$N = 552960$							
s	t	lcm	lcm*2	lcm*4	gcd	gcd/2	gcd/4
120	180	22.4	24.2	27.2	22.5	24.7	28.2
120	540	24.2	27.7	34.5	24.5	28.3	36.3
240	180	23.1	25.3	29.4	23.4	25.7	30.4
240	540	24.9	28.7	36.5	25.2	29.3	38.6
480	180	24.2	27.4	34.0	24.2	28.0	35.4
480	540	25.9	31.0	40.9	26.2	31.5	43.5

$N = 2211840$							
s	t	lcm	lcm*2	lcm*4	gcd	gcd/2	gcd/4
120	180	84.6	87.5	91.1	85.3	89.8	94.8
120	540	86.4	91.0	105.9	87.5	93.3	112.9
240	180	86.0	88.2	92.5	86.6	90.5	96.6
240	540	87.7	91.7	107.0	88.8	94.0	114.5
480	180	86.5	89.9	96.7	87.3	92.1	102.6
480	540	88.3	93.5	110.3	89.5	95.6	120.5

Table 1: Execution times (ms) for $cyclic(s)$ to $cyclic(t)$ redistribution on 32 processors.

other block sizes t. Fig. 3 shows the total times in milliseconds for a $cyclic(192)$ to $cyclic(8)$ redistribution on 32 processors for increasing data sizes. This redistribution corresponds to the $cyclic(Yt)$ to $cyclic(t)$ case with $Y = 24$. The two-phase redistribution is: $cyclic(192) \rightarrow cyclic(48) \rightarrow cyclic(8)$. The two-phase strategy performs better than the single-phase strategy up to a data size of approximately $90K$. Note that the cross over between the single-phase and two-phase redistribution occurs at a lower data size for the case when $Y = 24$ than for the case when $Y = 30$. This behavior can be explained by noting that the reduction in the number of message startups with the two-phase redistribution is greater for $Y = 30$. Similar patterns were observed for the $cyclic(s)$ to $cyclic(Ys)$ redistribution.

We now evaluate the effect of choice of common multiples and divisors on the general $cyclic(s)$ to $cyclic(t)$ redistribution. Table. 1 shows the two-stage redistribution times for various source and target block sizes. The redistribution times for various multiples and divisors are presented. It can be observed that the two-stage redistributions using the lcm and the gcd as intermediate block sizes have nearly equal execution times and perform better than the two-stage redistributions using a common multiple greater than the lcm and a common divisor smaller than the gcd.

7 Conclusion

We have presented a multi-phase approach for performing communication efficient data redistribution for block-cyclically distributed arrays. We have developed precise closed form expressions for the send and receive processor and data index sets for two special cases where the block size of the source block-cyclic distribution is a multiple of the block size of the target block-cyclic distribution and vice versa. These closed forms facilitate the development of a distributed scheduling algorithm for performing the all-to-many personalized communication, and the developement of a communication cost model for array redistribution. Based on this model, we demonstrate that the use of the multi-phase redistribution strategy can reduce the total cost for array redistribution. Performance results on the Cray T3D show that the multi-phase strategy can improve performance over the single phase strategy for array redistribution.

The multi-phase strategy is being extended to handle array redistribution of multi-dimensional arrays.

Acknowledgments

We would like to thank the Ohio Supercomputer Center for access to the Cray T3D.

References

[1] B. M. Chapman, P. Mehrotra, and H. P. Zima. Vienna Fortran – a Fortran language extension for distributed memory multiprocessors. In J. Saltz and P. Mehrotra, editors, *Language, Compilers and Runtime Environments for Distributed Memory Machines*, pages 39–62. 1992.

[2] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C.-W. Tseng, and M. Wu. Fortran-D Language Specification. Technical Report TR-91-170, Dept. of Computer Science, Rice University, Dec. 1991.

[3] S. K. S. Gupta, S. D. Kaushik, C.-H. Huang, J. R. Johnson, R. W. Johnson, and P. Sadayappan. A methodology for the generation of data distributions to optimize communication. In *Proc. of IEEE Symposium on Parallel and Distributed Processing*, pages 436–441, Dec. 1992.

