
ELSEVIER

Parallel Computing 26 (2000) 1325–1343

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Contention-free communication scheduling for array redistribution

Minyi Guo [a,*], Ikuo Nakata [b], Yoshiyuki Yamashita [a]

[a] *Institute of Information Sciences and Electronics, University of Tsukuba, 305-0006 Tsukuba, Japan*
[b] *University of Library and Information Science, 305-8550 Tsukuba, Japan*

Received 12 March 1999; received in revised form 22 October 1999; accepted 28 January 2000

Abstract

Array redistribution is required often in programs on distributed memory parallel computers. It is essential to use efficient algorithms for redistribution, otherwise the performance of the programs may degrade considerably. The redistribution overheads consist of two parts: index computation and interprocessor communication. If there is no communication scheduling in a redistribution algorithm, the communication contention may occur, which increases the communication waiting time. In order to solve this problem, in this paper, we propose a technique to schedule the communication so that it becomes contention-free. Our approach initially generates a communication table to represent the communication relations among sending nodes and receiving nodes. According to the communication table, we then generate another table named communication scheduling table. Each column of communication scheduling table is a permutation of receiving node numbers in each communication step. Thus the communications in our redistribution algorithm are contention-free. Our approach can deal with multi-dimensional "shape changing redistribution". © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Parallelizing compilers; HPF; Array redistribution; Communication scheduling; Distributed memory machines

* Corresponding author. Present address: NEC Software Ltd., 136-8608 Tokyo, Japan. Tel.: +81-3-5569-3207.

E-mail address: guo@mxb.nes.nec.co.jp (M. Guo).

PII: S0167-8191(00)00027-2

1. Introduction

Array redistribution problem has recently received considerable attention. This interest is motivated largely by the HPF [5] programming style, in which scientific applications are decomposed into phases. At each phase, there is an optimal distribution of the arrays onto the processor grid. Because the optimal distribution changes from phase to phase, array redistribution turns out to be a critical operation.

Basically, the redistribution algorithms consist of two parts: index computation and interprocessor communication. The index computation overheads are incurred when each processor computes indices of array elements that are to be communicated with the other processors, as well as the destination processors of such array elements. The communication overheads are incurred when the processors exchange array elements. These include software start-up overheads for invocation of the send/receive system calls, transmission costs for sending data over the interconnection network, and overheads due to the node contention. Our efforts attempted to reduce the index computation overhead in [2,3]. However, without a proper communication scheduling the redistribution overhead can be enormous. Especially, node contention can significantly influence the communication performance. In this paper, we focus on reducing the actual communication cost of redistribution. A communication scheduling method is proposed to avoid node contention.

1.1. Communication contention problem

Figs. 1(a) and (b) are the framework of the redistribution algorithm with the naive communication approach. Fig. 1(a) shows the part of the algorithm executed on the source processors and Fig. 1(b) is the part executed by the destination processors, where *proc* is the number of processors. Fig. 2(a) shows the sequence of events that can occur during a redistribution involving four source and four destination processors using the above redistribution algorithm [2,3]. We see that each destination

```
Sending part
   array index computation part
   of the redistribution;
/* send data to all other processors: */
   for p = 0, proc-1
      pack data into buffer[p];
      send(buffer[p],p);
   endfor
```

(a)

```
Receiving part
/* receive data from all other
   processors and unpack to
   target array: */
 for p=0, proc-1
    receive(buffer, p);
    unpack data into target
    local array from buffer;
 endfor
```

(b)

Fig. 1. Redistribution algorithm with the naive communication approach.

processor receives all of its messages simultaneously; this may lead to communication contention.

The communication contention problem can be described as follows: For a set of processors, since the receiving processor typically can receive messages from only one processor at once, if there are more than two of sending processors they may have to wait for other processors to complete their communication, in this case we say that the communication (or node) contention occurred.

We show that the communication contention has a deteriorating effect on the total time required for communication. Fig. 3 shows the impact of the communication contention on CP-PACS [11] Pilot3, a 64-processor MIMD distributed memory parallel machine developed at the University of Tsukuba. In these experiments, processor P_0 is the receiving node, and processors P_i $(1 \leqslant i < 64)$ are the sending nodes. In each step, each sending node sends an equal amount of data (1K or 4 KB) to P_0 simultaneously. We record the time (in ms) taken for P_0 to complete receiving all incoming data and for other sending nodes to complete sending data.

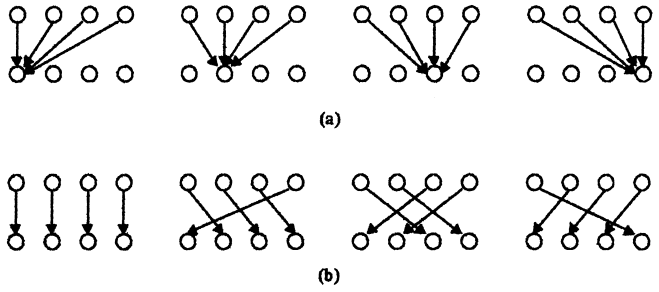

Fig. 2. Communication contention for array redistribution.

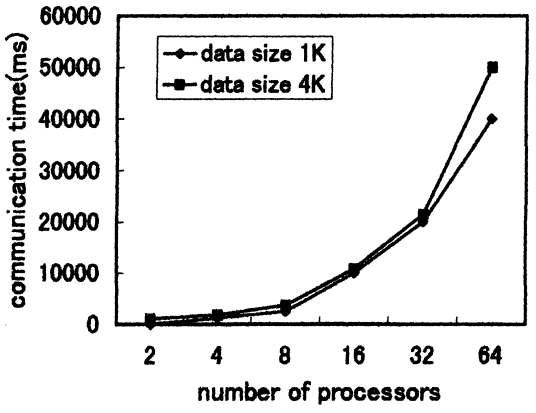

Fig. 3. Experiment about communication contention on CP-PACS Pilot3 (P_0 is the receiving node; P_i $(1 \leqslant i < 64)$ are the sending nodes).

The results reveal that when the number of messages sent to the same node (at the same time) increases, the communication time increases. Thus it is inefficient to allow more than one node to send a message to the same processor simultaneously.

These observations suggest that node contention will result in overall performance degradation. Avoiding node contention should therefore be considered as an important factor when we conduct the communication scheduling.

