Optimizing Scheduling Stability for Runtime Data Alignment*

Ching-Hsien Hsu, Chao-Yang Lan, and Shih-Chang Chen

Department of Computer Science and Information Engineering
Chung Hua University, Hsinchu, Taiwan 300, R.O.C.
chh@chu.edu.tw

Abstract. Runtime data alignment has been paid attention recently since it can allocate data segment to processors dynamically according to applications' requirement. One of the key optimizations of this problem is to schedule simultaneous communications to avoid contention and to minimize the overall communication costs. The NP-completeness of the problem has instigated researchers to propose different heuristic algorithms. In this paper, we present an algorithm independent technique for optimizing scheduling stability of different scheduling heuristics. The proposed technique introduces a new scheduling policy, Local Message Reduction (LMR), to obtain better communication schedule adaptive to different environments. o evaluate the performance of the proposed technique, we have implemented LMR along with two existing algorithms, the two-phase degree reduction and the list scheduling algorithms. The experimental results show that the proposed technique is effective in terms of scheduling stability, communication efficiency and easy to implement.

1 Introduction

The data parallel programming model has become a widely accepted paradigm for parallel programming on distributed memory systems. To efficiently execute a data parallel program, appropriate data distribution is critical. An optimal distribution of data can balance the computational load, increase data locality, and reduce inter-processor communication. In general, `BLOCK-CYCLIC` is employed for regular data distribution while `GEN_BLOCK` allows different size of data segments be partitioned and aligned to different computing nodes, which is usually referred as irregular data distribution.

In many applications, an origin data distribution that is well-suited for one phase of a parallel program may not be good, in terms of performance, for a subsequent phase. Therefore, dynamic data alignment (or data redistribution) may be invoked as runtime operation or occur implicitly at subprogram boundaries. The use of dynamic data alignment represents a performance tradeoff between the expected higher efficiency of the new distribution scheme for subsequent computation and the overheads of

* The work of this paper is supported by National Science Council, Taiwan, under grant number NSC94-2213-E-216-002.

X. Zhou et al. (Eds.): EUC Workshops 2006, LNCS 4097, pp. 825 – 835, 2006.

redistributing data among processors. Thus, optimizing the communication costs of data redistribution has obvious merit.

Some research results were presented in previous published literature. However, the variant of transmission overheads of local and remote messages was not considered in these scheduling heuristics. That is, local and remote transmissions are distinguished only by message size in existing scheduling policies. This usually leads scheduling instability, namely, a local transmission waits the completion of the other remote transmission in the same communication step if both have the same message size. To remedy this shortcoming, improving the scheduling stability is the main objective of this paper.

The main idea of the proposed technique is first estimate remote access time (*RAT*) of transmitting a message to other computing nodes through the interconnection network and the local access time (*LAT*) of moving a message in local memory upon the same computing node. According to the ration of remote to local access time (*RLR*), a constant for reducing size of local message, one can schedule communications of irregular data redistribution more precisely. The proposed Local Message Reduction (*LMR*) technique is an algorithm independent technique for scheduling `GEN_BLOCK` redistribution. *LMR* enhances scheduling stability and produces better communication efficiency.

The rest of this paper is organized as follows. In section 2, a brief survey of related work will be presented. In section 3, we will give an example of scheduling `GEN_BLOCK` data redistribution as preliminary. Section 4 presents the proposed local message reduction (*LMR*) technique. A detailed description of incorporating the *LMR* scheme with *TPDR* scheduling algorithm [7] will also be investigated. The theoretical performance analysis and experimental comparison will be given in Section 5. Section 6 briefly concludes this paper.

2 Related Work

Techniques for dynamic data alignments are discussed in many researches and can be classified into regular and irregular instances.

