An Efficient Scheduling Algorithm for Irregular Data Redistribution

Kun-Ming Yu and Yi-Lin Tsai

Department of Computer Science and Information Engineering
Chung Hua University, Hsinchu, Taiwan 300, ROC.
Tel : 886-3-5186412
Fax: 886-3-5329701
Email: yu@chu.edu.tw

Abstract. Dynamic data redistribution is used to enhance the performance of an algorithm and to achieve data locality in parallel programs on distributed memory multi-computers. The data redistribution problem has been extensively studied. Previous results focus on reducing index computational cost, schedule computational cost, and message packing/unpacking cost. However, irregular data redistribution is more flexible than regular data redistribution; it can distribute different sizes of data segments of each processor to those processors according to their own computation capability. High Performance Fortran 2 (HPF-2), the current version of HPF, provides an irregular distribution functionality, such as GEN_BLOCK which addresses some requirements of irregular applications for the distribution of data in an irregular manner and explicit control of load balancing. In this paper, we present a degree-reduction-and-coloring (DRC) algorithm for scheduling HPF2 irregular array redistribution. We devoted to obtain the minimal number of transmission steps as well as to reduce the overall redistribution time. In the proposed algorithm, we try to reduce the number of maximum transmission messages in the first phase and then apply graph-coloring mechanism to obtain the final schedule. The proposed method not only avoids node contention, but also shortens the overall redistribution time. To evaluate the performance of DRC algorithm, we have implemented DRC algorithms along with the Divide-and-Conquer algorithm. The simulation results show that DRC algorithm has significant improvement on communication costs compared with the Divide-and-Conquer algorithm.

Keywords: Irregular redistribution, communication scheduling, GEN_BLOCK, degree-reduction

1. Introduction

Parallel computing systems have been widely adopted to solve complex scientific and engineering problems. To efficiently execute a data-parallel program on distributed memory multi-computers, an appropriate data distribution is critical to the performance. Appropriate distribution can balance the computational load, increase data locality, and reduce inter-processor communication. Array redistribution is crucial for system performance because a specific array distribution may be appropriate for the current phase, but incompatible for the subsequent one. Many parallel programming languages thus support run-time primitives for rearranging the array distribution of a program. The data redistribution problem has been widely studied in the literature. In general, data redistribution can be classified into two categories: the regular data redistribution [1,5,6,7,9,11,13,15,18] and the irregular data redistribution [4,8,22-24]. The regular data redistribution decomposes data of equal sizes into processors. There are three types of this data redistribution, called BLOCK, CYCLIC, and BLOCK-CYCLIC(n). The irregular data distribution employs user-defined functions to specify data distribution unevenly. High Performance FORTRAN 2 (HPF-2) provides GEN_BLOCK functionality and makes it possible to handle different processors dealing with appropriate data size according to their computation capability. Previous works emphasized the minimal steps of data redistribution and scheduled the ordering of messages with minimal total transmission size. In the regular array redistribution, [15] proposed an Optimal Processor Mapping (OPM) scheme to minimize the data transmission cost for general BLOCK-CYCLIC regular data realignment. Optimal Processor Mapping (OPM) utilized the maximum matching of realignment logical processors to achieve the maximum data hits for reducing the amount of data exchange transmission cost. In the irregular array redistribution problem, [22, 23] proposed a greedy algorithm to utilize the Divide-and-Conquer technique to obtain near optimal scheduling while attempting to minimize the size of total communication messages as well as the number of steps.

In this paper, we bring up the Degree-Reduction-and-Coloring (DRC) algorithm to efficiently perform GEN_BLOCK array redistribution. In section 2, we define the data communication model of irregular data redistribution and give an example of GEN_BLOCK data redistribution as the preliminary. Section 3 describes the DRC algorithm for the irregular redistribution problem. The performance analysis, simulation results and practical transmission with MPI on SMP/Linux cluster are presented in section 4. Finally, the conclusions are given in section 5.