[4] S. K. S. Gupta, S. D. Kaushik, S. Mufti, S. Sharma, C.-H. Huang, and P. Sadayappan. On the generation of efficient data communication for distributed-memory machines. In *Proc. of Intl. Computing Symposium*, pages 504–513, 1992.

[5] S. D. Kaushik, C.-H. Huang, R. W. Johnson, and P. Sadayappan. An approach to communication efficient data redistribution. In *Proc. of Intl. Conf. on Supercomputing*, pages 364–373, July 1994.

[6] S. D. Kaushik, C.-H. Huang, J. Ramanujam, and P. Sadayappan. Multiphase array redistribution: Modeling and Evaluation. Technical Report OSU-CISRC-9/94-TR52, Department of Computer and Information Science, The Ohio State University., Sep. 1994.

[7] C. Koelbel. Compile-time generation of communication for scientific programs. In *Proc. of Supercomputing '91*, pages 101–110, Nov. 1991.

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

[9] L. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 3:62–76, Feb. 1993.

[10] S. Ramaswamy and P. Banerjee. Automatic generation of efficient array redistribution routines for distributed-memory multicomputers. Technical Report UILA-ENG-94-2213, CRHC-94-09, CRHPC, Coordinated Science Laboratory, University of Illinios, Apr. 1994.

[11] S. Ranka, J.-C. Wang, and M. Kumar. Irregular personalized communication on distributed memory systems. *Journal of Parallel and Distributed Computing*. To appear.

[12] S. Ranka, J.-C. Wang, and M. Kumar. Personalized communication avoiding node contention on distributed memory systems. In *Proc. of Intl. Conf. on Parallel Processing*, volume I, pages 241–243, 1993.

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

[14] A. Wakatani and M. Wolfe. A new approach to array redistribution: Strip Mining Redistribution. In *Proc. of Parallel Architectures and Languages Europe (PARLE 94)*, July 1994.

by

$$\sum_{i=0}^{k-1}\left(Y_i t_s + \frac{N}{P}t_e\right) \quad = \quad \left(\sum_{i=0}^{k-1} Y_i\right) t_s + k\frac{N}{P}t_e.$$

The multi-phase strategy trades off a decrease in message startups with an increase in the data volume transmitted. The choice of the intermediate distributions will depend on the actual values of the message setup and transfer times. Formulation of the problem of determining the intermediate distributions as an optimization problem, algorithms for identifying the optimal sequence of intermediate distributions when Y is a perfect power, and heuristics for the general case are presented in [6]. The multi-phase approach for the $cyclic(s)$ to $cyclic(Ys)$ redistribution can be similarly developed.

5.2 $Cyclic(s)$ to $Cyclic(t)$ Redistribution

We now consider the general case of redistribution from $cyclic(s)$ to $cyclic(t)$ where $s \not| t$ and $t \not| s$. This redistribution can be performed in two-stages such that each of the two stages corresponds to either a $cyclic(s)$ to $cyclic(Ys)$ redistribution or a $cyclic(Yt)$ to $cyclic(t)$ redistribution. The block size of the intermediate distribution is chosen as either a common multiple m or a common divisor d of s and t. The choice of the divisor/multiple should be such that the total communication cost is reduced.

Using the cost metrics developed for the special array redistribution cases, the cost of the two-stage $cyclic(s) \rightarrow cyclic(m) \rightarrow cyclic(t)$ distribution is

$$\left(\frac{m}{s}t_s + \frac{N}{P}t_e\right) + \left(\frac{m}{t}t_s + \frac{N}{P}t_e\right) = m\left(\frac{1}{s}+\frac{1}{t}\right)t_s + 2\frac{N}{P}t_e$$

(assuming that P is sufficiently large). Clearly the choice of a multiple m which minimizes $m\left(\frac{1}{s}+\frac{1}{t}\right)$ should provide the lowest communication cost. Since the least common multiple $l = lcm(s,t)$ minimizes the expression, the two-stage redistribution using $cyclic(l)$ as the intermediate distribution should have the lowest communication cost. Similarly, the cost for the two-stage redistribution using a common divisor d can be estimated as $\frac{1}{d}(s+t)t_s + 2\frac{N}{P}t_e$. It follows that the intermediate distribution $cyclic(g)$ where $g = gcd(s,t)$ should have the lowest communication cost among all common divisors of s and t.

