A Generalized Processor Mapping Technique for Array Redistribution

Ching-Hsien Hsu, Yeh-Ching Chung, *Member*, *IEEE Computer Society*, Don-Lin Yang, *Member*, *IEEE Computer Society*, and Chyi-Ren Dow

Abstract—In many scientific applications, array redistribution is usually required to enhance data locality and reduce remote memory access in many parallel programs on distributed memory multicomputers. Since the redistribution is performed at runtime, 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 generalized processor mapping technique to minimize the amount of data exchange for BLOCK-CYCLIC(*kr*) to BLOCK-CYCLIC(*r*) array redistribution and vice versa. The main idea of the generalized processor mapping technique is first to develop mapping functions for computing a new rank of each destination processor. Based on the mapping functions, a new logical sequence of destination processors can be derived. The new logical processor sequence is then used to minimize the amount of data exchange in a redistribution. The generalized processor mapping technique can handle array redistribution with arbitrary source and destination processor sets and can be applied to multidimensional array redistribution. We present a theoretical model to analyze the performance improvement of the generalized processor mapping technique. To evaluate the performance of the proposed technique, we have implemented the generalized processor mapping technique on an IBM SP2 parallel machine. The experimental results show that the generalized processor mapping technique can provide performance improvement over a wide range of redistribution problems.

Index Terms—Array redistribution, generalized processor mapping, distributed memory multicomputers, runtime support.

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, 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) [9], Fortran D [6], 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 irregular array distribution uses user-defined array distribution functions to specify array distribution.

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 [21], 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 at runtime. Therefore, many data parallel programming languages support runtime primitives for changing a program's array decomposition [1], [2], [9], [28], [33]. Since array redistribution is performed at runtime, there is a performance trade-off between the efficiency of a new data decomposition for a subsequent phase of an algorithm and the cost of redistributing arrays 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 generalized processor mapping technique to minimize the amount of data exchange of BLOCK-CYCLIC(*kr*) to BLOCK-CYCLIC(*r*) redistribution and vice versa. The data transmission cost of a redistribution can be reduced. Compared with the technique proposed by Kalns et al. [12], the generalized processor mapping technique is effective not only on BLOCK to BLOCK-CYCLIC(*r*) (or vice versa) redistribution but also on BLOCK-CYCLIC(*kr*) to BLOCK-CYCLIC(*r*) and BLOCK-CYCLIC(*r*) to BLOCK-CYCLIC(*kr*) array redistribution. Another contribution of the generalized processor mapping technique is the ability to handle array redistribution with arbitrary source and destination processor sets. We also present a theoretical model to compute the amount of data that is retained locally and to analyze the performance improvement through a redistribution. The generalized processor mapping technique has the following characteristics:

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

Manuscript received 29 Jan. 1999; revised 6 Sept. 2000; accepted 22 Jan. 2001. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 109077.

- The generalized processor mapping technique can minimize the amount of data that needs to be communicated in BLOCK-CYCLIC(kr) to BLOCK-CYCLIC(r) and BLOCK-CYCLIC(r) to BLOCK-CYCLIC(kr) array redistribution. The data transmission cost of a redistribution can be reduced.
- The generalized processor mapping technique can handle array redistribution with arbitrary source and destination processor sets and also multidimensional arrays.
- The proposed mapping functions determine a unique logical processor sequence that achieves the maximum amount of data retained locally in a redistribution.
- If the source processor set and destination processor set of a redistribution are two disjoint sets, then the generalized processor mapping technique will be stultified.

We have implemented the generalized processor mapping technique on an IBM SP2 parallel machine. The experimental results show that the generalized processor mapping technique provides performance improvement for most redistribution samples.

The rest of this 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 generalized processor mapping technique for BLOCK-CYCLIC(kr) to BLOCK-CYCLIC(r) and BLOCK-CYCLIC(r) to BLOCK-CYCLIC(kr) redistribution. In Section 5, we will present the generalized processor mapping technique for multidimensional array redistribution. The performance analysis and experimental results 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 [27] and runtime support techniques. We briefly describe the related research in these two approaches.

Gupta et al. [7] derived closed form expressions to efficiently determine the send/receive processor/data sets. They also provided a virtual processor approach [8] for addressing the problem of reference index-set identification for array statements with BLOCK-CYCLIC(c) distribution and formulated active processor sets as closed forms. A recent work in [16] extended the virtual processor approach to address the problem of memory allocation and index-set 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. A similar approach that addressed the problems of the index set and the communication sets identification for array statements with BLOCK-CYCLIC(c) distribution was presented in [24]. In [24], the BLOCK-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 BLOCK-CYCLIC(k) distribution can be generated in terms of unions and intersections of some CYCLIC distributions.

In [3], Chatterjee et al. enumerated the local memory access sequence of communication sets for array statements with BLOCK-CYCLIC(c) distribution based on a finite-state machine. In this approach, the local memory access sequence can be characterized by an FSM at most c states. In [17], Kennedy et al. also presented algorithms to compute the local memory access sequence for array statements with BLOCK-CYCLIC(c) distribution. Lee and Chen [18] derived communication sets for statements of arrays which were distributed in arbitrary BLOCK-CYCLIC(c) fashion. They also presented closed form expressions of communication sets for restricted block size. In [4], we proposed a basic-cycle calculation to efficiently generate the communication sets for array redistribution. The greatest advantage of this method is the ability of fast indexing. In [11], we proposed efficient algorithms for BLOCK-CYCLIC(kr) to BLOCK-CYCLIC(r) and BLOCK-CYCLIC(r) to BLOCK-CYCLIC(kr) redistribution. The most significant improvement of the algorithms is that a processor does not need to construct the send/receive data sets for a redistribution.

Thakur et al. [25], [26] presented algorithms for runtime 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. In [10], 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 are computed by a scanning approach that exploits the repetitive pattern of the intersection of two index sets. In [22], [23], Ramaswamy and Banerjee used a mathematical representation, *PITFALLS*, for regular data redistribution. The basic idea of *PITFALLS* is to find all intersections between source and destination distributions. Based on the intersections, the send/receive processor/data sets can be determined and general redistribution algorithms can be devised. Prylli and Touranchean [21] proposed a runtime scan algorithm for BLOCK-CYCLIC array redistribution. Their approach has the same time complexity as that proposed in [23] but has a simple basic operation compared to that proposed in [23]. The disadvantage of these approaches is that, when the number of processors is large, iterations of the outermost loop in intersection algorithms increases as well. This leads to high indexing overheads and degrades the performance of a redistribution algorithm.

The above researches focus on efficient generation of communication sets. For the communication part, a spiral mapping technique [32] 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 [12], [13] proposed a processor mapping technique to minimize the

amount of data exchange for BLOCK to BLOCK-CYCLIC(r) redistribution and vice versa. Using the data to logical processor mapping, they show that the technique can achieve the maximum ratio between data retained locally and the total amount of data exchanged. 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 a synchronous and an asynchronous scheme. Since the excessive synchronization overheads occurred from the synchronous scheme, they also presented the random and optimal scheduling algorithms for BLOCK-CYCLIC array redistribution.

Kaushik et al. [14], [15] proposed a multiphase redistribution approach for BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution. The main idea of multiphase redistribution is to perform a redistribution as a sequence of redistributions such that the communication cost of data movement among processors in the sequence is less than that of direct redistribution. Instead of redistributing the entry 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 [19], 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. They also extended this technique to solve some multidimensional redistribution problems [20]. However, as the array size increased, the above methods will have a large amount of extra transmission costs and degrades the performance of a redistribution algorithm.

3 Preliminaries

In general, a BLOCK-CYCLIC(s) over P processors to BLOCK-CYCLIC(t) over Q processors redistribution can be classified as one of three types:

1. s is divisible by t, i.e., BLOCK-CYCLIC($s = kr$) to BLOCK-CYCLIC($t = r$) redistribution,
2. t is divisible by s, i.e., BLOCK-CYCLIC($s = r$) to BLOCK-CYCLIC($t = kr$) redistribution, and
3. s is not divisible by t and t is not divisible by s.

To simplify the presentation, we use $kr_{(P)} \rightarrow r_{(Q)}$, $r_{(P)} \rightarrow kr_{(Q)}$, and $s_{(P)} \rightarrow t_{(Q)}$ to represent the first, the second, and the third types of redistribution, respectively, for the rest of the paper. In this section, we first present the terminology used in this 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 $s_{(P)} \rightarrow t_{(Q)}$, the source local array of processor P_i, denoted by $SLA_i[0 : N/P - 1]$, is defined as the set of array elements that are distributed to processor P_i in the source distribution, where $0 \leq i \leq P - 1$. The destination local array of processor Q_j, denoted by $DLA_j[0 : N/Q - 1]$, is defined as the set of array elements that are distributed to processor Q_j in the destination distribution, where $0 \leq j \leq Q - 1$.*

Definition 3. *Given an $s_{(P)} \rightarrow t_{(Q)}$ redistribution on $A[1 : N]$, the source processor of an array element in $A[1 : N]$ or $DLA_j[0 : N/Q - 1]$ is defined as the processor that owns the array element in the source distribution, where $0 \leq j \leq Q - 1$. The destination processor of an array element in $A[1 : N]$ or $SLA_i[0 : N/P - 1]$ is defined as the processor that owns the array element in the destination distribution, where $0 \leq i \leq P - 1$.*

Definition 4. *Given an $s_{(P)} \rightarrow t_{(Q)}$ redistribution on $A[1 : N]$, a global complete cycle (GCC) of $A[1 : N]$ is defined as $GCC = lcm(s \times P, t \times Q)$. We define $A[1 : GCC]$ as the first global complete cycle of $A[1 : N]$,*

$$A[GCC + 1 : 2 \times GCC]$$

as the second global complete cycle of $A[1 : N]$, and so on.

