Syracuse University

SURFACE

Northeast Parallel Architecture Center | College of Engineering and Computer Science

1993

Complete Exchange on a Wormhole Routed Mesh

Rajeev Thakur
Syracuse University, Northeast Parallel Architectures Center, thakur@npac.syr.edu

Alok Choudhary
Syracuse University, Northeast Parallel Architectures Center

Geoffrey C. Fox
Syracuse University, Northeast Parallel Architectures Center

Follow this and additional works at: https://surface.syr.edu/npac

 Part of the Computer Sciences Commons

Recommended Citation

Thakur, Rajeev; Choudhary, Alok; and Fox, Geoffrey C., "Complete Exchange on a Wormhole Routed Mesh" (1993). *Northeast Parallel Architecture Center*. 65.
https://surface.syr.edu/npac/65

This Article is brought to you for free and open access by the College of Engineering and Computer Science at SURFACE. It has been accepted for inclusion in Northeast Parallel Architecture Center by an authorized administrator of SURFACE. For more information, please contact surface@syr.edu.

Complete Exchange on a Wormhole Routed Mesh

Rajeev Thakur * Alok Choudhary * Geoffrey Fox †
Northeast Parallel Architectures Center
111 College Place, Rm. 3-228
Syracuse University
Syracuse, NY 13244-4100
thakur, choudhar, gcf @npac.syr.edu

Abstract

The complete exchange (or all-to-all personalized) communication pattern occurs frequently in many important parallel computing applications. We discuss several algorithms to perform complete exchange on a two dimensional mesh connected computer with wormhole routing. We propose algorithms for both power-of-two and non power-of-two meshes as well as an algorithm which works for any arbitrary mesh. We have developed analytical models to estimate the performance of the algorithms on the basis of system parameters. These models take into account the effects of link contention and other characteristics of the communication system. Performance results on the Intel Touchstone Delta are presented and analyzed.

1 Introduction

Low-dimension high-bandwidth interconnection networks, such as a mesh, have recently emerged as a popular alternative to the earlier high-dimension low-bandwidth networks, such as the hypercube, for distributed memory multicomputers. The Intel Touchstone Delta, the Intel Paragon and the Symult 2010 use a two-dimensional mesh while the MIT J-machine and the Mosaic computer developed at Caltech use a three-dimensional mesh [5]. All these machines use *wormhole routing*, an important feature of which is that the network latency is almost independent of the path length when there is no link contention and the packet size is large. In this paper, we discuss four algorithms to perform complete exchange on a mesh connected computer with wormhole routing. The complete exchange or all-to-all personalized communication pattern is one in which all processors simultaneously need to communicate with all other processors. It occurs in many applications like parallel quicksort, some implementations of the 2D FFT, matrix transpose, array redistribution etc. It is the densest form of communication which can result in a lot of link contention. Hence it is necessary to use efficient algorithms to perform complete exchange.

Complete exchange algorithms for a hypercube architecture are described in [2, 4]. Ponnusamy, Thakur et al [6] discuss complete exchange on the fat tree architecture of the CM-5. These algorithms assume that the number of processors is a power-of-two, which is a valid assumption for those architectures. The mesh architecture introduces different problems because of high contention and the fact that the user can allocate a mesh size which need not be a power-of-two and may even be an odd number (eg. 5×5). Bokhari and Berryman [3] describe two algorithms for a circuit-switched mesh, which assume that the number of processors is a power-of-two. In this paper, we discuss algorithms for both power-of-two and non power-of-two meshes. We have developed analytical models to estimate the performance of the algorithms. We present performance results on the Intel Touchstone Delta.

Section 2 describes the architecture of the Delta and the performance model used for the algorithms. The algorithms are described in Section 3. The performance of the algorithms on the Delta is discussed in Section 4 followed by Conclusions in Section 5.

2 Architecture and Performance Model

The Intel Touchstone Delta is a 16×32 mesh of computational nodes, each of which is an Intel i860/XR microprocessor. The two-dimensional mesh interconnection network has bidirectional links with wormhole routing. It uses deterministic XY routing in which packets are first sent along the X dimension and then along the Y dimension. In wormhole routing, a packet is divided into a number of *flits* (flow control digits) for transmission. The size of a flit is typically the same as the channel width. The header flit of a packet determines the route and remaining flits follow in a pipeline

*Also with the Dept. of Electrical and Computer Eng., Syracuse University

†Also with the Dept. of Computer and Information Science, Syracuse University

fashion. The network latency for wormhole routing is $(L_f/B)D + L/B$, where L_f is the length of each flit, B is the channel bandwidth, D is the path length, and L is the length of the message. Thus, if $L_f << L$, the path length D will not significantly affect the network latency provided there is no link contention. Details of wormhole routing techniques can be found in [5].

