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

In this paper, we focus on the practical issues of designing efficient 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 studied and tested on a cluster platform. These algorithms feature their own communication schedule to avoid node and switch contention so as to fully utilize the available bandwidth. 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 under the Head-Of-Line phenomenon. Performance studies of the four algorithms on both input-buffered and shared-buffered switches are also reported.

Keywords: complete exchange, all-to-all personalized communication, cluster computing, head-of-line

1 Introduction

Complete exchange, also known 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. To minimize the communication delay, all processes are actively participating in the communication. It is known to be the most stringent communication requirement imposed on the interconnection network.

Complete exchange operation has been extensively studied in the past. Most of the studies are focused on designing communication schedules to avoid contention delay induced by the topological constraints of the underlying networks, such as hypercubes [1], meshes [7], tori [9], fat-trees [6], multistage interconnection networks [11] and multi-dimension networks [2]. In recent years, clusters become an important stream in high-performance computing. They are usually built on switch-based interconnects like Fast/Gigabit Ethernet, Scalable Coherent Interface (SCI) and Myrinet. These switches provide flexibility in network design, and particularly, some technologies even support non-blocking switching capability up to a few hundred nodes [3].

Theoretically, connecting all nodes via a single non-blocking router switch provides optimal performance. However, contention still exists if message exchanges are not well scheduled, e.g. contention for the same outgoing port. Furthermore, the distributed nature of clusters cannot afford to have a lock-step schedule as synchronization cost on these platforms is high, since they are implemented by software means. Besides, the buffering mechanism used within the switch could hinder its actual performance. While there are many variations in switch architectures, most switches fall into one or a combination of three basic types: input-buffered, output-buffered and shared-buffered.

With the input-buffered architecture, incoming packets are queued in buffers, one per input port. This is the simpliest design as the internal speed of the buffer only operates at the same speed as the input/output links. However, it is known to have the *Head-Of-Line* (HOL) blocking problem. Packets block 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.

While for the other two architectures, output-buffered and shared-buffered, they do not suffer from the HOL problem and thus support higher throughput than input buffered switch. However, due to technological constraints, the performance of the buffers

[*]This research was supported by Hong Kong Research Grants Council grant HKU 7030/97E.

must be fast enough to sustain simultaneous access [10], and this requires more complex and stringent design.

In this paper, we focus on the practical issues of achieving optimality for the complete exchange operation on a commodity cluster interconnected by a non-blocking network. 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.

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 System Model

In our model, a cluster is defined as a collection of autonomous machines that are interconnected by 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, full duplex, 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 performance of those communication schedules, cost formulae are constructed based on our communication model [8]. Under no conflict, the point-to-point communication cost is encapsulated by the following cost formula,

$$T_{ptp}(M) = O_s + (k-1)g + L + O_r + U_r \qquad (1)$$

where

- $T_{ptp}(M)$ represents the total time spent on a point-to-point communication of transfering a M-byte message between two user processes on different nodes;
- $k = \frac{M}{b}$, corresponds to the fragmentation of a M-byte message to k data packets of size b bytes, usually $b = mtu$ for optimal performance;
- O_s stands for the software overhead associated with the send process for sending a b-byte packet;
- O_r and 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;
- g_s and g_r encapsulate the minimum time between consecutive injection or reception of b-byte 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 of moving the b-byte packet from the physical memory of the source node to the physical memory of the destination node;

To simplify the expression, let $T_w = O_s + L - g + O_r + U_r$, and the point-to-point communication cost becomes $T_{ptp}(M) = kg + T_w$.

3 Complete Exchange Algorithms

Based on the system model, each node is capable to send and/or receive a message in one time unit, such that $(O_s + O_r + U_r) < g < L$. With this capability, a process can actively send and receive at the same time, thus can fully utilizes the communication network. To simplify the analysis, we assume that each data block corresponds to k data packets. So the minimum amount of packets being sent and received in the complete exchange operation per process is $2k(p-1)$ packets or $2kb(p-1)$ bytes if each data packet is of size b bytes.

