Efficient Scheduling of Complete Exchange on Clusters

Anthony T.C. Tam and Cho-Li Wang
The Systems Research Group
Department of Computer Science and Information Systems
University of Hong Kong
{atctam, clwang} @csis.hku.hk

Abstract

One of the performance limitations of clusters is their message passing capability, while complete exchange is known to be the severest communication pattern on all types of message passing machines. In this paper, we focus on the practical issues of designing high-speed complete exchange algorithms on a commodity cluster interconnected by a non-blocking crossbar switch. Four complete exchange algorithms, including, shift exchange, pairwise exchange, group shuffle exchange and synchronous shuffle exchange algorithms are implemented and tested on a cluster platform. To avoid node and link contention, these algorithms schedule the communication at the packet-level. They aim at fully utilizing the available communication bandwidth both in the links and switch and avoiding the Head-Of-Line problem which would stall the pipelines and decrease the overall efficiency. Both the analytical and measured results show that the synchronous shuffle exchange algorithm can achieve the best performance. It can reach 97% of the available bandwidth in our tests; while the group shuffle exchange performs almost as good as the synchronous shuffle exchange algorithm but scales better when it works on an input-buffered switch. Both shift and pairwise exchanges become inefficient when exchanging small messages, as they cannot fill up those network pipelines effectively. Performance studies of the four algorithms on both input-buffered and shared-buffered switches are also reported.

Keywords: All-to-all personalized communication, complete exchange, cluster computing, input-buffered switch, head-of-line blocking

1 Introduction

In recent years, clusters become an important stream in high-performance computing. Their major features are the cost-effectiveness, freedom of configuration and enhancement, which are built from the most up-to-date commodity components. One limitation of commodity clusters is the limited bandwidth available to such systems as compared to traditional parallel computers. Therefore, scheduling of communications is even more crucial to the performance on cluster computing.

Complete exchange, also named as *all-to-all personalized communication*, is a collective operation takes place with a set of processes, and each process has a distinct set of data to transmit to every other process in the set. It is known to be the most stringent communication requirement imposed on the interconnection network. Such a communication pattern occurs in numerous numerical and scientific applications.

Complete exchange communication has been extensively studied in the past. Most of these algorithms are designed to handle the topological constraints of the underlying networks, such as hypercubes, meshes, tori, fat-trees, and multi-dimension networks [1,3,4,6,11,13,14,15]. These algorithms exploit the full performance of the underlying networks by carefully scheduling communications so as to avoid node and link contention. General speaking, algorithms for complete exchange can be classified into two categories, the *direct* or *indirect* approaches. For the direct algorithm, each process directly sends those data blocks to each of the destination processes using separate communication steps. For the indirect algorithm, data blocks for a set of destination processes are combined to form a larger block and are sent to a representative process of the set which is then forward to the correct destination processes. The indirect approach reduces the number of communication steps to reduce the startup cost, however it introduces more traffic in the network. Thus, indirect approach favors small-size messages exchange, while direct approach favors long messages exchange [5,6,12].

To avoid link and node contention, communications are scheduled in multiple phases. Each phase corresponds to a contention-free routing of messages between nodes. This approach restrains the parallelism between different phases, as a process would not enter next phase unless it has finished up the message exchange of the current phase. Besides, not all processes are active in each phase [3,20], because inactive processes have to be kept idle for the whole phase. To achieve this contention-free synchronism, the schedule would induce substantial overhead. First, processes ahead of schedule cannot continue, which means some of the links carry no data. Second, it is hard to enforce this synchronism on a distributed system and synchronization achieved by software solutions (e.g., barrier) may waste the bandwidth. In [21], it pointed out that using a relaxed synchronization scheme which possibly increases the network contention, the overall performance of the complete exchange communication could be improved.

In this paper, we focus on the practical issues of achieving optimality for the complete exchange operation on a commodity cluster interconnected by a high-performance network. Theoretically, connecting all nodes via a single non-blocking crossbar switch provides optimal performance. However, the buffering mechanism used within the switch can hinder its actual performance [17]. Input buffering, in which packets queued at the switch inputs, is known to have the *Head-Of-Line* (HOL) blocking problem. Packets blocked at the head of the queue also block the packets behind them, even if some of these packets are destined for idle output ports. By using queuing analysis, HOL blocking is shown to reduce throughput to 58% even under uniform traffic. To avoid this, output buffers or central shared buffers are used which provide higher performance but require more complex and stringent design. As cluster performance relies on the network performance, which in turn, affects by the switch performance. Use of input-buffered switches could pose a threat on the achievable performance of the complete exchange operation.

Scheduling the communication to avoid contention is crucial in all cases. In our studies, we have a different approach. Instead of having a contention-free schedule at the message level, we construct a contention-free schedule at the packet level with no explicit synchronization is required. We have implemented four complete exchange algorithms, including, *shift exchange, pairwise exchange, group shuffle exchange* and *synchronous shuffle exchange* algorithms on a cluster platform. To compare the performance, they are tested on two different switches with input-buffered and shared-buffered architecture respectively.

The rest of the paper is organized as follows. Section 2 lays down the architecture model of the clusters, which becomes the foundation of our analysis. In section 3, we present and analyze various communication schedules for the complete exchange operation. Section 4 contains our experimental analysis. Finally, conclusions are presented in section 5.