2. Data communication models

In this section, we present some properties of irregular data redistribution with GEN_BLOCK functionality. There are no repetitive communication patterns in the irregular GEN_BLOCK array redistribution. A data redistribution can be represented by a bipartite graph, called a *redistribution graph*. To simplify the presentation, notations and terminologies used in this paper are defined in the following.

***Definition* 1**: Given an irregular GEN_BLOCK redistribution on array $A[SPi]$ and $A[DPi]$ over P processors, the *source processors* of array data elements $A[SPi]$ are denoted as SPi; the *destination processors* of array elements $A[DPi]$ are denoted as DPi where $1 \leq i \leq P$.

***Definition* 2**: A bipartite graph $G = (V, E)$ is used to represent the communications of an irregular data redistribution between source and destination processors. Vertices of G are used to represent the processors. Edge e_{ij} in G denotes the message sent from SPi to DPj, where $e_{ij} \in$ E. $|E_{ij}|$ denotes the transmission message size through the redistribution.

***Definition* 3**: Every message transmission link in irregular data redistribution is not overlapped. Hence, the total number of message transmission link E is $P \leq E \leq 2 \times P - 1$.

***Definition* 4**: Each processor has more than one e_{ij} to send data to destination processors or receive data from other source processors. The number D of e_{ij} owned by one processor is denoted as *D-degree* and the maximum *D-degree* of all processors is denoted as *Max-degree*. We denote that the processors have the *Max-degree* number of messages as P_{max}.

***Definition* 5**: If SPi sends messages to $DPj-1$ and $DPj+1$, the transmission between SPi and DPj must exist, where $1 \leq i, j \leq P$. This result was mentioned as the consecutive communication property [12].

Fig.1 shows an example of redistributing two GEN_BLOCK distributions on $A[SPi]$ and $A[DPi]$. Table 1(a) shows mapped communication message size to source processors and destination processors, respectively. The communications between source and destination processor sets are depicted in Fig 1. There are 13 transmission messages, e_{11}, e_{21}, e_{22}, …e_{77} among the processors involved in the redistribution. Due to the considerable influence of node contention, a processor can only send at most one message to another processor in each communication step and the same is true for the receiving message. The messages, which cannot be scheduled in the same communication step, are called conflict tuple. For instance, $\{e_{11},e_{21}\}$ is a conflict tuple since they have a common destination processor *DP1*; $\{e_{21},e_{22}\}$ is also a conflict tuple because of the common source processor *SP2*. Table 1(b) shows a simple schedule result for this example.

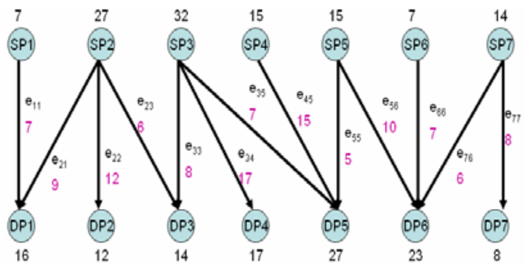


Figure 1. An example of data redistribution

Table 1(a). The total message size of redistribution data for each processor in Fig. 1.

SP1	SP2	SP3	SP4	SP5	SP6	SP7
7	27	32	15	15	7	14
DP1	DP2	DP3	DP4	DP5	DP6	DP7
16	12	14	17	27	23	8

Table 1(b). A simple schedule

Schedule Table	
Step1:	e_{34}, e_{45}, e_{22}, e_{77}, e_{11}, e_{66}
Step2:	e_{56}, e_{23}, e_{35}
Step3:	e_{21}, e_{33}, e_{55}, e_{76}

3. Proposed Algorithm

The performance of a data redistribution procedure is determined by four costs: index computational cost T_i, schedule computational cost T_s, message packing/unpacking cost T_p, and data transfer cost. The data transfer cost for each communication step consists of start-up cost T_{su} and transmission cost T_t. Let the unit transmission time τ denote the cost of transferring a message of unit length. In general, the message startup cost is directly proportional to the number of communication steps. The total number of communication steps is denoted by N. The total redistribution time equals T_i+T_s+ $\sum_{i=1}^{i=N}(T_p + T_{su} + m_i\tau)$, where m_i = Max$\{e_1, e_2, e_3, .., e_k\}$ and e_j represents the size of message scheduled in the i^{th} communication step for j = 1 to k. In irregular redistribution, messages of varying sizes are scheduled in the same communication step. Therefore, the largest size of message in the same communication step dominates the data transfer time required for this communication step.

