Multi-Level Communication Scheduling for Irregular Data Redistribution

Shih-Chang Chen[1] and Ching-Hsien Hsu[2]
[1] *Institute of Engineering and Science*
[2] *Department of Computer Science and Information Engineering*
Chung Hua University, Hsinchu, Taiwan 300, R.O.C.
{scc, robert}*@grid.chu.edu.tw*

Abstract

Irregular array redistribution has been paid attention recently since it can distribute different size of data segment to heterogeneous processors according to their computational ability. It's also the reason why it has been kept an eye on load balance. High Performance Fortran Version 2 (HPF2) provides GEN_BLOCK distribution format which facilitates generalized block distributions. In this paper, we present a Multi-level scheduling method to minimize the communication cost in such operation. The main idea of the proposed technique is to specify two categories of messages and schedule separate steps. The performance evaluation is given in section 5. The results show the proposed method successfully adapts to such environment and minimize the length of schedules.

1. Introduction

Parallel computing systems such as PC clusters provide powerful computational ability to solve various kinds of scientific problems. A complex scientific problem may consist of several computation phases, each phase is responsible for different purposes and requires different data distribution scheme. In order to achieve a good load balance, it is necessary to redistribute data according to distribution scheme. Generally, data distribution can be classified into regular and irregular. The regular distribution employs `BLOCK`, `CYCLIC` and `BLOCK-CYCLIC(`*`c`*`)` to specify array decomposition. The irregular distribution uses user-defined function to specify unevenly array decomposition such as `GEN_BLOCK`.

PARAMETER (***old*** = /9, 8, 9, 16, 25, 33/)

```
!HPF$   PROCESSORS P(6)
        REAL A(100), new (6)
!HPF$   DISTRIBUTE A (GEN_BLOCK(old)) onto P
!HPF$   DYNAMIC
        new = /28, 17, 5, 10, 22, 18/
!HPF$   REDISTRIBUTE A (GEN_BLOCK(new))
```

Above is a code segment that High Performance Fortran version 2 (HPF2) provides the `GEN_BLOCK` distribution format to facilitate generalized block distribution. The ***old*** and ***new*** are defined as the distribution schemes of the source and destination processors (nodes), respectively. DISTRIBUTE directive decomposes array **A** onto 6 processors according to the ***old*** scheme in source phase. When the program goes to next phase, REDISTRIBUTE directive realign array **A** according to ***new*** scheme.

Two-Phase Degree Reduction algorithm (*TPDR*) [3] is one of the efficient scheduling algorithms that provide a schedule of data redistribution for better load balancing. After generating messages according to ***old*** and ***new*** schemes, messages are scheduled by *TPDR* in minimum time steps and with cost as lower as possible. *Local message reduction* (*LMR*) [3], which is an optimization technique, shows *TPDR* can still be improved by given appropriate cost of messages. Authors of *LMR* find the cost function of messages could be redefined to present more practical cost for each message. The experiments show the improvement of *LMR* on *TPDR* by improving the cost function. In [3], messages can be classified into two categories: one is to be transmitted from one node to another (remote data access); the other one is to be transmitted in local memory (local data access, which is happened in the same node). *LMR* defines remote access time (*RAT*) for remote data access and local access time (*LAT*) for local data access. Both of *RAT* and *LAT* represent the transmitting rate while

transmitting data with the fixed size. The ratio of remote to local access time (*RLR*) is defined as *RAT* divided by *LAT*. The performance evaluation in [3] shows *TPDR* with *LMR* can give better schedules in most cases.

Schedules illustrate steps for messages to be transmitted in proper time. The cost of each step is represented by the largest cost of messages. The total cost of a schedule is the summation of costs of each step. A phenomenon is observed that most messages of local data access do not dominate the cost of each step because they are not large enough in practical. Since a node can send and receive only one message in the same time step [3], the arranged position of each message becomes important. Generally, messages of local data access do not dominate the cost of schedule steps, the position should not be occupied if a better schedule could be given. For decreasing costs of data redistribution, messages of local data access are picked out and to be arranged in a specific time step.

