Block-Cyclic Redistribution over Heterogeneous Networks

Prashanth B. Bhat * and Viktor K. Prasanna*† C.S. Raghavendra
Department of EE-Systems, EEB 200C
University of Southern California
Los Angeles, CA 90089-2562
Ph: (213) 740-4483
{prabhat + prasanna}@halcyon.usc.edu

C.S. Raghavendra
The Aerospace Corporation
P. O. Box 29257
Los Angeles, CA 90009
Ph: (310) 336-1686
raghu@aero.org

Abstract

Clusters of workstations and networked parallel computing systems are emerging as promising computational platforms for HPC applications. These systems consist of several heterogeneous networks such as Ethernet, ATM, and FDDI, among others.

This paper presents techniques for efficiently performing block-cyclic redistribution of arrays over such heterogeneous networks. We consider the important case where the block size changes by a factor of K. This arises very frequently in many HPC applications.

We represent the communication scheduling problem using a timing diagram formalism. Here, each interprocessor communication event is represented as a rectangle whose height represents the time to perform this event over the heterogeneous network. The communication scheduling problem is then one of appropriately positioning the rectangles in the timing diagram so as to minimize the completion time of all the communication events. We develop a heuristic algorithm with a worst case bound of 2 over the optimal completion time. The running time of the heuristic is $O(PK^2)$, where P is the number of processors, and K is the factor by which the block size changes. The schedule is computed independently by all the processors.

Our heuristic algorithm is adaptive to variations in network performance, and derives schedules at run-time, based on current information about available network bandwidth. This is important because the available bandwidth in distributed heterogeneous systems typically varies with network traffic and load. Experimental results show that our schedules always have communication times that are very close to the optimal value. Our schedules achieve significant and consistent performance improvements over well known communication schedules used for homogeneous networks.
Keywords: Mapping and scheduling, communication algorithms, block-cyclic redistribution.

1 Introduction

Due to advances in high-speed networks, the use of loosely connected distributed computing architectures for High Performance Computing applications is becoming widespread. Distributed systems are being developed using many different kinds of communication networks, such as ATM, HiPPI, and Ethernet. For example, [7] evaluated the performance of HPC applications on a cluster

*Supported by the DARPA/ITO Quorum Program through the Naval Postgraduate School under subcontract number N62271-97-M-0931.

†Partially supported by NSF under grant CCR-9317301.

of workstations interconnected with ATM and FDDI networks. Wide area networking technology has also enabled the development of *metacomputers* [18], wherein geographically distributed supercomputers and visualization devices are used in a coordinated way.

Distributed heterogeneous computing has important applications in defense environments as well. The HiPer-D project [5] investigates the use of high performance distributed computing capabilities in battle management systems on U.S. Navy cruisers. The Battlefield Awareness and Data Dissemination (BADD) program develops techniques for delivering multimedia data to mobile troops over a combination of wired and wireless networks [2].

In such scenarios, heterogeneity in the interconnection network characteristics is typical. For example, the I-WAY (Information Wide Area Year) network [9], was based on ATM technology and linked dozens of the country's fastest computers and advanced visualization machines. The overall network consisted of 10 networks of varying bandwidths, protocols, and routing technology. For scalable application performance on such a platform, efficient algorithms for collective communication are necessary. In this paper, we consider the problem of block-cyclic redistribution over such heterogeneous networks. We consider the important case where the block size changes by a factor of K. Our techniques can also be extended to other redistribution problems.

The *block-cyclic* distribution is widely used in many HPC applications to partition an array over multiple processors. For example, in signal processing applications, the block-cyclic distribution is the natural choice for radar and sonar data cubes. Many of the frequently occurring communication patterns, such as the corner turn operation [10], can be then viewed as block-cyclic redistribution operations. ScaLAPACK, a widely used mathematical software for dense linear algebra computations, also uses a block-cyclic distribution for good load balance and computation efficiency [4]. Matrix transpose operations, which often occur in linear algebra computations, are a special case of the block-cyclic redistribution. HPF provides directives for specifying block-cyclic distribution and redistribution of arrays.

The problem of block-cyclic redistribution in a tightly-coupled homogeneous parallel system has been well researched. In [11], we developed a uniform framework to derive efficient communication schedules for this problem. We presented direct, indirect, and hybrid communication schedules with the help of our framework. The heterogeneity in the network makes it necessary to develop new communication scheduling techniques. An important characteristic of the distributed computing platforms discussed above is that the available bandwidth of the heterogeneous network could undergo considerable variations with time. The bandwidth depends upon factors such as network traffic and load. The algorithm that we present in Section 4 is adaptive to such variations in network performance. The schedule is derived at run-time, based on current information about available network bandwidth. In contrast, techniques developed for block-cyclic redistribution over a homogeneous network have a fixed schedule.

The rest of the paper is organized as follows. Section 2 discusses the characteristics of the block-cyclic redistribution communication pattern. Section 3 discusses some previous research efforts on block cyclic redistribution. Section 4 introduces our communication model for the heterogeneous network and presents our algorithm for block-cyclic redistribution. Section 5 evaluates the communication performance of our algorithm. Section 6 concludes the paper.

2 The Block-Cyclic Redistribution Problem

The block-cyclic distribution of an array can be defined as follows [22]: given P processors, an array of size N, and a block size x, the distribution first partitions the array elements into contiguous