To model the performance of the algorithms, we use an approach similar to that used by Barnett et al in [1]. The following notations are used in our models :-

α	startup time per message
β_{ex}	transfer time per byte for an exchange with no link conflicts
β_{sr}	transfer time per byte to send to and receive from different processors with no link conflicts
β_{sat}	transfer time per byte on a saturated link
L	number of bytes to be exchanged per processor pair
$f(i)$	maximum number of messages contending for a saturated link at step i
r	number of rows in the mesh
c	number of columns in the mesh
p	total number of processors $= r \times c$

The time taken for an exchange operation may be different from the time to send to and receive from different processors, because in the latter case the incoming and outgoing messages may traverse links with different amount of contention. Hence, we use β_{ex} or β_{sr} depending on the algorithm. We assume that the time taken is independent of distance, a property of wormhole routing. Thus, the time required for an exchange step i is given by

$$T = \alpha + L\ max(\beta_{ex}, f(i)\beta_{sat})$$

We assume that conflicting messages share the bandwidth of a network link and that there exists some positive integer γ such that $\beta = 2^{\gamma}\beta_{sat}$. For the Delta, $\gamma = 1$ is a good approximation [1]. In other words, even if two messages contend for a link, there is no increase in communication time. Note that since the Delta has bidirectional links, two messages contend for a link only if they need to travel in the same direction simultaneously.

3 Algorithms

Scott [7] has shown that $a^3/4$ is the lower bound on the number of phases required to perform a complete exchange on an $a \times a$ mesh such that there is no link contention in any phase. However, if we allow link contention to exist, the operation can be performed in fewer steps. We have adopted this approach of allowing a small amount of link contention to exist, thereby reducing the number of steps and keeping all processors active at every step. This approach takes advantage of the fact that in machines like the Delta and the Paragon, the links have excess bandwidth, so that a small number of contending messages will not affect the communication time.

```
do i=1, p − 1
   destination = xor(mynumber, i)
   Exchange with destination
end do
```

Figure 1: Algorithm for PEX

Table 1: Communication Schedule for PEX on 8 Procs

Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7
$0 \leftrightarrow 1$	$0 \leftrightarrow 2$	$0 \leftrightarrow 3$	$0 \leftrightarrow 4$	$0 \leftrightarrow 5$	$0 \leftrightarrow 6$	$0 \leftrightarrow 7$
$2 \leftrightarrow 3$	$1 \leftrightarrow 3$	$1 \leftrightarrow 2$	$1 \leftrightarrow 5$	$1 \leftrightarrow 4$	$1 \leftrightarrow 7$	$1 \leftrightarrow 6$
$4 \leftrightarrow 5$	$4 \leftrightarrow 6$	$4 \leftrightarrow 7$	$2 \leftrightarrow 6$	$2 \leftrightarrow 7$	$2 \leftrightarrow 4$	$2 \leftrightarrow 5$
$6 \leftrightarrow 7$	$5 \leftrightarrow 7$	$5 \leftrightarrow 6$	$3 \leftrightarrow 7$	$3 \leftrightarrow 6$	$3 \leftrightarrow 5$	$3 \leftrightarrow 4$

3.1 Pairwise Exchange for Power-of-Two Mesh (PEX)

The best algorithm for a hypercube architecture is the pairwise exchange algorithm described in [2, 7] as it guarantees no link contention in the hypercube at every step. This algorithm has also been shown to perform well on the fat tree architecture of the CM-5 [6]. The algorithm is described in Figure 1. It requires $p-1$ steps and the communication schedule is as follows. In step i, $1 \leq i \leq p-1$, each processor exchanges a message with the processor determined by taking the exclusive-or of its processor number with i. Therefore, this algorithm has the property that the entire communication pattern is decomposed into a sequence of pairwise exchanges. The communication schedule of PEX for 8 processors is given in Table 1. The entry $i \leftrightarrow j$ in the table indicates that processors i and j exchange data.

