psi^r}(I^n)$.

Consider Example 2. By Theorem 1, while applying the iteration partition $P_\Psi(I^2)$ to loop $L2$ where $\Psi = \text{span}(\{(1,-1), (\frac{1}{2},\frac{1}{2})\})$, loop $L2$ needs to be executed sequentially based on the non-duplicate data strategy. Due to both arrays $A$ and $B$ in loop $L2$ being fully duplicable arrays, the *partitioning space* $\Psi^r$ is $\text{span}(\phi)$. While applying the iteration partition $P_{\Psi^r}(I^2)$ to loop $L2$, loop $L2$ can be executed in fully parallel. Clearly, using duplicate data strategy can obtain more parallelism than using non-duplicate one in loop $L2$. By duplicate data strategy, the overall results of partitioned data and iteration blocks in loop $L2$ are respectively shown in Figure 3 and Figure 4 where the relations of output dependence are omitted.

## 4. Performance Evaluation

In this section, we compare the performance of non-duplicate and duplicate data strategies. Consider the matrix multiplication algorithm.

```
for i = 1 to M
    for j = 1 to M
        for k = 1 to M
            C[i,j] := C[i,j] + A[i,k] * B[k,j] ;      (L3)
        end
    end
end
```

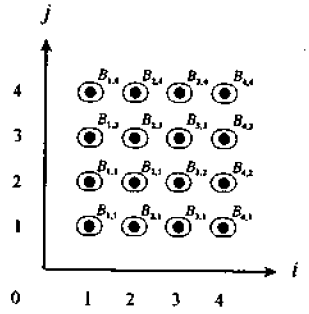


Figure 4. Partition of iteration space of loop $L2$ using the iteration partition $P_{\Psi^r}(I^2)$.

For arrays $A$, $B$, and $C$, the respective *reference spaces* $\Psi_A = \text{span}(\{(0,1,0)\})$, $\Psi_B = \text{span}(\{(1,0,0)\})$, and $\Psi_C = \text{span}(\{(0,0,1)\})$. By Theorem 1, the *partitioning space* $\Psi$ is $\text{span}(\{(0,1,0)\} \cup \{(1,0,0)\} \cup \{(0,0,1)\})$. That is, the matrix multiplication algorithm needs to be executed sequentially while using the non-duplicate data strategy.

Next considered is that if only some of fully or partially duplicable arrays are duplicated, there may sacrifice little parallelism than all of them. Note that both arrays $A$ and $B$ are fully duplicable arrays and array $C$ is a partially duplicable array. Thus, the *reduced reference spaces* $\Psi^r_A = \text{span}(\phi)$, $\Psi^r_B = \text{span}(\phi)$, and $\Psi^r_C = \text{span}(\{(0,0,1)\})$ for respective arrays $A$, $B$, and $C$. Demonstrated in the following is that only the array $B$ is duplicated in loop $L3$. Due to not replicating data of array $A$, let $\Psi' = \text{span}(\{(0,1,0)\} \cup \{(0,0,1)\})$ such that the communication-free iteration partition $P_{\Psi'}(I^3)$ can be obtained. Consider a $p_1 \times p_2$ mesh multicomputer as the target machine where the number of processors is $p = p_1 \times p_2$. Assume $\sqrt{p} = p_1 = p_2$, and $M$ is a multiple of $p$. The processor $PE_a$ for $0 \le a \le p-1$ will execute the following loop $L3'$ by our program transformation and processor assignment strategies [2].

```
forall i = (1 + (a - 1) mod p) to M step p
    for j = 1 to M
        for k = 1 to M
            C[i,j] := C[i,j] + A[i,k] * B[k,j] ;      (L3')
        end
    end
end-forall
```

Because we do not replicate the data of array $A$ to each processor, the whole array $B$ must be duplicated to each processor for parallel execution without inter-processor communication. Because the processor $PE_a$, $0 \le a \le p-1$, requires accessing the array elements

$$A[\alpha, 1:M], \text{ for } \alpha = (1 + (a-1) \bmod p) + lp, l \in \mathbf{Z}, 1 \le \alpha \le M,$$

the host processor must send these data to the corresponding processor in a pipelined fashion. In addition, because all processors require accessing the same array elements $B[1:M, 1:M]$, the host processor must broadcast the whole array $B$ to each node processor. Nevertheless, if only the array $A$, not array $B$, is duplicated, the similar results can be obtained.