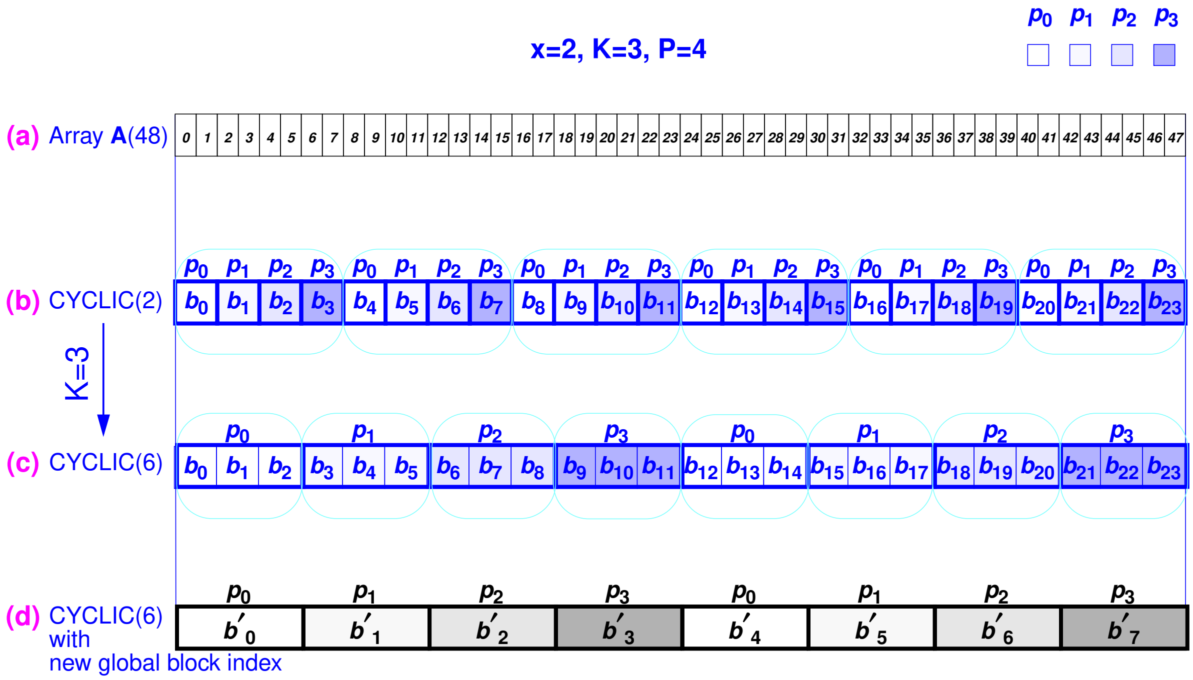

Figure 1: Redistribution from *cyclic*(2) to *cyclic*(6) on 4 processors

blocks of x items each. b_i is the i^{th} block, $0 \leq i < \frac{N}{x}$ [1]. The blocks are then assigned to processors in a round robin fashion so that b_i is assigned to processor $(i \bmod P)$. We denote a block-cyclic distribution of block size x as $cyclic(x)$. The block-cyclic data redistribution problem consists of reorganizing an array from one block-cyclic distribution to another. The most frequently encountered version of this redistribution problem is the $cyclic(x)$ to $cyclic(Kx)$ redistribution, which is the problem we consider in this paper. We denote the $cyclic(x)$ to $cyclic(Kx)$ redistribution among P processors as $\Re_x(K, P)$.

Figure 1 shows the example of $\Re_2(3, 4)$. The array **A** which has $N = 48$ elements is shown in Figure 1(a). Figure 1(b) shows the initial distribution, $cyclic(2)$. Here, b_i is of size 2 elements, and has a global block index i. The blocks are assigned to $P = 4$ processors in a round robin fashion. If the block size is increased by a factor of $K(= 3)$, *i.e.*, the new block size becomes 6, each set of three consecutive blocks becomes a new block, as shown in Figure 1(c) and (d).

Block-cyclic redistribution consists of three main phases:

1. **Index set computation and message generation:** Each processor computes indices of array elements that are to be communicated with the other processors, as well as the destination processors of such array elements. The elements are then packed into message buffers, one for each destination processor.

2. **Communication scheduling:** A given processor contains messages for a total of K processors, and will also receive messages from K processors. These messages must be scheduled so as to reduce contention and overall communication time.

[1] For simplicity, we assume that x divides N.

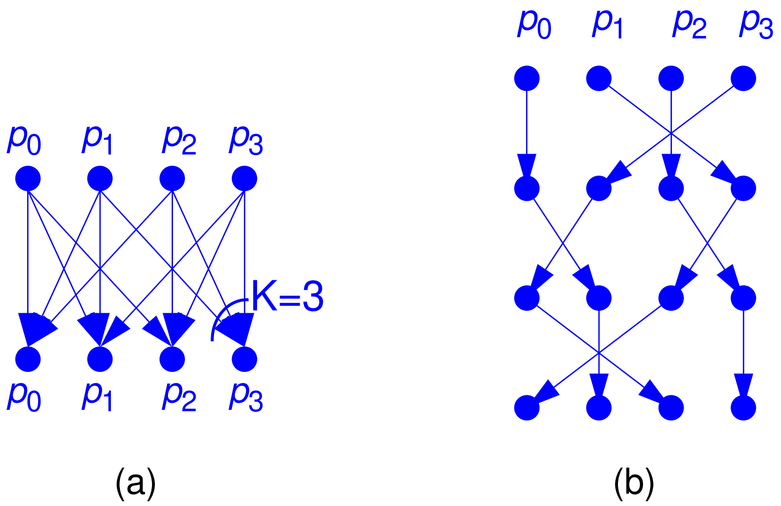

Figure 2: (a)Communication pattern for $\Re_x(3,4)$ (b)Contention-free schedule

3. **Interprocessor communication:** The processors send and receive messages in the order specified by the communication schedule. This phase incurs software start-up overheads for invocation of the send and receive system calls, transmission costs for sending data over the interconnection network, and overheads due to node contention.

Some of the previous research efforts do not perform communication scheduling. The interprocessor communication phase can therefore become very inefficient, due to node contention.

The interprocessor communication pattern of $\Re_x(K,P)$ can be represented by a communication graph, shown in Figure 2(a). Each edge represents a message that is to be sent between the corresponding processors. Note that each processor $P_i, 0 \leq i < P$ must send messages to K destinations, and receive messages from K sources. For given values of x, K, and P, the position of edges in this graph and the array indices corresponding to each edge are computed during the index computation phase.

Figure 2(b) shows an example of a K-step communication schedule for $\Re_x(3,4)$. Here, the communication pattern of Figure 2(a) is broken down into a series of contention-free communication steps. In [11], we have developed techniques for the communication scheduling phase. In a system with a homogeneous network, all the communication events within any step of Figure 2(b) would take the same amount of time. However, when the network links are heterogeneous, the time taken for each message varies with the available network bandwidth between the corresponding processors. Due to the non-uniformity in message communication times, idle cycles would be introduced in the schedule of Figure 2(b) if it was used without modification. It is therefore necessary to develop new communication scheduling techniques for cyclic redistribution over heterogeneous networks. Section 4 presents our new algorithms for this redistribution problem.

