Message Encoding Techniques For Efficient Array Redistribution[1]

Yeh-Ching Chung[2] and Ching-Hsien Hsu

Department of Information Engineering
Feng Chia University, Taichung, Taiwan 407, ROC
Tel : 886-4-4517250 x2706
Fax : 886-4-4515517
Email : ychung, chhsu@pine.iecs.fcu.edu.tw

Abstract- In this paper, we present message encoding techniques to improve the performance of BLOCK-CYCLIC(kr) to BLOCK-CYCLIC(r) (and vice versa) array redistribution algorithms. The message encoding techniques are machine independent and could be used with different algorithms. By incorporating the techniques in array redistribution algorithms, one can reduce the computation overheads and improve the overall performance of array redistribution algorithms. To evaluate the performance of the techniques, we have implemented the message encoding techniques into some array redistribution algorithms on an IBM SP2 parallel machine. The experimental results show that the execution time of array redistribution algorithms with the message encoding techniques is 3% to 22% faster than those without the message encoding techniques.

Keywords: array redistribution, distributed memory multicomputers, message encoding.

1. Introduction

Array redistribution, in general, can be performed in two phases, the send phase and the receive phase. In the send phase, a processor P_i has to determine all the data sets that will be sent to destination processors, pack those data sets, and send those packed data sets to their destination processors. In the receive phase, a processor P_i has to determine all the data sets that will be received from source processors, receive those data sets, and unpack data elements in those data sets to their corresponding local array positions. This means that each processor P_i should compute the following four sets.

- Destination Processor Set (DPS[P_i]) : the set of processors to which P_i has to send data.
- Send Data Sets ($\bigcup_{P_j \in DPS[P_i]}$ SDS[P_i, P_j]) : the sets of array elements that processor P_i has to send to its destination processors, where SDS[P_i, P_j] denotes the set of array elements that processor P_i has to send to its destination processor P_j.
- Source Processor Set (SPS[P_j]) : the set of processors from which P_j has to receive data.
- Receive Data Sets ($\bigcup_{P_i \in SPS[P_j]}$ RDS[P_j, P_i]) : the sets of array elements that P_j has to receive from its source processors, where RDS[P_j, P_i] denotes the set of array elements that processor P_j has to receive from its source processor P_i.

Since array redistribution is performed at run-time, there is a performance trade-off between the efficiency of a new data decomposition for a subsequent phase of an algorithm and the cost of redistributing data among processors. Thus efficient methods for performing array redistribution are of great importance for the development of distributed memory compilers. In this paper, we present the *message encoding techniques* to improve the performance of array redistribution algorithms. For the message encoding techniques, in the send phase, a source processor encodes the unpacking information into messages that will be sent to its destination processors. In the receive phase, for a destination processor, according to the encoded unpacking information, one can perform unpacking process without calculating the RDS.

The paper is organized as follows. In Section 2, a brief survey of related work will be presented. In Section 3, the message encoding techniques for array redistribution will be described in details. The encoding and unpacking algorithms used by the message encoding techniques for array redistribution will be given in Section 4. The performance evaluation will be presented in Section 5

2. Related Work

Gupta *et al.* [2] derived closed form expressions

[1] The work of this paper was partially supported by NSC of R.O.C. under contract NSC-86-2213-E035-023.
[2] The correspondence addressee.

Proceedings of the 1997 International Conference on Parallel Processing (ICPP'97)
0-8186-8108-X 97 $10.00 © 1997 IEEE

to efficiently determine the send/receive processor/data sets. Similar approaches was also presented in [1,6,9,12]. Thakur *et al.* [10, 11] presented algorithms for run-time array redistribution in HPF programs. In [8], Ramaswamy *et al.* used a mathematical representation, PITFALLS, for regular data redistribution. Similar approach in finding the intersections between LHS and RHS of array statements was also presented in [3].

Kaushik *et al.* [5] proposed a multi-phase redistribution approach for array redistribution. In [14], portion of array elements were redistributed in sequence in order to overlap the communication and computation. In [15], a spiral mapping technique was proposed to reduce communication conflicts when performing a redistribution. Kalns and Ni [4] proposed a processor mapping technique to minimizes the amount of data exchange for redistribution. In [7], a generalized circulant matrix formalism was proposed to reduce the communication overheads redistribution. Walker *et al.* [13] used the standardized message passing interface, MPI, to express the redistribution operations.

3. Message Encoding Techniques

In general, the BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution can be classified into three types,

- s is divisible by t, i.e. BLOCK-CYCLIC(s=kr) to BLOCK-CYCLIC(t=r) redistribution,
- t is divisible by s, i.e. BLOCK-CYCLIC(s=r) to BLOCK-CYCLIC(t=kr) redistribution,
- s is not divisible by t and t is not divisible by s.