Definition 5. *Given an $s_{(P)} \rightarrow t_{(Q)}$ redistribution on $A[1 : N]$, a local complete cycle of a local array is defined as $LCC_s = GCC/P$ in the source distribution and $LCC_d = GCC/Q$ in the destination distribution. We define*

$$SLA_i[0 : LCCs - 1](DLA_j[0 : LCC_d - 1])$$

as the first local complete cycle of

$$SLA_i[0 : N/P - 1](DLA_j[0 : N/Q - 1])$$

and

$$SLA_i[LCC_s : 2 \times LCC_s - 1](DLA_j[LCC_d : 2 \times LCC_d - 1])$$

as the second local complete cycle of

$$SLA_i[0 : N/P - 1]\ (DLA_j[0 : N/Q - 1])$$

and so on.

We now give examples to clarify the above definitions. Given a one-dimensional array $A[1 : 100]$ and $P = Q = 5$ processors, Fig. 1 shows a BLOCK to BLOCK-CYCLIC(10) redistribution on A over five processors. In this paper, we assume that the local array index starts from 0 and the global array index starts from 1. According to Definitions 4 and 5, the size of global complete cycle (GCC) is equal to 100 and the size of the local complete cycle is equal to 20 in both source and destination distributions.

4 The Generalized Processor Mapping Technique for $kr_{(P)} \rightarrow r_{(Q)}$ and $r_{(P)} \rightarrow kr_{(Q)}$ Array Redistribution

To perform the redistribution shown in Fig. 1, computation as well as communication costs are required in array redistribution. The computation cost consists of the indexing time and the packing/unpacking time. The communication cost includes message startup time and data transmission time. In general, the communication cost

Source distribution : BLOCK					
Global Array Index	1 ~ 20	21 ~ 40	41 ~ 60	61 ~ 80	81 ~ 100
Local Array Index	0 ~ 19	0 ~ 19	0 ~ 19	0 ~ 19	0 ~ 19
Source Processor	P_0	P_1	P_2	P_3	P_4

↓

Destination distribution: BLOCK-CYCLIC(10)										
Global Array Index	1 ~ 20		21 ~ 40		41 ~ 60		61 ~ 80		81 ~ 100	
Local Array Index	0 ~ 9	0 ~ 9	0 ~ 9	0 ~ 9	0 ~ 9	10 ~ 19	10 ~ 19	10 ~ 19	10 ~ 19	10 ~ 19
Destination processor	P_0	P_1	P_2	P_3	P_4	P_0	P_1	P_2	P_3	P_4

Fig. 1. A BLOCK to BLOCK-CYCLIC(10) redistribution on a one-dimensional array $A[1:100]$ over five processors.

is more expensive in terms of the execution time than the computation cost. Therefore, techniques for reducing communication costs are very important. In [12], a processor mapping technique was proposed to minimize the amount of data exchange in a redistribution. The proposed techniques addressed the case of BLOCK to BLOCK-CYCLIC(x) redistribution. Fig. 2a shows an example of the processor mapping technique for the redistribution shown in Fig. 1. In Fig. 2a, in the destination distribution, the "NS" represents the *normal sequence* of logical processor ranks that start from 0 to M-1, where M is the number of processors. The "MS" represents the *mapping sequence* of logical processor ranks that is generated by the mapping function of the processor mapping technique. The shaded portions represent the data that were retained on the same logical processor through the redistribution. In the normal sequence scheme, there are 20 array elements retained locally. However, in the mapping sequence scheme, there are 50 array elements retained locally. Since the global array size is equal to 100, the processor mapping technique provides 30 percent improvement in terms of data transmission time for the redistribution shown in Fig. 1.

We consider another two examples. Fig. 2b and Fig. 2c show the redistribution with different array sizes and destination distribution factors, respectively. In Fig. 2b, a BLOCK to BLOCK-CYCLIC(10) redistribution with larger array size $A[1:500]$ is shown. Both the normal sequence scheme and the mapping sequence scheme have the same amount of array elements retained locally. The processor mapping technique does not provide a larger amount of local data in this case. In Fig. 2c, a BLOCK to BLOCK-CYCLIC(4) redistribution on a one-dimensional array $A[1:100]$ over five processors is shown. Similar to the result of Fig. 2b, the normal sequence and the mapping sequence schemes have the same amount of array elements retained locally. We have the following two observations:

1. Given a BLOCK to BLOCK-CYCLIC(r) redistribution with fixed destination distribution factor r, the processor mapping technique is not effective when the array size is larger than the threshold.
2. Given a BLOCK to BLOCK-CYCLIC(r) redistribution with fixed array size N, the processor mapping technique is not effective when the destination distribution factor r is smaller than the threshold.

In fact, BLOCK to BLOCK-CYCLIC(r) redistribution (or vice versa) is a special case of $kr_{(P)} \rightarrow r_{(Q)}$ (or $r_{(P)} \rightarrow kr_{(Q)}$) array redistribution, when $k = N/P_r$ (or $k = N/Q_r$) where N is the array size. For general redistribution problems, we derive a generalized processor mapping technique for $kr_{(P)} \rightarrow r_{(Q)}$ (or $r_{(P)} \rightarrow kr_{(Q)}$) array redistribution to minimize the amount of data exchange.

According to the values of LCC_s, LCC_d, and kr, $kr_{(P)} \rightarrow r_{(Q)}$ and $r_{(P)} \rightarrow kr_{(Q)}$ array redistributions can be classified into two different types: *optimal type* and *general type*, as shown in Table 1. In the optimal type, the generalized processor mapping technique can derive a mapping sequence such that the amount of data exchange is minimal. In the general type, the generalized processor mapping technique can derive a mapping sequence to reduce the amount of data exchange. We will discuss the generalized processor mapping technique for the optimal type and the general type in Section 4.1 and Section 4.2, respectively.

4.1 The Optimal Type

4.1.1 $kr_{(P)} \rightarrow r_{(Q)}$ Array Redistribution

A. P = Q: Based on the characteristics of a redistribution, we have the following lemma:

Lemma 1. *Given an $s \rightarrow t$ redistribution on $A[1:N]$ over M processors, for a source processor P_i, $SLA_i[m]$, $SLA_i[m + LCC]$, $SLA_i[m + 2 \times LCC], \ldots,$ and*

$$SLA_i[m + N/M \times LCC]$$

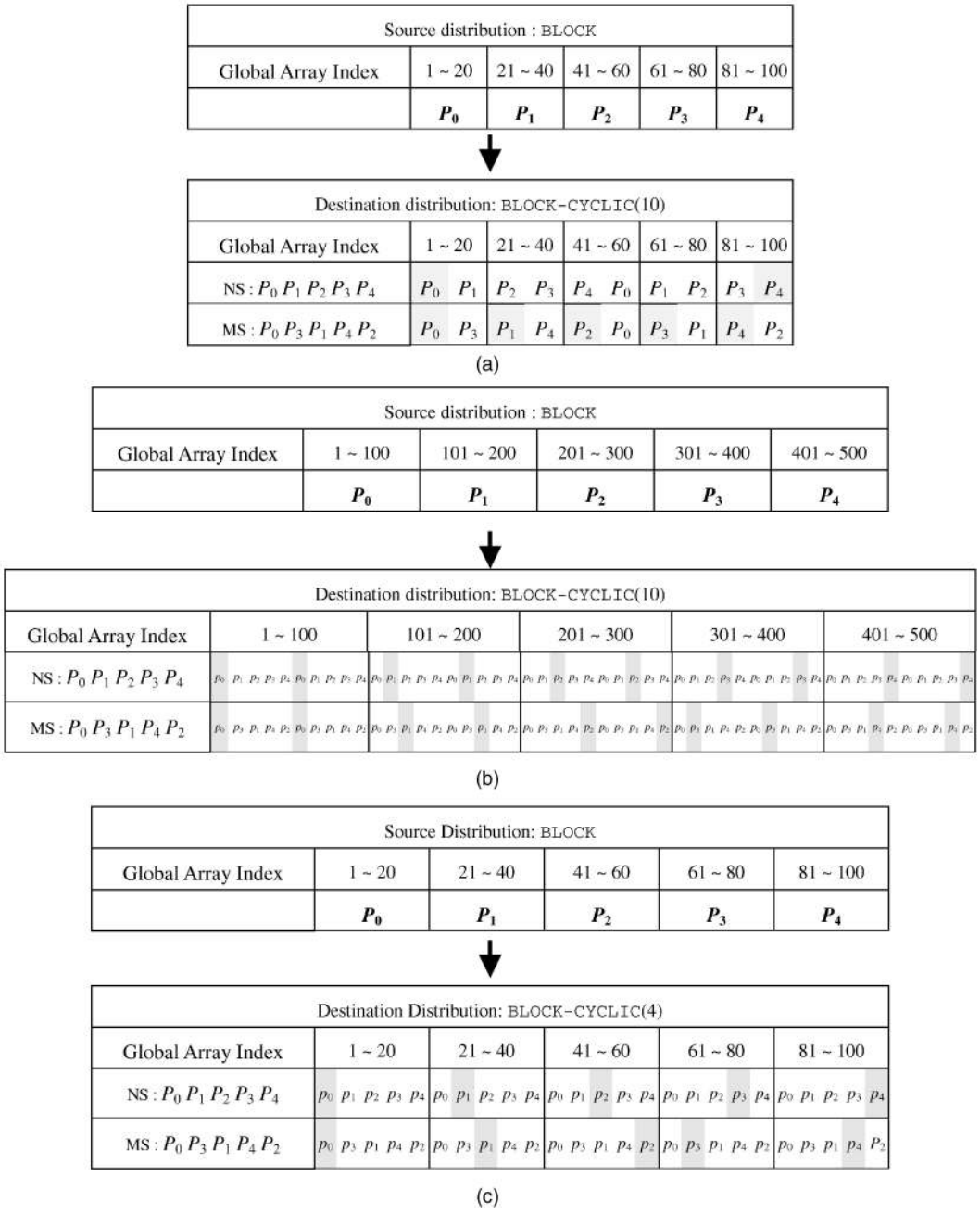


Fig. 2. Examples of processor mapping technique for BLOCK to BLOCK-CYCLIC(*r*) redistribution on $A[1:N]$ over five processors. (a) $r = 10$, $N = 100$. (b) $r = 10$, $N = 500$. (c) $r = 4$, $N = 100$.

have the same destination processor, where $0 \le i \le M-1$ *and* $0 \le m \le LCC - 1$.