The main idea of the Degree-Reduction-and-Coloring (DRC) algorithm is to diminish the degree of P_{max} repeatedly by scheduling the message in the first step of data redistribution process until *Max-degree* is equal to 2. The remaining messages are then scheduled into the communication steps by utilizing the concept of bipartite graph coloring mechanism. The details of the steps will be described in the following subsections.

3.1 Degree-Reduction Step

The goal in this step is to reduce *Max-degree* repeatedly in each iteration, until *Max-degree* is equal to 2. An example of 4-degree communication redistribution has taken as shown in Fig 2. In the first phase (phase-1) of degree-reduction step, the messages are sorted by the non-increasing order of $|E_{ij}|$, and the result is shown in Table 2. Then, DRC selects the messages into step1 of the schedule according to non-increasing order of message size except those messages causing the conflict. After phase-1, the *Max-degree* will be decreased by 1 (from 4 to 3). Fig 3(a) and Table 3(b) show this scenario. DRC repeat the procedure until the *Max-degree* reaches 2, which is depicted in Fig 4.

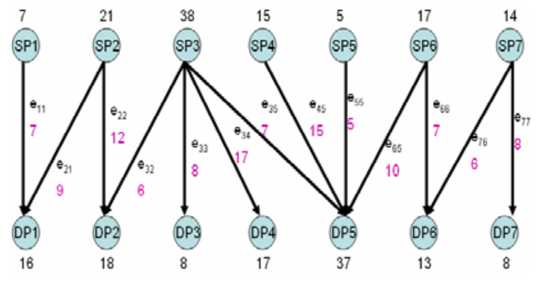
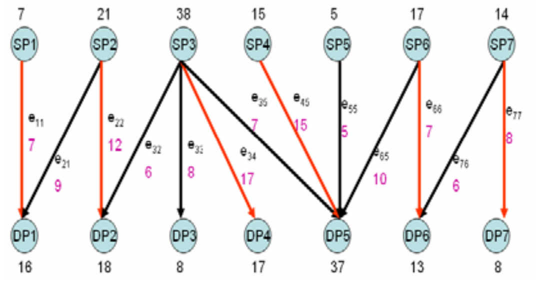


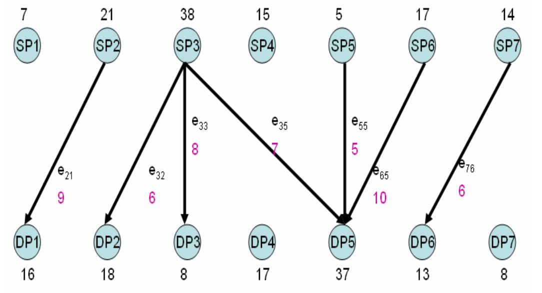
Figure 2. A data redistribution example with *Max-degree* = 4

Table 2. The messages are sorted by non-increasing order of message size

Msg no.	e_{34}	e_{45}	e_{22}	e_{65}	e_{21}	e_{33}	e_{77}	e_{11}	e_{35}	e_{66}	e_{32}	e_{76}	e_{55}
Msg size	17	15	12	10	9	8	8	7	7	7	6	6	5



(a)



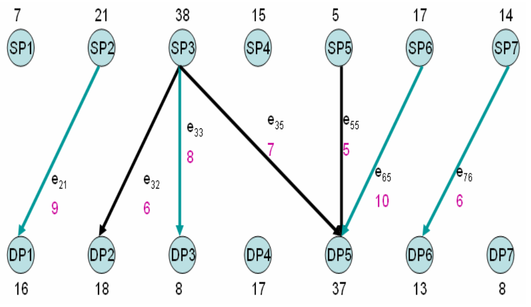
(b)

Figure 3. The messages communication (a) before phase-1 of the degree-reduction step; (b) after phase-1 of the degree-reduction step.

