An Optimal Processor Replacement Scheme for Efficient Communication of Runtime Data Redistribution

Ching-Hsien Hsu, Kun-Ming Yu, Chi-Hsiu Chen, Chang Wu Yu, and Chiu Kuo Lian

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- Dynamic data distribution is used to enhance data locality and algorithm performance with reducing inter-processor communication in data parallel programs on distributed memory multi-computers. Since the exchange of data is performed at run-time, there is a performance tradeoff between the efficiency of the new data decomposition for a subsequent phase of an algorithm and the cost of exchanging data among processors. In this paper, we present an Optimal Processor Replacement (OPR) scheme to minimize data transmission cost for general BLOCK-CYCLIC data redistribution. The main idea of the proposed techniques is to employ a size oriented greedy matching method or a maximum bipartite matching theory for exploring an ideal one-to-one mapping between logical processors. Based on the matching policy, a realigned sequence of destination processors can be derived and is then used to perform data redistribution in the destination phase. A significant improvement of this approach is that OPR achieves the highest rate of data remain in local space and leading minimum inter-processor communication. The optimal processor replacement scheme can handle array redistribution with arbitrary source and destination distribution in BLOCK-CYCLIC type and can be applied to multidimensional arrays. To evaluate the performance of the proposed technique, we have implemented the OPR method on an SMP Cluster with 24 nodes. The theoretical analysis and experimental results show that our technique provides considerable improvement for runtime data redistribution.

Keywords: Processor Replacement, Communication Optimization, Maximum Matching, Data Redistribution, Parallelizing Compiler, Runtime Support.

1 The correspondence address.
1. Introduction

The data parallel programming model has become a widely accepted paradigm for parallel programming on distributed memory multicomputers. 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. Many data parallel programming languages such as High Performance Fortran (HPF), Fortran D, High Performance C (HPC) and Vienna Fortran provide compiler directives for programmers to specify data distribution. Data distribution provided by these languages, in general, can be classified into two categories, regular and irregular. The regular array distribution has three types, BLOCK, CYCLIC, and BLOCK-CYCLIC(c). The irregular array distribution uses user-defined array distribution functions to specify array distribution. Dongarra et al. [4] have shown that the regular types of distribution are essential for many scientific algorithms design on distributed memory parallel machines.

In some scientific applications, such as multi-dimensional fast Fourier transform, the Alternative Direction Implicit (ADI) method for solving two-dimensional diffusion equations, and linear algebra solvers, algorithms are decomposed into phases; a distribution that is well-suited for one phase may not be good for a subsequent phase in terms of performance. In each phase, there exist suitable distributions of data onto processor grid. The optimal distribution depends on the characteristics of an algorithm, as well as on the attributes of the target architecture. Because the optimal distribution changes from one phase to another, data redistribution turns out to be a critical operation during runtime, as mentioned in [13, 25]. Therefore, many data parallel programming languages support run-time primitives for changing a program’s data decomposition. Since data redistribution is performed at run-time, there is a performance trade-off between the efficiency of the new data decomposition for a subsequent phase of an algorithm and the cost of redistributing matrix data among processors. Thus efficient methods for performing data redistribution are of great importance for the development of parallelizing compilers for those languages in which data layouts can be dynamically specified via
REDISTRIBUTE directives.

In general, data redistribution consists of computational cost, wherein communication sets calculation and message packing/unpacking is involved, as well as communication cost, wherein data are exchanged between processors. Typically, the communication costs play an important factor and dominate overall performance of a redistribution operation. In this paper, we present an optimal processor replacement scheme to minimize data transmission cost for general BLOCK-CYCLIC data redistribution. The main idea of the proposed techniques is to employ a size oriented greedy matching method or a maximum bipartite matching theory for exploring an ideal one-to-one mapping between logical processors. Based on matching policy, a realigned sequence of destination processors can be derived and is then used to perform data redistribution. The present technique has two major characteristics.

