Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

A Basic-Cycle Calculation Technique for Efficient Dynamic Data Redistribution[1]

Yeh-Ching Chung[2], Ching-Sheng Sheu, and Sheng-Wen Bai

Department of Information Engineering
Feng Chia University, Taichung, Taiwan 407, ROC
Tel : 886-4-4517250 x2706
Fax : 886-4-4515517
Email : ychung, cssheu, swbai@pine.iecs.fcu.edu.tw

Abstract

In this paper, we present a basic-cycle calculation technique to efficiently perform the general BLOCK-CYCLIC(s) *to* BLOCK-CYCLIC(t) *redistribution. In the basic-cycle calculation, closed form representations for computing source/destination processors of array elements in a basic cycle, which is defined as lcm(s,t)/ gcd(s,t), are derived. To evaluate the performance of the basic-cycle calculation, we have implemented the basic-cycle calculation on an IBM SP2 parallel machine along with the multi-phase method [7]. The experimental results show that the basic-cycle calculation outperforms the two-stage multi-phase method for the case where s is not divisible by t and vice versa for all test samples. For the case where s is divisible by t or vice versa, the basic-cycle calculation outperforms the two-stage multi-phase method when the array size is over a threshold. However, in this case, the one-stage multi-phase method outperforms the basic-cycle calculation for all test samples.*

1 Introduction

Many data parallel programming languages such as High Performance Fortran (HPF) [6], Fortran D [3], Vienna Fortran [12], and High Performance C (HPC) [13] provide compiler directives for programmers to specify data array distribution. Data distribution, in general, has three types, BLOCK, CYCLIC, and BLOCK-CYCLIC(c). The BLOCK-CYCLIC(c) is the most general regular data distribution among them. Dongarra *et al* [2] has shown that these kind of distributions are essential for many dense matrix algorithms design in distributed memory machines.

In some algorithms, such as multi-dimensional fast Fourier transform [14] and the Alternative Direction Implicit (ADI) method for solving two-dimensional diffusion equations, a distribution that is well-suited for one phase may not be good for a subsequent phase in terms of performance. Data redistribution is required for those algorithms during run-time. Since data redistribution is performed at run-time, there is a performance tradeoff between the efficiency of a new data decomposition for a subsequent phase of an algorithm and the cost of redistributing data among processors. Thus efficient methods for performing data redistribution are of great importance for the development of distributed memory compilers for those languages.

Many methods for performing data redistribution have been presented in the literature. Kalns and Ni [8, 9] proposed a processor mapping technique to minimize the amount of data exchange for BLOCK distribution to BLOCK-CYCLIC(c) distribution and vice versa. In [5], Gupta *et al* derived closed form expressions to efficiently determine the send/receive processor/data sets. They also provided a virtual processor approach [4] for addressing the problem of reference index-set identification for array statements with BLOCK-CYCLIC(c) distribution. Chatterjee *et al.* [1] enumerated the local memory access sequence of communication sets based on a finite-state machine. Thakur *et al* [11] presented algorithms for run-time array redistribution in HPF programs. In [10], Ramaswamy and Banerjee used a mathe-matical representation, PITFALLS, for regular data redistribution. The basic idea of PITFALLS is to find all intersections between source and destination distributions. Based on the inter-sections, the send/receive processor/data sets can be determined and general redistribution algorithms can be devised.

Kaushik *et al.* [7] proposed a multi-phase redistribution approach for a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution. The main idea of the multi-phase method is to perform a redistribution as a

[1] The work of this paper was partially supported by NSC of R.O.C. under contract NSC-86-2213-E-035-023.
[2] Correspondence addressee.

sequence of redistributions such that the communication cost of data movement among processors in the sequence is less than that of the direct redistribution.

This paper presents a *basic-cycle calculation* technique to efficiently perform the general BLOCK-CYCLIC redistribution. In the basic-cycle calculation, closed form representations for computing source/destination processors of array elements in a *basic cycle* are derived. From the source/destination processor/data sets of a basic cycle, we can easily construct the complete send/receive processor/data sets for a redistribution.

The paper is organized as follows. In Section 2, we will introduce notations and terminology used in this paper. In Section 3, algorithm of the basic-cycle calculation will be described in details. The cost models and performance comparisons of the basic-cycle calculation and the multi-phase method will be presented in Section 4. The conclusions and future work will be given in Section 5.

