A Basic-Cycle Calculation Technique for Efficient Dynamic Data Redistribution

Yeh-Ching Chung, Ching-Hsien Hsu, and Sheng-Wen Bai

Abstract—Array redistribution is usually required to enhance algorithm performance in many parallel programs on distributed memory multicomputers. Since it is performed at run-time, there is a performance trade-off between the efficiency of the new data decomposition for a subsequent phase of an algorithm and the cost of redistributing data among processors. In this paper, we present a *basic-cycle calculation* technique to efficiently perform `BLOCK-CYCLIC(s)` to `BLOCK-CYCLIC(t)` redistribution. The main idea of the basic-cycle calculation technique is, first, to develop closed forms for computing source/destination processors of some specific array elements in a basic-cycle, which is defined as $lcm(s, t)/gcd(s, t)$. These closed forms are then used to efficiently determine the communication sets of a basic-cycle. From the source/destination processor/data sets of a basic-cycle, we can efficiently perform a `BLOCK-CYCLIC(s)` to `BLOCK-CYCLIC(t)` redistribution. To evaluate the performance of the basic-cycle calculation technique, we have implemented this technique on an IBM SP2 parallel machine, along with the *PITFALLS* method and the multiphase method. The cost models for these three methods are also presented. The experimental results show that the basic-cycle calculation technique outperforms the *PITFALLS* method and the multiphase method for most test samples.

Index Terms—Data redistribution, the basic-cycle calculation technique, the PITFALLS method, the multiphase method, distributed memory multicomputers.

1 Introduction

The data parallel programming model has become a widely accepted paradigm for programming distributed memory multicomputers. To efficiently execute a data parallel program on a distributed memory multicomputer, an appropriate data decomposition is critical. The data decomposition involves *data distribution* and *data alignment*. The data distribution deals with how data arrays should be distributed. The data alignment deals with how data arrays should be aligned with respect to one another. The purpose of data decomposition is to balance the computational load and minimize the communication overheads.

Many data parallel programming languages, such as High Performance Fortran (HPF) [8], Fortran D [5], Vienna Fortran [33], and High Performance C (HPC) [28], provide compiler directives for programmers to specify array distribution. The array distribution provided by those languages, in general, can be classified into two categories, *regular* and *irregular*. The regular array distribution, in general, has three types, `BLOCK`, `CYCLIC`, and `BLOCK-CYCLIC`(c). The `BLOCK-CYCLIC`(c) is the most general regular array distribution among them. Dongarra et al. [4] have shown that these distributions are essential for many dense matrix algorithms design in distributed memory machines. Examples of distributing a one-dimensional array with 18 elements to three processors using `BLOCK`, `CYCLIC`, and `BLOCK-CYCLIC`(c) distribution are shown in Fig. 1. The irregular array distribution uses user-defined array distribution functions to specify array distribution.

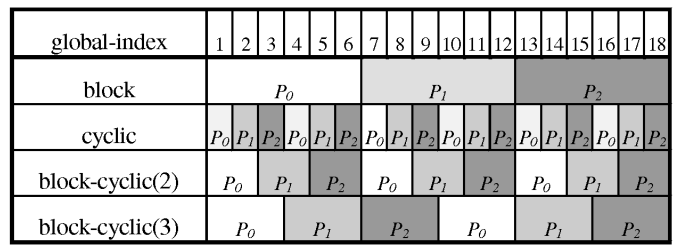


Fig. 1. Examples of regular data distributions.

In some algorithms, such as multidimensional fast Fourier transform [29], the Alternative Direction Implicit (ADI) method for solving two-dimensional diffusion equations, and linear algebra solvers [20], an array distribution that is well-suited for one phase may not be good for a subsequent phase in terms of performance. Array redistribution is required for those algorithms during run-time. Therefore, many data parallel programming languages support run-time primitives for changing a program's array decomposition [1], [2], [8], [28], [33]. Since array redistribution is performed at run-time, there is a performance trade-off between the efficiency of the new data decomposition for a subsequent phase of an algorithm and the cost of redistributing array among processors. Thus, efficient methods for performing array redistribution are of great importance for the development of distributed memory compilers for those languages.

In this paper, we present a *basic-cycle calculation* (*BCC*) technique to efficiently perform `BLOCK-CYCLIC`(s) to `BLOCK-CYCLIC`(t) redistribution. In HPF, it supports array redistribution with arbitrary source and destination processor sets. The technique developed in this paper assumes that the source and the destination processor sets are the same. The

• *The authors are with the Department of Information Engineering, Feng Chia University, Taichung, Taiwan 407, Republic of China.*
E-mail: {ychung, chhsu, swbai}@pine.iecs.fcu.edu.tw.

Manuscript received 26 July 1996; revised 15 Oct. 1997.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 100252.

main idea of the basic-cycle calculation technique is first to develop closed forms for computing source/destination processors of some specific array elements in a basic-cycle, which is defined as $lcm(s, t)/gcd(s, t)$. These closed forms are then used to efficiently determine the communication sets of a basic-cycle. From the source/destination processor/data sets of a basic-cycle, we can efficiently perform a `BLOCK-CYCLIC`(s) to `BLOCK-CYCLIC`(t) redistribution. Although, in this paper, we present this technique for one-dimensional array redistribution, this technique can be extended to multidimensional array redistribution as well. The basic-cycle calculation technique has the following characteristics:

- It is a simple one phase method to perform the general `BLOCK-CYCLIC`(s) to `BLOCK-CYCLIC`(t) redistribution.
- The indexing overhead of the basic-cycle calculation technique is independent of the number of processors and the array size involved in a redistribution. Given a `BLOCK-CYCLIC`(s) to `BLOCK-CYCLIC`(t) redistribution on a one-dimensional array $A[1 : N]$ over M processors, a processor only needs to scan $(s + t)/gcd(s, t)$ array elements in a basic-cycle and the destination/source processors for all array elements in its local array can be determined.
- Since the basic-cycle calculation technique uses an asynchronous communication scheme, the computation and the communication overheads can be overlapped. This leads to a better performance for a redistribution.

To evaluate the performance of the basic-cycle calculation technique, we have implemented this technique on an IBM SP2 parallel machine along with the *PITFALLS* method [22] and the multiphase method [13]. The cost models for these three methods are also presented. The experimental results show that the basic-cycle calculation technique outperforms the *PITFALLS* method and the multiphase method for most test samples.

The paper is organized as follows. In Section 2, a brief survey of related work will be presented. In Section 3, we will introduce notations and terminology used in this paper. Section 4 presents the algorithm of the basic-cycle calculation technique in details. The cost models and experimental results for these three methods will be presented in Section 5. The conclusions and future work will be given in Section 6.

2 Related Work

Many methods for performing array redistribution have been presented in the literature. These techniques can be classified into multicomputer compiler techniques and runtime support techniques. We briefly describe the related research in these two approaches.

Gupta et al. [6] derived closed form expressions for determining the send/receive processor/data sets for the `BLOCK` to `CYCLIC` redistribution (or vice versa). They also provided a virtual processor approach [7] to address the problem of reference index-sets identification for array statements with `BLOCK-CYCLIC`(c) distribution and formulated active processor sets as closed forms. Their approaches did not discover the repetitive patterns in communications sets. Koelbel [16] derived techniques for compile-time address and communication generation for array statements with `BLOCK` and `CYCLIC` distributions.

A recent work [14] extended the virtual processor approach to address the problem of memory allocation and index-sets identification. By using their method, closed form expressions for index-sets of arrays that were mapped to processors using one-level mapping can be translated to closed form expressions for index-sets of arrays that were mapped to processors using two-level mapping and vice versa. In [17], a similar approach that addressed the problems of the index-sets and the communication sets identification for array statements with `BLOCK-CYCLIC`(c) distribution was presented. Lee and Chen [17] derived communication sets for statements of arrays that were distributed in arbitrary `BLOCK-CYCLIC`(c) fashion. They also presented closed form expressions of communication sets for restricted block sizes.

In [9], an approach for generating communication sets by computing the intersections of index-sets corresponding to the LHS and RHS of array statements was presented. The intersections were computed by a scanning approach that exploited the repetitive pattern of the intersection of two index-sets. Kennedy et al. [15] also presented algorithms to compute the local memory access sequence for array statements with `BLOCK-CYCLIC`(c) distribution. In [23], the `CYCLIC`(k) distribution was viewed as a union of k `CYCLIC` distribution. Since the communication sets for `CYCLIC` distribution is easy to determine, communication sets for `CYCLIC`(k) distribution can be generated in terms of unions and intersections of some `CYCLIC` distributions. This method utilizes the repetitive pattern in communication sets calculation. Thirumalai and Ramanujam [26] also discussed communication sets generation and optimization for HPF array statements.

Chatterjee et al. [3] enumerated the local memory access sequence of communication sets for array statements with `BLOCK-CYCLIC`(c) distribution based on a finite-state machine (FSM). In this approach, the local memory access sequence can be characterized by an FSM at most c states. Their approach can handle the two-level mapping with hole compression. In [27], Thirumalai and Ramanujam represented the local memory access sequence as an integer lattice. They also derived closed form expressions for the basis vectors of integer lattices. Therefore, the basis vector generation is needless at runtime.

In [21], [22], Ramaswamy et al. used a mathematical representation, *PITFALLS*, for regular data redistribution. The main idea of *PITFALLS* is to find all intersections between source and target distributions. Based on the intersections, the send/receive processor/data sets can be determined and general redistribution algorithms can be devised. This method utilizes the repetitive pattern in communication sets calculation. The disadvantage of this approach is that, when the number of processors is large, iterations of the out-most loop in the FALLS intersection algorithm increased as well. This leads to high indexing overheads and degrades the performance of a redistribution algorithm. However, since one of the main goal of the *PITFALLS*

method is the handling of arbitrary source and target processor sets, this method does provide a total solution for a general array redistribution, i.e., this method can handle all the redistribution mechanisms that provided by HPF.

Prilli and Tourancheau [20] proposed a runtime scan algorithm for `BLOCK-CYCLIC` array redistribution. Their approach is similar to that of [21], [22], but has simpler indexing calculation than that of [21], [22]. This method utilizes the repetitive pattern in communication sets calculation.

Thakur et al. [24], [25] presented algorithms for run-time array redistribution in HPF programs. For `BLOCK-CYCLIC`(kr) to `BLOCK-CYCLIC`(r) redistribution (or vice versa), in most cases, a processor scanned its local array elements once to determine the destination (source) processor for each block of array elements of size r in the local array. Based on their `BLOCK-CYCLIC`(kr) to `BLOCK-CYCLIC`(r) and `BLOCK-CYCLIC`(r) to `BLOCK-CYCLIC`(kr) redistribution algorithms, they used a two-phase method to perform the general `BLOCK-CYCLIC`(s) to `BLOCK-CYCLIC`(t) redistribution. Based on the work of [24], [25], Kaushik et al. [12], [13] proposed a general two-phase redistribution approach for `BLOCK-CYCLIC`(s) to `BLOCK-CYCLIC`(t) redistribution. The main idea of two-phase redistribution is to perform a redistribution as a sequence of redistribution such that the communication cost of data movement among processors in the sequence is less than that of direct redistribution. Based on the closed form representations, a cost model for estimating the communication and the indexing overheads for array distribution was developed. From the cost model, algorithms for determining the sequence of intermediate array distributions that minimizes the total redistribution time were presented.

Instead of redistributing the entire array at one time, a strip mining approach was presented in [31]. In this approach, portions of array elements were redistributed in sequence in order to overlap the communication and computation. In [32], a spiral mapping technique was proposed. The main idea of this approach was to map formal processors onto actual processors such that the global communication can be translated to the local communication in a certain processor group. Since the communication is local to a processor group, one can reduce communication conflicts when performing a redistribution. Kalns and Ni [10], [11] proposed a processor mapping technique to minimize the amount of data exchange for `BLOCK` to `BLOCK-CYCLIC`(c) redistribution and vice versa. Using the data to logical processors mapping, they showed that the technique can achieve the maximum ratio between data retained locally and the total amount of data exchanged. In [18], a generalized circulant matrix formalism was proposed to reduce the communication overheads for `BLOCK-CYCLIC`(r) to `BLOCK-CYCLIC`(kr) redistribution. Using the generalized circulant matrix formalism, the authors derived direct, indirect, and hybrid communication schedules for the cyclic redistribution with the block size changed by an integer factor k. This method utilizes the repetitive pattern in communication sets calculation. They also extended this technique to solve some multidimensional redistribution problems [19].

Walker and Otto [30] used the standardized message passing interface, MPI, to express the redistribution operations. They implemented the `BLOCK-CYCLIC` array redistribution algorithms in both synchronous and asynchronous schemes. Since the excessive synchronization overheads are incurred from the synchronous scheme, they also presented the random and optimal scheduling algorithms for `BLOCK-CYCLIC` array redistribution. The experimental results showed that the performance of the synchronous method with optimal scheduling algorithm was comparable to that of the asynchronous method. This method utilizes the repetitive pattern in communication sets calculation.

3 Preliminaries

In this section, we will present the notations and terminology used in this paper. To simplify the presentation, we use $s \rightarrow t$ to represent the `BLOCK-CYCLIC`(s) to `BLOCK-CYCLIC`(t) redistribution for the rest of the paper.