To simplify the presentation, we use $kr \rightarrow r$, $r \rightarrow kr$, and $s \rightarrow t$ to represent the first, the second, and the third types of redistribution, respectively, for the rest of the paper.

Definition 1: Given a BLOCK-CYCLIC(s) to BLOCK-CYCLIC(t) redistribution, BLOCK-CYCLIC(s), BLOCK-CYCLIC(t), s, and t are called the *source distribution*, the *destination distribution*, the *source distribution factor*, and the *destination distribution factor* of the redistribution , respectively.

Definition 2: Given an $s \rightarrow t$ redistribution on $A[1{:}N]$ over M processors, the *source* (*destination*) *local array* of processor P_i (P_j), denoted by $SLA_i[0{:}N/M{-}1]$ ($DLA_j[0{:}N/M{-}1]$), is defined as the set of array elements that are distributed to processor P_i (P_j) in the source (destination) distribution, where $0 \le i, j \le M{-}1$.

Definition 3: Given an $s \rightarrow t$ redistribution on $A[1{:}N]$ over M processors, the *source* (*destination*) *processor* of an array element in $A[1{:}N]$ or $DLA_j[0{:}N/M{-}1]$ ($SLA_i[0{:}N/M{-}1]$) is defined as the processor that owns the array element in the source (destination) distribution, where $0 \le i, j \le M{-}1$.

Definition 4: Given an $s \rightarrow t$ redistribution on $A[1{:}N]$ over M processors, we define $SG : SLA_i[m] \rightarrow A[k]$ is a function that converts a source local array element $SLA_i[m]$ of P_i to its corresponding global array element $A[k]$ and $DG : DLA_j[n] \rightarrow A[l]$ is a function that converts a destination local array element $DLA_j[n]$ of P_j to its corresponding global array element $A[l]$, where $1 \le k, l \le N$ and $0 \le m, n \le N/M{-}1$.

Definition 5: Given an $s \rightarrow t$ redistribution on $A[1{:}N]$ over M processors, a *global complete cycle* (*GCC*) of $A[1{:}N]$ is defined as M times the least common multiple of s and t, i.e., $GCC{=}M{\times}lcm(s,t)$. We define $A[1{:}GCC]$ as the first global complete cycle of $A[1{:}N]$, $A[GCC{+}1{:}2{\times}GCC]$ as the second global complete cycle of $A[1{:}N]$, and so on.

Definition 6: Given an $s \rightarrow t$ redistribution, a *local complete cycle* (*LCC*) of a local array $SLA_i[0{:}N/M{-}1]$ (or $DLA_j[0{:}N/M{-}1]$) is defined as the least common multiple of s and t, i.e., $LCC = lcm(s, t)$. We define $SLA_i[0{:}LCC{-}1]$ ($DLA_j[0{:}LCC{-}1]$) as the first local complete cycle of $SLA_i[0{:}N/M{-}1]$ ($DLA_j[0{:}N/M{-}1]$), $SLA_i[LCC{:}2{\times}LCC{-}1]$ ($DLA_j[LCC{:}2{\times}LCC{-}1]$) as the second local complete cycle of of $SLA_i[0{:}N/M{-}1]$ ($DLA_j[0{:}N/M{-}1]$), and so on.

3.1 The Message Encoding Technique for $kr \rightarrow r$ Redistribution

Due to the page limitation, we omit the proof of lemmas presented in this paper.

Lemma 1: Given an $s \rightarrow t$ redistribution on $A[1{:}N]$ over M processors, $SLA_i[m]$, $SLA_i[m{+}LCC]$, $SLA_i[m{+}2{\times}LCC]$, ..., and $SLA_i[m{+}N/M{\times}LCC]$ have the same destination processor, where $0 \le i \le M{-}1$ and $0 \le m \le LCC{-}1$. ■

Lemma 2: Given a $kr \rightarrow r$ redistribution on $A[1{:}N]$ over M processors, for a source processor P_i and array elements in $SLA_i[x{\times}LCC{:}(x{+}1){\times}LCC{-}1]$, if the destination processor of $SG(SLA_i[a_0])$, $SG(SLA_i[a_1])$, ..., $SG(SLA_i[a_{\gamma-1}])$ is P_j, then $SG(SLA_i[a_0])$, $SG(SLA_i[a_1])$, ..., $SG(SLA_i[a_{\gamma-1}])$ are in the consecutive local array positions of $DLA_j[0{:}N/M{-}1]$, where $0 \le x \le N/GCC{-}1$ and $x{\times}LCC \le a_0 < a_1 < a_2 < \ldots < a_{\gamma-1} < (x{+}1){\times}LCC$. ■