2 Preliminaries

In this section, we present the notations and terminology used in this paper. In the following, we assume that a redistribution is from a BLOCK-CYCLIC(s) distribution to a BLOCK-CYCLIC(t) distribution on an one-dimensional array with N elements $A(1{:}N)$ over M processors. The 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 1: The *source distribution pattern array*, denoted by $SDPA(1{:}Y)$ of the redistribution is defined as follows:

1. There are $M{\times}s$ array elements in the *SDPA*, i.e., $Y = M{\times}s$.

2. The values of $SDPA(1{:}s)$, $SDPA$ $(s{+}1{:}\ 2s)$, ..., $SDPA((M{-}1){\times}s{+}1{:}M{\times}s)$ are equal to 0, 1, ..., M-1, respectively.

The *destination distribution pattern array*, denoted by $DDPA(1{:}Y)$ of the redistribution is defined as follows:

1. There are $M{\times}t$ array elements in the *DDPA*, i.e., $Y = M{\times}t$.

2. The values of $DDPA(1{:}t)$, $DDPA$ $(t{+}1{:}2t)$, ..., $DDPA((M{-}1){\times}t{+}1{:}M{\times}t)$ are equal to 0, 1, ..., M-1, respectively.

Definition 2: The *source distribution pattern position* (*SDPP*) of an array element $A(k)$ is defined as $((k - 1)\ mod\ (M{\times}s)) + 1$, where $k = 1, ..., N$. The *destination distribution pattern position* (*DDPP*) of an array element $A(k)$ is defined as $((k - 1)\ mod\ (M{\times}t)) + 1$, where $k = 1, ..., N$.

Definition 3: The *source local array* of a source processor P_i, denoted by $SLA_i(1{:}N/M)$, is defined as the set of array elements that it owns. The *destination local array* of a destination processor P_j, denoted by $DLA_j(1{:}N/M)$, is defined as the set of array elements that it owns.

Definition 4: A *send processor/data set* of a source processor P_i is defined as $\{(SLA_i(k), P_j) \mid P_j$ is the destination processor of $SLA_i(k)\}$; and a *receive processor/data set* of a destination processor P_j is defined as $\{(DLA_j(k), P_i) \mid P_i$ is the source processor of $DLA_j(k)\}$

Definition 5: 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 of a source (destination) local array of processor P_i (P_j), $SLA_i(BC{+}1{:}2{*}BC)$ $(DLA_j(BC{+}1{:}\ 2{*}BC))$ as the second basic cycle of a source (destination) local array of processor P_i (P_j), and so on.

Definition 6: A basic cycle can be divided into BC/s (CC/t) blocks. We define those blocks as the *source* (*destination*) *sections* of a basic cycle.

We now give examples to clarify the above definitions. In Figure 1(a), a BLOCK-CYCLIC(4) to BLOCK-CYCLIC(3) redistribution on a one-dimensional array with N=48 elements, $A(1{:}48)$, over M=2 processors is shown. The local array indices are represented as italic numbers while the global array indices are represented as normal numbers. The basic cycle of the redistribution is 12. It can be divided into 3 source sections (size = 4) for each source processor and 4 destination sections (size = 3) for each destination processor. The source distribution pattern position of $SLA_1(12)$ is equal to 8. The destination distribution pattern position of $DLA_0(12)$ is equal to 3. Figure 1(b) illustrates the source and the destination distribution pattern arrays of the redistribution.

3 The basic-cycle calculation for data redistribution

To perform a data redistribution, 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 on 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 basic idea of the basic-cycle calculation is that every processor computes 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 of every processor, the complete send/receive processor/data sets of a redistribution can be constructed.

In Figure 2, a BLOCK-CYCLIC(3) to BLOCK-CYCLIC(2) redistribution on a one-dimensional array with 48 elements over $M = 4$ processors is shown. The basic cycle of the redistribution is equal to 6, there are two basic cycles in each source/destination local array. For each source (destination) local array, the kth array elements of the first and the second basic cycles 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. Therefore, the complete send/receive processor/data sets of a redistribution can be constructed based on the calculation of the send/receive processor/data sets in the first basic cycle of a local array.

