Interconnection Network Switch Architectures and Combining Strategies[1]

Susan Dickey, Allan Gottlieb and Yue-Sheng Liu[2]
Ultracomputer Research Laboratory
Courant Institute of Mathematical Sciences
New York University
715 Broadway, 10th Floor
New York, N.Y. 10003

[1] This work was supported in part by the U.S. National Science Foundation under grant number MIP-8915488, and in part by the U.S. Department of Energy grant number DE-FG-2-88ER25052.

[2] Yue-Sheng Liu is currently at Dow Jones Telerate, Harborside Financial Center 600, Plaza 2, Jersey City, NJ 07311.

Abstract

We have studied a variety of switch architectures for use in the multistage interconnection network of a shared memory multiprocessor. In this paper, we describe a taxonomy of buffered switch architectures and give performance results for the different types we have defined. We investigate two techniques for improving network performance in the presence of contention: the use of multiple handshaking signals and the use of message combining. Simulations results are provided for implementation alternatives using switches of varying capabilities, in order to compare the effectiveness of different methods as system size grows. We show that a practical combining switch design, the two-and-a-half-way combining switch, provides performance as good as other more expensive designs.

Key Words: Network Performance, Interconnection Network, Parallel Processing, Hot-Spot, Combining, MIMD Shared Memory Machine.

1. Introduction

A multistage interconnection network connecting processing elements (PEs) to an equal number of shared memory modules (MMs) can be used to solve the communication problem between hundreds or thousands of PEs in a shared memory multiprocessor. The most commonly discussed and analyzed class of multistage networks includes delta networks [26], the Omega network [21], and others, all of which can be shown to be essentially equivalent [1]. Such networks can provide bandwidth linear in the number of PEs and latency logarithmic in the number of PEs, allowing the construction of parallel architectures that scale to thousands of processors.

Networks in this class have the properties that exactly one path connects a given network input to a given network output, and that routing can be done locally by a simple scheme in which the output at the *i*th stage in the network is selected by the *i*th bit in a routing address. Many transmissions from different PEs to different MMs may be active in the network simultaneously. Since simultaneous transmissions may require traversal of the same link, some method for handling contention for a link is required. As discussed in section 3, buffers are required to maintain linear bandwidth. This paper describes a variety of buffered switch structures possible in such networks.

Our studies of switch design grew out of a consideration of the options for hardware implementation of the 2×2 buffered crossbar switches in the NYU Ultracomputer [14], and use a traffic model for the individual switches in which uniform memory traffic in the system as a whole results in the uniform distribution of the messages from a given switch input among the outputs of the switch. An $N \times N$ square delta network, as defined in [27], composed of $k \times k$ switches, with $\log_k N = n$ stages, has this property; a given input of a switch will favor one of the k outputs only if there is non-uniformity in the overall memory access pattern. In contrast, this property may not obtain in mesh-connected multiprocessors, where the routing algorithm can result in most messages being routed ‘‘straight-through’’ a switch either horizontally or vertically, even for uniform traffic.

This paper provides a unified framework for describing and analyzing switches that satisfy a practical switch-to-switch protocol, described in section 2. The bandwidth limitation and delay of unbuffered networks is discussed in section 3. Many buffered switch architectures for interconnection networks have been proposed and analyzed; relevant previous work is cited in section 4, where a taxonomy of buffered switch types is presented and performance results are given. Section 5 describes the use of multiple clear-to-send signals to improve performance when one buffer in a switch is full, but the other is not. Several combining strategies and their performances are compared in section 6.

[3] Often such systems will be designed with each PE packaged together with an MM. A direct path may then be provided between this PE and its co-located MM for accessing local data; we will be concerned only with the (shared) memory access through the network.

2. Ultraswitch Protocol

Consider an $N \times N$ interconnection network in an MIMD shared memory machine. We think of the N PEs as being at one side of the network and the N MMs at the other. The network is bidirectional, and we consider it to be composed of two unidirectional networks: a forward network with PEs at inputs and MMs at outputs and a return network with MMs at inputs and PEs at outputs. The return network is the reverse network of the forward network, i.e. the reverse network is obtained by reversing the directions of all edges in the forward network. In our discussion to follow, switches and networks refer to the forward direction except where the return network is explicitly mentioned.

In this study, we restrict the switch nodes to a class that supports a specific communication protocol, the *Ultraswitch protocol,* developed by the NYU Ultracomputer project [10, 31]. By omitting some implementation specific features, our description of the protocol is somewhat more general than the actual Ultracomputer hardware implementation.