Since each step of PEX involves an exchange between pairs of processors, the maximum number of messages contending for a link at any step is limited by $max(r,c)/2$. An exact expression for the maximum number of messages contending for a link at step i is

$$f(i) = 2^{\lfloor \lg\{max(mod(i,c), i/c)\}\rfloor}$$

Hence, the time taken for step i is

$$T(i) = \alpha + L\ max(\beta_{ex}, f(i)\beta_{sat})$$

The cost of PEX can be determined by summing over all steps of the algorithm :

$$\begin{aligned} T_{PEX} &= \textstyle\sum_{i=1}^{p-1}[\alpha + L\ max(\beta_{ex}, f(i)\beta_{sat})] \\ &= (p-1)\alpha + L \textstyle\sum_{i=1}^{p-1} max(\beta_{ex}, f(i)\beta_{sat}) \end{aligned}$$

```
q = 2^⌈lg p⌉
do j=1, q − 1
   destination = xor(mynumber, j)
   if (destination < p) then
      Exchange with destination
   end if
end do
```

Figure 2: Algorithm for PEX-GEN

3.2 Pairwise Exchange for General Mesh (PEX-GEN)

The PEX algorithm cannot be directly used if the number of processors is not a power-of-two as the exclusive-or function will not create all the required processor pairs in $p-1$ steps. The Pairwise Exchange for General Mesh (PEX-GEN) algorithm described in Figure 2 is an extension of PEX for non power-of-two meshes. The algorithm first finds the smallest power-of-two (say q) greater than the number of processors and uses this number to schedule $q-1$ steps of the pairwise exchange. In each step, every processor checks to see if the calculated destination processor number is less than the actual number of processors. If so, it exchanges data with the processor, else it goes ahead to the next step. Thus, the algorithm requires $q-1$ steps where q is the nearest power-of-two larger than the number of processors. Clearly, the algorithm takes more steps than necessary and many processors remain idle in several of the steps. However, this reduces the link contention in each step. The maximum contention in each step is upper bounded by that in the PEX algorithm.

3.3 PEX-GEN with Shift (PEX-GEN-SHIFT)

The motivation for the Pairwise Exchange for General Mesh with Shift (PEX-GEN-SHIFT) algorithm can be explained with the help of Figure 3(a). Assume that the user has allocated a mesh of 20 processors numbered 0 to 19. The nearest power-of-two larger than 20 is 32, so PEX-GEN will require 31 steps. In the first 15 steps of PEX-GEN, processors 0 to 15 exchange completely among themselves and processors 16 to 19 exchange completely among themselves. In the next 16 steps, processors 0 to 15 exchange with processors 16 to 19. Since there are only 4 processors greater than 15, many of the processors 0 to 15 do not do any communication in many of the last 16 steps. Hence there is high link contention in steps 1 to 15 and very little or no link contention in steps 16 to 31. In general, if there are p processors where p is not a power of two, PEX-GEN will require $q-1$ steps where $q = 2^{\lceil \lg p \rceil}$. In the first $\lfloor (q-1)/2 \rfloor$ steps, the first $q/2$ processors are active and in the remaining steps, several of them are inactive.

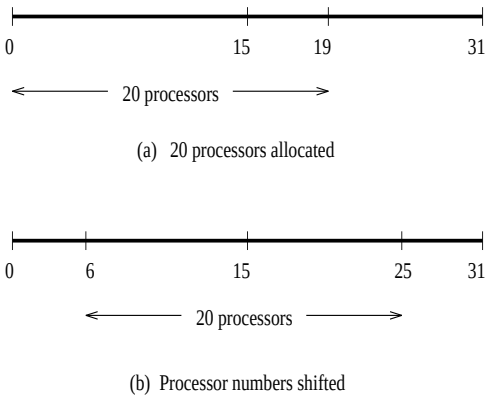

Figure 3: Processor Shift

```
q = 2^⌈lg p⌉
shift = (q − p)/2
myvirtual = mod(mynumber + shift, p)
do j= 1, q − 1
   virtual_destination = xor(myvirtual, j)
   destination = virtual_destination – shift
   if (destination < 0) then
      destination = destination + q
   end if
   if (destination < p) then
      Exchange with destination
   end if
end do
```

Figure 4: Algorithm for PEX-GEN-SHIFT

The Pairwise Exchange for General Mesh with Shift (PEX-GEN-SHIFT) algorithm described in Figure 4 maintains a balance of the number of active and inactive processors in all steps. This is done by defining virtual processor numbers such that the real processors 0 to 19 are numbered 6 to 25 as shown in Figure 3(b). The processor numbers are shifted by an amount equal to half the absolute difference between the number of processors and the nearest higher power of two. Thus the empty space which earlier existed only in the half 16 — 31 is now equally divided among the two halves. So, even in the first 15 steps of the algorithm, there are equal number of idle processors in both halves, which balances the contention among all the steps of the algorithm. This algorithm also takes $q-1$ steps where q is the smallest power-of-two larger than the number of processors. The maximum contention in each step is upper bounded by that in the PEX algorithm.

```
do j=1, p − 1
   destination = MOD(mynumber + j, p)
   source = mynumber − j
   if (source < 0) then
      source = source + p
   end if
   send to destination
   receive from source
end do
```