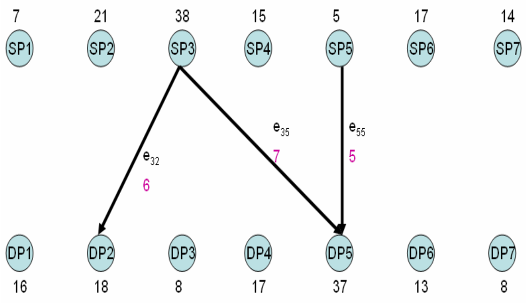
Table 4. The schedule after phase-1

Message no.	e_{34}	e_{45}	e_{22}	e_{65}	e_{21}	e_{33}	e_{77}	e_{11}	e_{35}	e_{66}	e_{32}	e_{76}	e_{55}
Message size	**17**	**15**	**12**	**10**	**9**	**8**	**8**	**7**	**7**	**7**	**6**	**6**	**5**

Schedule Table	
Step1:	**e_{34}, e_{45}, e_{22}, e_{77}, e_{11}, e_{66}**
Step2:	
Step3:	
Step4:	



(a)



(b)

Figure 4. The messages communication (a) before phase-2 of the degree-reduction step; (b) after phase-2 of the degree-reduction step.

Table 5. The schedule after the procedure of phase-2

Message no.	e_{34}	e_{45}	e_{22}	e_{65}	e_{21}	e_{33}	e_{77}	e_{11}	e_{35}	e_{66}	e_{32}	e_{76}	e_{55}
Message size	**17**	**15**	**12**	**10**	**9**	**8**	**8**	**7**	**7**	**7**	**6**	**6**	**5**

Schedule Table	
Step1:	**e_{34}, e_{45}, e_{22}, e_{77}, e_{11}, e_{66}**
Step2:	**e_{65}, e_{21}, e_{33}, e_{76}**
Step3:	
Step4:	

3.2 Message-Coloring Step

After completing the degree-reduction step, we can obtain a redistribution graph with *Max-degree* of 2 and the resulting redistribution graph is *2-edge colorable* [2], since it is a bipartite graph and its maximum degree is equal to 2. In the Message-Coloring Step, DRC schedules the left messages into the same step in a non-increasing order to accomplish an optimal scheduling unless a conflict occurs. Figure 5 shows the outcome of message-coloring and Table 6 shows the final schedule obtained from DRC algorithm.

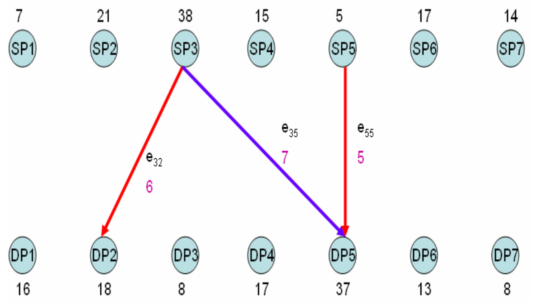


Figure 5. The outcome of redistribution graph after applying the message coloring mechanism

Table 6. The final schedule obtained from DRC

Msg no.	e_{34}	e_{45}	e_{22}	e_{65}	e_{21}	e_{33}	e_{77}	e_{11}	e_{35}	e_{66}	e_{32}	e_{76}	e_{55}
Msg size	17	15	12	10	9	8	8	7	7	7	6	6	5

Schedule Table	
Step1:	e_{34}, e_{45}, e_{22}, e_{77}, e_{11}, e_{66}
Step2:	e_{65}, e_{21}, e_{33}, e_{76}
Step3:	e_{35}
Step4:	e_{32}, e_{55}

