Efficient Algorithms for Block-Cyclic Array Redistribution between Processor Sets

Neungsoo Park* and Viktor K. Prasanna*
Department of EE-Systems, EEB-200C
University of Southern California
Los Angeles, CA 90089-2562
{neungsoo + prasanna}@halcyon.usc.edu
http://ceng.usc.edu/~prasanna

Cauligi Raghavendra
The Aerospace Corporation
P.O. Box 92957
Los Angeles, CA 90009-2957
raghu@rush.aero.org

(Student Paper)

Abstract

Run-time array redistribution is necessary to enhance the performance of parallel programs on distributed memory supercomputers. In this paper, we present an efficient algorithm for array redistribution from $cyclic(x)$ on P processors to $cyclic(Kx)$ on Q processors. The algorithm reduces the overall time for communication by considering the data transfer, communication schedule, and index computation costs. The proposed algorithm is based on a generalized circulant matrix formalism. Our algorithm generates a schedule that minimizes the number of communication steps and eliminates node contention in each communication step. The network bandwidth is fully utilized by ensuring that equal-sized messages are transferred in each communication step. Furthermore, the procedure to compute the schedule and the index sets is extremely fast. It takes $O(max(P,Q))$ time. Therefore, our proposed algorithm is suitable for run-time array redistribution. To evaluate the performance of our scheme, we have implemented the algorithm using C and MPI. The experiments were conducted on the IBM SP2. The experimental results show that the proposed algorithm outperforms well-known algorithms when the total communication time including the data transfer and schedule and index computation times are considered.

Permission to make digital/hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SC '98, Orlando, FL
(c) 1998 IEEE 0-89792-984-X/98/0011 $3.50

Ex.1157 / IPR2025-00916 / Page 1 of 13
Intel Corporation v. Advanced Cluster Systems, Inc.

1 Introduction

Many High Performance Computing (HPC) applications, including scientific computing and signal processing, consist of several stages. Some examples of such applications include the multidimensional Fast Fourier Transform, the Alternative Direction Implicit (ADI) method for solving two-dimensional diffusion equation, and linear algebra solvers. As the program execution proceeds from one stage to another, the data access pattern and the number of processors required for fully exploiting parallelism in the application may change. These changes cause the data distribution in a stage to be unsuitable for a subsequent stage. If data redistribution is not performed, the remote access overheads can be excessive. Since the requirements of different computation stages are, in general, not known at compile time, run-time data redistribution is necessary between consecutive stages. However, the cost of redistribution can offset the performance benefits achieved by the redistribution. Therefore, run-time redistribution must be implemented efficiently to ensure overall performance improvement.

Array data are typically distributed in a *block-cyclic* pattern onto a given set of processors. The block-cyclic distribution with block size x is denoted as $cyclic(x)$. A block contains x consecutive array elements. Blocks are assigned to processors in a round-robin fashion. Other distribution patterns, *cyclic* and *block* distribution, are special cases of the block-cyclic distribution. In general, the block-cyclic array redistribution problem is the reorganization of an array from one block-cyclic distribution to another, *i.e.*, from $cyclic(x)$ to $cyclic(y)$. An important case of this redistribution problem is redistribution from $cyclic(x)$ to $cyclic(Kx)$ which arises in many scientific and signal processing applications. This type of data redistribution can occur within a same processor set, or it can occur between different processor sets.

A general form of this array redistribution from $cyclic(x)$ on P processors to $cyclic(Kx)$ on Q processors has received considerable attention [5, 11, 15]. In performing the redistribution, the following four costs must be considered:

Index Computation Cost: For a given initial (final) data distribution, each source (destination) processor needs to compute the local index of an array element that it has in the initial (final) distribution. This index set is used by a source processor while packing array elements in a message. Similarly, a destination processor uses this set to store the received data in appropriate locations. There is a cost associated with the computation of the index for each array element in the source (as well as in the destination) processor.

Schedule Computation Cost: The communication schedule specifies sender-receiver pairs in each communication step. Since in each communication step, a processor can send at most one message and a processor can receive at most one message, careful scheduling is required to avoid contention while minimizing the number of communication steps. Time to compute this communication schedule can be significant. Keeping this cost low is an important criterion in performing run-time redistribution.

Message Packing/Unpacking Cost: At each sender, a message consists of words from different memory locations which need to be gathered in a buffer in the sending node. Typically, this requires a memory read and a memory write operation to gather data and form a compact message in the buffer. The time to perform this data gathering at the sender is the message packing cost. Similarly, at the receiving side, a message needs to be unpacked and data words need to be stored at appropriate memory locations.

