ARRAY REDISTRIBUTION ALGORITHMS FOR BIDIRECTIONAL PROCESSOR RINGS

Stavros Souravlas[1], Athanasios Margaris[2], and Manos Roumeliotis[3]
University of Macedonia, Applied Informatics Department,
156 Egnatia St, 54006, Thessaloniki, Greece
sourstav@uom.gr[1], amarg@uom.gr[2], manos@uom.gr[3]

Abstract

The Block Cyclic Array Redistribution problem occurs in many important applications in parallel computing. In this paper, we consider this problem on bidirectional processor rings. We present a message combining (MC) approach that splits any array redistribution problem in a series of broadcasts where all sources send messages of the same size, thus a balanced traffic load is achieved. Unlike existing array redistribution algorithms, the message combining scheme introduced in this work eliminates the need for data reorganization in the memory of the source and target processors. Moreover, the processing of the scheduled broadcasts is pipelined, thus the total cost of redistribution is reduced.

Keywords: Block cyclic array redistribution, classes, superclasses, pipeline, processor rings

1. Introduction-Related Work

The problem of array redistribution between several processors is very important, affecting the performance of parallel programs. Many complicated parallel computing applications are composed of several stages. As the program proceeds from one stage to another, it may require different and efficient redistribution of data between several processor sets. Such applications are the alternate direction implicit method and the multidimensional Fast Fourier Transform [Kaushik et. Al. (1994)].

The principal issues that should be taken into account for an efficient solution of the array redistribution problem are the *index computation overheads*, the *total communication overhead*, and the *data reorganization*. The index computation overheads consist of computing the target processor and the memory positions where each element will be located. The total communication overheads incur during data redistribution between parallel processors. Data reorganization is a very important issue for runtime array redistribution; redistributed array elements must be reorganized in every communication step so that, the data blocks sent and received are contiguous in the data array [N. S. Sundar et. Al. (2001)].

Many methods for array redistribution can be found in the literature. In [C.-H. Hsu et. Al. (2001), Huang and Chu (2006)], processor mapping techniques for dynamic data redistribution are described. Thakur et al. [Thakur et. Al. (1996)] also provided algorithms for array redistribution. Their work is divided into two cases: the general case of *Cyclic(x)* to *Cyclic(y)* redistribution, where there is no relation between *x* and *y*, and a special case where *x* is a multiple of *y* or *y* is a multiple of *x*.

Walker and Otto [D.W Walker, S.W Otto (1996)] worked on the problem of data redistribution from *cyclic(x)* to *cyclic(Kx)* on a grid of *P* processors. They provided synchronized and unsynchronized schemes that were free of conflicts. Desprez et al [Desprez et. Al. (1998)] focused their effort on solving the general redistribution problem, moving from *cyclic (r)* on a *P*-processor grid, to *cyclic (s)* on a *Q*-processor grid. The main idea behind their algorithm was to create homogeneous communication patterns which they called *classes*. Processor pairs in a certain class, exchanged messages of the same size.

In this paper, we focus on scheduling block cyclic array redistribution on a processor ring. We present a block-cyclic array redistribution strategy where target blocks are formed by exchanging messages between specified sets of neighbouring source processors, with no need for data reorganization. This strategy allows for equal sized messages to be exchanged at any time in any direction of the network, thus, messages can be pipelined. This makes the scheme particularly efficient in networks with bounded node degree like rings. The rest of the paper is organized as follows: In Section 2 we briefly present the preliminaries for the block cyclic redistribution problem. In Section 3, we present our message combining algorithm (named MC). In Section 4, we show how MC applies on a bidirectional processor ring and perform cost analysis. Section 5 concludes the paper.