Definition 1. *Given a* `BLOCK-CYCLIC`(s) *to* `BLOCK-CYCLIC`(t) *redistribution,* `BLOCK-CYCLIC`(s), `BLOCK-CYCLIC`(t), s, *and* t *are called the* source distribution, *the* destination distribution, *the* source distribution factor, *and the* destination distribution factor *of the redistribution, respectively.*

Definition 2. *Given an $s \rightarrow t$ redistribution on a one-dimensional array $A[1:N]$ over M processors, the* source local array *of processor P_i, denoted by $SLA_i[1:N/M]$, is defined as the set of array elements that are distributed to processor P_i in the source distribution, where $0 \le i \le M-1$. The* destination local array *of processor P_j, denoted by $DLA_j[1:N/M]$, is defined as the set of array elements that are distributed to processor P_j in the destination distribution, where $0 \le j \le M-1$.*

Definition 3. *Given an $s \rightarrow t$ redistribution on a one-dimensional array $A[1:N]$ over M processors, the* source processor *of an array element in $A[1:N]$ or $DLA_j[1:N/M]$ is defined as the processor that owns the array element in the source distribution, where $0 \le j \le M-1$. The* destination processor *of an array element in $A[1:N]$ or $SLA_i[1:N/M]$ is defined as the processor that owns the array element in the destination distribution, where $0 \le i \le M-1$.*

Definition 4. *Given an $s \rightarrow t$ redistribution on a one-dimensional array $A[1:N]$ over M processors, a basic-cycle (BC) of the redistribution is defined as the quotient of the least common multiple of s and t to the greatest common divisor of s and t, i.e., $BC = lcm(s,t)/gcd(s,t)$. We define $SLA_i[1:BC]$ ($DLA_j[1:BC]$) as the first basic-cycle in the source (destination) local array of processor P_i (P_j), $SLA_i[BC+1:2\times BC]$ ($DLA_j[BC+1:2\times BC]$) as the second basic-cycle in the source (destination) local array of processor P_i (P_j), and so on.*

Definition 5. *Given an $s \rightarrow t$ redistribution, a basic-cycle BC of a source (destination) local array can be divided into BC/s (BC/t) blocks. We define $SLA_i[1:s]$ ($DLA_j[1:t]$) as the first* source (destination) section *of $SLA_i[1:BC]$ ($DLA_j[1:BC]$) of processor P_i (P_j), $SLA_i[s+1:2\times s]$ ($DLA_j[t+1:2\times t]$) as the* second source (destination) section *of $SLA_i[1:BC]$ ($DLA_j[1:BC]$) of processor P_i (P_j), and so on.*

	Source : BLOCK-CYCLIC(3)											
index	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
SLA_0	1	2	3	7	8	9	13	14	15	19	20	21
SLA_1	4	5	6	10	11	12	16	17	18	22	23	24

	Destination : BLOCK-CYCLIC(2)											
index	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
DLA_0	1	2	5	6	9	10	13	14	17	18	21	22
DLA_1	3	4	7	8	11	12	15	16	19	20	23	24

(a)

$A[k]$	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
SDPP	0	1	2	3	4	5	0	1	2	3	4	5
DDPP	0	1	2	3	0	1	2	3	0	1	2	3
$A[k]$	***13***	***14***	***15***	***16***	***17***	***18***	***19***	***20***	***21***	***22***	***23***	***24***
SDPP	0	1	2	3	4	5	0	1	2	3	4	5
DDPP	0	1	2	3	0	1	2	3	0	1	2	3

(b)

Fig. 2. (a) A **BLOCK-CYCLIC**(3) to **BLOCK-CYCLIC**(2) redistribution with $M = 2$ and $N = 24$. (b) The corresponding source/destination distribution pattern positions of global array elements.

DEFINITION 6. *Given an $s \rightarrow t$ redistribution on a one-dimensional array $A[1 : N]$ over M processors, the* source distribution pattern position (SDPP) *of an array element $A[k]$ is defined as*

$$SDPP(A[k]) = mod\left(\left\lceil \frac{k}{gcd(s,t)} \right\rceil - 1, \frac{M \times s}{gcd(s,t)}\right),$$

where $k = 1, ..., N$. The destination distribution pattern position (DDPP) *of an array element $A[k]$ is defined as*

$$DDPP(A[k]) = \text{mod}\left(\left\lceil \frac{k}{gcd(s,t)} \right\rceil - 1, \frac{M \times t}{gcd(s,t)}\right),$$

where $k = 1, ..., N$.

Given a **BLOCK-CYCLIC**(s) to **BLOCK-CYCLIC**(t) redistribution on a one-dimensional array $A[1 : N]$ over M processors, if the source and the destination distribution pattern positions of $A[k]$ are x and y, respectively, then the source and destination processors of $A[k]$ are P_i and P_j, respectively, where $k = 1, ..., N$, $i = \lfloor x/s \rfloor$, and $j = \lfloor y/t \rfloor$.

We now give examples to clarify the above definitions. Fig. 2a shows a **BLOCK-CYCLIC**(3) to **BLOCK-CYCLIC**(2) redistribution on a one-dimensional array with $N = 24$ elements, $A[1 : 24]$ over $M = 2$ processors. Fig. 2b gives the corresponding source/destination pattern positions of global array elements. In Fig. 2a, the local array indices are represented as italic numbers, while the global array indices are represented as normal numbers. From Fig. 2a, we know that the basic-cycle of the redistribution is six. For a source processor, there are two source sections (size = 3) in each basic-cycle. For a destination processor, there are three destination sections (size = 2) in each basic-cycle. The first basic-cycle of source processor P_1 is $SLA_1[1 : 6]$. The second basic-cycle of source processor P_1 is $SLA_1[7 : 12]$. From Fig. 2b, we know that the source and the destination distribution pattern positions of $SLA_0[6] = A[9]$ are equal to two and zero, respectively. The source and the destination distribution pattern positions of $DLA_1[6] = A[12]$ are equal to five and three, respectively.

4 The Basic-Cycle Calculation Technique for Array Redistribution

To perform a redistribution, we first need to determine the send processor/data sets of source processors and the receive processor/data sets of destination processors. Then, a physical data movement among processors can be carried according to those sets. A naive way to get those sets is to scan every array element once and to compute those sets. Since a redistribution is performed at run-time, if an array size is very large, the time to determine those sets by scanning every array element once may greatly offset the performance of a program by performing the redistribution. Instead of scanning all array elements once, the main idea of the basic-cycle calculation technique is that every processor determines the send/receive processor/data sets on the first basic-cycle that it owns. According to the send/receive processor/data sets of the first basic-cycle, one can perform a redistribution very efficiently. We now give examples to explain the basic-cycle calculation technique.

Given a one-dimensional array $A[1 : 48]$ and $M = 4$ processors, Fig. 3 shows a **BLOCK-CYCLIC**($s = 3$) to **BLOCK-CYCLIC**($t = 2$) redistribution on A over M processors. In Fig. 3, the basic-cycle of the redistribution is equal to six. There are two basic-cycles in each source/destination local array. For each source (destination) local array, array elements in the kth position of the first and the second basic-cycle have the same destination (source) processor, i.e., both of them will be sent to (received from) the same destination (source) processor during the redistribution, where $k = 1$ to 6. This observation shows that each basic-cycle of a local array has the same communication pattern.

Another example of a **BLOCK-CYCLIC**(6) to **BLOCK-CYCLIC**(4) redistribution on $A[1 : 96]$ over $M = 4$ processors is shown in Fig. 4a. The basic-cycle of the redistribution is equal to six as well. However, the observation that we obtained from Fig. 3 (each basic-cycle of a local array has the same communication pattern) cannot be applied to the case shown in Fig. 4a directly. For example, array elements $SLA_0[1]$ and $SLA_0[7]$ are in the first position of the first and the second basic-cycle of source processor P_0, respectively. The destination processors of these two array elements are P_0 and P_2, respectively. Although they are in the same array position in two different basic-cycles, they do not have the same destination processor. The reason is that the value of $gcd(6, 4)$ is not equal to one. By grouping every $gcd(6, 4)$

	Source : BLOCK-CYCLIC(3)											
index	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
P_0	1	2	3	13	14	15	25	26	27	37	38	39
P_1	4	5	6	16	17	18	28	29	30	40	41	42
P_2	7	8	9	19	20	21	31	32	33	43	44	45
P_3	10	11	12	22	23	24	34	35	36	46	47	48

	Destination : BLOCK-CYCLIC(2)											
index	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
P_0	1	2	9	10	17	18	25	26	33	34	41	42
P_1	3	4	11	12	19	20	27	28	35	36	43	44
P_2	5	6	13	14	21	22	29	30	37	38	45	46
P_3	7	8	15	16	23	24	31	32	39	40	47	48

Fig. 3. A `BLOCK-CYCLIC`(3) to `BLOCK-CYCLIC`(2) redistribution with $M = 4$ and $N = 48$.

global array indices of array A to a meta-index, array $A[1 : N]$ can be transformed to a meta-array $B[1 : N/gcd(6, 4)]$, where $B[k] = \{A[(k-1) \times gcd(6, 4) + 1], ..., A[k \times gcd(6, 4)]\}$ and $k = 1$ to $N/gcd(6, 4)$. Then, the observation that we obtained from Fig. 3 can be held if we use array B for the redistribution. For example, if we use the meta-array $B[1 : 48]$, which is shown in Fig. 4b, for the redistribution, array elements $SLA_0[1] = \{A[1], A[2]\}$ and $SLA_0[7] = \{A[49], A[50]\}$ are the first array element of the first and the second basic-cycle of source processor P_0, respectively. The destination processors of these two meta-array elements are P_0. It is consistent to the observation obtained from Fig. 3. An example of using meta-array for the array redistribution of Fig. 4a is shown in Fig. 4b.

In the following discussion, we assume that an $s \rightarrow t$ redistribution on $A[1 : N]$ over M processors is given. We also assume that $gcd(s, t)$ is equal to one. If $gcd(s, t)$ is not equal to one, we use $s/gcd(s, t)$ and $t/gcd(s, t)$ as the source and destination distribution factors of the redistribution, respectively.

4.1 Send Phase

Given a source processor P_i and its corresponding source local array SLA_i, one can scan every array element $SLA_i[k]$ in $SLA_i[1 : BC]$ once and compute its destination processor by the following equation:

$$rank(P_j) = \begin{cases} M-1 & \text{if } \alpha = 0 \\ \left\lfloor \frac{\alpha - 1}{t} \right\rfloor & \text{otherwise.} \end{cases} \quad (1)$$

The value of α is defined as follows:

$$\alpha = mod\left(s \times \left(\left\lfloor \frac{k-1}{s} \right\rfloor \times (M-1) + rank(P_i) \right) + k,\ M \times t \right), \quad (2)$$

where $k = 1$ to BC, $rank(P_i)$ is the rank of a source processor P_i, and $rank(P_i) = 0$ to $M - 1$. However, if the value of BC is large, it may take a lot of time to compute the destination processors of array elements in a basic-cycle using (1) and (2). Since array elements in a source section have consecutive global array indices, if we know the destination distribution pattern position of the first array element of a source section, we can easily determine the destination processors of array elements in the source section according to Definition 6. For example, in Fig. 2, there are two source sections in $SLA_1[1 : BC]$ of a source processor P_1. The destination distribution pattern position of $SLA_1[1] = A[4]$ is equal to three. Since array elements in a source section have consecutive global array indices, from Definition 6, we can derive that $DDPP(SLA_1[2] = A[5]) = 0$ and $DDPP(SLA_1[3] = A[6]) = 1$, respectively. The corresponding destination processors of $SLA_1[1]$, $SLA_1[2]$, and $SLA_1[3]$ are equal to $\left\lfloor \frac{DDPP(SLA_1[1])}{t} \right\rfloor = P_1$, $\left\lfloor \frac{DDPP(SLA_1[2])}{t} \right\rfloor = P_0$, and $\left\lfloor \frac{DDPP(SLA_1[3])}{t} \right\rfloor = P_0$, respectively, where $t = 2$. For a source processor P_i, there are t source sections in $SLA_i[1 : BC]$. From the above analysis, each source processor P_i only needs to scan the first array element of the t source sections in $SLA_i[1 : BC]$ and the destination processors of array elements in $SLA_i[1 : BC]$ can be determined.

Given an $s \rightarrow t$ redistribution over M processors, for each source processor P_i, the destination distribution pattern position of $SLA_i[1]$ can be determined by the following equation:

$$DDPP = mod\left(rank(P_i) \times s,\ M \times t\right), \quad (3)$$

where $rank(P_i)$ is the rank of a source processor P_i, and $rank(P_i) = 0$ to $M - 1$. The destination distribution pattern position of the first array element $SLA_i[u]$ in the vth source section of $SLA_i[1 : BC]$ can be determined by the following equation:

$$DDPP(SLA_i[u]) = mod(DDPP + \beta \times v,\ M \times t), \quad (4)$$

where $v = 1$ to BC/s, $u = (v-1) \times s + 1$, and $\beta = mod(M \times s, M \times t)$. By merging (3) and (4), we have the following equation:

$$DDPP(SLA_i[u]) = mod\left(\left((v-1) \times M + rank(P_i)\right) \times s,\ M \times t\right). \quad (5)$$

According to (5) and Definition 6, we can easily construct the send processor/data sets of the first basic-cycle of source processors. The send processor/data sets of the first basic-cycle of source processors in Fig. 2 are shown in Fig. 5.

