

## Block-Cyclic Redistribution over Heterogeneous Networks

Prashanth B. Bhat \* and Viktor K. Prasanna\*<sup>†</sup>  
Department of EE-Systems, EEB 200C  
University of Southern California  
Los Angeles, CA 90089-2562  
{prabhat + prasanna}@halcyon.usc.edu

C.S. Raghavendra  
The Aerospace Corporation  
P. O. Box 29257  
Los Angeles, CA 90009  
raghu@aero.org

### Abstract

*Clusters of workstations and networked parallel computing systems are emerging as promising computational platforms for HPC applications. The processors in such systems are typically interconnected by a collection of heterogeneous networks such as Ethernet, ATM, and FDDI, among others. In this paper, we develop techniques to perform block-cyclic redistribution over  $P$  processors interconnected by such a collection of heterogeneous networks.*

*We represent the communication scheduling problem using a timing diagram formalism. Here, each interprocessor communication event is represented by a rectangle whose height denotes the time to perform this event over the heterogeneous network. The communication scheduling problem is then one of appropriately positioning the rectangles so as to minimize the completion time of all the communication events. For the important case where the block size changes by a factor of  $K$ , we develop a heuristic algorithm whose completion time is at most twice the optimal. The running time of the heuristic is  $O(PK^2)$ .*

*Our heuristic algorithm is adaptive to variations in network performance, and derives schedules at run-time, based on current information about available network bandwidth. Our experimental results show that our schedules always have communication times that are very close to optimal.*

**Keywords:** *Workstation clusters, heterogeneous networks, communication scheduling, block-cyclic redistribution.*

### 1. Introduction

Due to advances in high-speed networks, workstation clusters and loosely connected distributed systems are being used as platforms for High Performance Computing. Wide area networking technology has also enabled the development of *metacomputers* [11], wherein grand challenge ap-

plications are parallelized across geographically distributed supercomputers and visualization devices. Such distributed systems are typically interconnected with a collection of many different kinds of communication networks, such as ATM, HiPPI, and Ethernet.

Prototype systems with such heterogeneous networks have been built. For example, [6] evaluated the performance of HPC applications on a cluster of workstations interconnected with ATM and FDDI networks. The I-WAY (Information Wide Area Year) metacomputer at SC '95 consisted of over 10 networks of varying bandwidths, protocols, and routing technology. The HiPer-D project investigates the use of networked 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 [12].

From the above examples, it is clear that heterogeneity is a salient characteristic of the interconnection network in most distributed computational environments. Further, the network is shared among multiple applications. The performance therefore depends upon the current traffic conditions, and typically varies over time.

For scalable performance on such a platform, support for fast application-level communication is necessary. Efficient implementations of important collective communication kernels must be incorporated into communication libraries. In this paper, we develop communication techniques for *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, can be then viewed as block-cyclic redistribution opera-

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

<sup>†</sup>Partially supported by NSF under grant CCR-9317301.

tions. ScaLAPACK, a widely used mathematical software for dense linear algebra computations, also uses a block-cyclic distribution for good load balance and computational efficiency. 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. However, the heterogeneity and sharing of the network make it necessary to develop new communication scheduling techniques. In Section 4, we present a communication scheduling algorithm that is well suited for heterogeneous networks. The algorithm is adaptive to variations in network performance. The schedule is derived at run-time, based on current information about network load.