Since we are interested in finding the scheme which requires the lowest communication cost, a question that arises is which of the two-stage redistribution strategies has lower communication cost. As shown by Lemma 5.3, the common divisor and common multiple approach using the lcm and gcd, respectively, should have equal communication cost.

Lemma 5.3 *For $s, t \in \mathcal{Z}^+$, and $l = lcm(s,t)$, $g = gcd(s,t)$, $l\left(\frac{1}{s}+\frac{1}{t}\right) = \frac{1}{g}(s+t)$.*

After the common divisor or multiple has been identified, the multi-phase approach can be independently applied for each of the two stages. For example, the $cyclic(36)$ to $cyclic(80)$ redistribution can be performed as the two-stage sequence: $cyclic(36) \rightarrow cyclic(720) \rightarrow cyclic(80)$. The first redistribution $cyclic(36) \rightarrow cyclic(720)$ corresponds to the $cyclic(s) \rightarrow cyclic(Ys)$ case while the second corresponds to the $cyclic(Yt) \rightarrow cyclic(t)$ case. The multi-phase approach can now be independently applied to the $cyclic(36) \rightarrow cyclic(720)$ and $cyclic(720) \rightarrow cyclic(80)$ redistributions performing them as $cyclic(36) \rightarrow cyclic(180) \rightarrow cyclic(720)$ and $cyclic(720) \rightarrow cyclic(240) \rightarrow cyclic(80)$, respectively. Using the multi-phase approach the number of message startups is halved while the data transmission cost is doubled.

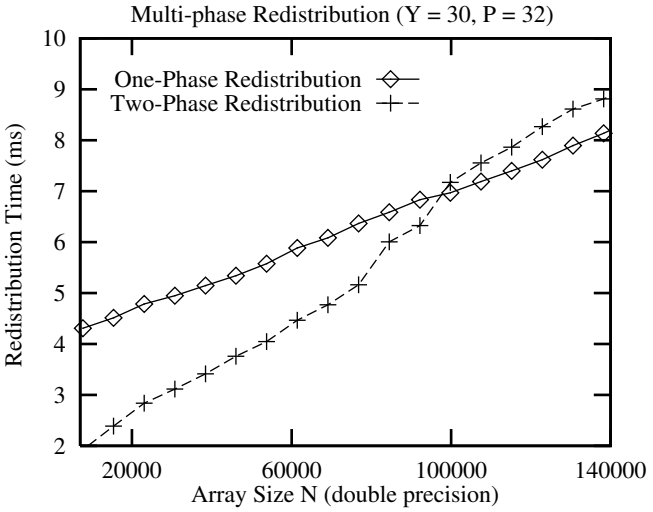

Figure 2: Execution times (ms) for $cyclic(240)$ to $cyclic(8)$ redistribution on 32 processors.

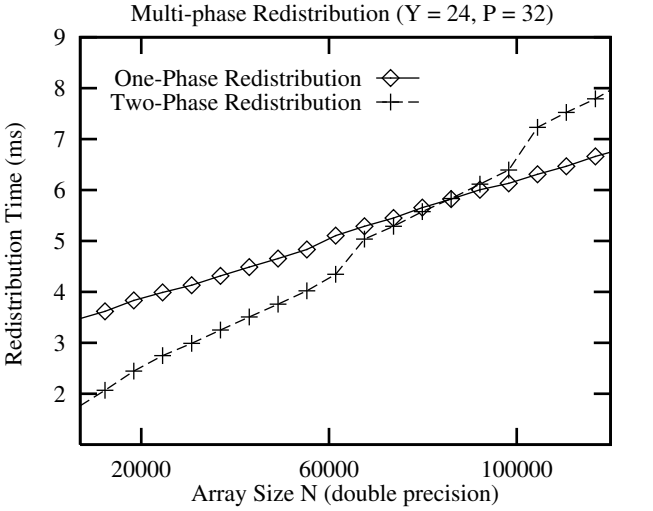

Figure 3: Execution times (ms) for $cyclic(192)$ to $cyclic(8)$ redistribution on 32 processors.