2 Architecture and Communication Model

We need to provide an abstract model that confines the capability and limitations of the cluster systems. In our model, a cluster is defined as a collection of autonomous machines that are interconnected by an indirect network. The network is consisted of a global router switch and assumes fully-connected. Each node can send and receive one data packet in one communication step. For the router switch, we assume it is a packet-switched, synchronous, pipelined network, with cut-through feature. Buffers are provided to resolve output conflicts, but the amount of buffers is assumed to be finite.

To analyze the real performance of those communication schedules, cost formulae are constructed based on our communication model [10]. Under no conflict, the communication cost is encapsulated by the following cost formula,

$$\begin{aligned} T_{ptp}(M) &\approx O_s + (k-1)g + L + O_r + U_r \\ &\approx kg + O_s + L - g + O_r + U_r \\ &\approx kg + T_w \,. \end{aligned}$$

Where

- $\boldsymbol{T_{ptp}}$ **(*M*)** represents the time to finish a remote data transfer of an *M* bytes message from one processor to another;
- ***k* = *M*/*b*,** corresponds to the fragmentation of an *M* bytes message to *k* data packets of size *b* bytes, usually *b*=mtu for optimal performance;
- $\boldsymbol{O_s}$ stands for the software overhead associated with the send process for sending a *b*-bytes data packet;
- $\boldsymbol{O_r}$ and $\boldsymbol{U_r}$ stands for the software overhead induced by the asynchronous reception of a *b*-bytes packet; in which, O_r captures the costs of all kernel events such as interrupt and memory copy, while U_r captures the cost of user-space events such as data processing and high-level protocol handling.
- $\boldsymbol{g_s}$ and $\boldsymbol{g_r}$ encapsulate the minimum time between consecutive injection or reception of *b*-bytes packets to or from the network by the communication hardware; for a homogenous cluster, we can generally assume that $g_s \approx g_r$ and simplify the expression by $g=\max(g_s,g_r)$;
- ***L*** is the network latency in moving the *b*-bytes data packet from the source memory to the destination memory.
- $\boldsymbol{T_w=O_s+L-g+O_r+U_r}$, is a simplified expression which represents the messaging overhead associated to the point-to-point communication.

3 All-to-all Personalized Communication Algorithms

Now we first derived some lower bound conditions for the complete exchange operation on our cluster model. Based on our model, each node is capable to send and/or receive a message in one time unit, such that $(O_s + O_r + U_r) < \max(g_s, g_r) < L$. With this capability, a process can actively send and receive at the same time, but still can fully utilizes the communication network. This implies that the cluster node performance is higher than the network performance. We are only interested in the case of exchanging large data blocks, as this is the general scenario found in most scientific and numerical computations on message passing machines. To simplify the analysis, we assume that each data block corresponds to *k*-data items/packets. So the minimum amount of data items being sent and received in the complete exchange operation per process is $2k(p-1)$ items/packets or $2kb(p-1)$ bytes if each data packet is of size *b* bytes.

We can deduce that the minimal time required for performing the complete exchange operation under such a cluster communication abstraction is

$$\begin{aligned} T_{ata} &= k(p-1)\max(g_s, g_r) + T_w \\ &= k(p-1)g + T_w \, . \end{aligned}$$

Since each cluster node needs to send or receive $k(p-1)$ data packets for the complete exchange, and the minimal time in sending or receiving a data packet of size *b* bytes is bounded by the send gap (g_s) and receive gap (g_r) respectively. As each machine can inject or receive no more than one packet within this gap, we have the above deduction.

Thus, any solution to the *k*-items complete exchange operation on the cluster would be optimal if it takes T_{ata} time units to finish the operation. Carry on with the deduction, we can see that the necessary conditions to satisfy the above optimality are:

1. Each data item/packet is being sent directly to the target node without detour.
2. Each cluster node is actively sending and receiving the $2k(p-1)$ data packets without network stalling during the whole course of operation.

With the assumption of complete-connected topology, links are sufficient but contention still appears if message transmissions are not well scheduled. The distributed nature of the clusters cannot afford to have a lock-step schedule as the synchronization overhead hurts the overall performance. Besides, if the global switch is of input-buffered architecture, the performance of the complete exchange could be restrained by the head-of-line phenomenon. In the following sections, we review the communication schedules used by different algorithms for the complete exchange operation, and use our communication model to analyze their performance on cluster machines.