Prylli *et al*. [9] focused on index sets generation for the regular problems. Zapata *et al.* [1] presented algorithmic codes for parallel sparse redistribution based on CRS data structure. For communication scheduling techniques, different heuristic scheduling algorithms for regular `BLOCK-CYCLIC` data redistribution between arbitrary processor sets were proposed in [10]. In [3], Guo *et al*. also presented an approach for scheduling all-to-many communications in redistribution. Hsu *et al*. [5] discussed processor mapping techniques for array redistribution. Wakatani *et al*. [11] proposed techniques for overlapping communication and computational overheads. The multiphase redistribution method for minimizing the startup cost was presented in [6]. Guo *et al*. [2] presented techniques for message generation and communication scheduling which not only minimize communication steps but also eliminate node contention. They also presented an efficient data distribution technique on distributed memory parallel computers.

For irregular problems, Guo *et al.* [4] proposed communication optimization techniques for generating GEN_BLOCK communication sets. For communication schedule, Lee *et al.* [8] presented logical processor reordering algorithms to minimize the size of messages that will be transmitted through the redistribution. Wang *et al.* [12] proposed a Divide-and-Conquer (*DC*) algorithm for communication schedule. In *DC*, messages are separated into groups and then merged to produce minimal communication steps. Yook *et al.* [13] proposed a reallocation scheduling algorithm. Messages are sorted first and then scheduled in a decreasing order. Once a message can't be scheduled into minimal steps, the allocated messages will be reallocated until an appropriate schedule is found.

In this paper, we present a scheduling stability improving scheme which is an algorithm independent technique for scheduling GEN_BLOCK redistribution. The proposed scheduling algorithm introduces a new scheduling policy that based on Local Message Reduction (*LMR*) to obtain better scheduling result.

3 Preliminary

The following code segment is an example of GEN_BLOCK data redistribution in HPF. An origin GEN_BLOCK distribution is declared as parameter ***S*** which is used to perform data decomposition onto five processors. Parameter ***new*** declares a different GEN_BLOCK distribution scheme and then used to redistribute array A.

```
      PARAMETER (S = /99, 10, 113, 114, 164/)
!HPF$ PROCESSORS   P(5)
            REAL A(500), new (5)
!HPF$ DISTRIBUTE A (GEN_BLOCK(S)) onto P
!HPF$ DYNAMIC
            new = /132, 76, 114, 122, 26/
!HPF$ REDISTRIBUTE A (GEN_BLOCK(new))
```

Figure 1 (a) shows the two GEN_BLOCK distribution schemes on array A[1:500] as shown in the above HPF code. The GEN_BLOCK distribution schemes of ***S*** and ***new*** are regarded as the source and destination distribution for different computational phases, respectively. SP_0~SP_4 and DP_0~DP_4 are denoted as those processors who have corresponding data according to the source and destination distribution schemes, respectively. *Size* represents the size of data segment that distributed to each corresponding processor.

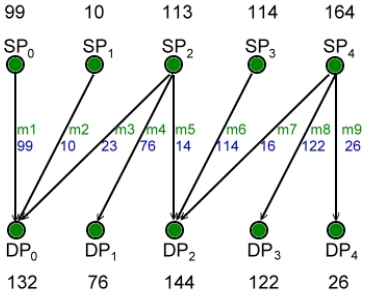
A convex bipartite graph is used to illustrate the communication patterns between source and destination processors in figure 1 (b). Vertices are denoted as processors while edges are denoted as communication messages. Numbers that are attached with SP_0~SP_4 and DP_0~DP_4 represent the size of data segment that owned in each processor according to the source and destination distribution parameters. The weight of edges denotes the size of messages. Indices of m_1, m_2, …, m_9 show the serial of these messages.

In general, node contention is usually avoided by ensuring one processor sends or receives at most one message at a communication step. For example, if messages m_1, m_2 and m_3 are sent to DP_0 in the same communication step, contention will be occurred. These messages are usually scheduled into different communication steps to avoid such situation. Those messages that can not be scheduled in the same communication step are called conflict tuple [12].