The algorithm of the Degree-Reduction-Coloring is given as follows.

```
==========================================
Algorithm DRC
generating messages;
// generate messages from AS[Pi] to AD[Pi]
step = maximum degree;
sort_msgSize();
// sorting in decreasing order by message size
while (step > 2 )
   {
         choose_msg(step);
         // selecting message without conflict tuple, set
            into (maximal degree - step + 1) schedule
            step
         step--
   }  // degree-reduction iteration
while ( remaining_messages != null )
   {
         selecting_msg(maximal degree-1);
         // selecting message set into maximal degree-1
schedule step
         check_msg_continue_set();
         // check remaining message set
         coloring_maximal_msg(maximal degree);
         // color the maximal message with degree
            maximal degree -1 and the neighbor
            message with maximal degree
   }  // message coloring mechanism
end of   DRCM
==========================================
```

4. Performance Evaluation

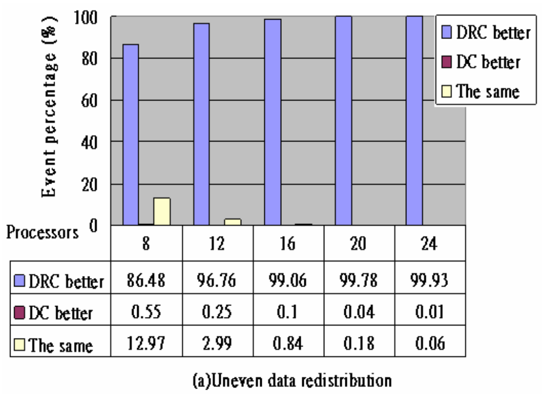
To evaluate the performance of the proposed methods, we have implemented the DRC along with the Divide-and-Conquer algorithm [23]. The performance simulation is discussed in two categories, even GEN_BLOCK and uneven GEN_BLOCK distributions. In even GEN_BLOCK distribution, each processor owns similar size of data. In contrast to even distribution, few processors might be allocated by grand volumes of data with uneven distribution. Since data elements could be centralized to some specific processors, it is also possible for those processors to have the maximum degree of communications.

The simulation program generates a set of random integer number and the size of message as *A*[*SPi*] and *A*[*DPi*]. Moreover, the total message size sending from SPi equals to the total size receiving to DPi keeping the balance between source processors and destination processors.

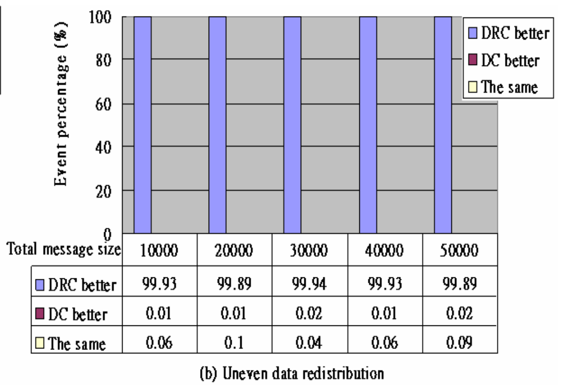
We assume that the data computation (communication) time in the simulation is represented by the transmission size $|E_{ij}|$. In the following figures, the percentage of events is plotted as a function of the message size and the number of processors. Also, in the figures, "DRC Better" represents the percentage of the number of events that the DRC algorithm has lower total computation (communication) time than the Divide-and-Conquer algorithm, while "DC Better" gives the reverse situation. If both algorithms have the same total computation (communication) time, "The Same Results" represents the number of that event.

In the uneven distribution, the size of message's up-bound is set to be B*1.7 and that of low-bound is set to be B*0.3, where B is equal to the sum of total transmission message size / total number of processors. In the even distribution, the size of message's up-bound is set to be B*1.3 and that of low-bound is set to be B*0.7. The total message-size is 10M.

Fig 6(a) and 6(b) show the simulation results of both the DRC and the Divide-and-Conquer algorithm with different number of processors and total message size. The number of processors is from 8 to 24. We can observe that the DRC algorithm has better performance in the uneven data redistribution compared with Divide-and-Conquer algorithm. Since the data is concentrated in the even case, from Fig 7(a) and 7(b), we can observe that DRC has better performance compared with the uneven case. In both even and uneven cases, DRC performs better than the Divide-and-Conquer algorithm.