2. Preliminaries

Consider an array of *M* elements that is initially distributed CYCLIC (*r*) on *P* processors, where *r* is the block size. The index of blocks is *l*, where $l \in [0..M/Pr)$ and the local position of a data element inside a block is *x*. Our aim is to provide a target distribution CYCLIC (*s*) to a set of *Q* processors. The block size of the target distribution is *s*, the index of blocks is given by *m*, where *m* $[0..M/Qs)$ and the local position of a data element inside a block is given by *y*. The distribution from CYCLIC(*r*) on *P* processors to CYCLIC (*s*) on *Q* processors is described by the redistribution equation:

$$(lP+p)r+w=(mQ+q)s+z \qquad \textbf{(1)}$$

where (*p,q*) is a set of communicating processors.

The number of quadruples (*l,m,x,y*) that satisfy (1) is the number of elements redistributed from *p* to *q* or the communication cost $C_{p,q}$. Block cyclic redistribution is periodical and it repeats for every *L*=LCM (*Pr,Qs*) data elements, where *L* is the least common multiplier (LCM) of (*Pr,Qs*). According to Desprez et al. [Desprez et. Al. (1998)], a processor pair belongs to a class *b(k)* if the following equality is satisfied:

$$(pr\text{-}qs) \bmod g = k \qquad \textbf{(2)}$$

where g is the greatest common divisor of Pr,Qs, that is g=gcd(Pr,Qs). The number of classes that exists in a redistribution problem is g.

We now extend the idea of classes and introduce the *superclasses*. The use of superclasses is a key idea for implementing the message combining scheme that will be introduced in the next paragraph. Initially, let us define the function *dist* (k_1,k_2) that computes the distance between two classes k_1 and k_2 such that:

$$\text{dist}\,(k_1,k_2) = \begin{cases} k_2\text{-}k_1, \text{ if } k_2 \geq k_1 \\ g\text{-}k_1+k_2, \text{ otherwise} \end{cases} \qquad \textbf{(3)}$$

DEFINITION 1: A group of classes V=$k_1, k_2 ... k_{n-1}, k_n$ that differ by r mod g, that is: dist (k_1,k_2)= dist (k_2,k_3)= …..= dist (k_{n-1},k_n)= r mod g, is called *superclass*.

In the following paragraph we will show how the communication between processor pairs from a superclass forms target blocks redistributed to receiving nodes.

3. The Message Combining Scheme

In this Section we present the message combining (MC) scheme based on the superclasses discussed in the previous Section. Our approach is based on the idea that each block for any target processor q is formed by the mappings described in the lemmas below:

Lemma 1
The first bytes of every target block in an array redistribution problem, derive from the communication of processor pairs p,q that belong to a specific set of r in total processor classes $V = k_0, k_1, k_2 ... k_{r-1}$ that satisfy:

$$-x \bmod g = k,\ x \in [0..r\text{-}1] \qquad \textbf{(4)}$$

PROOF: Equation (1) is rewritten as $mQs - lPr = pr\text{-}qs + (y\text{-}x)$. We know that g=gcd (Pr,Qs), making $mQs\text{-}lPr$ a multiple of g. This means that there is an integer λ, such that: $mQs\text{-}lPr=\lambda g$. If we also set $\xi = x\text{-}y$, Equation (1) is rewritten as:

$$-\lambda g - \xi = pr\text{-}qs \qquad \textbf{(5)}$$

If we divide both parts of Equation (5) with g, we get: $(\lambda g\text{-}\xi)$ mod g = $(pr\text{-}qs)$ mod g $\Rightarrow$ $(\lambda g$ mod $g)$-$(\xi$ mod $g)$ = $(pr\text{-}qs)$ mod g $\Rightarrow$ $-\xi$ mod g = $(pr\text{-}qs)$ mod g. Since $\xi=x\text{-}y$, it is obvious that $-\xi=y\text{-}x$. Therefore, we obtain the equation:

$$(y\text{-}x) \ mod \ g = (pr\text{-}qs) \ mod \ g \qquad \textbf{(6)}$$

Considering that y is the local position of a data element in the target distribution block, if we set y=0 we get the proof of the lemma.

DEFINITION 2: Every class k that satisfies equation (4) will be referred to as *generator class*. The source processors in each generator class are called *generator processors* or *generator nodes*.