Figure 5: Algorithm for GEN

3.4 General Algorithm for any Mesh (GEN)

The above algorithms require one less than a power of two number of steps, because they use the exclusive-or function to obtain processor pairs which exchange with each other. For non power-of-two meshes, it would be advantageous to have an algorithm which requires only $p-1$ steps. Figure 5 describes such an algorithm, which we call the General Algorithm for any Mesh (GEN), because it works for any number of processors. In the GEN Algorithm, processor pairs do not exchange with each other. Instead, at step i, a processor j sends data to processor $mod(j+i,p)$ and receives data from processor $j-i$ if $j \geq i$, and $j-i+p$ if $j < i$. Clearly, this algorithm will require only $p-1$ steps, for any value of p.

The maximum contention at step i is given by

$$f(i) = min[mod(i,c), c-mod(i,c)]+min[i/c, (p-i)/c]$$

The total time for all steps can obtained as :

$$T_{GEN} = \sum_{i=1}^{p-1}[\alpha + Lmax(\beta_{sr}, f(i)\beta_{sat})] = (p-1)\alpha + L\sum_{i=1}^{p-1} max(\beta_{sr}, f(i)\beta_{sat})$$

4 Experimental Results

We implemented all the algorithms on the Delta and studied their performance for different mesh configurations and message sizes. The performance of PEX is shown in Table 2. The number of processors is varied from 16 to 512 with message size varied from 256 bytes to 16 Kbytes. Message size refers to the amount of data communicated in each send and receive operation, so the total amount of data communicated increases as the number of processors is increased. Hence, the time taken increases almost linearly with the number of processors.

The performance of PEX-GEN is given in Table 3. We have chosen some mesh sizes which are non power-of-two. We observe that for mesh sizes which are only slightly less than the nearest higher power-of-two, the performance is close to that of PEX for that power-of-two. But, if the mesh size is only slightly higher than the nearest smaller power-of-two, the time taken is almost twice the time taken by PEX for that power-of-two. For example, the time taken by PEX-GEN on a 16×9 mesh is much higher than the time taken by PEX on a 16×8 mesh, but the time taken by PEX-GEN on a 16×14 mesh is very close to the time taken by PEX on a 16×16 mesh. This is because of the difference in the number of steps required. Another interesting observation is that the time taken by PEX-GEN on a 16×30 mesh is in fact higher than the time taken by PEX on a 16×32 mesh. This is because since the processors are numbered in row major order, a change in the number of columns from a power-of-two to a non power-of-two, changes the communication pattern in the mesh completely for an algorithm which uses the exclusive-or function to determine processor pairs. Hence, there is more contention in the 16×30 case than in the 16×32 case.

Table 4 shows the performance of PEX-GEN-SHIFT. In most cases, it performs better than PEX-GEN. Table 5 gives the performance of GEN on a power-of-two mesh. GEN performs better than PEX for small message sizes and small number of processors. However, for large number of processors (≥ 64) and large message sizes ($>$ 1 Kbytes) PEX performs

Table 2: Performance of PEX

Message Size (bytes)	Time in sec. for a Mesh Configuration				
	4 × 4	8 × 8	16 × 8	16 × 16	16 × 32
256	0.004	0.022	0.045	0.094	0.203
1K	0.008	0.064	0.120	0.290	0.860
4K	0.023	0.114	0.355	0.999	3.218
8K	0.034	0.228	0.692	2.068	6.794
16K	0.064	0.441	1.413	4.145	13.61

Table 3: Performance of PEX-GEN

Message Size (bytes)	Time in sec. for a Mesh Configuration				
	4 × 5	6 × 8	16 × 9	16 × 14	16 × 30
256	0.008	0.019	0.085	0.092	0.211
1K	0.017	0.038	0.191	0.270	0.899
4K	0.037	0.091	0.576	0.977	3.588
8K	0.073	0.174	1.188	2.007	7.616
16K	0.138	0.333	2.403	4.056	15.82

Table 4: Performance of PEX-GEN-SHIFT

Message Size (bytes)	Time in sec. for a Mesh Configuration				
	4 × 5	6 × 8	16 × 9	16 × 14	16 × 30
256	0.008	0.019	0.085	0.092	0.211
1K	0.017	0.038	0.188	0.263	0.894
4K	0.036	0.091	0.543	0.933	3.526
8K	0.071	0.170	1.111	1.948	7.515
16K	0.129	0.333	2.242	3.844	15.74

