

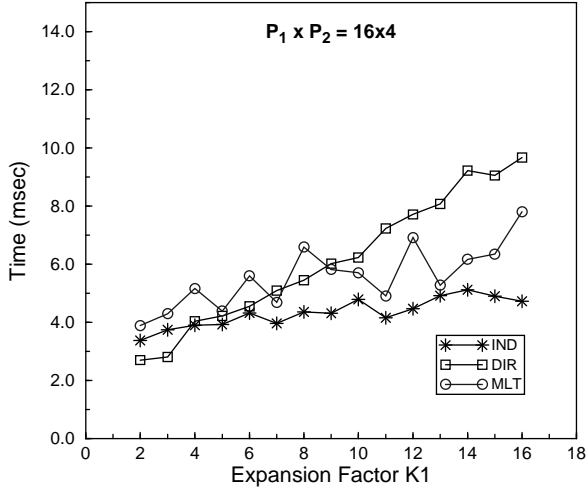
sors on the IBM SP-2. The processor topology used in the present experiments was  $16 \times 4$ . The expansion factor  $\kappa_2$  was varied from 2 to 4 and  $\kappa_1$  was varied from 2 to 16. Figure 4 shows the redistribution times as  $\kappa_1$  is varied and  $\kappa_2$  is fixed at 3 and 4. The reported times include the time for index set computation, for buffer copy operations (*i.e.*, packing and unpacking) as well as for interprocessor communication. In Figure 4, our indirect algorithm outperforms the direct and multiphase algorithms as  $\kappa_1$  and  $\kappa_2$  increase. Due to limited space, we have not shown other experimental results. In general, we have observed that for other processor topologies, the indirect algorithm is superior to the direct and multiphase algorithms as the expansion factors increase.

## 5 Conclusion

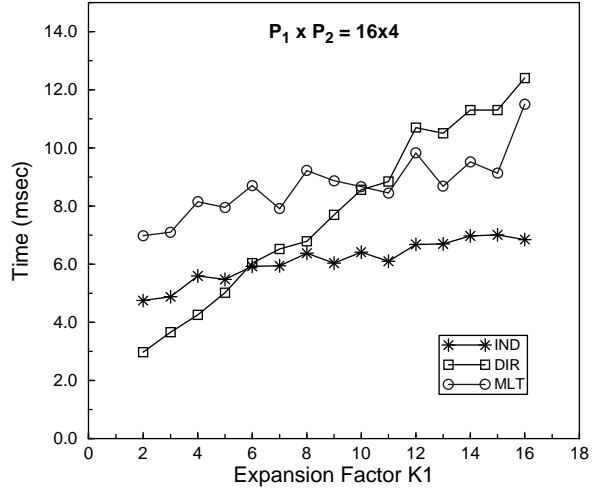
In this paper, we have shown an efficient approach for the multi-dimensional block-cyclic redistribution problem. Our communication schedules are designed using the generalized circulant matrix formalism. The indirect schedule can be viewed as the process of aligning the diagonally located entries in a circulant matrix into a vertical line in logarithmic steps by cyclic shift operations. This concept is quite general so that it includes various forms of “combine-and-forward” techniques. The generalized circulant matrix formalism also provides a systematic way of computing index sets in each communication step. Many other data redistribution patterns arise in typical HPC applications. For example, in signal processing, it is required to redistribute data from a set of source processors to a different set of destination processors. This redistribution problem is different from those we have considered in this paper, where the data is reorganized among the same set of processors. However, our algorithms can be extended to perform this redistribution. We are developing efficient communication schedules for these patterns [14].

## References

- [1] J. Bruck, C.-H. Ho, S. Kipnis, and Weathersby. Efficient Algorithms for All-to-All Communications in Multi-Port Message-Passing Systems. In *6th Annual ACM Symp. on Para. Alg. and Arch.*, pages 298-309, July 1994.
- [2] C. Koelbel, D. Loveman, R. Schreiber, G. Steele Jr., and M. Zosel. *The High Performance Fortran Handbook*. The MIT Press, 1994.
- [3] C.-L. Wang, P.B. Bhat, and V.K. Prasanna. High-Performance Computing for Vision. *Proceedings of IEEE*, 84:931-946, 1996.
- [4] D.W. Walker and S.W. Otto. Redistribution of Block-Cyclic Data Distributions using MPI. Technical Report ORNL/TM-12999, ORNL, June 1995.
- [5] J. Choi, J. Dongarra, and D. Walker. Parallel Matrix Transpose Algorithms on Distributed Memory Concurrent Computers. Technical Report ORNL/TM-12309, ORNL, Oct 1993.
- [6] J. Dongarra *et al.* ScaLAPACK: A Portable Linear Algebra Library of Distributed Memory Computers - Design Issues and Performance. Technical Report LAPACK Working Note 95, ORNL, 1995.
- [7] 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.
- [8] 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.
- [9] S.D. Kaushik, C.-H. Huang, J. Ramanujam, and P. Sadayappan. Multiphase array redistribution: Modeling and Evaluation. Technical Report OSU-CISRC-9/94-TR52, September 1994.
- [10] 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.
- [11] Y.W. Lim, P.B. Bhat, and V.K. Prasanna. Efficient Algorithms for Block-Cyclic Redistribution of Arrays. In *IEEE Symp. on Para. and Dist. Proc.*, Oct 1996.
- [12] Y.W. Lim, P.B. Bhat, and V.K. Prasanna. Efficient Data Remapping Algorithms for Embedded Signal Processing Applications. In *10th Inter. Conf. High Perf. Comp.*, 1996.
- [13] Y.W. Lim and V.K. Prasanna. Scalable Portable Implementations of Space-Time Adaptive Processing. In *10th Inter. Conf. High Perf. Comp.*, 1996.
- [14] J. Suh and V.K. Prasanna. Portable Implementation of Real-Time Benchmarks on HPC. Submitted to *Supercomputing '97*.