Obviously, each target block has its own generator node. The number of generator classes is r since $x \in$ [0..r-1]. We define the *vol* function that computes the elements a generator node p contributes to a target block m that will be redistributed to a target node q. In other words, the *vol* function returns the number of quadruples (l,m,x,y)= (l,m,x,0) that satisfy the redistribution equation (1).

$$vol\ (p,q)= \{(l,m,x,y)=(l,m,x,0) \text{: } mQs\text{-}\ lPr=pr\text{-}qs+(x\text{-}y)\ \} \qquad \textbf{(7)}$$

Lemma 2

Two neighboring processors p_γ, p_δ send data elements on the same target distribution block of the same receiving processor q if processor pairs p_γ, q and p_δ,q belong the same superclass.

PROOF: We will show that if $p_\gamma ,q \in$ k_1 and $p_\delta,q \in$ k_2, the distance between k_1 and k_2 equals r mod g. Assume that processors p_γ and p_δ send data elements on the same block of receiving processor q and $p_\gamma,q \in$ k_1, $p_\delta,q \in$ k_2. For processor pair p_γ,q equation (6) is rewritten as: $(y\text{-}x)$ mod g= $(p_\gamma r\text{-}qs)$ mod g. Similarly, for p_δ,q we have $(y\text{-}x)$ mod g =(p_δ $r\text{-}qs)$ mod g. With no loss of generality, we assume that the indices of two neighboring source nodes p_γ, p_δ differ by 1 (the proof is similar for any other integer value). Thus, $(y\text{-}x)$ mod g = $(p_\delta r\text{-}qs)$ mod g becomes: $(y\text{-}x)$ mod g= $(p_\gamma r$+ $r\text{-}qs)$ mod g. We summarize the set of equations for the two processor pairs:

$$\text{(y-x) mod g} = \begin{cases} (p_\gamma r\text{-}qs)\ mod\ g = k_1, \text{ for } (p_\gamma,q) \\ (p_\gamma r+ r\text{-}qs)\ mod\ g= k_2, \text{ for } (p_\delta,q) \end{cases} \qquad \textbf{(8)}$$

To prove the lemma, we simply compute dist (k_1 ,k_2) for the following cases:
(i) $p_\gamma r$+ $r\text{-}qs$ <g and r<g, (ii) $p_\gamma r$+ $r\text{-}qs$ <g and r≥g, and (iii) $p_\gamma r$+ $r\text{-}qs$ >g.

Based on the lemmas above, we formally describe our sending algorithm for array redistribution. There are two subphases: (i) Target blocks formulation phase, and (ii) Block redistribution phase. The target blocks formulation phase involves interprocessor communication between all source nodes of each superclass. This results in blocks of size *s* being stored in the memory of the generator nodes of each superclass. The block redistribution phase includes all the communications required to transfer the blocks from the generator nodes to the target processors. One more function needs to be defined beforehand; the *stepping* function *S* that will be used to compute the next class-member of a superclass as follows:

$$S(k) = \begin{cases} r \bmod g, & \text{if } (r \bmod g) + k < g \\ (r \bmod g)-g, & \text{if } (r \bmod g) + k \geq g \end{cases} \qquad \textbf{(9)}$$

Phase 1: Forming target blocks in the generator nodes' memory

STEP 1: Start from a generator class k_0. Source nodes p, $p\in$ $[0..P-1]$) are scheduled to send s'=vol (p,q) to destination processors q ($q \in [0..Q-1]$) to form a target block.
STEP 2: If $s'=s$ the target block has been formed. We pick another generator class and move to step 1. If $s'<s$, there remain to be added a number of elements from other nodes. Move to step 3.
STEP 3: Use the stepping function *S* to get the next class-member of the superclass k_1:$k_1=k_0+S(k_0)$.There are two cases:
Case 3.1: k_1 is also a generator class: In this case, all sources in k_1 contain both the last data elements ($s-s'$ in total) of block *m* **and** the initial elements of another target block indexed m', for which they are generator nodes. The target block *m* is formed in the generator nodes' memory by passing a message from source nodes of k_1 to source nodes of k_0 (generator nodes of *m*).
Case 3.2: k_1 is not a generator class: In this case, all $C_{p,q}$ elements are appended to *m*. A message of size $C_{p,q}$ is passed from all source nodes of k_1 to all source nodes of k_0. There are two cases:
Case 3.2.1: If $s'+C_{p,q}<s$: We set $s'=s'+C_{p,q}$ and we use the stepping function *S* to obtain the class that contains some or all the remaining elements: $k_2=k_1+S(k_1)$. We return to case 3.1.
Case 3.2.2: $s'+C_{p,q}=s$: Block *m* has been formed. Go to STEP 1.