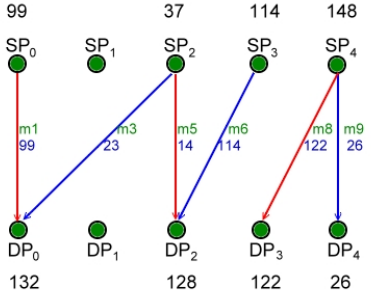
Distribution Scheme of Source Processor					
SP	SP_0	SP_1	SP_2	SP_3	SP_4
Size	99	10	113	114	164

Distribution Scheme of Destination Processor					
DP	DP_0	DP_1	DP_2	DP_3	DP_4
Size	132	76	114	122	26

(a)



(b)



(c)

Schedule Table		Cost	
Step 1	m_3 、m_6 、m_8	122	(m_8)
Step 2	m_1 、m_5 、m_9	99	(m_1)
Step 3	m_2 、m_4 、m_7	76	(m_4)
Total cost = 297			

(d)

Fig. 1. An example of scheduling irregular data redistribution using *TPDR*. (a) `GEN_BLOCK` distribution schemes. (b) A bipartite graph represents the communications between source and destination processors. (c) The adjustable coloring mechanism colors edges red and blue. (d) The scheduling result of *TPDR*.

TPDR scheduling algorithm [7] employs two scheduling phases: a degree reduction iteration phase and an adjustable coloring mechanism phase. The aim of degree reduction phase is to reduce the maximal degree (d_{max}) of the bipartite graph equals to 2. As d_{max} is reduced to 2, the *TPDR* algorithm goes into the coloring phase. The adjustable coloring mechanism separates messages by two colors as step 1 and step 2. It is usual to see that the degree reduction phase results the bipartite graph consists of several connected components. Therefore, colors can be exchanged among these groups to shorten the overall cost. In figure 1(c), messages are colored red (m_1, m_5 and m_8.) and blue (m_3, m_6 and m_9.). There are two connected components in the resulting bipartite graph from the first phase's completion. The first sub-graph includes

messages m_1, m_3, m_5 and m_6. The second one includes m_8 and m_9. The two colors lead messages to be scheduled in different steps. The scheduling result of *TPDR* is shown in figure 1(d). (If size of m_8 is 26 and m_9 is 122, the adjustable coloring mechanism will exchange the colors of m_8 and m_9). Communication cost of a step is defined by the maximal message size in that step. The theoretical communication cost will be the sum of communication cost of each step. In this example, the cost of steps 1~3 are 122, 99 and 76, respectively. The total communication cost of these three steps is 297.

The above example demonstrates that both local and remote transmissions are scheduled together. For those messages sent to other processors must be transmitted through interconnection network while local messages are moved in local memory, e.g. m_1, m_5 and m_9. It is known that local access time is smaller than remote access time. This notion should hint that ignoring the difference may lead inefficient scheduling results.

4 Local Message Reduction Technique

Local message reduction is an algorithm independent technique for improving scheduling efficiency of dynamic data redistribution. The notion of *LMR* is to modify the communication cost of local messages to reflect actual data transmission overheads. The scheduling of communications will be performed based on the size-modified messages.

In order to generate communication cost of messages more fairly, local message reduction method is used to adjust the way we evaluate the transmitting cost of a message. The concept of incorporating *LMR* with *TPDR* is abstracted as follows.

Phase 1: Estimate *RLR*, the remote to local data access ratio, which is defined as *RLR* = *RAT* / *LAT*, where *RAT* is remote access time and *LAT* is local access time. In our implementation, *RAT* and *LAT* are measured by transferring 10 MB data between two processors that are interconnected by 100 MB Layer 2 switch.

Phase 2: Identify all local messages in a given irregular data redistribution instance.

Phase 3: Reduce the communication cost of those messages found in last phase by the factor *RLR*.

We are now giving an example to clarify the above description. For the example in Figure 1, the transmitting cost of m_1 is 99 originally, let *RLR* be 8, and then the cost of m_1 will be reduced by a factor, *RLR*=8, becomes 13.

In the example illustrated in figure 1, there are three local messages. The transmitting cost of m_1, m_5 and m_9 are modified to be 13, 2 and 4, respectively. The cost of step 2 will become 13, and total cost becomes 211. The complete schedule is given in figure 2.