3 Related Work

Communication performance in the presence of multiple heterogeneous networks has been investigated in [7, 8]. Experiments were performed on a local cluster of workstations, interconnected with ATM, Ethernet, and Fibre-Channel networks. The performance characteristics of each of the networks were first evaluated by sending messages of various sizes over the particular network. These characteristics were used to choose a suitable technique for data communication. The *Performance Based Path Selection (PBPS)* technique selects one of the networks to be used for a communication event, depending on the size of the message. The *Aggregation* technique uses multiple networks at the same time, by breaking up the message into multiple parts and sending these parts over different networks. The sizes of the individual parts depend inversely on the speed of the network over which the part is transmitted. These techniques were experimentally tested, and performance results were reported in [7, 8]. However, this research only considered point-to-point communication between a pair of nodes in the system. In comparison, our paper investigates the collective communication pattern of block-cyclic redistribution.

The Management System for Heterogeneous Networks (MSHN) [14] project at Naval Postgraduate School, USC, and Purdue University is designing and implementing a Resource Management System (RMS) for distributed heterogeneous and shared environments. MSHN assumes heterogeneity in resources, processes, and QoS requirements. Processes may have different priorities, deadlines, and compute characteristics. The goal is to assign processor and network resources to individual applications so that their QoS requirements are satisfied. In this context, data staging techniques for distributed systems with heterogeneous networks have been considered [19].

Block-cyclic redistribution over a homogeneous network has been well studied. Many research efforts have been focused on efficiently performing the index computation phase. In [20, 21], Choudhary *et. al.* present efficient index computation algorithms for $\Re_x(K,P)$, when $P \bmod K = 0$. They also consider the redistribution from $cyclic(x)$ to $cyclic(y)$, for general x and y, using gcd and lcm methods. These are two phase algorithms where $cyclic(x)$ is first redistributed to $cyclic(m)$, followed by the redistribution of $cyclic(m)$ to $cyclic(y)$. Here, m can be gcd or lcm of x and y. It is shown in [21] that multidimensional arrays can be redistributed by applying these algorithms to each dimension of the array separately.

In [15], Banerjee *et. al.* use a line segment formalism to represent a $cyclic(x)$ distribution. The array elements that map to a processor are represented as a set of strided line segments. Using this formalism, the array elements to be exchanged between a pair of processors is computed by the intersection of the respective line segments. The technique can compute index sets for arbitrary source and target processor sets and multidimensional arrays.

Sadayappan *et. al.* [6] and Walker *et. al.* [22] have developed algorithms for the communication scheduling phase. In [22], a K step schedule is given for $\Re_x(K,P)$. At each step, processors exchange data in a contention-free manner: each processor sends data to exactly one processor and receives data from exactly one processor. A similar communication schedule was developed in [6].

In [11, 12], three classes of techniques were presented for $\Re_x(K,P)$: direct, indirect, and hybrid. Efficient algorithms were developed for reducing both the index computation and communication overheads. In the direct approach, a block is sent directly from a source processor to its destination without being sent to intermediate processors. The direct approach performs the $\Re_x(K,P)$ communication in K communication steps. In each step, each processor moves $\frac{1}{K}$ of its portion of the array. The indirect approach performs $\Re_x(K,P)$ in atmost $\lceil \log_2 K \rceil + 2$ steps. Here, the array elements are communicated in a "combine and forward" manner. The reduction in the number of communication steps comes at the expense of increased overall data traffic. At every step, each processor moves half of its portion of the array. The hybrid approach is a combination of the direct and indirect approaches.

In [3], communication schedules are developed for the general redistribution problem of $cyclic(r)$ to $cyclic(s)$. The array is initially distributed in $cyclic(r)$ over a set P of processors. The final distribution is $cyclic(s)$ over a different set Q of processors. Graph matching techniques are used to develop communication schedules in this work. The communication pattern is represented by a bi-partite graph, and a series of matchings are found in this graph. Each matching determines one step of the communication schedule. Two variations are explored: maximum weight matching at each step, and maximum cardinality matching at each step. The communication schedules developed by both techniques are direct contention-free schedules. These techniques have two important drawbacks: (i) The communication scheduling phase is expensive, due to the use of graph matching algorithms, and (ii) All processors are synchronized after each step in the interprocessor communication phase. This increases the interprocessor communication time. The communication scheduling technique that we present in Section 4 can be extended to this general redistribution problem in a straightforward way.

4 Our Communication Scheduling Approach

We first discuss the assumptions and communication model that we shall use to analyze our communication schedule. Section 4.3 presents our communication scheduling algorithm.

4.1 Communication Model

Consider a system of P processors interconnected by several heterogeneous networks. Each network could interconnect all the processors of the system, or only a subset of the processors. We can model such a network as a completely connected virtual network with heterogeneous performance between each pair of processors. Thus, the path between any pair of processors can be modeled as a single link, with the effective performance of the heterogeneous path. Techniques to aggregate the performance of different networks into a single virtual network have been considered, and are an active area of research [7]. We model the communication performance of the path between a pair of processors P_i and P_j by a start-up cost $T_{i,j}$ and a data transmission rate $B_{i,j}$. Thus, to send a m byte message between P_i and P_j, the time taken is $T_{i,j} + \frac{m}{B_{i,j}}$. When the message sizes are large, the data transmission cost is the dominating component, and the start-up cost can be ignored [2]. Typical values for the start-up cost could be in the range of 10 to 50 μ s, while typical values for the bandwidth could be in the range of a few Mb/s to hundreds of Mb/s.

We assume that a node is allowed to simultaneously participate in at most one send and one receive operation. When a node has multiple messages to send, it performs these send operations one after another. Current hardware and software do not easily enable multiple distinct messages to be transmitted simultaneously. If multiple nodes simultaneously send to any node P_j, these messages are received one after the other at P_j. We say that node contention occurs at P_j. The validity of this assumption can be seen by examining the events involved in a message transmission from P_i to P_j. A control message is first transmitted by P_i. The actual data is sent only after this control message is acknowledged by P_j. If P_j is busy receiving from a different node, it sends the acknowledgement to P_i only after completing the previous receive operation. We do not consider the use of wild-card non-blocking receives. Although this can allow a processor to simultaneously wait for many receives, large buffer space overheads are incurred.

4.2 Timing Diagrams

Communication schedules can be conveniently represented by timing diagrams. An example of a timing diagram for $\Re_x(3,4)$ is shown in Figure 3. A timing diagram consists of P columns, one

[2]We make this approximation in our experiments.