Phase 2: Redistribution of data blocks

STEP 1: Select a generation class. Send the blocks of size *s* from the generator nodes to the proper targets.

STEP 2: Repeat STEP 1 until all generation classes have been selected and all messages have been sent.

4. Applying the MC Scheme in Bidirectional Processor Rings

4.1 The Communication Model

In this paragraph, we show how the message combining MC is utilized on bidirectional processor rings. Initially, let us consider a ring T_n where n is the number of nodes. The w-axis and z-axis evenly partition the ring into four quadrants. Each processor p_i on the ring has a maximum of five neighbors. If we denote N_i the set of neighbors for processor p_i, we have: $N_i= \{p_i^{left}, p_i^{right}, p_i^{w}, p_i^{z}, p_i^{wz}\}$ where p_i^{w}, p_i^{z} are the nodes symmetric to p_i with respect to the w-axis and z-axis, p_i^{left}, p_i^{right} are the nodes that lie next to p_i in both directions of the ring, and p_i^{wz} is the node symmetric to p_i with respect to the origin [Tseng, Gupta (1996)].

In this paper, we assume a communication model with full-duplex channels, where each node can simultaneously send and receive messages, while it can communicate with all its neighboring nodes. Assuming that p_i and p_j are two distinct nodes, we can define the necessary communications on the ring for the formulation of target blocks in the generator nodes' memory R^+, and for the redistribution of data blocks, R^-. Each of the two phases is composed of a number of directed paths; at any time, a message of R^+ is sent out in one direction from the origin, while a message of R^- is sent out in the opposite direction.

4.2 Forming the Target Blocks

Messages of R^+ do not use the links that connect symmetric nodes with respect to z-axis or w-axis. As shown in the previous paragraph, the idea behind the formulation of target blocks in the generator nodes' memory, is to provide a set of communications on a set of β neighboring nodes, where β is the number of classes in each superclass. Figure 1 shows a redistribution where there are four superclasses V_0-V_3, each having β =4 classes.

The generator nodes of the superblocks are P_j, P_i, P_i^z, and P_j^z respectively: $V_0= \{P_i^{wz} \rightarrow P_j^z \rightarrow P_i^z \rightarrow P_i \rightarrow P_j\}$, $V_1= \{P_j^{wz} \rightarrow P_i^{wz} \rightarrow P_j^z \rightarrow P_i^z \rightarrow P_i\}$, $V_2= \{P_i^{w} \rightarrow P_j^{wz} \rightarrow P_i^{wz} \rightarrow P_j^z \rightarrow P_i^z\}$, $V_2= \{P_j^{w} \rightarrow P_i^{w} \rightarrow P_j^{wz} \rightarrow P_i^{wz} \rightarrow P_j^z\}$. For the ease of showing how communications are performed on the ring, assume a processor ring as shown in Figure 2 and that R^+ starts at time $t=1$. At time $t=1$, the first member of each chain forwards the proper data to its right neighboring node. Through these communications, the last bytes of each target block will be redistributed, from source nodes P_i^{wz}, P_j^{wz}, P_i^{w} and P_j^{w} to generator nodes P_j, P_i, P_i^z, and P_j^z respectively. In other words, link $P_i^{wz} \rightarrow P_j^z$ handles communication performed on chain V_0, $P_j^{wz} \rightarrow P_i^{wz}$ is busy with communication performed on V_1, and links $P_i^w \rightarrow P_j^{wz}$ and $P_j^w \rightarrow P_i^w$ handle the communications on superclasses V_2 and V_3 respectively.