- The optimal processor replacement scheme achieves the highest ratio of data remain in local space and leading minimum inter-processor communication.
- The optimal processor replacement scheme can handle array redistribution with arbitrary source and destination distribution in BLOCK-CYCLIC type and can be applied to multidimensional arrays.

The rest of this paper is organized as follows. In Section 2, a brief survey of related work will be presented. Section 3 gives notations, terminologies and the cost model used in this paper. In section 4, we will introduce the optimal processor replacement scheme for data redistribution. Performance analysis and experimental results will be given in Section 5. Section 6 briefly concludes this paper.

2. Related Work

Many previous research efforts have been focused on the problem of data distribution and redistribution in multi-computer compiler techniques or runtime support techniques. Some work has concentrated mainly on the indexing and communication sets identification while some has addressed on the communication efficiency. We briefly survey the literature on these two aspects.

For index sets calculation of array statements with BLOCK-CYCLIC(c) distribution, Chatterjee et al. [2], enumerated the local memory access sequence of
communication sets for array statements with \texttt{BLOCK-CYCLIC}(c) distribution based on a finite-state machine at most \(c\) states. Gupta \textit{et al.} \cite{gupta1992} derived closed form expressions to efficiently determine the send/receive processor/data sets. They also provided a virtual processor approach for addressing the problem of reference index-set identification for array statements with \texttt{BLOCK-CYCLIC}(c) distribution and formulated active processor sets as closed forms. In \cite{lee1994}, an approach for generating communication sets by computing the intersections of index sets corresponding to the LHS and RHS of array statements was presented. The intersections are computed by a scanning approach that exploits the repetitive pattern of the intersection of two index sets. Lee \textit{et al.} \cite{lee1996} derived communication sets for statements of arrays that were distributed in arbitrary \texttt{BLOCK-CYCLIC}(c) fashion. They also presented closed form expressions of communication sets for restricted block size. A similar approach that addressed the problems of the index set and the communication sets identification for array statements with \texttt{BLOCK-CYCLIC}(c) distribution was presented in \cite{barker1996}. The \texttt{BLOCK-CYCLIC}(k) distribution was viewed as a union of \(k\) \texttt{CYCLIC} distribution. Since the communication sets for \texttt{CYCLIC} distribution is easier to determine, communication sets for \texttt{BLOCK-CYCLIC}(k) distribution can be generated in terms of unions and intersections of some \texttt{CYCLIC} distributions.

The above studies have concentrated on efficiently generating the communication messages at compile time. Techniques for dynamic data redistribution are discussed in many papers \cite{piepie,scalapack,pitfalls,pitfallsh,scalapackh,scalapackj,scalapackk,scalapackl,scalapackm,scalapackn,scalapacko,scalapackp}. Substantially, these researches can be classified in two aspects, the computation efficiency and the communication optimizations. Methods for the first part devote developing efficient algorithms to perform runtime array redistribution. The \texttt{PITFALLS} \cite{pitfalls} and the \texttt{ScalAPACK} \cite{scalapack} methods are typical examples. They pay more attention on the communication sets identification and message packing/unpacking issues. Some methods assume that the redistribution of an array is under the same source/destination processor set. They proposed algorithms to generate the communication sets for some specific type of redistribution, such as \texttt{BLOCK} to \texttt{CYCLIC} redistribution \cite{scalapack1}, \texttt{BLOCK-CYCLIC}(kr) to \texttt{BLOCK-CYCLIC}(r)
redistribution. For communication optimizations, methods in this category, in general, address different approaches to reduce the communication overheads in a redistribution operation. Examples are the processor mapping technique [13] for minimizing data transmission overheads, the multiphase redistribution strategy [14] for reducing message startup cost, the communication scheduling approach [3, 6, 8, 18, 19, 27] for avoiding node contention, the strip mining approach [24] for overlapping the communication and computation steps and the spiral mapping technique [26] for enhancing communication locality.