3.1 Shift Exchange

This algorithm is the simplest way to schedule communications between different nodes without node contention. Since a process needs to send and receive data packets with *p-1* partners, this algorithm takes *p-1* rounds. During each round, each process sends out *k* data packets to a partner, and receives *k* packets from another partner, which is determined by a shift pattern.

```
Shift Exchange Algorithm
for i = 1 to p-1 do
    from = (myid+p-i) mod p
    to = (myid+i) mod p
    for (s=1 to k) & (r=1 to k) in parallel do
        status = send_item_to(to_s, to)
        if (status = success)
            inc s
        endif
        status = recv_item_from(from_r, from)
        if (status = success)
            inc r
        endif
    endfor
endfor
```

Figure 1 Shift Exchange Algorithm

Figure 1 shows the pseudocode that summaries the shift exchange algorithm. In each round, each node directly sends out all the data packets to the corresponding destination. Since each node uniquely maps to one *sendto* and *recvfrom* partners at each round, no node contention is observed. However, the non network-stalling condition is not enforced in this communication schedule. Although there is no explicit synchronization appeared between consecutive rounds, and both send and receive operations are of non-blocking semantics, the p-1 communication rounds have an implicit synchronization cost which introduces bubbles to the network pipelines. For example, in each round, both the send and receive channels are idle until the first byte of the first packet is being injected into the network. Similarly, after receiving the last byte of the last packet, both channels are idle until the cluster node has finished handling the last packet of this round. The predicted total communication cost for this complete exchange operation is

$$\begin{aligned} T_{shift} &\approx (p-1)(O_s + kg + L - g + O_r + U_r) \\ &\approx kg(p-1) + (p-1)T_w \end{aligned}$$

From the cost formula, we see that this algorithm is not optimal as there is a startup overhead which is proportional to the number of cluster nodes.

3.2 XOR Pairwise Exchange

This algorithm (as shown in Figure 2) is similar to the shift exchange algorithm with the exception of having a unique partner for both send and receive operations in each round. The communication cost of this algorithm coincides with that of the shift exchange algorithm as both algorithms involve the same number of message transmissions and receptions in each round, such that from the analytical viewpoint we have

$$T_{XOR} = T_{shift} \; .$$

```
XOR Pairwise Exchange Algorithm
for i = 1 to p-1 do
    partner = myid XOR 2^i
    for (s=1 to k) & (r=1 to k) in parallel do
        status = send_item_to(partner_s, partner)
        if (status = success)
            inc s
        endif
        status = recv_item_from(partner_r, partner)
        if (status = success)
            inc r
        endif
    endfor
endfor
```

Figure 2 XOR Pairwise Exchange Algorithm

One major drawback of the XOR pairwise exchange is the requirement of $p = 2^X$ in order to symmetrically pairing up all the cluster nodes. For the case with $p \neq 2^X$, the number of communication rounds in XOR pairing becomes $2^{\lceil log_2 p \rceil} - 1$, and during each round, not all the cluster nodes find a matching partner.

3.3 Generalized Pairwise Exchange

The limitation of the XOR pairing lies on the bitwise operation, which is perfect only for cluster sizes that are power of two. It would be beneficial if there exist other pairing mechanisms that are more general in nature.

The pairing problem can be formulated as an edge-coloring problem of a connected graph. It is defined as:

Given a graph G=(V,E), with $|V|$=p nodes connected by a set of edges (E), what is the minimum number of colors needed to color the edges of G so that no two adjacent edges are assigned the same color.

In graph theory, this is also defined as the edge-chromatic index $\chi'(G)$ of G. The solution to this problem corresponds to the pairing problem, as the set of edges with the same color maps directly to the set of disjoint pairs in each communication round. The edge-chromatic index becomes the number of communication rounds for the complete exchange operation.

```
Generalized Pairwise Exchange
Round = odd(p) ? p : p-1
for i = 1 to round do
    partner = EdgeColor(i,myid,p)
    if (partner = -1)
        I am idle in this step
    else
        for (s=1 to k) & (r=1 to k) in parallel do
            status = send_item_to(partner_s, partner)
            if (status = success)
                inc s
            endif
            status = recv_item_from(partner_r, partner)
            if (status = success)
                inc r
            endif
        endfor
    endif
endfor

EdgeColor(i, myid, p)
χ′ = odd(p) ? p : p-1
if (myid < χ′)
    v = (i + χ′ - myid) mod χ′
else
    v = odd(i) ? (((i+χ′)/2) mod χ′) : i/2
endif
if (odd(p) AND v = myid)
    return -1
else if (v = myid)
    return χ′
else
    return v
endif
```

Figure 3 Generalized Pairwise Exchange Algorithm

For our case, as we assume a complete graph, the solution to our schedule problem is to find an algorithm for edge-coloring the complete graph. Due to the unique feature of the complete graph, there exists a simple numerable solution similar to the XOR bitwise operation for $p \geq 3$, and is being described and proved in [2]. By incorporated this algorithm to the pairwise complete exchange scheme, we have the generalized pairwise exchange solution to the complete exchange operation (Figure 3).

Regarding the communication complexity, this algorithm has the same communication cost as compared to the shift exchange algorithm for all even cases. As for the odd cases, the performance is only slightly deteriorated with one additional round. Hence, the cost formula for this algorithm is

$$T_{GPE} = \begin{cases} T_{shift} & p \text{ is even} \\ kpg + pT_w & p \text{ is odd} \end{cases}$$

3.4 Synchronous Shuffle Exchange

The above three algorithms having a messaging overhead which is depended on the number of communication rounds, hence, the overhead is proportional to the number of cluster nodes. If k is small and p is large, we would expect to have poor performance. A simple solution to this problem is by removal or reduction of this messaging overhead. We observe that the above communication schedules are arranged to avoid node contention at the message communication level, such that during the exchange, the whole message (k data packets) is being sent "continually" to the destination.