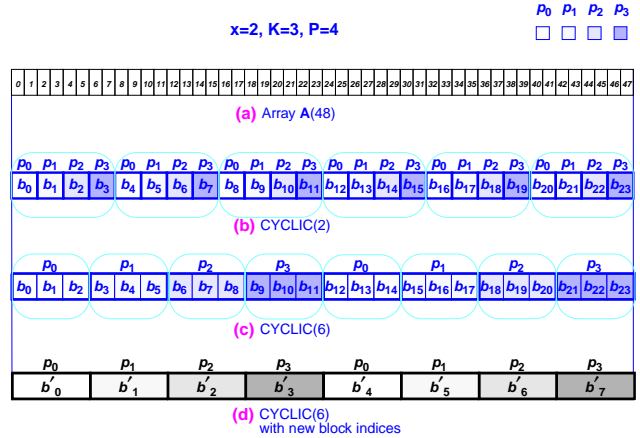
Our scheduling approach is based on a communication model that represents the communication performance between every processor pair using two parameters: a start-up time and a data transmission rate. We formalize the communication scheduling problem using a timing diagram representation. Each interprocessor communication event is represented as a rectangle whose height equals the time to perform the communication over the heterogeneous network. The height is calculated using our communication model. 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. Our heuristic algorithm derives a communication schedule whose completion time is always within twice the optimal. 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 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 heuristic algorithm for block-cyclic redistribution. Section 5 presents performance results from the experimental implementation of our algorithm. Section 6 concludes the paper and discusses future research directions.

## 2. The Block-Cyclic Redistribution Problem

The block-cyclic distribution of an array can be defined as follows [14]: given  $P$  processors, an array with  $N$  elements, and a block size  $x$ , the distribution first partitions the array elements into contiguous 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

<sup>1</sup>For simplicity, we assume that  $x$  divides  $N$ .



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

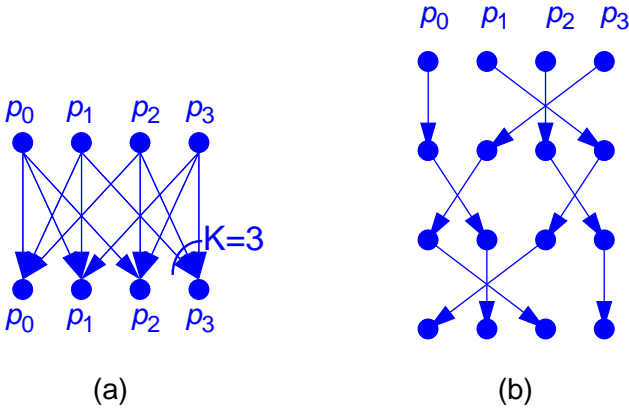
processors in a round robin fashion so that  $b_i$  is assigned to processor  $(i \bmod P)$ . We denote a block-cyclic distribution of block size  $x$  as  $cyclic(x)$ .

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  $\mathfrak{R}_x(K, P)$ .

Figure 1 shows the example of  $\mathfrak{R}_2(3, 4)$ . 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. The aim of communication scheduling is to reduce the overall communication time. During this phase, each processor determines an ordering among its send and receive events,



**Figure 2. (a) Communication pattern for  $\mathfrak{R}_x(3, 4)$  (b) Contention-free schedule for a homogeneous network.**

so as to reduce contention.

- 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, and transmission costs for sending data over the interconnection network. In the absence of communication scheduling, this phase can become very inefficient due to node contention.

The interprocessor communication pattern of  $\mathfrak{R}_x(K, P)$  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. In [7], we have developed efficient techniques for index computation, for systems with homogeneous networks.

Figure 2(b) shows an example of a  $K$ -step communication schedule for  $\mathfrak{R}_x(3, 4)$ . Here, the communication pattern of Figure 2(a) is broken down into a series of contention-free communication steps. Node contention occurs when multiple processors simultaneously send messages to a receiver. When the network is homogeneous, all the communication events within any step of Figure 2(b) would take the same amount of time. In [7], we have developed communication scheduling techniques for such a homogeneous scenario. However, when the network links are heterogeneous, the time taken for each message varies with the available network bandwidth between the corresponding proces-

sors. Due to this non-uniformity in message communication times, node contention and 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 problem.

### 3. Related Work

The block-cyclic redistribution problem has been the focus of several research efforts. Techniques have been developed for both the index computation phase and the communication scheduling phase over a homogeneous network. In [13], Choudhary *et. al.* present efficient index computation algorithms for  $\mathfrak{R}_x(K, P)$ , 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.

In [9], Banerjee *et. al.* represent a *cyclic*( $x$ ) distribution 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.