The processor mapping technique produces a significant improvement among the above techniques; thus, it is adopt by many researches [11, 12, 13, 16]. In [11], two mapping functions were proposed for $kr \rightarrow r$ and $r \rightarrow kr$ data redistribution. A processor reordering technique based on size-oriented and number-oriented schemes was presented in [16] and a special algorithm for communication free data redistribution was presented in [12]. In the following, we will discuss the method that can be applied to general redistribution problems and achieve the highest ratio of data locality. This is the main objective of this work.

3. Preliminaries and Cost Model

In this section, we describe some notations and terminology used in this paper. To simplify the presentation, we use $BC_{x \rightarrow y}$ to represent the BLOCK-CYCLIC($x$) to BLOCK-CYCLIC($y$) data redistribution for the rest of the paper.

3.1 Preliminaries

Data redistribution, in general, can be performed in two phases, the sending phase and the receiving phase. In the sending phase, a source processor $P_i$ has to determine all the data sets that it needs to send to other processors (destination processors), pack those data sets into messages, and send messages to their destination processors. In the receiving phase, a destination processor $P_i$ has to determine all the data sets that it needs to receive from other processors (source processors), receive messages from source processors, and unpack elements in messages to their corresponding local array positions. This means that each
processor $P_i$ should compute the following four sets.

- **Destination Processor Set** ($\text{DPS}(P_i)$) : the set of processors to which $P_i$ has to send data.
- **Send Data Sets** ($\bigcup_{P_i \in \text{DPS}(P)} \text{SDS}(P_i \rightarrow j)$) : the sets of data that processor $P_i$ has to send to its destination processors, where $\text{SDS}(P_i \rightarrow j)$ denotes the set of data that processor $P_i$ has to send to its destination processor $P_j$.
- **Source Processor Set** ($\text{SPS}(P_j)$) : the set of processors from which $P_j$ has to receive data.
- **Receive Data Sets** ($\bigcup_{P_{i-1}} \text{RDS}(P_{j-1})$) : the sets of data that $P_j$ has to receive from its source processors, where $\text{RDS}(P_{j-1})$ denotes the set of data that processor $P_j$ has to receive from its source processor $P_i$.

We use an example to clarify the above description. Figure 1 show a $\text{BC}_{10 \rightarrow 5}$ data redistribution on $A[1:100]$ over five processors. The DPS of source processors $P_0$ to $P_4$ are $\text{DPS}(P_0) = \{P_0, P_1\}$, $\text{DPS}(P_1) = \{P_2, P_3\}$, $\text{DPS}(P_2) = \{P_4, P_0\}$, $\text{DPS}(P_3) = \{P_1, P_2\}$ and $\text{DPS}(P_4) = \{P_3, P_4\}$, respectively. The Send Data Set, for example, source processor $P_0$ has $\text{SDS}(P_0 \rightarrow 0) = \{A[0:4], A[50:54]\}$ and $\text{SDS}(P_0 \rightarrow 1) = \{A[5:9], A[55:59]\}$. The Receive Data Set, for example, destination processor $P_0$ has $\text{RDS}(P_0 \leftarrow 0) = \{A[0:4], A[50:54]\}$ and $\text{RDS}(P_0 \leftarrow 2) = \{A[25:29], A[75:79]\}$. The complete DPS, SDS, SPS, and RDS of each processor for the above example are shown in Figure 2(a) to 2(d).

![Figure 1](image-url)

Figure 1: A $\text{BC}_{10 \rightarrow 5}$ redistribution on a one-dimensional array $A[1:100]$ over 5 processors.
Figure 2: Communication Sets for $\text{BC}_{10\rightarrow 5}$ on $A[1:100]$ over 5 processors. (a) DPS (b) SDS (c) SPS (d) RDS

Given a $\text{BC}_{s\rightarrow t}$ over $P$ processors, a bipartite graph ($BG$) with $P$ nodes in each side can be used to represent the source and destination processor sets. To formulate the communication patterns of above example, we use $|\text{DPS}[P_i]|$, the number of destination processors in $\text{DPS}[P_i]$ and $|\text{SDS}[P_i\rightarrow j]|$, the number of elements in $\text{SDS}[P_i\rightarrow j]$ to represent the out degree of node $P_i$ and the weight ($w_{ij}$) of edge $e_{ij}$ in $BG$, respectively, as shown in Figure 3

Figure 3: Bipartite representation of communication patterns in $\text{BC}_{10\rightarrow 5}$ redistribution over five processors.