Data Transfer Cost: The data transfer cost for a message consists of start-up cost and trans-

Ex.1157 / IPR2025-00916 / Page 2 of 13
Intel Corporation v. Advanced Cluster Systems, Inc.

mission cost [2]. The start-up cost can be reduced by minimizing the number of message transfer steps. The transmission cost is incurred in transferring the bits over the network. To reduce the total data transfer cost, we must minimize the number of communication steps, ensure that all processors send equal sized messages in a communication step, and avoid destination node contention in data transfers.

All of the known algorithms for data redistribution ignore one or more of the above costs. Some schemes focus only on efficient index set computation and completely ignore scheduling the communication events [4, 13, 14, 15, 21]. Based on the index of a block, these schemes focus on finding its destination processor and generating messages for the same destination. Communication scheduling is not considered. These lead to node contention in the communication steps. This in turn leads to higher data transfer costs as some nodes have to wait before sending their data. Other schemes eliminate node contention by explicitly scheduling the communication events [3, 17, 22]. Although the schemes in [3, 17, 22] have an efficient scheduling algorithm, these were designed for data redistribution on the same processor set. For redistribution between different processor sets, the Caterpillar algorithm was proposed in [11]. It uses a simple round robin schedule to avoid node contention. However, this algorithm does not fully utilize the network bandwidth. This leads to increased data transfer cost. The schemes in [5] reduce the data transfer cost. However, the schedule computation cost is significant. The bipartite graph matching used in [5] takes $O((|P|+|Q|)^4)$ time which is 100's of *msecs* for P and Q of interest. This scheduling cost can be larger than the data transfer cost, and hence, such a scheme is suitable for compile-time redistribution only. Table 1 summarizes the costs considered by the well known data distribution algorithms in the literature.

In this paper, we propose a novel and efficient algorithm for data redistribution from $cyclic(x)$ on P processors to $cyclic(Kx)$ on Q processors. Our algorithm uses *optimal* number of communication steps and *fully* utilizes the network bandwidth in each step. Optimal communication schedules are determined using the generalized circulant matrix framework. The schedule computation cost is $O(max(P,Q))$. Our implementations show that the schedule computation time is in the $\mu secs$ range when P and Q are in the range 50-100. The index set and the communication schedule are also computed extremely fast. Each processor computes its own index set and its communication schedule only using a set of equations derived from our generalized circulant matrix formulation. The schedule computation time is negligible compared with the data transfer cost for array sizes of interest. The message packing/unpacking cost is same as that of any other scheme that generates an optimal communication schedule. Thus, the total data redistribution cost is minimized in our scheme. This makes our scheme attractive for run-time data redistribution.

Our redistribution scheme has been implemented using MPI and C. It can be easily ported to various HPC platforms. Our techniques can be used for developing scalable redistribution libraries, for developing `REDISTRIBUTE` directive in HPF [1], and for developing parallel algorithms for supercomputer applications.

The rest of this paper is organized as follows. Section 2 discusses the key ideas of our redistribution scheme and explains our algorithm in detail. Section 3 reports our experimental results on the IBM SP-2. Section 4 gives some concluding remarks.

2 Key Ideas

This section describes our redistribution algorithms. Section 2.1 introduces communication tables for redistribution communication. Section 2.2 defines a generalized circulant matrix. Section 2.3

Ex.1157 / IPR2025-00916 / Page 3 of 13
Intel Corporation v. Advanced Cluster Systems, Inc.

presents our redistribution algorithm using communication tables and generalized circulant matrix.

2.1 Communication tables for redistribution

We denote the redistribution problem from *cyclic*(x) on P processors to *cyclic*(Kx) on Q processors as $\Re_x(P, K, Q)$. The input array is initially distributed on P processors as block-cyclic distribution *cyclic*(x). This can be represented as a distribution table. Each column of this table corresponds to a processor and each row index is a local block index. An element (i, j) in the distribution table represents the ith local block of the jth processor, where $i \geq 0$ and $0 \leq j < P$. The corresponding block is identified by b_g. Here, g is the global index of the corresponding block, $g = i \times P + j$.

In the redistribution, the block movement pattern repeats after some number of blocks. Such a set of blocks after which the movement pattern repeats is called a *superblock* in [3]. The period of the block movement can be easily shown as $lcm(P, KQ)$. Therefore, in computing the communication schedule as well as the index set, it is sufficient to consider only the first superblock.