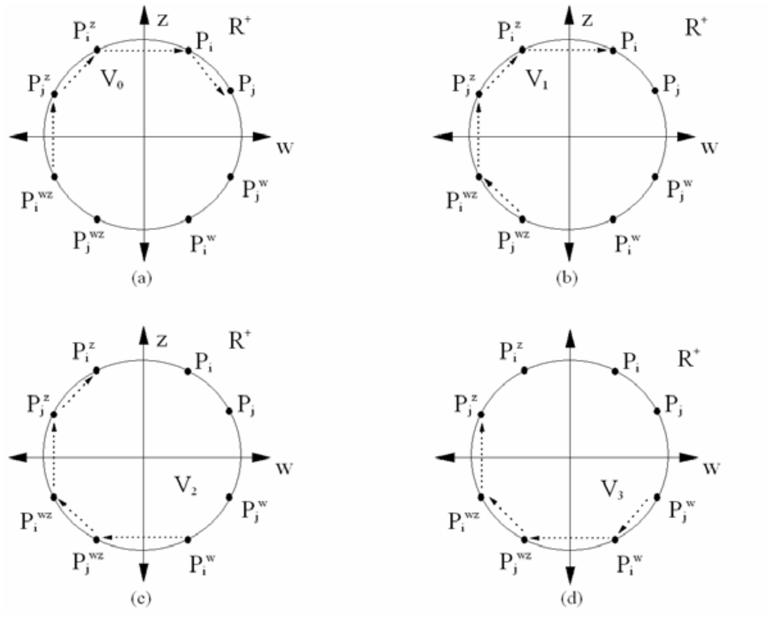

Figure 1: Forming target blocks in the generator nodes' memory (R^+)

At time $t=2$, communication on V_0 is handled by link $P_j^z \rightarrow P_i^z$; this means that P_j^z creates a new message composed of: (1) the elements from its "left" neighbor P_i^{wz}, and (2) the elements that P_j^z forwards to the generator node of V_0, P_j. Similarly, $P_i^{wz} \rightarrow P_j^z$ that previously handled V_0, now handles V_1, and links $P_j^{wz} \rightarrow P_i^{wz}$ *and* $P_i^w \rightarrow P_j^{wz}$, are busy with communications on V_2 and V_3 respectively. Continuing in this manner, at time $t=4$, complete target blocks m_0, m_1, m_2, and m_3 have been temporarily stored in memory of the generator nodes P_j, P_i, P_i^z, P_j^z respectively.

Assuming that a superclass includes β classes, communication involves message passing through β-1 links. According to the pattern described for the T_n ring, communication is implemented in a pipeline fashion, therefore the cost of forming a block in the memory of a generator node equals the cost of communication between all processor pairs p,q in one superclass C_V:

$$C_v = \sum_{i=1}^{\beta-1} C_{p,q} \tag{10}$$

where $C_{p,q}$ is the data volume that each source in the superclass p contributes to the formulation of a data block to a target node q.