Figure 2 shows the example of *LMR* been applied after the communications are scheduled by *TPDR*. In fact, the *LMR* policy should be considered before the communications are scheduled. Namely, the communication cost of messages should be reduced properly before they are scheduled. The result of using *LMR* before scheduling is given in figure 3. Comparing with the result in figure 2, the

communication cost is reduced by 56. This example demonstrates that modifying the transmitting cost of messages could make *TPDR* schedule better result which is expected more precisely. We will discuss this expectation by the experimental test in next section.

Schedule Table		Cost	
Step 1	m_3 、m_6 、m_8	122	(m_8)
Step 2	**m_1 、m_5 、m_9**	**13**	(m_1)
Step 3	m_2 、m_4 、m_7	76	(m_4)
Total cost = **211**			

Fig. 2. Scheduling result by *LMR* been applied after the communications are scheduled by *TPDR*

Schedule Table		Cost	
Step 1	m_1 、m_4 、m_6 、m_8	122	(m_8)
Step 2	m_3 、m_7	23	(m_3)
Step 3	m_2 、m_5 、m_9	10	(m_2)
Total cost = 155			

Fig. 3. Scheduling result by *LMR* been applied before the communications are scheduled by *TPDR*

5 Performance Evaluation

5.1 Simulation Comparison

To evaluate the performance of the proposed technique, we have implemented the *TPDR* scheduling algorithm, the coloring scheduling strategy, a modified size-oriented *TPDR* scheduling algorithm and list scheduling with coloring mechanism to verify the beneficial of the *LMR* technique. To simplify the presentation, we use *TPDR*, *Coloring*, *Max-TPDR* and *List-Coloring* to represent these four algorithms. We briefly describe each of these algorithms as follows.

Coloring is a simple scheduling method that guarantees minimal communication steps for `GEN_BLOCK` redistribution. The idea comes from the edge coloring on bipartite graph. This algorithm focuses on communication steps but disregards the transmitting cost of messages.

TPDR employs two phases to schedule messages, the degree reduction phase and the adjustable coloring phase. Details of *TPDR* were described in section 3.

Max-TPDR is a *TPDR* based scheduling algorithm. It employs the same policy as the *TPDR* does. The degree reduction method is first used to schedule communications in the first phase and followed by the adjustable coloring mechanism. The main difference between *TPDR* and *Max-TPDR* is that *TPDR* schedules small messages in the degree reduction phase. In contrast, *Max-TPDR* schedules large messages first. This modification makes both algorithms of *TPDR* and *Max-TPDR* deliver different scheduling result although they are almost the same.

The *List-Coloring* is composed by two scheduling methods, a list scheduling (size-oriented) algorithm and a coloring strategy. In the first phase, the *List-Coloring* sorts messages in non-increasing order according to message size. Larger cost messages will be scheduled first. The first d-2 communication step(s) will be determined in this phase. The second phase of *List-Coloring* employs adjustable coloring mechanism to schedule messages that haven't been scheduled. The second phase of *List-Coloring* is the same with *TPDR* and *Max-TPDR* scheduling.

For simulation, 1000 GEN_BLOCK data redistribution instances were randomly generated based on an array A[1:10000] over *P* processors. The average size of data segment been assigned to one processor will be 10000/*P*. The heterogeneity of irregular data decomposition is accomplished by varying the size of data segment from 1 to (10000/*P*)*2.

Figure 4 shows the simulation results of estimating total communication costs for 1000 randomly generated redistribution instances. Four different scheduling algorithms were compared. In figure 4, the notations "*LMR*–" and "–*LMR*" denote that local message reduction policy is applied before and after the communications are scheduled, respectively. The total communication cost is calculated by summing total costs of 1000 schedules in which local messages are reduced (by *RLR*=8). The simulation results in figure 4 show the effect of algorithms incorporate with local message reduction policy. An important observation is that *LMR-XXX* algorithm performs better than *XXX-LMR* algorithm. This phenomenon matches the results illustrated in figure 3.