4.2 Receive Phase

Given a destination processor P_j and its corresponding destination local array DLA_j, one can scan every array element $DLA_j[k]$ in $DLA_j[1 : BC]$ once and compute its source processor by the following equation:

	Source : BLOCK-CYCLIC(6)																					
index	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***	***19***	***20***	***21***	***22***	***23***	***24***
P_0	1	2	3	4	5	6	25	26	27	28	29	30	49	50	51	52	53	54	73	74	75	76	77	78
P_1	7	8	9	10	11	12	31	32	33	34	35	36	55	56	57	58	59	60	79	80	81	82	83	84
P_2	13	14	15	16	17	18	37	38	39	40	41	42	61	62	63	64	65	66	85	86	87	88	89	90
P_3	19	20	21	22	23	24	43	44	45	46	47	48	67	68	69	70	71	72	91	92	93	94	95	96

	Destination : BLOCK-CYCLIC(4)																					
index	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***	***19***	***20***	***21***	***22***	***23***	***24***
P_0	1	2	3	4	17	18	19	20	33	34	35	36	49	50	51	52	65	66	67	68	81	82	83	84
P_1	5	6	7	8	21	22	23	24	37	38	39	40	53	54	55	56	69	70	71	72	85	86	87	88
P_2	9	10	11	12	25	26	27	28	41	42	43	44	57	58	59	60	73	74	75	76	89	90	91	92
P_3	13	14	15	16	29	30	31	32	45	46	47	48	61	62	63	64	77	78	79	80	93	94	95	96

(a)

	Source : BLOCK-CYCLIC(6)																					
local index	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*
meta-index	***1***		***2***		***3***		***4***		***5***		***6***		***7***		***8***		***9***		***10***		***11***		***12***	
P_0	1, 2		3, 4		5, 6		25, 26		27, 28		29, 30		49, 50		51, 52		53, 54		73, 74		75, 76		77, 78	
P_1	7, 8		9, 10		11, 12		31, 32		33, 34		35, 36		55, 56		57, 58		59, 60		79, 80		81, 82		83, 84	
P_2	13, 14		15, 16		17, 18		37, 38		39, 40		41, 42		61, 62		63, 64		65, 66		85, 86		87, 88		89, 90	
P_3	19, 20		21, 22		23, 24		43, 44		45, 46		47, 48		67, 68		69, 70		71, 72		91, 92		93, 94		95, 96	

	Destination : BLOCK-CYCLIC(4)																					
local index	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*
meta index	***1***		***2***		***3***		***4***		***5***		***6***		***7***		***8***		***9***		***10***		***11***		***12***	
P_0	1, 2		3, 4		17, 18		19, 20		33, 34		35, 36		49, 50		51, 52		65, 66		67, 68		81, 82		83, 84	
P_1	5, 6		7, 8		21, 22		23, 24		37, 38		39, 40		53, 54		55, 56		69, 70		71, 72		85, 86		87, 88	
P_2	9, 10		11, 12		25, 26		27, 28		41, 42		43, 44		57, 58		59, 60		73, 74		75, 76		89, 90		91, 92	
P_3	13, 14		15, 16		29, 30		31, 32		45, 46		47, 48		61, 62		63, 64		77, 78		79, 80		93, 94		95, 96	

(b)

Fig. 4. (a) A **BLOCK-CYCLIC**(6) to **BLOCK-CYCLIC**(4) redistribution with $M = 4$ and $N = 96$. (b) An example of grouping source/destination local arrays to metasource/destination local arrays for the redistribution in (a).

$$rank(P_i) = \begin{cases} M-1 & \text{if } \gamma = 0 \\ \left\lfloor \frac{\gamma - 1}{s} \right\rfloor & \text{otherwise.} \end{cases} \quad (6)$$

The value of γ is defined as follows:

$$\gamma = mod\left(t \times \left(\left\lfloor \frac{k-1}{t} \right\rfloor \times (M-1) + rank(P_j) \right) + k, M \times s \right), \quad (7)$$

where $k = 1$ to BC, $rank(P_j)$ is the rank of a destination processor P_j, and $rank(P_j) = 0$ to $M - 1$. However, if the value of BC is large, it may take a lot of time to compute the source processors of array elements in a basic-cycle by using (6) and (7). Since array elements in a destination section have consecutive global array indices, if we know the source distribution pattern position of the first array element of a destination section, we can easily determine the source processors of array elements in the destination section according to Definition 6. For example, in Fig. 2, there are three destination sections in $DLA_1[1 : BC]$ of a destination processor P_1. The source distribution pattern position of $DLA_1[1] = A[3]$ is equal to two. Since array elements in a destination section have consecutive global array indices, from Definition 6, we can derive that $SDPP(DLA_1[2] = A[4]) = 3$. The corresponding source processors of $DLA_1[1]$ and $DLA_1[2]$ are equal to $\left\lfloor \frac{SDPP(DLA_1[1])}{s} \right\rfloor = P_0$ and $\left\lfloor \frac{SDPP(DLA_1[2])}{s} \right\rfloor = P_1$, respectively, where $s = 3$. From the

local index	SLA_0					
	1	*2*	*3*	*4*	*5*	*6*
Destination processor	P_0	P_0	P_1	P_1	P_1	P_0
local index	SLA_1					
	1	*2*	*3*	*4*	*5*	*6*
Destination processor	P_1	P_0	P_0	P_0	P_1	P_1

Fig. 5. The send processor/data sets of the first basic-cycle for a `BLOCK-CYCLIC`(3) to `BLOCK-CYCLIC`(2) redistribution over two processors.

Local index	DLA_0					
	1	*2*	*3*	*4*	*5*	*6*
Source processor	P_0	P_0	P_1	P_1	P_0	P_0
Local index	DLA_1					
	1	*2*	*3*	*4*	*5*	*6*
Source processor	P_0	P_1	P_0	P_0	P_1	P_1

Fig. 6. The receive processor/data sets of the first basic-cycle for a `BLOCK-CYCLIC`(3) to `BLOCK-CYCLIC`(2) redistribution over two processors.

above analysis, each destination processor P_j only needs to scan the first array element of the s destination sections in $DLA_i[1 : BC]$ and the source processors of array elements in $DLA_i[1 : BC]$ can be determined.

Given an $s \rightarrow t$ redistribution over M processors, for each destination processor P_j, the source distribution pattern position of $DLA_j[1]$ can be determined by the following equation:

$$SDPP = mod\left(rank\left(P_j\right) \times t, M \times s\right), \tag{8}$$

where $rank(P_j)$ is the rank of a destination processor P_j, and $rank(P_j) = 0$ to $M - 1$. The source distribution pattern position of the first array element $DLA_j[z]$ in the wth destination section of $DLA_j[1 : BC]$, can be determined by the following equation:

$$SDPP\left(DLA_j[z]\right) = mod(SDPP + \varepsilon \times w, M \times s), \tag{9}$$

where $w = 1$ to BC/t, $z = (w - 1) \times t + 1$, and $\varepsilon = mod(M \times t, M \times s)$. By merging (8) and (9), we have the following equation:

$$SDPP\left(DLA_j[z]\right) = mod\left(\left((w-1) \times M + rank\left(P_j\right)\right) \times t, M \times s\right). \tag{10}$$

According to (10) and Definition 6, we can easily construct the receive processor/data sets of the first basic-cycles of destination processors. The receive processor/data sets of the first basic-cycle of destination processors in Fig. 2 are shown in Fig. 6.

The algorithm of the basic-cycle calculation algorithm technique is given as follows.

Algorithm basic_cycle_calculation (s, t, M)
1. $i = get\_my\_rank$ ( ); $x = lcm(s, t)$; $y = gcd(s, t)$;
2. $BC = x/y$; $s = s/y$; $t = t/y$;

/* Send phase */

3. *max_local_index* = the length of the source local array of processor P_i;

/* Construct the send processor/data set of the first basic cycle */

4. $DDPA\_length = M \times t$; $k = 0$; $stc_j = 0$, where $j = 0, ..., M - 1$;
5. **while** ($k < BC$)
6. { $DDPP = mod(((k/s) \times M + i) \times s, DDPA\_length)$;
7. **for** ($l = 1$; $l <= s$; l++)
8. {$j = \lfloor DDPP/t \rfloor$;
9. stc_j ++;
10. $s\_template[j + 1, stc_j] = k \times y$; k++; DDPP++;
11. **if** ($DDPP = DDPA\_length$) $DDPP = 0$; }}

/* Packing and Send data sets */

12. **for** ($j = i$, z=0 ; $z < M$; j++, z++)
13. {$j = mod(j, M)$;
14. **if** ($stc_j <> 0$)
15. { $length = 1$;
16. **for** ($sbase = 1$; $sbase <= max\_local\_index$; $sbase$ + = x)
17. { **for** ($k = 1$; $k <= stc_j$; k++)
18. { $index = sbase + s\_templete[j + 1, k]$;
19. **for** ($l = 0$; $l < y$; l++)
20. { $out\_buffer[length] = SLA_i[index]$;
21. $length$++; $index$++; } } }
22. Send out_buffer to processor P_j;
23. }
24. }

/* Receive phase */

/* Construct the receive processor/data set of the first basic cycle */

25. $SDPA\_length = M \times s$; $k = 0$; $rtc_j = 0$, where $j = 0, ..., M - 1$;
26. **while** ($k < BC$)
27. { $SDPP = mod(((k/t) \times M + i) \times t, SDPA\_length)$;
28. **for** ($l = 1$; $l <= t$; l++)
29. { $j = \lfloor SDPA/s \rfloor$;
30. rtc_j++;
31. $r\_template[j + 1, rtc_j] = k \times y$; k++; SDPP++;
32. **if** ($SDPP = SDPA\_length$) $SDPP = 0$; }}
33. **for** ($ri = j = 0$; $j < M$; j++)
34. **if** ($rtc_j <> 0$) ri++;
35. **while** ($ri > 0$)
36. { Receive data sets *in_buffer* from any source processor P_j.
37. *max_local_index* = the length of the destination local array of processor P_i;

/* Unpacking data sets */

38. $length = 1$;
39. **for** ($rbase = 1$; $rbase <= max\_local\_index$; $rbase$ += x)
40. { **for** ($k = 1$; $k <= rtc_j$; k++)

```
41.          {   index = rbase + r_templete[j + 1, k];
42.              for (l = 0; l < y; l++)
43.              {   DLA_i[index] = in_buffer[length];
44.                  length++; index++; } } }
45.      ri--;
46. }
end_of_basic_cycle_calculation
```

5 Performance Evaluation and Experimental Results

To evaluate the performance of the basic-cycle calculation technique, we compare the proposed approach with the multiphase method [12], [13], [24], [25] and the *PITFALLS* method [21], [22]. Both theoretical and experimental performance evaluation are conducted. We first develop cost models for these three methods and analyze their performance in terms of the computation and the communication overheads. The cost models developed for the multiphase method and the *PITFALLS* method are based on algorithms proposed in [25] and [21], respectively. We then execute these three methods on an IBM SP2 parallel machine and use the cost models to analyze the experimental results.

5.1 Cost Models

Given an $s \rightarrow t$ redistribution on a one-dimensional array $A[1 : N]$ over M processors, the time for an algorithm to perform the redistribution, in general, can be modeled as follows:

$$T = T_{comp} + T_{comm}, \quad (11)$$

where T_{comp} is the time for an algorithm to compute the source/destination processors of local array elements, pack source local array elements that have the same destination processors to the same message, and unpack array elements in messages that received from source processors to their corresponding destination local array positions; and T_{comm} is the time for an algorithm to send and receive data among processors. We said that T_{comp} and T_{comm} are the computation and communication time of an algorithm to perform a redistribution, respectively.

For the basic-cycle calculation technique, according to (11), the time to perform an $s \rightarrow t$ redistribution on a one-dimensional array $A[1 : N]$ over M processors can be modeled as follows:

$$T(BCC) = T_{comp}(BCC) + \delta \times T_s + \left(N/M\right) \times T_d, \quad (12)$$

where $T_{comp}(BCC)$ is the computation time of the basic-cycle calculation technique to perform a redistribution, δ is the maximum number of processors that a source processor needs to send data to, T_s is the startup time of the interconnection network of a parallel machine, and T_d is the data transmission time of the interconnection network of a parallel machine. The value of δ can be determined by the following equation:

$$\delta = min\left(M, \left\lceil \frac{max(s,t)}{min(s,t)} \right\rceil \times \frac{BC \times gcd(s,t)}{max(s,t)}\right). \quad (13)$$

For the multiphase method, an $s \rightarrow t$ redistribution may be decomposed into several phases, i.e., `BLOCK-CYCLIC`$(s_0 = s)$ to `BLOCK-CYCLIC`(s_1), `BLOCK-CYCLIC`(s_1) to `BLOCK-CYCLIC`(s_2), ..., `BLOCK-CYCLIC`(s_{n-1}) to `BLOCK-CYCLIC`$(s_n = t)$, where $s_{i-1} = c_1 s_i$ or $s_i = c_2 s_{i-1}$, for some integers $c_1 > 0$, $c_2 > 0$, and $i = 1, ..., n$. Therefore, according to (11), for the multiphase method, the time to perform an $s \rightarrow t$ redistribution on a one-dimensional array $A[1 : N]$ over M processors through n phases can be modeled as follows:

$$T(MP) = \sum_{i=1}^{n} T_{comp}(MP)_i + \sum_{i=1}^{n} \left(k_i \times T_s + \left(N/M\right) \times T_d\right), \quad (14)$$

where $T_{comp}(MP)_i$ is the computation time of the multiphase method to perform a `BLOCK-CYCLIC`(s_{i-1}) to `BLOCK-CYCLIC`(s_i) redistribution and $k_i = min(M, max(s_{i-1}, s_i)/min(s_{i-1}, s_i))$, for $i = 1, ..., n$; and T_s and T_d are the same as those in (12).

For the *PITFALLS* method, according to (11), the time to perform an $s \rightarrow t$ redistribution on a one-dimensional array $A[1 : N]$ over M processors can be modeled as follows:

$$T(PITFALLS) = T_{comp}(PITFALLS) + \delta \times T_s + \left(N/M\right) \times T_d, \quad (15)$$

where $T_{comp}(PITFALLS)$ is the computation time of the *PITFALLS* method to perform a redistribution, and δ, T_s, and T_d are the same as those in (12).

To analyze the computation and the communication costs for these three methods, we have the following assumptions:

1) For the multiphase method presented in [12], in general, the two-stage multiphase ($2P$) method outperforms three or more stage multiphase methods. Therefore, for the multiphase method, we only consider the two-stage ($2P$) multiphase method in our analysis.
2) In our analysis, we use the synchronous communication scheme to analyze the communication costs for these three methods. In real situations, the basic-cycle calculation technique, the *PITFALLS* method, and the multiphase method use some sort of asynchronous communication schemes to overlap the computation and the communication overheads. Since the communication and the computation overheads cannot be overlapped in the synchronous communication scheme, the communication costs presented in this section provide upper bounds for the actual communication costs of these three methods.