	Source : BLOCK-CYCLIC(4)																							
local	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
SLA_0	1	2	3	4	9	10	11	12	17	18	19	20	25	26	27	28	33	34	35	36	41	42	43	44
$SDPP$	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
SLA_1	5	6	7	8	13	14	15	16	21	22	23	24	29	30	31	32	37	38	39	40	45	46	47	48
$SDPP$	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8

↓

	Destination : BLOCK-CYCLIC(3)																							
local	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
DLA_0	1	2	3	7	8	9	13	14	15	19	20	21	25	26	27	31	32	33	37	38	39	43	44	45
$DDPP$	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
DLA_1	4	5	6	10	11	12	16	17	18	22	23	24	28	29	30	34	35	36	40	41	42	46	47	48
$DDPP$	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6

(a)

Source : BLOCK-CYCLIC(4) distribution pattern array								
pattern position	1	2	3	4	5	6	7	8
$SDPA$	0	0	0	0	1	1	1	1

Destination : BLOCK-CYCLIC(3) distribution pattern array						
pattern position	1	2	3	4	5	6
$DDPA$	0	0	0	1	1	1

(b)

Figure 1 : (a) A BLOCK-CYCLIC(4) to BLOCK-CYCLIC(3) redistribution with M=2 and N=48. (b) The source and the destination distribution pattern arrays of the redistribution.

Another example of a BLOCK-CYCLIC(6) to BLOCK-CYCLIC(4) redistribution on a one-dimensional array $A(1{:}N)$ with $N = 96$ elements over $M = 4$ processors is shown in Figure 3(a). The basic cycle of the redistribution is equal to 6 as well. However, the observation that we obtained from Figure 2 cannot be applied to the case shown in Figure 3(a) directly. For example, the destination processors of the second array elements in the first and the second basic cycles of the source local array of processor P_0 are P_0 and P_2, respectively. The reason for this result is that the value of $gcd(6, 4)$ is not equal to 1. By grouping every $gcd(6, 4)$ 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 $B(k) = \{A((k-1)\times gcd(6, 4)+1), \ldots, A(k \times gcd(6, 4))\}$ and $k = 1$ to $N/gcd(6, 4)$. Then, the observation that we obtained from Figure 2 can be held if we use array B for the redistribution. Example of using meta-array for the data redistribution of Figure 3(a) is shown in Figure 3(b), which is the same as that shown in Figure 2. In the following discussion, we assume that a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution on a one-dimensional array with N elements, $A(1{:}N)$, over M processors is given. We also assume that $gcd(s, t)$ is equal to 1. If $gcd(s, t)$ is not equal to 1, we use $s/gcd(s, t)$ and $t/gcd(s, t)$ as the source and destination distribution factors of the redistribution, respectively.

	Source : BLOCK-CYCLIC(3)											
local	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)											
local	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

Figure 2 : A BLOCK-CYCLIC(3) to BLOCK-CYCLIC(2) redistribution with M=4 and N=48.

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

Figure 3 : (a) BLOCK-CYCLIC(6) to BLOCK-CYCLIC(4) redistribution with M= 4 and N=96. (b) Example of grouping local arrays to meta-local arrays for the redistribution in (a).

A Send phase

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 the destination distribution pattern array. For example, in Figure 2, there are two source

sections in $SLA_2(1{:}BC)$ of a source processor P_2. The destination distribution pattern position of $SLA_2(1) = A(7)$ is equal to 7. Since array elements in a source section have consecutive global array indices, we can derive that the destination distribution pattern positions of $SLA_2(2) = A(8)$ and $SLA_2(3) = A(9)$ are equal to 8 and 1, respectively. According to Definition 1, we can determine that the corresponding destination processors of $SLA_2(1)$, $SLA_2(2)$, and $SLA_2(3)$ are equal to $DDPA(7) = P_3$, $DDPA(8) = P_3$, and $DDPA(9) = P_0$, respectively. For a source processor P_i, there are t source sections in $SLA_i(1{:}BC)$. From the above analysis, for each source processor P_i, we only need to scan t array elements in $SLA_i(1{:}BC)$ and we can determine the destination processors of array elements in $SLA_i(1{:}BC)$.

Local / source processor	1	2	3	4	5	6
P_0	P_0	P_0	P_1	P_2	P_2	P_3
P_1	P_1	P_2	P_2	P_3	P_0	P_0
P_2	P_3	P_3	P_0	P_1	P_1	P_2
P_3	P_0	P_1	P_1	P_2	P_3	P_3

Figure 4 : The send processor/data sets of the first basic cycle for a BLOCK-CYCLIC(3) to BLOCK-CYCLIC(2) redistribution.

local / source processor	1	2	3	4	5	6	7	8	9	10	11	12
P_0	P_0	P_0	P_1	P_2	P_2	P_3	P_0	P_0	P_1	P_2	P_2	P_3
P_1	P_1	P_2	P_2	P_3	P_0	P_0	P_1	P_2	P_2	P_3	P_0	P_0
P_2	P_3	P_3	P_0	P_1	P_1	P_2	P_3	P_3	P_0	P_1	P_1	P_2
P_3	P_0	P_1	P_1	P_2	P_3	P_3	P_0	P_1	P_1	P_2	P_3	P_3

Figure 5 : The complete send processor/data sets for a BLOCK-CYCLIC(3) to BLOCK-CYCLIC(2) redistribution.