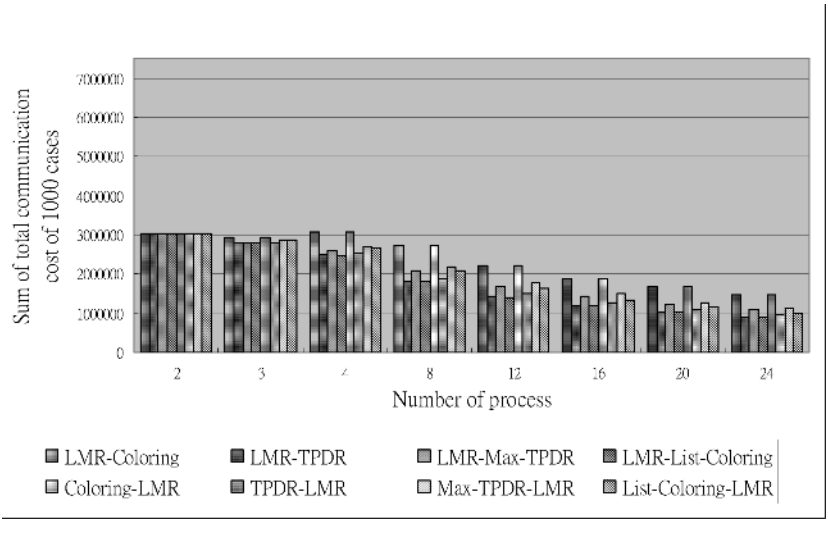


Fig. 4. Total communication cost of 1000 randomly generated redistribution instances of different scheduling algorithms

Figure 5 shows the speedup of the *LMR-TPDR*, *LMR-Max-TPDR* and *LMR-List-Coloring* scheduling algorithms. The speedups of these three methods are defined as follows.

$Speedup_{LMR\text{-}TPDR}$ = (Total cost of *LMR-Coloring*) / (Total cost of *LMR-TPDR*).

$Speedup_{LMR\text{-}Max\text{-}TPDR}$ = (Total cost of *LMR-Coloring*) / (Total cost of *LMR-Max-TPDR*).

$Speedup_{LMR\text{-}List\text{-}Coloring}$ = (Total cost of *LMR-Coloring*) / (Total cost of *LMR-List-Coloring*).

LMR-TPDR and *LMR-List-Coloring* deliver similar speedup. This result implies that *LMR-TPDR* and *LMR-List-Coloring* are well suited for scheduling irregular redistribution problems if the *LMR* strategy is applied before scheduling communications of irregular data redistribution.

Figure 6 shows the speedup of the *TPDR-LMR*, *Max-TPDR-LMR* and *List-Coloring-LMR* scheduling algorithms. The speedups of these three methods are defined as follows.

$Speedup_{TPDR\text{-}LMR}$ = (Total cost of *Coloring-LMR*) / (Total cost of *TPDR-LMR*).
$Speedup_{Max\text{-}TPDR\text{-}LMR}$ = (Total cost of *Coloring-LMR*) / (Total cost of *Max-TPDR-LMR*).
$Speedup_{List\text{-}Coloring\text{-}LMR}$ = (Total cost of *Coloring-LMR*) / (Total cost of *List-Coloring-LMR*).

The speedup of *TPDR-LMR* is better than *List-Coloring-LMR*. The result of figure 6 encourages that *TPDR-LMR* is the best choice for scheduling irregular redistribution problems if the *LMR* strategy will be applied.