In the following, both arrays $A$ and $B$ in loop $L3$ are to be duplicated. Thus the communication-free iteration partition $P_{\Psi''}(I^3)$ can be obtained, where the *partitioning space* $\Psi'' = \text{span}(\{(0,0,1)\})$. By our program transformation and processor assignment strategies [2], the following results can thus be obtained. The processor $PE_{a_1,a_2}$ for $0 \le a_1 \le p_1 - 1$ and $0 \le a_2 \le p_2 - 1$ is to execute the following loop $L3''$.

```
forall i = (1 + (a1 - 1) mod p1) to M step p1
    forall j = (1 + (a2 - 1) mod p2) to M step p2
        for k = 1 to M
            C[i,j] := C[i,j] + A[i,k] * B[k,j] ;      (L3'')
        end
    end
end-forall
```



Proceedings of the International Conference on Parallel Processing (ICPP'93)
0-8493-8983-6/0-7 $20.00 © 1993 IEEE

Table I. Execution time of loops $L3$, $L3'$, and $L3''$. (unit: second)

| Number of processors | Loop | Problem size ($M$) | | | | |
|---|---|---|---|---|---|---|
| | | 16 | 32 | 64 | 128 | 256 |
| $p=1$ | $L3$ | 0.0399 | 0.3162 | 2.5241 | 20.1691 | 161.2546 |
| $p=4$ | $L3'$ | 0.0144 | 0.0956 | 0.6961 | 5.2895 | 41.3058 |
| | $L3''$ | 0.0127 | 0.0855 | 0.6467 | 5.1405 | 40.7988 |
| $p=16$ | $L3'$ | 0.0135 | 0.0543 | 0.2869 | 1.7908 | 12.3584 |
| | $L3''$ | 0.0080 | 0.0326 | 0.2043 | 1.4326 | 10.6513 |

Table II. Speedup of loops $L3'$ and $L3''$.

| Number of processors | Loop | Problem size ($M$) | | | | |
|---|---|---|---|---|---|---|
| | | 16 | 32 | 64 | 128 | 256 |
| $p=4$ | $L3'$ | 2.77 | 3.31 | 3.63 | 3.81 | 3.89 |
| | $L3''$ | 3.14 | 3.70 | 3.90 | 3.92 | 3.95 |
| $p=16$ | $L3'$ | 2.96 | 5.82 | 8.80 | 11.26 | 13.05 |
| | $L3''$ | 4.99 | 9.70 | 12.35 | 14.08 | 15.14 |

```
        end
    end-forall
end-forall
```

Assume $M$ is a multiple of $\sqrt{p}$. Because the processors $PE_{a_1,a_2}$, $0 \le a_1 \le \sqrt{p}-1$, require accessing the same array elements

$$A[\alpha, 1:M], \text{ for } \alpha = (1+(a_2-1) \bmod \sqrt{p}) + l\sqrt{p}, \quad l \in \mathbf{Z}, 1 \le \alpha \le M,$$

for $0 \le a_2 \le \sqrt{p}-1$, the host processor must send the same data to the corresponding row processors by multicasting in a pipelined fashion. Similarly, because the processors $PE_{a_1,a_2}$, $0 \le a_2 \le \sqrt{p}-1$, require accessing the same array elements

$$B[1:M, \alpha], \text{ for } \alpha = (1+(a_1-1) \bmod \sqrt{p}) + l\sqrt{p}, \quad l \in \mathbf{Z}, 1 \le \alpha \le M,$$

for $0 \le a_1 \le \sqrt{p}-1$, the host processor must send the same data to the corresponding column processors by multicasting in a pipelined fashion. Because of only replicating the partial data of both arrays $A$ and $B$ to processors for loop $L3''$, the communication cost of distributing the initial data to each processor is less than that of loop $L3'$.