We have implemented the proposed optimization technique to improve *TPDR* scheduling algorithm. The performance evaluation shows that the proposed optimization technique provides performance improvement in most `GEN_BLOCK` array redistribution cases.

The rest of this paper is organized as follows. Section 2 presents a brief survey of related work. Definitions and an example of schedule are given in section 3. In section 4, the *multi-level communication scheduling method* is introduced to reduce the cost of data redistribution. We also provide an example to demonstrate the improvement. In section 5, the simulation results and performance analysis are given to weigh the pros and cons. Finally, the conclusions are presented in section 6.

2. Related Work

Many methods have been developed for performing array redistribution. Researches were proposed for regular and irregular problems [7] in multi-computer compiler techniques or runtime support techniques. A brief survey of related work is given bellow.

Researches for regular array redistribution are classified into three categories: the communication sets identification; message packing and unpacking techniques; communication optimizations. Researches for communications sets identification techniques include the *PITFALLS* [15] and the *ScaLAPACK* [14] methods for index sets generation. *CFS* and *ED* are proposed for sparse array distribution by Lin and Chung [13]. Researches for message packing and unpacking techniques include ECC method [1] which was proposed for a processor to pack/unpack array elements efficiently. Researches for communication optimizations include the processor mapping techniques [6, 8] for minimizing data transmission overheads. Lim *et al*. [20] proposed the multiphase redistribution strategy to reduce message startup cost. The communication scheduling approaches were proposed to avoid node contention in [4, 12].

Researches for irregular array redistribution are focused on message generation and communication efficiency. Researches for message generation include a symbolic analysis method which was proposed by Guo *et al*. [9]. Researches for communication efficiency include improving the relocation scheduling algorithm [17] by combining the divide-and-conquer and relocation algorithms. HCS and HRS were proposed by Chang *et al*. [2] to improve the solutions of data access on data grids. Chen *et al*. [3] proposed *TPDR* scheduling algorithm to reduce communication cost and *LMR* to improve the result of *TPDR*.

3. Preliminary

To simplify the presentation, notations and terminology used in this paper are defined as follows:

Definition 1: Given an irregular `GEN_BLOCK` redistribution on a 1-D array $A[1{:}N]$ over P processors, $\boldsymbol{SP_i}$ denotes the *source processors* of array elements $A[1{:}N]$; $\boldsymbol{DP_i}$ denotes the *destination processors* of array elements $A[1{:}N]$, where $0 \le i \le P-1$.

Definition 2: Given a bipartite graph $G = (V, E)$ to represent the communication patterns of an `GEN_BLOCK` array redistribution on $A[1{:}N]$ over $\boldsymbol{P}$ processors, vertices of $\boldsymbol{G}$ are used to represent the source and destination processors. Figure 1 gives an example of bipartite graph representing communication patterns between four source and destination processors with seven messages to be communicated.

Definition 3: Given a directed bipartite graph $G = (V, E)$, $Degree_{max}$ denotes the maximal in-degree (or out-degree) of vertices in G. For example, the bipartite graph shown in Figure 2 is with $Degree_{max} = 3$, which is equal to the out-degree of the white vertex.

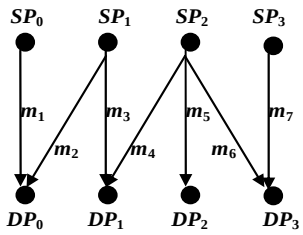


Figure 1: A bipartite graph representing communication patterns.

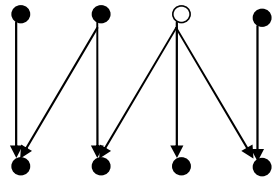
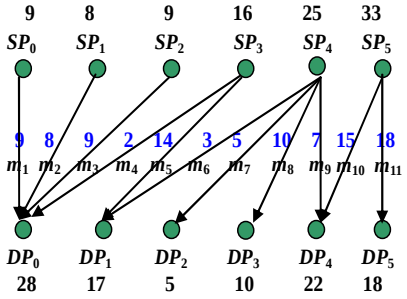


Figure 2: A bipartite graph with $Degree_{max}$=3.