To further discover the improvement achievable by *LMR* technique, figure 7 compares *LMR-TPDR* and *TPDR-LMR* separately using 1000 randomly generated instances. The notation "*LMR-TPDR* better" represents *LMR-TPDR* performs better than *TPDR-LMR*; the notation "*TPDR-LMR* better" represents *LMR-TPDR* performs

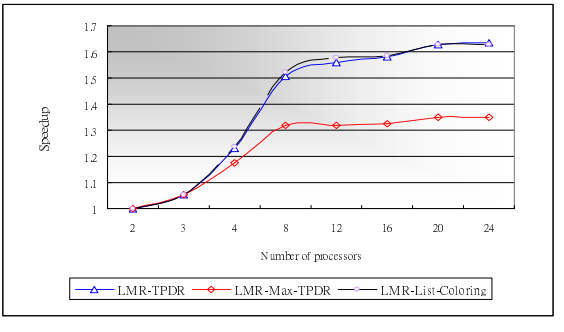


Fig. 5. Speedup of *LMR-TPDR*, *LMR-Max-TPDR* and *LMR-List-Coloring*

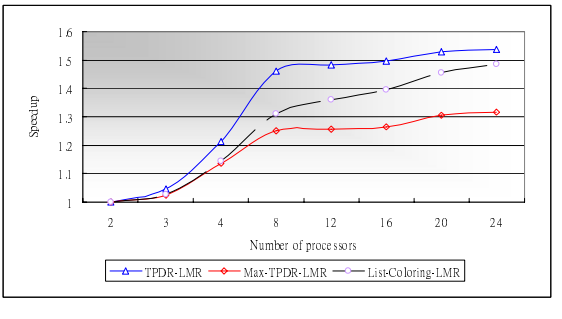


Fig. 6. Speedup of *TPDR-LMR*, *Max-TPDR-LMR* and *List-Coloring-LMR*

worse than *TPDR-LMR*; When the number of processors is 2 and 3, there is no *TPDR-LMR* better case, *LMR-TPDR* performs much better than *TPDR-LMR* when processor number is from 4 to 24. This result encourages us *LMR-TPDR* is a suitable policy for scheduling irregular data redistribution.

5.2 Experimental Results

We have implemented the *TPDR*, *Coloring*, *Max-TPDR* and *List-Coloring* algorithms incorporated with "*LMR-*" and "*-LMR*" policies. All programs were written in single program multiple data (SPMD) programming paradigm with C+MPI code and executed on a 16-nodes PC cluster. The mean time of ten randomly generated `GEN_BLCOK` redistribution instances is used to report the execution time of an algorithm. Experiments are performed base on array A[1:1000] over 4, 8, 12 and 16 processors. Each data element is a 1 MB data file in this implementation. In other words, the size of total data is 1 GB. The value of *RLR* is set as 8. The execution result

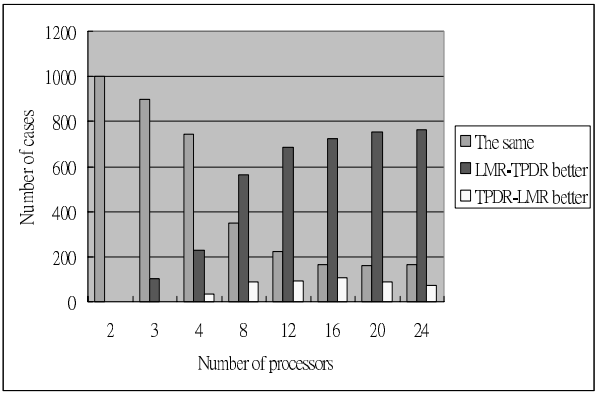


Fig. 7. The comparison of *LMR-TPDR* and *TPDR-LMR*

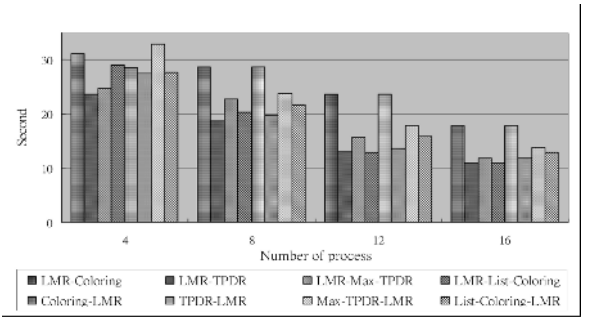


Fig. 8. The average communication time of 10 randomly generated `GEN_BLCOK` redistribution instances executed on 4, 8, 12 and 16 processors

is plotted in figure 8. As predicted in figure 7, *LMR-TPDR* outperforms other methods for most cases. This result also matches the expectation described in sections 4 and 5.1.