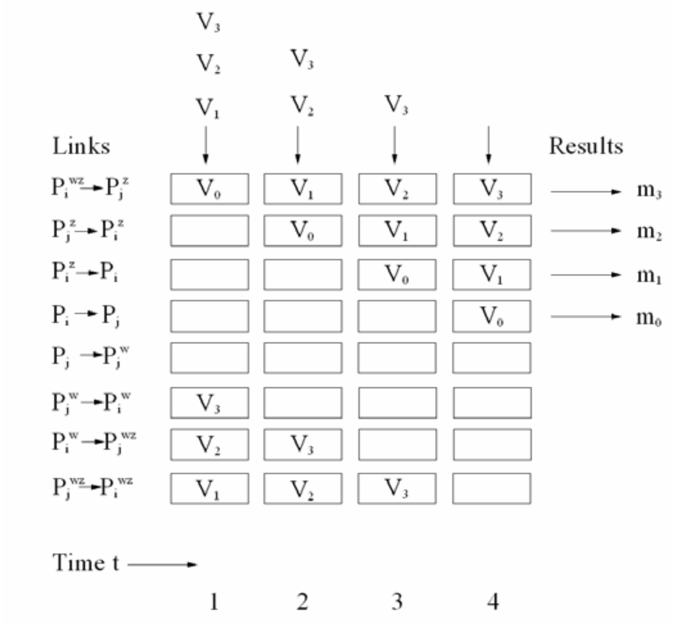

Figure 2: Communication on neighboring nodes

The total cost of R$^+$ is the sum of all communications that incur in a total of *r* (see Lemma 1 and Definition 2) superclasses:

$$C_{R^+} = \sum_{i=1}^{r} C_{V_i} \qquad \textbf{(11)}$$

4.3 Redistributing the Target Blocks

Messages of R^- are also completed in a set of directed paths; after a target block has been formed, it is sent out in the opposite direction than the messages of phase R^+. Moreover, messages of R^- do use the links that connect symmetric nodes with respect to *z*-axis or *w*-axis, in both directions. We have predefined dedicated communication paths, so that the nodes exchange data blocks in accordance to the quadrant they lie. These paths are shown in Figure 3.

When an entire target block originates from a generator node *p* to a receiving node *q*, communication is performed with respect to the quadrant where *p,q* are located as the following cases indicate:

Case 1: p,q are symmetric with respect to *z* or *w* axis (See Figure 3(a))

1.1 $p=p_i$, $q=p_i^{wz}$ or vice versa: *p* sends to *q* through link $p_i \to p_i^{wz}$ or $p_i^{wz} \to p_i$

1.2 $p=p_i$,$q=p_i^{w}$ or vice versa: *p* sends to *q* through link $p_i \to p_i^{w}$ or $pi^{w} \to p_i$

1.3 $p=p_{i,}$, $q=p_i^{z}$ or vice versa: *p* sends to *q* through link $p_i \to p_i^{z}$ or $p_i^{z} \to p_i$

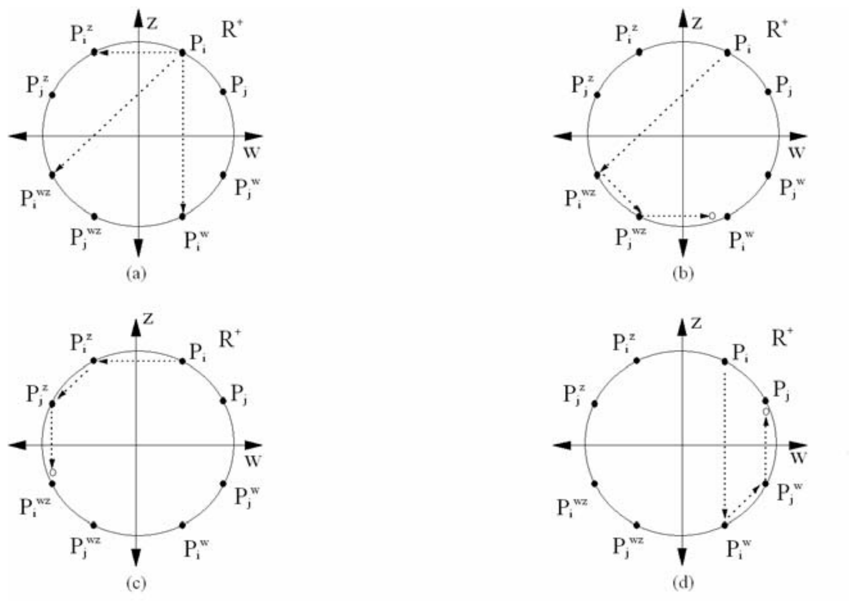

Figure 3: Predefined directed paths for R$^-$

Case 2:$p=p_i$ and q is located between p_j^{wz} and p_i^{w} (see Figure 3(b)): p transfers the block to its symmetric node p_i^{wz}. The block is then distributed to the target node through processors $p_i^{wz} \rightarrow p_j^{wz} \rightarrow p_j^{wz^{right}} \rightarrow \ldots\ldots \rightarrow p_i^{w^{left}}$. The small cycle adjacent to the arrow from p_j^{wz} to p_i^{w} indicates that all processors up to p_i^{w} are included in the path, but not p_i^{w}.