Proof. In a $s \rightarrow r$ redistribution, $GCC = M \times lcm(s,t)$ and $LCC = lcm(s,t)$. In the source distribution, for a source processor P_i, if the global array index of $SLA_i[m]$ is α, then the global array indices of $SLA_i[m + LCC]$, $SLAi[m + 2 \times LCC], \ldots,$ and $SLA_i[m + N/M - LCC]$ are $\alpha + GCC$, $\alpha + 2 \times GCC, \ldots,$ and $\alpha + N/M - GCC$, respectively, where $0 \le i \le M-1$ and

$$0 \le m \le LCC - 1.$$

Since $GCC = M \times lcm(s,t)$ and $LCC = lcm(s,t)$, in the destination distribution, if $A[\alpha]$ is distributed to the destination processor P_j, so are $A[\alpha + GCC]$,

$$A[\alpha + 2 \times GCC], \ldots,$$

and $A[\alpha + N/M - GCC]$, where $0 \le j \le M-1$ and $1 \le \alpha \le GCC$. □

Lemma 2. *Given an* $s \rightarrow t$ *redistribution on* $A[1:N]$ *over* M *processors, for a destination processor* P_j, $DLA_j[m]$, $DLA_j[m + LCC]$, $DLA_j[m + 2 \times LCC], \ldots,$ *and*

$$DLA_j[m + N/M - LCC]$$

TABLE 1
The Optimal and General Types of $kr_{(P)} \rightarrow r_{(Q)}$ and $r_{(P)} \rightarrow kr_{(Q)}$ Array Redistribution

Optimal Type	*General Type*
1. $kr_{(P)} \rightarrow r_{(Q)}$ with $LCC_s = kr$	1. $kr_{(P)} \rightarrow r_{(Q)}$ with $LCC_s \neq kr$
2. $r_{(P)} \rightarrow kr_{(Q)}$ with $LCC_d = kr$	2. $r_{(P)} \rightarrow kr_{(Q)}$ with $LCC_d \neq kr$

have the same source processor, where $0 \leq j \leq M-1$ *and* $0 \leq m \leq LCC-1$.

Proof. The proof of this lemma is similar to Lemma 1. □

Given a one-dimensional array $A[1:100]$ and $M = 5$ processors, Fig. 3 shows a BLOCK-CYCLIC(10) to BLOCK-CYCLIC(5) redistribution on A over M processors. According to Lemmas 1 and 2, we know that each local complete cycle (*LCC*) has the same communication patterns. In Fig. 3, for source processor P_2, array elements $SLA_2[0:9]$ and $SLA_2[10:19]$ are in the first and the second *LCC*, respectively. $SLA_2[0:9]$ and $SLA_2[10:19]$ have the same communication patterns. Therefore, for $kr \rightarrow r$ redistribution, a processor only needs to construct the communication sets for its first *LCC*. Then, it can perform the redistribution. Similarly, to present the generalized processor mapping technique, we only discuss how to derive a mapping sequence in the first *LCC*.

Given a $kr \rightarrow r$ redistribution on a one-dimensional array $A[1:N]$ over M processors, we use

$$< P_0, P_1, P_2, \ldots, P_{M-1} >$$

and

$$< P_{\alpha(0)}, P_{\alpha(1)}, P_{\alpha(2)}, \ldots, P_{\alpha(M-1)} >$$

to represent the normal sequence and the mapping sequence, respectively, where $\alpha(j)$ represents the new logical processor rank of P_j. The main idea of the generalized processor mapping technique is to distribute the global array elements onto destination processors according to the mapping sequence instead of the normal sequence in the destination distribution. For a destination processor P_j, the new logical processor rank of P_j can be determined by the following equation:

$$\alpha(j) = (j \bmod k) \times \left\lceil \frac{M}{k} \right\rceil + \left\lceil \frac{j}{k} \right\rceil, \quad (1)$$

where $j = 0$ to $M-1$.

Fig. 4 shows a BLOCK-CYCLIC(10) to BLOCK-CYCLIC(5) redistribution on a one-dimensional array $A[1:100]$ over five processors. There are two kinds of logical processor sequences illustrated in this example. Since the normal sequence of destination processor ranks is P_0, P_1, P_2, P_3, and P_4. According to (1), the new ranks of destination processors P_0, P_1, P_2, P_3, and P_4 are equal to 0, 3, 1, 4, and 2, respectively. Therefore, the mapping sequence of destination processors is P_0, P_3, P_1, P_4, and P_2. From Fig. 4, we can see that there are 20 array elements retained locally in a normal sequence scheme while there are 50 array elements retained locally in a mapping sequence scheme. The generalized processor mapping technique provides a larger amount of local data. The following lemma shows that the mapping sequence generated by (1) can achieve the maximum amount of data that was retained on the same logical processor through a redistribution:

Lemma 3. *Given a* $kr \rightarrow r$ *redistribution on a one-dimensional array* $A[1:N]$ *over M processors, (1) determines a mapping sequence of destination processors to achieve the maximum ratio* $\lceil \frac{k}{M} \rceil : k$, *between local data and the global array size.*

Proof. We prove the lemma in two parts: 1) The maximum ratio is $\lceil \frac{k}{M} \rceil : k$. 2) The mapping sequence generated by (1) can achieve the maximum ratio $\lceil \frac{k}{M} \rceil : k$.

1. Given a $kr \rightarrow r$ redistribution, for a source processor P_i, where $0 \leq i \leq M-1$: If $k < M$, then at most r elements are retained on the local array in each local complete cycle. Since there are M local complete cycles in a *GCC*, the total

Source : BLOCK-CYCLIC(10)

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P_0	1	2	3	4	5	6	7	8	9	10	51	52	53	54	55	56	57	58	59	60
P_1	11	12	13	14	15	16	17	18	19	20	61	62	63	64	65	66	67	68	69	70
P_2	21	22	23	24	25	26	27	28	29	30	71	72	73	74	75	76	77	78	79	80
P_3	31	32	33	34	35	36	37	38	39	40	81	82	83	84	85	86	87	88	89	90
P_4	41	42	43	44	45	46	47	48	49	50	91	92	93	94	95	96	97	98	99	100

Destination : BLOCK-CYCLIC(5)

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P_0	1	2	3	4	5	26	27	28	29	30	51	52	53	54	55	76	77	78	79	80
P_1	6	7	8	9	10	31	32	33	34	35	56	57	58	59	60	81	82	83	84	85
P_2	11	12	13	14	15	36	37	38	39	40	61	62	63	64	65	86	87	88	89	90
P_3	16	17	18	19	20	41	42	43	44	45	66	67	68	69	70	91	92	93	94	95
P_4	21	22	23	24	25	46	47	48	49	50	71	72	73	74	75	96	97	98	99	100

Fig. 3. Communication patterns of SLA_2 in BLOCK-CYCLIC(10) to BLOCK-CYCLIC(5) redistribution.

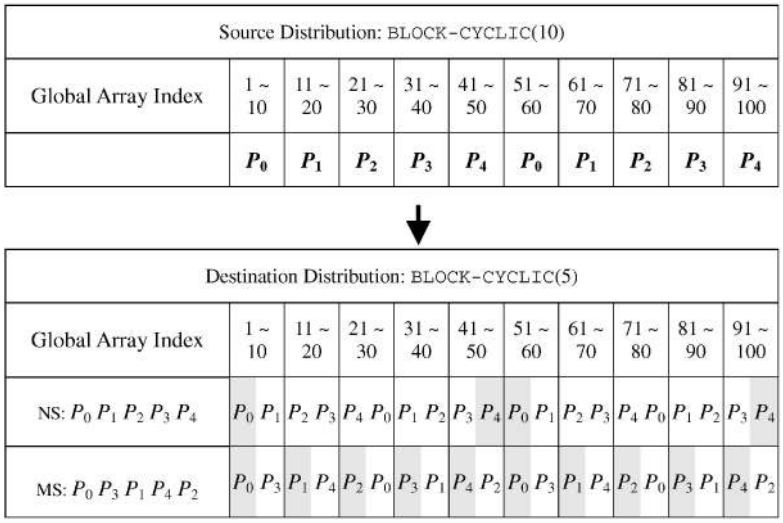


Fig. 4. $kr \rightarrow r$ redistribution with normal sequence $< P_0, P_1, P_2, P_3, P_4 >$ and mapping sequence $< P_0, P_3, P_1, P_4, P_2 >$.

amount of data that is retained on the local array is Mr. In a $kr \rightarrow r$ redistribution, $GCC = Mkr$, therefore, the ratio between local data and the number of array elements in a *GCC* is

$$Mr : GCC = Mr : Mkr = 1 : k.$$

This is equal to $\lceil \frac{k}{M} \rceil : k = 1 : k$, when $k < M$.

If $k \geq M$, then at most $\lceil \frac{k}{M} \rceil$ elements are retained on local array in each local complete cycle. Since there are M local complete cycles in a *GCC*, the total amount of data retained on local array is $Mr \times \lceil \frac{k}{M} \rceil$. In a $kr \rightarrow r$ redistribution, $GCC = Mkr$, therefore, the ratio between local data and the number of array elements in a *GCC* is $Mr : GCC = Mr \times \lceil \frac{k}{M} \rceil : Mkr = \lceil \frac{k}{M} \rceil : k$.

From the above description, the maximum ratio is $\lceil \frac{k}{M} \rceil : k$.