Given a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution over M processors, for each source processor P_i, the destination distribution pattern position ($DDPP$) of the first array element in each source section k of $SLA_i(1{:}BC)$ can be determined by the following equation:

$$DDPP(k) = ((k \times M + rank(P_i) \times s) \bmod (M \times t) \qquad (1)$$

where k = 1 to BC/s, $rank(P_i)$ is the rank of a source processor P_i, and $rank(P_i)$ = 0 to M-1. According to Equation 1, Definition 1, and Definition 2, we can easily construct the send processor/data sets of the first basic cycles of source processors. The send processor/data sets of the first basic cycles of source processors in Figure 2 are shown in Figure 4. Since each basic cycle in a local array has the same communication pattern, for each source processor P_i, we can pack array elements that have the same array index in basic cycles to the same out buffer according to the send processor/data set of the first basic cycle of the source processor. The complete send processor/data sets of the redistribution shown in Figure 2 are shown in Figure 5.

To send out buffers to their corresponding destination processors, we need to determine the sending order of out buffers. In the basic-cycle calculation, the order of sending out buffers of a source processor to their corresponding destination processors is determined by the appearance order of destination processors in the send processor/data set of the first basic cycle of a source processor. By using this scheduling order, we can avoid node contention mostly.

B Receive phase

In the receive phase, for each destination processor, the source distribution pattern position ($SDPP$) of the first array element in each destination section k of $DLA_i(1{:}BC)$ can be determined by the following equation:

$$SDPP(k) = ((k \times M + rank(P_j) \times t) \bmod (M \times s) \qquad (2)$$

where k = 1 to BC/t, $rank(P_j)$ is the rank of a destination processor P_j, and $rank(P_j)$ = 0 to M-1. According to Equation 2, we use the same method in the send phase to construct the receive processor/data sets of the first basic cycle of destination processors. This is shown in Figure 6. The basic-cycle calculation algorithm is given as follows.

Algorithm basic_cycle_calculation(s, t, M)
/* Send phase */
1. i = MPI_Comm_rank(); $x = lcm(s, t)$; $y = gcd(s, t)$;
2. $BC = x / y$; $s = s / y$; $t = t / y$;
3. *max_local_index* = the length of the source local array of processor P_i;
4. for (k=0; k<=M-1; k++)
5. for (l=$k \times t$+1; l<= $k \times t$+t; l++) $DDPA(l) = k$;
/* Construct the send processor/data set of the first basic cycle */
6. $DDPA\_length = M \times t$;
7. while (k <= BC)
8. { $DDPP = ((k \times M + i \times s) \bmod (M \times t)$;
9. for (l=1; l <= s; l++)
10. { $template(k) = DDPA(DDPP)$; k++;
11. if ($DDPP$ = $DDPA\_length$) $DDPP$ = 1 else $DDPP$++; } }
/* Packing data sets */
12. k = 1; *index* = 1; $length_j$ = 1, where j=0, ..., M-1;
13. while (*index* <= *max_local_index*)
14. { $j = template(k)$; l = 1;
15. while ((l <= y) && (*index* <= *max_local_index*))
16. { $out\_buffer_j(length_j) = SLA_i(index)$;
17. $length_j$++; *index*++; l++; }
18. if (k = BC) k = 1 else k++; }
19. Send data sets to their corresponding destination processors.
/* Receive phase */
20. Receive data sets in_buffer_j from source processors P_j.

```
21.  max_local_index = the length of the destination local array of
     processor Pi;
22.  for (k=0; k<=M-1; k++)
23.     for (l=k × s+1; l<= k × s+s; l++)   SDPA(l) = k;
/* Construct the receive processor/data set of the first basic cycle */
24.   SDPA_length = M × s;
25.  while (k <= BC)
26.    { SDPP = ( ( k × M + i × t ) mod ( M × s );
27.      for (l=1; l <= s; l++)
28.        { template(k) = SDPA(SDPP);  k++;
29.            if (SDPP = SDPA_length) SDPP = 1 else
     SDPP++;  } }
/* Unpacking data sets */
30.  k = 1; index = 1; lengthj = 1, where j=0, ..., M-1;
31.  while (index <= max_local_index)
32.    { j = template(k); l = 1;
33.      while ((l <= y) && (index <= max_local_index))
34.        { DLAi(index) = in_bufferj(lengthj);
35.          lengthj++;  index++;  l++;  }
36.      if (k = BC) k = 1 else k++;  }
         end_of_basic_cycle_calculation
```

destination \ local	1	2	3	4	5	6
P_0	P_0	P_0	P_2	P_3	P_1	P_1
P_1	P_0	P_1	P_3	P_3	P_2	P_2
P_2	P_1	P_1	P_0	P_0	P_2	P_3
P_3	P_2	P_2	P_0	P_1	P_3	P_3

Figure 6 : The receive processor/data sets of the first basic cycle for the redistribution in Figure 2..

