Efficient Algorithm for Symmetric Matrix Data Redistribution

Min-Hao Chen, Ching-Hsien Hsu and Shih-Chang Chen
Department of Computer Science and Information Engineering
Chung Hua University, Hsinchu, Taiwan 300, ROC
Tel: 886-3-5186410 Fax: 886-3-5186416
Email: chh@chu.edu.tw

Abstract*-In this paper, we present an efficient algorithm for* `BLOCK-CYCLIC` *data redistribution to minimize communication costs on symmetric matrix. The main idea of the algorithm is to explore all identical elements in local space. By realigning those elements into corresponding destination data layout, inter-processor data transmission overheads can be reduced. Given a* `BLOCK-CYCLIC (x, *)` *to* `BLOCK-CYCLIC (y, *)` *symmetrical matrix data re-decomposition over P processors, the theoretical analysis shows 1/P data packing/unpacking and communication costs can be reduced. The experimental results reflect small extra computational overheads of the Symmetric Redistribution Algorithm (SRA) will be incurred. However, the SRA technique can still provides superior overall performance.*

Keywords: data redistribution, coordinates transformation, symmetric matrix, communication optimization

1 Introduction

The data parallel programming model is more and more important for programming distributed memory multi-computers. Suitable data decomposition in order to efficiently execute a data parallel program on a distributed memory multi-computer is important. A good data distribution can let the computational load balance, increase data locality, and minimize interprocessor communication overloads.

Many data parallel programming languages such as High Performance Fortran (HPF), Fortran D, Vienna Fortran, and High Performance C (HPC) provide compiler directives for programmers to design a data distribution module. Regular distributions provided by these languages, on general, one-dimensional array distribution has three types; `BLOCK`, `CYCLIC`, and `BLOCK-CYCLIC(`*c*`)`. In a similar way, multi-dimensional array distributions also have those types. However, as a result of multi-dimension array distribution has column-wise and row-wise orientation, the structures become more complicated.

In some scientific applications, it is possible for an algorithm to process more than one data distribution layout during different computational phases in the same program. For example, `BLOCK-CYCLIC (*, CYCLIC)` and `BLOCK-CYCLIC (*, BLOCK)` are good distributions for LU decomposition problem in parallel program, the `BLOCK-CYCLIC (CYCLIC, *)` distribution is suitable for matrix multiplication. However, the LU decomposition and matrix multiplication are usually collaborated for the same computation. Therefore, data re-decomposition is required to accomplish data locality during runtime.

In general, data redistribution costs consist of computation costs and communication costs. The computation costs include indexing costs and message packing/unpacking costs. The indexing (communication sets generation) refers to calculate the destination/source processors of local elements for exchanging data with other processors. The packing/unpacking costs refer as the time complexity to construct data buffers for message passing. The communication cost is the time to send/receive messages with remote processors.

In this paper, we propose an efficient algorithm, the Symmetric Redistribution Algorithm (*SRA*), for symmetric matrix data redistribution. The main idea of the *SRA* method is to reuse all identical elements in local space to minimize runtime data transmission overheads. Given a `BLOCK-CYCLIC (x, *)` to `BLOCK-CYCLIC (y, *)` symmetrical matrix data re-decomposition over *P* processors, it is proved that 1/*P* packing/unpacking and data transmission costs can be reduced.

The rest of this paper is organized as follows. In Section 2, a brief discussion of related work will be presented. In section 3, we will introduce notations and terminology used in this paper and then present our algorithm for redistribute a symmetric matrix with `BLOCK-CYCLIC (x,*)` to `BLOCK-CYCLIC (y,*)`. In Section 4, a theoretical model for performance analysis and simulation test will be given. Section 5 briefly concludes this paper.

2 Related works

Data distribution and redistribution problems have been studied for many years. Some work has been pay attention to communication set generation, while some specialize on communication optimization. For example, the communication scheduling can avoid node contention, the processor mapping technology can increase data hits and minimize the amount of data

exchange for reducing communication overheads, and the multi-phase redistribution strategy can reduce message startup cost. We briefly explain these researches in both multi-computer compiler techniques and runtime support techniques in the following.