Lemma 3: Given a $kr \rightarrow r$ redistribution on $A[1{:}N]$ over M processors, for a source processor P_i, if $SLA_i[a]$ and $SLA_i[b]$ are the first element in $SLA_i[x{\times}LCC{:}\ (x{+}1) \times LCC - 1]$ and $SLA_i[(x{+}1) \times LCC{:}\ (x{+}2) \times LCC - 1]$, respectively, with the same destination processor P_j and $SG(SLA_i[a]) = DG(DLA_j[\alpha])$, then $SG(SLA_i[b]) = DG(DLA_j[\alpha + kr])$, where $0 \le x \le N / GCC{-}2$ and $0 \le \alpha \le N / M - 1$. ■

Given a $kr \rightarrow r$ redistribution on $A[1{:}N]$ over M processors, for a source processor P_i, we assume that there are γ array elements in $SLA_i[0{:}LCC{-}1]$ whose destination processor is P_j. In the receive phase, if the first array element of the message will be

Proceedings of the 1997 International Conference on Parallel Processing (ICPP'97)
0-8186-8108-X 97 $10.00 © 1997 IEEE

unpacked to $DLA_j[\alpha]$, according to Lemmas 1, 2, and 3, the first γ array elements of the message will be unpacked to $DLA_j[\alpha:\alpha+\gamma-1]$, the second γ array elements of the message will be unpacked to $DLA_j[\alpha+kr:\alpha+kr+\gamma-1]$, the third γ array elements of the message will be unpacked to $DLA_j[\alpha+2kr:\alpha+2kr+\gamma-1]$, and so on. Therefore, if we know the values of α and γ in the send phase and encode the values of α and γ as the first and the second elements of a message, respectively, then we can perform the unpacking process without computing the receive data sets in the receive phase.

Given a $kr \rightarrow r$ redistribution on $A[1:N]$ over M processors, for a source processor P_i, the values of α and γ can be computed by the following equations:

$$\alpha = \begin{cases} \lfloor rank(P_i) \times k / M \rfloor \times r & \text{if } rank(P_i) \geq mod(rank(P_i) \times k, M) \\ (\lfloor rank(P_i) \times k / M \rfloor + 1) \times r & \text{otherwise} \end{cases} \quad (1)$$

$$\gamma = \begin{cases} (\lfloor k/M \rfloor + 1) \times r & \text{if } mod(rank(P_i) + M - \\ & \quad mod(rank(P_i) \times k, M), M) < mod(k, M) \\ \lfloor k/M \rfloor \times r & \text{otherwise} \end{cases} \quad (2)$$

where $rank(P_i)$ and $rank(P_j)$ are the ranks of processors P_i and P_j, respectively.

3.2 The Message Encoding Technique for $r \rightarrow kr$ Redistribution

Lemma 4: Given a $r \rightarrow kr$ redistribution on $A[1:N]$ over M processors, for a source processor P_i and array elements in $SLA_i[x \times LCC:(x+1) \times LCC-1]$, if the destination processor of $SG(SLA_i[a_0])$, $SG(SLA_i[a_1])$, ..., $SG(SLA_i[a_{\gamma-1}])$ is P_j, and $SG(SLA_i[a_0]) = DG(DLA_j[\alpha])$, then $SG(SLA_i[a_r]) = DG(DLA_j[\alpha + Mr])$, $SG(SLA_i[a_{2r}]) = DG(DLA_j[\alpha + 2Mr])$, ..., and $SG(SLA_i[a_{\gamma-r}]) = DG(DLA_j[\alpha + (\gamma/r-1) \times Mr])$, where $0 \leq \alpha \leq N/M-1$, $0 \leq x \leq N/GCC-1$ and $x \times LCC \leq a_0 < a_1 < a_2 < \ldots < a_{\gamma-1} < (x+1) \times LCC$. ■