Case 3: $p=p_i$ and q is located between $=pi^z$ and p_i^{wz} (see Figure 3(c)): p transfers the block to its symmetric node p_i^{z}. The block is then redistributed to the target node via processors $p_i^{z} \rightarrow p_j^{z} \rightarrow p_i^{wz}$.

Case 4:$p=p$ and q is located between p_i^{w} and p_j (see Figure 3(d)): p transfers the block to its symmetric node p_i^{w}. The block is redistributed to the target node through processors $p_i^{w} \rightarrow p_j^{w} \rightarrow \ldots\ldots \rightarrow p_j$.

Since messages cannot be simultaneously transmitted on a single channel in the same direction, redistribution is implemented in a set of communications between couples of quadrants of the ring. The cost of redistributing the target blocks in any array redistribution problem is at most $3sa\ (n/4)^2$, where a is the startup cost. For proof, consider the communication between two quadrants of the ring. There are at most $n/4$ messages to be transmitted from the generator (source) nodes to the receiving nodes. The startup cost of each message is *a*. Since message transmission works in a pipeline fashion, the total delay of each step equals the maximum number of links that messages go through at each step, that is, $n/4$ links. This gives a total delay of $s(n/4)^2$ for the communication between two quadrants. Since there exist 6 couples in total, we need 3 communication steps to complete the redistribution of target blocks. Thus, the total delay is at most $3sa\,(n/4)^2$.

5. Conclusions

In this work, we have proposed a message combining scheme for redistribution of arrays for bidirectional rings. The proposed strategy has an important feature; it assures that neighboring processors can communicate in such an order that results in a data block formed in the memory of the relay processors while the data elements are stored in correct order. Hence, each relay only needs to transfer the block to the target that simply stores the block to the proper memory location.

6. References

Desprez F.,.Dongarra J., Petitet A., Randriamaro C., Robert Y. (1998), "Scheduling Block-Cyclic Array Redistribution", *IEEE Transactions on Parallel and Distributed Systems*, Vol.9, NO.2, February 1998, pp.192-205.

Hsu C.H., Chung Y.C., Yang D.L, and Dow C.R. (2001), "A Generalized Processor Mapping Technique for Array Redistribution", *IEEE Transactions on Parallel and Distributed Systems*, Vol.12, NO.7, pp.743-757.

Huang J.W, and Chu C.P (2006), "An Efficient Communication Scheduling Method for the Processor Mapping Technique Applied Data Redistribution", *TheJournal of Supercomputing*, 37, pp.297-318.

Kaushik S.D., Huang C.H., Johnson R.W.and Sadayappan P.(1994), "*An approach to Communication-Efficient Data Redistribution*", 8th ACM International Conference on Supercomputing, July 1994, Manchester, England.

Sundar N.S., Jayasimha D.N, Dhabaleswar K.P., and Sadayappan P. (2001), "Hybrid Algorithms for Complete Exchange in 2D Meshes", *IEEE Transactions on Parallel and Distributed Systems*, pp.1201-1218.

Thakur R, Choudhary A., Ramanujam J. (2001), "Efficient Algorithms For Array Redistribution", *IEEE Transactions on Parallel and Distributed Systems*, Vol.7, NO. 6, June 1996, pp.587-594.

Tseng Y.C., and Gupta S.K.S. (1996), "All-to-All Personalized Communication in a Wormhole-Routed Torus", *IEEE Transactions on Parallel and DistributedSystems*, Vol.7, NO. 5, May 1996, pp.498-505.

Walker D.W, and Otto S.W (1996), "Redistribution of Block-Cyclic Data Distributions Using MPI", *Concurrency: Practice and Experience*, vol.8, no.9, pp.707-728.