For those studies, Walker *et al.* [10] used the standardized message passing interface, MPI, to express the redistribution operations. They implemented the BLOCK-CYCLIC array redistribution algorithms in both synchronous and asynchronous schemes. Since the excessive synchronization overheads incurred from the synchronous scheme, they also presented the random and optimal scheduling algorithms for BLOCK-CYCLIC array redistribution. The experimental results showed that the performance of the synchronous method with optimal scheduling algorithm was comparable to that of the asynchronous method.

For communication optimization techniques, the communication scheduling problem has been proved step optimal on Jack Dongarra's study [2]. GGP [4] is a novel inter-cluster scheduling algorithm for redistributing data. The DAP optimization of redistribution is to avoid unnecessary remapping by eliminating partially dead and partially redundant distribution changes [5]. The Processor Mapping Technique [7] solves block to cyclic*(x)* problem for minimizing data transmission costs. In [6, 9], techniques for overlapping communication and computation were addressed on various cases.

For indexing techniques, the PITFALLS [8] developed efficient algorithm to perform array redistribution between two disjoint processor sets in the source and destination distribution; the Basic-Cycle Calculation (BCC) method [1] is an example to generate communication sets using pattern attribute in each section. The Generalized Basic-Cycle Calculation (GBCC) method [3] extends BCC to solve array redistribution with different source and destination processor sets.

3. Data Redistribution on Symmetric Matrix

3.1 Preliminaries

Generally, data redistribution can be performed in two phases, the sending phase and the receiving phase. The sending phase is proceeded with the original distribution and source processor set P that composed by numbers of processors p_i, where $i = 0, \ldots, P-1$. In the sending phase, p_i has to determine all data sets which need to be sent to corresponding destination processors, packs those data sets into messages, then sends messages to their destination processors. The receiving phase is to accomplish the target distribution which is over destination processor set Q that composed by numbers of processor q_j, where $j = 0, \ldots, Q-1$. In the receiving phase, q_i has to determine all data sets which need to be received from their source processors, unpacks elements in messages to their corresponding local array positions. Without loose of generality, we assume that $P=Q$ and $p_i = q_j$. Therefore, each processor has to calculate four communication sets, i.e., the Destination Processor Set (DPS [q_i]), the Send Data Set (SDS [$P_{i\to j}$]), the Source Processor Set (SPS [P_i]), and the Receive Data Set (RDS [$P_{i\leftarrow j}$]).

To simplify the illustration of this paper, we use BC(x_1, y_1)→BC(x_2, y_2) to represent BLOCK-CYCLIC(x_1, y_1) to BLOCK-CYCLIC(x_2, y_2) redistribution. Notations and terminologies used in this article are defined as follows.

Definition 1 : Given a BC(x_1, y_1)→BC(x_2, y_2) data redistribution on matrix $M_{n\times n}$ over P processors, the *source local matrix* of processor P_i is denoted by SLM_i, the *destination local matrix* of processor P_j is denoted by DLM_j, where $0 \le i, j < P$.

Definition 2 : Given a BC(x_1, y_1)→BC(x_2, y_2) data redistribution on matrix $M_{n\times n}$ over P processors, the *source processor* of an element in $M_{n\times n}$ or DLM_j is defined as the processor that owns the element in the source distribution. The destination processor of an element in $M_{n\times n}$ or SLM_i is defined as the processor that owns the element in the destination distribution, where $0 \le i, j < P$.

Definition 3 : Given a matrix $M_{n\times n}$, an *Adjacent Block* denoted by $\tilde{M}^{(x,y)}_{(x',y')}$ is defined as the set of elements clustered in a rectangular region with an upper-left coordinate (x, y) and a lower-right coordinate (x', y'), where $0 \le x \le x' \le n-1$, $0 \le y \le y' \le n-1$. E.g. $\tilde{M}^{(0,0)}_{(2,2)} = \{A[0, 0{:}2], A[1, 0{:}2], A[2, 0{:}2]\}$.