1.2. Avoidance of communication contention

Using the communication scheduling we can avoid the communication contention in redistribution operations. For instance, we can use simple schemes to get rid of the contention shown in Fig. 2(a); the effect of such schemes is illustrated in Fig. 2(b). One of the schemes we use is to have each source processor starting with a different destination processor index, the modified algorithm implementing this scheme is shown in Fig. 4, where *myid* is the executing processor number.

The communication scheme used in the above redistribution algorithm is called all-to-all communication (each sending processor has to communicate with all receiving processors in exactly *proc* communication steps). In the more complicated situation, however, some communications in redistribution algorithms are all-to-many communications, i.e., each sending processor should communicate with the subset of all receiving processors. Fox example, consider the redistribution from cyclic(6) to cyclic(2) for the total number of processors 4. In this case, each processor has only to send messages to three processors. The above scheduling method cannot simply be applicable for this case. Hence, we focus on the contention-free communication scheduling in the case of all-to-many communications in this paper. In other words, we mainly consider solutions for redistribution among block-cyclic distributed arrays. In each array's dimension, the redistribution is processed from a cyclic(b) distribution on a P-processor grid to a cyclic(b') distribution on a P'-processor grid and $b = \beta * b'$, or $b' = \beta * b$, where β is a positive integer.

The rest of the paper is organized as follows. Section 2 discusses important related work in this area. Section 3 gives an overview of the redistribution algorithm based

```
Sending part
   array index computation part
   of the redistribution;
/* send data to all other processors: */
   for p = 0, proc-1
      pp = (myid+p) mod proc;
      pack data into buffer[pp];
      send(buffer[pp],pp);
   endfor
```

(a)

```
Receiving part
/* receive data from all other
   processors and unpack to
   target array: */
 for p=0, proc-1
    pp = (myid-p) mod proc;
    receive(buffer, pp);
    unpack data into target
    local array from buffer;
 endfor
```

(b)

Fig. 4. Redistribution algorithm with the all-to-all communication scheduling.

on the Local Data Descriptor (LDD) described in [2–4]. In Section 4, we give our communication scheduling approach for one-dimensional redistribution and multi-dimensional redistribution, and show how to get the receiving communication scheduling table. Section 5 gives some experimental results for our approaches compared with the redistribution algorithms without communication scheduling. Finally, we conclude this paper and discuss possible future extensions to our work in Section 6.

2. Related work

Many researches have mainly concentrated on the efficient index computation for generating the communication messages to be exchanged by the processors involved in the redistribution [2,3,12,15,16]. However, the question of how to efficiently schedule the messages has received little attention. The following are some researches concerned with the communication optimization in redistribution.

Lim et al. [9,10] developed the algorithms that redistribute an array from one block-cyclic scheme to another, where the source and target schemes have the special relation. Their framework is based on a generalized circulant matrix formalism. Through the transformation of the rows/columns of the matrix, data communication is performed in a conflict-free manner using direct, indirect, and hybrid algorithms. In the direct algorithm, a data block is transferred directly to its destination processor. In the indirect algorithm, data blocks are moved from source to destination processors through intermediate relay processors. The relay processors combine data blocks with the same destination. The hybrid algorithm is a combination of the direct and indirect algorithms.

However, in their approaches, the scheduling algorithm for multi-dimensional redistribution cannot reuse the one for one-dimensional redistribution. They use different approaches to process the one-dimensional and multi-dimensional problems.

Kalns and Ni [8] presented a technique for mapping data to processors in order to minimize the total amount of data that must be communicated during redistribution but they do not further specify the general case of communication scheduling. A multi-phase redistribution approach is suggested in [6,7]. They use the tensor product representation of data distributions and the network contention model by expressing the communication as a sequence of permutations, each of which can be executed in a fixed number of contention-free steps. They developed a multi-phase strategy which performs the redistribution as a sequence of redistributions such that the total cost of the sequence is less than that of direct redistribution.

Desprez et al. [1] proposed an algorithm for scheduling of those messages – how to organize the message exchanges into ''structured'' communication steps that minimize contention. They built a scheduling for moving from a cyclic(r) distribution on a P-processor grid to a cyclic(s) distribution on a Q-processor grid for one-dimensional redistribution, where the values of P, Q, r, and s are arbitrary. They considered the size of the communication messages as a term of scheduling.

However, they did not give the description of the approach how to get the information about the sources and destinations of the communication and the message sizes.

Ranka et al. [13] developed algorithms to perform message routing for all-to-many personalized communication. They also decomposed all-to-many communication into a set of disjoint partial permutations by using communication matrix. An approximate analysis showed that with n processors, and assuming that every processor sends and receives d messages to random destinations their algorithm can perform the scheduling in $O(d * n * \ln d)$ time, on average, and can use an expected number of $O(d + \log d)$ partial permutations to carry out the communication. In [14], they extended this work to the case where the message sizes which each processor sends (receives) are with high variance. They showed that such many-to-many personalized communication with non-uniform messages can be performed using two stages of all-to-all communication with uniform messages.

Our aim in this paper is to extend the concepts and results presented in our previous work [2–4], in order to solve the communication scheduling problem in array redistribution. Our method completely deals with all the processes of communication scheduling – determines the sources, destinations, and message sizes of the communications, schedules the communications into steps. To avoid resource contentions, each participating processor neither sends nor receives more than one message at each step. We consider that the communication of array redistribution is a special case of general personalized communication. By efficiently using its peculiarity, the overhead of communication scheduling of array redistribution becomes less than the general all-to-many communication scheduling (see Section 4.2). The multi-dimensional algorithms are simply obtained by reusing the one-dimensional algorithm.

3. Overview of redistribution based on Local Data Descriptor

A redistribution problem can be definitely represented as following. A redistribution $\mathscr{R}$ is the set of routines that change the distribution schemes such that, given an multi-dimensional array A on a set of source processors $\mathscr{P}_s$ with distribution scheme $\mathscr{D}_s$, transfer all the elements of the array to a set of target processors $\mathscr{P}_t$ with a target distribution scheme $\mathscr{D}_t$. In a general case, $\mathscr{D}_s$ and $\mathscr{D}_t$ can specify arbitrary regular data distributions along each dimension of the array. Therefore, the redistribution routines, which are determined by two pairs $(\mathscr{P}_s, \mathscr{D}_s)$ and $(\mathscr{P}_t, \mathscr{D}_t)$, need to figure out exactly what data need to be sent (received) by each source (target) processor.