5.1.1 Analysis of Computation Costs

The computation overheads consist of the indexing cost and the packing/unpacking cost. The indexing cost is the time to construct the send/receive processor/data sets for a redistribution. The packing/unpacking cost is the time to gather array elements that have the same destination processors into a message in the send phase and put array elements from the received messages into their corresponding destination local array positions in the receive phase.

For the two-stage multiphase method, according to the algorithms proposed in [25], the indexing cost of a processor to perform the ith redistribution phase can be modeled as follows:

$$T_{comp}(MP)_i = \begin{cases} O\left(\frac{N \times c_i}{M \times s_{i-1}} + c_i\right) & \text{if } s_{i-1} = c_i s_i \ \&\ \left(c_i \le M,\ \text{mod}(M, c_i) = 0\right) \\ O\left(\frac{N \times c_i}{M \times s_{i-1}} + \frac{N}{M \times s_i}\right) & s_{i-1} = c_i s_i \\ O\left(\frac{N \times c_i}{M \times s_i} + c_i\right) & s_i = c_i s_{i-1}, \end{cases} \tag{16}$$

where i = 1 to 2. The packing/unpacking cost of the two-stage multiphase method is $2 \times O(N/M)$.

For the *PITFALLS* method, the indexing cost of a processor to perform the efficient *FALLS* intersection algorithm [21] in the send phase is

$$O\left(M \times \frac{lcm(s,t)}{s} \times \left(min\left(\left\lceil \frac{\left(lcm(s,t)+1\right) \times s}{2 \times t} \right\rceil, \frac{lcm(s,t)}{t} - 1\right) - max\left(0, \left\lceil \frac{lcm(s,t) \times s + t}{2 \times t} \right\rceil\right)\right)\right). \tag{17}$$

The value of (17) is approximate to $O\left(M \times \frac{|s-t|}{2 \times gcd(s,t)}\right)$. Since the indexing costs in the send phase and the receive phase are the same, the total indexing cost is $O\left(M \times \frac{|s-t|}{gcd(s,t)}\right)$. The packing/unpacking cost of the *PITFALLS* method is $O(N/M)$.

For the basic-cycle calculation technique, according to the algorithm presented in Section 4, the indexing cost is $O\left(\frac{lcm(s,t)}{gcd(s,t)}\right)$. The packing/unpacking cost of the basic-cycle calculation technique is $O(N/M)$.

From the above analysis, we observe that the indexing cost of the multiphase method depends on array size (N) and the number of processors (M). The indexing cost of the *PITFALLS* method depends on the number of processors (M). However, the indexing cost of the basic-cycle calculation technique is independent of the array size and the number of processors. The packing/unpacking costs of the basic-cycle calculation technique and the *PITFALLS* method are similar and are less than that of the two-stage multiphase method.

5.1.2 Analysis of Communication Costs

According to (12) and (15), the communication costs for the two single-phase methods, the basic-cycle calculation technique, and the *PITFALLS* method are the same. To simplify the analysis, we use the basic-cycle calculation to represent the single-phase method in the following discussion.

Given an $s \rightarrow t$ redistribution on a one-dimensional array $A[1 : N]$ over M processors, the relationship of s and t can be classified into the following three cases:

Case 1 : s is not divisible by t (or vice versa), the value of $gcd(s, t)$ is equal to one.
Case 2 : s is not divisible by t (or vice versa), the value of $gcd(s, t)$ is not equal to one.
Case 3 : s is divisible by t (or vice versa), i.e., $s = kt$ or $t = ks$, for some integer k.

For Case 1, according to (12), the communication time for the basic-cycle calculation technique to perform this redistribution is $T_{comm}(BCC) = \delta \times T_s + (N/M) \times T_d$. For the two-stage multiphase method, according to [13], a `BLOCK-CYCLIC`(s) to `BLOCK-CYCLIC`($lcm(s, t)$) redistribution followed by a `BLOCK-CYCLIC`($lcm(s, t)$) to a `BLOCK-CYCLIC`(t) redistribution will produce the best performance for the two- stage multiphase method in this redistribution. The communication time for the two-stage multiphase method to perform this redistribution through a `BLOCK-CYCLIC`(s) to `BLOCK-CYCLIC`($lcm(s, t)$) redistribution followed by a `BLOCK-CYCLIC`($lcm(s, t)$) to a `BLOCK-CYCLIC`(t) redistribution is $T_{comm}(MP) = (min(M, t) + min(M, s)) \times T_s + 2 \times (N/M) \times T_d$. We have the following lemma.

LEMMA 1. *Given an $s \rightarrow t$ redistribution on a one-dimensional array $A[1 : N]$ over M processors, where s is not divisible by t, t is not divisible by s, and the value of $gcd(s, t)$ is equal to one, the communication time for the two-stage multiphase method to perform this redistribution is always greater than that of the basic-cycle calculation technique.*

PROOF. The communication time for the basic-cycle calculation technique and the two-stage multiphase method to perform this redistribution are $T_{comm}(BCC) = \delta \times T_s + (N/M) \times T_d$ and $T_{comm}(MP) = (min(M, t) + min(M, s)) \times T_s + 2 \times (N/M) \times T_d$, respectively. Since

$$\delta = \min\left(M, \left\lceil \frac{\max(s,t)}{\min(s,t)} \right\rceil \times \frac{BC}{\max(s,t)}\right) \le min(M,t) + min(M,s),$$

we have $T_{comm}(BCC) < T_{comm}(MP)$. □

For Case 2, according to (12), the communication time for the basic-cycle calculation technique to perform this redistribution is $T_{comm}(BCC) = \delta \times T_s + (N/M) \times T_d$. For the two-stage multiphase method, it chooses `BLOCK-CYCLIC`($lcm(s, t)$) as the intermediate distribution of the redistribution. The communication time for the two-stage multiphase method to perform this redistribution is $T_{comm}(MP) = (min(M, t/gcd(s, t)) + min(M, s/gcd(s, t))) \times T_s + 2 \times (N/M) \times T_d$. We have the following lemma:

LEMMA 2. *Given an $s \rightarrow t$ redistribution on a one-dimensional array $A[1 : N]$ over M processors, where s is not divisible by t, t is not divisible by s, and the value of $gcd(s, t)$ is not equal to one, the communication time for the two-stage multiphase method to perform this redistribution is always greater than that of the basic-cycle calculation technique.*

PROOF. The communication time for the basic-cycle calculation technique and the two-stage multiphase method to perform this redistribution are $T_{comm}(BCC) = \delta \times T_s + (N/M) \times T_d$ and $T_{comm}(MP) = (min(M, t/gcd(s, t)) + min(M, s/gcd(s, t))) \times T_s + 2 \times (N/M) \times T_d$, respectively. Since

$$\delta = min\left(M, \left\lceil \frac{max(s,t)}{min(s,t)} \right\rceil \times \frac{BC \times gcd(s,t)}{max(s,t)}\right) \le min\left(M, t/gcd(s,t)\right) + min\left(M, s/gcd(s,t)\right),$$

we have $T_{comm}(BCC) < T_{comm}(MP)$. □

TABLE 1
THE COMPUTATION AND COMMUNICATION COSTS REQUIRED BY DIFFERENT METHODS FOR AN $s \rightarrow t$ REDISTRIBUTION ON A ONE-DIMENSIONAL ARRAY $A[1 : N]$ OVER M PROCESSORS

$s \rightarrow t$ Redistribution		*MP*	*PITFALLS*	*BCC*
Computation	*Indexing costs*	$O(\sum_{i=1}^{n} T_{comp}(MP)_i)$	$O(M \times \frac{\lvert s-t \rvert}{gcd(s,\ t)})$	$O(\frac{lcm\ (s,\ t)}{gcd\ (s,t)})$
	Packing/unpacking costs	$2 \times O(N/M)$	$O(N/M)$	$O(N/M)$
Communication	*Startup costs*	$(\sum_{i=1}^{2} k_i) \times T_s$	$\delta \times T_s$	$\delta \times T_s$
	Transmission costs	$2 \times (N/M \times T_d)$	$N/M \times T_d$	$N/M \times T_d$

For Case 3, according to (12), the communication time for the basic-cycle calculation technique to perform this redistribution is $T_{comm}(BCC) = \delta \times T_s + (N/M) \times T_d$. For the multiphase method, it can use one-stage or two-stage approaches to perform the redistribution. According to (14), the communication time for the one-stage method to perform this redistribution is $T_{comm}(MP) = min(M, max(s, t)/min(s, t)) \times T_s + (N/M) \times T_d$. We have the following lemma.

LEMMA 3. *Given an $s \rightarrow t$ redistribution on a one-dimensional array $A[1 : N]$ over M processors, where s is divisible by t or t is divisible by s, the communication time for the one-stage multiphase method to perform this redistribution is the same as that of the basic-cycle calculation technique.*

PROOF. The communication time for the basic-cycle calculation technique and the one-stage multiphase method to perform this redistribution is $T_{comm}(BCC) = \delta \times T_s + (N/M) \times T_d$ and $T_{comm}(MP) = min(M, max(s, t)/min(s, t)) \times T_s + (N/M) \times T_d$, respectively. Since

$$\delta = min\left(M, \left\lceil \frac{max(s,t)}{min(s,t)} \right\rceil \times \frac{BC \times gcd(s,t)}{max(s,t)}\right) = min\big(M, max\,(s,t)/min(s,t)\big),$$

we have $T_{comm}(BCC) = T_{comm}(MP)$. □

The communication time for the two-stage multiphase method to perform this redistribution is $T_{comm}(MP) = k \times T_s + 2 \times (N/M) \times T_d$, where $k = k_1 + k_2$. The communication time of the basic-cycle calculation technique is less than that of the two-stage multiphase method if and only if (18) is true.

$$T_{comm}(BCC) < T_{comm}(MP) \Leftrightarrow \frac{N}{M} > \frac{(\delta - k)T_s}{T_d}. \quad (18)$$

Table 1 summarizes the computation and the communication costs of these three methods to perform an $s \rightarrow t$ redistribution on a one-dimensional array $A[1 : N]$ over M processors.

5.2 Experimental Results

To verify the performance analysis presented in Section 5.1, we have implemented these three methods on an IBM SP2 parallel machine. All algorithms were written in the single program multiple data (SPMD) programming paradigm with C+MPI codes. For each case, we selected two different redistribution as test samples. They are given as follows.

Case 1 :

case 1-1 `BLOCK-CYCLIC`(5) to `BLOCK-CYCLIC`(8)
case 1-2 `BLOCK-CYCLIC`(100) to `BLOCK-CYCLIC`(3)

Case 2 :

case 2-1 `BLOCK-CYCLIC`(40) to `BLOCK-CYCLIC`(300)
case 2-2 `BLOCK-CYCLIC`(300) to `BLOCK-CYCLIC`(200)

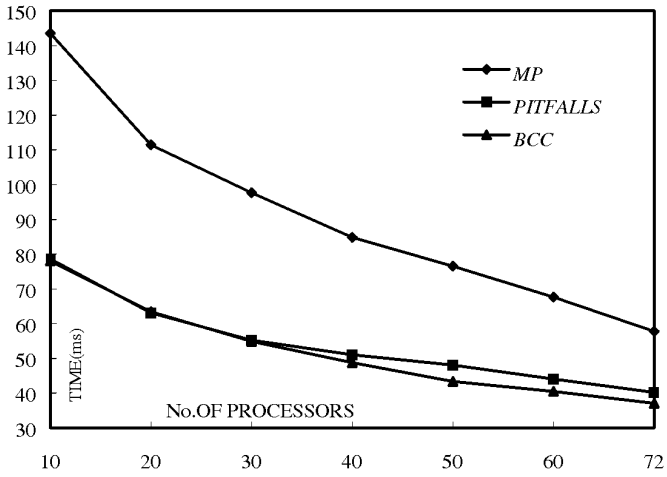
Case 3 :

case 3-1 `BLOCK-CYCLIC`(60) to `BLOCK-CYCLIC`(3)
case 3-2 `BLOCK-CYCLIC`(10) to `BLOCK-CYCLIC`(500)

To get the experimental results, each test sample with a particular array size N was executed 10 times by each algorithm on M processors, where $N \in$ {360K, 720K, 1.08M, 1.44M, 1.8M} and $M \in$ {10, 20, 30, 40, 50, 60, 72}. The mean time of these 10 tests that were executed by an algorithm was used as the time to perform a redistribution with a particular array size N on M processors of that algorithm. The single-precision array was used for the test. From the experimental results, we plotted the performance of these three methods for a redistribution on a fixed array size over different numbers of processors to a figure. We found that the figures for array size N = 360K, N = 720K, N = 1.08M, N = 1.44M, and N = 1.8M on different number of processors demonstrate similar characteristics. Therefore, in the following subsections, we only show the performance of these three methods for a redistribution on array size N = 1.8M over different numbers of processors M and the performance of these three methods for a redistribution on various array size N over M = 72 processors.