3.2 Cost Model

Given a $BC(x_1, y_1) \to BC(x_2, y_2)$ redistribution on a two-dimensional matrix M [N: N] over P processors, the time for an algorithm to perform the redistribution, in general, can be modeled as follows:

$$T_{cost} = T_{comp} + T_{comm} \qquad (1)$$

where T_{comp} is the time for communication set generation that employ an algorithm to compute source/destination processors of local matrix elements and the time to pack elements in source local matrix that have the same destination processors to a sending buffer, and unpack elements in messages that received from source processors to their corresponding destination local matrix positions; T_{comm} is the time for an algorithm to send and receive data among every processors. We said that T_{comp} and T_{comm} are the computation and communication time of a data redistribution algorithm, respectively.

For the Symmetric Redistribution Algorithm (*SRA*), according to the above description, we can have the following formulation,

$$T_{cost}(SRA) = T_{comp}(SRA) + S \times T_s + D \times T_d \quad (2)$$

Where $T_{comp}(SRA)$ is the computation time of *SRA* to perform $BC(x,*) \rightarrow BC(y,*)$ data redistribution; A detail analysis of this part will be discussed in next section; S is the maximum number of processors that a source processor needs to send data to, i.e., max{|DPS $[P_i]$| | $0 \le i < P$ }; D is the maximum total message size of a source processor P_i; T_s is the startup time of the interconnection network of a parallel machine; and T_d is the data transmission time of the interconnection network of a parallel machine.

3.3 The Proposed Algorithm

Communication time is an important factor to the overall performance of runtime data redistribution. Several optimizations have been proposed to minimize the communication overheads in various ways. For example, the multi-phase redistribution method was used to reduce message startup cost, the communication scheduling approach was used to avoid node contention, and the processor mapping technique was used to minimize data transmission cost. In this paper, we attempt to discuss the method that can reduce data transmission cost in redistributing symmetric matrices. Utilize the symmetrical attribute of matrix elements, the desired destination data layout can be achieved via reconfigured parts of elements in local space without receiving remote data from other processors. Consequently, the data transmission overheads during runtime could be reduced.

3.3.1 Motivating Example

Figure 1 shows an example of $BC(BLOCK,*) \rightarrow BC(2,*)$ on $M_{12\times12}$ over three processors. Matrices figured in the upper and the lower diagrams represent the layouts of the source and the destination distributions, respectively.

According to the preliminary explication, the communication sets of DPS and SPS for all processors can be listed as follows,

DPS [P_0] = { P_0, P_1 }	SPS [P_0] = { P_0, P_1 }
DPS [P_1] = { P_0, P_2 }	SPS [P_1]= { P_0, P_2 }
DPS [P_2] = { P_1, P_2 }	SPS [P_2] = { P_1, P_2 }

For the communication sets of RDS, we select P_1 as an example to illustrate the generation of communication messages. Considering source local matrix of P_1, i.e., SLM_1 and the Receive Data Sets of P_1 as listed follows, which can be determined by mathematical close forms that described in [3]

RDS [$P_{1\leftarrow0}$] = {1b, 2b, 0, 3c, **3d, 3e, 3f, 3g**, 3h, 3i, 3j, 3k 1c, 2c, 3c, 0, **4d, 4e, 4f, 4g**, 4h, 4i, 4j, 4k},

RDS [$P_{1\leftarrow2}$] = {1h, 2h, 3h, 4h, **5h, 6h, 7h, 8h**, 0, 9i, 9j, 9k 1i, 2i, 3i, 4i, **5i, 6i, 7i, 8i**, 9i, 0, 0j, 0k },

SLM_1 = {1d, 2d, **3d, 4d**, 0, 5e, 5f, 5g, **5h, 5i**, 5j, 5k, 1e, 2e, **3e, 4e**, 5e, 0, 6f, 6g, **6h, 6i**, 6j, 6k, 1f, 2f, **3f, 4f**, 5f, 6f, 0, 7g, **7h, 7i**, 7j, 7k, 1g, 2g, **3g, 4g**, 5g, 6g, 7g, 0, **8h, 8i**, 8j, 8k}

We find that RDS [$P_{1\leftarrow0}$] and RDS [$P_{1\leftarrow2}$] have some data which are the same with SLM_1, the boldface fonts of above example show this situation.