In [2–4], we proposed an approach to generate the redistribution algorithm, which is based on the representation called LDD. An LDD expresses the set of the array elements partitioned onto a local distributed memory. We also defined some operations on LDDs and referred to the fact that the data being redistributed between two processors are indicated by the intersection of LDDs of the processors. Because we will use some concepts of LDD in the following sections, in this section, we give

an overview of the redistribution algorithm based on LDD described in our earlier papers [2–4]. The further details can be found in them.

Definition of one-dimensional LDD. A 4-tuple $d = (o, b, s, n)$ is called an LDD which describes a set of the global array index for a particular processor. Intuitively, d represents a finite set of equally spaced, equally sized blocks of elements, where o is the starting index of the global array elements distributed onto the processor; b the length of the block; s the stride between two consecutive blocks; and n is the number of blocks distributed onto the processor.

Consider a one-dimensional array A of size G. Using the notion of LDD, it is possible to represent the set of elements of A owned by a processor under any regular distribution. An LDD corresponds to a set of the global array index defined as follows:

$$S[d] = \{i \mid o + s * u \leqslant i < o + b + s * u,\ 0 \leqslant u < n\}.$$

Intersection of LDDs. Let $d_1 = (o_1, b_1, s_1, n_1)$ and $d_2 = (o_2, b_2, s_2, n_2)$ be two LDDs, and their corresponding array index sets are $S[d_1]$ and $S[d_2]$ (namely LDD set), respectively. The intersection of $S[d_1]$ and $S[d_2]$ is as follows:

$$\begin{aligned} S[d_1] \cap S[d_2] = \{i \mid\ & \max(o_1 + s_1 * u_1, o_2 + s_2 * u_2) \\ & \leqslant i < \min((o_1 + s_1 * u_1) + b_1, (o_2 + s_2 * u_2) + b_2),\ 0 \leqslant u_1 < n_1, \\ & 0 \leqslant u_2 < n_2\}. \end{aligned}$$

Lemma 1. *Let* $d_1 = (o_1, b_1, s_1, n_1)$ *and* $d_2 = (o_2, b_2, s_2, n_2)$, $S[d_1] \cap S[d_2] = \emptyset \Longleftrightarrow \forall u_1, u_2 (0 \leqslant u_1 < n_1 \wedge 0 \leqslant u_2 < n_2)$, $\max(o_1 + s_1 * u_1, o_2 + s_2 * u_2) \geqslant \min(o_1 + b_1 + s_1 * u_1, o_2 + b_2 + s_2 * u_2)$.

Definition of multi-dimensional LDD. A multi-dimensional LDD is defined as

$$D = (\vec{O}, \vec{B}, \vec{S}, \vec{N}),$$

where $\vec{O} = \langle o_1, \ldots, o_\delta \rangle$, $\vec{B} = \langle b_1, \ldots, b_\delta \rangle$, $\vec{S} = \langle s_1, \ldots, s_\delta \rangle$, $\vec{N} = \langle n_1, \ldots, n_\delta \rangle$, and δ is the number of dimensions. For $1 \leqslant i \leqslant \delta$, $d_i = (o_i, b_i, s_i, n_i)$ is called the LDD of ith dimension.

For a multi-dimensional LDD D, its index set $S[D]$ is defined as the Cartesian product of the index sets of its each dimensional LDD d_i:

$$S[D] = S[d_1] \times S[d_2] \times \cdots \times S[d_\delta].$$

We will abbreviate the above formula as

$$D = d_1 \times d_2 \times \cdots \times d_\delta.$$

Also, the intersection of two multi-dimensional LDD can be computed by dimension-by-dimension intersection. That is, the intersection of two LDDs is the Cartesian product of the intersection of each dimensional LDDs. We can abbreviate this result as

$$D_i \cap D_j = \left(d_{i_1} \times \cdots \times d_{i_\delta}\right) \cap \left(d_{j_1} \times \cdots \times d_{j_\delta}\right) = \left(d_{i_1} \cap d_{j_1}\right) \times \cdots \times \left(d_{i_\delta} \cap d_{j_\delta}\right).$$

Lemma 2. $D_i \cap D_j \neq \emptyset \Longleftrightarrow (d_{i_1} \cap d_{j_1} \neq \emptyset) \wedge \cdots \wedge (d_{i_\delta} \cap d_{j_\delta} \neq \emptyset)$.

Redistribution algorithm based on LDD. To perform a redistribution, for any source–target processor pair, one has to look at the set of elements owned by the source processor before the redistribution (based on the source distribution scheme) and the set of elements owned by the target processor after the redistribution (based on the target distribution). An intersection of these sets is the data that need to be transferred between the pair of source–target processors. The LDD representation is particularly useful to determine which pair of source–target processors need to communicate.

Theorem 1. *Let D_i be the LDD of a source processor P_i under the source distribution scheme $\mathscr{D}_s$, and $\tilde{D}_j$ be the LDD of a target processor $\tilde{P}_j$ under the target distribution scheme $\mathscr{D}_t$. In a redistribution from $(\mathscr{P}_s, \mathscr{D}_s)$ to $(\mathscr{P}_t, \mathscr{D}_t)$, where $\mathscr{P}_s$ and $\mathscr{P}_t$ are the source and target processor set, respectively, the data that each processor P_i $(P_i \in \mathscr{P}_s)$ should communicate with target processor $\tilde{P}_j$, $(\tilde{P}_j \in \mathscr{P}_t)$, are indicated by the intersection of D_i and $\tilde{D}_j$.*

4. Communication scheduling for array redistribution

In the following discussions, we focus on the contention-free communication scheduling in the case of all-to-many communication. We first consider the sending communication scheduling in one-dimensional case, and the multi-dimensioal algorithm will appear in Section 4.3. The case of $b' = \beta * b$ and receiving communication scheduling will be described in Section 4.4. We also assume that the array index starts from 1 while processors are numbered starting from 0.