2. In a $kr \rightarrow r$ redistribution, each *GCC* has the same communication patterns, therefore, we only need to prove that the mapping sequence can achieve the maximum ratio $\lceil \frac{k}{M} \rceil : k$ in the first *GCC*. In a $kr \rightarrow r$ redistribution, there are M local complete cycles in each *GCC* and are denoted as LCC_0, $LCC_1, \ldots,$ and LCC_{M-1}, respectively.

If $k < M$, in the source distribution, the source processors of array elements in LCC_0, $LCC_1, \ldots,$ and LCC_{M-1} (i.e., $A[1 : kr]$, $A[kr + 1 : 2kr], \ldots,$ and $A[(M-1) \times kr + 1 : Mkr]$) are P_0, $P_1,$, and P_{M-1}, respectively. In the destination distribution, the destination processors of the first r array elements of local complete cycles LCC_0, $LCC_1, \ldots,$ and LCC_{M-1} (i.e., $A[1 : r]$,

$$A[kr + 1 : kr + r], \ldots,$$

and

$$A[(M-1) \times kr + 1 : (M-1) \times kr + kr])$$

are P_0, $P_{k \bmod M}$, $P_{2k \bmod M}, \ldots,$ and $P_{(M-1)k \bmod M}$, respectively. According to (1), the new logical processor ranks for P_0, $P_{k \bmod M}$, $P_{2k \bmod M}, \ldots,$ and $P_{(M-1)k \bmod M}$ are equal to $\alpha(0) = 0$

$$\alpha(k \bmod M) = 1,$$

$\alpha(2k \bmod M) = 2, \ldots,$ and

$$\alpha((M-1) \bmod M) = M - 1,$$

respectively. Therefore, there are Mr array elements retained on the same logical processor in the source and destination distribution in a *GCC*. The ratio between local data and global array size in a *GCC* is equal to

$$Mr : GCC = Mr : Mkr = 1 : k.$$

This is equal to $\lceil \frac{k}{M} \rceil : k = 1 : k$, when $k < M$ That means the mapping sequence can achieve the ratio $1 : k$.

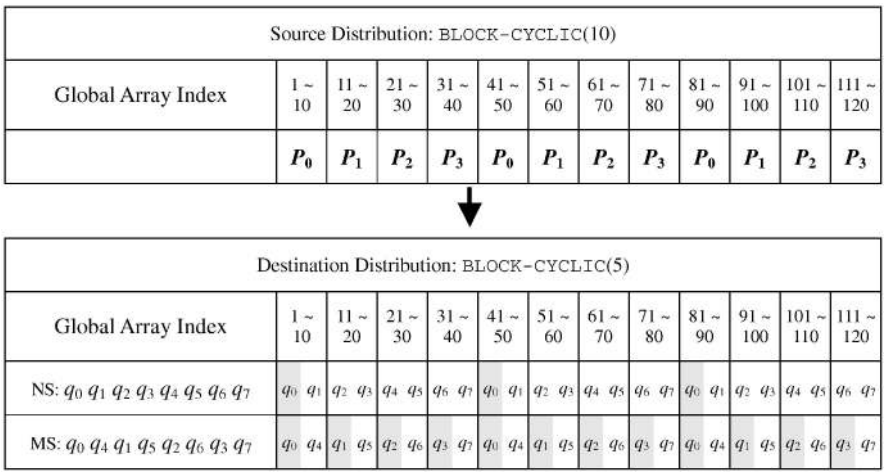
For the case of $k \geq M$, the proof of this part is similar to above. Therefore, from these two parts, we know that (1) can determine a logical sequence of destination processors to achieve the maximum ratio $\lceil \frac{k}{M} \rceil : k$, between local data and global array size. □

B. P ≠ Q: Given a $kr_{(P)} \rightarrow r_{(Q)}$ redistribution on a one-dimensional array $A[1 : N]$, we use $< q_0, q_1, q_2, , q_{Q-1} >$ and $< q_{\chi(0)}, q_{\chi(1)}, q_{\chi(2)}, \ldots, q_{\chi(Q-1)} >$ to represent the normal sequence and the mapping sequence, respectively, where $\chi(j)$ represents the new logical processor rank of q_j. For a destination processor q_j, the new logical processor rank of q_j can be determined by the following equation:

$$\chi(j) = (j \bmod k) \times \left\lceil \frac{Q}{k} \right\rceil = \left\lfloor \frac{j}{k} \right\rfloor, \tag{2}$$

where $j = 0$ to $Q - 1$.

An example of the generalized processor mapping technique for $kr_{(P)} \rightarrow r_{(Q)}$ redistribution with different source and destination processor sets is shown in Fig. 5. In Fig. 5, there are four source processors and eight destination processors. According to (2), the mapping sequence of



Source Distribution: BLOCK-CYCLIC(10)												
Global Array Index	1 ~ 10	11 ~ 20	21 ~ 30	31 ~ 40	41 ~ 50	51 ~ 60	61 ~ 70	71 ~ 80	81 ~ 90	91 ~ 100	101 ~ 110	111 ~ 120
	P_0	P_1	P_2	P_3	P_0	P_1	P_2	P_3	P_0	P_1	P_2	P_3

Destination Distribution: BLOCK-CYCLIC(5)												
Global Array Index	1 ~ 10	11 ~ 20	21 ~ 30	31 ~ 40	41 ~ 50	51 ~ 60	61 ~ 70	71 ~ 80	81 ~ 90	91 ~ 100	101 ~ 110	111 ~ 120
NS: $q_0\ q_1\ q_2\ q_3\ q_4\ q_5\ q_6\ q_7$	$q_0\ q_1$	$q_2\ q_3$	$q_4\ q_5$	$q_6\ q_7$	$q_0\ q_1$	$q_2\ q_3$	$q_4\ q_5$	$q_6\ q_7$	$q_0\ q_1$	$q_2\ q_3$	$q_4\ q_5$	$q_6\ q_7$
MS: $q_0\ q_4\ q_1\ q_5\ q_2\ q_6\ q_3\ q_7$	$q_0\ q_4$	$q_1\ q_5$	$q_2\ q_6$	$q_3\ q_7$	$q_0\ q_4$	$q_1\ q_5$	$q_2\ q_6$	$q_3\ q_7$	$q_0\ q_4$	$q_1\ q_5$	$q_2\ q_6$	$q_3\ q_7$

Fig. 5. $kr_{(P)} \rightarrow r_{(Q)}$ redistribution on $A[1:N]$ with different sequence of destination processor ranks, where $k=2$, $r=5$, $N=120$, $P=4$, $Q=8$.

destination processors q_0, q_1, q_2, q_3, q_4, q_5, q_6, and q_7 are equal to 0, 4, 1, 5, 2, 6, 3, and 7, respectively. In the mapping sequence scheme, there are 20 array elements retained locally in a global complete cycle. Since $GCC = 40$, the ratio between local data and global array size is equal to $20:40 = 1:2$. According to Lemma 3, the mapping sequence of $< q_0, q_1, q_5, q_2, q_6, q_3, q_7 >$ achieves the maximum ratio $\left\lceil \frac{k}{Q} \right\rceil : k$ for the redistribution shown in Fig. 5. The following Lemma shows that the generalized processor mapping technique can achieve the maximum amount of data retained locally for $kr_{(P)} \rightarrow r_{(Q)}$ redistribution, if LCC_s is equal to kr.

Lemma 4. *Given a $kr_{(P)} \rightarrow r_{(Q)}$ redistribution on a one-dimensional array $A[1:N]$: If $LCC_s = kr$, (2) determines a mapping sequence of destination processors to achieve the maximum ratio $\left\lceil \frac{k}{Q} \right\rceil : k$ between local data and global array size.*

Proof. We prove the lemma in two parts: 1) The maximum ratio is $\left\lceil \frac{k}{Q} \right\rceil : k$. 2) The mapping sequence generated by (2) can achieve the maximum ratio.

1. Given a $kr_{(P)} \rightarrow r_{(Q)}$ redistribution, for a source processor P_i, where $0 \le i \le P-1$: If $k < Q$, then at most r elements are retained on the local array in each local complete cycle. Since there are P local complete cycles in a GCC, the total amount of data retained on the local array is Pr. In a $kr_{(P)} \rightarrow r_{(Q)}$ redistribution, $GCC = Pkr$, therefore, the ratio between local data and the number of array elements in a GCC is

$$Pr : GCC = Pr : Pkr = 1 : k.$$

This is equal to $\left\lceil \frac{k}{Q} \right\rceil : k = 1 : k$, when $k < Q$.

If $k \ge Q$, then at most $\left\lceil \frac{k}{Q} \right\rceil$ elements are retained on local array in each local complete cycle. Since there are P local complete cycles in a GCC, the total amount of data that is retained on the local array is $Pr \times \left\lceil \frac{k}{Q} \right\rceil$. In a $kr_{(P)} \rightarrow r_{(Q)}$ redistribution, $GCC = Pkr$, therefore, the ratio between the local data and the number of array elements in a GCC is

$$Pr \times \left\lceil \frac{k}{Q} \right\rceil : GCC = Pr \times \left\lceil \frac{k}{Q} \right\rceil : Pkr = \left\lceil \frac{k}{Q} \right\rceil : k.$$

From the above description, the maximum ratio between local data and global array size is $\left\lceil \frac{k}{Q} \right\rceil : k$.

2. In a $kr_{(P)} \rightarrow r_{(Q)}$ redistribution, each GCC has the same communication patterns, therefore, we only need to prove that the generalized processor mapping technique can achieve the maximum ratio $\left\lceil \frac{k}{Q} \right\rceil : k$ in the first GCC. In a $kr_{(P)} \rightarrow r_{(Q)}$ redistribution, there are P local complete cycles in each GCC and are denoted as LCC_0, $LCC_1, \ldots,$ and LCC_{P-1}, respectively.