In our framework, the redistribution can be conceptually considered as a table conversion process. This conversion process can be accomplished through a sequence of column and row transformations. While the column transformation is a local operation rearranging the data within each processor's memory, the row transformation corresponds to interprocessor communication. The first superblock in the initial distribution is represented as a table which is denoted as D_i. The destination processor of each block in the initial distribution table is determined by the redistribution parameters and its global block index. The communication events table can be constructed by replacing each block index in the initial distribution table with its destination processor index. This is denoted as the destination processor table (*dpt*) $\mathbf{P}$. The $(i, j)^{th}$ entry of $\mathbf{P}$ is the destination processor index of jth local block in source processor i. It is a $L_s \times P$ matrix, where $L_s = \frac{lcm(P,KQ)}{P}$. In general, the use of *dpt* itself as a schedule for communication leads to node contention, since several processors try to send their data to the same destination processor in a communication step. To eliminate node contention, the *dpt* $\mathbf{P}$ is reorganized to $\mathbf{S}$ through column transformations. $\mathbf{S}$ is called a send communication schedule. It is a $L_s \times P$ matrix as well. To maintain the correspondence between D_i and $\mathbf{P}$, the same set of column transformations is applied to D_i. In a communication step, blocks in a row of the distribution table corresponding to $\mathbf{S}$ are transferred to their destination processor specified by the corresponding entries. It is called row transformation. The received blocks in destination processors are stored in their proper memory location by another column transformation. The key idea is to choose an $\mathbf{S}$ that supports easy to compute contention-free communication schedule.

2.2 Communication schedule using generalized circulant matrix

In this subsection, we shall define the notion of a generalized circulant matrix and use this to derive an efficient and contention-free communication schedule for $\Re_x(P, K, Q)$.

Definition 1 *An $m \times n$ matrix is a* circulant matrix *if it satisfies the following property:*

1. *If $m \leq n$, row k = row 0 circularly right shifted k times, $0 \leq k < m$.*
2. *If $m > n$, column l = column 0 circularly down shifted l times, $0 \leq l < n$.*

Definition 2 *An $M \times N$ matrix is a* generalized circulant matrix *if the matrix can be partitioned into submatrices of size $m \times n$, where $M = s \cdot m$ and $N = t \cdot n$, for some s, $t > 0$ such that the*

Ex.1157 / IPR2025-00916 / Page 4 of 13
Intel Corporation v. Advanced Cluster Systems, Inc.

submatrices form a circulant matrix and each submatrix is either a circulant matrix or a generalized circulant matrix.

There are two observations about the generalized circulant matrix: *(i) the s submatrices along each diagonal are identical, and (ii) if all elements in row 0 are distinct, then in each row all elements are distinct.* Figure 1 (a) illustrates a generalized circulant matrix.

If the communication schedule table **S** is a generalized circulant matrix and if all the elements in row 0 are distinct, then it corresponds to a schedule without node contention. For example, consider $\Re_x(6,4,9)$. Figure 1 (b) shows an example where *dpt* **P** of $\Re_x(6,4,9)$ case is converted to generalized circulant matrix form **S** by column transformations.

Therefore, the generalized circulant matrix formalism and the column transformation provide a systematic framework for developing contention-free communication schedules.

2.3 Efficient Redistribution Algorithms

The redistribution problem $\Re_x(P,K,Q)$ can be classified into three communication patterns as shown in the following lemma. (See [5] for an alternative formulation for the *cyclic*(x) to *cyclic*(y) problem). Let G be $gcd(P,KQ)$.

Lemma 1 *The communication pattern induced by $\Re_x(P,K,Q)$ is: (i) a non all-to-all communication case if $G > K$, (ii) an all-to-all communication case with a fixed message size if $K = \lambda G$, where λ is an integer greater than 0, and (iii) an all-to-all communication case with different message size if $G < K$ and $K \neq \lambda G$.*

Among these three cases, the case of all-to-all processor communication with the same message size can be optimally scheduled using a trivial round-robin schedule. However, it is non trivial to achieve the same message size between all pairs of nodes in a communication step for all-to-all case with different message size. Therefore, we focus on the two cases of scheduling non all-to-all communication and all-to-all communication with different message sizes.

2.3.1 Non all-to-all communication

We briefly explain how the initial distribution table *dpt* **P** is transformed to the send schedule matrix **S**.

Given the redistribution parameters P, Q, and K, we can get the $L_s \times P$ initial distribution table D_i and its *dpt* **P**. Let $G_1 = gcd(P,K)$, $K_1 = \frac{K}{G_1}$ and $P_1 = \frac{P}{G_1}$. In the *dpt*, every K_1 row has a similar pattern. It has different destination processor index. We shuffle the rows such that similar rows are adjacent. The shuffled *dpt* $\mathbf{P_1}$ is divided into Q_1 slices in the row direction, $Q_1 = \frac{L_s}{K_1}$. It is divided into P_1 slices in the column direction. Now, *dpt* can be considered as a $K_1 \times P_1$ block matrix made of $Q_1 \times G_1$ submatrices. This block matrix is converted into a generalized circulant matrix by circular-shifting submatrices downwards and rotating individual columns within submatrix by appropriate amounts. This results in a communication schedule matrix **S**. In this procedure, the K identical values in row 0 of the *dpt* are distributed to K distinct rows, and hence, row 0 has distinct values. Since **S** is a generalized circulant matrix, all rows are distinct and we achieve a conflict-free schedule. This table conversion process is called column transformation. In the above transformation, an element is moved within its column. So, it does not incur any interprocessor communication.