4.1. One-dimensional scheduling algorithm

We construct a communication matrix (table) COM for the redistribution $(\mathscr{D}_s, \mathscr{P}_s)$ to $(\mathscr{D}_t, \mathscr{P}_t)$. A "1" in the (i, j) entry represents the fact that processor P_i needs to communicate to processor P_j. That is, $\mathrm{COM}[i, j] = 1$ if and only if processor P_i sends data to processor P_j. According to the usage of LDDs, let d_i and d_j be the LDDs of processors P_i and P_j, respectively, then

$$\mathrm{COM}[i, j] = \begin{cases} 1, & d_i \cap d_j \neq \emptyset, \\ 0, & d_i \cap d_j = \emptyset. \end{cases} \tag{1}$$

Our goal is to generate an algorithm that derives another table from COM, called communication scheduling table $\mathrm{CS}[i, k]$, where $0 \leqslant i < P$, $0 \leqslant k < \mathscr{K}$ ($\mathscr{K}$ is the number of communication steps), and if $\mathrm{COM}[i, j] = 1$, there exists a k, $0 \leqslant k < \mathscr{K}$ such that $\mathrm{CS}[i, k] = j$ and each column of CS is a (partial) permutation of processor $0, 1, \ldots, P' - 1$. Fig. 5 shows the examples of the communication table and its corresponding communication scheduling table.

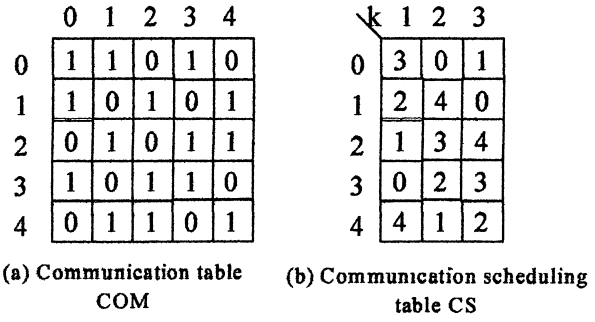

Fig. 5. Examples of the communication table and the communication scheduling table.

Observing the communication table COM with size $P \times P'$, we can obtain the following properties:

1. The vector $(j_1, \ldots, j_{\mathscr{L}_i})$ when $\mathrm{COM}[i, j_k] = 1$ for $1 \leqslant k \leqslant \mathscr{L}_i \leqslant \mathscr{K}$ in row i of COM represents the sending vector of processor P_i, where $\mathscr{L}_i = \sum_{j=0}^{P'-1} \mathrm{COM}[i, j]$, which contains information of the destination nodes. Similarly, the vector $(i_1, \ldots, i_{\mathscr{L}'_j})$ when $\mathrm{COM}[i_k, j] = 1$ for $1 \leqslant k \leqslant \mathscr{L}'_j \leqslant \mathscr{K}'$ in column j of COM represents the receiving vector of processor P_j, where $\mathscr{L}'_j = \sum_{i=0}^{P-1} \mathrm{COM}[i, j]$, which contains information of the source nodes.
2. The number of sending communication steps, $\mathscr{K}$, is the maximum length of the sending vectors: $\mathscr{K} = \max_{0 \leqslant i < P} (\mathscr{L}_i)$, and the number of receiving communication steps $\mathscr{K}'$, is the maximum length of the receiving vectors: $\mathscr{K}' = \max_{0 \leqslant j < P'} (\mathscr{L}'_j)$.

If a reasonable contention-free scheduling can be derived, $\mathscr{K}' \leqslant \mathscr{K}$ needs to be satisfied, otherwise, the number of source processors that send message to a destination processor simultaneously is larger than the communication step. Thus there may be at least two source processors which send message to a destination processor simultaneously after communication scheduling. If the number of processors P, P' and block lengths b, b' can be arbitrary values, it is possible that $\mathscr{K}' > \mathscr{K}$ in some cases (for example, when $P = 12, P' = 8, b = 4, b' = 3$, we have $\mathscr{K}' = 4$ but $\mathscr{K} = 2$). In this paper, we only consider the scheduling algorithm when $\mathscr{K}' \leqslant \mathscr{K}$.

Theorem 2. *In a redistribution operation, if a sending processor P_i need (not) send a message to a receiving processor P_j, then for another sending processor $P_{i'}, 0 \leqslant i' < P$, there certainly exists a receiving processor $P_{j'}$ such that $P_{i'}$ need (not) send a message to $P_{j'}$, and $j' = (j + (i' - i) * (b/b')) \bmod P'$. Using the notion of communication table, this means*

$$\mathrm{COM}[i, j] = 0 \ (or = 1) \iff \mathrm{COM}[i', j'] = 0 \ (or = 1),$$

and $j' = (j + (i' - i) * (b/b')) \bmod P'$.

Proof. Assume $d_i = (i * b + 1, b, s, n)$ and $d_j = (j * b' + 1, b', s', n')$, are two LDDs of the sending processor P_i and receiving processor P_j, respectively, then, according to Formula (1) and Lemma 1,

$$\begin{aligned}
&\mathrm{COM}[i,j] = 0 \\
&\iff\ d_i \cap d_j = \emptyset \iff \forall u,\ v(0 \leqslant u < n, 0 \leqslant v < n'), \\
&\max(i * b + 1 + s * u, j * b' + 1 + s' * v) \\
&> \min(i * b + b + s * u, j * b' + b' + s' * v).
\end{aligned}$$

For another processors $P_{i'}$ and $P_{j'}$ where $j' = (j + (i' - i) * (b/b')) \bmod P'$,

$$\begin{aligned}
&\max\left(i' * b + 1 + s * u, j' * b' + 1 + s' * v\right) \\
&\quad = \max\left(i' * b + 1 + s * u, 1 + \left(j + \left((i' - i) * \frac{b}{b'}\right) \bmod P'\right) * b' + s' * v\right) \\
&\quad = \max\left(1 + i' * b + s * u, 1 + j * b' + \left((i' - i) * \frac{b}{b'} * b'\right)\right. \\
&\qquad\qquad \left.\times \bmod (P' * b') + s' * v\right). \qquad (2)
\end{aligned}$$