4 Performance evaluation and experimental results

To evaluate the performance of the basic-cycle calculation (BCC), we compare the proposed approach with the multi-phase (MP) method. We first develop cost models for both approaches. Then, we execute both algorithms on an IBM SP2 parallel machine and use the cost models to analyze the experimental results.

4.1 Cost Models

Given a one-dimensional array with N elements, $A(1{:}N)$ and M processors, the time for an algorithm to perform a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution on A over M, in general, can be modeled as follows:

$$T = T_{comp} + T_{comm}, \qquad (3)$$

where T_{comp} is the time for an algorithm to compute the source/destination processors of local array elements, pack array elements that have the same destination processors of source processors to the same out buffer, and unpack array elements that received from source processors to their corresponding positions; and T_{comm} is the time for an algorithm to send and receive data among processors.

For the basic-cycle calculation, according to Equation 3, the time to perform a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution on an array $A(1{:}N)$ over M processors can be modeled as follows:

$$T(BCC)=T_{comp}(BCC)+\delta \times T_S+(N/M) \times T_d, \qquad (4)$$

where $T_{comp}(BCC)$ is the computation time for the basic-cycle calculation 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:

$$\delta = \min\left(M, \left\lceil \frac{\max(s,t)}{\min(s,t)} \right\rceil \times \frac{BC \times \gcd(s,t)}{\max(s,t)}\right) \qquad (5)$$

For the multi-phase method, a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution may be decomposed into several phases, i.e., BLOCK-CYCLIC($s{=}s_0$) 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 $c_1 > 0$, $c_2 > 0$, $n > 0$, and $i = 1, \ldots, n$. Therefore, according to Equation 3, the time for the multi-phase method to perform a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution on an array $A(1{:}N)$ over M processors through n phases can be modeled as follows:

$$T(MP) = \sum_{i=1}^{n} T_{comp}(MP)_i + \sum_{i=1}^{n} (k_i \times T_S + (N/M) \times T_d), \qquad (6)$$

where $T_{comp}(MP)_i$ is the computation time of the multi-phase 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 $i = 1, \ldots, n$; 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.

4.2 Performance Analysis

In this subsection, we analyze the performance of the basic-cycle calculation and the multi-phase method. For the multi-phase method, in our analysis, we only consider the one-stage(1P) and two-stage(2P) multi-phase methods.

Given a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution, the relationship of s and t can be classified into three cases : Case 1 : s is not divisible by t, t is not

divisible by s, and the value of $gcd(s, t)$ is equal to 1.. Case 2 : s is not divisible by t, t is not divisible by s, and the value of $gcd(s, t)$ is not equal to 1. Case 3 : s is divisible by t or t is divisible by s, i.e., $s = kt$ or $t = rs$, for some integers k and r.

For Case 1, according to Equation 4, the time for the basic-cycle calculation to perform this redistribution is $T(BCC) = T_{comp}(BCC) + \delta \times T_S + (N/M) \times T_d$. For the multi-phase method, it takes at least two phases to perform this distribution. According to [7], a BLOCK-CYCLIC(s) to BLOCK-CYCLIC($lcm(s,t)$) redistribution followed by a BLOCK-CYCLIC($lcm(s,t)$) to a BLOCK-CYCLIC(t) redistri-bution will produce the best performance for the two-stage multi-phase method in this redistribution. The time for the two-stage multi-phase method to perform this redistribution is $T(MP) = \sum_{i=1}^{2} T_{comp}(MP)_i + (min(M,t) + min(M,s)) \times T_S + 2 \times (N/M) \times T_d$.

We have the following lemma.

Lemma 1: Given a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution on an array $A(1{:}N)$ over M processors in case 1, the communication time for the two-stage multi-phase method to perform this redistribution is always greater than that of the basic-cycle calculation.