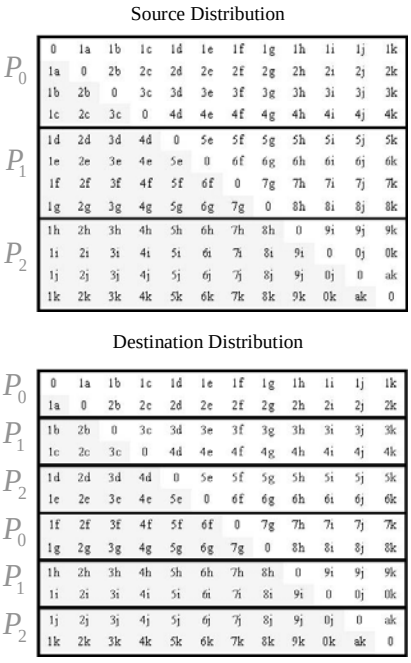


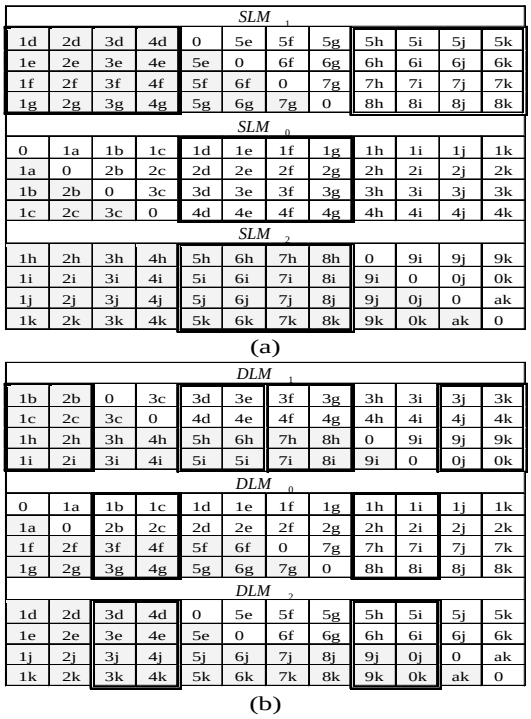
Figure 1: BC(BLOCK, *) to BC (2, *) data re-decomposition on $M_{12\times12}$ over 3 processors.

3.3.2 `BC(x, *)` to `BC(y, *)` data redistribution on symmetric matrix

To illustrate the symmetrical matrix redistribution method, we use figure 2 as the follow-up example of the motivating case. The first observation is made to the source local matrix of P_0 in source distribution as shown in figure 2(a). Adjacent blocks $\tilde{SLM}_{1(3,3)}^{(0,0)}$ and $\tilde{SLM}_{1(3,11)}^{(0,8)}$ are identical to the transposition of $\tilde{SLM}_{0(3,7)}^{(0,4)}$ and $\tilde{SLM}_{2(3,7)}^{(0,4)}$, respectively. This relationship leads the following lemma.

Lemma 1 : Given a $BC(x,*) \rightarrow BC(y,*)$ redistribution on an $n \times n$ symmetric matrix over P processors, for any source processor p sends data to destination processor q, in the source distribution, $\bigcup \tilde{SLM}_{p_{(s,t)}}^{(0,g)} = (\bigcup \tilde{SLM}_{q_{(s,u)}}^{(0,r)})^T$, where $g = q \times x/P$, $r = p \times x/P$, $s = n/P - 1$, $t = g + x - 1$, $u = g + x - 1$, $0 \le p, q < P$.