Synchronous Shuffle Exchange Algorithm

```
for (s=1 to k) & (r=1 to k) in parallel do
    for (i_s= 1 to p-1) & (i_r= 1 to p-1) do
        to = (myid+i_s) mod p
        from = (myid+p-i_r) mod p
        status = send_item_to(to_s, to)
        if (status = success)
            inc i_s
        endif
        status = recv_item_from(from_r, from)
        if (status = success)
            inc i_r
        endif
    endfor
endfor
```

Figure 4 Synchronous Shuffle Exchange Algorithm

The synchronous shuffle schedule (Figure 4) effectively multiplexes all the p-1 messages in a single round by applying a contention-free schedule on each data packet. This can significantly reduce the overall messaging overhead. Based on that packet-level scheduling, at a particular instant i^j (assume logically synchronized), each process is sending the j^{th} packet for the process *(myid+i)%p* directly. As each process can uniquely match to different process at this instant, this schedule guarantees no two packets are directed to the same destination at that instant, thus no node contention. Therefore, the predicted communication cost for this complete exchange operation is

$$\begin{aligned} T_{perfect} &\approx O_s + kg(p-1) + L - g + O_r + U_r \\ &\approx kg(p-1) + T_w \ . \end{aligned}$$

From the cost formula, we notice that the messaging overhead is kept constant, and is not depended on p or k. Regards to the communication complexity, this cost formula matches exactly to our optimal formula T_{ata}. This corresponds to a better utilization of the network when the non-stalling requirement is attained.

3.5 Group Shuffle Exchange

Synchronous shuffle exchange algorithm is pushing the cluster system to its limits. If every cluster node is well synchronized, every operation is executed on schedule, and the network resources are scalable, the uniform permutation scheme of the synchronous shuffle exchange can be finished in minimal time.

However, in reality, logical synchronization is not enforced together with the distributed nature of the cluster system, random delays between events could break the uniformity and result in "transient hot-spot". Furthermore, even with uniform traffic, not all switch architectures can sustain such a heavy loading, for example, the maximum throughput of an input-buffered multistage network under uniform traffic is about 62% of the theoretical bandwidth, while output or shared buffered network could reach 90% of the theoretical bandwidth [17]. This is caused by the head-of-line blocking of the input-buffered switches, or the *tree-saturation* phenomenon arising as a result of hot spot in a buffered multistage network. When two or more packets contend for the same output link, or for a shared internal link, blocking of conflicting packets would result in routing delay. Observed that the more packets are targeting to the same output link, the higher chance of having conflicts even under a uniform traffic pattern. We may expect that the synchronous shuffle exchange algorithm may not be scaling linearly with the cluster size on input-buffered switches since the head-of-line problem becomes more serious.

Group Shuffle Exchange

$$\text{round} = \left\lceil \frac{p-1}{\omega} \right\rceil \qquad \text{// assume } p \text{ is even}$$

```
for i = 1 to round do
    group = {}
    for j = 1 to ω do
        group_j = EdgeColor((i-1)*ω+j,myid,p)
    endfor
    for (s=1 to k) & (r=1 to k) in parallel do
        for (j_s= 1 to ω) & (j_r= 1 to ω) do
            partner = group_j
            status = send_item_to(partner_s, partner)
            if (status = success)
                inc j_s
            endif
            status = recv_item_from(partner_r, partner)
            if (status = success)
                inc j_r
            endif
        endfor
    endfor
endfor
```

Figure 5 Group Shuffle Exchange Algorithm

Group shuffle exchange algorithm (Figure 5) is a mixed approach that combines the pairwise exchange and the synchronous shuffle exchange algorithms. The main idea of group shuffle exchange is to overcome the HOL blocking problem but still achieving comparable performance as compared to the synchronous shuffle exchange scheme. Observed in pure pairwise exchange scheme, packets appear in each input port are destined to an unique outgoing port, thus HOL blocking is avoided even under input-buffered switch. However, in a pairwise exchange schedule, we need at least p-1 communication rounds, thus, the startup cost is linearly proportional to the number of participating nodes. For the group shuffle exchange, we reduce the number of communication round to $\left\lceil \frac{p-1}{\omega} \right\rceil$. In each round, a processor is performing a synchronous shuffle exchange with at most ω partners. The main idea of the algorithm is to limit the degree of fan out (ω) during individual shuffle exchange phases, while keeping the number of communication rounds to minimal.

As this communication schedule comprises of more communication rounds, the startup cost of this algorithm would be higher than that of the synchronous shuffle exchange but lower than pairwise exchange algorithm. The expect communication cost formula for this algorithm is, (assume ω divides p-1)

$$\begin{aligned} T_{group} &\approx \frac{p-1}{\omega}\left(O_s + kg\omega + L - g + O_r + U_r\right) \\ &\approx kg(p-1) + \frac{p-1}{\omega} T_w \ . \end{aligned}$$

4 Experimental Results

Our experimental platform is a cluster consists of 16 standard PCs running Linux 2.0.36. Each node is equipped with a 450MHz Pentium III processor with 512KB external L2 cache and 128MB of main memory. Our interconnecting network is the standard Fast Ethernet driven by our Direct Point (DP) communication system [18]. Each node includes a DEC21140-based Fast Ethernet card and is connected to a Fast Ethernet Switch. Two IBM switches with different internal architectures are tested. One is the model 8275-326, which consists of 24 ports, supports virtual cut through packet switching, and is being revealed as an input-buffered architecture. Another switch is the model 8275-416 that consists of 16 ports, supports store-and-forward packet switching, and is being revealed as a shared-buffered architecture. To compare the performance of those direct algorithms, we have implemented the shift exchange, pairwise exchange, group shuffle exchange and synchronous shuffle exchange algorithms on this cluster platform.