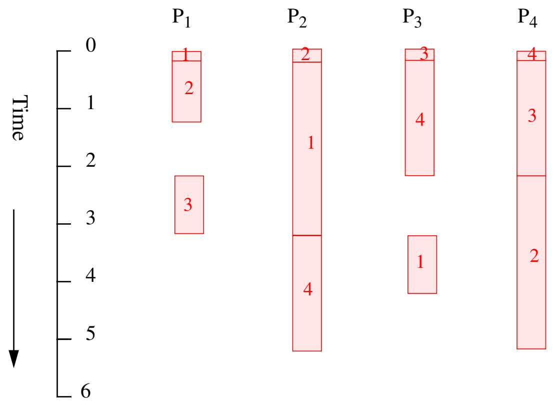

Figure 3: A communication schedule for $\Re_x(3,4)$.

per processor. The vertical axis represents time. The communication events in column i represent the messages sent from processor P_i. The rectangle labeled j in column i represents the message sent from P_i to P_j [3]. The height of the rectangle denotes the time for the communication event. The width of the rectangle does not have any significance. Once the message sizes and T_i and B_i parameters for all processors are known, the heights of all the rectangles can be determined. Thus, the timing diagram inherently absorbs the heterogeneity in network parameters and message lengths.

Any communication scheduling algorithm must determine the positions of the communication events so that an efficient schedule is found. The goal is to find the schedule that has the minimum *completion time, i.e.,* the time at which the last communication event is completed. Since a node cannot send multiple messages simultaneously, no overlap is allowed among any of the rectangles in a column. Similarly, since multiple simultaneous receive events are not permitted at a processor, all the rectangles with the same label j must have mutually disjoint time intervals. Thus, the completion time of the schedule cannot be less than the summation of send times or receive times at any processor, whichever is larger. This quantity is therefore a *lower bound* on the completion time of any schedule.

4.3 Our Scheduling Algorithm

Each processor first performs the index computation and communication scheduling phase independently. Each processor computes the entire schedule. During the interprocessor communication phase, each processor performs its send and receive operations in the sequence specified by the communication schedule. Initially, a single non-blocking receive and a non-blocking send is posted by each processor. If the receive finishes first, the next receive operation is immediately posted. If the send finishes first, the next send operation is initiated. This continues until all the communication events have been completed.

We now discuss the communication scheduling phase in further detail. The communication matrix **C**, which represents the time for each point-to-point communication event, is computed based on the values of P, K, N, and the network performance parameters. $\mathbf{C}[i,j]$ is the height of

[3] A receive schedule can be similarly constructed, where the communication events in column i represent messages received by processor P_i.

the rectangle labeled i in column j of the timing diagram.

It can be shown that the problem of finding the optimal communication schedule is NP-complete [4]. We have therefore developed a heuristic algorithm for this problem.

Each processor is considered as two independent entities, a sender and a receiver. The following data structures are maintained by the algorithm:

- For each sender $i, 0 \leq i < P$, a set R_i of receivers is maintained. These are the receivers to which i must send a message sometime during the schedule. For the $\Re_x(K, P)$ communication pattern, each set R_i will initially consist of K elements. The R_i's are obtained from the communication matrix in a straightforward way. These are the row indices of the non-zero elements in column i of $\mathbf{C}$.

- The P-element arrays *sendavail* and *recvavail* contain information about the availability of the corresponding senders and receivers. For example, the i^{th} element of *sendavail* specifies the earliest time at which sender i can participate in future send operations. All elements of both these arrays are initialized to 0.

The algorithm then computes the schedule as follows:

- Whenever a sender i becomes available (at time t, say), its receiver set R_i is scanned. If a receiver j is found which is available at time t or earlier, the communication event from i to j is scheduled. *sendavail[i]* and *recvavail[j]* are assigned the value $t + \mathbf{C}[j, i]$, since the sender i and receiver j will be busy until this time. Further, j is deleted from R_i. On the other hand, if none of the receivers in R_i is available at time t, *sendavail[i]* is assigned the value of the completion time of the next communication event in the partial schedule. Idle cycles are inserted in sender i's schedule until this time. When the next communication event causes a receiver to become free, the algorithm will again attempt to schedule a communication event on sender i.

- If multiple senders become available at the same time (for example, at time 0), they are processed in an arbitrary order. However, all senders that become available at time t are processed before any senders that become available at a later time. The algorithm maintains a list of senders in increasing order of their time of availability.

- Whenever a sender is finished with all its operations, it is deleted from this list. The algorithm terminates when all the senders are thus deleted.

The total number of communication events to be scheduled is PK. The scheduling of each event takes $O(K)$ time, since the elements of the corresponding receiver set must be scanned. The algorithm therefore runs in time $O(PK^2)$. An unoptimized version of our algorithm running on a single node of the Cray T3E executed in about 70 ms for $P = 64$ and $K = 63$. This represents the cost of the communication scheduling phase. The completion time of the schedule is a measure of the cost of the interprocessor communication phase. This will be discussed in Section 5.

Observe that the algorithm is a greedy one. At any time a sender is free, the heuristic assigns a communication event to any of the elements in its receiver set. Idle cycles are inserted in a sender's schedule only if none of its potential receivers are available.

Lemma: *The heuristic algorithm is guaranteed to find a communication schedule whose completion time is within twice the optimal.*

[4] This proof will be included in the final version of the paper.

Proof: Assume, without loss of generality, that the last sender to finish all its transmissions is i. Let j be its last receiver in the schedule, *i.e.,* j is the receiver that i sends its last message to. It can be deduced that during the idle cycles in sender i's schedule, receiver j must have been always busy. If this were not the case, the greedy algorithm would have scheduled the communication event from i to j at this time. Thus, we can conclude that the sum of the idle cycles in sender i's schedule is bounded by the total communication time for events incident at receiver j, *i.e.,* the sum of elements in row j of $\mathbf{C}$. The completion time is the sum of idle cycles in sender i and the total time for send events from sender i, *i.e.,* the sum of elements in column i of $\mathbf{C}$. Thus, the completion time is at most the sum of a row and a column in the communication matrix $\mathbf{C}$, and is hence within twice the lower bound. □

Our communication problem has some similarities to the open shop scheduling problem [1, 16]. Here, a set of M machines execute a set of N jobs. Each job J_i has a task to be executed on every machine M_j, the execution time for which is given by $t_{i,j}$. The tasks may be executed in any order. However, atmost one machine may process a given job at any time. Also, atmost one job may be processed by a given machine at any time. The goal is to schedule the tasks on the machines to minimize the completion time. The problem is known to be NP-complete, and an algorithm similar to the above greedy heuristic has been used for the open shop problem as well [17].