Proof: Due to page limitation, we omit the proof of this lemma in this version. ■



SLM_1											
1d	2d	3d	4d	0	5e	5f	5g	5h	5i	5j	5k
1e	2e	3e	4e	5e	0	6f	6g	6h	6i	6j	6k
1f	2f	3f	4f	5f	6f	0	7g	7h	7i	7j	7k
1g	2g	3g	4g	5g	6g	7g	0	8h	8i	8j	8k
SLM_0											
0	1a	1b	1c	1d	1e	1f	1g	1h	1i	1j	1k
1a	0	2b	2c	2d	2e	2f	2g	2h	2i	2j	2k
1b	2b	0	3c	3d	3e	3f	3g	3h	3i	3j	3k
1c	2c	3c	0	4d	4e	4f	4g	4h	4i	4j	4k
SLM_2											
1h	2h	3h	4h	5h	6h	7h	8h	0	9i	9j	9k
1i	2i	3i	4i	5i	6i	7i	8i	9i	0	0j	0k
1j	2j	3j	4j	5j	6j	7j	8j	9j	0j	0	ak
1k	2k	3k	4k	5k	6k	7k	8k	9k	0k	ak	0

(a)

DLM_1											
1b	2b	0	3c	3d	3e	3f	3g	3h	3i	3j	3k
1c	2c	3c	0	4d	4e	4f	4g	4h	4i	4j	4k
1h	2h	3h	4h	5h	6h	7h	8h	0	9i	9j	9k
1i	2i	3i	4i	5i	5i	7i	8i	9i	0	0j	0k
DLM_0											
0	1a	1b	1c	1d	1e	1f	1g	1h	1i	1j	1k
1a	0	2b	2c	2d	2e	2f	2g	2h	2i	2j	2k
1f	2f	3f	4f	5f	6f	0	7g	7h	7i	7j	7k
1g	2g	3g	4g	5g	6g	7g	0	8h	8i	8j	8k
DLM_2											
1d	2d	3d	4d	0	5e	5f	5g	5h	5i	5j	5k
1e	2e	3e	4e	5e	0	6f	6g	6h	6i	6j	6k
1j	2j	3j	4j	5j	6j	7j	8j	9j	0j	0	ak
1k	2k	3k	4k	5k	6k	7k	8k	9k	0k	ak	0

(b)

Figure 2: SLM_s and DLM_s in the BC(BLOCK, *) to BC (2, *) data redistribution on $M_{12\times12}$ over 3 processors.

The above example shows that two difference processors might have the same data in the source distribution. Since data is distributed over processors in a regular block-cyclic manner, to accomplish data exchange, it is possible for one to reuse data elements that reside in local space if the data is necessary for next computation in the target distribution phase.

A similar observation can be extracted from the target distribution layout as shown in figure 2(b). As the above example, we examine the destination local matrix of P_1. Adjacent blocks $\tilde{DLM}_{1(3,1)}^{(0,0)} \cup \tilde{DLM}_{1(3,7)}^{(0,6)}$ and $\tilde{DLM}_{1(3,5)}^{(0,4)} \cup \tilde{DLM}_{1(3,11)}^{(0,10)}$ are identical to the transposition of $\tilde{DLM}_{0(3,3)}^{(0,2)} \cup \tilde{DLM}_{0(3,9)}^{(0,8)}$ and $\tilde{DLM}_{2(3,3)}^{(0,2)} \cup \tilde{DLM}_{2(3,11)}^{(0,10)}$, respectively. We consequently obtain the following two lemmas.

Lemma 2：Given $BC(x,*) \rightarrow BC(y,*)$ redistribution on an $n\times n$ symmetric matrix over P processors, for any destination processor q receives data from source processor p, in the target distribution, $\bigcup \tilde{DLM}_{p(s,t)}^{(0,g)} = (\bigcup \tilde{DLM}_{q(s,u)}^{(0,r)})^T$, where $g = q \times y/P$, $r = p \times y/P$, $s = n/P - 1$, $t = g + y - 1$, $u = g + y - 1$, $0 \le p, q < P$.