To evaluate the performance of these algorithms, we first need the model parameters of our experimental cluster. Table 2 shows all the necessary model parameters for this cluster and are derived from the benchmark tests as described in [10]. We opt to use a constant value for most of the parameters as all experiments are conducted with a fixed size packet, which is the maximum payload (1492 bytes) available for a DP packet.

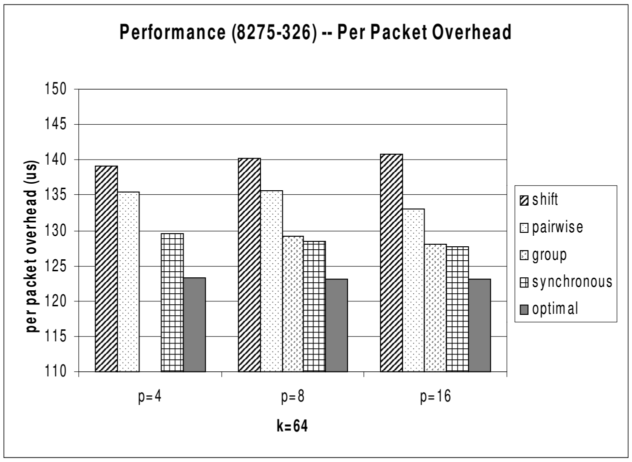
Table 1 Parameters for modeling the point-to-point communication on our cluster

parameters	O_s	g_s	g_r	O_r	U_r	L for 326	L for 416
Time (µs)	12.5	122	123	20	7	0.3387p+149	0.3413p+264

From the table, we observe that our cluster platform satisfies the assumption of $(O_s + O_r + U_r) < g < L$. Therefore, the performance bottleneck of the complete exchange operation falls on the communication network.

4.1 Complete Exchange Performance

First, we validate the optimality of these algorithms by comparing the measured results with the optimal prediction. Figure 6 and 7 show the analysis on the per packet overhead of these direct algorithms for p = 4,8,16 with k = 64 on the same cluster but interconnected by the two IBM switches respectively. The per packet overhead is a metric used to measure the average latency experiences by the processor in exchanging a data item. This is calculated by dividing the measured time with the total number of data items. The measured results conform with our analytical studies, that is, the synchronous shuffle exchange algorithm is the best amongst all of these direct algorithms, and their performance rankings match with their cost formulae in the previous section. As the implementation of the pairwise exchange algorithm utilizes the piggyback optimization, we can see that the pairwise exchange algorithm works faster than the shift exchange algorithm in all cases. Furthermore, for the group shuffle exchange, we use ω=5 at p=16, i.e. there are $\left\lceil \frac{16-1}{5} \right\rceil = 3$ rounds and ω=4 at p=8 (2 rounds), it works as the second best which is almost closed to the performance of the synchronous shuffle exchange algorithm (which only has one communication round).



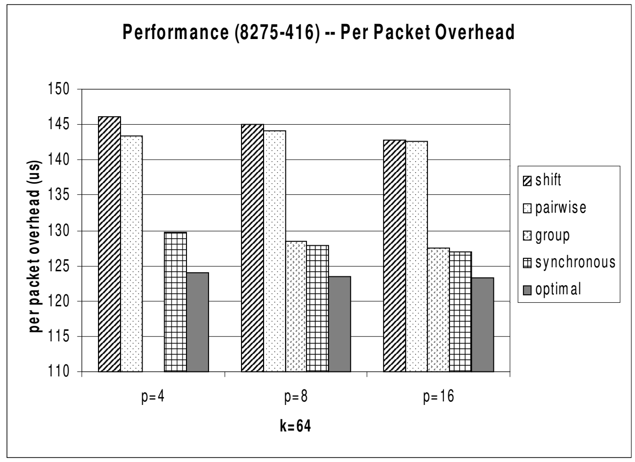
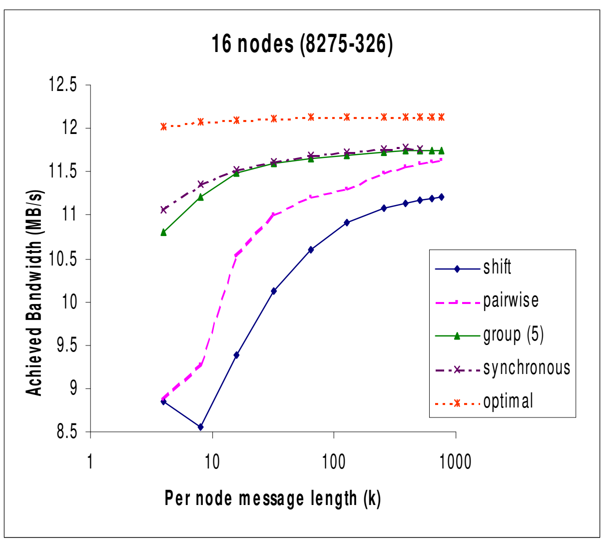