Because $P' * b' = s'$ and $x \bmod y = x - y * w$ for a constant w. Assume $v' = v - w$, thus,

$$\begin{aligned}
\text{Eq. (2)} &= \max\left(1 + i' * b + s * u, 1 + j * b' + (i' - i) * \frac{b}{b'} * b' + s' * v'\right) \\
&= \max\left(1 + i * b + s * u, 1 + j * b' + s' * v'\right) + (i' - i) * b \\
&> \min\left((i + 1) * b + s * u, (j + 1) * b' + s' * v'\right) + (i' - i) * b \\
&= \min\left((i' + 1) * b + s * u, (j + 1) * b' + (i' - i) * b + s' * v'\right) \\
&= \min\left((i' + 1) * b + s * u, \left(\left(j + (i' - i) * \frac{b}{b'}\right) \bmod P' + 1\right)\right. \\
&\qquad \left. * b' + s' * v\right) \\
&= \min\left(i' * b + b + s * u, j' * b' + b' + s' * v\right).
\end{aligned}$$

By using the Lemma 1 again, the above inequality means

$$D_{i'} \cap D_{j'} = \emptyset \iff \mathrm{COM}[i', j'] = 0. \qquad \square$$

Definition 1. According to Theorem 2, two entries (i, j) and (i', j') of COM are called symmetrical if and only if

$$j' = \left(j + (i' - i) * \frac{b}{b'}\right) \bmod P'.$$

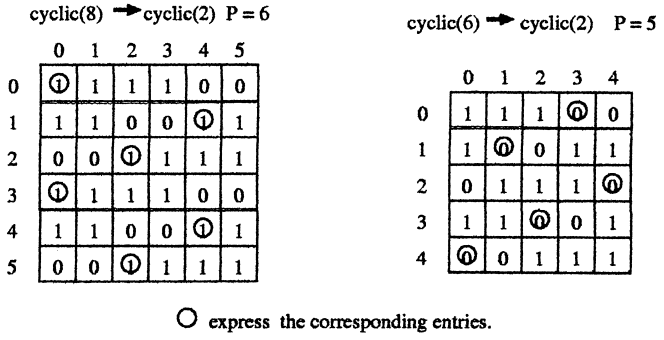

Fig. 6. Examples of communication tables and the symmetrical entries for one-dimensional redistribution.

Fig. 6 shows two communication tables with $P = P' = 6$, $\mathscr{K} = 4$ and $P = P' = 5$, $\mathscr{K} = 3$, respectively. The entries enclosed in a circle are symmetrical entries.

Corollary 1. *For a sending processor P_i, if there exist j_1 and j_2, such that* $\mathrm{COM}[i, j_1] = 0$ *and* $\mathrm{COM}[i, j_2] = 0$ *and* $(j_2 - j_1) \bmod P' = d$, *then for each other sending node $P_{i'}$, there certainly exist j'_1 and j'_2, such that* $\mathrm{COM}[i', j'_1] = 0$ *and* $\mathrm{COM}[i', j'_2] = 0$ *and* $(j'_2 - j'_1) \bmod P' = d$, *where d is a constant, namely distance.*

Proof. According to Theorem 2, $j'_1 = (j_1 + (i' - i) * (b/b')) \bmod P'$ and $j'_2 = (j_2 + (i' - i) * (b/b')) \bmod P'$, therefore, $(j'_2 - j'_1) \bmod P' = (j_2 - j_1) \bmod P' = d$. □

Definition 2. A scheduled sending vector of processor P_i is a sequence of destination processor

$$SV_i = (P_{j_0}, P_{j_1}, \ldots, P_{j_{\mathscr{K}-1}}),$$

where $\mathrm{COM}[i, j_k] = 1$, $0 \leqslant k < \mathscr{K}$, $\mathscr{K}$ is the number of the communication steps, $j_k = (j_0 + k) \bmod \mathscr{K}$, [1] and $j_u \prec j_v$ [2] when $u < v$. SV^k $(= P_{j_k})$ represents the kth entry in vector SV. SV^0 is called the start element of SV.

According to the above definition, the communication scheduling table CS is composed of some SV_i. SV_i is the ith row of CS. From Corollary 1, if the start elements of two sending vectors are different, then all the elements at the same entry are different for these two sending vectors. Therefore, in the following discussions, we only focus on the algorithm that generates the start elements.

[1] Because we assume $b = \beta * b'$, all sending processors redistribute data to some neighbouring receiving processors.

[2] $\prec$ is a special ascending order designated as "$<$" but taken a round of P'. For example, if $P' = 6$ and $j_0 = 3$, then $3 \prec 4 \prec 5 \prec 0 \prec 1 \prec 2$.

From Theorem 2, if $\mathrm{COM}[i_0, j_0] = 1$ (or $=0$) for a sending node P_{i_0}, then there certainly exists corresponding j_k for each other sending node P_{i_k} such that $\mathrm{COM}[i_k, j_k] = 1$ (or $= 0$), $1 \leqslant k < P$. These j_k form a group $(j_0, \ldots, j_{P-1})$, called *relative group RG*.

Definition 3. Suppose J^* is a "1" entry in the row i of COM for a sending node P_i, the kth "1" entry J following J^* in the row i of COM is defined as that $\mathrm{COM}[i, J] = 1$, $J^* \prec J$, and there exist $J^{(1)}, \ldots, J^{(k-1)}$ such that $J^* \prec J^{(1)} \prec \cdots \prec J^{(k-1)} \prec J$, $\mathrm{COM}[i, J^{(l)}] = 1$, $1 \leqslant l \leqslant k-1$.

Algorithm 1. Because the sending vector SV can be determined according to SV^0, we only need to find the start element SV_i^0 for each sending node P_i according to the following steps:

1. Find out a "1" entry as the first "1" entry J_0^* for sending node P_0 where $\mathrm{COM}[0, J_0^*] = 1$ and its relative group $RG = (J_0^*, \ldots, J_{P-1}^*)$.
   (Take Fig. 7 as an example. $RG = (0,4,2,0,4,2)$ and $RG' = (0,3,1,4,2)$ for the communication tables shown in Figs. 7(a) and (b), respectively.)
2. If some J^*s in RG are equal to each other, i.e., $J_{i_1}^* = J_{i_2}^* = \cdots = J_{i_n}^*$ ($J_{i_u}^* \in RG$, $0 \leqslant i_u < P$), then put the sending node $P_{i_1}, \ldots, P_{i_n}$ into a sub-group SN. Thus P sending nodes can be divided into m sub-groups $SN_0, \ldots, SN_{m-1}$, each sub-group SN_i has the same number of elements n, where $P = m * n$.
   (For Fig. 7(a), $SN_0 = (P_0, P_3)$, $SN_1 = (P_1, P_4)$, $SN_2 = (P_2, P_5)$.)