5.2.1 Experimental Results for Case 1

Case 1-1 `BLOCK-CYCLIC`(5) to `BLOCK-CYCLIC`(8). The performance of these three algorithms to execute a `BLOCK-CYCLIC`(5) to `BLOCK-CYCLIC`(8) redistribution with array size N = 1.8M on different numbers of processors is shown in Fig. 7a. To perform this redistribution, the two-stage multiphase method chooses `BLOCK-CYCLIC`(40) as the intermediate distribution. In Fig. 7a, for the same test sample, the



(a)

	MP			PITFALLS			BCC		
	Indexing	Packing/ unpacking	Comm.	Indexing	Packing/ Unpacking	Comm.	Indexing	Packing/ unpacking	Comm.
10	10.24	62.222	71.087	0.673	32.027	45.876	0.522	32.545	44.892
20	9.901	55.467	46.090	0.817	29.053	33.239	0.526	29.98	32.936
30	9.265	50.759	37.638	0.987	25.170	29.030	0.525	25.686	28.638
40	8.319	43.033	33.416	1.361	22.722	26.965	0.523	22.895	25.306
50	7.508	37.676	31.342	1.687	20.417	25.922	0.525	19.728	23.107
60	6.838	31.569	29.235	1.993	17.170	24.854	0.524	17.08	22.878
72	6.191	23.744	27.818	2.263	13.813	24.116	0.521	13.752	22.806

Time(ms)

(b)

Fig. 7. (a) Performance of different algorithms to perform a `BLOCK-CYCLIC`(5) to `BLOCK-CYCLIC`(8) redistribution on different number of processors with fixed array size N = 1.8M. (b) The indexing time, the packing/unpacking time, and the communication time for (a).

execution time of these three algorithms has the order $T(BCC) \le T(PITFALLS) < T(MP)$. When the number of processors is less than a threshold, the execution time of the basic-cycle calculation technique is similar to that of the *PITFALLS* method. However, when the number of processors is over the threshold, the execute time of the basic-cycle calculation technique is less than that of the *PITFALLS* method.

Fig. 7b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 7a. From Fig. 7b, we can see that the indexing time of the basic-cycle calculation technique is independent of the number of processors. The indexing time of the *PITFALLS* method depends on the number of processors. When the number of processors increases, the indexing time of the *PITFALLS* method increases as well. The indexing time of the two-stage multiphase method decreases when the number of processors increases. These phenomena match the performance analysis shown in Table 1. For the same test sample, the indexing time of these three algorithms has the order $T_{indexing}(BCC) < T_{indexing}(PITFALLS) < T_{indexing}(MP)$.

For the same test sample, the two-stage multiphase method has higher packing/unpacking time than that of the basic-cycle calculation technique and the *PITFALLS* method. The packing/unpacking time of the basic-cycle calculation technique is similar to that of the *PITFALLS* method. These phenomena also match the performance analysis shown in Table 1.

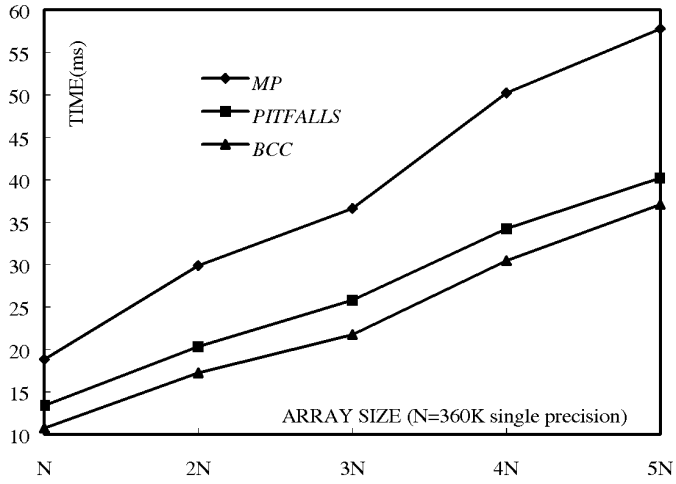
When performing a redistribution, both the basic-cycle calculation technique and the *PITFALLS* method use asynchronous communication schemes. However, the basic-cycle calculation technique unpacks any received messages in the receive phase while the *PITFALLS* method unpacks messages in a specific order. Therefore, in general, we can expect that the communication time of the basic-cycle calculation technique is less than or equal to that of the *PITFALLS* method. To perform the redistribution shown in this case, the two-stage multiphase method needs to execute two redistribution, `BLOCK-CYCLIC`(5) to `BLOCK-CYCLIC`(40) and `BLOCK-CYCLIC`(40) to `BLOCK-CYCLIC`(8). In `BLOCK-CYCLIC`(5) to `BLOCK-CYCLIC`(40) redistribution ($r \rightarrow kr$), the two-stage multiphase method uses the same asynchronous communication scheme that used in the basic-cycle calculation technique. In `BLOCK-CYCLIC`(40) to `BLOCK-CYCLIC`(8) redistribution ($kr \rightarrow r$), the two-stage multiphase method uses a synchronous communication scheme. Therefore, the two-stage multiphase has higher message startup costs and transmission costs than those of the basic-cycle calculation technique and the *PITFALLS* method in this case. From Fig. 7b, for the same test sample, the communication time of these three algorithms has the order $T_{comm}(BCC) \le T_{comm}(PITFALLS) < T_{comm}(MP)$.

Fig. 8a shows the performance of these three algorithms to execute a `BLOCK-CYCLIC`(5) to `BLOCK-CYCLIC`(8) redistribution with various array size on 72 processors. To perform this redistribution, the two-stage multiphase method chooses `BLOCK-CYCLIC`(40) as the intermediate distribution. For the basic-cycle calculation technique and the *PITFALLS* method, each source processor needs to send total $N/72$ array elements to 10 destination processors according to (13). The basic-cycle calculation technique and the *PITFALLS* method take $T(BCC) = T_{comp}(BCC) + 10T_s + 4 \times (N/72) \times T_d$ and $T(PITFALLS) = T_{comp}(PITFALLS) + 10T_s + 4 \times (N/72) \times T_d$ time to perform this redistribution, respectively. For the two-stage multiphase method, each source processor needs to send total $N/36$ array elements to 8 + 5 = 13 destination processors according to (14). The time for the two-stage multiphase method to perform this redistribution is

$$T(MP) = \sum_{i=1}^{2} T_{comp}(MP)_i + 13T_s + 4 \times (N/36) \times T_d .$$

Fig. 8b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 8a. From Fig. 8b, we can see that the indexing costs of the basic-cycle calculation technique and the *PITFALLS* method are independent of the array size. The indexing time of the two-stage multiphase method increases when the array size increases. These phenomena match the performance analysis shown in Table 1. For the same test sample, the indexing time of these three algorithms has the order $T_{indexing}(BCC) < T_{indexing}(PITFALLS) < T_{indexing}(MP)$.

The packing/unpacking costs of the basic-cycle calculation technique and the *PITFALLS* method are similar and are less than that of the two-stage multiphase method. These results match the performance analysis shown in Table 1.



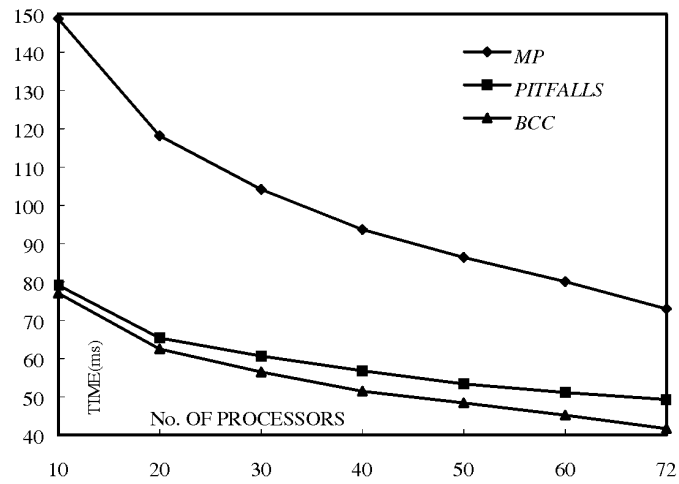
(a)

	MP			PITFALLS			BCC		
	Indexing	Packing/ unpacking	Comm.	Indexing	Packing/ Unpacking	Comm.	Indexing	Packing/ unpacking	Comm.
N	3.036	6.556	9.241	2.265	3.826	7.300	0.523	3.832	6.354
2*N*	4.069	11.120	14.699	2.263	6.067	10.002	0.521	5.505	9.327
3*N*	5.056	15.539	20.006	2.258	8.741	14.787	0.527	8.236	12.986
4*N*	5.637	19.141	25.446	2.265	11.470	19.488	0.523	11.153	17.704
5*N*	6.191	23.744	27.818	2.263	13.813	24.116	0.521	13.752	22.806

Time(ms)

(b)

Fig. 8. (a) Performance of different algorithms to execute a **BLOCK-CYCLIC**(5) to **BLOCK-CYCLIC**(8) redistribution with various array size on 72-node SP2. (b) The indexing time, the packing/unpacking time, and the communication time for (a).



(a)

	MP			PITFALLS			BCC		
	Indexing	Packing/ unpacking	Comm.	Indexing	Packing/ Unpacking	Comm.	Indexing	Packing/ unpacking	Comm.
10	18.856	58.837	71.061	2.435	30.859	45.850	1.456	30.856	44.760
20	17.847	53.809	46.534	2.937	28.471	33.938	1.455	28.976	32.083
30	16.306	48.972	38.852	3.441	26.727	30.484	1.454	26.667	28.396
40	14.987	43.464	35.211	4.115	23.691	28.975	1.453	23.302	26.701
50	14.329	38.175	33.874	4.677	20.022	28.658	1.452	20.45	26.462
60	13.812	33.886	32.413	5.156	17.771	28.225	1.454	17.464	26.295
72	12.136	28.942	31.895	5.774	15.222	28.320	1.457	14.904	25.327

Time(ms)

(b)

Fig. 9. (a) Performance of different algorithms to execute a **BLOCK-CYCLIC**(100) to **BLOCK-CYCLIC**(3) redistribution on different number of processors with fixed array size N = 1.8M. (b) The indexing time, the packing/unpacking time, and the communication time for (a).

For the communication overheads, since the two-stage multiphase method needs more message startup costs and transmission costs than those of the basic-cycle calculation technique and the *PITFALLS* method, the communication cost of the two-stage multiphase method is greater than those of the basic-cycle calculation technique and the *PITFALLS* method. For the basic-cycle calculation technique and the *PITFALLS* method, the basic-cycle calculation technique unpacks any received messages in the receive phase while the *PITFALLS* method unpacks messages in a specific order. Therefore, the communication time of the basic-cycle calculation technique is less than or equal to that of the *PITFALLS* method.

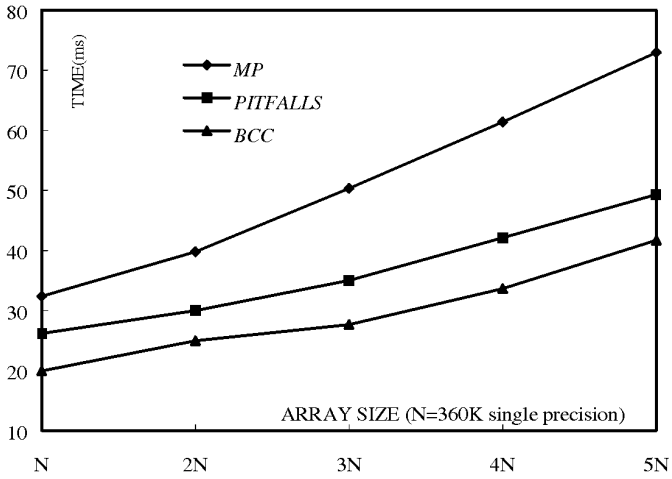
Case 1-2 BLOCK-CYCLIC(100) to BLOCK-CYCLIC(3). The performance of these three algorithms to execute a **BLOCK-CYCLIC**(100) to **BLOCK-CYCLIC**(3) redistribution with array size N = 1.8M on different numbers of processors is shown in Fig. 9a. The two-stage multiphase method chooses **BLOCK-CYCLIC**(300) as the intermediate distribution. In Fig. 9a, for the same test sample, the execution time of these three algorithms has the order $T(BCC) < T(PITFALLS) < T(MP)$.

Fig. 9b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 9a. From Fig. 9b, we have similar observations as those obtained from Fig. 7b.

Fig. 10a shows the performance of these three algorithms to execute a **BLOCK-CYCLIC**(100) to **BLOCK-CYCLIC**(3) redistribution with various array size on 72 processors. The two-stage multiphase method chooses **BLOCK-CYCLIC**(300) as the intermediate distribution. For the basic-cycle calculation technique and the *PITFALLS* method, each source processor needs to send total $N/72$ array elements to 72 destination processors according to (13). The basic-cycle calculation technique and the *PITFALLS* method take $T(BCC) = T_{comp}(BCC) + 72T_s + 4 \times (N/72) \times T_d$ and $T(PITFALLS) = T_{comp}(PITFALLS) + 72T_s + 4 \times (N/72) \times T_d$ time to perform this redistribution, respectively. For the two-stage multiphase method, each source processor needs to send total $N/36$ array elements to 3 + 72 = 75 destination processors according to (14). The time for the two-stage multiphase method to perform this redistribution is

$$T(MP) = \sum_{i=1}^{2} T_{comp}(MP)_i + 75T_s + 4 \times (N/36) \times T_d .$$

Fig. 10b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 10a. From Fig. 10b, we have similar observations as those obtained from Fig. 8b.