6 Conclusions

In this paper, we have proposed a novel technique for enhancing the stability of scheduling `GEN_BLOCK` redistribution. By measuring remote to local access ratio, a local message reduction (*LMR*) strategy is incorporated into our previous proposed scheduling heuristic, named *LMR-TPDR* scheduling algorithm. *LMR-TPDR* reflects fair transmitting cost and enhances scheduling algorithms to obtain better scheduling results. The *LMR* scheduling scheme is an algorithm independent technique and easy to implement. The simulation analysis shows that *LMR-TPDR* is in most cases equal or superior to other scheduling algorithms due to the enhanced scheduling upon actual communication cost. Experimental results verified the simulation is trustworthy. Overall speaking, the performance of the *LMR* technique has been observed to fit most irregular data redistribution.

References

1. G. Bandera and E.L. Zapata, "Sparse Matrix Block-Cyclic Redistribution," *Proceeding of IEEE Int'l. Parallel Processing Symposium* (IPPS'99), San Juan, Puerto Rico, April 1999.
2. Minyi Guo and I. Nakata, "A Framework for Efficient Array Redistribution on Distributed Memory Multicomputers," *The Journal of Supercomputing*, vol. 20, no. 3, pp. 243-265, 2001.
3. Minyi Guo, I. Nakata and Y. Yamashita, "Contention-Free Communication Scheduling for Array Redistribution," *Parallel Computing*, vol. 26, no.8, pp. 1325-1343, 2000.
4. Minyi Guo, Yi Pan and Zhen Liu, "Symbolic Communication Set Generation for Irregular Parallel Applications," *The Journal of Supercomputing*, vol. 25, pp. 199-214, 2003.
5. C.-H Hsu, Dong-Lin Yang, Yeh-Ching Chung and Chyi-Ren Dow, "A Generalized Processor Mapping Technique for Array Redistribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, vol. 7, pp. 743-757, July 2001.
6. S. D. Kaushik, C. H. Huang, J. Ramanujam and P. Sadayappan, "Multiphase data redistribution: Modeling and evaluation," *Proceeding of IPPS*'95, pp. 441-445, 1995.
7. Shih-Chang Chen, Ching-Hsien Hsu, Chao-Yang Lan, Chao-Tung Yang and Kuan-Ching Li, "Efficient Communication Scheduling Methods for Irregular Array Redistribution in Parallelizing Compilers," *Lecture Notes in Computer Science*, Vol. 3606, pp. 216-225, Springer-Verlag, Sep. 2005. (PaCT'05).
8. S. Lee, H. Yook, M. Koo and M. Park, "Processor reordering algorithms toward efficient *GEN_BLOCK* redistribution," *Proceedings of the ACM symposium on Applied computing*, 2001.
9. L. Prylli and B. Tourancheau, "Fast runtime block cyclic data redistribution on multiprocessors," *JPDC*, vol. 45, pp. 63-72, Aug. 1997.
10. Neungsoo Park, Viktor K. Prasanna and Cauligi S. Raghavendra, "Efficient Algorithms for Block-Cyclic Data redistribution Between Processor Sets," *IEEE Trans. on PDS*, vol. 10, No. 12, pp.1217-1240, Dec. 1999.

11. Akiyoshi Wakatani and Michael Wolfe, “Optimization of Data redistribution for Distributed Memory Multicomputers,” short communication, *Parallel Computing*, vol. 21, no. 9, pp. 1485-1490, September 1995.
12. Hui Wang, Minyi Guo and Daming Wei, "Divide-and-conquer Algorithm for Irregular Redistributions in Parallelizing Compilers”, *The Journal of Supercomputing*, vol. 29, no. 2, 2004.
13. H.-G. Yook and Myung-Soon Park, “Scheduling *GEN_BLOCK* Array Redistribution,” *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems*, November, 1999.