Proof: The communication time for the basic-cycle calculation and the two-stage multi-phase 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 $\delta = \min(M, \left\lceil \frac{\max(s,t)}{\min(s,t)} \right\rceil \times \frac{BC}{\max(s,t)}) \le min(M,t)+min(M,s)$, we have $T_{comm}(BCC)<T_{comm}(MP)$. ■

For Case 2, the time for the basic-cycle calculation and for the two-stage multi-phase method to perform this redistribution is $T(BCC) = T_{comp}(BCC)+ \delta \times T_S + (N/M) \times T_d$ and $T(MP) = \sum_{i=1}^{2} T_{comp}(MP)_i + (min(M, t/gcd(s,t)) + min(M, s/gcd(s,t))) \times T_S + 2 \times (N/M) \times T_d$ respectively. We have the following lemma.

Lemma 2: Given a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution on an array $A(1{:}N)$ over M processors in case 2, the communication time for the two-stage multi-phase method to perform this redistribution is always greater than that of the basic-cycle calculation.

Proof: The proof of this Lemma is similar to Lemma 1.

From Lemma 1 and Lemma 2, we predict that the basic-cycle calculation produces better performance than the two-stage multi-phase method in case 1 and case 2, if both algorithms have the same computation time.

For Case 3, according to Equation 4, the time for the basic-cycle calculation to perform this redistribution is the same as case 1. For the multi-phase method, it can use one-stage or two-stage approaches to perform the redistribution. The time for the one-stage multi-phase method to perform this redistribution is $T(MP) = T_{comp}(MP) + min(M, max(s,t)/min(s,t)) \times T_S + (N/M) \times T_d$. We have the following lemma.

Lemma 3: Given a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution on an array $A(1{:}N)$ over M processors in case 3, the communication time for the one-stage multi-phase method to perform this redistribution is the same as that of the basic-cycle calculation.

Proof: The communication time for the basic-cycle calculation and the one-stage multi-phase method to perform this redistribution are $T_{comm}(BCC) = \delta \times T_S + (N/M) \times T_d$ and $T_{comm}(MP) = min(M, max(s,t)/min(s,t)) \times T_S + (N/M) \times T_d$, respectively. Since $\delta = \min(M, \left\lceil \frac{\max(s,t)}{\min(s,t)} \right\rceil \times \frac{BC \times \gcd(s,t)}{\max(s,t)}) = min(M, max(s,t)/min(s,t))$, we have $T_{comm}(BCC) = T_{comm}(MP)$. ■

From Lemma 3, for Case 3, we predict that the basic-cycle calculation should produce the same performance as that of the one-stage multi-phase method if both algorithms have the same computation time. The time for the two-stage multi-phase method to perform this redistribution is $T(MP) = \sum_{i=1}^{n} T_{comp}(MP)_i + \sum_{i=1}^{2} (k_i \times T_S + (N/M) \times T_d)$. The array size, the number of processors, and the value of k_i will determine which algorithm has better performance.

4.3 Experimental results

To verify the performance analysis that presented in Section 4.2, we have implemented both algorithm on an IBM SP2 parallel machine. Both algorithms were written in the single program multiple data (SPMD) programming paradigm. To get the experimental results, a redistribution with a particular array size were executed 10 times by both algorithms on a 20-node configuration. The mean time of these 10 tests which were executed by an algorithm was used as the time of performing a redistribution with a particular array size of that algorithm. The single-precision array was used for the test.

A Experimental results for case 1

The time for both algorithms to perform two redistribution samples, BLOCK-CYCLIC(8) to BLOCK-CYCLIC(5) and BLOCK-CYCLIC(100) to BLOCK-

CYCLIC(3) with different array size N on a 20-node SP2 parallel machine was shown in Table 1. To perform these two samples, the two-stage multi-phase method uses BLOCK-CYCLIC(40) and BLOCK-CYCLIC(300) as the intermediate distribution for each redistribution. The time for the basic-cycle calculation to perform these two redistribution are $T(BCC) = T_{comp}(BCC) + 10T_S + 4 \times (N/20) \times T_d$ and $T(BCC) = T_{comp}(BCC) + 20T_S + 4 \times (N/20) \times T_d$ respectively. And the time for the two-stage multi-phase method to perform these two redistribution are $T(MP) = \sum_{i=1}^{2} T_{comp}(MP)_i + 13T_S + 4 \times (N/10) \times T_d$ and $T(MP) = \sum_{i=1}^{2} T_{comp}(MP)_i + 23T_S + 4 \times (N/10) \times T_d$ respectively. Note that each array element is 4-byte long.