If $k < Q$, in the source distribution, the source processors of array elements in LCC_0, $LCC_1, \ldots,$ and LCC_{P-1} (i.e., $A[1:kr]$, $A[kr+1:2kr], \ldots,$ and $A[(M-1) \times kr + 1 : Mkr]$) are p_0, $p_1, \ldots,$ and $p_{(P-1)}$, respectively. In the destination distribution, the destination processors of the first r array elements of local complete cycles LCC_0, LCC_1, $LCC_2, \ldots,$ and LCC_{P-1} (i.e.,

$$A[kr+1 : kr+r], \ldots,$$

and

$$A[(M-1) \times kr + 1 : (M-1) \times kr + kr])$$

are q_0, $q_{k \bmod Q}$, $q_{2k \bmod Q}, \ldots,$ and $q_{(Q-1)k \bmod Q}$, respectively. According to (2), the new logical processor ranks for q_0, $q_{k \bmod Q}$, $q_{2k \bmod Q}, \ldots,$ and $q_{(Q-1)k \bmod Q}$ are equal to $\chi(0) = 0$, $\chi(k \bmod Q) = 1$, $\chi(2k \bmod Q) = 2, \ldots,$ and

$$\chi((Q-1)k \bmod Q) = Q-1,$$

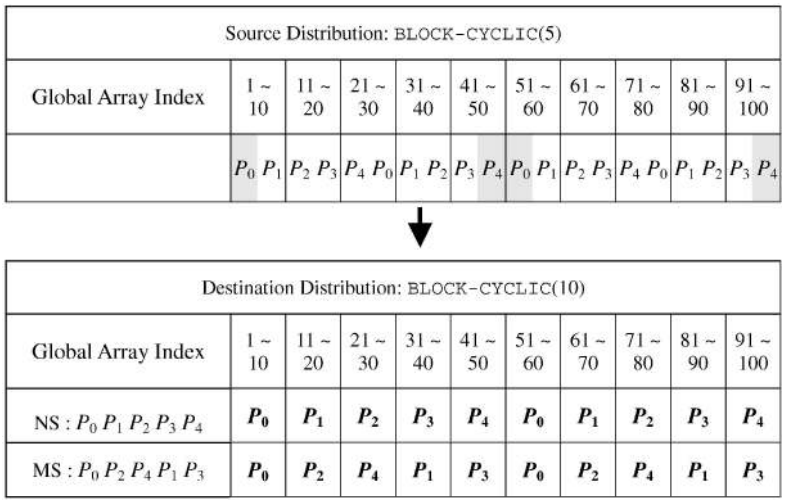


Fig. 6. $r \rightarrow kr$ redistribution with normal sequence $< P_0, P_1, P_2, P_3, P_4 >$ and mapping sequence $< P_0, P_2, _P 4, P_1, P_3 >$.

respectively. Therefore, there are Pr array elements retained on the same logical processor in the source and destination distribution in a *GCC*. The ratio between local data and global array size in a *GCC* is equal to

$$Pr : GCC = Pr : Pkr = 1 : k.$$

Since $k < Q$, we have $\left\lceil \frac{k}{Q} \right\rceil : k = 1 : k$. That means the mapping sequence can achieve the ratio $\left\lceil \frac{k}{Q} \right\rceil : k = 1 : k$.

For the case of $k \geq Q$, the proof of this part is similar to above. Therefore, we know that (2) can determine a logical sequence of destination processors to achieve the maximum ratio $\left\lceil \frac{k}{Q} \right\rceil : k$, between local data and global array size. □

4.1.2 $r_{(P)} \rightarrow kr_{(Q)}$ Array Redistribution

A. P = Q: In this section, we present the generalized processor mapping technique for $r_{(P)} \rightarrow kr_{(Q)}$ array redistribution with same source and destination processor sets. Given an $r \rightarrow kr$ redistribution on a one-dimensional array $A[1 : N]$ over *M* processors, we use $< P_0, P_1, P_2, \ldots, P_{M-1} >$ and

$$< P_{\beta(0)}, P_{\beta(1)}, P_{\beta(2)}, \ldots, P_{\beta(M-1)} >$$

to represent the normal sequence and the mapping sequence, respectively, where $\beta(j)$ represents the new logical processor rank of P_j. For a destination processor P_j, the new logical processor rank of P_j can be determined by the following equation:

$$\beta(j) = (j \bmod k) \times \left\lceil \frac{M}{k} \right\rceil + \left\lfloor \frac{j \times k}{M} \right\rfloor, \quad (3)$$

where $j = 0$ to $M - 1$.

Fig. 6 shows a BLOCK-CYCLIC(5) to BLOCK-CYCLIC(10) redistribution on a one-dimensional array $A[1 : 100]$ over five processors. In Fig. 6, two kinds of logical processor sequences are illustrated. The normal sequence of the destination processor ranks is P_0, P_1, P_2, P_3, and P_4. According to (3), the new ranks of destination processors P_0, P_1, P_2, P_3, and P_4 are equal to 0, 2, 4, 1, and 3, respectively. Therefore, the mapping sequence of destination processors is P_0, P_2, P_4, P_1, and P_3. From Fig. 6, we can see that there are 20 array elements retained locally in the normal sequence scheme while there are 50 array elements retained locally in a mapping sequence scheme. The generalized processor mapping technique provides a larger amount of local data. The following lemma states that the mapping sequence generated by (3) can achieve the maximum amount of data that is retained on the same logical processor through an $r \rightarrow kr$ redistribution.

Lemma 5. *Given an $r \rightarrow kr$ redistribution on a one-dimensional array $A[1 : N]$ over M processors, (3) determines a mapping sequence of destination processors to achieve the maximum ratio $\left\lceil \frac{k}{M} \right\rceil : k$ between local data and the global array size.*

Proof. The proof of this lemma can be easily established according to Lemma 3. □

B. P ≠ Q: Given an $r_{(P)} \rightarrow kr_{(Q)}$ redistribution on a one-dimensional array $A[1 : N]$, we use $< q_0, q_1, q_2, \ldots, q_{Q-1} >$ and $< q_{\delta(0)}, q_{\delta(1)}, q_{\delta(2)}, \ldots, q_{\delta(Q-1)} >$ to represent the normal sequence and the mapping sequence, respectively, where $\delta(j)$ represents the new logical processor rank of q_j. For a destination processor q_j, the new logical processor rank of q_j can be determined by the following equation:

$$\delta(j) = (j \bmod k) \times \left\lceil \frac{Q}{k} \right\rceil + \left\lfloor \frac{j \times k}{Q} \right\rfloor, \quad (4)$$

where $j = 0$ to $Q - 1$.

Lemma 6. *Given an $r_{(P)} \rightarrow kr_{(Q)}$ redistribution on a one-dimensional array $A[1 : N]$, if $LCC_d = kr$, (4) determines a logical sequence of destination processors to achieve the maximum ratio $\left\lceil \frac{k}{Q} \right\rceil$, between local data and global array size.*

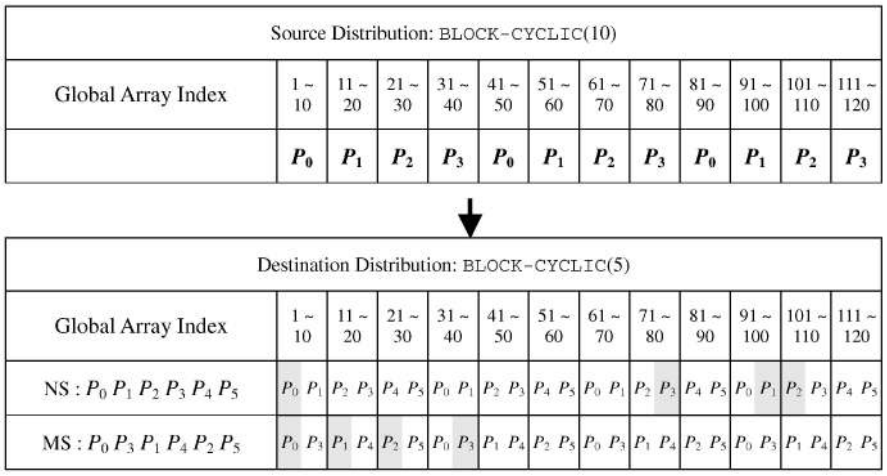


Fig. 7. $kr_{(P)} \rightarrow r_{(Q)}$ redistribution on $A[1:N]$ with different sequence of destination processor ranks, where $k=2$, $r=5$, $N=120$, $P=4$, $Q=6$.

Proof. The proof of this lemma can be easily established according to Lemma 4. □

4.2 General Type

According to Table 1, there are two types of redistribution in the general type: $kr_{(P)} \rightarrow r_{(Q)}$ redistribution with $LCC_s \neq kr$ and $r_{(P)} \rightarrow kr_{(Q)}$ redistribution with $LCC_d \neq kr$. For $kr_{(P)} \rightarrow r_{(Q)}$ redistribution, the mapping function is the same as (2). For $r_{(P)} \rightarrow kr_{(Q)}$ redistribution, the mapping function is the same as (4). Fig. 7 shows an example of $kr_{(P)} \rightarrow r_{(Q)}$ redistribution with $LCC_s = 3kr$. According to (2), the mapping sequence is $P_0, P_3, P_1, P_4, P_2, P_5$. In Fig. 7, both the normal sequence scheme and the mapping sequence scheme provide the same amount of local data. The generalized processor mapping technique does not provide a larger amount of local data than that of the normal method in this case. Fig. 8 shows another example of $kr_{(P)} \rightarrow r_{(Q)}$ redistribution with $LCC_s = 2kr$. According to (2), the mapping sequence is

$$P_0, P_6, P_1, P_7, P_2, P_8, P_3, P_9, P_4, P_{10}, P_5, P_{11}.$$

In Fig. 8, the mapping sequence scheme provides a larger amount of local data than that of the normal sequence scheme. From the above two examples, we know that the processor mapping technique provides a different improvement for different $kr_{(P)} \rightarrow r_{(Q)}$ redistribution. In Section 6, we will present a theoretical model to analyze the amount of local data in the generalized processor mapping technique. The mathematical model can also calculate the improvement of the generalized processor mapping technique for $kr_{(P)} \rightarrow r_{(Q)}$ redistribution or vice versa.