Figure 6 Performance of different complete exchange algorithms with *k*=64 on a 16 nodes cluster inter-connected by the 8275-326

Figure 7 Performance of different complete exchange algorithms with *k*=64 on a 16 nodes cluster inter-connected by the 8275-416

Figure 8 and 9 show the achievable bandwidth of these direct algorithms for p=16 against different values of k ranges from 4 to 768 on two different switches. The data shows that for very long message (large k), both shift exchange and pairwise exchange are catching up with the performance of the synchronous shuffle exchange and group shuffle exchange algorithms, but we still observe there are relative differences between these algorithms. For instances, at k=256, the achieved bandwidth for shift, pairwise, group shuffle, and synchronous shuffle exchanges are 11.07, 11.48, 11.72 and 11.77 MB/s respectively on the IBM 326 switch.



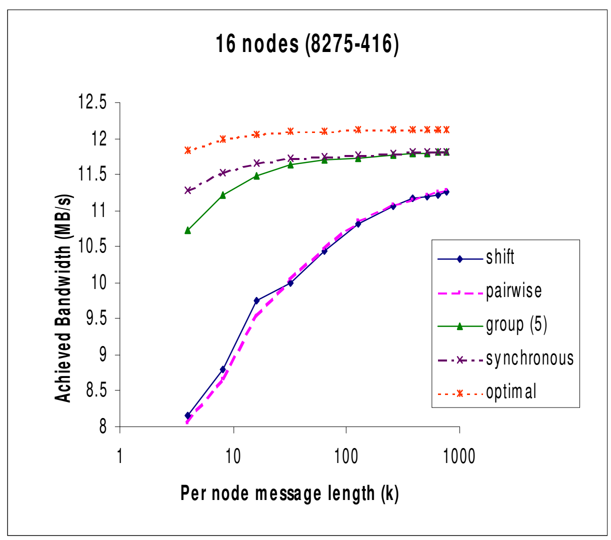


Figure 8 Achievable bandwidth per node for different algorithms as compared to the optimal prediction on IBM 326 switch

Figure 9 Achievable bandwidth per node for different algorithms as compared to the optimal prediction on IBM 416 switch

For messages of short length (small *k*), we notice there are significant differences between those direct algorithms. As explained by their cost formulae, for small *k*, both shift and pairwise exchanges perform poorly due to the high startup cost and the implicit synchronization between communication rounds. As can be seen, the synchronous shuffle and the group shuffle exchanges effectively reduce these costs, and perform much better even up to *k*=100 (that corresponds to per node message size of 145KB).

4.2 Comparison of Different Switching Mechanisms

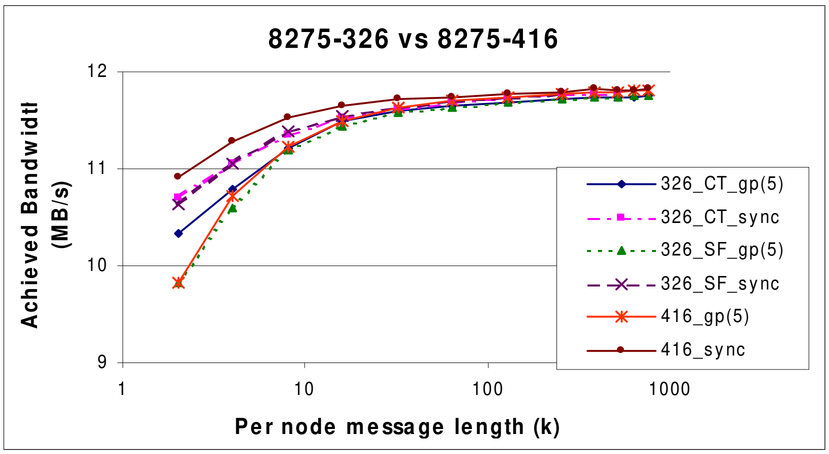


Figure 10 Comparison of performance between a virtual cut through switch (326) and a store-and-forward switch (416) on the synchronous shuffle and group shuffle algorithms

As complete exchange operation induces such an intensive communication pattern, it stresses on the communication subsystem severely. Hence, it could be used as a tool to compare the relative performance of different hardware components.

We make use of the above achievable bandwidth concept to compare the performance between the two switches. Figure 10 shows the achieved bandwidth of the group shuffle exchange and synchronous shuffle exchange algorithms for *p*=16 on two switches. Generally, both switches have the similar performance for long messages, so not much difference between the cut-through or store-and-forward modes. But for short message sizes, we can see their difference. If we cannot fully utilize the network, then we cannot effectively mask away the higher latency of store-and-forward switching, this is being reflected by the slower performance of the group shuffle exchange algorithm on both switches when they are working in store-and-forward mode. Interestingly, we see that the switch internal buffering mechanism does affect the overall performance. Observed that the performance of the synchronous shuffle exchange algorithm on the 8275-416 switch is always better than the same algorithm on the 8275-326 switch with both packet forwarding modes.