Ex.1157 / IPR2025-00916 / Page 5 of 13
Intel Corporation v. Advanced Cluster Systems, Inc.

While the *dpt* is converted, the same set of transformations are applied to the data distribution table corresponding to the *dpt*. It is expensive to rearrange data in the local memory. In a real implementation, each source processor has a table which points to the data blocks packed in a communication step. This table is an index table for data to be packed. It is denoted as *send data location table* $\mathbf{D_s}$. Therefore, our scheme computes the schedule and data index set at the same time.

With some algebraic manipulations, the above procedure gives the following two equations to directly compute the individual entries of $\mathbf{S}$ and $\mathbf{D_s}$. In these equations, $i_1 = i/K_1$ denotes the quotient of integer division; $i_2 = i \bmod K_1$ denotes the remainder of the integer division; $j_1 = j/G_1$ and $j_2 = j \bmod G_1$.

$$\mathbf{S}(i,j) = \{\{n(j_1 - i_1)\} \bmod P_1 + \{(i_2 - j_2) \bmod Q_1\}P_1\} \bmod Q \quad (1)$$

$$\mathbf{D_s}(i,j) = \{m(j_1 - i_1)\} \bmod K_1 + \{(i_2 - j_2) \bmod Q_1\}K_1 \quad (2)$$

where n and m are solutions of $nK_1 - mQ_1 = 1$.

The proof of correctness of the above and the mathematical formulations will be given in the full-version of the paper. Our scheme minimizes the number of communication steps as well as avoids node contention. In each communication step, equal-sized messages are transferred. Therefore, our scheme minimizes the data transfer cost. The above formulae for computing the communication schedule and index set for redistribution are extremely efficient compared with the methods presented in [5], which use a bipartite matching algorithm. Furthermore, using our formulae, each processor computes only entries which it needs in its send communication schedule table. Hence, the schedule and index set computation can be performed in a distributed way and the total cost of computing the schedule and index set is $O(max(P,Q))$.

2.3.2 All-to-all communication

The all-to-all communication case arises if $G(= G_1G_2) \leq K$ as stated in Lemma 1, where $G_2 = gcd(P_1, Q)$. From the first superblock, the *dpt* is constructed. The *dpt* is $L_s \times P$ matrix, where $L_s = K_1Q_1$. Since $Q = Q_1G_2$ and $G_2 \leq K_1$, $L_s \geq Q$. Therefore, each processor has more entries than Q destination processors. In each column, several blocks are transferred to the same destination. The column transformations as stated in Section 2.3.1 are applied to the *dpt*. It results in a generalized circulant matrix which is a $K_1 \times P_1$ circulant block matrix and each block is a $Q_1 \times G_1$ circulant matrix. In the block matrix, the first G_2 blocks in each column are distinct. Blocks in every G_2 row have the same entries but different circular-shifted patterns. These blocks can be folded onto the block in their first row. Therefore, the first G_2 rows in the block matrix are used as a send communication schedule table $\mathbf{S}$. It is a $Q \times P$ generalized circulant matrix. Since blocks in every G_2 row are folded onto blocks in their fist row, for all-to-all communication case with different message sizes, blocks in the first $(K_1 \bmod G_2)$ rows of $\mathbf{S}$ have $\lceil \frac{K_1}{G_2} \rceil$ messages, blocks in the remaining rows have $\lfloor \frac{K_1}{G_2} \rfloor$ messages. Figure 2 shows an example of the send communication schedule table generated for all-to-all case with different message sizes $\Re_x(4,3,6)$. In this example, $K_1 = 3$ and $G_2 = 2$. The message size of blocks in the first row is two since blocks in the 3th row are folded onto the first row. If K_1 is a multiple of G_2, the message size in every step is same. This case is the all-to-all communication case with the same message size. Therefore, the network bandwidth

Ex.1157 / IPR2025-00916 / Page 6 of 13
Intel Corporation v. Advanced Cluster Systems, Inc.

is fully utilized by sending equal sized message in each communication step. The detailed proof of this procedure will be given in the full-version of our paper.

3 Results

This section shows the preliminary timing results of our experiments. These experiments were conducted on the IBM SP2. The algorithms were written in C and MPI. MPI communication primitives were used for interprocessor communication. The redistribution time was measured using the `MPI_Wtime()` call. Section 3.1 shows the experimental results for the data transfer time and section 3.2 reports the schedule computation times. The data pack/unpack time is not considered since this cost is similar to any algorithm that uses optimal number of communication steps.

3.1 Data Transfer Cost

We performed experiments for three cases of interest. They were the non all-to-all communication case and the all-to-all case with different and the same message sizes.