Lemma 3：Given $BC(x,*) \rightarrow BC(y,*)$ redistribution on an $n\times n$ symmetric matrix over P processors, if P_i needs to send S_j elements to P_j, then $S_j \times (1/P)$ elements within S_j exist originally in DLM_j, where $0 \le i, j < P$, $i \ne j$.

3.3.3 Algorithms

In general, data redistribution can be divided into sending and receiving phases. The detailed operations are performed in 6 steps.

Step 1：Indexing of source distribution. Calculate the data sets of source local array that need to be transmitted.

Step 2：Message packing. Gather the data according to data sets that obtained in Step 1 and pack them together into local buffers.

Step 3：Sending messages. Send all messages to corresponding destination processors.

Step 4：Indexing of destination distribution. Calculate the data sets of the destination local array that need to be received.

Step 5：Receive data. Receive all messages from their source processors into local buffers

Step 6：Message unpacking. Unpack messages from receiving buffers to destination local array according to the communication sets that obtained in Step 4.

According to the descriptions in section 3.3.2, we add two optimization phases into above operations. The modified steps are given as follows and represented in italic fonts.

Step 2：*Minimize Sending Data Sets. Calculate the index that destination processors already have the same data*; message packing, gather the data according to data sets that obtained in this step and pack them together into local buffers.

Step 4：Indexing of destination distribution. Calculate the data sets of the destination local array that need to be received. *Minimize Receiving Data Sets. Calculate the index that source processors are already have the same data and put the data into destination local matrix.*

The algorithm for symmetrical matrix redistribution is given as follows.

```
Algorithm_Symmetric_Redistribution
01. /*Indexing of Send*/
02.   for i=0 to local_row{
03.     golbal_row_index[i]= P_i *x+(i%x)+((i/x)*((numproc-1)*x));
04.   }
05.   for i=0 to local_row{
06     send_index[i][*]=((golbal_row_index[i]/y)%numproc);
```

```
07.   }
08./*Minimize Sending */
09.   for j=0 to local_col {
10.   brother_index[*][j]=j /x%P;
11.   }
12. /* Message Packing*/
13. for i= 0 to local_row{
14.     for j=0 to loca_col{
15.       if send_index[i][j]!=brother_index[i][j]{
16.          k=((gol_row_index[i]/y)%numproc);
17.         buff_k [size]= a_ij
18.     }}}
19. /*Send data*/
20. send data in  buff_k  to  P_k
21 /*indexing of Receive*/
22.   for i=0 to local_row{
23.    golbal_row_index[i]= P_i *y+(i%y)+((i/y)*((numproc-1)*y));
24.   }
25.   for i=0 to local_row{
26    receive_index[i][*]=((golbal_row_index[i]/x)%numproc);
27.   }
28./*Minimize Receiving */
29.   for j=0 to local_col {
30.   brother_index[*][j]=j /y%P;
31.   }
32, if receive_index[i][j]!=brother_index[i][j]{
33          M_i[g][h] = a_ij ;//g=0 to n/P×x , h=n/P×y
34      }
35,    transposition  M_i  to  M_i^T ;
36,    for k=1 to local_row {
37.          for j= 1 to g {
38.           if k/y% P=I {
39,         b_k,* = M_i[ j][*] ;
40.     }}}
41. /*Receive data*/
42. receive data in  buff_k  to  P_k
43 /*Unpacking*/
44. for i= 0 to local_row{
45.     for j=0 to loca_col{
46.       if receive_index[i][j]!=brother_index[i][j]{
47.          k=((gol_row_index[i]/x)%numproc);
48.          b_ij = buff_k [size]
49.     }}}
50.     end Algorithm_Symmetric_Redistribution
```