4.3 Effects on Group Size ω and Problem Size *k*

In figure 11 we have a comparison of the efficiency of the group shuffle exchange algorithms with different group size ω for *p*=16 on various per node message length *k*=4, 40, 400. Technical speaking, with ω=1, the synchronous shuffle exchange reduces to the pairwise exchange, while ω=15 corresponds to the synchronous shuffle exchange. Clearly, the figure shows that synchronous shuffle exchange always performs the best in all aspects. The group shuffle exchange performs considerably well even under small ω for medium to large messages as compared to pairwise exchange.

Although group shuffle exchange cannot out-perform the synchronous shuffle exchange, it becomes more favorable when we scale the problem size on an input-buffered switch. Figure 12 shows the attainable efficiencies of those direct algorithms when operated on an input-buffered switch (8275-326) with *p*=16 and ω=5.

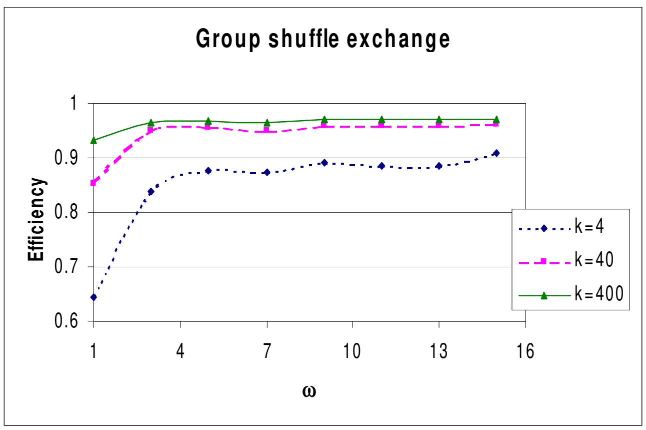


Figure 11 The efficiency of the group shuffle exchange algorithm as a function of group size ω for various message sizes *k* on 16 nodes.

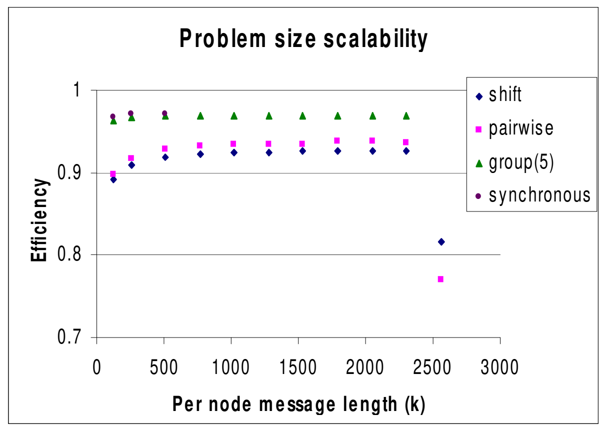


Figure 12 Comparison of the problem size scalability of those direct algorithms on an input-buffered switch for *p*=16.

For this test, an algorithm is considered to stop operating when it fails to terminate within reasonable time on a test of 50 iterations, this reflects there is considerably slowdown due to the HOL blocking. Although synchronous shuffle exchange has the best performance, it cannot continue for *k*>512, such that the per node message length is of 746KB. While the efficiency of group shuffle exchange is only slightly less than the synchronous shuffle exchange, it continues to operate sub-optimally until *k*>2304 (around 3357KB). Lastly, we observe that both pairwise exchange and shift exchange continue to work for very large message length (around 2560), but the performance drops dramatically as the total communication buffer size is approaching the machine limit.

5 Conclusions

Scheduling strategies for complete exchange, that avoid node and link contentions, have been studied on classical parallel machines, and most were focused on exchanging small messages which resulted in developing a class of indirect algorithms. For those existing direct algorithms, which are designed for exchanging long messages on particular network topologies, cannot efficiently map to the cluster platforms as these algorithms require high synchronization cost or could not fully utilized the underlying resources.

Based on the architecture and communication model of cluster, we observe that to achieve optimal result, the network pipes should be fully utilized and any waiting stage would stall the pipelines and decrease the overall efficiency of the communication. This has been validated analytically and experimentally on the shift exchange and pairwise exchange algorithms. Pairwise exchange algorithm uniquely pairs up those cluster nodes in each communication rounds and avoids node and link contentions. It works efficiently when message size (i.e., k) is large. However, for exchanging small messages on a large cluster, the waiting time incurred by each communication round cannot be masked away and results in poor performance. The same problem appears on the shift exchange algorithm. But the pairwise exchange can outperform the shift exchange, as logical synchronization is needed only between a pair of nodes at each round. Thus, it is usually faster than the shift exchange algorithm which involves coordination between three nodes. Moreover, optimization technique such as piggyback acknowledgement can be adopted in the pairwise exchange algorithm to hide messaging latencies.