5 Multidimensional Array Redistribution

The generalized processor mapping technique can be extended to multidimensional arrays. To simplify the presentation, we use

$$\texttt{BC}(k_0 r_0, k_1 r_1, \ldots, k_{n-1} r_{n-1}) \rightarrow \texttt{BC}(r_0, r_1, \ldots, r_{n-1})$$

to represent an n-dimensional (BLOCK-CYCLIC($k_0 r_0$), BLOCK-CYCLIC($k_1 r_1$), ..., BLOCK-CYCLIC($k_{n-1} r_{n-1}$)) to (BLOCK-CYCLIC(r_0), BLOCK-CYCLIC(r_1), ..., BLOCK-CYCLIC(r_{n-1})) redistribution and

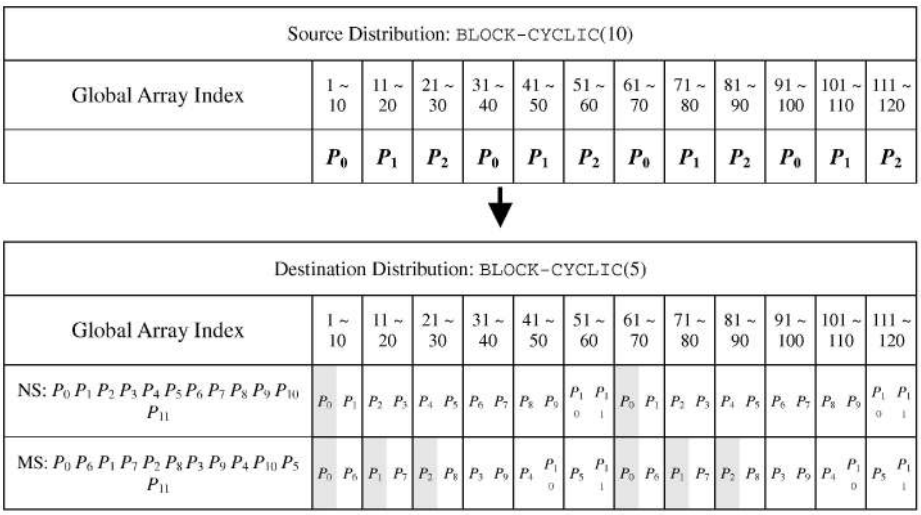


Fig. 8. $kr_{(P)} \rightarrow r_{(Q)}$ redistribution on $A[1:N]$ with different sequence of destination processor ranks, where $k=2$, $r=5$, $N=120$, $P=3$, $Q=12$.

$$\texttt{BC}(r_0, r_1, \ldots, r_{n-1}) \rightarrow \texttt{BC}(k_0 r_0, k_1 r_1, \ldots, k_{n-1} r_{n-1})$$

to represent the reverse case. Since the source and destination processor sets may be different, we use $P(P_0, P_1, \ldots, P_{n-1})$ and $Q(Q_0, Q_1, \ldots, Q_{n-1})$ to represent the source and the destination processor grids, respectively. The mapping functions 2 and 4 can be extened as follows:

Given a

$$\texttt{BC}(k_0 r_0, k_1 r_1, \ldots, k_{n-1} r_{n-1}) \rightarrow \texttt{BC}(r_0, r_1, \ldots, r_{n-1})$$

redistribution on an n-dimensional array

$$A[1 : m_0, 1 : m_1, \ldots, 1 : m_{n-1}],$$

for a destination processor q_j in the ℓth dimension, if the new logical processor rank of q_j is denoted by $\eta(j)$, then the value of $\eta(j)$ can be determined by the following equation:

$$\eta(j) = (j \bmod k_\ell) \times \left\lceil \frac{Q_\ell}{k_\ell} \right\rceil + \left\lfloor \frac{j}{k_\ell} \right\rfloor, \tag{5}$$

where $0 \leq j \leq Q_\ell - 1$ and $0 \leq \ell \leq n - 1$.

Given a

$$\texttt{BC}(r_0, r_1, \ldots, r_{n-1}) \rightarrow \texttt{BC}(k_0 r_0, k_1 r_1, \ldots, k_{n-1} r_{n-1})$$

redistribution on an n-dimensional array

$$A[1 : m_0, 1 : m_1, \ldots, 1 : m_{n-1}]$$

for a destination processor q_j in the ℓth dimension, if the new logical processor rank of q_j is denoted by $\gamma(j)$, then the value of $\gamma(j)$ can be determined by the following equation:

$$\gamma(j) = (j \bmod k_\ell) \times \left\lceil \frac{Q_\ell}{k_\ell} \right\rceil + \left\lfloor \frac{j k_\ell}{Q_\ell} \right\rfloor, \tag{6}$$

where $0 \leq j \leq Q_\ell - 1$ and $0 \leq \ell \leq n - 1$.

6 Performance Evaluation and Experimental Results

6.1 Theoretical Analysis

From the description in Section 4, we know that the generalized processor mapping technique can reduce the data transmission cost for $kr_{(P)} \rightarrow r_{(Q)}$ redistribution and vice versa. In this section, we present a theoretical model to analyze the performance of the generalized processor mapping technique.

6.1.1 $kr \rightarrow r$ and $r \rightarrow kr$ Array Redistribution

We first consider the case of $kr \rightarrow r$ and $r \rightarrow kr$ array redistribution with the same source and destination processor set. Given a $kr \rightarrow r$ (or $r \rightarrow kr$) array redistribution on a one-dimensional array $A[1 : N]$ over M processors, since each global complete cycle (GCC) has the same communication patterns, we only consider the redistributing patterns in a GCC. To analyze the normal method and the generalized processor mapping technique, we use L_{normal} and L_{mapping} to represent the amount of local data generated by normal sequence and mapping sequence in a GCC, respectively. Therefore, the total number of local data for a redistribution is equal to $L_{\text{normal}} \times \frac{N}{GCC}$ (or $L_{\text{mapping}} \times \frac{N}{GCC}$).

Given a $kr \rightarrow r$ (or $r \rightarrow kr$) redistribution on a one-dimensional array $A[1 : N]$ over M processors, the value of L_{normal} can be determined by the following equation:

$$L_{\text{normal}} = \left(\left\lfloor \frac{k}{M} \right\rfloor \times M + \mu \right) \times r, \tag{7}$$

where μ is defined as follows:

$$\mu = \sum_{i=0}^{M-1} A(i), \tag{8}$$

where $A(i)$ is defined as follows:

$$A(i) = \Gamma[((i + M) - (ik \bmod M)) \bmod M < k \bmod M], \tag{9}$$

where $\Gamma[e]$ is called Iverson's function. If the value of e is true, then $\Gamma[e] = 1$; otherwise $\Gamma[e] = 0$.

The value of L_{mapping} can be determined by the following equation:

$$L_{\text{mapping}} = \left\lceil \frac{k}{M} \right\rceil \times Mr. \tag{10}$$

According to (7) and (10), we can have the following equation:

$$L_{\text{mapping}} > L_{\text{normal}} \Leftrightarrow M > \mu. \tag{11}$$

The generalized processor mapping technique provides a larger amount of local data than that of the normal method, when $M > \mu$. Since the value of μ is smaller than or equal to M (according to (8)), the generalized processor mapping technique is effective for all $kr \rightarrow r$ (or $r \rightarrow kr$) redistribution.

6.1.2 $kr_{(P)} \rightarrow r_{(Q)}$ and $r_{(P)} \rightarrow kr_{(Q)}$ Array Redistribution

Given a $kr_{(P)} \rightarrow r_{(Q)}$ (or $r_{(P)} \rightarrow kr_{(Q)}$) redistribution with different source and destination processor sets, the theoretical analysis for $kr_{(P)} \rightarrow r_{(Q)}$ and $r_{(P)} \rightarrow kr_{(Q)}$ redistribution are constructed as follows:

- $\mathbf{kr_{(P)} \rightarrow r_{(Q)}}$: Given a $kr_{(P)} \rightarrow r_{(Q)}$ redistribution, if $LCC_s = mkr$, where m is a positive integer, the value of L_{normal} can be determined by the following equation:

$$L_{\text{normal}} = \left(\left\lfloor \frac{k}{Q} \right\rfloor \times P + \varepsilon \right) \times r, \tag{12}$$

where δ is defined as follows:

$$\varepsilon = \sum_{i=0}^{m \times P - 1} B(i), \tag{13}$$

where $B(i)$ is defined as follows:

$$B(i) = \mathbf{\Gamma}[((i \bmod P) + Q - (ik \bmod Q)) \bmod Q < k \bmod Q], \tag{14}$$

where $\mathbf{\Gamma}[e]$ is called Iverson's function. If the value of e is true, then $\mathbf{\Gamma}[e] = 1$; otherwise $\mathbf{\Gamma}[e] = 0$. The value of L_{mapping} can be determined by the following equation:

TABLE 2
The Time of *GPMT_KRR* and *KRR* to Execute Different $kr \rightarrow r$ (and $r \rightarrow kr$) Redistribution on a One-Dimensional Array on a 50-Node SP 2, Where $N = 16$ *K*Bytes and $r = 2$

	$kr \rightarrow r$						$r \rightarrow kr$					
	GPMT_KRR			*KRR*			*GPMT_KRR*			*KRR*		
	k =2	k =5	k =10	k =2	k =5	k =10	k =2	k =5	k =10	k =2	k =5	K =10
N	1.22	1.72	1.27	1.64	1.95	1.33	1.40	2.13	2.62	1.88	2.41	2.75
$N \times 10^1$	2.77	3.85	2.79	3.86	4.41	2.96	2.09	4.19	4.23	2.91	4.80	4.48
$N \times 10^2$	14.36	18.32	22.24	20.67	21.21	23.66	16.73	20.80	22.75	23.07	24.08	23.74
$N \times 10^3$	126.63	181.12	190.34	189.13	212.09	203.58	139.76	185.36	203.37	208.6	217.06	217.51
$N \times 10^4$	1313.31	1811.55	1997.03	2118.25	2146.39	2150.58	1295.98	1774.22	1960.44	2090.3	2102.16	2108.61

Time(ms)

$$L_{\text{mapping}} = \left\lceil \frac{k}{Q} \right\rceil \times Pr. \quad (15)$$

- $\mathbf{r_{(P)} \rightarrow kr_{(Q)}}$: Given an $r_{(P)} \rightarrow kr_{(Q)}$ redistribution with $LCC_d = mkr$, where m is a positive integer, the theoretical model for $r_{(P)} \rightarrow kr_{(Q)}$ redistribution can be constructed by exchanging the variables P and Q in (12) to (15).