4. Performance Analysis and Experimental Results

4.1 Theoretical Analysis

To investigate the efficiency of the *SRA* method, we select a traditional redistribution algorithm, denoted as *TR*, to proceed our analysis. Given a `BC(x, *)` to `BC(y, *)` redistribution on an $n \times n$ symmetric matrix over P processors, the indexing cost of the traditional redistribution algorithm (*TR*), according to [3], can be defined as follow

$$T_{index}(TR) = O\left(\frac{lcm(x,y)}{\gcd(x,y)}\right) \qquad (3)$$

For packing/unpacking cost, since the size of local matrix is n^2/P, we can obtain that

$$T_{pack}(TR) = T_{unpack}(TR) = O\left(\frac{n^2}{P}\right) \qquad (4)$$

For the *SRA* method, in order to explore those identical pairs of communication patterns, each processor calculates the communication sets for matrix elements in both dimensions. This is twice as $T_{index}(TR)$, i.e., $2 \times T_{index}(TR)$. After the indexing phase, an intersection operation with asymptotic complexity is $O(n^2/P)$ should be carried out. Therefore, we have

$$T_{index}(SRA) = O\left(\frac{n^2}{P}\right) \qquad (5)$$

For the packing and unpacking process, according to lemma 3, each processor only needs to transmit $S_j \times ((P-1)/P)$ elements to destination processors P_j, where S_j is the size of message that should be sent to P_j. Therefore, the packing and unpacking costs of *SRA* can be defined as

$$T_{pack}(SRA) = T_{pack}(TR) \times \frac{P-1}{P} \qquad (6)$$

For the communication costs, because the *SRA* method minimizes the amount of data that need to be communicated, the analysis of this part will be focused on data transmission overheads. Since the data transmission overheads is directly proportional to the size of outgoing messages, therefore the communication costs can be similarly obtained from equation (6). Given the communication costs of *TR* method as $T_{comm}(TR)$, then the communication cost of *SRA* can be modeled as

$$T_{comm}(SRA) = T_{comm}(TR) \times \frac{P-1}{P} \qquad (7)$$

4.2 Experimental Result

To evaluate the performance of the proposed method, we have implemented the *SRA* and *TR* methods. Both methods were written in the single program multiple data (*SPMD*) programming paradigm with C+MPI codes and executed on an SMP/Linux cluster consisted of 16 SMP nodes, which are interconnected by 100M switch. Each SMP node has one AMD Athlon XP2000+ CPU and 1GB main memory. The operating system used is Linux kernel version 2.4.18. The mpich and gcc compiler we used is version 1.2.4 and gcc 3.2.1, respectively.

Table 1 shows the results of *SRA* and *TR* algorithms to perform a `BC` (`BLOCK`, *) to `BC` (`CYCLIC`, *) data redistribution on different matrix size and number of processors. In table 1, the terms *SRA_comm* and *TR_comm* represent the communication time of the algorithms; the terms *SRA_comp* and *TR_comp* represent the computational time for message

generation, the terms *SRA_total* and *TR_total* represent the overall execution time of *SRA* and *TR* algorithms, respectively. From table 1, we observed that *SRA_comp* is larger than *TR_comp*. Although the *SRA* method reduces 1/*P* message packing and unpacking costs, the *SRA* method constructs message sets twice in its local space as described in last section, which is two times to that of *TR* algorithm, i.e., $O(2 \times N/P)$; Therefore, the *SRA* method has little extra computational overheads.

For communication overheads, because *SRA* can reduce *1/P* transmission data, the *SRA_comm* is smaller than *TR_comm*. Since the communication time plays as the major factor to overall performance in runtime data redistribution, the extra computational overheads of *SRA* method will not offset the benefit of the reduced communication costs. Consequently, we can observe that *SRA_total* outperforms *TR_total*.