The overall execution results for loops $L3$, $L3'$, and $L3''$ are simulated on Transputer multicomputers with 16 processors are shown in Table I and Table II. The execution time of loops $L3$, $L3'$, and $L3''$ are illustrated in Table I with problem sizes $M$ = 16, 32, 64, 128 and 256. The speedup derived from Table I is illustrated in Table II. When the number of processors is equal to 1, we only consider the computation time not including the time of allocating arrays $A$ and $B$. Although duplicating data seems to waste the time of allocating initial data, it can increase great amounts of parallelism and incur no communication overhead during parallel execution of programs. Therefore, the time of parallel execution is less than that of sequential execution as shown in Table I. However, because data locality in loop $L3$ is not exploited during sequential execution, the speedup becomes more and more better whenever the problem size becomes more and more larger as shown in Table II. This implies that exploiting data locality is also important during program execution in each processor [12]. Due to existing large amounts of communication overhead in loop $L3'$ as distributing whole array $B$, the speedup of loop $L3''$ is more efficient than that of loop $L3'$. By the above analysis, the communication time of distributing the initial referenced elements of arrays must be as small as possible in order to obtain better efficiency during parallel execution. In addition, which kind of duplication of arrays is suitable for replicating their referenced data can be appropriately estimated such that parallelized programs can gain better performance during parallel execution.

## 5. Conclusions

Two automatic array partitioning strategies, non-duplicate and duplicate data, have been proposed in this paper such that no data transfer during parallel execution is incurred and the parallelism of nested loops can be exploited as large as possible. Under the duplicate data strategy, more parallelism can be extracted than non-duplicate one. By the matrix multiplication algorithm, the performance of the strategies with non-duplicate and duplicate data is discussed, and the overall results are simulated on Transputer multicomputers. By our analysis of performance, obtaining the better efficiency of executing programs is dependent on the extracted parallelism and the communication overhead of distributing the initial data under the communication-free criteria.

## References


[1] D. Callahan and K. Kennedy, "Compiling Programs for Distributed-Memory Multiprocessors," *The Journal of Supercomputing*, Vol. 2, pp. 151-169, Oct. 1988.

[2] T. S. Chen and J. P. Sheu, "Communication-free Data Allocation Techniques for Parallelizing Compilers on Multicomputers," Technique Report, Department of Electrical Engineering, National Central University, Taiwan, R.O.C., October 1992.

[3] D. Gannon, W. Jalby and J. Gallivan, "Strategies for Cache and Local Memory Management by Global Program Transformations," *Journal of Parallel and Distributed Computing*, Vol. 5, No. 5, pp. 587-616, Oct. 1988.

[4] M. Gupta and P. Banerjee, "Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 2, pp. 179-193, March 1992.

[5] C. T. King, W. H. Chou and L. M. Ni, "Pipelined Data-Parallel Algorithms: Part II-Design," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 4, pp. 486-499, Oct. 1990.

[6] C. Koelbel and P. Mehrotra, "Compiling Global Name-Space Parallel Loops for Distributed Execution," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 440-451, Oct. 1991.

[7] M. Lu and J. Z. Fang, "A Solution of the Cache Ping-Pong Problem in Multiprocessor Systems," *Journal of Parallel and Distributed Computing*, pp. 158-171, October 1992.

[8] D. A. Padua and M. J. Wolfe, "Advanced Compiler Optimizations for Supercomputers," *Communication of ACM*, pp. 1184-1201, Dec. 1986.

[9] J. Ramanujam and P. Sadayappan, "A Methodology for Parallelizing Programs for Multicomputers and Complex Memory Multiprocessors," *Proceedings of ACM International Conference on Supercomputing*, pp. 637-646, 1989.

[10] J. Ramanujam and P. Sadayappan, "Compile-Time Techniques for Data Distribution in Distributed Memory Machines," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 472-482, Oct. 1991.

[11] J. P. Sheu and T. H. Tai, "Partitioning and Mapping Nested Loops on Multiprocessor Systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 430-439, Oct. 1991.

[12] M. E. Wolf and M. S. Lam, "A Data Locality Optimizing Algorithm," *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation*, pp. 30-44, June 1991.

[13] M. E. Wolf and M. S. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 452-471, Oct. 1991.

[14] M. J. Wolfe, "Optimizing Supercompilers for Supercomputers," London and Cambridge, MA: Pitman and the MIT Press, 1989.




Proceedings of the International Conference on Parallel Processing (ICPP'93)
0-8493-8983-6/0-7 $20.00 © 1993IEEE