6.2 Experimental Results

To verify the performance analysis that was presented in Section 6.1, we have implemented the generalized processor mapping technique into the algorithms proposed in [11] for $kr \rightarrow r$ and $r \rightarrow kr$ redistribution. We called algorithms with and without the generalized processor mapping technique *GPMT_KRR* and *KRR*, respectively. All algorithms were written in the single program multiple data (SPMD) programming paradigm with C+MPI codes and executed on an IBM SP2 parallel machine. To get the experimental results, each test sample with a particular array size was executed 14 times by each algorithm. The mean time of these 14 tests (except the two maximum and the two minimum values) that were executed by an algorithm was used as the time to perform a redistribution. The single-precision array was used for the test.

Table 2 shows the time of *GPMT_KRR* and *KRR* to execute different $kr \rightarrow r$ and $r \rightarrow kr$ redistribution on a 50-node SP2. From Table 2, we have the following two observations:

1. The improvement of *GPMT_KRR* increases as the value of k decreases.
2. The improvement of *GPMT_KRR* is more significant when array size increases.

The reason for the first observation is that the local data provided by the normal sequence is extremely less than that of the mapping sequence when the value of k is small. For example, for the case when k is equal to 2, the values of $\mathbf{L}_{\text{normal}}$ and $\mathbf{L}_{\text{mapping}}$ are equal to 4 and 100, respectively. In this case, GCC = 200. That means the mapping sequence provides 96/200 (= 48 percent) improvements. For the case when k is equal to 4, the values of $\mathbf{L}_{\text{normal}}$ and $\mathbf{L}_{\text{mapping}}$ are equal to 12 and 100, respectively. In this case, $GCC = 500$, the mapping sequence provides 88/500 (= 17.6 percent) improvements. Therefore, when the value of k increases, the performance of *GPMT_KRR* and *KRR* will become close. These phenomena match the performance analysis presented in Section 6.1. Fig. 9 shows the performance of *GPMT_KRR* and *KRR* to execute the redistribution samples (k = 2, 5, 10) shown in Table 2. The array size is 1.6×10^8 bytes. From Fig. 9a and Fig. 9b, we can see that the

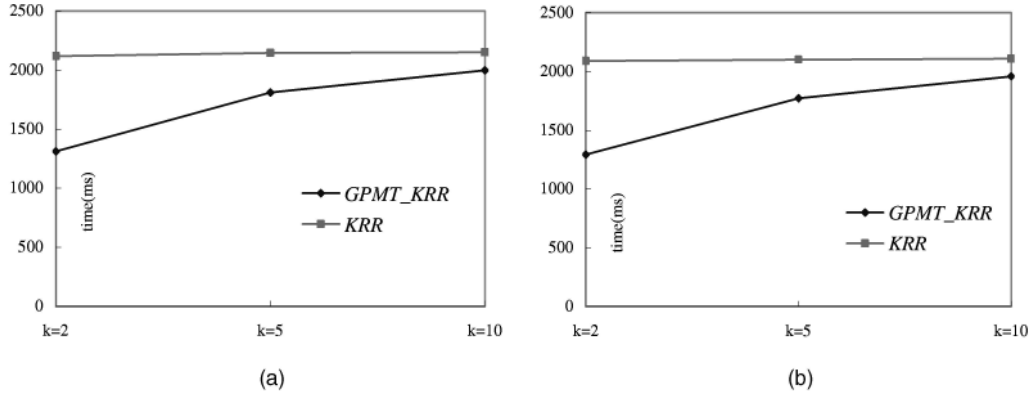


Fig. 9. Performance of different algorithms to execute $kr \rightarrow r$ redistribution and vice versa with various values of k ($N = 1.6 \times 10^8$ bytes) on a 50-node SP2. (a) $kr \rightarrow r$ redistribution. (b) $r \rightarrow kr$ redistribution.

TABLE 3
The Time of *GPMT_KRR* and *KRR* to Execute Different $kr_{(P)} \rightarrow r_{(Q)}$ (and $r_{(P)} \rightarrow kr_{(Q)}$) Redistribution on a One-Dimensional Array, Where $N = 16$ *K*Bytes, $P = 50$, $Q = 40$ and $r = 2$

	$kr_{(P)} \rightarrow r_{(Q)}$						$r_{(P)} \rightarrow kr_{(Q)}$					
	GPMT_KRR			*KRR*			*GPMT_KRR*			*KRR*		
	k =2	k =5	k =10	k =2	k =5	k =10	k =2	k =5	k =10	k =2	k =5	K =10
N	1.71	2.57	2.36	2.41	2.81	2.47	1.95	2.83	2.76	2.74	3.10	2.89
$N \times 10^1$	2.84	5.15	5.14	4.18	5.71	5.42	3.78	5.33	5.11	5.55	5.91	5.38
$N \times 10^2$	16.23	21.28	24.98	24.97	23.92	26.47	15.92	23.15	26.34	24.50	26.39	27.92
$N \times 10^3$	145.80	207.23	229.75	235.64	236.17	245.07	148.27	212.47	231.41	239.64	242.14	246.84
$N \times 10^4$	1305.33	1973.17	2102.91	2221.85	2281.13	2258.16	1348.50	1988.41	2098.59	2295.32	2298.75	2253.53

Time(ms)

performance of *GPMT_KRR* and *KRR* are approximate when the value of k increases.

The reason for the second observation is that when the array size is small, the communication time is not significant, in terms of the total time of redistribution. The improvement of the generalized processor mapping technique is not significant either. When the array size is large, the communication time dominates the performance of a redistribution. Therefore, the improvement of the generalized processor mapping technique is more significant.

Table 3 shows the time of *GPMT_KRR* and *KRR* to execute different $kr_{(P)} \rightarrow r_{(Q)}$ and $r_{(P)} \rightarrow kr_{(Q)}$ redistributions with different source and destination processor sets, where $P = 50$ and $Q = 40$. Fig. 10 shows the performance of *GPMT_KRR* and *KRR* to execute the redistribution samples ($k = 2, 5, 10$) shown in Table 3. From Table 3 and Fig. 10, we

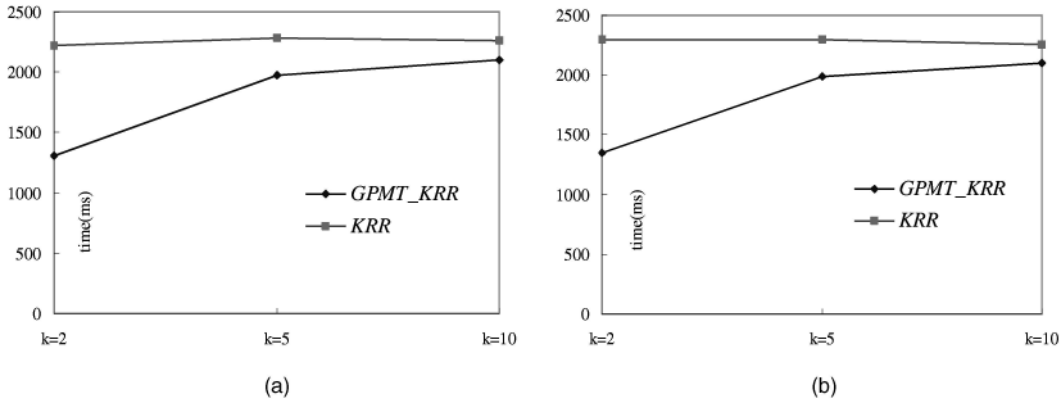


Fig. 10. Performance of different algorithms to execute $kr_{(P)} \rightarrow r_{(Q)}$ redistribution and vice versa with various values of k ($N = 1.6 \times 10^8$ bytes) on SP2, where $P = 50$ and $Q = 40$. (a) $kr_{(P)} \rightarrow r_{(Q)}$ redistribution. (b) $r_{(P)} \rightarrow kr_{(Q)}$ redistribution.

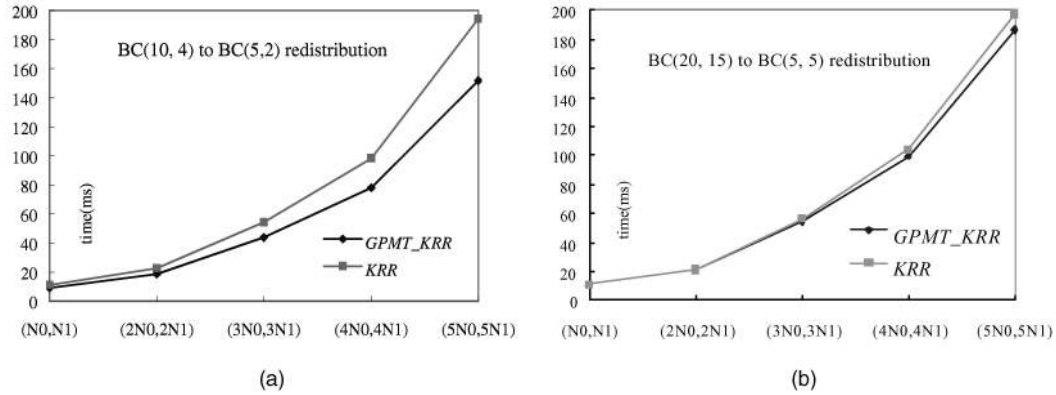


Fig. 11. Performance of different algorithms to execute two-dimensional array redistribution on a 50-node SP2. (a) $kr \rightarrow r$ redistribution. (b) $kr \rightarrow r$ redistribution.