The communication time depends on the length[1] of each communication step. The length of a step is dominated by the largest message. In general, the transmission cost is directly proportional to the length of total steps[2] which determines the data transmission overheads. To avoid node contention in a time step, three scheduling policies are described as follows:

- An *SP* sends a message to only one *DP* and can not send another message to another *DP* in the same communication step.
- *DP* receives a message from only one *SP* and can not receive another message from another *SP* in the same communication step.
- A processor can send a message and receive another message simultaneously.

Given ***old*** and ***new*** `GEN_BLOCK` distribution schemes, the communication patterns are illustrated in Figure 3 (a). Above $SP_{0\sim5}$ are numbers representing ***old*** scheme, below $DP_{0\sim5}$ are numbers representing ***new*** scheme. Arrows between *SP* and *DP* represent messages while redistributing data from source processors to destination processors. The $m_{1\sim11}$ represent the index of messages and the numbers above the index are the theoretical cost which are the most important reference for algorithms to schedule messages. Since a processor can send and receive only one message in one step, there must be four time steps at least due to $Degree_{max}$ is four. Figure 3 (b) gives a schedule of Figure 3 (a). In Figure 3 (b), there are four steps and 11 messages in relative row of steps represent the data to be transmitted in relative time steps. The cost of each step is dominated by the messages with largest cost. The cost of step 1, 2, 3 and 4 are 14, 15, 18 and 10, respectively. The length of this simple schedule is 57, which is the summation of four costs.



(a)

A simple schedule		
No. of step	No. of message	Cost of step
Step 1	m_1(9), m_5(14), m_9(7)	14
Step 2	m_2(8), m_6(3), m_{10}(15)	15
Step 3	m_3(9), m_7(5), m_{11}(18)	18
Step 4	m_4(2), m_8(10)	10
Total cost		57

(b)

Figure 3: (a) A bipartite graph representing communications between *SP* and *DP*. (b) A simple schedule of ***old*** and ***new*** `GEN_BLOCK` distribution schemes.

4. The Proposed Method

In Figure 3 (b), the messages that dominate the cost of each step can be found easily, which are m_5, m_{10}, m_{11} and m_8 for step 1~4, respectively. If the *RLR* is considered for more practical cost of messages, the message which dominates the cost of each step may be changed. Figure 4 shows the effect of *RLR* on cost of local data access. Costs of messages such as m_1, m_9 and m_{11} are divided by *RLR* and are reduced to the values in relative round brackets while the *RLR* is assumed 8. The cost of each step in column of "Effect of *RLR*" in Figure 4 reflects more practical transmission cost instead of those in column of "Cost of step" in Figure 3 (b). The dominator of step 3 is

[1] Length of a scheduling step is equal to the maximal data size of messages in this scheduling step.
[2] Length of a schedule is the sum of length of all scheduling steps.

changed from m_{11} to m_3 since the cost of m_{11} is not large as expected in Figure 4. As mentioned in section 1, messages of local data access do not dominate the cost of steps in many cases. Therefore, the key position should not be occupied by them if a better schedule could be given by scheduling algorithms.

A simple schedule		
No. of step	No. of message	Effect of RLR
Step 1	m_1(1.25), m_5(14), m_9(0.875)	14
Step 2	m_2(8), m_6(3), m_{10}(15)	15
Step 3	m_3(9), m_7(5), m_{11}(2.25)	9
Step 4	m_4(2), m_8(10)	10
Total cost		48

Figure 4: Column of effect of *LMR* shows the changes of theoretical cost of m_1, m_9 and m_{11} and the dominator of step two is changed.

The *LMR* describes the importance of distinguishing both data accesses from each other. Above example in Figure 4 shows that local data access plays a dominator at first, but has small effect in step 3. Such situations can influence scheduling algorithms not to arrange messages well in the key positions, like m_{11} which could be scheduled in other steps instead. Since the position of messages can influence others, local data access should be processed additionally for lower costs of `GEN_BLOCK` data redistribution.