(a) Expansion factor  $K_2$  is fixed at 3



(b) Expansion factor  $K_2$  is fixed at 4

Figure 4: Comparison of direct, multi-phase, and indirect schedules for redistribution of 0.4 Mbytes array on a 64-node IBM SP-2.

a macroblock. The reorganization operation on the macroblock can be looked upon as a 1- $d$   $cyclic(x)$  to  $cyclic(\kappa_2 x)$  using  $P_2$  processors. The corresponding 1- $d$  processor send schedule table is  $\mathbf{P}_s^{(1)}[\kappa_2, P_2]$ .

The processor index  $j$  is represented using  $P_2$ -radix system, *i.e.*,  $j = P_2 j_1 + j_2$ , where  $0 \leq j_1 < P_1$ , and  $0 \leq j_2 < P_2$ . The communication step  $i$  is represented using  $\kappa_2$ -radix system, *i.e.*,  $i = \kappa_2 i_1 + i_2$ , where  $0 \leq i_1 < \kappa_1$ , and  $0 \leq i_2 < \kappa_2$ .

We can compute the expressions for  $\mathbf{P}_s^{(2)}[\kappa, P](i, j)$  and  $\mathbf{D}_s^{(2)}[\kappa, P](i, j)$  by using the 1- $d$  processor send schedule tables and the above representations.  $\square$

We can generalize the above index set computations to multi-dimensional redistribution:

**Corollary 3** A destination processor table

$\mathbf{P}_s^{(n)}[\kappa, P]$  in generalized circulant matrix form and the corresponding send data schedule table  $\mathbf{D}_s^{(n)}[\kappa, P]$  for  $\mathfrak{R}_B(\kappa_1, \kappa_2, \dots, \kappa_n; P_1, P_2, \dots, P_n)$  can be constructed as follows:

$$\mathbf{P}_s^{(n)}[\kappa, P](i, j) = \sum_{l=1}^n \mathbf{P}_s^{(1)}[\kappa_l, P_l](i_l, j_l) \cdot \prod_{m=l+1}^{n+1} P_m$$

$$\mathbf{D}_s^{(n)}[\kappa, P](i, j) = \sum_{l=1}^n \mathbf{D}_s^{(1)}[\kappa_l, P_l](i_l, j_l) \cdot \prod_{m=l+1}^{n+1} \kappa_m$$

where  $P = \prod_{l=1}^n P_n$ ,  $\kappa = \prod_{l=1}^n \kappa_l$ , and  $P_{n+1} = \kappa_{n+1} = 1$ .

The index of processor  $j$ ,  $0 \leq j < P$ , and communication step  $i$ ,  $0 \leq i < \kappa$ , are decomposed along each

dimension. Therefore, indices for processor  $j$  are given by  $j_n = j \bmod P_n$  and  $j_l = j \operatorname{div} (\prod_{m=l+1}^n P_m)$ , for  $l = 1, 2, \dots, n-1$ . Communication step  $i$  is decomposed into  $i_n = i \bmod \kappa_n$  and  $i_l = i \operatorname{div} (\prod_{m=l+1}^n \kappa_m)$ , for  $l = 1, 2, \dots, n-1$ .

## 4 Experimental Results

This section shows preliminary timing results from the implementations of our 2- $d$  redistribution algorithms. We are currently conducting experiments for  $\mathfrak{R}_B(\kappa_1, \kappa_2; P_1, P_2)$  and  $\mathfrak{R}_T(x_1, x_2; P_1, P_2)$ . At the time of this writing, we have obtained experimental results for  $cyclic(x_1, x_2)$  to  $cyclic(\kappa_1 x_1, \kappa_2 x_2)$  redistribution over  $P(=P_1 \times P_2)$  processors. The algorithms were coded in C, and MPI function calls were used for interprocessor communication. We measured the turn around time for redistribution using `MPI_Wtime`.

While the multiphase algorithm improves on the direct algorithm for composite values of  $\kappa_1$  and  $\kappa_2$ , it becomes the 2-phase approach when both  $\kappa_1$  and  $\kappa_2$  are prime numbers. Our indirect algorithm is uniformly applicable for both prime and composite  $\kappa_1$  and  $\kappa_2$ . Apart from the fact that our algorithms use fewer communication steps, the amount of data moved in each step per processor is less than or equal to  $\lfloor \frac{N}{2P} \rfloor$ , where  $N$  is the total number of array elements and  $P$  is the total number of processors, *i.e.*,  $N = N_1 \cdot N_2$  and  $P = P_1 \cdot P_2$ . In comparison, the multiphase approach moves the entire array in each phase and thus  $\lceil \frac{N}{P} \rceil$  data is moved by each processor during each communication phase.

The experiments were performed using 64 proces-

0	1	2	3	0	1	4	5	6	7	4	5	8	9	10	11	8	9	12	13	14	15	12	13
2	3	0	1	2	3	6	7	4	5	6	7	10	11	8	9	10	11	14	15	12	13	14	15
0	1	2	3	0	1	4	5	6	7	4	5	8	9	10	11	8	9	12	13	14	15	12	13
2	3	0	1	2	3	6	7	4	5	6	7	10	11	8	9	10	11	14	15	12	13	14	15
16	17	18	19	16	17	20	21	22	23	20	21	0	1	2	3	0	1	4	5	6	7	4	5
18	19	16	17	18	19	22	23	20	21	22	23	2	3	0	1	2	3	6	7	4	5	6	7
16	17	18	19	16	17	20	21	22	23	20	21	0	1	2	3	0	1	4	5	6	7	4	5
18	19	16	17	18	19	22	23	20	21	22	23	2	3	0	1	2	3	6	7	4	5	6	7
8	9	10	11	8	9	12	13	14	15	12	13	16	17	18	19	16	17	20	21	22	23	20	21
10	11	8	9	10	11	14	15	12	13	14	15	18	19	16	17	18	19	22	23	20	21	22	23
8	9	10	11	8	9	12	13	14	15	12	13	16	17	18	19	16	17	20	21	22	23	20	21
10	11	8	9	10	11	14	15	12	13	14	15	18	19	16	17	18	19	22	23	20	21	22	23