3.2 Cost Model

The time ($T$) for an algorithm to perform runtime data redistribution, in general, consists of computation ($T_{\text{comp}}$) and communication ($T_{\text{comm}}$) time. The computation time includes indexing, packing and unpacking overheads. Indexing is the cost for an algorithm to compute the source/destination processors of local elements, packing is the cost to pack local elements that have the same destination processor to message,
and unpacking is the cost to unpack elements in messages that received from source processors to corresponding local positions; The communication time is for an algorithm to send and receive data among processors. It can be divided into two parts, the message startup cost \( T_s \) and the data transmission costs \( T_d \) of the interconnection network of a parallel machine. Therefore, for processor \( P_i \), the time to perform runtime data redistribution can be modeled as follows,

\[
T = T_{\text{comp}} + \alpha_i \times T_s + \delta_i \times T_d,
\]

where \( \alpha_i = |DPS(P_i)| \) is the number of destination processors in \( DPS(P_i) \) and \( \delta_i = \sum_{P \in DPS(P_i)} |SDS[P \rightarrow i]| \) is the total number of elements in all \( SDS[P \rightarrow i] \), for all \( j \in DPS[P_i] \).

In general, data transmission cost is directly proportional to the size of redistributing arrays and influence the execution time of runtime data redistribution. To evaluate data transmission cost, we recall the bipartite representation in section 3.1 for deriving a simple formulation. Given a runtime data redistribution problem, for source processor \( P_i \), \( \delta_i \) is equal to the summation of edge weight for all directional edges that has source vertex \( P_i \), i.e., \( \delta_i = \sum_{\forall \neq P_i} w_{ij} \). Because the edge \( e_{ij} \) will not incur inter-processor communication when \( i = j \), we can have the following equation,

\[
\delta_i = \sum_{\forall \neq P_i, i \neq j} w_{ij}.
\]

It is possible for processors to have uneven data transmission cost \( (\delta_i) \). The global measurement is an alternative strategy to obtain legitimate performance analysis. Let \( \delta \) represents the total data transmission cost in a redistribution, we can have the following equation,
\[ \delta = \sum_{i=0}^{P-1} \delta_i, \text{ where } P \text{ is the number of processors} \quad (3) \]

Another structure widely used to demonstrate the inter-processor communication patterns is *Communication Table* (*CT*). Given a \( BC_{a \rightarrow \epsilon} \) redistribution over \( P \) processors, a communication table is a \( P \times P \) matrix. Recall the bipartite representation, in communication table, \( CT_{ij} = w_{ij} \), the edge weight of \( e_{ij} \) in \( BG \). For example, consider again the redistribution illustrated in section 3.1, the communication table is shown in Figure 4. Since \( CT_{ij} \) will not incur inter-processor communication when \( i = j \), equation 3 can be represented as

\[ \delta = \sum_{i,j=0}^{P-1} CT_{ij} - \sum_{i=0}^{P-1} CT_{ii} \quad (4) \]

Let \( L_d \) be summation of the value of elements in the main diagonal in \( CT \), i.e., \( L_d = CT_{00} + CT_{11} + \ldots + CT_{P-1 P-1} \). equation 4 can be represented as follows

\[ \delta = \sum_{i,j=0}^{P-1} CT_{ij} - L_d \quad (5) \]

<table>
<thead>
<tr>
<th>( CT )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_0 )</td>
</tr>
<tr>
<td>( P_0 )</td>
</tr>
<tr>
<td>( P_1 )</td>
</tr>
<tr>
<td>( P_2 )</td>
</tr>
<tr>
<td>( P_3 )</td>
</tr>
</tbody>
</table>

Figure 4: Communication Table of \( BC_{10 \rightarrow 5} \) redistribution on \( A[1:100] \) over five processors.