Definition 4: Given a directed bipartite graph $G = (V, E)$, SP_i and $DP_j \in V$, where $0 \le i, j \le P-1$. The $m_k \in E$, where $P \le k \le 2P-1$. While $i = j$, m_k represents the messages of local data access, otherwise m_k represents remote data access.

The proposed *multi-level communication scheduling method* (*MLC*) schedules messages of local data access and remote data access in separate steps for `GEN_BLOCK` data redistribution. The first level processes the arrangement of local data access, the second level is responsible for the arrangement of remote data access. *MLC* is performed by following processes:

1. Find out messages m_k which are transmitted by SP_i and DP_j, where $i = j$.
2. Divide the cost of above found m_k using *RLR*.
3. To give a schedule of first level.
4. Find $Degree_{max}$ and start a new step.
5. If $Degree_{max} > 2$, sort vertices with $Degree_{max}$ in decreasing order according to data size and give $\{v_1, v_2, v_3, \ldots\} \in V$. Select a m_k with relative minimal cost from each vertex in $\{v_1, v_2, v_3, \ldots\}$ and then arrange the message in step.
6. Arrange more m_k, whose cost is smaller than the length and has not been arranged in any step, if possible.
7. $Degree_{max} = Degree_{max} - 1$. If $Degree_{max} > 2$, repeat processes 4~7.
8. Use coloring theory to arrange m_k in two steps.
9. Exchange position of any m_k in the two-step schedule if lower cost is achievable.
10. To give a schedule of second level.

Figure 5 is a schedule given by *MLC*. The values in last column also reflect more practical transmission costs. The result shows that removing local data access from m_k can help scheduler handle key positions better and reduce total length of a schedule. Following above processes of *MLC*, the messages of local data access are selected first. In the example shown in Figure 3 (a), the candidates are m_1, m_9 and m_{11}, and are arranged in step 4 in Figure 5. After that, all of m_k are messages of remote data access. The processes then go to 4~7 and arrange m_4 and m_7 in step 3. Using processes 8 and 9, *MLC* arranges messages in two steps at last. The length of this schedule is 31.25 and is smaller than the length in Figure 4.

A schedule of MLC		
No. of step	No. of message	Effect of RLR
Step 1	m_2(8), m_5(14), m_8(10), m_{10}(15)	15
Step 2	m_3(9), m_6(3)	9
Step 3	m_4(2), m_7(5)	5
Step 4	m_1(1.25), m_9(0.875), m_{11}(2.25)	2.25
Total cost		31.25

Figure 5: A schedule of *MLC*.

5. Performance Evaluation

To evaluate the performance of proposed methods, *MLC* were implemented along with *TPDR*. A huge amount of cases were provided to evaluate *MLC* and *TPDR*. 1,000 cases were provided for each comparison, and 12,000 were provided in total. Array size in each `GEN_BLOCK` distribution scheme is 10,000. The numbers of nodes, *P*, are 8, 32 and 128 in three simulation comparisons, respectively. The ***Avg*** represents the value of array size divided by *P*. Four sets of lower bounds and upper bounds were provided to define the size range of each node as shown in Figure 6.

Size range of each node		
Symbol of ranges	Lower bound	Upper bound
α	0.5* *Avg*	2 * *Avg*
β	1	2 * *Avg*
γ	1	4 * *Avg*
δ	1	8 * *Avg*

Figure 6: Lower bounds and upper bounds of four size ranges for each node.

Figures 7~9 give the results of comparing *MLC* and *TPDR* with various number of nodes and different size range of nodes. In Figure 7, the *MLC* and *TPDR* were compared on 8 nodes with α, β, γ and δ. The plots of *MLC* better represents *MLC* performs better than *TPDR*; the plots of *TPDR* better represents *TPDR* performs better than *MLC*; the plots of the same represent the costs of schedules given by both *MLC* and *TPDR* are the same. Results in Figure 7 show that *MLC* outperforms *TPDR* in most cases with α, β, γ and δ on 8 nodes. Comparing the results of *MLC*, the *MLC* better with α is found much less than the *MLC* better with β due to lots of tie cases. The size range of α is smaller than β and results in little variation. Since the variation is not large, the choices of dominators become few and result of the same is relative higher. While comparing the results with β, γ and δ, both numbers of *MLC* better and *TPDR* better are increased and number of the same is decreased due to larger variation.