(a) Initial Destination Processor Table  $\mathbf{P}$ 

0	0	0	0	0	0	1	1	1	1	1	1	4	4	4	4	4	5	5	5	5	5	5	5
2	2	2	2	2	2	3	3	3	3	3	3	6	6	6	6	6	7	7	7	7	7	7	7
8	8	8	8	8	8	9	9	9	9	9	9	12	12	12	12	12	13	13	13	13	13	13	13
10	10	10	10	10	10	11	11	11	11	11	11	14	14	14	14	14	15	15	15	15	15	15	15
16	16	16	16	16	16	17	17	17	17	17	17	20	20	20	20	20	21	21	21	21	21	21	21
18	18	18	18	18	18	19	19	19	19	19	19	22	22	22	22	22	23	23	23	23	23	23	23

(b) Compact form after reordering the columns

0	18	16	10	8	2	1	19	17	11	9	3	4	22	20	14	12	6	5	23	21	15	13	7
2	0	18	16	10	8	3	1	19	17	11	9	6	4	22	20	14	12	7	5	23	21	15	13
8	2	0	18	16	10	9	3	1	19	17	11	12	6	4	22	20	14	13	7	5	23	21	15
10	8	2	0	18	16	11	9	3	1	19	17	14	12	6	4	22	20	15	13	7	5	23	21
16	10	8	2	0	18	17	11	9	3	1	19	20	14	12	6	4	22	21	15	13	7	5	23
18	16	10	8	2	0	19	17	11	9	3	1	22	20	14	12	6	4	23	21	15	13	7	5

(c) Transformed Destination Processor Table  $\mathbf{P}_s$ Figure 3: Transformations on  $dpt$  corresponding to  $\mathfrak{R}_T(4, 6)$ 

Now,  $\mathbf{P}_s$  is in the generalized circulant matrix form and each row is a permutation of the processor indices. Using Theorem 1,  $\mathfrak{R}_B(K_1, K_2; P_1, P_2)$  can be performed in  $K_1 K_2$  steps by a direct schedule and  $\lceil \log K_1' \rceil + \lceil \log G_1 \rceil + \lceil \log K_2' \rceil + \lceil \log G_2 \rceil + 1 \leq \lceil \log(K_1 K_2) \rceil + 3$  by an indirect schedule.

**Example 2:**  $\mathfrak{R}_T(P_1, P_2)$

In this example, the process topology  $P_1 \times P_2 = 4 \times 6$  is transposed to  $6 \times 4$ . In Figure 3 (a),  $\mathbf{P}$  contains  $2 \cdot 3 (= P_1' P_2')$  distinct rows and each distinct row repeats twice. In *Step 1*, a compact  $dpt$  is constructed from this initial  $dpt$ . In *Step 2*, the columns of this  $dpt$  are reordered. The compact form of  $dpt$  of size  $12/2 \times 24 (= L/G \times P)$  after reordering the columns is shown in Figure 3 (b). The  $dpt$  after column transformation in *Step 3* is shown in Figure 3 (c).

**Corollary 1**  $\mathfrak{R}_B(K_1, K_2; P_1, P_2)$  can be performed in a contention free manner in, (i)  $K_1 K_2$  communication steps using a direct schedule, (ii)  $\lceil \log K_1 K_2 \rceil + 3$  communication steps using an indirect schedule, and (iii)  $d + \lceil \frac{K_1 K_2}{2^d} \rceil$  communication steps using a hybrid schedule with  $d$  degree of indirection.

**Corollary 2**  $\mathfrak{R}_T(P_1, P_2)$  can be performed in a con-

tention free manner in, (i)  $L/G$  communication steps using a direct schedule, (ii)  $\lceil \log(L/G) \rceil + 1$  communication steps using an indirect schedule, and (iii)  $d + \lceil \frac{L/G}{2^d} \rceil$  communication steps using a hybrid schedule with  $d$  degree of indirection.

To compute the index sets of 2- $d$  redistribution, the index set computation approach for 1- $d$  redistribution is applied to each dimension. In [11], we showed that the index sets can be computed in a distributed fashion at each node without interprocessor communication for the case of 1- $d$  redistribution. The destination processor table (or processor send schedule table) and the corresponding send data schedule table for *cyclic*( $x$ ) to *cyclic*( $Kx$ ) redistribution on  $P$  processors are denoted as  $\mathbf{P}_s^{(1)}[K, P]$  and  $\mathbf{D}_s^{(1)}[K, P]$ , in which the superscript is used to represent the number of dimensions.  $\mathbf{P}_s^{(1)}[K, P](i, j)$  is the destination processor receiving data from processor  $j$  during communication step  $i$  and the corresponding  $\mathbf{D}_s^{(1)}[K, P](i, j)$  is the local index of the data to be sent from processor  $j$  during communication step  $i$  (of the direct schedule). Theorem 3 shows how to compute the index sets for  $\mathfrak{R}_B(K_1, K_2; P_1, P_2)$  of 2- $d$  case. This is generalized to  $n$ - $d$  case in Corollary 3. In the following,  $x \bmod y$  is the remainder when  $x$  is divided by  $y$ , for integers  $x$  and  $y$ .