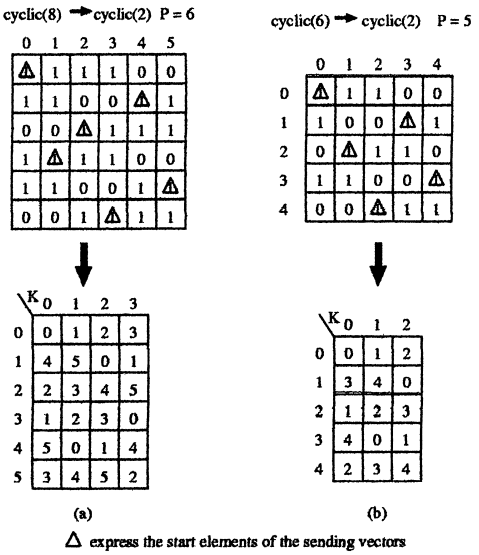

Fig. 7. Generation of scheduling tables from communication tables.

3. If the number of elements of all the sub-groups is 1, that is, there are total P sub-groups, then the start element of each row i is the first "1" entry J_i^*.

(For Fig. 7(b), $SN_0 = (P_0)$, $SN_1 = (P_1)$, $SN_2 = (P_2)$, $SN_3 = (P_3)$, $SN_4 = (P_4)$. Hence, the start elements are $(0, 3, 1, 4, 2)$.)

4. Otherwise, for the sending nodes $P_{i_0}, \ldots, P_{i_{n-1}}$ in a sub-group, the start element is the first "1" entry , 2nd "1" entry, ..., nth "1" entry for the row $i_0, \ldots, i_{n-1}$, respectively.

(Hence, the start elements of (P_0, P_3), (P_1, P_4) and (P_2, P_5) are $(0, 1)$, $(4, 5)$ and $(2, 3)$, respectively.)

5. All such sequences of the processor number which begin at the start element SV^0 are composed of sending vectors SV, which are the rows of the scheduling table CS.

Fig. 7 gives some examples of generation of the scheduling tables from the communication tables, according to Algorithm 1. The entries enclosed in a triangular is the start elements of the sending vectors.

Theorem 3. *Any column of a CS generated from Algorithm* 1 *is a permutation of the processor numbers* $(0, 1, \ldots, P' - 1)$.

Proof.

1. If the SV^0 of each sending vector is derived from Step 3, according to the fact described in Step 2, the start elements of various sending vectors are different from each other.
2. If the SV^0 are derived from Step 4, the maximum number of elements in a sub-group is less than or equal to $\mathscr{K}$. Otherwise, at least in a column j of COM, the number of entry "1" is less than $\mathscr{K}$. In other words, there is a receiving node P_j which receives messages from less than $\mathscr{K}$ node. It is impossible for array redistribution. Therefore, for the nodes in the same sub-group, we can guarantee there exist enough different start elements.
3. According to Theorem 2, for each 1st, 2nd, ..., nth "1" entry of the sending nodes in a sub-group, there must exist corresponding entry in other sub-group and such entry is different each other.
4. According to Corollary 1, all the elements at the same entry are different for these P sending vectors.

Therefore, the elements in any column of a CS are different from each other, that is, the columns are the permutations of processor node numbers. □

4.2. Estimation of the time complexity

Optimizing the scheduling algorithm itself can influence the efficiency of the data-parallel compilers. On the other hand, because the produced results of two kinds of algorithms – communication table, which is used in the application programs, are same, it is more interesting to compare the various efficiency of an application program under using the communication scheduling or not. Therefore, we will do

some experiments to illustrate the results about these in the next section, while only compare the efficiency of our algorithm and Ranka's one by estimating the time complexity in this subsection.

Ranka et al. estimated the approximately average time complexity of their algorithm (compact global masking algorithm) in [13,14] as

- time for compressing COM into another matrix CCOM with size $P \times k$ (no scheduling): $O(P^2)$;
- time for performing the scheduling: $O(k * P * \ln k)$;
- time for performing the communication: $O(k(T_s + T_d * b'))$, where T_s is the communication start-up time and T_d is the transmission time per byte.

Using the same estimating method as the one in [13], we can make the following complexity analysis for Algorithm 1:

- for Step 1, time for finding out J_0^* and its relative group is $O(P)$ in sequential program;
- for Steps 2–4, using the distance d_i far from processor P_0, time for dividing each J_i^* into sub-group and determining the start elements is $O(P) * O(k) = O(P * k)$;
- time for scheduling is $O(k(T_s + T_d * b'))$.

That is, time for scheduling of our algorithm is $O(P) + O(P * k) = O(P * k)$. Comparing with the time complexity of Ranka's algorithm, we can observe that our scheduling algorithm gets better performance, due to the usage of the peculiarity of redistribution. Applying Theorem 2 and Corollary 1 to our algorithm reduces the computation cost for scheduling.

4.3. Multi-dimensional scheduling algorithm

For the multi-dimensional case, if the redistribution is the "shape retaining" case, it can be done by simply looking at the representations for each dimension and performing redistributions dimension-by-dimension independently. In this section, we only consider the "shape changing" case. For the sake of simplicity, in the following discussions, we use two-dimensional case to explain the multi-dimensional redistribution problems. We assume that the processor grid is $P = P_1 \times P_2$ before the redistribution and $P' = P'_1 \times P'_2$ after the redistribution. Hence, the processor P_i can be represented in two-dimensional coordinate, that is $i = (i_1, i_2)$.

With respect to 2D array redistribution, as we proposed in Lemma 2 of Section 3,

$$D_i \cap D_j \neq \emptyset \iff d_{i_1} \cap d_{j_1} \neq \emptyset \wedge d_{i_2} \cap d_{j_2} \neq \emptyset,$$

where $i = (i_1, i_2)$ is the source processor number and $j = (j_1, j_2)$ is the target processor number. Using the notation of the communication table COM, it can be represented as

$$\mathrm{COM}[i, j] = \mathrm{COM}[(i_1, i_2), (j_1, j_2)] = \mathrm{COM}_1[i_1, j_1] \wedge \mathrm{COM}_2[i_2, j_2],$$

where COM_1 and COM_2 are the communication tables corresponding to the first and second dimension, respectively.