For the non all-to-all communication case, we performed two experimental sets. One case is $\Re_2(28, 14, 36)$ and another is $\Re_2(28, 7, 36)$. For the experiment of $\Re_2(28, 14, 36)$ case, the proposed algorithm requires half the number of steps as compared with the Caterpillar algorithm. This should lead to a 50% reduction in redistribution time compared with the Caterpillar algorithm. In the experiment, the data transfer time of our algorithm is reduced from 44.6% to 62.2% compared with the Caterpillar algorithm. The variance results from the network contention. For $\Re_2(28, 7, 36)$ case, our proposed algorithm performs the redistribution in 9 steps. Our algorithm achieved around 75% reduction. Our experimental results confirm these for two cases (see Figure 3).

For the all-to-all communication case with the same message size, both algorithms have the same performance, since both algorithms have the same number of communication steps and avoid node contention. Also, all messages are of the same size in redistribution communication. We performed the experiment for $\Re_2(28, 8, 36)$ case. In this case, the whole messages have two blocks. In the experiment, both algorithms show the same performance. (See in Figure 4.)

In the all-to-all communication case with different message sizes, we have to consider the message size in each communication step. The proposed algorithm is superior to the Caterpillar algorithm. The $\Re_2(8, 6, 10)$ case is an example of all-to-all communication case with different message sizes. For this case, both algorithms have the same number of steps (10 steps). Half the messages are two blocks while the other half are one block. In our proposed algorithm, equal-sized messages are transferred in each communication step. Therefore, during half the steps, two-block messages are exchanged while one-block messages are exchanged during the other half. The Caterpillar algorithm does not consider the size of the messages transferred. It does not attempt to schedule the communicate operations to send equal-sized messages. Therefore, the data transfer cost in each step is the time to transfer the maximum message size. In these experiments, we obtained a 25% time reduction in data transfer time compared with the Caterpillar algorithm. The experimental results are shown in Figure 5.

3.2 Schedule Computation Cost

The time for computing the schedule for both Caterpillar algorithm and our algorithm is negligible compared with the total redistribution time. Even though the schedule in Caterpillar algorithm

Ex.1157 / IPR2025-00916 / Page 7 of 13
Intel Corporation v. Advanced Cluster Systems, Inc.

is simpler than ours, it has to pay for the index computation time to find a block index. This is almost the same as our schedule computation time. Both algorithms take approximately 10 $\mu secs$ on a single node of IBM SP2.

The schedule computation cost of Desperez's algorithm [5] is much more expensive than the Caterpillar algorithm and our proposed algorithm. It is in the order of *msecs*. It is quite significant. The schedule computation cost of Desprez's algorithm increases rapidly as the number of processors increases. When P and Q are larger than 64, its schedule computation time is 100's *msecs*. On the other hand, our algorithm computes the send communication schedule efficiently. Each processor computes the entries which it needs in its send communication schedule table. Thus the schedule is performed in a distributed way. The schedule computation time is 10's $\mu secs$. Furthermore, in the case that P and Q are around 128, it takes only 100's $\mu secs$. The schedule time comparison of our scheme with Desperez's algorithm is shown in Table 2. Here, the time for our scheme includes the index computation time and the time for Desperez's algorithm is only for the schedule computation time. In Table 2, We used the procedure in [23] for bipartite matching. The times are in the same range for $P \neq Q$.

4 Conclusions

In this paper, we showed an efficient algorithm for performing redistribution from $cyclic(x)$ on P processors to $cyclic(Kx)$ on Q processors. The proposed algorithm was represented using generalized circulant matrix. Our algorithm minimizes the number of communication steps and avoids node contention in each communication step. The network bandwidth is fully utilized by ensuring that messages of the same size are transferred in each communication step. Therefore, the total data transfer cost is minimized.

The schedule and index computation costs are also important in performing run-time redistribution. In our algorithm, the schedule and the index set are computed in $O(max(P,Q))$ time. This computation is extremely fast compared with the bipartite matching scheme in [5] which it takes $O((|P|+|Q|)^4)$ time. These times are small enough to be negligible compared with the data transfer time. Therefore, our algorithm is suitable for run-time data redistribution.

Ex.1157 / IPR2025-00916 / Page 8 of 13
Intel Corporation v. Advanced Cluster Systems, Inc.

References

[1] C. Koelbel, D. Loveman, R. Schreiber, G. Steele Jr., and M. Zosel. *The High Performance Fortran Handbook*. The MIT Press, 1994.

[2] C.-L. Wang, P.B. Bhat, and V.K. Prasanna. High-Performance Computing for Vision. *Proceedings of IEEE*, 84:931–946, 1996.

[3] D.W. Walker and S.W. Otto. Redistribution of Block-Cyclic Data Distributions Using MPI. Technical Report ORNL/TM-12999, ORNL, June 1995.