From Table 1, for sample 1, we can see that the computation time of the basic-cycle calculation is less than that of the two-stage multi-phase method when the array size is less than a threshold. When the array size is over a threshold, the computation time of the basic-cycle calculation is greater than that of the two-stage multi-phase method. However, $(13T_S + 4 \times (N/10) \times T_d) - (10T_S + 4 \times (N/20) \times T_d)$ is still greater than $T_{comp}(BCC) - \sum_{i=1}^{2} T_{comp}(MP)_i$ when the array size is over the threshold. Therefore, the basic-cycle calculation outperforms the multi-phase method in this redistribution for all test samples. For sample 2, we can see that the computation time of the basic-cycle calculation is greater than that of the two-stage multi-phase method for test samples. However, $(23T_S + 4 \times (N/10) \times T_d) - (20T_S + 4 \times (N/20) \times T_d)$ is still greater than $T_{comp}(BCC) - \sum_{i=1}^{2} T_{comp}(MP)_i$ for test samples. Therefore, the basic-cycle calculation outperforms the two-stage multi-phase method in this redistribution for all test samples.

B Experimental results for case 2

The time for both algorithms to perform two redistribution samples, BLOCK-CYCLIC(25) to BLOCK-CYCLIC(20) and BLOCK-CYCLIC(300) to BLOCK-CYCLIC(200) was shown in Table 2. To perform these two samples, the two-stage multi-phase method uses BLOCK-CYCLIC(100) and BLOCK-CYCLIC(600) as the inter-mediate distribution for each redistribution. Table 2 gives us the same observations as described in Table 1.

C Experimental results for case 3

The time for both algorithms to perform a BLOCK-CYCLIC(60) to BLOCK-CYCLIC(3) redistribution was shown in Table 3. The time for the basic-cycle calculation to perform this redistribution is $T(BCC) = T_{comp}(BCC) + 20T_S + 4 \times (N/20) \times T_d$. For the multi-phase method, the time for the one-stage multi-phase method to perform this redistribution is $T(MP) = T_{comp}(MP) + 20T_S + 4 \times (N/20) \times T_d$. For the two-stage multi-phase method, it performs a BLOCK-CYCLIC(60) to BLOCK-CYCLIC(15) redistri-bution followed by a BLOCK-CYCLIC(15) to BLOCK-CYCLIC(3) redistribution. The time for the two-stage multi-phase method to perform this redistribution is $T(MP) = \sum_{i=1}^{2} T_{comp}(MP)_i + 9T_S + 4 \times (N/10) \times T_d$. From Table 3, we can see that the computation time and the communication time of the basic-cycle calculation are both greater than those of the one-stage multi-phase method for test samples. Therefore, the one-stage multi-phase method outperforms the basic-cycle calculation in this redistribution. However, the basic-cycle calculation out-performs the two-stage multi-phase method when the array size is over a threshold. The time for both algorithms to perform a BLOCK-CYCLIC(1000) to BLOCK-CYCLIC(50) redistribution was also shown in this table, and have similar conclusions as those described for the BLOCK-CYCLIC(60) to BLOCK-CYCLIC(3) redistribution.

5 Conclusions and future work

In this paper, we have presented a *basic-cycle calculation* technique to efficiently perform the general BLOCK-CYCLIC redistribution. To evaluate the performance of the basic-cycle calculation, we have implemented the basic-cycle calculation along with the multi-phase method. The experimental results show that the BCC outperforms the two-stage multi-phase method for all test samples in case1 and case 2. For case 3, BCC outperforms the two-stage multi-phase method when the array size is over a threshold. However, in this case, the one-stage multi-phase method outperforms the basic-cycle calculation for all test samples.

Considering of the redistribution in case 3, the communication cost of the basic-cycle calculation is the same as that of the one-stage multi-phase method. But the computation cost of the basic-cycle calculation is greater than that of the one-stage multi-phase method. In the future, we will study methods to reduce the computation time of the basic-cycle calculation for this case in the hope that the basic-cycle calculation can produce better performance than the one-stage multi-phase method.

References

[1] 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," *Journal of Parallel and Distributed Computing*, Vol. 26, pp. 72-84, 1995.

[2] J. J. Dongarra, R. van de Geijn, and D. W. Walker," A look at scalable dense linear algebra libraries," Technical Report ORNL/TM-12126 from Oak Ridge National Laboratory, Apr. 1992.

[3] 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 University, Dec. 1991.