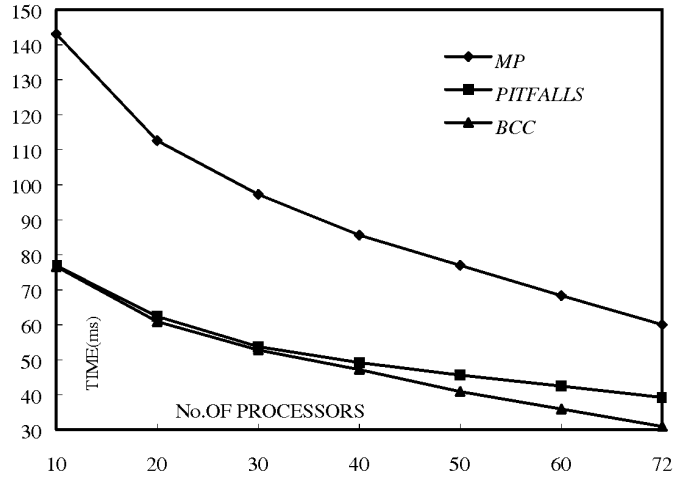
(a)

	MP			PITFALLS			BCC		
	Indexing	Packing/ unpacking	Comm.	Indexing	Packing/ Unpacking	Comm.	Indexing	Packing/ unpacking	Comm.
N	6.491	7.631	18.231	5.775	3.877	16.541	1.451	3.482	15.814
2*N*	7.702	10.377	21.699	5.772	5.944	18.251	1.458	5.684	17.823
3*N*	9.559	13.726	27.047	5.778	7.271	21.948	1.456	7.514	18.702
4*N*	10.570	20.331	30.468	5.774	11.747	24.601	1.454	11.812	20.394
5*N*	12.136	28.942	31.895	5.774	15.222	28.320	1.457	14.904	25.327

Time(ms)

(b)

Fig. 10. (a) Performance of different algorithms to execute a **BLOCK-CYCLIC**(100) to **BLOCK-CYCLIC**(3) redistribution with various array size on a 72-node SP2. (b) The indexing time, the packing/unpacking time, and the communication time for (a).



(a)

	MP			PITFALLS			BCC		
	Indexing	Packing/ unpacking	Comm.	Indexing	Packing/ unpacking	Comm.	Indexing	Packing/ unpacking	Comm.
10	14.019	57.951	71.005	0.743	30.252	45.883	0.511	30.251	45.855
20	13.262	52.940	46.398	0.911	27.79	33.654	0.511	27.363	33.041
30	12.545	46.685	37.975	1.242	23.082	29.428	0.512	23.238	28.994
40	11.773	40.133	33.728	1.707	20.153	27.335	0.515	20.027	26.684
50	10.917	34.345	31.648	2.038	17.237	26.320	0.513	17.319	23.069
60	10.237	28.475	29.562	2.442	14.795	25.285	0.514	14.723	20.695
72	9.594	22.263	28.188	2.831	11.892	24.530	0.511	11.438	19.036

Time(ms)

(b)

Fig. 11. (a) Performance of different algorithms to execute a **BLOCK-CYCLIC**(40) to **BLOCK-CYCLIC**(300) redistribution on different number of processors with fixed array size N = 1.8M. (b) The indexing time, the packing/unpacking time, and the communication time for (a).

5.2.2 Experimental Results for Case 2

Case 2-1 BLOCK-CYCLIC(40) to BLOCK-CYCLIC(300). The performance of these three algorithms to execute a **BLOCK-CYCLIC**(40) to **BLOCK-CYCLIC**(300) redistribution with array size N = 1.8M on different numbers of processors is shown in Fig. 11a. To perform this redistribution, the two-stage multiphase method chooses **BLOCK-CYCLIC**(600) as the intermediate distribution. In Fig. 11a, for the same test sample, the execution time of these three algorithms has the order $T(BCC) < T(PITFALLS) < T(MP)$.

Fig. 11b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 11a. From Fig. 11b, we have similar observations as those obtained from Fig. 7b.

Fig. 12a shows the performance of these three algorithms to execute a **BLOCK-CYCLIC**(40) to **BLOCK-CYCLIC**(300) redistribution with various array size on 72 processors. To perform this redistribution, the two-stage multiphase method chooses **BLOCK-CYCLIC**(600) as the intermediate distribution. For the basic-cycle calculation technique and the *PITFALLS* method, each source processor needs to send total $N/72$ array elements to 16 destination processors according to (13). The time for *BCC* and *PITFALLS* to perform this redistribution are $T(BCC) = T_{comp}(BCC) + 16T_s + 4 \times (N/72) \times T_d$ and $T(PITFALLS) = T_{comp}(PITFALLS) + 16T_s + 4 \times (N/72) \times T_d$, respectively. For the two-stage multiphase method, each source processor needs to send total $N/36$ array elements to 15 + 2 = 17 destination processors according to (14). The time for the two-stage multiphase method to perform this redistribution is

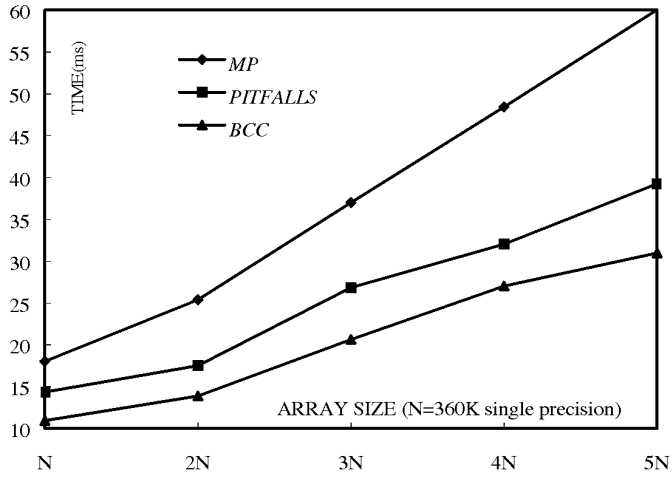
$$T(MP) = \sum_{i=1}^{2} T_{comp}(MP)_i + 17T_s + 4 \times (N/36) \times T_d .$$

Fig. 12b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 12a. From Fig. 12b, we have similar observations as those obtained from Fig. 8b.

Case 2-2 BLOCK-CYCLIC(300) to BLOCK-CYCLIC(200). The performance of these three algorithms to execute a **BLOCK-CYCLIC**(300) to **BLOCK-CYCLIC**(200) redistribution with array size N = 1.8M on different numbers of processors is shown in Fig. 13a. To perform this redistribution, the two-stage multiphase method chooses **BLOCK-CYCLIC**(600) as the intermediate distribution. In Fig. 13a, for the same test sample, the execution time of these three algorithms has the order $T(BCC) < T(PITFALLS) < T(MP)$.

Fig. 13b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 13a. From Fig. 13b, we have similar observations as those obtained from Fig. 7b.

Fig. 14a shows the performance of these three algorithms to execute a **BLOCK-CYCLIC**(300) to **BLOCK-CYCLIC**(200)



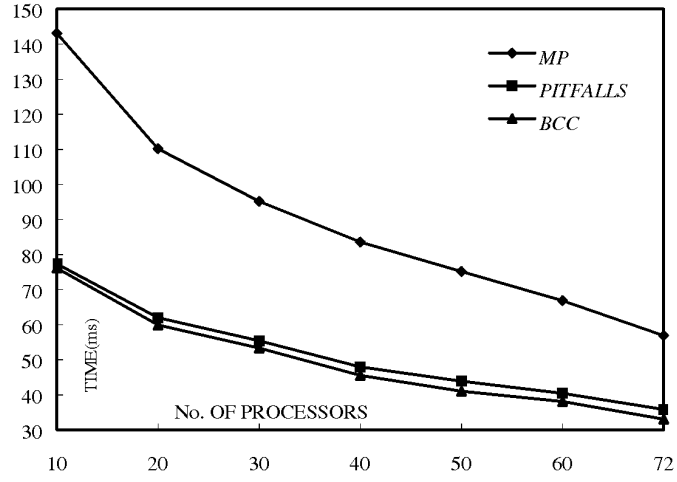
(a)

	MP			PITFALLS			BCC		
	Indexing	Packing/ unpacking	Comm.	Indexing	Packing/ Unpacking	Comm.	Indexing	Packing/ unpacking	Comm.
N	3.698	5.774	10.531	2.833	3.727	7.792	0.512	3.295	6.403
2*N*	5.003	8.366	14.999	2.834	4.282	11.402	0.514	4.588	9.783
3*N*	7.131	12.517	20.347	2.837	6.885	17.102	0.514	6.867	13.244
4*N*	8.094	18.576	25.768	2.832	9.386	19.817	0.517	9.188	17.896
5*N*	9.594	22.263	28.188	2.831	11.892	24.530	0.511	11.438	19.036

Time(ms)

(b)

Fig. 12. (a) Performance of different algorithms to execute a **BLOCK-CYCLIC**(40) to **BLOCK-CYCLIC**(300) redistribution with various array size on a 72-node SP2. (b) The indexing time, the packing/unpacking time, and the communication time for (a).



(a)

	MP			PITFALLS			BCC		
	Indexing	Packing/ unpacking	Comm.	Indexing	Packing/ unpacking	Comm.	Indexing	Packing/ unpacking	Comm.
10	12.001	60.303	70.794	0.719	31.139	45.458	0.306	31.683	44.100
20	11.464	53.162	45.521	0.796	28.316	32.869	0.305	28.491	31.089
30	10.921	47.119	37.158	0.863	25.889	28.664	0.301	25.725	27.275
40	10.175	40.425	32.964	0.964	20.449	26.503	0.306	20.884	24.287
50	9.691	34.643	30.836	1.014	17.3	25.589	0.308	17.308	23.418
60	9.221	28.883	28.706	1.085	14.87	24.480	0.309	14.802	22.935
72	8.758	20.746	27.371	1.155	10.923	23.710	0.303	10.917	21.845

Time(ms)

(b)

Fig. 13. (a) Performance of different algorithms to execute a **BLOCK-CYCLIC**(300) to **BLOCK-CYCLIC**(200) redistribution on different number of processors with fixed array size N = 1.8M. (b) The indexing time, the packing/unpacking time, and the communication time for (a).

redistribution with various array size on 72 processors. To perform this redistribution, the two-stage multiphase method chooses **BLOCK-CYCLIC**(600) as the intermediate distribution. For the basic-cycle calculation technique and the *PITFALLS* method, each source processor needs to send total $N/72$ array elements to four destination processors according to (13). The time for *BCC* and *PITFALLS* to perform this redistribution are $T(BCC) = T_{comp}(BCC) + 4T_s + 4 \times (N/72) \times T_d$ and $T(PITFALLS) = T_{comp}(PITFALLS) + 4T_s + 4 \times (N/72) \times T_d$, respectively. For the two-stage multiphase method, each source processor needs to send total $N/36$ array elements to 2 + 3 = 5 destination processors according to (14). The time for the two-stage multiphase method to perform this redistribution is

$$T(MP) = \sum_{i=1}^{2} T_{comp}(MP)_i + 5T_s + 4 \times (N/36) \times T_d .$$

Fig. 14b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 14a. From Fig. 14b, we have similar observations as those obtained from Fig. 8b.

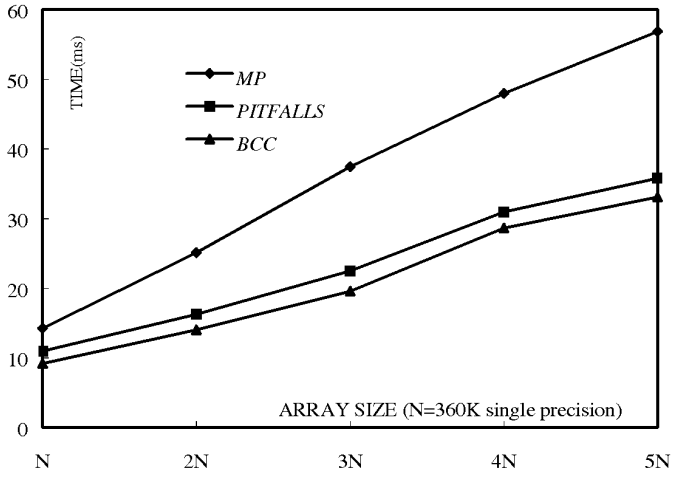
5.2.3 Experimental Results for Case 3

Since the redistribution in Case 3 can be performed by the one-stage (1*P*) method [25] directly, we also include the performance of the one-stage method for cases presented in this section.

Case 3-1 BLOCK-CYCLIC(60) to BLOCK-CYCLIC(3). The performance of these four algorithms to execute a **BLOCK-CYCLIC**(60) to **BLOCK-CYCLIC**(3) redistribution with array size N = 1.8M on different numbers of processors is shown in Fig. 15a. To perform this redistribution, the two-stage (2*P*) multiphase method selects **BLOCK-CYCLIC**(15) as the intermediate distribution. In Fig. 15a, for the same test sample, the execution time of these four algorithms has the order $T(BCC) < T(PITFALLS) < T(1P) < T(2P)$.

Fig. 15b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 15a. In Fig. 15b, for the same test sample, the indexing time of these four algorithms has the order $T_{indexing}(BCC) < T_{indexing}(PITFALLS) < T_{indexing}(1P) < T_{indexing}(2P)$. The packing/unpacking costs of the basic-cycle calculation technique, the *PITFALLS* method, and the one-stage method are similar and are less than that of the two-stage multiphase method.