As the minimal time in sending or receiving a packet of size b bytes is bounded by the send gap (g_s) and receive gap (g_r), and each machine can inject or receive no more than one packet within this gap, so we deduce that the minimal time required for the complete exchange operation under such a cluster communication abstraction is

$$\begin{aligned} T_{ata} &= O_s + \max((k(p-1)-1)g_s, (k(p-1)-1)g_r) \\ &\qquad + L + O_r + U_r \\ &= k(p-1)g + T_w \qquad (2) \end{aligned}$$

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. The necessary conditions to satisfy the above optimality are:

1. Each data packet is being sent directly to the target node without detour.
2. Each cluster node is actively sending and receiving the data packets without network stalling during the whole course of operation.

3.1 Shift Exchange

This algorithm is the simplest way to schedule communications without node contention. It takes p-1 rounds, and during each round, each process sends out k items to a partner, and receives k items from another partner, which is determined by a shift pattern.

Algorithm 1 Shift Exchange

```
for $i$ = 1 to $p$-1 do
    from_partner = (myid+$p$-$i$) mod $p$
    to_partner = (myid+$i$) mod $p$
    for ($s$=1 to $k$) & ($r$=1 to $k$) *in parallel* do
        if (send_item_to(to_partner$_s$, to) = success) then
            inc $s$
        if (recv_item_from(from_partner$_r$, from) = success) then
            inc $r$
    end
end
```

As depicted in Alg. 1, during each round, each node uniquely maps to one *sendto* and *recvfrom* partners, thus no node contention is achieved. However, the non network-stalling condition is not enforced under this scheme. Although there is no explicit synchronization appeared between consecutive rounds and both send and receive operations are of non-blocking semantics, the p-1 rounds have an implicit synchronization cost that 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 communication cost for this complete exchange operation is

$$\begin{aligned} T_{shift} &= (p-1)(O_s + kg + L - g + O_r + U_r) \\ &= kg(p-1) + (p-1)T_w \end{aligned} \tag{3}$$

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

3.2 Pairwise Exchange

Unlike shift exchange, nodes are pairing up for direct exchange in each round. Traditionally, the pairing pattern is based on the Exclusive Or (XOR) binary operation. The communication cost of this algorithm coincides with that of the shift exchange as both algorithms involve the same number of message transmissions and receptions, such that from the analytical viewpoint, we have $T_{pair} = T_{shift}$.

The major drawback of the XOR bitwise operation is the requirement of $p = 2^X$ in order to symmetrically pairing up all the nodes. For the case with $p \neq 2^X$, the

```
**Procedure** EdgeColor(*round*, myid, $p$)
    $\chi'$ = odd($p$) ? $p$ : $p$-1
    if (myid < $\chi'$) then
        v = (*round* + $\chi'$ - myid) mod $\chi'$
    else
        v = odd(*round*) ? (($\frac{round+\chi'}{2}$) mod $\chi'$) : $\frac{round}{2}$
    if (odd($p$) AND v = myid) then
        return -1            // idle for this round
    else if (v = myid) then
        return $\chi'$
    else
        return v
```

Figure 1: Edgecolor Pairing Algorithm

number of rounds becomes $2^{\lceil log_2 p \rceil} - 1$, and during each round, not all the nodes find a matching partner. The solution to our pairing problem is to find an algorithm for edge-coloring the complete graph. Due to its uniqueness, there exists a simple numerable solution similar to the XOR operation for $p \geq 3$, and is being described and proved in [4]. By incorporated this algorithm (Figure 1) to the pairwise exchange scheme, we have the generalized pairwise exchange algorithm. Under this mapping scheme, the performance is only slightly deteriorated with p communication rounds for all odd cases, instead of having p-1 communication rounds for all even cases.

3.3 Synchronous Shuffle Exchange

The above two algorithms have a messaging overhead which is depended on the number of communication rounds. If k is small and p is large, we would expect to have poor performance. A simple solution to this problem is by reduction of this overhead. The synchronous shuffle schedule (Alg. 2), effective-

Algorithm 2 Synchronous Shuffle Exchange

```
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 $p$
        if (send_item_to(to$_s$, to) = success) then
            inc $i_s$
        if (recv_item_from(from$_r$, from) = success) then
            inc $i_r$
    end
end
```

ly multiplexes all the p-1 messages in a single round by applying a contention-free schedule at the packet level. 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, it guarantees no two packets are directed to the same destination at the same instant, thus no node contention. Therefore, the predicted communication cost for this complete exchange operation is

$$\begin{aligned} T_{syn} &= O_s + kg(p-1) + L - g + O_r + U_r \\ &= k(p-1)g + T_w \end{aligned} \quad (4)$$

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 shows that the scheme can effectively utilize the send and receive channels by multiplexing all the messages seamlessly to a single pipeline flow without unnecessary synchronization delay.