6 Performance Results

In this section, we evaluate the performance of the multi-phase redistribution strategy for specific source and target data distributions on a 32-processor Cray T3D. The total time required for the redistribution of an array of double precision (eight byte) numbers using a single-phase and a multi-phase strategy was measured for increasing data sizes N. This time includes the indexing time for packing and unpacking data and the actual communication time. The times were measured using the realtime clock `rtclock()` and the maximum time among all processors reported.

Fig. 2 shows the total times in milliseconds for a $cyclic(240)$ to $cyclic(8)$ redistribution on 32 processors for increasing data sizes. This redistribution corresponds to the $cyclic(Yt)$ to $cyclic(t)$ case with $Y = 30$. The two-phase redistribution is: $cyclic(240) \rightarrow cyclic(40) \rightarrow cyclic(8)$. As seen in Fig. 2, the two-phase strategy performs better than the single-phase strategy up to a data size of approximately $100K$. A similar pattern was observed for

communication into a sequence of permutations and determining the cost of each permutation.

In the absence of link contention, the cost of transmitting the largest message in a permutation gives the approximate cost of the permutation. If the message startup cost is t_s, the network bandwidth is $\frac{1}{t_e}$, and the largest message size in the permutation is M, then the cost of the permutation is approximately $t_s + M t_e$. We now model the communication cost for the $cyclic(Yt)$ to $cyclic(t)$ redistribution. The cost for the $cyclic(s)$ to $cyclic(Ys)$ redistribution can be similarly modeled.

Using the send processor and data index sets developed in Theorem 3.1, the total number of message startups required and the volume of data to be communicated by a processor p is:

$$\begin{aligned} |PSend(p)| &= \min(P, Y), \\ |DSend(p,q)| &\approx \frac{N}{P * \min(P,Y)}, \forall q \in PSend(p). \end{aligned}$$

When $Y > P$, the communication corresponds to a complete exchange which can be factored into P permutations in a straightforward fashion. We focus on the case when $Y < P$. A factoring of the all-to-many communication into permutations will contain at least Y permutations. In Fig. 1, we provide a distributed method for scheduling the communication as Y permutations. In the algorithm, the send is non-blocking. The blocking receive is performed so as to receive any incoming message and the source processor p_r is determined from the received message.

From Theorem 3.1, for $Y < P$, the first processor that p communicates with is $(pY) \ mod \ P$. We refer to processors q such that $q = (sY) \ mod \ P$ for some $0 \leq s < P$ as *first destination processors*. After the first destination processor $(pY) \ mod \ P$, processor p communicates with every consecutive processor (with a possible wrap-around at $P-1$) up to and including $(pY + Y - 1) \ mod \ P$. If each processor p communicates with a distinct first processor, then clearly every consecutive processor it communicates with will be distinct and the all-to-many communication will be performed as a sequence of permutations. However several processors can communicate with the same first processor. Lemma 4.1 shows that the processors can be partitioned into groups, such that all processors in a group communicate with the same first destination processor.

Lemma 4.1 *All processors* $p + \frac{k * lcm(P,Y)}{Y}$, $0 \leq k < gcd(P,Y)$ *have the same first destination processor, where* $0 \leq p < \frac{lcm(P,Y)}{Y}$.

Let $G(p)$ denote the group of processors $p + \frac{k * lcm(P,Y)}{Y}$ where $0 \leq k < gcd(P,Y)$. There are totally $\frac{lcm(P,Y)}{Y}$ such groups, each group identified by a $p \in \left(0 : \frac{lcm(P,Y)}{Y} - 1\right)$. Clearly not all processors are first destination processors. Lemma 4.2 characterizes the first destination processors for a group $G(p)$.

Lemma 4.2 *All processors* $q \in G(p)$ *communicate first with a processor* r, *such that* $gcd(P,Y) \mid r$.