Given an $r \rightarrow kr$ redistribution on $A[1:N]$ over M processors, for a source processor P_i, we assume that there are γ array elements in $SLA_i[0:LCC-1]$ whose destination processor is P_j. In the receive phase, if the first array element of the message will be unpacked to $DLA_j[\beta]$, according to Lemmas 1, and 4, the first γ array elements of the message will be unpacked to $DLA_j[\beta:\beta+r-1]$, $DLA_j[\beta + Mr : \beta + Mr + r - 1]$, $DLA_j[\beta + 2Mr : \beta + 2Mr + r - 1]$, ..., and $DLA_j[\beta+(\gamma/r-1) \times Mr:\beta+(\gamma/r-1) \times Mr+r-1]$; the second γ array elements of the message will be unpacked to $DLA_j[\beta + kr : \beta + kr + r - 1]$, $DLA_j[\beta + kr + Mr : \beta + kr + Mr + r - 1]$, $DLA_j[\beta + kr + 2Mr : \beta + kr + 2Mr + r - 1]$, ..., and $DLA_j[\beta + kr + (\gamma/r-1) \times Mr:\beta+kr+(\gamma/r-1) \times Mr+r-1]$, and so on. Therefore, if we know the values of β and γ β, then we can perform the unpacking process without computing the RDS in the receive phase.

Given an $r \rightarrow kr$ redistribution on $A[1:N]$ over M processors, for a source processor P_i, the value of γ can be computed by Equation 2. The value of β can be computed by the following equation,

$$\beta = mod(rank(P_i) + M - mod(rank(P_j) \times k, M), M) \times r \quad (3)$$

4. Incorporate Message Encoding Techniques with Array Redistribution Algorithms

To incorporate the message encoding techniques with the $kr \rightarrow r$ and $r \rightarrow kr$ redistribution algorithms, we need the following four algorithms.

```
Algorithm kr_to_r_encoding(k, r, M)
1.  For each destination processor P_j in DPS[P_i] do
2.    { calculate α and γ using Equations
        1 and 2, respectively;
3.    send_mes_j[0] = α;
4.    send_mes_j[1] = γ; }
end_of_kr_to_r_encoding
```

```
Algorithm kr_to_r_unpacking(k, r, M, N)
1.  P_j receives a message recv_mes_i from source
    processor P_i
2.  α = rcev_mes_i[0]; γ = recv_mes_i[1];
3.  length_i = 2; cycle = N / (M × kr);
    count = 0; index = α − kr;
4.  while (count < cycle)
5.    { index += kr ;
6.      for (x = 0 ; x < γ; x++)
7.        DLA_j[index+x]
                = recv_mes_i[length_i++];
8.    count++; }
end_of_kr_to_r_unpacking
```

```
Algorithm r_to_kr_encoding(k, r, M)
1.  For each destination processor P_j in DPS[P_i] do
2.    { calculate β and γ using Equations
        3 and 2, respectively;
3.    send_mes_j[0] = β;
4.    send_mes_j[1] = γ }
end_of_r_to_kr_encoding
```

```
Algorithm r_to_kr_unpacking(k, r, M, N)
1.  P_j receives a message in recv_mes_i from source
    processor P_i
2.  β = recv_mes_i[0]; γ = recv_mes_i[1];
3.  length_i = 2; cycle = N / (M × kr);
4.  count = 0; index = β − kr;
5.  local_index = 0;
6.  while (count < cycle)
7.    { index += kr ;
8.      local_index = index − M × r;
9.      for (x = 0 ; x < γ / r; x++)
10.       { local_index += M × r;
11.         for(y = 0 ; y < r ; y++)
12.           local_array(local_index+y)
                = recv_mes_i(length_i++) ; }
13.   count ++; }
end_of_r_to_kr_unpacking
```

5. Performance Evaluation and Experimental Results

To evaluate the performance of the proposed

Proceedings of the 1997 International Conference on Parallel Processing (ICPP'97)
0-8186-8108-X 97 $10.00 © 1997 IEEE

message encoding techniques, we have implemented the message encoding techniques into algorithms presented in [10, 11] for $kr \rightarrow r$ and $r \rightarrow kr$ redistribution on a 16-nodes SP2. We called algorithms with and without the message encoding techniques MET_REDIS and REDIS, respectively.

Table 1 gives the execution time and the percentages of the performance improvement of MET_REDIS over REDIS. The execution time of redistribution in the synchronous communication model is about 15% to 22% faster than that of REDIS. In the asynchronous model, the execution time of redistribution is about 3% to 7% faster than that of REDIS. We have noted that the improvement percentage of the synchronous model is greater than that of the asynchronous model. This is because that the computation and communication can be overlapped in the asynchronous model, but can not be overlapped in the synchronous model. For the cases of k = 10, 20, 50, and BLOCK to CYCLIC (and vice-versa) redistribution, we have similar results (Due to the page limitation, we did not show the results here).

6. Conclusions

In this paper, based on $kr \rightarrow r$ and $r \rightarrow kr$ redistribution, we have developed the message encoding techniques. The message encoding techniques are machine independent and could be used with different array redistribution algorithms. By incorporating the techniques in array redistribution algorithms, one can reduce the computational overheads. The experimental results show that the execution time of array redistribution algorithms with the message encoding techniques is 3% to 22% faster than those without the message encoding techniques.