For the basic-cycle calculation technique, the *PITFALLS* method, and the two-stage multiphase method, we have similar observations as those obtained from Fig. 7b for the communication overheads. In this case, the one-stage method uses a synchronous communication scheme while the basic-cycle calculation technique and the *PITFALLS* method use asynchronous communication schemes. Therefore, the



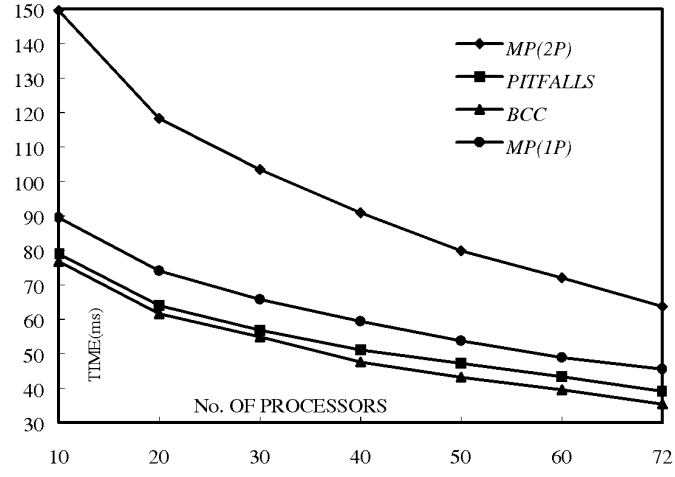
(a)

	MP			PITFALLS			BCC		
	Indexing	Packing/ unpacking	Comm.	Indexing	Packing/ unpacking	Comm.	Indexing	Packing/ unpacking	Comm.
N	2.186	4.337	7.722	1.149	2.825	6.984	0.303	2.733	6.124
2*N*	4.173	8.743	12.193	1.151	4.385	10.689	0.301	4.451	9.262
3*N*	6.131	12.716	18.575	1.156	6.926	14.390	0.302	6.963	12.284
4*N*	7.661	16.343	23.974	1.156	8.701	19.086	0.308	8.949	17.378
5*N*	8.758	20.746	27.371	1.155	10.923	23.710	0.303	10.917	21.845

Time(ms)

(b)

Fig. 14. (a) Performance of different algorithms to execute a `BLOCK-CYCLIC`(300) to `BLOCK-CYCLIC`(200) redistribution with various array size on a 72-node SP2. (b) The indexing time, the packing/unpacking time, and the communication time for (a).



(a)

	MP(2P)			MP(1P)			PITFALLS			BCC		
	indexing	Packing/ unpacking	Comm.	indexing	Packing/ unpacking	Comm.	indexing	Packing/ unpacking	Comm.	indexing	Packing/ unpacking	Comm.
10	13.271	65.181	71.063	9.737	33.869	45.881	1.287	33.82	43.888	0.368	33.871	42.512
20	12.602	59.842	45.802	9.216	30.824	33.975	1.456	30.901	31.647	0.366	30.929	30.333
30	12.064	53.948	37.435	8.718	27.265	29.754	1.665	27.228	27.927	0.369	27.697	26.791
40	11.217	46.487	33.209	8.036	23.644	27.696	1.858	23.17	26.029	0.367	23.205	24.011
50	10.613	38.068	31.139	7.584	19.661	26.526	2.006	19.447	25.722	0.368	19.619	23.164
60	10.033	32.948	29.040	7.063	16.268	25.5400	2.105	16.508	24.683	0.365	16.338	22.826
72	9.484	26.640	27.615	6.566	14.113	24.839	2.382	13.652	24.669	0.367	13.563	19.952

Time(ms)

(b)

Fig. 15. (a) Performance of different algorithms to execute a `BLOCK-CYCLIC`(60) to `BLOCK-CYCLIC`(3) redistribution on different number of processors with fixed array size N = 1.8M. (b) The indexing time, the packing/unpacking time, and the communication time for (a).

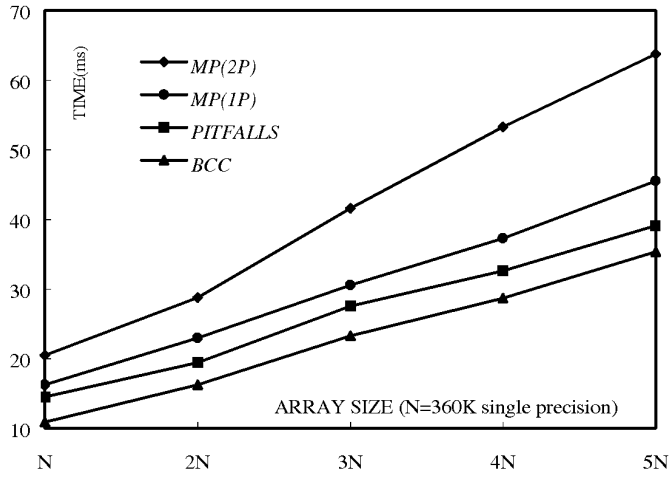
communication overheads of the basic-cycle calculation technique and the *PITFALLS* method are less than that of the one-stage method. In Fig. 15a, for the same test sample, we have $T_{comm}(BCC) \le T_{comm}(PITFALLS) < T_{comm}(1P) < T_{comm}(2P)$.

Fig. 16a shows the performance of these four algorithms to execute a `BLOCK-CYCLIC`(60) to `BLOCK-CYCLIC`(3) redistribution with various array size on 72 processors. To perform this redistribution, the two-stage (2*P*) multiphase method selects `BLOCK-CYCLIC`(15) as the intermediate distribution. For the basic-cycle calculation technique, the *PITFALLS* method, and the one-stage method, each source processor needs to send total $N/72$ array elements to 20 destination processors according to (13). The basic-cycle calculation technique, the *PITFALLS* method, and the one-stage method take $T(BCC) = T_{comp}(BCC) + 20T_s + 4 \times (N/72) \times T_d$, $T(PITFALLS) = T_{comp}(PITFALLS) + 20T_s + 4 \times (N/72) \times T_d$, and $T(1P) = T_{comp}(MP) + 20T_s + 4 \times (N/72) \times T_d$ time to perform this redistribution, respectively. For the two-stage multiphase method, each source processor needs to send total $N/36$ array elements to 4 + 5 = 9 destination processors according to (14). The time for the two-stage multiphase method to perform this redistribution is

$$T(MP) = \sum_{i=1}^{2} T_{comp}(MP)_i + 9T_s + 4 \times (N/36) \times T_d .$$

Fig. 16b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 16a. In Fig. 16b, for the same test sample, the indexing time of these four algorithms has the order $T_{indexing}(BCC) < T_{indexing}(PITFALLS) < T_{indexing}(1P) < T_{indexing}(2P)$. The packing/unpacking costs of the basic-cycle calculation technique, the *PITFALLS* method, and the one-stage method are similar and are less than that of the two-stage multiphase method.

For the communication overheads, we have three observations. First, although the two-stage multiphase method reduces the message startup costs (it needs nine while others need 20 in this case), the communication overheads are still greater than those of the basic-cycle calculation technique, the *PITFALLS* method, and the one-stage method. This is because the extra data transmission costs offset the reduced startup costs. This result can be verified by (18). Second, compared to the *PITFALLS* method and the basic-cycle calculation technique, the one-stage method also has more communication overheads. The reason is that the one-stage method uses a synchronous communication scheme while the basic-cycle calculation technique and the *PITFALLS* method use asynchronous communication schemes in this case. The communication and computation overheads can not be overlapped in the one-stage method. Therefore, the one-stage method has more communication overheads than those of the basic-cycle calculation technique and the *PITFALLS* method. Third, the communication time of the basic-cycle calculation technique is less than or



(a)

	MP(2P)			*MP*(1P)			*PITFALLS*			*BCC*		
	indexing	Packing/ unpacking	Comm.	indexing	Packing/ Unpacking	Comm.	indexing	Packing/ unpacking	Comm.	Indexing	Packing/ unpacking	Comm.
N	3.919	7.557	9.007	3.164	4.017	9.038	2.381	4.037	8.065	0.363	4.104	7.128
2*N*	4.711	9.580	14.471	3.781	5.441	13.742	2.383	5.474	12.573	0.364	5.698	10.168
3*N*	6.198	15.524	19.870	4.337	8.777	17.440	2.382	8.841	17.332	0.368	8.752	14.144
4*N*	8.13	20.913	24.243	5.593	11.519	20.146	2.381	11.131	20.873	0.360	11.193	18.784
5*N*	9.484	26.640	27.615	6.566	14.113	24.839	2.382	13.652	24.669	0.367	13.563	19.952

Time(ms)

(b)

Fig. 16. (a) Performance of different algorithms to execute a `BLOCK-CYCLIC`(60) to `BLOCK-CYCLIC`(3) redistribution with various array size on a 72-node SP2. (b) The indexing, packing/unpacking, and the communication time for (a).

equal to that of the *PITFALLS* method. The reason is the same as that described for Fig. 8b.

Case 3-2 `BLOCK-CYCLIC`(10) to `BLOCK-CYCLIC`(500). The performance of these four algorithms to execute a `BLOCK-CYCLIC`(10) to `BLOCK-CYCLIC`(500) redistribution with array size N = 1.8M on different numbers of processors is shown in Fig. 17a. To perform this redistribution, the two-stage multiphase method selects `BLOCK-CYCLIC`(50) as the intermediate distribution. In Fig. 17a, for the same test sample, the execution time of these four algorithms has the order $T(BCC) < T(PITFALLS) < T(1P) < T(2P)$.

Fig. 17b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 17a. From Fig. 17b, we have similar observations as those obtained from Fig. 15b.

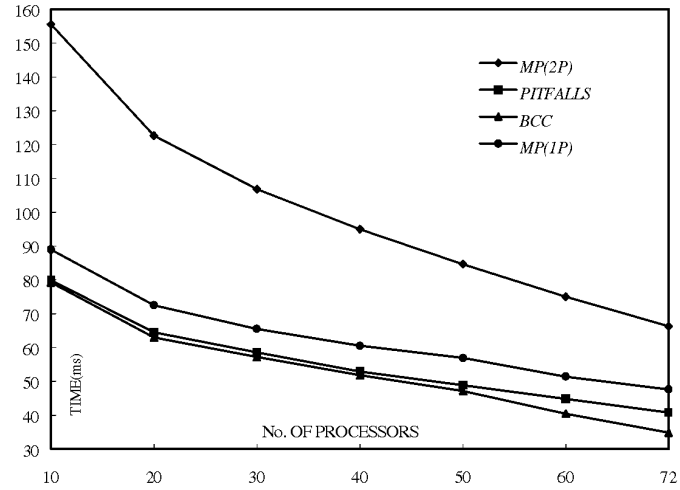
Fig. 18a shows the performance of these four algorithms to execute a `BLOCK-CYCLIC`(10) to `BLOCK-CYCLIC`(500) redistribution with various array size on 72 processors. To perform this redistribution, the two-stage multiphase method selects `BLOCK-CYCLIC`(50) as the intermediate distribution. For the basic-cycle calculation technique, the *PITFALLS* method, and the one-stage method, each source processor needs to send total $N/72$ array elements to 50 destination processors according to (13). The basic-cycle calculation technique, the *PITFALLS* method and the one-stage method take $T(BCC) = T_{comp}(BCC) + 50T_s + 4 \times (N/72) \times T_d$, $T(PITFALLS) = T_{comp}(PITFALLS) + 50T_s + 4 \times (N/72) \times T_d$, and $T(1P) = T_{comp}(MP) + 50T_s + 4 \times (N/72) \times T_d$ time to perform this redistribution, respectively. For the two-stage multiphase method, each source processor needs to send total $N/36$ array elements to 5 + 10 = 15 destination processors according to (14). The time for the two-stage multiphase method to perform this redistribution is

$$T(MP) = \sum_{i=1}^{2} T_{comp}(MP)_i + 15T_s + 4 \times \left(N/36\right) \times T_d .$$

Fig. 18b shows the indexing time, the packing/unpacking time, and the communication time for test samples shown in Fig. 18a. From Fig. 18b, we have similar observations as those obtained from Fig. 16b except that

1) the indexing time of the one-stage method is less than that of the *PITFALLS* method for the array size N = 360K and 720K; and
2) the communication cost of the two-stage multiphase method is less than those of the basic-cycle calculation technique, the *PITFALLS* method, and the one-stage method for the array size N = 360K and 720K.

The indexing time of the *PITFALLS* method depends on the number of processors while the indexing time of the one-stage method depends on the number of processors and the array size. When the array size is fixed and the number of processors is increasing, the number of array elements that will be processed by the one-stage method is decreasing. In this case, when the array size is N = 360K and the number of processors is M = 72, the indexing costs of the one-stage method and the *PITFALLS* method are $O(2(N/M/gcd(10, 500))$ and $O(M((500 - 10)/gcd(10, 500))$, respectively. Therefore, it is possible that the indexing time of the one-stage method



(a)

	MP(2P)			*MP*(1P)			*PITFALLS*			*BCC*		
	indexing	Packing/ unpacking	Comm.	indexing	Packing/ unpacking	Comm.	indexing	Packing/ unpacking	Comm.	indexing	Packing/ unpacking	Comm.
10	15.794	68.715	71.022	7.426	35.651	45.848	0.879	35.749	43.218	0.767	35.553	42.863
20	15.065	61.288	46.285	7.305	31.292	33.913	1.366	31.264	31.895	0.761	31.142	31.072
30	14.318	54.707	37.817	7.243	27.818	30.461	1.836	27.853	28.962	0.763	27.651	28.789
40	13.534	47.719	33.699	7.062	24.513	28.931	2.335	24.307	26.313	0.763	24.906	26.145
50	12.736	40.408	31.542	6.926	21.343	28.654	2.834	21.053	25.051	0.768	21.454	24.905
60	12.035	33.538	29.485	6.743	17.075	27.563	3.336	16.804	24.720	0.766	16.654	22.999
72	11.538	26.672	28.060	6.670	14.159	26.864	3.834	14.143	25.813	0.760	14.165	21.442

Time(ms)

(b)

Fig. 17. (a) Performance of different algorithms to execute a `BLOCK-CYCLIC`(10) to `BLOCK-CYCLIC`(500) redistribution on different number of processors with fixed array size N = 1.8M. (b) The indexing, packing/unpacking, and the communication time for (a).

is less than that of the *PITFALLS* method if the array size is small and the number of processors is large.

In this case, the communication cost of the two-stage multiphase method is $15T_s + 4 \times (N/36) \times T_d$ while others are $50T_s + 4 \times (N/72) \times T_d$. When the array size is small, it is possible that the communication costs of the two-stage multiphase method is less than those of the basic-cycle calculation technique, the *PITFALLS* method, and the one-stage method.

5.2.4 Discussions

Given an $s \rightarrow t$ redistribution on a one-dimensional array $A[1 : N]$ over M processors, from the above performance analysis and experimental results, we have the following remarks.