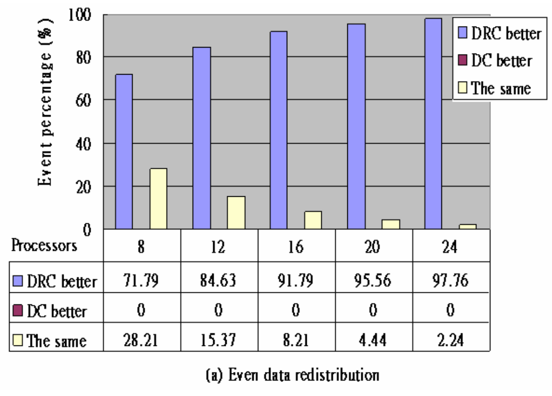


Processors	8	12	16	20	24
🟦 DRC better	86.48	96.76	99.06	99.78	99.93
🟥 DC better	0.55	0.25	0.1	0.04	0.01
🟨 The same	12.97	2.99	0.84	0.18	0.06

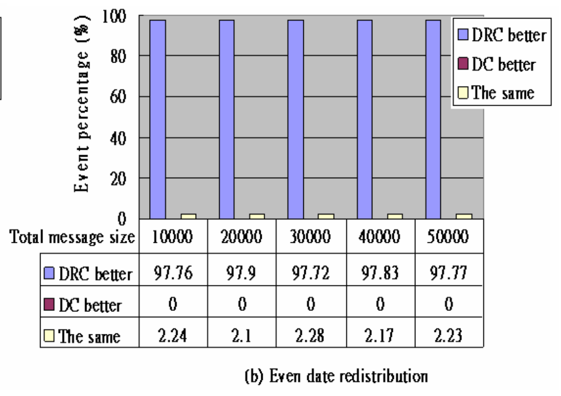


Total message size	10000	20000	30000	40000	50000
🟦 DRC better	99.93	99.89	99.94	99.93	99.89
🟥 DC better	0.01	0.01	0.02	0.01	0.02
🟨 The same	0.06	0.1	0.04	0.06	0.09

Figure 6. The events percentage of computing time is plotted (a) with different number of processors and (b) with different number of total message sizes in 24 processors, on the uneven data set.



Processors	8	12	16	20	24
🟦 DRC better	71.79	84.63	91.79	95.56	97.76
🟥 DC better	0	0	0	0	0
🟨 The same	28.21	15.37	8.21	4.44	2.24



Total message size	10000	20000	30000	40000	50000
🟦 DRC better	97.76	97.9	97.72	97.83	97.77
🟥 DC better	0	0	0	0	0
🟨 The same	2.24	2.1	2.28	2.17	2.23

Figure 7. The events percentage of computing time is plotted (a) with different number of processors and (b) with different number of total message sizes in 24 processors, on the even data set.

5.Conclusion

In this paper, we have presented a Degree-Reduction-Coloring (DRC) scheduling algorithm to efficiently perform HPF2 irregular array redistribution on a distributed memory multi-computer. The DRC algorithm is a simple method with low algorithmic complexity to perform GEN_BLOCK array redistribution. The DRC algorithm is an optimal algorithm in terms of minimal number of steps. In the same time, DRC algorithm is also a near optimal algorithm satisfying the condition of minimal message size of total steps. Effectiveness of the proposed methods not only avoids node contention, but also shortens the overall communication length.

For verifying the performance of our proposed algorithm, we have implemented DRC as well as the Divide-and-Conquer redistribution algorithm. The experimental results show improvement in communication costs and high practicability on different processor hierarchies. Also, the experimental results indicate that both of them have good performance on GEN_BLOCK redistribution. In many situations, DRC is better than the Divide-and-Conquer redistribution algorithm.

Reference

[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, 355 - 359 ,April 1999

[2] J.A. Bondy and U.S.R. Murty, Graph Theory with Applications, Macmillan, London, 1976.