It is important to note that our algorithm is *adaptive* to network performance parameters. The derived communication schedule depends on the entries of $\mathbf{C}$, which are in turn dependent on network latency and bandwidth. It is therefore feasible to perform the scheduling at runtime, using current values of these parameters. Since network loads could typically undergo changes from one execution to another, this adaptivity is very useful. In comparison, previous redistribution algorithms for homogeneous networks do not provide such adaptivity. The same communication schedule is used irrespective of network load and bandwidth. Such a "fixed" communication schedule can incur penalties in the form of idle cycles, and therefore result in a longer interprocessor communication phase.

5 Experimental Results

In this section, we evaluate the performance of our heuristic scheduling technique. We measure the time for the interprocessor communication phase. We compare this with the time for the interprocessor communication phase of the *direct* communication schedule. The direct schedule is a fixed communication schedule which has been shown to be effective in systems with homogeneous networks. It can be derived from the formalisms of [6], [11], and [22].

We have developed a simulator that accepts as input the network performance parameters, and the parameters of the cyclic redistribution problem. The simulator then derives the communication schedule, and calculates the expected completion time. We also implement the communication schedules on a Cray T3E, and measure the time taken to perform the redistribution. Our experimental methodology is summarized in the following key steps:

1. For an input value of P, our simulator first generates performance parameters for the heterogeneous network. The simulator accepts as input the minimum and maximum bandwidth values, and randomly generates bandwidth values in this range for every processor pair.

2. For given input values of K and N, the communication matrix $\mathbf{C}$ is then generated. The lower bound on any communication schedule is calculated as the maximum sum over all the rows and columns of $\mathbf{C}$.

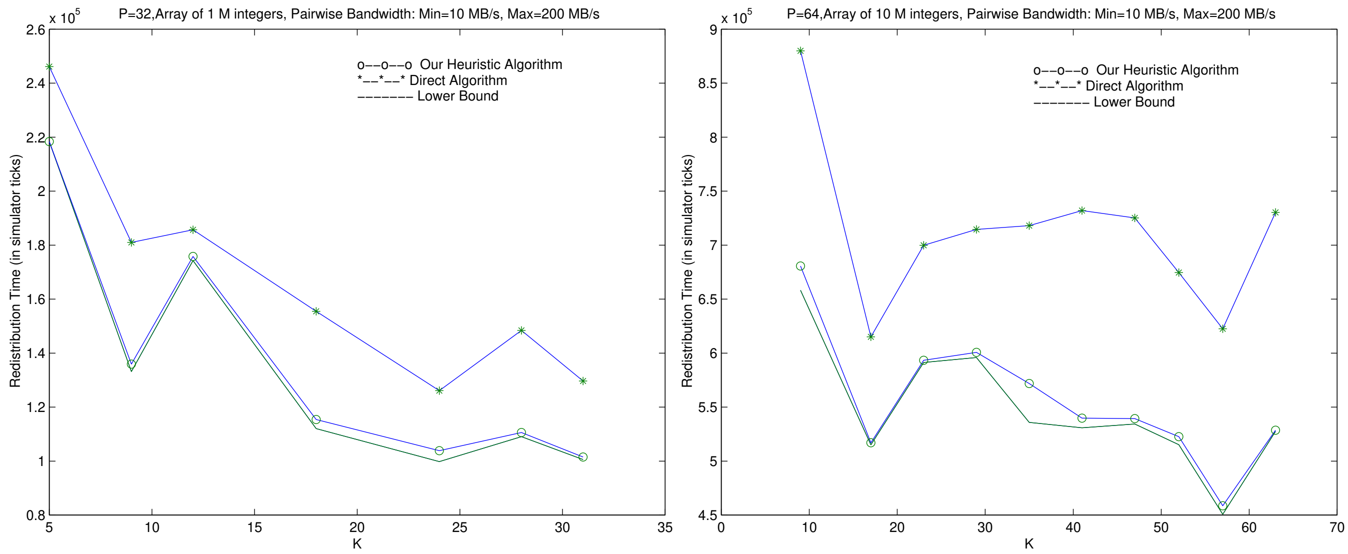

Figure 4: Simulation results: lower bound, heuristic schedule, and direct algorithm for 32 and 64 processors.

3. Our heuristic scheduling algorithm described in the previous section is then executed, and the sender and receiver schedules at each processor are computed. The simulator also computes the estimated completion time of the schedule.

4. We implement the schedule generated in Step 3, on a Cray T3E. Although the network of the Cray T3E is homogeneous, we can simulate the heterogeneous network of Step 1 by scaling the sizes of the messages appropriately. Thus, if the heterogeneous network has a bandwidth of B_1 between nodes i and j, and if B_2 is the node-to-node bandwidth of the Cray T3E, then we scale the message size between node i and j by a factor $\frac{B_2}{B_1}$. Each point-to-point event in the communication schedule is implemented by using MPI send and receive calls.

5. The communication performance of our heuristic schedule is compared with that of the direct schedule. We execute both algorithms on the Cray T3E for various values of P and K. The communication times are measured and tabulated.

Figure 4 shows the estimated completion times for the heuristic and direct schedules, as calculated by the simulator. The lower bound on the completion time is also shown in these figures. It can be seen that the completion time of our heuristic algorithm is always within 2 - 10 % of the lower bound.

Figures 5 to 8 show the timing results of experimental implementations of our heuristic schedule and the direct schedule on the Cray T3E. The simulated heterogeneous network has node-to-node bandwidths in the range of 10 MB/s to 200 MB/s. The Cray T3E has a bandwidth of 150 MB/s, which is the value we use for B_2 in Step 4 above. We can conclude from these plots that our heuristic schedule achieves significant and consistent performance improvements over the direct schedule. The communication time of the heuristic schedule is lower than that of the direct schedule by 10% to 60%.