Thus processors which are not multiples of $g = gcd(P,Y)$ are not first destination processors. If we ensure that the processors in $G(p)$ communicate with distinct first processors then the communication will be scheduled as a sequence of permutations. Since $G(p)$ contains $g-1$ processors excluding p and the processors $(pY \ mod \ P + 1 : pY \ mod \ P + g - 1)$ are not first destination processors, by ensuring that processor $p + \frac{k * lcm(P,Y)}{Y}$ first communicates with processor $pY \ mod \ P + k$ we can guarantee that the communication is performed as a sequence of permutations. In Fig. 1, the first processor p_{sf} that p communicates with is determined as $pY \ mod \ P + \frac{pY}{lcm(P,Y)}$. Thus we have:

Input: Processor id p, total processors P, source block size Yt, target block size t.

Output: Sequence of destination processors for performing all-to-many communication as a sequence of permutations.

begin
$p_f = p * Y \ mod \ P$; $p_l = (p * Y + Y - 1) \ mod \ P$;
$p_{sf} = p_f + (p * Y) \ div \ (lcm(P,Y))$; $p_s = p_{sf}$;
do $i = 0, \ \min(P,Y) - 1$
  Pack $A_loc(DSend(p, p_s))$ into tmp_sbuf;
  Send(p_s, tmp_sbuf);**Recv**(p_r, tmp_rbuf);
  Unpack tmp_rbuf into $B_loc(DRecv(p, p_r))$;
  $p_s = p_s + 1$;
  if $(p_f \leq p_l \wedge p_s > p_l)$ **then** $p_s = p_f$;
  else if $(p_f > p_l \wedge p_s = P)$ **then** $p_s = 0$;
  else if $(p_f > p_l \wedge p_s = p_l + 1)$ **then** $p_s = p_f$;
  endif
enddo

Figure 1: Distributed algorithm for performing communication for $cyclic(Yt)$ to $cyclic(t)$ redistribution.

Theorem 4.3 *The communication for the* $cyclic(Yt)$ *to* $cyclic(t)$ *redistribution is covered by exactly* $\min(P,Y)$ *permutations.*

The cost of each permutation is $t_s + \frac{N}{P * \min(P,Y)}$. Thus the total communication cost for the $cyclic(Yt)$ to $cyclic(t)$ redistribution is $\min(P,Y) t_s + \frac{N}{P} t_e$. The communication cost for the $cyclic(s)$ to $cyclic(Ys)$ redistribution is similarly determined to be $\min(P,Y) t_s + \frac{N}{P} t_e$.

5 A Multiphase Approach for Array Redistribution

In this section, we describe the multi-phase approach for array redistribution. We first consider the $cyclic(Yt)$ to $cyclic(t)$ redistribution and then the general $cyclic(s)$ to $cyclic(t)$ redistribution.

5.1 $Cyclic(Yt)$ to $Cyclic(t)$ redistribution

Consider an array $A(0:319)$ distributed onto 16 processors. A redistribution of A from a $cyclic(20)$ distribution to a $cyclic(2)$ distribution corresponds to the $cylic(Yt)$ to $cyclic(t)$ case with $Y = 10$ and $t = 2$. The cost for the redistribution is $10t_s + 20t_e$. However, if the redistribution is performed in two phases, $cyclic(20) \rightarrow cyclic(10) \rightarrow cyclic(2)$, the total communication cost is $7t_s + 40t_e$. If the message startup cost t_s is significant compared to the element transmission cost t_e for the underlying architecture, then the two-phase redistribution would have a smaller communication cost.