References

[1] S. Chatterjee, J. R. Gilbert, F. J. E. Long, R. Schreiber, and S.-H. Teng, "Generating Local Address and Communication Sets for Data Parallel Programs," *JPDC*, Vol. 26, pp. 72-84, 1995.

[2] S. K. S. Gupta, S. D. Kaushik, C.-H. Huang, and P. Sadayappan, "On Compiling Array Expressions for Efficient Execution on Distributed-Memory Machines," *JPDC*, Vol. 32, pp. 155-172, 1996.

[3] S. Hiranandani, K. Kennedy, J. Mellor-Crammey, and A. Sethi," Compilation technique for block-cyclic distribution," In *Proc. ACM Intl. Conf. on Supercomputing*, pp. 392-403, July 1994.

[4] E. T. Kalns, and L. M. Ni, "Processor Mapping Technique Toward Efficient Data Redistribution, " *IEEE TPDS*, vol. 6, no. 12 , December 1995.

[5] S. D. Kaushik, C. H. Huang, J. Ramanujam, and P. Sadayappan, "Multiphase array redistribution: Modeling and evaluation," In *Proc. of IPPS*, pp. 441-445, 1995.

[6] K. Kennedy, N. Nedeljkovic, and A. Sethi, "Efficient address generation for block-cyclic distribution," In *Proc. of Intl. Conf. on Supercomputing*, Barcelona, pp. 180-184, July 1995.

[7] Y.-W. Lim, Prashanth B. Bhat, and Viktor, K. Prasanna, "Efficient Algorithms for Block-Cyclic Redistribution of Arrays," *Proceedings of the Eighth IEEE Symposium on Parallel and Distributed Processing*, pp. 74-83, 1996.

[8] S. Ramaswamy, B. Simons, and P. Banerjee, "Optimization for efficient array Redistribution on Distributed Memory Multicomputers," *JPDC*, Vol. 38, pp. 217-228, 1996.

[9] J. M. Stichnoth, D. O'Hallaron, and T. R. Gross," Generating communication for array statements: design, implementation, and evaluation," *JPDC*, Vol. 21, pp. 150-159, 1994.

[10] R. Thakur, A. Choudhary, and G. Fox, "Runtime array redistribution in HPF programs, " *Proc. 1994 Scalable High Performance Computing Conf.* , pp. 309-316, May 1994.

[11] Rajeev. Thakur, Alok. Choudhary, and J. Ramanujam, "Efficient Algorithms for Array Redistribution, " *IEEE TPDS*, vol. 7, no. 6 , JUNE 1996.

[12] A. Thirumalai and J. Ramanujam, "HPF array statements: Communication generation and optimization," *3th workshop on Languages, Compilers and Run-time system for Scalable Computers*, Troy. NY, May 1995.

[13] David W. Walker, Steve W. Otto, "Redistribution of BLOCK-CYCLIC Data Distributions Using MPI," Technical Report ORNL/TM-12999, Computer Science and Mathematics Division, Oak Ridge National Laboratory, 1995.

[14] A. Wakatani and M. Wolfe, "A New Approach to Array Redistribution: Strip Mining Redistribution," In *Proc. of Parallel Architectures and Languages Europe*, July 1994.

[15] A. Wakatani and M. Wolfe, "Optimization of array redistribution for distributed memory multicomputer, " In *Parallel Computing(submitted)*, 1994.

Table 1 : The percentages of the performance improvement of MET_REDIS over REDIS.

BLOCK-CYCLIC(4) to BLOCK-CYCLIC(2)				BLOCK-CYCLIC(2) to BLOCK-CYCLIC(4)			
(Synchronous)				(Synchronous)			
SIZE	REDIS	MET_REDIS	Improvement	SIZE	REDIS	MET_REDIS	Improvement
64	0.7	0.597	14.7%	64	0.456	0.405	7.6%
640	0.811	0.627	22.7%	640	0.572	0.523	8.6%
6400	2.056	1.76	14.4%	6400	2.165	2.031	6.2%
(Asynchronous)				(Asynchronous)			
64	0.52	0.506	2.7%	64	0.424	0.395	6.8%
640	0.635	0.613	3.5%	640	0.455	0.431	5.3%
6400	1.816	1.694	6.7%	6400	2.087	1.957	6.2%

Time unit : ms

Proceedings of the 1997 International Conference on Parallel Processing (ICPP'97)
0-8186-8108-X 97 $10.00 © 1997 IEEE