3.4 Group Shuffle Exchange

If every operation is executed on schedule, and the network resources are scalable, then, the 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 this uniformity and result in "transient hot-spot" in the switch.

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 pattern. We may expect that the synchronous shuffle scheme could suffer on clusters with input-buffered switches due to the HOL problem.

Algorithm 3 Group Shuffle Exchange

```
round = $\lceil \frac{p-1}{\omega} \rceil$        // assume p is even
for $i$ = 1 to round do
    group = [ ]
    for $j$ = 1 to $\omega$ do
        group[$j$] = EdgeColor(($i$-1)*$\omega$+j,myid,p)
    end
    for ($s$ =1 to $k$) & ($r$ =1 to $k$) *in parallel* do
        for ($j_s$ = 1 to $\omega$) & ($j_r$ = 1 to $\omega$) do
            to = group[$j_s$]
            from = group[$j_r$]
            if (send_item_to(to$_s$, to) = success) then
                inc $j_s$
            if (recv_item_from(from$_r$, from) = success) then
                inc $j_r$
        end
    end
end
```

Group shuffle exchange algorithm (Alg. 3) is a mixed approach that combines the pairwise exchange and the synchronous shuffle exchange algorithms. The main idea is to overcome the HOL problem but still achieving comparable performance as compared to the synchronous shuffle scheme. In pure pairwise exchange scheme, packets appear in each input port are destined to a unique outgoing port, thus HOL blocking is rare even under input-buffered switch. However, in the pairwise scheme, the startup overhead is linearly proportional to the number of communication rounds, which hinders its efficiency. For the group shuffle exchange, we reduce the number of communication rounds to $\lceil \frac{p-1}{\omega} \rceil$. In each round, a processor is performing a synchronous shuffle exchange with at most ω partners. The main idea of this scheme 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 algorithm comprises of more communication rounds, the startup overhead would be higher than that of the synchronous shuffle scheme but lower than the pairwise scheme. The predicted communication cost for this algorithm is, (assume ω divides p-1)

$$\begin{aligned} T_{group} &= \frac{p-1}{\omega}(O_s + kg\omega + L - g + O_r + U_r) \\ &= kg(p-1) + \frac{p-1}{\omega}T_w \end{aligned} \quad (5)$$

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. The interconnection network is the Fast Ethernet driven by our Direct Point (DP) communication system [5]. Each node includes a DEC21140-based ethernet card and connects 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 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 switching, and is being revealed as a shared-buffered architecture. We have implemented the above algorithms on this cluster platform and compare their performances with the analytical formulae.

To evaluate the performance of these algorithms, model parameters of our experimental cluster are required. Table 1 shows all the necessary model parameters for this cluster and are derived from the benchmark tests as described in [8]. We opt to use a constant value for most of the parameters as all experiments are

Table 1: Model parameters for 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	$0.3387p+149$	$0.3413p+264$

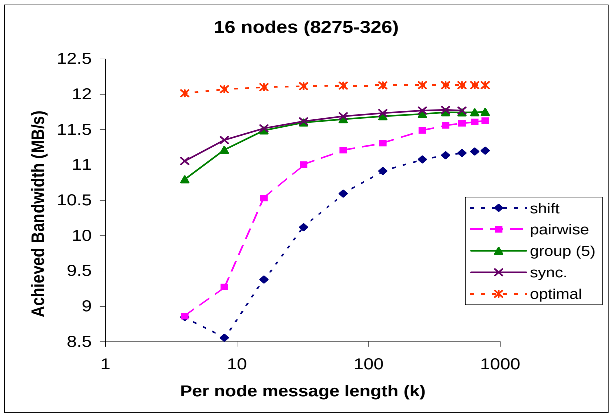

Figure 2: Achievable bandwidth per node for different algorithms as compared to the optimal prediction on 8275-326