In general, the multi-phase strategy for a $cyclic(Yt)$ to $cyclic(t)$ redistribution, of an array $A(0 : N-1)$ distributed onto P processors, corresponds to performing the redistribution as a sequence: $cyclic(Yt) \rightarrow cyclic(\frac{Y}{Y_1}t) \rightarrow cyclic(\frac{Y}{Y_1 \times Y_2}t) \rightarrow \cdots \rightarrow cyclic(\frac{Y}{Y_1 \times \cdots \times Y_{k-1}}t) \rightarrow cyclic(t)$ where $Y = Y_1 \times \cdots \times Y_k$. Phase i corresponds to a redistribution from a $cyclic(\frac{Y}{Y_1 \times \cdots \times Y_i}t)$ distribution to a $cyclic(\frac{Y}{Y_1 \times \cdots \times Y_{i+1}}t)$ distribution and has a cost $Y_{i+1} t_s + \frac{N}{P} t_e$. The total cost of the k-phase redistribution is given

2 Index Set Representation

In this section, we describe a mixed radix representation based on the tensor product for representing the index sets. For details of the tensor product the reader is referred to [3]. We first briefly describe the the regular data distributions supported in HPF.

Consider an array $A(0:N-1)$ distributed on P processors. Under a block-cyclic distribution with block size b, referred to as a $cyclic(b)$ distribution, the array is partitioned into sections of size b which are cyclically assigned to P processors. The array A is divided into P arrays, each residing in the local memory of one of the processors. The partitioned array A is referred to as A_loc in the SPMD program generated by an HPF compiler. The block distribution corresponds to a $cyclic(\lceil N/P \rceil)$ distribution while the cyclic distribution corresponds to a $cyclic(1)$ distribution.

Let A be an $m \times n$ matrix and B a $p \times q$ matrix. The tensor product of A and B is the block matrix obtained by replacing each element $a_{i,j}$ of A by $a_{i,j}B$. $A \otimes B$ is the $mp \times nq$ matrix defined as

$$A \otimes B = \begin{bmatrix} a_{0,0}B & \cdots & a_{0,n-1}B \\ \vdots & \ddots & \vdots \\ a_{m-1,0}B & \cdots & a_{m-1,n-1}B \end{bmatrix}.$$

We focus on the tensor product of vector bases. A *vector basis* of size m consists of basis vectors $e_i^m,\ 0 \le i < m$, where e_i^m is a column vector of size m with a one at position i and zeros elsewhere. We use e_i^m to represent the vector basis of size m. The tensor product $e_i^m \otimes e_j^n$ of two vector bases, e_i^m and e_j^n is called a *tensor basis* and $e_i^m \otimes e_j^n = e_{in+j}^{mn}$. The size of the tensor basis $e_i^m \otimes e_j^n$ is mn.

In the following sections, we develop closed form expressions for the processor and data index sets for special cases of array redistribution involving block-cyclically distributed arrays. We represent these sets in terms of regular sections or a tensor basis augmented with ranges for the indices of the vector bases. A regular section $(a:b:s)$ represents the indices $\{i \mid i = a + js,\ 0 \le j < \lfloor (b-a)/s \rfloor\}$. A tensor product basis of the form $e_*^C \otimes e_p^b$ represents all the indices $\{i \mid i = i_1 * b + p,\ 0 \le i_1 < C\}$. The symbol $*$ is used to represent all the index values from 0 to $C-1$ where C is the size of the corresponding vector basis. In the above example, the index i_1 iterates over all the values specified by the $*$ symbol. The set of indices represented by a tensor product basis can be enumerated by a perfect loop nest. A **do** loop is generated for every vector basis in the tensor basis with either a $*$ as index or an index for which a range is specified. If a $*$ is specified, the lower loop bound is 0 and the upper loop bound is equal to one less than the size of the corresponding vector basis. If a range is specified then the loop scans the specified range. For example, the indices i specified by $e_*^C \otimes e_{y_1}^Y \otimes e_*^b,\ y \in (0:Y-1:P)$ can be enumerated by the following loop nest:

```
do i1 = 0, C − 1
    do y = 0, Y − 1, P
        do i2 = 0, b − 1
            i = i1 ∗ Y ∗ b + y ∗ b + i2
enddo enddo enddo
```

We now develop the index sets for special cases of array redistribution involving block-cyclically distributed arrays.