[3] Frederic Desprez, Jack Dongarra and Antoine Petitet, "Scheduling Block-Cyclic Data redistribution," IEEE Trans. on PDS, vol. 9, no. 2, pp. 192-205, Feb. 1998.

[4] Minyi Guo, "Communication Generation for Irregular Codes," The Journal of Supercomputing, vol. 25, no. 3, pp. 199-214, 2003.

[5] 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.

[6] Minyi Guo, I. Nakata and Y. Yamashita, “Contention-Free Communication Scheduling for Array Redistribution,” Parallel Computing, vol. 26, no.8, pp. 1325-1343, 2000.

[7] Minyi Guo, I. Nakata and Y. Yamashita, “An Efficient Data Distribution Technique for Distributed Memory Parallel Computers,” Joint Symp. on Parallel Processing (JSPP'97), pp.189-196, 1997.

[8] 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.

[9] Edgar T. Kalns, and Lionel M. Ni, “Processor Mapping Technique Toward Efficient Data Redistribution,” IEEE Trans. on PDS, vol. 6, no. 12, pp. 1234-1247, December 1995.

[10] S. D. Kaushik, C. H. Huang, J. Ramanujam and P. Sadayappan, “Multiphase data redistribution: Modeling and evaluation,” International Parallel Processing Symposium (IPPS’95), pp. 441-445, 1995.

[11] Peizong Lee, Academia Sinica, and Zvi Meir Kedem, “Automatic Data and Computation Decomposition on Distributed Memory Parallel Computers,” ACM Transactions on Programming Languages and systems, Vol 24, No. 1, pp. 1-50, January 2002.

[12] 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, pp . 539-543, 2001.

[13] Y. W. Lim, Prashanth B. Bhat and Viktor and K. Prasanna, “Efficient Algorithms for Block-Cyclic Redistribution of Arrays,” Algorithmica, vol. 24, no. 3-4, pp. 298-330, 1999.

[14] C.-H Hsu, S.-W Bai, Y.-C Chung and C.-S Yang, “A Generalized Basic-Cycle Calculation Method for Efficient Array Redistribution,” IEEE Transactions on Parallel and Distributed Systems, vol. 11, no. 12, pp. 1201-1216, Dec. 2000.

[15] Ching-Hsien Hsu, Kun-Ming Yu, “An Optimal Processor Replacement Scheme for Efficient Communication of Runtime Data Realignment,” Parallel and Distributed and Processing and Applications, - Lecture Notes in Computer Science, Vol. 3358, pp. 268-273, 2004.

[16] 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.

[17] Antoine P. Petitet and Jack J. Dongarra, “Algorithmic Redistribution Methods for Block-Cyclic Decompositions,” IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 12, pp. 1201-1216, Dec. 1999

[18] Neungsoo Park, Viktor K. Prasanna and Cauligi S. Raghavendra, “Efficient Algorithms for Block-Cyclic Data redistribution Between Processor Sets,” IEEE Transactions on Parallel and Distributed Systems, vol. 10, No. 12, pp.1217-1240, Dec. 1999.

[19] .L. Prylli and B. Touranchean, “Fast runtime block cyclic data redistribution on multiprocessors,” Journal of Parallel and Distributed Computing, vol. 45, pp. 63-72, Aug. 1997.

[20] S. Ramaswamy, B. Simons, and P. Banerjee, “Optimization for Efficient Data redistribution on Distributed Memory Multicomputers,” Journal of Parallel and Distributed Computing, vol. 38, pp. 217-228, 1996.

[21] 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.

[22] Hui Wang, Minyi Guo and Wenxi Chen, “An Efficient Algorithm for Irregular Redistribution in Parallelizing Compilers,” Proceedings of 2003 International Symposium on Parallel and Distributed Processing with Applications, LNCS 2745, pp 76-87, 2003.

[23] 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, pp. 157-170, 2004.

[24] H.-G. Yook and Myung-Soon Park, “Scheduling GEN_BLOCK Array Redistribution,” The Journal of Supercomputing, vol. 22, no. 3, pp 251-267, 2002