TABLE 4
The Time of *GPMT_KRR* and *KRR* to Execute Different BLOCK-CYCLIC Redistribution on a Two-Dimensional Array on a 50-Node SP 2, $(N_0, N_1) = (640, 720)$ Bytes

	BC(10, 4) → BC(5, 2)		BC(20, 15) → BC(5, 5)	
	GPMT_KRR	*KRR*	*GPMT_KRR*	*KRR*
(N_0,N_1)	9.05	10.86	11.19	11.35
$(2N_0,2N_1)$	18.49	22.57	20.76	21.36
$(3N_0,3N_1)$	43.68	54.13	54.10	56.46
$(4N_0,4N_1)$	78.02	98.24	99.33	104.41
$(5N_0,5N_1)$	151.71	194.20	186.66	197.64

Time(ms)

have similar observations as those obtained from Table 2 and Fig. 9.

Fig. 11a and Fig. 11b show the performance of *GPMT_KRR* and *KRR* to execute $\texttt{BC}(10, 4) \rightarrow \texttt{BC}(5, 2)$ and $\texttt{BC}(20, 15) \rightarrow \texttt{BC}(5, 5)$ redistributions, respectively. Table 4 shows the execution time of the redistribution shown in Fig. 11. From Fig. 11, we can see that the improvement of the generalized processor mapping technique in Fig. 11a is larger than that of the generalized processor mapping technique in Fig. 11b. For $\texttt{BC}(10, 4) \rightarrow \texttt{BC}(5, 2)$ and $\texttt{BC}(20, 15) \rightarrow \texttt{BC}(5, 5)$ redistribution, the values of (k_0, k_1) are equal to $(k_0, k_1) = (2, 2)$ and $(k_0, k_1) = (4, 3)$, respectively. The values of (k_0, k_1) in Fig. 11a are smaller than the values of (k_0, k_1) in Fig. 11b. According to the first observation in Table 2, the *GPMT_KRR* can have a larger improvement in Fig. 11a. From Fig. 11, we also observe that the improvement is more significant when the array size becomes large. The reason is the same as that described for Table 2.

From the above performance analysis and experimental results, we have the following remarks:

Remark 1. The generalized processor mapping technique can minimize the amount of data exchange for BLOCK-CYCLIC(*kr*) to BLOCK-CYCLIC(*r*) and BLOCK-CYCLIC(*r*) to BLOCK-CYCLIC(*kr*) array redistribution. The data transmission cost can be reduced.

Remark 2. The generalized processor mapping technique provides significant improvement when the value of *k* is small. However, when the value of *k* is large, the improvement of the generalized processor mapping technique will be limited.

Remark 3. The generalized processor mapping technique provides significant improvement when the array size is large.

7 Conclusions

Array redistribution is usually used in data-parallel programs to minimize the runtime cost of performing data exchange among different processors. Since it is performed at runtime, efficient methods are required for array redistribution. In this paper, we have presented a generalized processor mapping technique to minimize the amount of data needed to be communicated for BLOCK-CYCLIC(*kr*) to BLOCK-CYCLIC(*r*) array redistribution and vice versa. Based on the mathematical mapping functions, a new sequence of logical processors is derived to achieve the maximum amount of data that can be retained locally through a redistribution. The communication cost of a redistribution can be reduced when the data transmission costs become lower. The generalized processor mapping technique can handle array redistribution with arbitrary source and destination processor sets and can be applied to multidimensional arrays. The theoretical model and experimental results show that the generalized processor mapping technique can provide performance improvement over a wide range of redistribution problems. When array size is large and the value of "*k*" is small, the generalized processor mapping technique performs very well for BLOCK-CYCLIC(*kr*) to BLOCK-CYCLIC(*r*) array redistribution and vice versa.

Our techniques can only handle dense arrays and In-core programs. There are some possible extensions that could be made. One of the issues would be to consider out-of-core external array redistribution. Another important future research direction would be to investigate the redistribution techniques in irregular scientific computation programs. It would also be interesting to consider the array redistribution of sparse arrays.

Acknowledgments

The work of this paper was partially supported by NSC of ROC under contract NSC-88-2213-E-035-002.

References

[1] S. Benkner, "Handling Block-Cyclic Distribution Arrays in Vienna Fortran 90," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* 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] Y.-C. Chung, C.-H. Hsu, and S.-W. Bai, "A Basic-Cycle Calculation Technique for Efficient Dynamic Data Redistribution," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 4, pp. 359-377, Apr. 1998.

[5] F. Desprez, J. Dongarra, and A. Petitet, "Scheduling Block-Cyclic Array Redistribution," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 2, pp. 192-205, Feb. 1998.

[6] 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.

[7] 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 Computing Symp.,* pp. 504-513, 1992.

[8] 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.

[9] High Performance Fortran Forum, "High Performance Fortran Language Specification (Version 1.1)," Rice Univ., Nov. 1994.

[10] S. Hiranandani, K. Kennedy, J. Mellor-Crammey, and A. Sethi, "Compilation Technique for Block-Cyclic Distribution," *Proc. ACM Int'l Conf. Supercomputing,* pp. 392-403, July 1994.

[11] C.-H. Hsu and Y.-C. Chung, "Efficient Methods for $kr \rightarrow r$ and $r \rightarrow kr$ Array Redistribution," *J. Supercomputing,* vol. 12, no. 2, pp. 253-276, May 1998.

[12] 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.

[13] 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.

[14] S.D. Kaushik, C.H. Huang, R.W. Johnson, and P. Sadayappan, "An Approach to Communication Efficient Data Redistribution," *Proc. Int'l Conf. Supercomputing,* pp. 364-373, July 1994.

[15] 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.

[16] 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.

[17] K. Kennedy, N. Nedeljkovic, and A. Sethi, "Efficient Address Generation for Block-Cyclic Distribution," *Proc. Int'l Conf. Supercomputing,* pp. 180-184, July 1995.

[18] P.-Z. Lee and W.Y. Chen, "Compiler Techniques for Determining Data Distribution and Generating Communication Sets on Distributed-Memory Multicomputers," *Proc. 29th IEEE Hawaii Int'l Conf. System Sciences,* pp. 537-546, Jan. 1996.

[19] Y.W. Lim, P.B. Bhat, and V.K. Prasanna, "Efficient Algorithms for Block-Cyclic Redistribution of Arrays," *Proc. Eighth IEEE Symp. Parallel and Distributed Processing,* pp. 74-83, 1996.

[20] Y.W. Lim, N. Park, and V.K. Prasanna, "Efficient Algorithms for Multi-Dimensional Block-Cyclic Redistribution of Arrays," *Proc. 26th Int'l Conf. Parallel Processing,* pp. 234-241, 1997.

[21] L. Prylli and B. Tourancheau, "Fast Runtime Block Cyclic Data Redistribution on Multiprocessors," *J. Parallel and Distributed Computing,* vol. 45, pp. 63-72, Aug. 1997.

[22] S. Ramaswamy and P. Banerjee, "Automatic Generation of Efficient Array Redistribution Routines for Distributed Memory Multicomputers," *Frontier '95: The Fifth Symp. Frontiers of Massively Parallel Computation,* pp. 342-349, Feb. 1995.

[23] 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.

[24] 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.

[25] 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.

[26] R. Thakur, A. Choudhary, and J. Ramanujam, "Efficient Algorithms for Array Redistribution," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 6, June 1996.

[27] A. Thirumalai and J. Ramanujam, "HPF Array Statements: Communication Generation and Optimization," *Proc. Third Workshop on Languages, Compilers and Run-Time System for Scalable Computers,* May 1995.

[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," *Soc. Industrial and Applied Math.,* 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. Wakatani and M. Wolfe, "Optimization of Array Redistribution for Distributed Memory Multicomputers," *Parallel Computing,* vol. 21, no. 9, pp. 1485-1490, Sept. 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. 23665, Mar. 1992.

Ching-Hsien Hsu received the BS degree in computer science from Tung Hai University in 1995 and the PhD degree in information engineering from Feng Chia University in 1999. In ROTC, he is currently a teaching instructor in the Information and Multimedia Education Center at Fu Hsing Kang College. His research interests are in the areas of parallel and distributed computing, parallel algorithms, high performance compilers for data parallel programming languages, and mobile computing.

Yeh-Ching Chung received the BS degree in computer science from Chung Yuan Christian University in 1983 and the MS and the PhD degrees in computer and information science from Syracuse University in 1988 and 1992, respectively. Currently, he is a professor and the chair with the Department of Information Engineering at Feng Chia University where he directs the Parallel and Distributed Processing Laboratory. His research interests include parallel compilers, parallel programming tools, mapping, scheduling, load balancing, embedded systems and virtual reality. He is a member of the IEEE Computer Society.

Don-Lin Yang received the BE degree in computer science from Feng Chia University in 1973, the MS degree in applied science from the College of William and Mary in 1979, and the PhD degree in computer science from the University of Virginia in 1985. Prior to joining the Department of Information Engineering at Feng Chia University in 1991, he was a staff programmer at IBM Santa Teresa Laboratory from 1985 to 1987 and a member of technical staff at AT&T Bell Laboratories from 1987 to 1991. Dr. Yang is currently an associate professor. His research interests include distributed and parallel computing, image processing, data mining, and network management. He is a member of the IEEE Computer Society and the ACM.

Chyi-Ren Dow received the BS and MS degrees in information engineering from National Chiao-Tung University, Taiwan in 1984 and 1988, respectively, and the MS and PhD degrees in computer science from the University of Pittsburgh in 1992 and 1994, respectively. Currently, he is an associate professor in the Department of Information Engineering and Computer Science, Feng-Chia University, Taiwan. His research interests include distributed systems, mobile computing, and Internet engineering.