Sadayappan *et. al.* [5] and Walker *et. al.* [14] have developed algorithms for the communication scheduling phase. Here, a  $K$  step schedule is given for  $\mathfrak{R}_x(K, P)$ . 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.

In [7], we introduced a uniform framework to develop redistribution algorithms for  $\mathfrak{R}_x(K, P)$ . Based on this framework, efficient algorithms were developed for reducing both the index computation and communication overheads. Three classes of techniques were presented for  $\mathfrak{R}_x(K, P)$ : direct, indirect, and hybrid. 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  $\mathfrak{R}_x(K, P)$  communication in  $K$  communication steps. The indirect approach performs  $\mathfrak{R}_x(K, P)$  in atmost  $\lceil \log_2 K \rceil + 2$  steps. Here, the array elements are communicated in a “combine and forward” manner. The hybrid approach is a combination of the direct and indirect approaches.

In [2], communication schedules are developed for the general redistribution problem of *cyclic*( $r$ ) over a set of  $P$  processors, to *cyclic*( $s$ ), over a different set of  $Q$  processors. Graph matching algorithms are used to develop communication schedules in this work. 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.

[8] considers the problem of run-time redistribution from  $cyclic(x)$  on  $P$  processors to  $cyclic(Kx)$  on  $Q$  processors, over a homogeneous network. The algorithm is based on a generalized circulant matrix formalism. The generated schedule 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.

Performance studies of heterogeneous networks were reported in [6]. 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 communication over the heterogeneous network. 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. 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) 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. The goal is to schedule processor and network resources among individual applications so that their QoS requirements are satisfied. In this context, data staging techniques for distributed systems with heterogeneous networks have been considered [12].

## 4. Our Communication Scheduling Approach

As discussed in Section 2, block-cyclic redistribution consists of index computation, communication scheduling, and interprocessor communication. Since the index computation phase is independent of the network characteristics, techniques developed for homogeneous networks [7] can be used. In this section, we consider the communication scheduling phase. 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

The overall network in the  $P$  processor system consists of several heterogeneous network components. Each component interconnects 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 [6]. 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<sup>2</sup>. Typical values for the start-up cost could be in the range of 10 to 50  $\mu$ s, while typical values for the bandwidth could be in the range of a few Mb/s to hundreds of Mb/s.

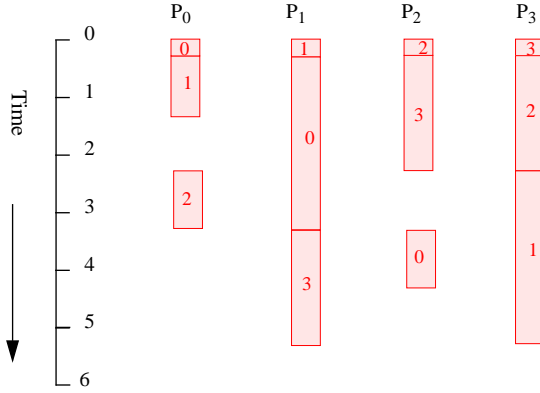
We assume that the effective network performance between any pair of processors will not change during the communication phase. This can be ensured if the application reserves network bandwidth for the duration of the communication. [3] and [15] discuss issues relating to reserving network resources.

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  $\mathfrak{R}_x(3, 4)$  is shown in Figure 3. A timing diagram consists of  $P$  columns, one per processor. The vertical axis rep-

<sup>2</sup>We make this approximation in our experiments.