**Theorem 3** A destination processor table  $\mathbf{P}_s^{(2)}[K, P]$  in generalized circulant matrix form and the corresponding send data schedule table  $\mathbf{D}_s^{(2)}[K, P]$  for  $\mathfrak{R}_B(K_1, K_2; P_1, P_2)$  can be constructed as follows:

$$\begin{aligned} \mathbf{P}_s^{(2)}[K, P](i, j) &= \mathbf{P}_s^{(1)}[K_1, P_1](i_1, j_1) \cdot P_2 \\ &\quad + \mathbf{P}_s^{(1)}[K_2, P_2](i_2, j_2) \\ \mathbf{D}_s^{(2)}[K, P](i, j) &= \mathbf{D}_s^{(1)}[K_1, P_1](i_1, j_1) \cdot K_2 \\ &\quad + \mathbf{D}_s^{(1)}[K_2, P_2](i_2, j_2) \end{aligned}$$

where  $0 \leq i < P$ ,  $0 \leq j < P$ ,  $i_1 = i \text{ div } K_2$ ,  $i_2 = i \bmod K_2$ ,  $j_1 = j \text{ div } P_2$ ,  $j_2 = j \bmod P_2$ ,  $P = P_1 \cdot P_2$ , and  $K = K_1 \cdot K_2$ .

**Proof Sketch:**

The proof is based on the properties observed in the proof of Theorem 2 for the case of  $\mathfrak{R}_B(K_1, K_2; P_1, P_2)$ . In *Step 1* of the proof, a block of size  $K_2 \times P_2$  is denoted as a macroblock. There are  $K_1 \times P_1$  macroblocks. The reorganization operations on  $K_1 \times P_1$  macroblocks can be looked upon as a 1- $d$  *cyclic*( $x$ ) to *cyclic*( $K_1 x$ ) redistribution using  $P_1$  processors. The corresponding 1- $d$  processor send schedule table is  $\mathbf{P}_s^{(1)}[K_1, P_1]$ . As discussed in *Step 2*, there are  $K_2 \times P_2$  blocks in

ownership of the blocks in a superblock in the initial and final distributions are denoted by  $\mathbf{S}_b$  and  $\mathbf{R}_b$  respectively, *i.e.*,  $\mathbf{S}_b(i, j)(\mathbf{R}_b(i, j))$  represents the processor which owns block  $b_{i,j}$  before(after) the redistribution. A *run* in a table refers to a longest sequence of consecutive entries which are the same. A run can wrap-around in row major order. The length of a run is the number of entries in the run.

**Case  $\mathfrak{R}_B(K_1, K_2; P_1, P_2)$ :** Initially,  $b_{i,j}$  belongs to processor  $\mathbf{S}_b(i, j)$ . After redistribution, it has to be moved to processor  $\mathbf{R}_b(i, j)$ . The initial *dpt*  $\mathbf{P}$  is given by  $\mathbf{P}(k, \mathbf{S}_b(i, j)) = \mathbf{R}_b(i, j)$  where  $k = (i \operatorname{div} P_1)K_2 + (j \operatorname{div} P_2)$ . Here,  $x \operatorname{div} y$  denotes  $\lfloor \frac{x}{y} \rfloor$ , for integers  $x$  and  $y > 0$ . The size of the *dpt* is  $K_1 K_2 \times P$ . In the following, the transformation of the initial *dpt* is described step by step. (See Example 1 below for an illustration.)

*Step 1:* Denote disjoint blocks of  $\mathbf{P}$  of  $K_2 \times P_2$  as a macroblock.  $\mathbf{P}$  can be viewed as a table consisting of  $K_1 P_1$  macroblocks, where each consecutive  $K_1$  macroblocks are the same in row major order. Thus, in  $\mathbf{P}$ , there are  $P_1$  runs and each run consists of  $K_1$  macroblocks. It can be shown that these  $K_1 P_1$  macroblocks can be reorganized into a generalized circulant matrix form.

*Step 2:* Consider the entries within a macroblock. Note that within each macroblock, there are exactly  $K_2 P_2$  distinct blocks. If we enumerate the  $K_2 P_2$  blocks in row major order, then there are  $P_2$  runs of size  $K_2$ . The  $K_2 P_2$  blocks within a macroblock can be similarly reorganized into a generalized matrix form by column transformations.

**Case  $\mathfrak{R}_T(P_1, P_2)$ :** Assume without loss of generality  $P_1 \leq P_2$ . The  $L \times P$  table  $\mathbf{P}$  is obtained by  $\mathbf{P}(k, \mathbf{S}_b(i, j)) = \mathbf{R}_b(i, j)$  and  $k = (i \operatorname{div} P_1)P_2 + (j \operatorname{div} P_2)$ .

*Step 1:* It is easy to verify that  $\mathbf{P}$  contains  $P_1 P_2'$  distinct rows and each distinct row repeats  $G$  times. By regarding these  $G$  rows as a single row, a compact *dpt* of size  $P_1 P_2' (= L/G) \times P$  is obtained.

*Step 2:* Consider the following column transformation on the compact *dpt*. First, within each column, blocks are reorganized in the increasing order of the destination processor indices. This results in  $G^2$  distinct columns. Each distinct column repeats  $P_1 P_2' (= L/G)$  times. Then, the columns are reordered so that the compact *dpt* is partitioned into sets of columns, with each partition having a distinct column. Each set has exactly  $L/G$  columns. Note that reordering the columns is not a row transformation.

*Step 3:* Consider  $L/G$  columns as a macroblock of size  $L/G \times L/G$ . The new *dpt* can be viewed as a ta-

0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	6	6	6	6	7	7
1	1	2	2	2	2	1	1	2	2	2	2	1	1	2	2	2	2	7	7	8	8	8	8
3	3	3	3	4	4	3	3	3	3	4	4	3	3	3	3	4	4	9	9	9	9	10	10
4	4	5	5	5	5	4	4	5	5	5	5	4	4	5	5	5	5	10	10	11	11	11	11
6	6	6	6	7	7	6	6	6	6	7	7	12	12	12	12	13	13	12	12	12	12	13	13
7	7	8	8	8	8	7	7	8	8	8	8	13	13	14	14	14	14	13	13	14	14	14	14
9	9	9	9	10	10	9	9	9	9	10	10	15	15	15	15	16	16	15	15	15	15	16	16
10	10	11	11	11	11	10	10	11	11	11	11	16	16	17	17	17	17	16	16	17	17	17	17
12	12	12	12	13	13	18	18	18	18	19	19	18	18	18	18	19	19	18	18	18	18	19	19
13	13	14	14	14	14	19	19	20	20	20	20	19	19	20	20	20	20	19	19	20	20	20	20
15	15	15	15	16	16	21	21	21	21	22	22	21	21	21	21	22	22	21	21	21	21	22	22
16	16	17	17	17	17	22	22	23	23	23	23	22	22	23	23	23	23	22	22	23	23	23	23