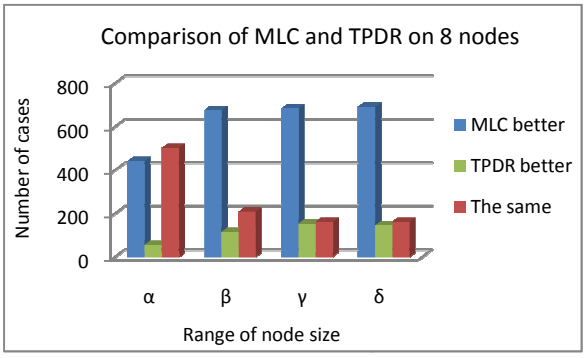


Figure 7: The results of comparisons on 8 nodes over different range of node size. *Avg* is 1250.

With more number of nodes and smaller *Avg*, the tie cases are almost disappears on 32 nodes in Figure 8. In the results of comparison, the *MLC* performs better than *TPDR* in about 65% to 75% cases. Due to the increased number of nodes, the importance of local data access drops a little. The number of *TPDR* better is increased due to more combinations of schedules to be chosen with growing number of remote data access messages, but *MLC* has advantage to perform better in most cases.

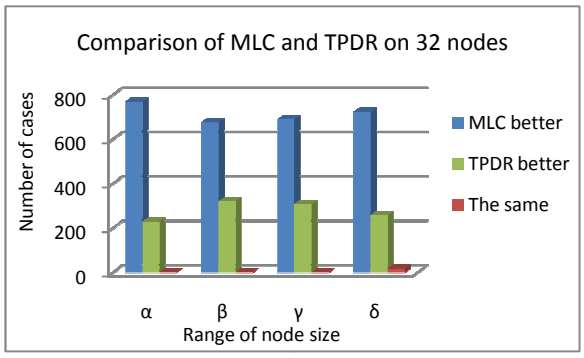


Figure 8: The results of comparisons on 32 nodes over different range of node size. *Avg* is 312.5.

While the number of nodes increases to 128, the advantage of *TPDR* method emerges in Figure 9. With 128 nodes and smaller *Avg*, the importance of remote data access increases substantially and the advantage of *MLC* method disappears. With small upper bound of α, β and γ, the variation is relative low and compresses the chances for *MLC* method to find better schedules. Although *TPDR* performs good with α, β and γ, it can not performs as well as *MLC* with δ where the variation is higher and is good for local data access oriented scheduling methods.

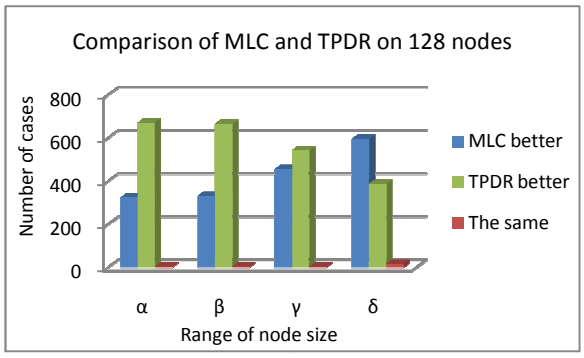


Figure 9: The results of comparisons on 128 nodes over different range of node size. *Avg* is 78.125.

6. Conclusions

In this paper, we have presented multi-level communication scheduling method to minimize the communication cost in irregular data redistribution. The proposed method adapts different data transmitting rate by modifying the communication cost using *RLR* and scheduling two categories of messages in separate steps. The performance analyses show *MLC* performs better in most cases and adapts to high variation of data arrangement in data redistribution.

REFERENCES

[1] Sheng-Wen Bai and Chu-Sing Yang, "Essential Cycle Calculation Method for Irregular Array Redistribution," *IEICE Transactions on Information and Systems*, Vol. E89-D, No. 2, pp. 789-797, Feb. 2006.