We observe that, for a fixed value of P, the communication time for both the schedules varies considerably with the value of K. This phenomenon is not observed in [11], where the direct

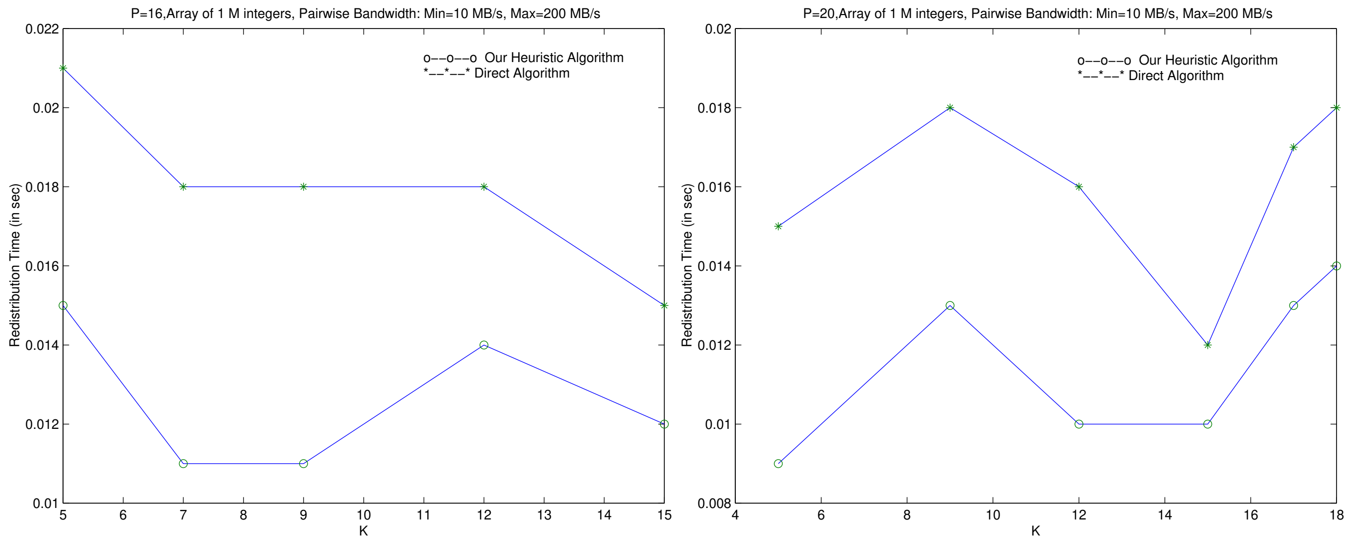

Figure 5: Communication times of our Heuristic algorithm and the Direct algorithm on 16 and 20 processors of the Cray T3E.

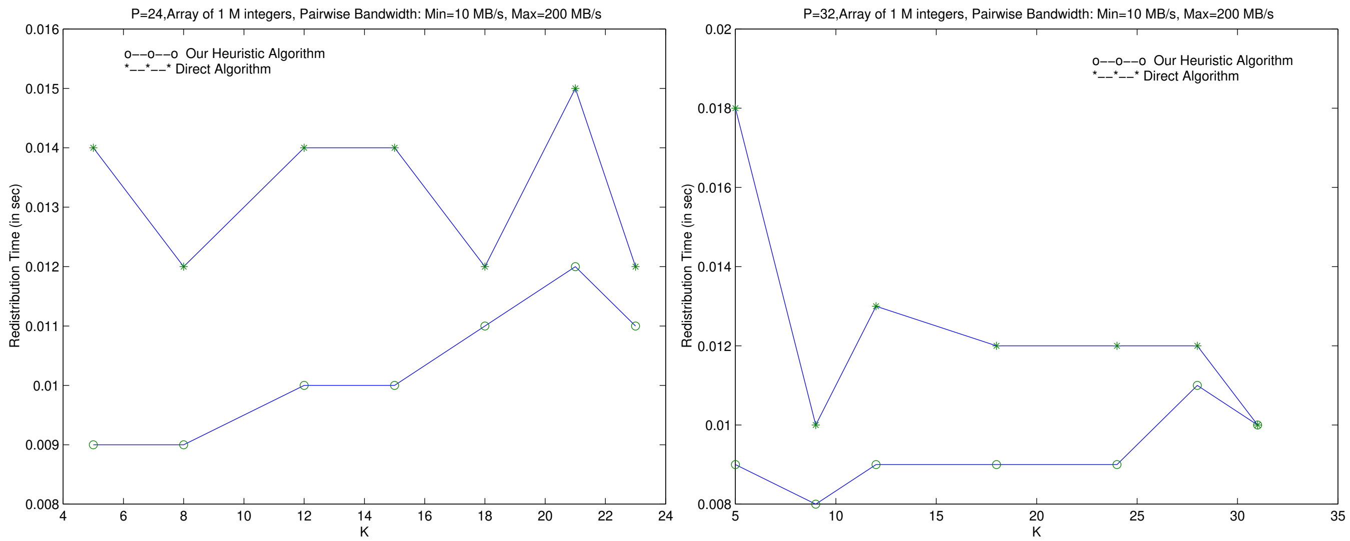

Figure 6: Communication times of our Heuristic algorithm and the Direct algorithm on 24 and 32 processors of the Cray T3E.