(a) Initial Destination Processor Table  $\mathbf{P}$

0	3	2	5	1	4	18	21	20	23	19	22	12	15	14	17	13	16	6	9	8	11	7	10
3	0	5	2	4	1	21	18	23	20	22	19	15	12	17	14	16	13	9	6	11	8	10	7
1	4	0	3	2	5	19	22	18	21	20	23	13	16	12	15	14	17	7	10	6	9	8	11
4	1	3	0	5	2	22	19	21	18	23	20	16	13	15	12	17	14	10	7	9	6	11	8
6	9	8	11	7	10	0	3	2	5	1	4	18	21	20	23	19	22	12	15	14	17	13	16
9	6	11	8	10	7	3	0	5	2	4	1	21	18	23	20	22	19	15	12	17	14	16	13
7	10	6	9	8	11	1	4	0	3	2	5	19	22	18	21	20	23	13	16	12	15	14	17
10	7	9	6	11	8	4	1	3	0	5	2	22	19	21	18	23	20	16	13	15	12	17	14
12	15	14	17	13	16	6	9	8	11	7	10	0	3	2	5	1	4	18	21	20	23	19	22
15	12	17	14	16	13	9	6	11	8	10	7	3	0	5	2	4	1	21	18	23	20	22	19
13	16	12	15	14	17	7	10	6	9	8	11	1	4	0	3	2	5	19	22	18	21	20	23
16	13	15	12	17	14	10	7	9	6	11	8	4	1	3	0	5	2	22	19	21	18	23	20

(b) Transformed Destination Processor Table  $\mathbf{P}_s$

Figure 2: Transformations on *dpt* corresponding to  $\mathfrak{R}_B(3, 4; 4, 6)$

ble consisting of  $G^2$  macroblocks. Within each macroblock, all entries in a row are the same. Hence, each macroblock can be converted into a circulant matrix by diagonalizing each row.

The above is illustrated in Example 2 below.  $\square$

**Example 1:**  $\mathfrak{R}_B(K_1, K_2; P_1, P_2)$

Consider the case when the block size is expanded by  $K_1 \times K_2 = 3 \times 4$  while the process topology remains fixed as  $P_1 \times P_2 = 4 \times 6$ . Figure 2 (a) shows the initial *dpt*. Figure 2 (b) shows the column transformed *dpt*  $\mathbf{P}_s$ . Following *Step 1* in the proof, we consider  $4 \times 6 (= K_2 \times P_2)$  distinct blocks as a macroblock. There are  $4 (= P_1)$  runs in the *dpt* and each run has  $3 (= K_1)$  consecutive macroblocks as shown in Figure 2 (a). These  $K_1 P_1$  macroblocks are reorganized into a generalized circulant matrix form. There are  $K_1$  identical macroblocks in a block diagonal. It takes  $\lceil \log K_1 \rceil$  communication steps to align these diagonals into vertical lines.

In *Step 2*, within a macroblock there are  $P_2$  runs of size  $K_2$ .  $P_2 K_2$  blocks in a macroblock is reorganized into a generalized circulant matrix form. Therefore, there are  $K_2'$  identical circulant matrices of size  $G_2 \times G_2$  along these block diagonals. It takes  $\lceil \log K_2' \rceil + \lceil \log G_2 \rceil$  steps to vertically align elements in each macroblock.

array in each processor's local memory are reorganized as well as the local memory location of each block is changed by redistribution. The reorganization pattern of a set of blocks repeats over all the blocks of a 2-dimensional array. Such a set of blocks is called a *superblock* [4]. Due to this periodic behavior, we only consider the first superblock in the following discussion. The first superblocks of the initial and final distributions are represented via a 2-dimensional table. In such a table, the actual local memory layout of the blocks can be depicted. By replacing the global block indices with the corresponding destination processor indices, we can represent all the required communication events using a table. The *destination processor table (dpt)* is defined as a table where the  $j^{\text{th}}$  column contains all the destination processor indices of the blocks in processor  $p_j$ , ( $0 \leq j < P$ , where  $P$  denotes the total number of processors). Note that, if the redistribution parameters (the change in the block size along each dimension and the number of processors) and the global block index assignment order are given, then each block's location in the *dpt* can be determined. Redistribution can then be conceptually viewed as a table conversion process. As shown in the following, this provides a systematic way of computing the index sets and communication schedule. By allowing either column-wise or row-wise movement of blocks, the conversion process can be decomposed into column and row transformations. The column (row) transformations permute the entries in each column (row) of a table. While column transformation is a local operation in each processor's local memory, the row transformation incurs interprocessor communication. The key idea of our approach is to choose column transformations so that the resulting *dpt* is in the generalized circulant matrix form. Once the *dpt* is in generalized circulant matrix form, a class of efficient and contention free communication schedules can be derived. We first define a generalized circulant matrix.

**Definition:** An  $m \times n$  matrix,  $m \leq n$ , is said to be a *circulant matrix* if row  $i =$  row 0 circularly right shifted  $i$  times, where  $0 \leq i < m$ .

**Definition:** Given an  $\kappa \times P$  matrix,  $\kappa \leq P$ , suppose the matrix can be partitioned into blocks of size of  $s \times t$ , where  $\kappa = m \cdot s$  and  $P = n \cdot t$ , for some  $s, t > 0$  and  $m \leq n$ . Then, the matrix is said to be a *generalized circulant matrix* if, row block  $i =$  row block 0 circularly right shifted  $i$  times ( $0 \leq i < m$ ) and each block is either a circulant matrix or is a generalized circulant matrix.