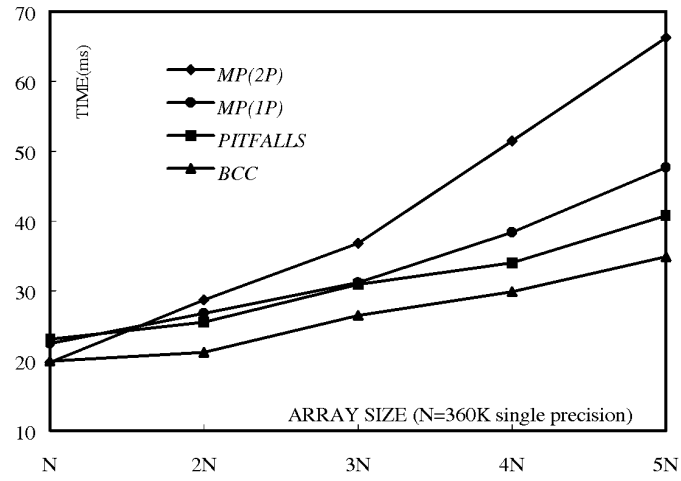
REMARK 1. *The indexing time of the basic-cycle calculation technique depends on the values of s and t and is independent of the values of M and N. The indexing time of the PITFALLS method depends on the values of s, t, and M; and is independent of the value of N. The indexing time of the two-stage multiphase method depends on the values of s, t, M, and N.*

REMARK 2. *The packing/unpacking costs of the basic-cycle calculation technique and the PITFALLS method are similar and are less than that of the two-stage multiphase method.*

REMARK 3. *Both the basic-cycle calculation technique and the PITFALLS method use asynchronous communication schemes. However, the basic-cycle calculation technique unpacks any received messages in the receive phase while the PITFALLS method unpacks messages in a specific order. Therefore, in general, we can expect that the communication time of the basic-cycle calculation technique is less than or equal to that of the PITFALLS method. For the two-stage multiphase method, it uses the same asynchronous communication scheme that is used in the basic-cycle calculation technique for an $r \rightarrow kr$ redistribution and a synchronous communication scheme for a $kr \rightarrow r$ redistribution. Therefore, in general, the two-stage multiphase has higher communication cost than those of the basic-cycle calculation technique and the PITFALLS method. However, if the array size is small and the number of processors is large, it is possible that the communication cost of the two-stage multiphase method is less than those of the basic-cycle calculation technique and the PITFALLS method for an $r \rightarrow kr$ redistribution.*

6 Conclusions and Future Work

In this paper, we have presented a basic-cycle calculation technique to efficiently perform `BLOCK-CYCLIC`(s) to `BLOCK-CYCLIC`(t) redistribution. The basic-cycle calculation technique is a simple method to perform `BLOCK-CYCLIC`(s) to `BLOCK-CYCLIC`(t) redistribution. The indexing overhead of the basic-cycle calculation technique is independent of



(a)

	MP(2P)			*MP*(1P)			*PITFALLS*			*BCC*		
	indexing	Packing/ unpacking	Comm.	indexing	Packing/ unpacking	Comm.	indexing	Packing/ unpacking	Comm.	indexing	Packing/ unpacking	Comm.
N	4.166	5.113	10.477	3.096	3.372	16.002	3.834	3.376	15.883	0.763	3.373	15.000
2*N*	5.133	8.737	14.840	3.48	4.592	18.733	3.831	4.531	17.156	0.760	4.763	15.670
3*N*	7.200	12.413	20.222	4.571	7.161	19.470	3.831	7.472	19.619	0.761	7.108	18.624
4*N*	8.767	18.048	25.652	5.766	9.505	23.115	3.833	9.566	21.622	0.761	9.574	19.561
5*N*	11.538	26.672	28.060	6.670	14.159	26.864	3.834	14.143	25.813	0.760	14.165	21.442

Time(ms)

(b)

Fig. 18. (a) Performance of different algorithms to execute a `BLOCK-CYCLIC`(10) to `BLOCK-CYCLIC`(500) redistribution with various array size on a 72-node SP2. (b) The indexing, packing/unpacking, and the communication time for (a).

the number of processors and the array size involved in a redistribution. It also uses an asynchronous communication scheme to overlap the computation overhead and the communication overhead. To evaluate the performance of the basic-cycle calculation technique, we compare it with the *PITFALLS* method and the two-stage multiphase method. Both theoretical and experimental analysis were conducted for these three methods. The experimental results demonstrate that the basic-cycle calculation technique outperforms the multiphase method and the *PITFALLS* method for most test samples.

Although, in this paper, we present this technique for one-dimensional array redistribution, this technique can be extended to multidimensional array redistribution as well. Given a multidimensional array redistribution, one can use this technique to determine the communication sets of array elements in each dimension. By taking the Cartesian product of communication sets in each dimension, we can obtain the final communication sets. Some details, such as the trade-off between indexing and packing/unpacking overheads, how to minimize the communication overheads, etc., still need to be addressed. We will discuss these issues in a future paper.

HPF supports array redistribution with arbitrary source and destination processor sets. The technique developed in this paper assumes that the source and the destination processor sets are the same. In the future, we will study efficient methods for array redistribution with arbitrary source and destination processor sets. Also, since multidimensional array redistribution is an important topic for data parallel compilers, in the future, we will study how to extend this technique for multidimensional array redistribution.

Acknowledgments

The work of this paper was partially supported by the National Science Council of the Republic of China under contract NSC-87-2213-E035-011.

References

[1] S. Benkner, "Handling BLOCK-CYCLIC Distribution Arrays in Vienna Fortran 90," *Proc. Int'l. Conf. Parallel Architectures and Compilation Techniques*, Limassol, Cyprus, June 1995.

[2] B. Chapman, P. Mehrotra, H. Moritsch, and H. Zima, "Dynamic Data Distribution in Vienna Fortran," *Proc. Supercomputing '93*, pp. 284-293, Nov. 1993.

[3] S. Chatterjee, J.R. Gilbert, F.J.E. Long, R. Schreiber, and S.-H. Teng, "Generating Local Address and Communication Sets for Data Parallel Programs," *J. Parallel and Distributed Computing*, vol. 26, pp. 72-84, 1995.

[4] J.J. Dongarra, R. Van De Geijn, and D.W. Walker, "A Look at Scalable Dense Linear Algebra Libraries," Technical Report ORNL/TM-12126, Oak Ridge Nat'l Laboratory, Apr. 1992.

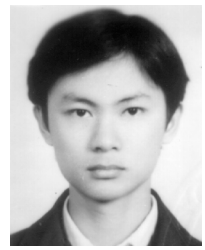
[5] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C.-W. Tseng, and M. Wu, "Fortran-D Language Specification," Technical Report TR-91-170, Dept. of Computer Science, Rice Univ., Dec. 1991.

[6] S.K.S. Gupta, S.D. Kaushik, C.-H. Huang, and P. Sadayappan, "On the Generation of Efficient Data Communication for Distributed-Memory Machines," *Proc. Int'l. Conf. Computing Symp.*, pp. 504-513, Taiwan, 1992.

[7] S.K.S. Gupta, S.D. Kaushik, C.-H. Huang, and P. Sadayappan, "On Compiling Array Expressions for Efficient Execution on Distributed-Memory Machines," *J. Parallel and Distributed Computing*, vol. 32, pp. 155-172, 1996.
[8] High Performance Fortran Forum, "High Performance Fortran Language Specification (version 1.1)," Rice Univ., Nov. 1994.
[9] S. Hiranandani, K. Kennedy, J. Mellor-Crammey, and A. Sethi, "Compilation Technique for BLOCK-CYCLIC Distribution," *Proc. Supercomputing '94*, pp. 392-403, July 1994.
[10] E.T. Kalns, and L.M. Ni, "Processor Mapping Technique Toward Efficient Data Redistribution," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, no. 12, Dec. 1995.
[11] E.T. Kalns and L.M. Ni, "DaReL: A Portable Data Redistribution Library for Distributed-Memory Machines," *Proc. 1994 Scalable Parallel Libraries Conf. II*, Oct. 1994.
[12] S.D. Kaushik, C.H. Huang, R.W. Johnson, and P. Sadayappan, "An Approach to Communication Efficient Data Redistribution," *Proc. Supercomputing '94* , pp. 364-373, July 1994.
[13] S.D. Kaushik, C.H. Huang, J. Ramanujam, and P. Sadayappan, "Multiphase Array Redistribution: Modeling and Evaluation," *Proc. Int'l. Parallel Processing Symp.*, pp. 441-445, 1995.
[14] S.D. Kaushik, C.H. Huang, and P. Sadayappan, "Efficient Index Set Generation for Compiling HPF Array Statements on Distributed-Memory Machines," *J. Parallel and Distributed Computing*, vol. 38, pp. 237-247, 1996.
[15] K. Kennedy, N. Nedeljkovic, and A. Sethi, "Efficient Address Generation for BLOCK-CYCLIC Distribution," *Proc. Supercomputing '95*, Barcelona, pp. 180-184, July 1995.
[16] C. Koelbel, "Compiler-Time Generation of Communication for Scientific Programs," *Proc. Supercomputing '91*, pp. 101-110, Nov. 1991.
[17] P-Z. Lee and W.Y. Chen, "Compiler Techniques for Determining Data Distribution and Generating Communication Sets on Distributed-Memory Multicomputers," *Proc. 29th Hawaii Int'l. Conf. System Sciences*, vol. 1, pp. 537-546, Jan. 1996.
[18] Y.W. Lim, P.B. Bhat, and V.K. Prasanna, "Efficient Algorithms for BLOCK-CYCLIC Redistribution of Arrays," *Proc. Eighth Symp. Parallel and Distributed Processing*, pp. 74-83, 1996.
[19] Y.W. Lim, N. Park, and V.K. Prasanna, "Efficient Algorithms for Multi-Dimensional Block-Cyclic Redistribution of Arrays," *Proc. Int'l. Conf. Parallel Processing*, pp. 234-241, 1997.
[20] L. Prylli and B. Touranchean, "Fast Runtime Block Cyclic Data Redistribution on Multiprocessors," *J. Parallel and Distributed Computing*, vol. 45, pp. 63-72, Aug. 1997.
[21] S. Ramaswamy and P. Banerjee, "Automatic Generation of Efficient Array Redistribution Routines for Distributed Memory Multicomputers," *Proc. Frontier '95: Fifth Symp. Frontiers of Massively Parallel Computation*. pp. 342-349, McLean, Va., Feb. 1995.
[22] S. Ramaswamy, B. Simons, and P. Banerjee, "Optimization for Efficient Array Redistribution on Distributed Memory Multicomputers," *J. Parallel and Distributed Computing* , vol. 38, pp. 217-228, 1996.
[23] J.M. Stichnoth, D. O'Hallaron, and T.R. Gross, "Generating Communication for Array Statements: Design, Implementation, and Evaluation," *J. Parallel and Distributed Computing* , vol. 21, pp. 150-159, 1994.
[24] R. Thakur, A. Choudhary, and G. Fox, "Runtime Array Redistribution in HPF Programs," *Proc. 1994 Scalable High Performance Computing Conf.* , pp. 309-316, May 1994.
[25] R. Thakur, A. Choudhary, and J. Ramanujam, "Efficient Algorithms for Array Redistribution," *IEEE Trans. Parallel and Distributed Systems*, vol. 7, no. 6, pp. 587-594, June 1996.
[26] A. Thirumalai and J. Ramanujam, "HPF Array Statements: Communication Generation and Optimization," *Proc. Third Workshop Languages, Compilers and Run-Time System for Scalable Computers*, Troy, N.Y., May 1995.
[27] A. Thirumalai and J. Ramanujam, "Efficient Computation of Address Sequences in Data Parallel Programs Using Closed Forms for Basis Vectors," *J. Parallel and Distributed Computing* , vol. 38, pp. 188-203, 1996.
[28] V. Van Dongen, C. Bonello, and C. Freehill, "High Performance C—Language Specification Version 0.8.9," Technical Report CRIM-EPPP-94/04-12, 1994.
[29] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*. SIAM, 1992.
[30] D.W. Walker and S.W. Otto, "Redistribution of BLOCK-CYCLIC Data Distributions Using MPI," *Concurrency: Practice and Experience*, vol. 8, no. 9, pp. 707-728, Nov. 1996.
[31] A. Wakatani and M. Wolfe, "A New Approach to Array Redistribution: Strip Mining Redistribution," *Proc. Parallel Architectures and Languages Europe*, July 1994.
[32] A.i Wakatani and M. Wolfe, "Optimization of Array Redistribution for Distributed Memory Multicomputers," *Parallel Computing*, vol. 21, no. 9, 1995.
[33] H. Zima, P. Brezany, B. Chapman, P. Mehrotra, and A. Schwald, "Vienna Fortran—A Language Specification Version 1.1," ICASE Interim Report 21, ICASE NASA Langley Research Center, Hampton, Va., Mar. 1992.

Yeh-Ching Chung received a BS degree in computer science from Chung Yuan Christian University in 1983, and MS and a PhD degrees in computer and information science from Syracuse University in 1988 and 1992, respectively. Since 1992, he has been an associate professor in the Department of Information Engineering at Feng Chia University. His research interests include parallel compilers, parallel programming tools, mapping, scheduling, and load balancing.

Ching-Hsien Hsu received a BS degree in computer science from Tung Hai University in 1995. He is currently a PhD student in the Department of Information Engineering at Feng Chia University. His research interests are in the areas of parallel and distributed computing, parallel algorithms, and high performance compilers for data parallel programming languages.



Sheng-Wen Bai received a BS degree in information engineering from Feng Chia University in 1996. He is currently a master's student in the Department of Information Engineering at Feng Chia University. His research interests are in the areas of parallel and distributed computing, performance analysis, and high performance compilers for data parallel programming languages.