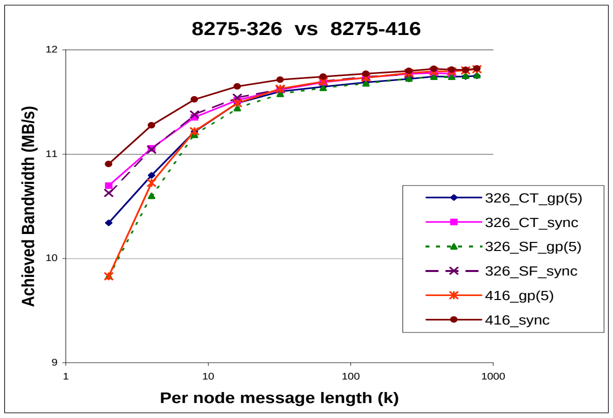

Figure 3: Comparison of performance between a virtual cut through switch (326) and a store-and-forward switch (416) on the synchronous shuffle and group shuffle algorithms

conducted with a fixed size packet, which is the maximum payload (1492 bytes) available for a DP packet.

4.1 Complete Exchange Performance

We validate the optimality of these algorithms by comparing the measured results with the optimal prediction. Figure 2 shows the achieved bandwidth of these algorithms for p=16 against different message sizes (k) ranges from 4 to 768 (data packets) on the same cluster. The achieved bandwidth is a metric which measures the efficiency of the algorithm in utilizing the network. This is calculated by dividing the total data message sizes with the measured communication time. The results show 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 performance differences between these algorithms. For messages of short length (small k), we notice there are significant differences between these algorithms. As explained by their cost formulae, for small k, both shift and pairwise exchanges perform poorly due to the high messaging cost and the implicit synchronization between communication rounds. On the other hand, the synchronous shuffle and the group shuffle exchanges effectively reduce these costs, and perform much better even up to k=100.

4.2 Comparing Switching Mechanisms

Figure 3 shows the achieved bandwidth of the group shuffle scheme and synchronous shuffle scheme 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 messages, we can see their difference. If we cannot fully utilize the network, 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 scheme on both switches when they are working in store-and-forward mode. Moreover, we see that the switch internal buffering mechanism does affect the overall performance. The performance of the synchronous shuffle exchange algorithm on the shared buffered switch (416) is always better than the same algorithm on the input buffered switch (326) with both packet forwarding modes.

4.3 Effects on Problem Size k

We have compared the scalability of these algorithms when operated on an input buffered switch (326) with p=16 and ω=5. 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 message length of each node is of 746KB. While the performance 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 Discussion and Conclusions

Based on the architecture and communication model of cluster, we observe that to achieve optimal result, the network pipes should be fully utilized. Any waiting stage would stall the pipelines and decrease the overall communication efficiency. Pairwise exchange algorithm uniquely pairs up those cluster nodes in each communication rounds and avoids node and switch 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 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 switch contentions. As waiting time is removed, those links are better utilized, and 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 our 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 induce head-of-line blocking 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.

References

[1] D.P. Bertsekas, et al., "Optimal Communication Algorithms for Hypercubes", *Journal of Parallel and Distributed Computing*, Vol. 11, pp. 263-275, 1991.

[2] V.V. Dimakopoulos and N.J. Dimopoulos, "A theory for total exchange in multidimensional interconnection networks", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 9, No. 7 , pp. 639-649, 1998.

[3] Extreme Networks. (`http://www.extremenetworks.com/products/core.asp`)

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

[5] 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)*, 1998.

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

[7] 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, pp. 442-458, 1998.

[8] A.T.C. Tam and C.L. Wang, "Realistic Communication Model for Parallel Computing on Cluster", *International Workshop on Cluster Computing,* 1999.

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

[10] J. Walrand and P. Varaiya. *High-Performance Communication Networks*. Morgan Kaufmann Publishers, 1996.

[11] Y. Yang and J. Wang, "Optimal All-to-All Personalized Exchange in Self-Routable Multistage Networks", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 11, No. 3, pp. 261-274, 2000.