Since \( \sum_{i,j=0}^{P-1} CT_{ij} \) is constant, increase \( L_d \) will lead lower \( \delta \) and decrease the total data transmission cost. In the following section, we will describe the techniques to minimize \( L_d \).
4. Processor Replacement Scheme

Runtime data redistribution is invoked when the data access patterns are changed during one of several computation phases in a parallel application. In order to minimize communication overheads, Ni et al. first proposed the processor mapping technique for BLOCK to CYCLIC data redistribution [11]. We then extend the techniques to generalized $B_{c_{kr}r}$ type [10]. In this paper, we present an optimal method to generalize the techniques to general type of array redistribution, i.e., $s \rightarrow t$, where $s$ and $t$ are arbitrary integers.

4.1 Motivating Example

We begin by illustrating the utility of processor mapping technique for one-dimensional data. Consider again the example used in section 3.1. Figure 5 shows data redistribution using two different Logical Processor Sequences (LPS), the Traditional Logical Processor Sequences (TLPS) and the Realigned Logical Processor Sequences (RLPS). The shadow blocks represent data locates in the same processor in both source and destination distribution. We mention these data as Local Data Sets ($L_d$). By constructing the communication tables for these two manners as shown in Figure 6, we observe that $L_d = CT_{00} + CT_{44} = 20$ in TLPS in Figure 6(a) and $L_d = CT_{00} + CT_{11} + CT_{22} + CT_{33} + CT_{44} = 50$ in RLPS in Figure 6(b). Obviously, the RLPS yields higher $L_d$ through the redistribution. The above example shows that a good realigned logical processor sequence can lead higher local data and reduce global inter-processor communication overheads.

![Figure 5: Data redistribution using Traditional Processor Sequence (TPS) and Realigned Processor Sequence (RPS).](image-url)
Figure 6: Communication Tables of $BC_{10 \rightarrow 5}$ redistribution on $A[1:100]$ over five processors. (a) CT for TLPS (b) CT for RLPS

4.2 Size Oriented Greedy Matching

A $BC_{s \rightarrow t}$ redistribution over $P$ processors determines an unique $CT$ which could be a global information to all processors involved in the redistribution computation. To obtain an idea logical processor sequence, apply the structure of communication table is a primitive method. Considering another example of $BC_{5 \rightarrow 4}$ over 12 processors. The communication table is shown in Figure 7(a). Since the communication is repetitive in each distribution cycle (i.e., $lcm(sP,tP)$), we only need to calculate the patterns in the first distribution cycle. According to the motivating example, we know that to permute the order of logical processors’ id for destination data-layout might increase the amount of elements in local data sets. A Size-Oriented Greedy (SOG) approach for reordering the logical processor sequence is quite useful in this problem.

Let $RLPS[0:P-1]$ be the realigned sequence of logical processors and $\beta$ is the largest $CT_{ij}$ in CT, for all $0 \leq i, j \leq P - 1$. The SOG method sets $RLPS[j]$ to $i$ if $CT_{ij} = \beta$. For example, In Figure 7(a), $\beta = 4$; for source processors (the first column) $P_0, P_3, P_4, P_7, P_8$ and $P_{11}$, the largest size of data blocks in a distribution cycle is equal to 4. Therefore, the $RLPS$ is set to $\{0, 3, 4, 7, 8, 11, -, -, -, -, -, -\}$. Following, processors $P_1, P_2, P_5, P_6, P_9$ and $P_{10}$ have largest block size = 3 (i.e., $\beta - 1$). For processor $P_1$, $CT_{11}, CT_{14}, CT_{17}$ and $CT_{10}$ are equal to 3. Since $RLPS[1]$ and $RLPS[4]$ were prior occupied by $P_3$ and $P_8$, thus, $RLPS[7]$ is set to $P_1$. Similarly, $RLPS[6]$ is set to $P_2$. Eventually, we have $RLPS = \{0, 3, 4, 7, 8, 11, 2, 1, 6, 5, 10, 9\}$. The communication tables for $BC_{5 \rightarrow 4}$ over 12 processors using $TLPS$ and $RLPS$ are shown in Figure 7(b). The algorithm of the size-oriented greedy approach is
Algorithm Size Oriented Greedy Matching