Our direct schedule performs the row transformations, by regarding the  $(i, j)^{\text{th}}$  entry of *dpt* as the des-

tinuation processor of the  $i^{\text{th}}$  communication event at processor  $p_j$ . Since every processor has a distinct destination processor in each communication event, node contention can be avoided. Our indirect schedule aligns diagonal entries vertically in logarithmic number of steps by cyclically shifting rows of *dpt*. Thus, the number of communication steps can be reduced. A hybrid schedule with degree of indirection  $d$ , performs the first  $d$  steps of the indirect schedule. Each of  $2^d$  column entries that have the same destination are then transferred using a direct schedule. Theorem 1 shows the required number of communication steps of direct, indirect, and hybrid schedules. All communication steps are contention free, *i.e.*, communication pattern is a permutation. The proof of Theorem 1 is adapted from [11].

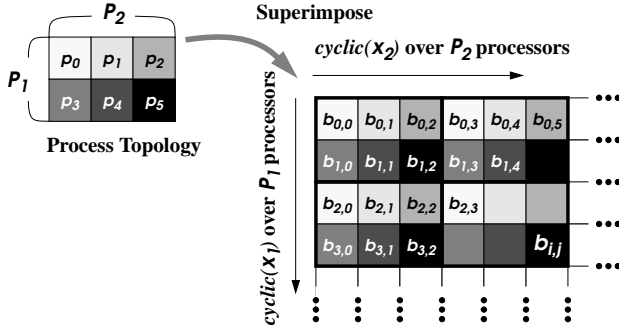
**Theorem 1** *If the destination processor table of size  $\kappa \times P$  is in generalized circulant matrix form and if every row is a permutation of  $\{0, 1, \dots, P - 1\}$ , the redistribution specified by the *dpt* can be performed in a contention free manner in (i)  $\kappa$  communication steps using a direct schedule, (ii)  $\lceil \log \kappa \rceil + 2$  communication steps using an indirect schedule, and (iii)  $d + \lceil \frac{\kappa}{2^d} \rceil$  communication steps using a hybrid schedule with  $d$  degree of indirection.*

Theorem 1 shows that the indirect schedule can perform a collective communication in  $O(\log \kappa)$  communication steps, if the *dpt* of size  $\kappa \times P$  can be transformed into a generalized circulant matrix form. Before discussing  $\mathfrak{R}_B(\kappa_1, \kappa_2; P_1, P_2)$  and  $\mathfrak{R}_T(P_1, P_2)$ , let  $G_1 = \text{gcd}(\kappa_1, P_1)$ ,  $G_2 = \text{gcd}(\kappa_2, P_2)$ ,  $K_1 = \kappa_1' G_1$ ,  $K_2 = \kappa_2' G_2$ ,  $G = \text{gcd}(P_1, P_2)$ ,  $L = \text{lcm}(P_1, P_2)$ , and  $P = P_1 P_2$ . The size of the *dpt* of  $\mathfrak{R}_B(\kappa_1, \kappa_2; P_1, P_2)$  and  $\mathfrak{R}_T(P_1, P_2)$  is  $\min(\kappa_1 \kappa_2, P) \times P$  and  $P_1' P_2' \times P$  respectively. Theorem 2 shows that the initial *dpt*'s of these cases can be transformed into the generalized circulant matrix form via operations within each column. Using Theorem 1, our algorithms perform  $\mathfrak{R}_B(\kappa_1, \kappa_2; P_1, P_2)$  and  $\mathfrak{R}_T(P_1, P_2)$  in  $O(\log(\kappa_1 \kappa_2))$  and  $O(\log(L/G))$  communication step, where  $1 < \kappa_1 \kappa_2 < P$  and  $1 < P_1' P_2' < P$ . If  $\kappa_1 \kappa_2 \geq P$  or  $L/G = P$ , then the *dpt* has size  $P \times P$ . In this case, our algorithms perform  $\mathfrak{R}_B(\kappa_1, \kappa_2; P_1, P_2)$  and  $\mathfrak{R}_T(P_1, P_2)$  in  $O(\log P)$  steps.

**Theorem 2** *The destination processor tables corresponding to  $\mathfrak{R}_B(\kappa_1, \kappa_2; P_1, P_2)$  and  $\mathfrak{R}_T(P_1, P_2)$ , can be transformed into generalized circulant matrices by column transformations.*

**Proof Sketch:**

In the following, it is assumed that the blocks and the processors are numbered in row major order. The



Matrix **A** distributed by  $(cyclic(x_1), cyclic(x_2))$

Figure 1: 2- $d$  block-cyclic distribution.

In Section 2, we present the background for multi-dimensional block-cyclic redistribution. In Section 3, we present the key ideas of our algorithms. In Section 4, we report experimental results on IBM SP-2. Section 5 concludes the paper.

## 2 Background

For the sake of simplicity, we use the 2- $d$  case to explain the multi-dimensional redistribution problems.

### 2.1 Definitions

A 2- $d$  block-cyclic distribution is specified by four parameters: block size  $x_1 \times x_2$  and process topology  $P_1 \times P_2$ . In the 2- $d$  block-cyclic distribution, a given 2- $d$  array of size  $N_1 \times N_2$  is first partitioned into blocks of size  $x_1 \times x_2$ . Let  $b_{i,j}$  denote the  $(i, j)^{th}$  block,  $0 \leq i \leq \frac{N_1}{x_1}$  and  $0 \leq j \leq \frac{N_2}{x_2}$ . These blocks are then distributed among  $P = P_1 P_2$  processors in the following manner: The blocks are distributed as  $cyclic(x_1)$  over  $P_1$  processors along the first dimension. Similarly, the blocks are distributed as  $cyclic(x_2)$  over  $P_2$  processors along the second dimension. We can view the 2- $d$  block-cyclic distribution as superimposing the process topology onto the 2- $d$  array which is partitioned into blocks. (See Figure 1)