[4] E.T. Kalns and L.M. Ni. Processor Mapping Techniques Toward Efficient Data Redistribution. *Proc. of International Parallel Processing Symposium*, April 1994.

[5] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro and Y. Robert. Scheduling Block-Cyclic Array Redistribution. University of Tennessee Computer Science Technical Report, UT-CS-97-349, LAPACK Working Note 120, February 1997.

[6] J. Bruck, C.-H. Ho, S. Kipnis, and Weathersby. Efficient Algorithms for All-to-All Communications in Muti-Port Message-Passing Systems. In *6th Annual ACM Symp. on Para. Alg. and Arch.*, pages 298–309, July 1994.

[7] J. Choi, J. Dongarra, and D. Walker. Parallel Matrix Transpose Algorithms on Distributed Memory Concurrent Computers. Technical Report ORNL/TM-12309, Oak Rigde National Laboratory, October 1993.

[8] J. Dongarra and *et al.* ScaLAPACK: A Portable Linear Algebra Library of Distributed Memory Computers - Design Issues and Performance. Technical Report LAPACK Working Note 95, Oak Rigde National Laboratory, 1995.

[9] J. Suh, M. Ung, and V.K. Prasanna. Parallel Implementation of Synthetic Aperture Radar on High Performance Computing Platforms. *International Conference on Algorithms And Architectures for Parallel Processing '97*, December 1997.

[10] J. Ward. Space-Time Adaptive Processing for Airborne Radar. Technical Report 1015, MIT Lincoln Lab., December 1994.

[11] L. Prylli and B. Tourancheau. Fast Runtime Block Cyclic Data Redistribution on Multiprocessors. *Journal of Parallel and Distributed Computing*, volume 45, August 1997

[12] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. Technical Report CS-94-230, University of Tennessee, Knoxville, May 1994.

[13] R. Thakur, A. Choudhary, and G. Fox. Runtime Array Redistribution in HPF Programs. In *Proc. of Scalable High Performance Computing Conference*, pages 309–316, May 1994.

[14] R. Thakur, A. Choudhary, and J. Ramanujam. Efficient Algorithms for Array Redistribution. To appear in IEEE Trans. on Parallel and Distributed Systems

[15] S. Ramaswamy and P. Banerjee. Automatic Generation of Efficient Array Redistribution Routines for Distributed Memory Multicomputers. In *Proc. of 5th Symp. Frontiers of Massively Parallel Computation, McLean, VA*, pages 342–349, Feburary 1995.

[16] S. Hiranandani, K. Kennedy, J. Mellor-Crummey, and A. Sethi. Compilation Techniques for Block-Cyclic Distributions. In *Proc. of Intl. Conf. on Supercomputing*, pages 392–403, July 1994.

[17] S.D. Kaushik, C.-H. Huang, J. Ramanujam, and P. Sadayappan. Multiphase Array Redistribution: Modeling and Evaluation. Technical Report OSU-CISRC-9/94-TR52, Ohio State University, September 1994.

[18] S.D. Kaushik, C.-H. Huang, J. Ramanujam, and P. Sadayappan. Multiphase Array Redistribution: Modeling and Evaluation. In *Proc. of Intl. Parallel Processing Symposium*, pages 441–445, 1995.

[19] S.D. Kaushik, C.-H. Huang, R.W. Johnson, and P. Sadayappan. An Approach to Communication Efficient Data Redistribution. In *Proc. of Intl. Conf. on Supercomputing*, pages 364–373, July 1994.

[20] W. Liu, W. Kostis, and V.K. Prasanna. Communication Issues in Heterogeneous Embedded Systems. In *Proc. of Workshop on Para. and Dist. Real Time Sys.*, Apr 1996.