1. \( maxmsg_i \): sorted list of distinct message size for \( P_i \);
2. \( CT \): the communication table, is a \( P \times P \) matrix;
3. \( \beta \): sorted list of distinct message size in \( CT \);
4. \( RLPS \): vector of length \( P \), initialize \( MAXINT \); \( k = \beta \);
5. \( ST \): Boolean vector \([0: P-1]\);
6. \( \text{while} \ (k) \) {
7. \( \text{for} \ (i = 0; i < P; i++) \) {
8. \( \text{if} \ ((ST[i]) \&\& (maxmsg_i == \beta)) \) {
9. \( \text{for} \ (j = 0; j < P; j++) \) {
10. \( \text{if} \ ((CT[i][j] == \beta) \&\& (RLPS[j] != MAXINT)) \) {
11. \( ST[i] = \text{true}; RLPS[j] = i; \)
12. \} \)
13. \)
14. \)
15. \)
16. \( \text{select next} \ \beta \text{and set it to} \ k; \)
17. \( \text{for} \ (i = 0; i < P; i++) \) {
18. \( \text{if} \ ((ST[i]) \text{set maxmsg_i to next size;} \)
19. \)
20. \)
21. \( \text{assign remainder RLPS null entries;} \)

end_of_Size Oriented Greedy Matching

4.3 Maximum Matching

It is possible that an \( RLPS \) derived from the \( SOG \) matching is not an optimal result for a given general \( BLOCK-CYCLIC \) redistribution problem. For example, Figure 8(a) shows the communication table for a \( BC_{8 \rightarrow 6} \) over 11 processors. The amount of elements in local data sets \((L_d)\) is equal to 24 for the \( TLPS \). The \( SOG \) given as follows.

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>P9</th>
<th>P10</th>
<th>P11</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 7: Communication Tables of \( BC_{5 \rightarrow 4} \) over 12 processors. (a) \( CT \) for \( TLPS \) \( (P_0 P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_{10} P_{11}) \). (b) \( CT \) for \( RLPS \) \( (P_0 P_3 P_4 P_7 P_8 P_{11} P_2 P_1 P_6 P_5 P_{10} P_9) \).
matching derives the $RLPS = \{0, 5, 3, 4, 2, 7, 1, 6, 9, 8, 10\}$ and yield $L_d = 100$ as shown in Figure 8(b). Another $RLPS$ derived from a maximum bipartite matching algorithm is $\{0, 5, 10, 4, 9, 3, 8, 2, 7, 1, 6\}$ which produces most local data elements ($L_d = 100$). This example shows that the $SOG$ matching does not guarantee the derived $RLPS$ is an optimal result. Since a communication table ($CT$) can be modeled as a bipartite graph directly that describe in section 3.1, we apply Hopcroft and Karp’s maximum matching algorithm for deriving an optimal $RLPS$. The time complexity of Hopcroft and Karp’s maximum matching algorithm is $O(n^2 \log n)$, where $n$ is the number of vertices (i.e., number of processors). We will describe the variance between $SOG$ matching and maximum matching in the following section.