3 Array Redistribution

In this section, we present the processor and data index sets for array redistribution when the block size of the source block-cyclic distribution is a multiple of the block size of the target block-cyclic distribution. This redistribution is referred to as a $cyclic(Yt)$ to $cyclic(t)$ redistribution. The redistribution in which the target block size is a multiple of the source block size (a $cyclic(s)$ to $cyclic(Ys)$ redistribution) is symmetric to the $cyclic(Yt)$ to $cyclic(t)$ redistribution. The details for the $cyclic(s)$ to $cyclic(Ys)$ redistribution are provided in [6].

Consider an array $A(0:N-1)$ distributed onto P processors. Let $C = N/(PYt)$ denote the number of courses in the source block-cyclic distribution. We have:

Theorem 3.1 *For a $cyclic(Yt)$ to $cyclic(t)$ array redistribution, the send processor set is*

$$PSend(p) = \begin{cases} ((pY)\ mod\ P : (pY+Y-1)\ mod\ P) & \text{if } Y < P \\ (0 : P-1) & \text{if } Y \ge P. \end{cases}$$

For processor $q \in PSend(p)$, the send data index set is

$$\begin{aligned} DSend(p,q) &= \begin{cases} e_*^C \otimes e_{y_1}^Y \otimes e_*^t & \text{if } Y < P \\ e_*^C \otimes e_{y_2}^Y \otimes e_*^t & \text{if } Y \ge P, \end{cases} \quad \textit{where} \\ y_1 &= (q - pY)\ mod\ P, \\ y_2 &\in ((q-pY)\ mod\ P : Y-1 : P). \end{aligned}$$

Note that the processor send set may involve a wrap-around at $P-1$ if $(pY\ mod\ P) > (pY+Y-1)\ mod\ P$. The receive processor and data index sets are:

Theorem 3.2 *For a $cyclic(Yt)$ to $cyclic(t)$ array redistribution, the receive processor set is*

$$PRecv(p) = \begin{cases} (p : YP-1 : P)\ div\ Y & \text{if } Y < P \\ (0 : P-1) & \text{if } Y \ge P. \end{cases}$$

For processor $q \in PRecv(p)$, the receive data index set is

$$\begin{aligned} DRecv(p,q) &= \begin{cases} e_*^C \otimes e_{y_1}^Y \otimes e_*^t & \text{if } Y < P \\ e_*^C \otimes e_{y_2}^Y \otimes e_*^t & \text{if } Y \ge P, \end{cases} \quad \textit{where} \\ y_1 &= \left\lfloor \frac{qY-p}{P} \right\rfloor \\ y_2 &\in \left(\left\lceil \frac{qY-p}{P} \right\rceil : \left\lfloor \frac{qY-p+Y-1}{P} \right\rfloor \right) \end{aligned}$$

For proofs of Theorem 3.1 and Theorem 3.2, the reader is referred to [6]. We now develop a communication cost model for array redistribution for these special cases.

4 Communication Cost Model

Modeling the communication cost for the all-to-many personalized communication for array redistribution requires accounting for the message startup and transmission costs and the overhead arising due to node and link contention. For worm-hole routed and circuit-switched networks, the cost of transmitting a large message is nearly independent of the distance between the source and target nodes [9]. *Node contention* arises as a processor can typically receive a single message at a time. *Link contention* arises when two or more message paths need to share the same interconnection network edge. Node contention can be avoided by scheduling the communication as a sequence of permutations. For several current architectures the impact of node contention on the communication cost is more significant than that of link contention [12]. For such architectures, it is reasonable to model the cost of all-to-many personalized communication by factoring the

Multi-Phase Array Redistribution: Modeling and Evaluation

S. D. Kaushik* C.-H. Huang* J. Ramanujam† P. Sadayappan*

Abstract

Array redistribution is used in languages such as High Performance Fortran to allow programmers to dynamically change the distribution of arrays across processors. Distributed-memory implementations of several scientific applications require array redistribution. In this paper, efficient methods for performing array redistribution are presented. Precise closed forms for determining the processors involved in the communication and the data elements to be communicated are developed for two special cases of array redistribution involving block-cyclically distributed arrays. The general array redistribution problem involving block-cyclically distributed arrays can be expressed in terms of these special cases. Using the closed forms, a cost model for estimating the communication overhead for array redistribution is developed. A multi-phase approach for reducing the communication cost of array redistribution is presented. Experimental results on the Cray T3D to evaluate the multi-phase approach are provided.