To avoid stalling the network, we need to completely remove all waiting time, and this is achieved by the synchronous shuffle exchange algorithm. This algorithm logically schedules the communication at the packet level in a pattern that avoids both node and link contentions. As waiting time is removed, those links are better utilized. As a result, we can exchange all the messages by minimal time, hence, achieved optimal. Theoretically, at the same instant, all packets arrived to different input ports are destined to different output ports according to other contention-free schedule, therefore, this schedule should operate efficiently on any non-blocking network. However in reality, since no global clock is implemented and the operations are not lock step synchronized. This shuffle pattern could still induce head-of-line blocking problem on the input-buffered switch. To alleviate the problem, group shuffle exchange is devised. This algorithm limits the degree of fan out and introduces minimal waiting time during the communication. From the measured results we show that it performs almost as good as the synchronous shuffle exchange algorithm but scale better than the synchronous shuffle exchange.

6 References

[1] D.P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng, and J.N. Tsitsiklis, Optimal Communication Algorithms for Hypercubes, *Journal of Parallel and Distributed Computing*, 11, pp. 263-275, 1991.

[2] J. Gross and J. Yellen. Graph Theory and its Applications, CRC press, 1999

[3] C.C. Lam, C-H. Huang, and P. Sadayappan. Optimal Algorithms for All-to-all Personalized Communication on Rings and Two-Dimensional Tori, *Journal of Distributed and Parallel Computing*, No. 43, pp. 3-13, 1997.

[4] Dimakopoulos, V.V.; Dimopoulos, N.J. A theory for total exchange in multidimensional interconnection networks, *IEEE Trans. on Parallel and Distributed Systems,* Volume: 9 7 , July 1998 , pp. 639 –649.

[5] S. H. Bokhari. Multiphase Complete Exchange on Paragon, SP2 & CS-2, September 1995 (http://www.icase.edu/Dienst/Repository/2.0/Body/ncstrl.icase/TR-95-61/pdf)

[6] S. H.Bokhari. Multiphase complete exchange: a theoretical analysis . *IEEE Trans. on Computers*, Volume: 45 2 , Feb. 1996 , pp. 220 –229.

[7] The Computational Plant project at Sandia National Laboratories.

(http://www.cs.sandia.gov/cplant/)

[8] The Avalon Cluster (http://cnls.lanl.gov/avalon/)

[9] The Parnass2 Cluster (http://wwwwissrech.iam.uni-bonn.de/research/projects/parnass2/)

[10] Anthony T.C. Tam and Cho-Li Wang. Realistic Communication Model for Parallel Computing on Cluster, *International Workshop on Cluster Computing* 1999.

[11] Yu-Chee Tseng; Ting-Hsien Lin; Gupta, S.K.S.; Panda, D.K., Bandwidth-optimal Complete Exchange on Wormhole-routed 2D/3D Torus Networks: a Diagonal-propagation Approach, *IEEE Trans. on Parallel and Distributed Systems*, Vol 8, No. 4 , April 1997, pp. 380 –396.

[12] Jehoshua Bruck, Ching-Tien Ho, Shlomo Kipnis and Derrick Weathersby; Efficient Algorithms for All-to-all Communications in Multi-port Message-passing Systems; *the 6th Annual ACM Symposium on Parallel Algorithms and Architectures* , 1994, pp. 298 - 309

[13] Christina Christara Xiaoliang Ding and Ken Jackson. An Efficient Transposition Algorithm for Distributed Memory Computers.

(http://www.cs.toronto.edu/~ftp/pub/reports/na/transpose.99.ps.Z)

[14] Y.J. Suh and S. Yalamanchili. All-to-all Communication with Minimum Start-up Costs in 2D/3D Tori and Meshes, *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, no. 5, May 1998, pp. 442-458.

[15] Ponnusamy, R.; Thakur, R.; Choudhary, A.; Fox, G. , Scheduling regular and irregular communication patterns on the CM-5, *Supercomputing '92.*, 1992 , pp. 394 -402

[16] Dhabaleswar K. Panda., Issues in Designing Efficient and Practical Algorithms for Collective Communication on Wormhole-Routed Systems, *Proceedings of the 1995 ICPP Workshop on Challenges for Parallel Processing*, pp. 8-15.

(file://ftp.cis.ohio-state.edu/pub/communication/papers/icpp95-collective.ps.Z)

[17] B. Zhou and M. Atiquzzaman, A Performance Comparison of Buffering Schemes for Multistage Switches, *IEEE First International Conference on Algorithms and Architectures for Parallel Processing*, April 1995, Brisbane, Australia, pp. 809-818.

(http://www.engr.udayton.edu/faculty/matiquzz/papers/bin-ica3pp-buf-cam.ps)

[18] C.M. Lee, A. Tam, and C.L. Wang. Directed Point: An Efficient Communication Subsystem for Cluster Computing. *International Conference on Parallel and Distributed Computing Systems (IASTED)*, October 1998.

[19] D.S. Scott. Efficient All-to-All Communication Patterns in Hypercube and Mesh Topologies. *Proceedings of the 6th Distributed Memory Concurrent Computers Conference*, pp. 398-403, 1991.

[20] Y.C. Tseng and S.K.S. Gupta. All-to-All Personalized Communication in a Wormhole-Routed Torus. *IEEE Transaction on Parallel and Distributed Systems*, Vol. 7, No. 5, pp. 498-505, May 1996.

[21] S.H. Bokhari and D.M. Nicol. Balancing Contention and Synchronization on the Intel Paragon. *IEEE Concurrency,* Vol. 5, No. 2, pp. 74-83, April-June 1997.