We denote the general 2- $d$  redistribution problem as  $\mathfrak{R} : D_i(x_1, x_2; P_1, P_2) \rightarrow D_f(y_1, y_2; Q_1, Q_2)$ .  $D_i(x_1, x_2; P_1, P_2)$  is the initial distribution where blocks of size  $x_1 \times x_2$  are distributed as  $cyclic(x_1)$  ( $cyclic(x_2)$ ) over  $P_1$  ( $P_2$ ) processors along the first (second) dimension. In the final distribution  $D_f(y_1, y_2; Q_1, Q_2)$ , the blocks are similarly distributed. Usually, the process topology changes such that  $P_1 P_2 = Q_1 Q_2$ , since the array is redistributed among the same set of processors. However, this restriction is absent if, for example, task parallelism is used (see HPF-2), where array elements are redistributed between different sets of processors. In this paper, we

<sup>2</sup>For simplicity, we assume that  $x_1$  and  $x_2$  divide  $N_1$  and  $N_2$  respectively.

focus on the following cases that occur frequently:

1.  $n$ - $d$  redistribution in which the block size changes by a factor along each dimension and the process topology remains fixed. This is denoted  $\mathfrak{R}_B(K_1, K_2, \dots, K_n; P_1, P_2, \dots, P_n) : D_i(x_1, x_2, \dots, x_n; P_1, P_2, \dots, P_n) \rightarrow D_f(K_1 x_1, K_2 x_2, \dots, K_n x_n; P_1, P_2, \dots, P_n)$ .
2. 2- $d$  redistribution in which the process topology is transposed and the block size of the corresponding dimension remains fixed. This is denoted  $\mathfrak{R}_T(P_1, P_2) : D_i(x_1, x_2; P_1, P_2) \rightarrow D_f(x_2, x_1; P_2, P_1)$ .

### 2.2 Cost of Redistribution

Redistribution incurs overheads in performing index computation and in interprocessor communication. An index computation overhead for each array element is incurred in calculating its destination processor and the location of the element within that processor. Each array element can be referenced by a global index and a local index. The global index is the index of an array element from the array point of view. From the global index, we can calculate the processor at which an array element is located as well as the local memory location in that processor, *i.e.*, the local index. However, frequent computation of this information from the global index leads to significant index computation overhead. To reduce the index computation overhead, the regular structure of the block-cyclic array distribution must be utilized.

We have developed redistribution algorithms for block-cyclic redistribution based on a simple analytical model of distributed memory machines, the General purpose Distributed Memory (GDM) model [3, 12]. The model represents the communication time of a message passing operation using two parameters: the *startup time*  $T_d$  and the *unit data transmission time*  $\tau_d$ . The GDM model assumes that a processor can send at most one message and receive at most one message at a time. Messages originating from several source processors destined to the same processor, will compete with each other at the destination processor, *i.e.*, node contention occurs. The impact of node contention as communication cost is more significant than link contention in current HPC machines. We avoid node contention by choosing the schedule in each communication step to be a permutation.

## 3 Key Ideas

In the redistribution of a 2-dimensional array, each block of the array moves from the one processor to another, *i.e.*, the ownership is changed. The blocks of

Redistribution of n-d arrays		Direct Approach in [ 4 ]	Multi-phase Approach in [9]	Our Indirect Approach
<b>Block sizes of each dimension are increased by a factor of <math>K_i</math> over <math>P_1 \times P_2 \times \dots \times P_n</math> process topology</b>	For composite $K_i = \prod_{j=1}^{m_i} u_{i,j}$ ( $i=1,2,\dots,n$ )	$\prod_{i=1}^n K_i$ ( $i=1,2,\dots,n$ )	$\sum_{i=1}^n \sum_{j=1}^{m_i} u_{i,j}$	$\lceil \log(\prod_{i=1}^n K_i) \rceil$ $+ n + 1$
	For prime $K_i$ ( $i=1,2,\dots,n$ )		$\sum_{i=1}^n K_i$	
<b>Processor topology is transposed <math>P_1 \times P_2 \rightarrow P_2 \times P_1</math></b>	$L = lcm(P_1, P_2)$ $G = gcd(P_1, P_2)$	$L/G$	—	$\lceil \log(L/G) \rceil + 1$

Table 1: Number of communication steps required in various approaches to perform  $n$ - $d$  block-cyclic redistribution.

previous efforts for redistribution focus on reducing the index computation overhead, while others focus on reducing the actual communication cost of redistribution. Note that, without a proper communication schedule the redistribution overhead can be significant. Recently in [4], a communication schedule was proposed to avoid node contention. This can be classified as a direct schedule. In a direct schedule, array elements are sent directly to their destination. In [9], a multi-phase approach was proposed to reduce startup costs, using a tensor product formalism. Each phase corresponds to an intermediate block-cyclic redistribution. Within a phase, a direct schedule is used. All array elements are moved in each phase. The multiphase approach can offer superior performance when the block sizes increase by factors which are composite numbers. When the block sizes increase by prime numbers, it becomes a direct schedule. The matrix transpose problem has been studied in [5]. Both direct and indirect schedules have been proposed. However, their indirect schedule has excessive transmission costs compared with their direct schedule.

In this paper, we present a uniform framework for block-cyclic redistribution. The key idea of our approach lies in utilizing the notion of a generalized circulant matrix. Our approach fully exploits the regular characteristics of block-cyclic redistribution – the periodicities of the block assignment patterns and the underlying modulo systems arising from block-cyclic redistribution. This approach minimizes both the communication time and the index computation overhead. Efficient direct, indirect and hybrid schedules are derived from our uniform approach. The direct and indirect schedules belong to a class of hybrid schedules in our approach. Node contention is eliminated by reorganizing the communication events. Startup cost is reduced by reorganizing the communication pattern, *i.e.*, the array elements are sent to their destination through intermediate processors using an indirect schedule. The increase in the transmission costs

resulting from reorganizing the communication pattern is minimized, compared with those in [9, 1]. Our approach minimizes the redistribution overhead over a range of array size and the number of nodes by choosing an appropriate hybrid schedule.