**Figure 3. A communication schedule for  $\mathcal{R}_x(3, 4)$ .**

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  $P_j$ <sup>3</sup>. 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

During the index computation and communication scheduling phases, each processor independently computes the entire schedule. 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

<sup>3</sup> A receive schedule can be similarly constructed, where the communication events in column  $i$  represent messages received by processor  $P_i$ .

network performance parameters.  $C[i, j]$  is the height of 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. 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  $\mathcal{R}_x(K, P)$  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  $C$ .
- 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 proceeds as follows:

- Whenever a sender  $i$  becomes available at time *sendavail*[ $i$ ], its receiver set  $R_i$  is scanned, and the earliest available receiver  $j$  is selected. The communication event from  $i$  to  $j$  is scheduled to begin at time  $t = \max(\text{sendavail}[i], \text{recvavail}[j])$ . *sendavail*[ $i$ ] and *recvavail*[ $j$ ] are assigned the value  $t + C[j, i]$ , since the sender  $i$  and receiver  $j$  will be busy until this time. Further,  $j$  is deleted from  $R_i$ .
- 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.

Observe that the algorithm is a greedy one. At any time a sender is free, the heuristic assigns a communication event to one 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. Our algorithm is also *adaptive* to changes in network performance. The derived communication schedule depends on the entries of  $C$ , which are in turn

dependent on network load conditions. The schedule can be derived at runtime, using current values of network performance parameters. Previous redistribution algorithms for homogeneous networks do not provide such adaptivity. A “fixed” communication schedule is used irrespective of network load and bandwidth.

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. Our scheduling algorithm therefore runs in  $O(PK^2)$  time. On a single node of the Cray T3E, our algorithm executed in about 10 ms for  $P = 64$  and  $K = 63$ . This is the cost of the communication scheduling phase.

Based on the schedule, the processors perform send and receive operations during the interprocessor communication phase. 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. The cost of the interprocessor communication phase depends upon the completion time of the schedule.

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

**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.  $\square$

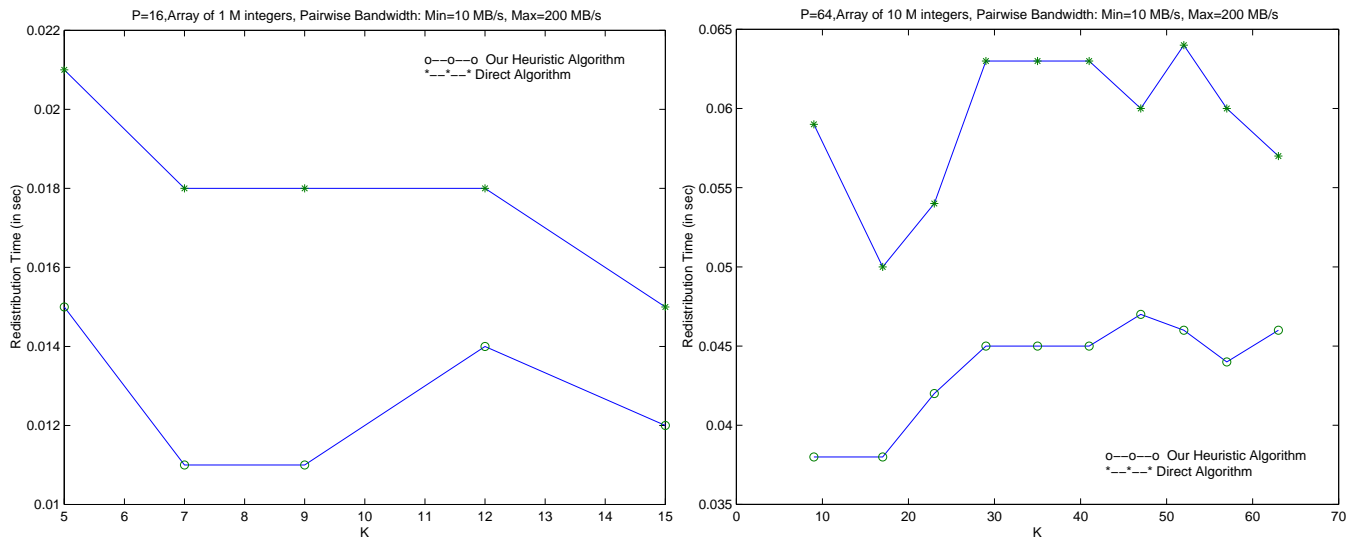
Our communication problem has some similarities to the open shop scheduling problem [4]. Here, a set of  $M$  machines execute a set of  $N$  jobs. Each job  $J_i$  has a task to be executed on every machine  $M_j$ , the execution time for which is given by  $t_{i,j}$ . The tasks may be executed in any order. However, at most one machine may process a given job at any time. Also, at most 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 [10].