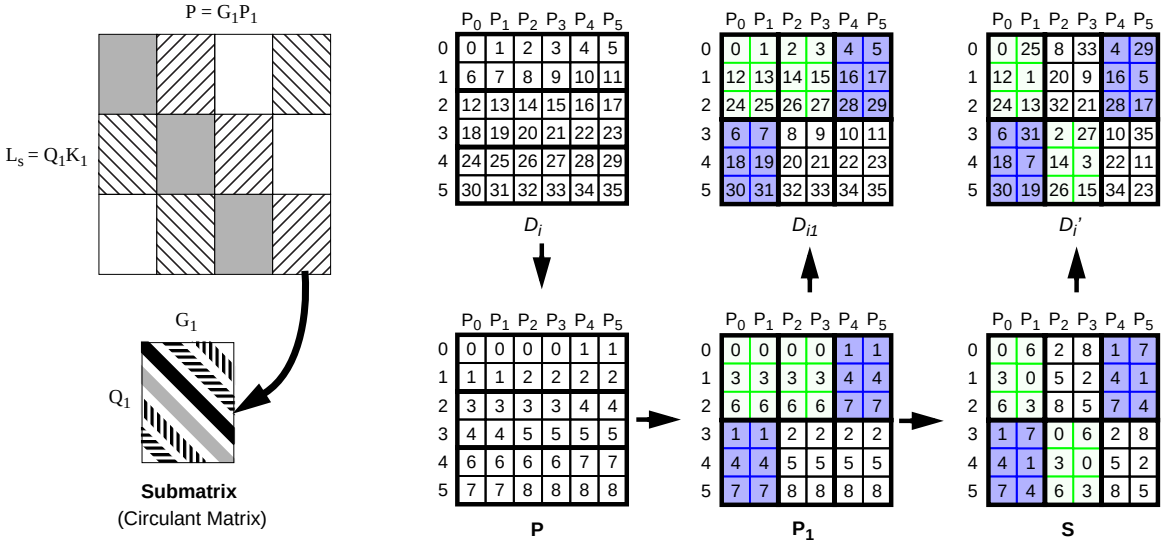
[21] Y.C. Chung, C.H. Hsu, and S.W. Bai. A Basic-Cycle Calculation Technique for Efficient Dynamic Data Redistribution. In *IEEE Tans. on Parallel and Distributed Systems*, Vol. 9, No. 4, April 1998.

[22] Y.W. Lim, P.B. Bhat, and V.K. Prasanna. Efficient Algorithms for Block-Cyclic Redistribution of an Array. In *Proc. IEEE Symposium on Parallel and Distributed Processing*, October 1996.

[23] R. Jonker and A. Volgenant. A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems. on *http://207.158.230.188/assignment.html*.

	Key Features	
	Schedule & Index Computation	**Communication**
PITFALLS [14]	• No communication scheduling • Index computation using a line segment formalism	• Node contention occurs • Does not minimize the transmission cost
BCC [21]	• No communication scheduling • Efficient index computation • Source and destination sets are same	• Node Contention occurs • Does not minimize the transmission cost
Caterpillar [11]	• Simple scheduling algorithm • Index computation by scanning the array segment	• No node contention • Does not minimize the transmission cost and the number of communication steps
Bipartite Matching Scheme [5]	• Large schedule computation overhead cost: $O((\|P\|+\|Q\|)^4)$	• No node contention • Stepwise strategy: minimizes the number of communication steps • Greedy strategy: minimizes the transmission cost
Our Scheme	• Fast schedule and index computations cost: $O(max(P,Q))$	• No node contention • Minimizes the number of communication steps and the transmission cost.

Table 1: Comparison of various schemes for array redistribution



(a) Generalized circulant matrix

(b) Example illustrating column transformation for P=6, Q=9, and K=4

Figure 1: Generalized circulant matrix and a send communication schedule table

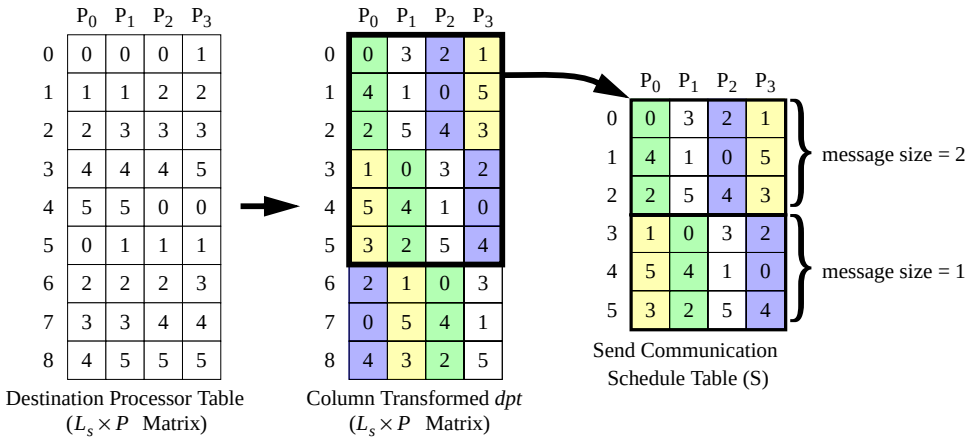


Figure 2: Example illustrating an all-to-all case with different message sizes: $\Re_x(4,3,6)$