[2] Ruay-Shiung Chang, Jih-Sheng Chang and Shin-Yi Lin, "Job scheduling and data replication on data grids," *Future Generation Computer Systems*, Vol. 23, No. 7, pp. 846-860, Jul. 2007.

[3] Shih-Chang Chen and Ching-Hsien Hsu, "ISO: Comprehensive Techniques Toward Efficient GEN_BLOCK Redistribution with Multidimensional Arrays," *Parallel Computing Technologies - Lecture Notes in Computer Science*, Vol 4671, pp. 507-515, Springer-Verlag, Sep. 2007. (PaCT'07)

[4] Frederic Desprez, Jack Dongarra and Antoine Petitet, "Scheduling Block-Cyclic Data redistribution," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 2, pp. 192-205, Feb. 1998.

[5] Ching-Hsien Hsu, Sheng-Wen Bai, Yeh-Ching Chung and Chu-Sing 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.

[6] Jih-Woei Huang and Chih-Ping Chu, "A flexible processor mapping technique toward data localization for block-cyclic data redistribution," *The Journal of Supercomputing*, Vol. 45, No. 2, pp. 151-172, Aug. 2008.

[7] Minyi Guo, "Communication Generation for Irregular Codes," *The Journal of Supercomputing*, Vol. 25, No. 3, pp. 199-214, 2003.

[8] Minyi Guo and Yi Pan, "Improving communication scheduling for array redistribution," *Journal of Parallel and Distributed Computing*, Vol. 65, No. 5, pp. 553-563, May 2005.

[9] Minyi Guo, Yi Pan and Zhen Liu, "Symbolic Communication Set Generation for Irregular Parallel Applications," *The Journal of Supercomputing*, Vol. 25, No. 3, pp. 199-214, Jul. 2003.

[10] Amit Karwande, Xin Yuan and David K. Lowenthal, "An MPI prototype for compiled communication on ethernet switched clusters," *Journal of Parallel and Distributed Computing*, Vol. 65, No. 10, pp. 1123-1133, Oct. 2005.

[11] S. D. Kaushik, Chua-Huang Huang, J. Ramanujam and P. Sadayappan, "Multi-phase array redistribution: modeling and evaluation," *Proceeding of IEEE International Parallel Processing Symposium* (*IPPS*'95), pp. 441-445, Apr. 1995.

[12] Young Won Lim, Prashanth B. Bhat and Viktor K. Prasanna, "Efficient Algorithms for Block-Cyclic Redistribution of Arrays," *Algorithmica*, Vol. 24, No. 3-4, pp. 298-330, Jul. 1999.

[13] Chun-Yuan Lin and Yeh-Ching Chung, "Data distribution schemes of sparse arrays on distributed memory multicomputers," *The Journal of Supercomputing*, Vol. 41, No. 1, pp. 63-87, Jul. 2007.

[14] Loic Prylli and Bernard Tourancheau, "Fast runtime block cyclic data redistribution on multiprocessors," *Journal of Parallel and Distributed Computing*, Vol. 45, No. 1, pp. 63-72, Aug. 1997.

[15] Shankar Ramaswamy, Barbara Simons, and Prithviraj Banerjee, "Optimizations for Efficient Array Redistribution on Distributed Memory Multicomputers," *Journal of Parallel and Distributed Computing*, Vol. 38, No. 2, pp. 217-228, Nov. 1996.

[16] Rajesh Sudarsan and Calvin J. Ribbens, "Efficient Multidimensional Data Redistribution for Resizable Parallel Computations," *Fifth International Symposium on Parallel and Distributed Processing and Applications*, Vol. 4742, pp. 182-194, 2007.

[17] Hui Wang, Minyi Guo and Daming Wei, "Message Scheduling for Irregular Data Redistribution in Parallelizing Compilers," *IEICE Transactions on Information and Sysmtes*, Vol. E89-D, No. 2, pp. 418-424, Feb. 2006.