Table 1 compares the performance of the proposed algorithms with the previous approaches. For example, consider 2- $d$  redistribution where the block sizes of each dimension are increased by factors of  $\kappa_1$  and  $\kappa_2$  assuming  $P_1 \times P_2$  process topology. Then, each processor sends its data to (as well as receives from)  $\kappa = \min(\kappa_1 \kappa_2, P_1 P_2)$  processors. Since  $\kappa = P_1 P_2$  when  $\kappa_1 \kappa_2 \geq P_1 P_2$ , without loss of generality we only consider  $\kappa_1$  and  $\kappa_2$  such that  $\kappa_1 \kappa_2 \leq P_1 P_2$ . Then, our indirect schedule performs the redistribution in  $O(\log(\kappa_1 \kappa_2))^1$  communication steps, while the direct schedule requires  $\kappa_1 \kappa_2$  communication steps. Our experimental results show that reducing the number of communication steps significantly improves the redistribution time over a range of array size per processor. In the multi-phase approach [9], the communication steps can be reduced to  $\sum_{j=1}^{m_1} u_{1,j} + \sum_{j=1}^{m_2} u_{2,j}$ , only if  $\kappa_1$  and  $\kappa_2$  can be factored such that  $\kappa_1 = u_{1,1} \cdot u_{1,2} \cdot \dots \cdot u_{1,m_1}$  and  $\kappa_2 = u_{2,1} \cdot u_{2,2} \cdot \dots \cdot u_{2,m_2}$ . If  $\kappa_1$  and  $\kappa_2$  are both prime numbers, then  $\kappa_1 + \kappa_2$  steps are required. The approach in [8] performs the 2- $d$  redistribution in two phases by performing two independent 1- $d$  redistributions along each dimension. Thus, it requires  $\kappa_1 + \kappa_2$  steps. Next, consider 2- $d$  redistribution where the process topology  $P_1 \times P_2$  is transposed. This redistribution is same as the matrix transpose problem. The direct schedule in [5] requires  $L/G$  steps, where  $L = lcm(P_1, P_2)$  and  $G = gcd(P_1, P_2)$ . Our indirect schedule performs this redistribution in  $O(\log(L/G))$  communication steps. Due to the lack of detailed information on how the approaches in [9, 8] are applied to this problem, we are unable to make detailed comparisons.

The rest of the paper is organized as follows.

<sup>1</sup>In this paper,  $\log$  is  $\log_2$ .

# Efficient Algorithms for Multi-dimensional Block-Cyclic Redistribution of Arrays <sup>\*</sup>

Young Won Lim

Neungsoo Park

Viktor K. Prasanna

Department of EE-Systems, EEB200C  
University of Southern California  
Los Angeles, CA 90089-2562  
<http://ceng.usc.edu/~prasanna>

## Abstract

*We present a uniform framework for a classical problem, redistribution of a multi-dimensional array. Using a generalized circulant matrix formalism, we derive efficient direct, indirect and hybrid contention-free communication schedules. Our indirect schedule reduces the number of communication steps significantly compared with the previous approaches. Our approach exploits the regularity of the block-cyclic redistribution to minimize the index computation overheads. For the case of 2-d redistribution, when the block size increases by factors of  $\kappa_1$  and  $\kappa_2$  along each dimension and the process topology remains fixed, our indirect schedule performs the redistribution in  $O(\log(\kappa_1\kappa_2))$  communication steps. For the case of fixed block size and the processor topology is transposed, our indirect schedule results in  $O(\log(L/G))$  communication steps. Implementations of our algorithms on the IBM SP-2 show superior performance over previous approaches.*

## 1 Introduction

Data distribution strongly influences the performance of an application on distributed memory parallel machines. In these machines, access to local data is much faster than access to remote data. These remote memory access overheads can be reduced by choosing data distributions that enhance data locality. The *block-cyclic distribution* matches the data access patterns of many High Performance Computing (HPC) applications. For example, the data distribution for radar and sonar signal processing [12, 10] can be viewed as a block-cyclic distribution. ScaLAPACK, a mathematical software for dense linear algebra computations, also uses a block-cyclic distribution for good load balance and computation efficiency [6].

In many HPC applications, the data access patterns change during the computation. Hence, it is desirable to reorganize the data distribution at intermediate points of the computation to minimize the remote access overhead. This leads to scalable performance (for an example, see [13]).

Data distributions and redistribution can be specified at varying levels of detail in application programs. When parallelizing compilers are used, the programmer specifies data distributions using high level compiler directives. For example, parallel programs developed in HPF use the **ALIGN**, **DISTRIBUTE**, and **REDISTRIBUTE** directives [2]. If applications are developed using “explicit” parallel algorithms, then data distribution and data movement between the processors are managed by the programmer. Message passing calls (such as calls to MPI) are used to perform interprocessor communication. In either approach, efficient redistribution algorithms are needed. Otherwise, the overheads of redistribution would offset the performance benefits resulting from improved data locality.

The block-cyclic redistribution problem is a classical research problem and has been well studied [8, 7, 9, 4]. In the multi-dimensional block-cyclic redistribution, a process topology, *i.e.*, a Cartesian representation of process(or) assignment to each dimension, is specified. Thus, we have to consider changes in process topology as well as in block sizes. When the block sizes change, a multi-dimensional redistribution can be performed by repeatedly applying a *1-d* redistribution algorithm along each dimension of the array. However, this approach cannot be used when the process topology changes. This case includes corner turn (matrix transpose). Among others, the redistribution of 2-*d* arrays (matrices) attracts attention, since many scientific computations involve matrices. Some of the

---

<sup>\*</sup>Work supported by DARPA under contract no. DABT63-95-C-0092.