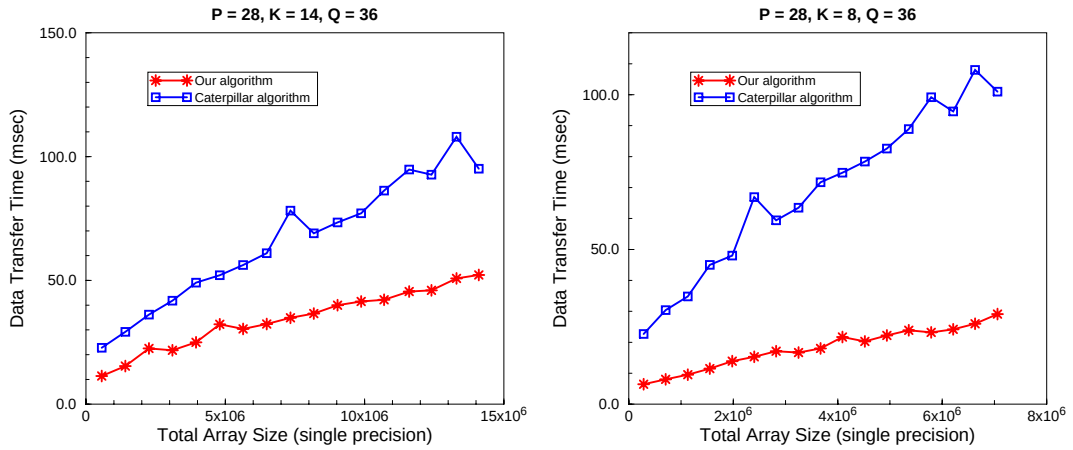


Figure 3: Experimental results for non all-to-all communication cases on the IBM SP2

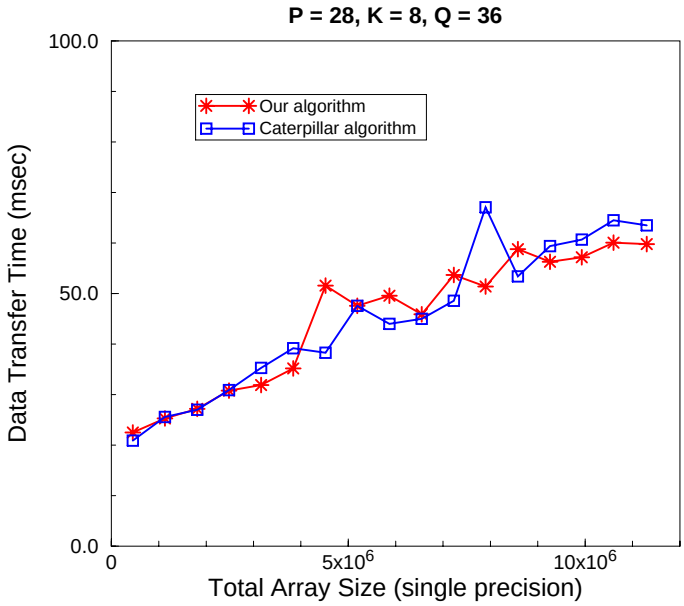


Figure 4: Experimental results for all-to-all communication case with the same message size on the IBM SP2

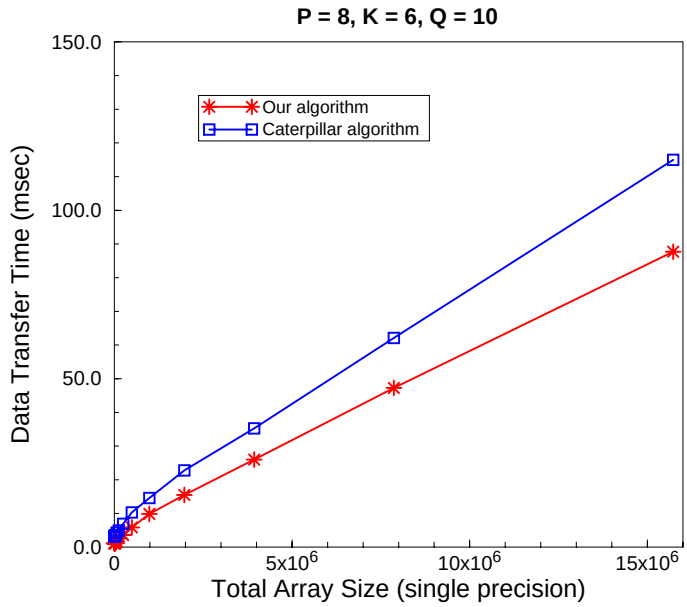


Figure 5: Experimental results for all-to-all communication case with different message sizes on the IBM SP2

P and **Q**	**K**	**Our Scheme**	**Bipartite Matching Scheme [5]**
P = 16 Q = 16	4	30.114	866.386
	8	50.866	2389.655
	12	77.692	3959.604
P = 32 Q = 32	8	51.539	7453.028
	16	94.923	16761.141
	24	146.758	25613.034
P = 64 Q = 64	16	94.245	62443.372
	32	178.569	133212.592
	48	276.587	207762.816
P = 128 Q = 128	32	175.960	534768.000
	64	345.812	1167586.304
	96	576.755	1842415.104

Table 2: Comparison of schedule computation time ($\mu secs$): We used the procedure in [23] for bipartite matching. The times are in the same range for $P \neq Q$.