![Figure 8: Communication Tables of $BC_{8\rightarrow6}$ over 11 processors. (a) CT for TLPS. (b) CT for RLPS derived from SOG matching. (c) CT for RLPS derived from Maximum matching.](image)

### 5. Performance Evaluation

To evaluate the performance of the proposed techniques, a set of redistribution samples as shown in Table 1 is used for theoretical analysis and experimental simulation. These test samples are classified into two classes, $S$ type and $R$ type. $S1$ to $S4$ are $BC_{s\rightarrow t}$ examples, in which $s$ is not divisible by $t$ and $t$ is not divisible by $s$. $R1$ to $R3$ are $BC_{s\rightarrow t}$ examples, in which $s$ is divisible by $t$ or $t$ is divisible by $s$.

<table>
<thead>
<tr>
<th>Table 1: Test samples used in this paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S2</td>
</tr>
<tr>
<td>S3</td>
</tr>
<tr>
<td>S4</td>
</tr>
<tr>
<td>R1</td>
</tr>
<tr>
<td>R2</td>
</tr>
<tr>
<td>R3</td>
</tr>
</tbody>
</table>
5.1 Theoretical Analysis

Table 2 shows the theoretical improvement rate of data transmission costs for two matching schemes, the SOG matching and the maximum matching. To simplify the presentation, we have the following declarations. Let

\[ L_{d,TR} \] be the amount of elements in local data sets for TLPS.

\[ L_{d,SOG} \] be the amount of elements in local data sets for RLPS that derived from SOG matching.

\[ L_{d,MM} \] be the amount of elements in local data sets for TLPS that derived from maximum matching.

The data transmission improvement rates are defined as

\[ IR_{SOG} = \frac{(L_{d,SOG} - L_{d,TR})}{\delta} \]  \hspace{1cm} (6)

\[ IR_{MM} = \frac{(L_{d,MM} - L_{d,TR})}{\delta} \]  \hspace{1cm} (7)

where \[ \delta = \sum_{i,j=0}^{P} CT_{ij}. \]

Table 2: The theoretical improvement rate of data transmission cost for maximum matching and SOG matching

<table>
<thead>
<tr>
<th></th>
<th>Maximum Matching</th>
<th>SOG Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P=12 P=16 P=24 P=48 P=64 P=128</td>
<td>P=12 P=16 P=24 P=48 P=64 P=128</td>
</tr>
<tr>
<td>S2</td>
<td>5.00 10.42 9.17 11.25 15.10 15.89</td>
<td>5.00 10.42 9.17 11.25 15.10 15.89</td>
</tr>
<tr>
<td>S3</td>
<td>3.75 4.38 6.25 8.44 9.06 9.84</td>
<td>3.75 4.38 6.25 8.44 9.06 9.84</td>
</tr>
<tr>
<td>S4</td>
<td>0.23 0.57 0.80 2.90 3.34 4.09</td>
<td>0.23 0.57 0.80 2.89 3.34 4.09</td>
</tr>
<tr>
<td>R1</td>
<td>41.67 43.75 45.83 47.92 48.44 49.22</td>
<td>41.67 43.75 45.83 47.92 48.44 49.22</td>
</tr>
<tr>
<td>R2</td>
<td>22.22 25.00 27.78 30.56 31.25 32.29</td>
<td>22.22 25.00 27.78 30.56 30.73 32.29</td>
</tr>
<tr>
<td>R3</td>
<td>12.5 18.75 18.75 21.88 23.44 24.22</td>
<td>12.5 18.75 18.75 21.88 23.44 24.22</td>
</tr>
</tbody>
</table>

In table 2, we found that the SOG matching and maximum matching have almost the same improvement rate for most of cases, except for S4 (P=48), R2 (P=16) and R2 (P=64). Due to this observation, we use the redistribution algorithm adopt RLPS derived from maximum matching to run the simulation test in our experiments.
5.2 Experimental Results

To get the performance comparison, we have implemented the traditional redistribution method and the optimal processor replacement scheme. To simplify the presentation, we use TR and OPR to represent the traditional method and the techniques proposed in this paper, respectively. Both of these two methods were written in the single program multiple data (SPMD) programming paradigm with C+MPI code and executed on an SMP/Linux cluster consisted of 24 SMP nodes, which are interconnected by 100M switch. Each SMP node has one AMD Athlon XP2000+ CPU and 1GB main memory. The mpich and gcc compiler we used is version 1.2.4 and gcc 3.2.1, respectively.