## 5. Experimental Results

In this section, we evaluate the performance of our heuristic scheduling technique by measuring the time for the interprocessor communication phase. We compare this with the time for the interprocessor communication phase of the direct communication schedule, which is a fixed communication schedule. For  $\mathfrak{R}_x(K, P)$ , this schedule breaks down the communication pattern into  $K$  contention-free steps. It has been shown to be effective in systems with homogeneous networks, when each communication event takes the same amount of time. It can be derived from the formalisms of [5], [7], and [14].

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  $\mathbf{C}$ .
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. We also implement the direct schedule on the Cray T3E. The communication performance of our heuristic schedule is compared with that of the direct schedule, for various values of  $P$  and  $K$ . The communication times are measured and tabulated.



**Figure 4. Interprocessor communication times of our heuristic schedule and the direct schedule on 16 and 64 processors of the Cray T3E.**

Figure 4 compares the interprocessor communication time of our heuristic schedule and the direct schedule. These times were measured by our experimental implementations of both schedules on the Cray T3E. The simulated heterogeneous network had 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. In Figure 4, results for 16 and 64 processors are shown.  $K$  is varied from a small number to  $P - 1$ . Experimental results for other values of  $P$  can be found in [1]. From Figure 4, we can conclude 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 [7], where the direct schedule is implemented on a homogeneous network. The variation occurs due to the heterogeneity in the network. The communication matrix for  $\mathfrak{R}_x(K, P)$  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.

Figure 5(a) 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.

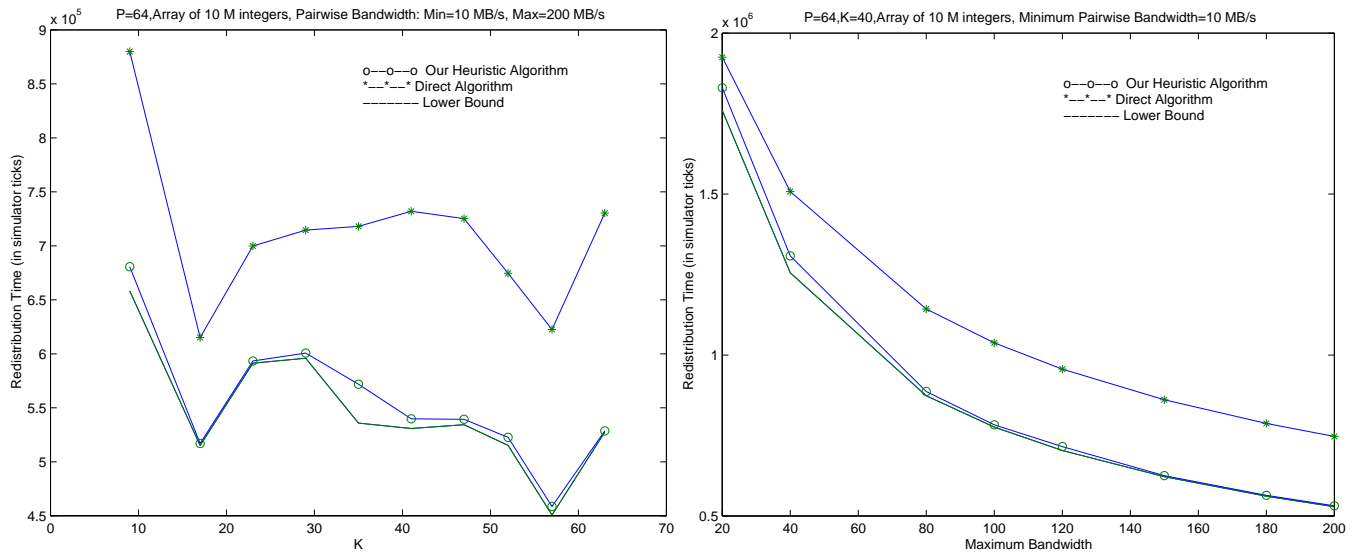
An important characteristic of our heuristic algorithm is