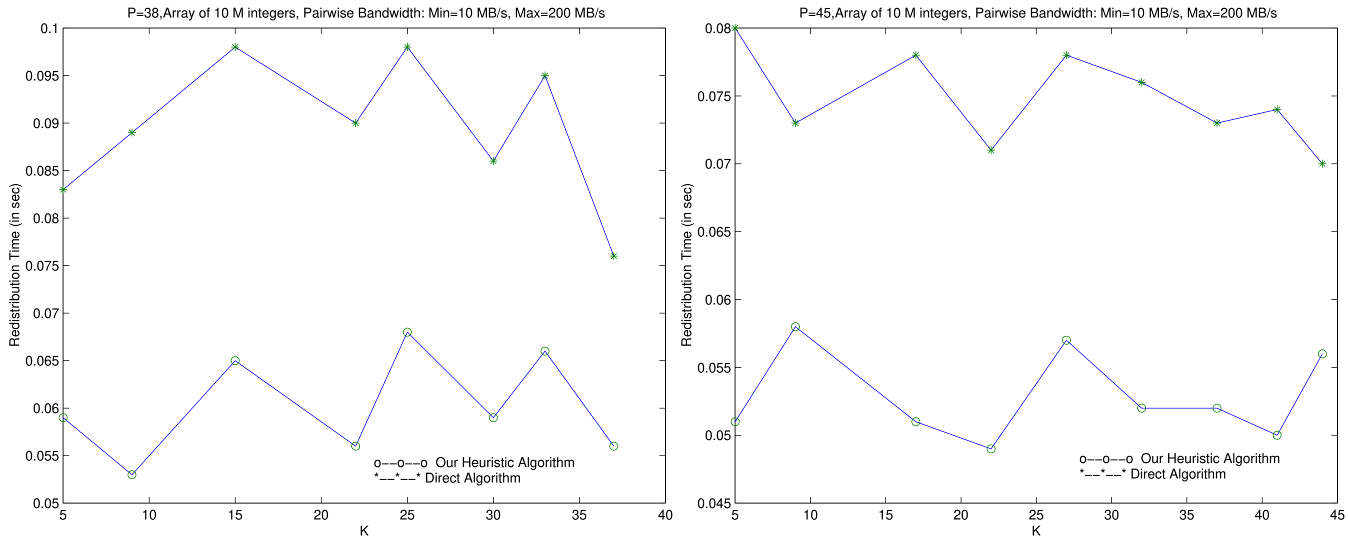

Figure 7: Communication times of our Heuristic algorithm and the Direct algorithm on 38 and 45 processors of the Cray T3E.

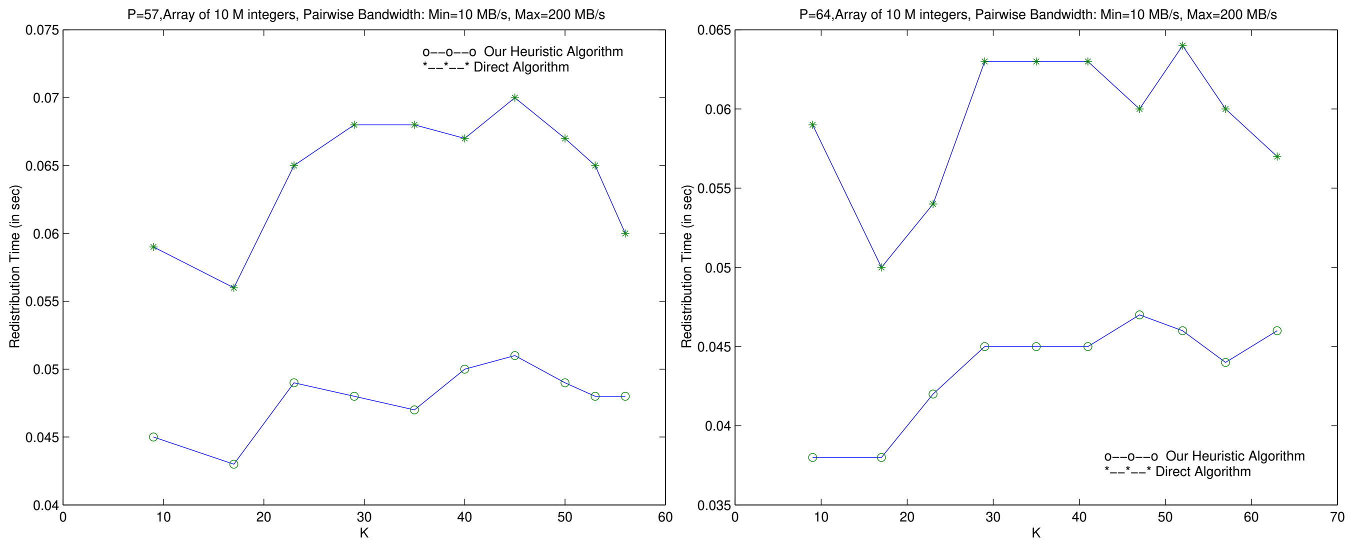

Figure 8: Communication times of our Heuristic algorithm and the Direct algorithm on 57 and 64 processors of the Cray T3E.

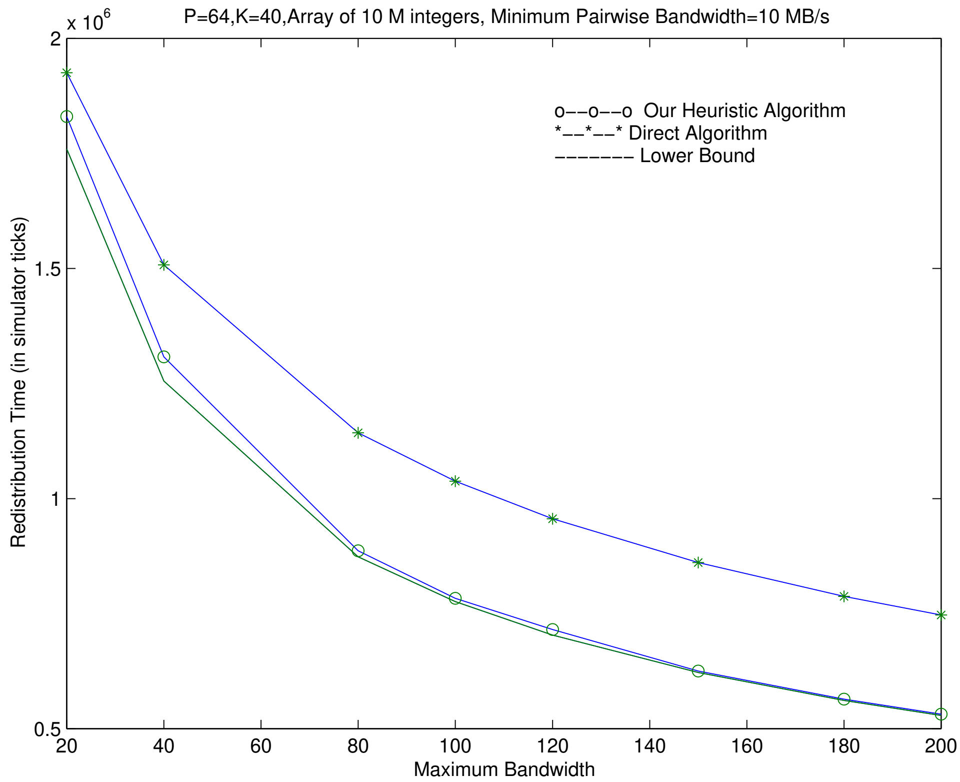

Figure 9: Communication performance of the direct algorithm and our heuristic algorithm for various degrees of network heterogeneity.

schedule is implemented on a homogeneous network. The variation occurs due to the heterogeneity in the network. The communication matrix for $\Re_x(K, P)$ has PK non-zero entries, while other entries are zeroes. For different values of K, messages are exchanged between a different subset of the total P^2 processor pairs.