The algorithm for determining the start element of each sending node is composed of the 1D algorithms applied repeatedly. The algorithm is as follows.

Algorithm 2. Similar to Algorithm 1, we also consider the start elements only.

1. First consider COM_1. Applying Algorithm 1, we can get the start elements $(J_1^0, \ldots, J_1^{P_1-1})$ for $I_1^0, \ldots, I_1^{P_1-1}$, such that,

$$\mathrm{COM}[(I_1^0, i_2), (J_1^0, j_2)], \ldots, \mathrm{COM}[(I_1^{P_1-1}, i_2), (J_1^{P_1-1}, j_2)]$$

form P_1 number of sub-communication tables. All these sub-communication tables are equal to another communication table COM_2 but no overlapping rows and columns in COM.

2. Then consider sub-communication tables

$$\mathrm{COM}[(I_1^0, i_2), (J_1^0, j_2)], \ldots, \mathrm{COM}[(I_1^{P_1-1}, i_2), (J_1^{P_1-1}, j_2)].$$

Applying Algorithm 1 again,

2.1. If $P_2 \leqslant P_2'$ we can directly apply Algorithm 1 to tables

$$\mathrm{COM}[(I_1^0, i_2), (J_1^0, j_2)], \ldots, \mathrm{COM}[(I_1^{P_1-1}, i_2), (J_1^{P_1-1}, j_2)],$$

and obtain the start elements $(J_2^0 \ldots, J_2^{P_2-1})$ for $I_2^0, \ldots, I_2^{P_2-1}$.

2.2. If $P_2 > P_2'$, it is possible there are not enough columns to get start elements in COM_2, then we compound two sub-communication tables $\mathrm{COM}[(I_1, i_2), (J_1, j_2)]$ and $\mathrm{COM}[(I_1, i_2), (\hat{J}_1, j_2)]$ into a sub-table and use the Algorithm 1 to it, where $\mathrm{COM}_1[I_1, J_1] = 1$ and $\mathrm{COM}_1[I_1, \hat{J}_1] = 1$ and $\hat{J}_1$ is the first "1" entry following J_1.

3. The pairs (J_1^u, J_2^v) are the start elements for each sending node $(I_1^u, I_2^v)(0 \leqslant (u, v) < (P_1, P_2))$.

For example, consider an array redistribution (BLOCK, BLOCK) to (BLOCK, BLOCK) on $P = 2 \times 4$ to $P' = 4 \times 2$. The sub-communication tables COM_1 and COM_2, the communication table COM, and the CS table derived from Algorithm 2 are shown in Fig. 8(a), (b), (c) and (d), respectively, where $\mathrm{COM}[(i_1, i_2), (j_1, j_2)]$ and $\mathrm{CS}[(i_1, i_2), k]$ are represented as $\mathrm{COM}[4i_1 + i_2, 2j_1 + j_2]$ and $\mathrm{CS}[4i_1 + i_2, k]$.

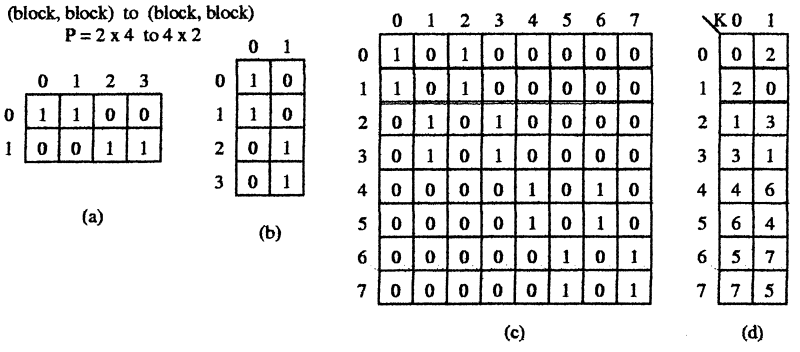

Fig. 8. Example for multi-dimensional redistribution.

4.4. Receiving communication scheduling table

The receiving scheduling table CS′ can be directly derived from the sending scheduling table CS:

$$\mathrm{CS}[i,k] = j \Rightarrow \mathrm{CS}'[j,k] = i,$$

where $0 \leqslant i < P,\ 0 \leqslant j < P',\ 0 \leqslant k < \mathscr{K}$. Also, the sending CS in the case of $b' = \beta * b$ is equal to the receiving CS′ in the case of $b = \beta * b'$.

5. Experimental results

In order to evaluate the ideas presented in this paper, we conducted some experiments implemented on CP-PACS, a 2048-processor MIMD distributed memory parallel computer developed at the University of Tsukuba. All the node programs are written in C, using PARALLELWARE [3] programming environment, a commercially available package that extends C and FORTRAN77 with a portable communication library.

As we mentioned in Section 1, most of the real applications (such as ADI, FFT) which need to use redistribution algorithms have the all-to-all communication pattern. However, the all-to-many redistribution algorithms are useful in the data-parallel compilers, such as HPF compiler, because the HPF users may often write such kind of *REDISTRIBUTE* directives. Therefore, we use an HPF program shown in Fig. 9 as our experimental sample. Because we concentrate our attention on the results that demonstrate the usefulness of the communication scheduling optimizations we presented in this paper, we only measure the execution time of the directive *REDISTRIBUTE* (the HPF compiler invokes a redistribution routine). We use our earlier algorithm without scheduling [2,3] as a base for comparison.

Fig. 10 shows the result of the experiment. Our LDD approach is applied in the index computation part of the redistribution algorithm. The curve "without scheduling" represents the performance of the redistribution without communication scheduling, and the curve "with scheduling" represents the performance of the redistribution with communication scheduling optimization presented in this paper.

Another similar experiment is done with the communication steps $\mathscr{K} = 6$. The result is shown in Fig. 11.

From these figures we observed that the algorithm with the communication scheduling optimization achieves better performance than the former algorithm, no matter how long the communication steps are. The performance improvement becomes more appreciable as the number of processors is increased. This means it is vital to use the communication scheduling in the redistribution algorithms.