Table 5: Performance of GEN on power-of-two mesh

Message Size (bytes)	Time in sec. for a Mesh Configuration				
	4 × 4	8 × 8	16 × 8	16 × 16	16 × 32
256	0.004	0.016	0.042	0.089	0.283
1K	0.008	0.042	0.123	0.346	1.217
4K	0.018	0.145	0.461	1.220	3.944
8K	0.037	0.290	0.933	2.511	8.007
16K	0.069	0.576	1.947	5.052	16.15

Table 6: GEN on non power-of-two mesh

Message Size (bytes)	Time in sec. for a Mesh Configuration				
	4 × 5	6 × 8	16 × 9	16 × 14	16 × 30
256	0.004	0.015	0.046	0.074	0.246
1K	0.009	0.027	0.146	0.285	1.069
4K	0.025	0.083	0.527	0.998	3.706
8K	0.052	0.186	1.071	2.011	7.752
16K	0.098	0.369	2.182	4.005	15.94

better. The GEN algorithm has a certain amount of asymmetry in the communication in the sense that each communication operation consists of a send to one processor and a receive from some other processor. Thus, the incoming and outgoing messages may traverse a different number of links with different amounts of contention, and the path which has the highest amount of contention adversely affects the communication time. On the other hand, in the PEX algorithm, processor pairs exchange with each other at every step, so the incoming and outgoing messages travel the same number of links with the same amount of contention.

The performance of GEN on non power-of-two meshes is given in Table 6. GEN reduces the number of steps from $q-1$ in PEX-GEN and PEX-GEN-SHIFT, where $q = 2^{\lceil \lg p \rceil}$, to $p-1$. For small number of processors, PEX-GEN performs the best and the improvement in performance is higher when $q-p$ is large. However, if $q-p$ is small and the number of processors is large, the performance of PEX-GEN-SHIFT tends to that of PEX and and the performance of GEN tends to that for a power-of-two mesh. So in this case, PEX-GEN-SHIFT performs better than GEN.

5 Conclusions

In this paper, we have discussed algorithms to perform complete exchange on a wormhole routed mesh with performance results on the Intel Touchstone Delta.

For power-of-two meshes, when the number of processors is small (< 64) and message size is small (< 1 Kbytes), the GEN algorithm performs the best. For larger message and mesh sizes, PEX performs better. For non power-of-two meshes, PEX-GEN-SHIFT performs better than PEX-GEN, but they both require $q-1$ steps where $q = 2^{\lceil \lg p \rceil}$. GEN reduces the number of steps to $p-1$ and performs better than PEX-GEN-SHIFT when $q-p$ is large. As p tends to q, the mesh tends to a power-of-two mesh and the performance of PEX-GEN-SHIFT tends to PEX, while the performance of GEN tends to that for a power-of-two mesh.

Acknowledgments

This work was supported in part by ARPA under contract no. DABT63-91-C-0028. The content of the information does not necessarily reflect the position or policy of the Government and no official endorsement should be inferred. Alok Choudhary's research is also supported by an NSF Young Investigator Award CCR-9357840 and a grant from Intel SSD. This research was performed in part using the Intel Touchstone Delta System operated by California Institute of Technology on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by the Center for Research on Parallel Computation.

References

[1] Barnett, M., Littlefield, R., Payne, D., and van de Geijn, R., "Global Combine on Mesh Architectures with Wormhole Routing", *Proc. of* 7^{th} *Int. Parallel Proc. Symp.*, April 1993.

[2] Bokhari, S., "Complete Exchange on the iPSC/860", *ICASE Technical Report 91-4*, 1991.

[3] Bokhari, S., and Berryman, H., "Complete Exchange on a Circuit Switched Mesh", *Proc. of Scalable High Perf. Computing Conf.*, 1992, pp. 300–306.

[4] Johnsson, S. L., and Ho, C., "Optimum Broadcasting and Personalized Communication in Hypercubes", *IEEE Trans. on Computers*, September 1989, pp. 1249–1268.

[5] Ni, L., and McKinley, P., "A Survey of Wormhole Routing Techniques in Direct Networks", *Computer*, February 1993, pp. 62–76.

[6] Ponnusamy, R., Thakur, R., Choudhary, A., and Fox G., "Scheduling Regular and Irregular Communication Patterns on the CM-5", *Proc. of Supercomputing 92*, November 1992, pp. 394–402.

[7] Scott, D., "Efficient All-to-All Communication Patterns in Hypercube and Mesh Topologies", *Proc. of* 6^{th} *Distributed Memory Computing Conf.*, 1991, pp. 398–403.