An important characteristic of our heuristic algorithm is that the schedule is adaptive to variations in network performance. In a metacomputing system, applications can query a directory service for current values of network performance. The heuristic uses information about the network performance during the communication scheduling phase. In contrast, the direct algorithm uses a fixed communication pattern, irrespective of the network performance. Figure 9 shows the simulated communication time of both schedules as the heterogeneity in the network is varied. Here, $P = 64$, $K = 40$, and $N = 10000000$. The variation in network bandwidth is increased from 100 to 1900%. When the variation is 100 %, the network bandwidth varies in the range 10 MB/s to 20 MB/s. When the variation is 1900 %, the network bandwidth varies in the range of 10 MB/s to 200 MB/s. The completion time of our heuristic algorithm is always very close to the lower bound, for all values of network heterogeneity.

6 Conclusion

In this paper, we have developed an effective communication scheduling technique for cyclic redistribution over a heterogeneous network. Due to advances in high-speed networks, the use of loosely connected distributed computing architectures for High Performance Computing applications is becoming widespread. For example, with high speed ATM or FDDI networks, a cluster of workstations located within a single building or at geographically distributed locations can be used as a high performance computer. Our redistribution algorithms can considerably enhance application performance in such scenarios.

Our communication scheduling technique is adaptive to variations in network performance. This is important because the available bandwidth of heterogeneous networks in typical distributed computing platforms varies considerably with time. The bandwidth depends upon various factors, including network traffic and load. Our communication schedule is derived at run-time, based on current information about available network bandwidth. In contrast, previous techniques for block-cyclic redistribution over a homogeneous network have a fixed schedule, and do not consider network performance issues.

We have presented performance results of our scheduling technique, using both simulations and experimental executions. In comparison with widely used techniques for cyclic redistribution over homogeneous networks, our algorithm shows excellent performance improvements, sometimes upto 60 %. The communication time is always very close to the optimal value. In future research, we shall consider situations where the performance of the network changes during the communication phase. Here, the schedule must be dynamically recomputed at intermediate points during the redistribution.

References

[1] P. Brucker, *Scheduling Algorithms*, Springer, 1995.

[2] DARPA ISO Web Page for the BADD Program, "https://www.iso.darpa.mil/WD@27000.cgi?get+iso::Program+badd+WDI_i_program_page"

[3] Frederic Desprez, Jack Dongarra, Antoine Petitet, Cyril Randriamaro, and Yves Robert, "Scheduling Block-Cyclic Array Redistribution," *IEEE Trans. Parallel and Distributed Systems,* Vol. 9, No. 2, Feb. 1998, pp. 192-205.

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

[5] HiPer-D project web page, "http://www.osf.org/RI/PubProjPgs/HiperD.htm"

[6] S.D. Kaushik, C.-H. Huang, J. Ramanujam, and P. Sadayappan, "Multiphase Array Redistribution: Modeling and Evaluation," *Proc. International Parallel Processing Symposium,* April 1995, pp. 441-445.

[7] JunSeong Kim and David J. Lilja, "Exploiting Multiple Heterogeneous Networks to Reduce Communication Costs in Parallel Programs," *Proc. Heterogeneous Computing Workshop,* pp. 83-95, April 1997.

[8] JunSeong Kim and David J. Lilja, "Utilizing Heterogeneous Networks in Distributed Parallel Computing Systems," *Proc. Sixth IEEE Intl. Symp. High Performance Distributed Computing,* 1997.

[9] Holly Korab and Maxine D. Brown, *Editors, Virtual Environments and Distributed Computing at SC '95: GII Testbed and HPC Challenge Applications on the I-WAY,* a publication of ACM/IEEE Supercomputing '95.

[10] Y.W. Lim, P.B. Bhat, and V.K. Prasanna, "Efficient Data Remapping Algorithms for Embedded Signal Processing Applications," *Proc. 10th International Conference on High Performance Computers,* June 1996.

[11] Y.W. Lim, P.B. Bhat, and V.K. Prasanna, "Efficient Algorithms for Block-Cyclic Redistribution of Arrays," *Algorithmica,* to appear.

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

[13] M. Maheswaran and H.J. Siegel, "Dynamic, Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment," *to appear in Proc. Heterogeneous Computing Workshop,* March 1998.

[14] MSHN Web Page, "http://www.cs.nps.navy.mil/mshn".

[15] S. Ramaswamy and P. Banerjee, "Automatic Generation of Efficient Array Redistribution Routines for Distributed Memory Multicomputers," *Proc. 5th Symposium on Frontiers of Massively Parallel Computation,* Feb. 1995, pp. 342-349.

[16] T. Gonzalez and S. Sahni, "Open Shop Scheduling to Minimize Finish Time," *Journal of the ACM,* Vol. 23, No. 4, Oct. 1976, pp. 665-679.

[17] D.B. Shmoys, C. Stein, and J. Wein, "Improved Approximation Algorithms for Shop Scheduling Problems," *SIAM J. Comput.,* Vol. 23, No.3, pp. 617-632, June 1994.

[18] L. Smarr and C.E. Catlett, "Metacomputing," *Commns. of the ACM,* June 1992, Vol. 35, No. 6, pp. 45-52.

[19] M. Tan, M.D. Theys, H.J. Siegel, N.B. Beck, and M. Jurczyk, "A Mathematical Model, Heuristic, and Simulation Study for a Basic Data Staging Problem in a Heterogeneous Networking Environment," *Proc. Heterogeneous Computing Workshop,* pp. 115-129, March 1998.

[20] R. Thakur, A. Choudhary, and G. Fox, "Runtime Array Redistribution in HPF Programs," *Proc. Scalable High Performance Computing Conference,* May 1994, pp. 309-316.

[21] R. Thakur, A. Choudhary, and J. Ramanujam, "Efficient Algorithms for Array Redistribution," *IEEE Transactions on Parallel and Distributed Systems,* (7)6:587-594, June 1996.

[22] D.W. Walker and S.W. Otto, "Redistribution of Block-Cyclic Data Distributions using MPI," *Technical Report,* ORNL/TM-12999, Oak Ridge National Labs, June 1995.