Figures 9(a) and (b) show the execution time of TR and OPR to perform S1 and R1 redistribution, respectively. As the theoretical prediction, OPR scheme outperforms the TR algorithm for both cases. This is because the amount of data needs to be exchanged in OPR is less than that in the TR algorithm. The data transmission cost is consequently decreased.

Figures 10(a) and (b) illustrate the improvement rate for six test samples on different number of processors. We observe that R-type redistribution has higher improvement rate than S type. In each type, the OPR scheme produces variant improvement on different test samples, e.g., S1>S2>S3, R1>R2>R3. These phenomenon matches the information given in Table 1. Another minor discovery is
that improvement rates obtained from experiments are generally lower than the theoretical value. This is because the re-indexing overhead was counted in the data redistribution time. Since the communication costs play as the most critical time in a redistribution, consequently, the OPR scheme can provide better performance.

![Figure 10: Improvement rate of OPR scheme on different cases. (a) S type redistribution. (b) R type redistribution (size = 1.152×10^7 Bytes)](image)

Figures 11(a) and (b) present the execution time of TR and OPR to perform S2 and R2 redistribution on different data sizes, respectively. We have similar observations as those in Figure 9.

![Figure 11: Performance of the TR and OPR methods to execute data redistribution on a 24-node PC cluster. (a) execution time of BC_{5→6} (b) execution time of BC_{15→5} (M = 1.152×10^7 Bytes)](image)

Figures 12(a) and (b) give the speedup of OPR scheme to TR algorithm on
different cases. The speedup is defined as $T_{TR}/T_{OPR}$, where $T_{TR}$ and $T_{OPR}$ represent execution time of TR and OPR to perform data redistribution, respectively. For S1 to S3, OPR has up to 1.14 speedup. For R1 to R3, OPR achieves 1.2 to 1.8 speedup.

From the above performance analysis and experimental results, we have the following remarks.

Remark 1: The OPR scheme can minimize data transmission cost. The simulation results show that OPR achieves 3% to 12% improvement for S type redistribution and 13% to 45% improvement rate for R type redistribution.

Remark 2: When the number of processors is numerous, the improvement rate becomes more significant.

Remark 3: Given a $BC_{s \rightarrow t}$ redistribution, the upper bound of improvement rate for R type redistribution by using OPR scheme is $1/k$, where $k$ is equal to $s/t$ or $t/s$.

Figure 12: Speedup of OPR scheme to TR algorithm on different cases. (a) S type redistribution. (b) R type redistribution ($M = 1.152 \times 10^7$ Bytes)

6. Conclusions

Dynamic BLOCK-CYCLIC data redistribution is used to enhance the performance of SPMD programs in many scientific applications. In this paper, we have presented an Optimal Processor Replacement (OPR) scheme to minimize data transmission cost for general BLOCK-CYCLIC data redistribution. By re-indexing the ranks of destination processors, the desired destination data-layout can be accomplished with a realigned logical processor sequence. A significant
improvement of this approach is that OPR achieves the highest rate of data remain in local space and leading minimum inter-processor communication. The optimal processor replacement scheme can handle array redistribution with arbitrary source and destination distribution in BLOCK-CYCLIC type and can be applied to multidimensional arrays. The theoretical analysis and experimental tests show the efficiency of the proposed technique is superior to traditional algorithms. The OPR technique provides 3% ~ 12% and 13% ~ 45% improvement rates for general and special cases of runtime data redistribution, respectively.

References

Redistribution, ” *IEEE Trans. on PDS*, vol. 6, no. 12, December 1995.