1 Introduction

Efficient array redistribution routines are essential for distributed-memory compilers for languages such as High Performance Fortran (HPF) [8], Fortran D [2], and Vienna Fortran [1]. These languages provide the programmer with a single address space along with directives for specifying array distributions among the processors of a distributed-memory machine. The regular distributions supported are the block, the cyclic and the block-cyclic distribution. Using the array redistribution primitives array distributions can be changed dynamically.

Performing array redistribution incurs an *indexing overhead* for identifying the processors involved in the communication and the array elements to be communicated. Specifically, each processor p has to determine the following data and processor index sets:

- Send processor set of p: set of processors to which p has to send array elements.
- Send data index set of p to processor q: indices of the array elements which are resident on p under the source distribution but are resident on q after redistribution.
- Receive processor set of p: set of processors from which p has to receive array elements.
- Receive data index set of p from processor q: indices of the array elements which are resident on q under the source distribution but are resident on p after redistribution.

A closed form characterization of these sets would reduce the indexing overhead. Such a characterization in terms of regular sections is possible for redistribution involving block and cyclically distributed arrays [4, 7]. However, in general for the block-cyclic distribution, such a closed form characterization is not possible.

Several methods for performing the indexing for array redistribution have been presented in the literature. Closed form expressions for determining the processor sets and data index sets for redistributing block and cyclically distributed arrays were developed in [4, 7]. In [4] a virtual processor approach for handling arbitrary block-cyclic distributions was proposed. Runtime strategies for performing redistribution for special cases of the block-cyclic distribution were presented in [13]. A general array redistribution algorithm which can handle arbitrary source and target processor sets was presented in [10].

Performing array redistribution also incurs a *communication overhead* arising from the all-to-many personalized communication among processors. Comparatively few methods have addressed the issue of reducing or hiding the communication cost for array redistribution. Strip Mining Redistribution, a method in which portions of the array are redistributed in sequence, as opposed to redistributing the entire array in one step to obtain good communication-computation overlap, is presented in [14]. A communication cost model and a multi-phase strategy for performing array redistribution to reduce the communication cost is developed in [5]. Run-time techniques for scheduling the all-to-many personalized communication are presented in [11, 12].

A multi-phase approach for performing array redistribution when the source and the target block-cyclic distributions satisfy certain conditions was presented in [5]. The underlying idea was to perform the array redistribution as a sequence of redistributions so that the total cost of the sequence is lower than the cost of the direct redistribution. Recently, efficient redistribution routines for the special cases when either the block size of the source block-cyclic distribution is a multiple of the block size of the target block-cyclic distribution or vice versa, were presented in [13]. Based on the ideas presented therein, we extend the multi-phase approach to handle arbitrary source and target block-cyclic distributions.

In this paper, a precise closed form characterization of the send and receive processor and data index sets for the special cases when either the source block size is a multiple of the target block size or vice versa is developed. This closed form characterization facilitates the development of a communication cost model and a distributed scheduling algorithm for the all-to-many personalized communication. Based on this cost model, a multi-phase strategy for performing array redistribution for arbitrary source and target block-cyclic distributions is developed.

The paper is organized as follows. Section 2 briefly describes a mixed radix representation for the index sets. In Section 3, closed forms for the processor and data index sets for special cases of array redistribution are presented. A cost model for the all-to-many communication is developed in Section 4. Section 5 describes the multi-phase approach and provides algorithms for determining the sequence of intermediate distributions. Performance results and conclusions are provided in Section 6 and Section 7, respectively.

*Dept. of Computer and Information Science, The Ohio State University, Columbus, OH 43210 ({kaushik,chh,saday}@cis.ohio-state.edu).

†Dept. of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803 (jxr@gate.ee.lsu.edu). Supported in part by an NSF Young Investigator Award CCR-9457768, an NSF grant CCR-9210422, and by LEQSF(1991-94)-RD-A-09