[4] S. K. S. Gupta, S. D. Kaushik, C.-H. Huang, and P. Sadayappan, "On Compiling Array Expressions for Efficient Execution on Distributed-Memory Machines," Technical Report OSE-CISRC-4/94-TR19, Department of Computer and Information Science, The Ohio State University, Apr. 1994.

[5] S. K. S. Gupta, S. D. Kaushik, C.-H. Huang, and P. Sadayappan, "On the Generation of Efficient Data Communication for Distributed-Memory Machine," *Proc. of Intl. Computing Symposium,* pp. 504-513, 1992.

[6] High Performance Fortran Forum, "High Performance Fortran Language Specification(version 1.1)," Rice University, November 1994.

[7] S. D. Kaushik, C. H. Huang, J. Ramanujam, and P. Sadayappan, "Multi-Phase array redistribution: A Communication-Efficient Approach to Array Redistribution," Technical Report OSU-CISRC-9/94-52, Department of Computer and Information Science, The Ohio State University, 1994.

[8] E. T. Kalns and L. M. Ni, "DaReL: A portable data redistribution library for distributed-memory machines," in *Proceedings of the 1994 Scalable Parallel Libraries Conference II*, Oct. 1994.

[9] Edgar T. Kalns, and Lionel M. Ni, "Processor Mapping Technique Toward Efficient Data Redistribution, " *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 12 , December 1995.

[10] S. Ramaswamy and P. Banerjee, "Automatic generation of efficient array redistribution routines for distributed memory multicomputers, " Tech. Rep. CRHC-94-09, Center for Reliable and High Performance Computing, Computer Systems and Research Laboratory, Univ. of Illinois, 1994.

[11] Rajeev. Thakur, Alok. Choudhary, and J. Ramanujam, "Efficient Algorithms for Array Redistribution, " *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 6 , JUNE 1996.

[12] H. Zima and P. Brezany and B. Chapman and P. Mehrotra and A. Schwald," Vienna Fortran - A Language Specification Version 1.1," ICASE Interim Report 21, ICASE NASA Langley Research Center, Hampton, Virginia 23665, March, 1992.

[13] 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.

[14] C. Van Loan," Computational Frameworks for the Fast Fourier Transform," *SIAM*, 1992.

Table 1 : The execution time of both algorithms for case 1 redistribution.

Case 1		B-C(8) TO B-C(5)					B-C(100) TO B-C(3)				
Size ($\times 10^3$)		4	48	120	480	2400	6	36	120	480	2400
BCC	Comp.	0.161	1.521	3.683	15.263	72.259	0.514	1.443	3.835	18.39	68.345
	Total	1.194	3.158	5.315	25.74	152.38	1.585	4.387	8.159	27.931	127.84
2P	Comp.	0.203	1.435	3.346	9.811	68.258	0.429	1.252	2.25	16.332	50.308
	Total	1.46	3.903	6.778	33.608	194.21	9.226	12.109	15.723	33.595	151.94

Time unit : ms

Table 2 : The execution time of both algorithms for case 2 redistribution.

Case 2		B-C(25) TO B-C(20)					B-C(300) TO B-C(200)				
Szie ($\times 10^3$)		6	48	120	480	2400	12	60	120	480	2400
BCC	Comp.	0.193	1.106	2.55	14.321	53.201	0.59	1.473	2.41	14.665	49.8
	Total	0.976	2.968	5.891	24.515	102.85	1.095	3.261	4.773	23.843	86.582
2P	Comp.	0.207	0.939	2.377	13.843	50.874	0.341	1.316	2.065	14.502	48.369
	Total	1.112	3.687	6.376	33.52	183.19	1.503	3.064	7.736	33.718	182.74

Time unit : ms

Table 3 : The execution time of both algorithms for case 3 redistribution.

Case 3		B-C(60) TO B-C(3)					B-C(1000) TO B-C(50)				
Size($\times 10^3$)		24	120	384	960	2400	20	100	320	960	2400
BCC	Comp.	0.814	3.203	8.229	20.486	52.29	0.802	2.641	8.154	20.125	52.363
	Total	2.76	7.514	22.514	52.384	138.44	3.912	7.115	23.884	41.54	108.29
1P	Comp.	0.625	2.661	8.153	23.55	60.716	0.757	2.507	7.133	19.359	49.112
	Total	2.417	6.525	23.665	79.755	173.54	2.174	6.002	17.783	51.883	141.05
2P	Comp.	0.353	1.148	4.085	10.264	28.366	0.395	0.962	2.893	8.442	21.685
	Total	1.758	5.28	18.87	40.006	105.5	3.365	5.586	17.525	38.725	90.08

Time unit : ms