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. Our heuristic can use such information during the communication scheduling phase. In contrast, the direct algorithm uses a fixed communication pattern, irrespective of the network performance. Figure 5(b) 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 the important problem of block-cyclic redistribution over a heterogeneous network. Our techniques target an emerging class of computational platforms, namely workstation clusters and distributed systems. The interconnection network in such systems is typically heterogeneous and shared. Our adaptive algorithms derive communication schedules using run-time information on network performance and heterogeneity.

Our scheduling techniques can lead to significant improvements in application performance. Our experimental results show reductions of upto 60 % in communication



**Figure 5. Simulation results: lower bound, heuristic schedule, and direct algorithm. (a)  $P=64$  processors,  $K=9$  to 63 (b)  $P=64$ ,  $K=40$ , network heterogeneity varies from 100% to 1900%.**

time, compared with widely used algorithms for homogeneous systems.

In our future research efforts, we shall study the performance of our algorithm using a real heterogeneous network. We are also developing techniques to reduce the cost of the communication scheduling phase, by using enhanced data structures. Our communication scheduling technique can be easily extended to other important redistribution problems, such as block-cyclic redistribution of multi-dimensional arrays, and redistribution from *cyclic*( $x$ ) to *cyclic*( $y$ ).

## References

- [1] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra. Block-cyclic redistribution over heterogeneous networks. *Manuscript*, Dept. of EE-Systems, University of Southern California, July 1998.
- [2] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling block-cyclic array redistribution. *IEEE Trans. Parallel and Distributed Systems*, 9(2):192–205, Feb. 1998.
- [3] D. Ferrari, A. Gupta, and G. Ventre. Distributed advance reservation of real-time connections. In *Proc. 5th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video*, Apr. 1995.
- [4] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23(4):665–679, Oct. 1976.
- [5] S. D. Kaushik, C. H. Huang, J. Ramanujam, and P. Sadayappan. Multiphase array redistribution: Modeling and evaluation. In *Proc. Intl. Parallel Processing Symposium*, pages 441–445, Apr. 1995.
- [6] J. Kim and D. J. Lilja. Utilizing heterogeneous networks in distributed parallel computing systems. In *Proc. Sixth IEEE Intl. Symp. High Performance Distributed Computing*, 1997.
- [7] Y. W. Lim, P. B. Bhat, and V. K. Prasanna. Efficient algorithms for block-cyclic redistribution of arrays. *Algorithmica*, to appear.
- [8] N. Park, V. K. Prasanna, and C. S. Raghavendra. Efficient algorithms for block-cyclic array redistribution between processor sets. In *Proc. Supercomputing '98*.
- [9] S. Ramaswamy and P. Banerjee. Automatic generation of efficient array redistribution routines for distributed memory multicomputers. In *Proc. 5th Symposium on Frontiers of Massively Parallel Computation*, pages 342–349, Feb. 1995.
- [10] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM J. Computing*, 23(3):617–632, June 1994.
- [11] L. Smarr and C. E. Catlett. Metacomputing. *Commns. of the ACM*, 35(6):45–52, June 1992.
- [12] 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. In *Proc. Heterogeneous Computing Workshop*, pages 115–129, Mar. 1998.
- [13] R. Thakur, A. Choudhary, and J. Ramanujam. Efficient algorithms for array redistribution. *IEEE Trans. Parallel and Distributed Systems*, 7(6):587–594, June 1996.
- [14] 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.
- [15] L. C. Wolf, L. Delgrossi, R. Steinmetz, S. Schaller, and H. Wittig. Issues of reserving resources in advance. In *Proc. 5th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video*, Apr. 1995.