Table 1：Execution time of different algorithms to perform BC(BLOCK, *) to BC(CYCLIC, *)

Processor #	3	6	9	12	15
Matrix Size N×N	N=1500	N=3000	N=4500	N=6000	N=7500
SRA_comm	0.5079	0.8354	1.3608	1.8112	2.4085
TR_comm	0.7797	1.0479	1.5390	2.0859	2.6626
SRA_comp	0.0958	0.2421	0.3742	0.6808	0.9785
TR_comp	0.0393	0.1409	0.3022	0.4969	0.7644
SRA_total	0.6037	1.0775	1.735	2.4920	3.3870
TR_total	0.8190	1.1888	1.8412	2.5828	3.4271

single precision second

Another minor discovery is that the improvement rate of the *SRA* method decreased as the number of processors increased. This is because that the improvement rate of data transmission cost for the *SRA* algorithm is directly proportional to $(1/P \times \textit{matrix size})$. Consequently, the *SRA* algorithm performs well when matrix size is large or the number of processors is small. These phenomena match our previous theoretical demonstrations.

5. Conclusion

In this paper, we have presented an algorithm for efficient symmetric matrices data redistribution. Applying the attribute of symmetrical matrices, parts of identical data that located in local space can be reused to avoid inter-processor data exchange (as illustrated in lemmas 1 and 2). The theoretical analysis of lemma 3 also proved that 1/*P* message packing/unpacking and communication costs can be reduced. The experimental results reflect small extra computational overheads of the *SRA* algorithm will be incurred. However, the *SRA* technique can still provide superior overall performance. A shortcoming of our proposed algorithm is that *SRA* can handle only one of the two dimensions of a matrix at a redistribution phase, i.e., BC (x, *) to BC (y, *) and BC (*, x) to BC (*, y).

There are some possible extensions could be made. One of the issues would be to consider the problem with generalized arbitrary source and destination processor sets. Another important future research direction would be to investigate the techniques in irregular scientific computation problems. It would also be interesting to extend this technique on computational grid architectures.

Acknowledgments

The authors are grateful to the anonymous referees whose insightful comments enabled us to make significant improvements. The work of this paper was supported in part by NSC of Taiwan under grant number NSC92-2213-E-216-029.

Reference

[1]. Y.-C Chung, C.-H Hsu , S.-W Bai, A Basic-Cycle Calculation Technique for Efficient Dynamic Data Redistribution, *IEEE Trans. on PDS*, Vol. 9, No. 4, pp. 359-377, April 1998.

[2]. 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.

[3]. C.-H Hsu, S.-W Bai, Y.-C Chung, C.-S Yang, “A Generalized Basic-Cycle Calculation Method for Efficient Array Redistribution,” *IEEE Trans. on PDS*, Vol. 11, No. 12, pp. 1201-1216, Dec. 2000.

[4]. Emmanuel Jeannot and Frédéric Wagner, “Two Fast and Efficient Message Scheduling Algorithms for Data Redistribution through a Backbone,” Proceedings of the 18th International Parallel and Distributed Processing Symposium, April 2004.

[5]. Jens Knoop , Eduard Mehofer, “Distribution Assignment Placement: Effective Optimization of Redistribution Costs,” *IEEE Trans. on PDS*, Vol. 13 No. 6, pp. 628-647, June 2002.

[6]. Kaiser T.H., Baden S.B., “Overlapping communication and computation with OpenMP and MPI,” Scientific Programming, Vol. 9, No. 2-3, pp. 69-71(3) 2001.

[7]. Edgar T. Kalns, and Lionel M. Ni, “Processor Mapping Technique Toward Efficient Data Redistribution,” *IEEE Trans. on PDS*, Vol. 6, No. 12, December 1995.

[8]. S. Ramaswamy, B. Simons, and P. Banerjee, "Optimization for Efficient Array Redistribution on Distributed Memory Multicomputers," *JPDC*, Vol. 38, pp. 217-228, 1996.

[9]. A.K. Somani and A.M. Sansano, Minimizing Overhead in Parallel Algorithms through Overlapping Communication/Computation, Tech. Report 97-8, NASA ICASE, Langley, VA., Feb. 1997.

[10]. D.W. Walker and S.W. Otto, “Redistribution of Block-Cyclic Data Distributions Using MPI,” *Concurrency: Practice and Experience*, Vol. 8, No. 9, pp. 707-728, 1996.