[3] PARALLELWARE is a trademark of Nippon Steel Corporation. The trademark of the same software in America is Express.

```
  REAL A(120000), B(120000)
!HPF$ PROCESSOR P(4)
!HPF$ DISTRIBUTE CYCLIC(6) ONTO P :: A
!HPF$ DISTRIBUTE CYCLIC(2) ONTO P :: B
  INTEGER I
  FORALL (I = 1, 120000, 6)
      A[I] = (A[I+1]+A[I+2]+A[I+3]+A[I+4]+A[I+5]) / 5
  FORALL (I = 1, 120000, 2)
      B[I] = (B[I] *B[I+1]) / 2
!HPF$ REDISTRIBUTE A(CYCLIC(2))
  FORALL (I = 1, 120000) B[I] = A[I] + B[I]
```

Fig. 9. A sample HPF program used in the experiment.

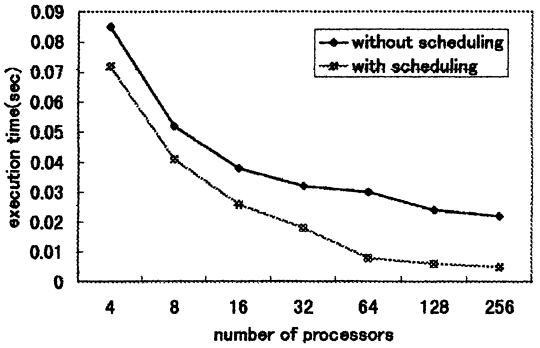

Fig. 10. Comparison of the performance of the redistribution with and without communication scheduling on CP-PACS (data size = 120000, $\mathscr{K} = 3$).

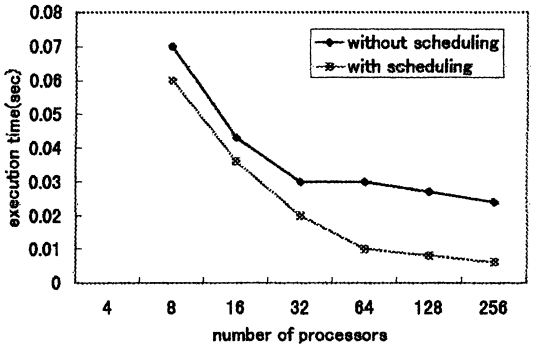

Fig. 11. Comparison of the performance of the redistribution with and without communication scheduling on CP-PACS (data size = 120000, $\mathscr{K} = 6$).

6. Conclusions

Redistribution operations of arrays can be optimized through two approaches – index computation optimization and communication scheduling. In this paper, we have shown an efficient approach for scheduling all-to-many communication in redistribution. Based on the notion of LDD proposed in [2–4], our communication schedulings are designed using the communication table and communication scheduling table. The communication scheduling results in a permutation of the destination processors in each communication step. The communication steps of each sending processor are exactly equal to the number of receiving processors which sending processor must communicate with. Thus the communications in our redistribution algorithm are node contention-free. Our approach can deal with multi-dimensional "shape changing redistribution". However, our current algorithm can only process the case of $b = \beta * b'$. In future, we will extend it to the general case and consider the factor of message length into our scheduling algorithms.

References

[1] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, Y. Robert, Scheduling block-cyclic array redistribution, IEEE Transactions on Parallel and Distributed Systems 9 (2) (1998) 192–205.
[2] M. Guo, Y. Yamashita, I. Nakata, Efficient implementation of multi-dimensional array redistribution, IEICE Transactions on Information and Systems E81-D (11) (1998) 1195–1204.
[3] M. Guo, Y. Yamashita, I. Nakata, Improving performance of multi-dimensional array redistribution on distributed memory machines, in: Proceedings of the Third International Workshop on High-Level Parallel Programming Models and Supportive Environments, March 1998, Orlando, FL, USA.
[4] M. Guo, Efficient techniques for data distribution and redistribution in parallelizing compilers, Ph.D. Thesis, University of Tsukuba, Japan, July 1998.
[5] HPF Forum: High Performance Fortran Language Specification, Rice University, Houston, TX, version 2.0 edition, November 1996.
[6] S.D. Kaushik, C.-H. Huang, R.W. Johmson, P. Sadayappan, An approach to communication-efficient data redistribution, in: Proceedings of the Eighth ACM International Conference on Supercomputing, July 1994, Manchester, UK.
[7] S.D. Kaushik, C.-H. Huang, J. Ramanujam, P. Sadayappan, Multi-phase redistribution: a communication-efficient approach to array redistribution, Technical Report, The Ohio Sate University, 1995.
[8] E.T. Kalns, L.M. Ni, Processor mapping techniques toward efficient data redistribution, IEEE Transactions on Parallel and Distributed Systems 6 (12) (1995) 1234–1247.
[9] Y.W. Lim, P.B. Bhat, V. Prasanna, Efficient algorithms for block-cyclic redistribution of arrays, in: Proceedings of the IEEE Symposium on Parallel and Distributed Processing, October 1996.
[10] Y.W. Lim, N. Park, V. Prasanna, Efficient algorithms for multi-dimensional block-cyclic redistribution of arrays, in: Proceedings of the 26th International Conference on Parallel Processing, August 1997, Bloomingdale, IL.
[11] K. Nakazawa, H. Nakamura, T. Boku, I. Nakata, Y. Yamashita, CP-PACS: a massively parallel processor at the University of Tsukuba, Parallel Computing 25 (13&14) (1999) 1635–1661.
[12] S. Ramaswamy, B. Simons, P. Banerjee, Optimizations for efficient array redistribution on distributed memory multicomputers, Journal of Parallel and Distributed Computing 38 (1996) 217–228.
[13] S. Ranka, J.-C. Wang, G. Fox, Static and run-time algorithms for all-to-many personalized communication on permutation networks, IEEE Transactions on Parallel and Distributed Systems 5 (12) (1994) 1266–1274.

[14] S. Ranka, R. Shankar, K. Alsabti, Many-to-many personalized communication with bounded traffic, in: Proceeding of Frontiers'95, 1995.
[15] R. Thakur, A. Choudhary, G. Fox, Runtime array redistribution in HPF programs, in: Proceedings of the Scalable High Performance Computing Conference, May 1994, pp. 309–316.
[16] R. Thakur, A. Choudhary, J. Ramanujam, Efficient algorithms for array redistribution, IEEE Transactions on Parallel and Distributed Systems 7 (6) (1996) 587–593.