In the Ultraswitch protocol, we assume that networks are $n = \log_k N$ stage Omega networks constructed from $k \times k$ switches. There are exactly N/k switches at each stage. On the input side of the network, there are N PEs. The outputs of the network are connected to N MMs. A message is the unit of information sent across the network. Messages are composed of packets that are sent one after another in pipelined fashion and are routed along the same path with no intervening packets from other messages; only the first packet of a message contains routing information. Each of N PEs issues s types of request messages, r_1, r_2, ..., r_s. Each request type r_i has m_i packets, for $i = 1,2,...,s$. When a request message of type r_i reaches its destination MM, a return message is generated after a certain time period and sent back through the return network to the requesting PE. The return message for a request message of type r_i has type $r_i{'}$ and contains $m_i{'}$ packets, for $i = 1,2,...,s$. The period between the time the first packet of the request message reaches the MM and the time the return message starts the return trip is called the *memory access time*. The memory access time for request type r_i is denoted by a_i. The period between the time a MM accepts a request message typed r_i and the time the MM is ready to accept a new request message is called the *memory cycle time*. The memory cycle time for request type r_i is denoted by c_i.

Each $k \times k$ switch consists of k input ports and k output ports. Each input port numbered i, has a set of input data signals *Di* and a pair of control signals. The size of *Di* is equal to the packet size plus some error checking bits. One control signal is an input signal *DVi*, which indicates that the input data is valid. The other control signal is an output signal *CTSo* (Clear-To-Send output), which tells the sender connected to the input port if the switch is able to receive data. The receiver sets *CTS* true if it is able to accept a message at the next cycle. At that point the sender may start to send packets of a message, without waiting for a response; the receiver has guaranteed that it is able to receive the entire message. Once the first packet of a message of type r_i has started transmission, the remaining $m_i - 1$ packets are sent in next $m_i - 1$ successive network cycles without regard to the *CTS* signal from the receiving input port.

Each output port has a set of output data signals *Do* and a pair of control signals, *DVo* and *CTSi*. *Do* lines are data output signals. *DVo* is an output signal, which is the data valid signal for the *Do* lines. When a packet in a message is sent out through the output port, the data of the packet are placed on the *Do* lines and during the same cycle the *DVo* is set to true to indicate valid data on *Do* lines. Otherwise the *DVo* signal is false. *CTSi* is an input signal from the receiver to the output port. The output port only starts to transmit a message when *CTSi* was true at end of the last cycle. However, once a message transmission is started, the *CTSi* signal is ignored until end of the message transmission. All the packets in a same message are always transmitted in consecutive network cycles.

In the Ultraswitch protocol, a destination address of a message is represented as an *n* digit base-*k* number. The *k* output ports are numbered $0, 1, ..., k-1$. An input message at stage *i*, for $i = 0, 1, ..., n-1$ is routed to the output port numbered by the i^{th} least significant (or most significant depending on the topology) base-*k* digit in the destination address. This digit is termed the *routing digit*. It is assumed that all necessary information for routing the message is contained in the first packet of the message. Note that it is not necessary for the first packet to contain the whole destination address. Suppose we rotate the destination address at each switch so that the routing digit for the next stage is always first. The first packet may then contain as little as a single base-*k* digit. This is the minimum required packet size for the Ultraswitch protocol (ignoring issues of message combining discussed in section 6). Once the first packet is routed to the appropriate output, the remaining packets in the message will follow the same route.

One important feature of the Ultraswitch protocol is that message transmission is *pipelined.* At each switch, a message is ready to be transmitted to the next stage as soon as its first packet arrives. There is no need to wait for the rest of the packets as in store-and-forward networks. This feature is similar to the concepts of *virtual cut-through* [3] and *wormhole routing* [4]

In addition to pipelining, our studies assume that a packet takes only one network cycle to go through a switch in the best case (no conflicts). Therefore, the complete transmission of an *m* packet message across an *n* stage network requires $n + m - 1$ network cycles when no conflicts occur.

3. Unbuffered Networks

First we consider the performance of the simplest switch possible. In such a switch design (see Figure 1) there are two routers on the two input ports of the switch, and two multiplexers on the two output ports of the switch. A *router* has one input and two outputs. The logic function of a router is to route the message at its input to the output specified by the routing digit of the input message. A *multiplexer* has two inputs and one output. If exactly one of its two inputs has a valid message, the multiplexer will select the input with a message and connect it to the output. If the two inputs both have messages, one of them will be selected according to a specified rule. For convenience of analysis, we assume a random selection rule.

Since there are no memory devices at the switch, when the two routers each routes a message to the same multiplexer one of the conflicting messages is lost. Lost messages must be retransmitted by the original sender.

Assume that each PE issues *p* request messages per cycle, each request message is a single packet, request interarrival times are memoryless, and the requested addresses are uniformly distributed. For a $k \times k$ switch at stage *i*, for $i = 1, 2, ..., n$, assume that each of its *k* inputs has a probability p_{i-1} of having a input message. The probability that an output of the switch has a message is

$$p_i = 1 - (1 - p_{i-1}/k)^k, \quad \text{for } i = 1,2,...,n. \tag{1}$$

where $p_0 = p$ is the arrival rate at an input port to the network and p_n is the throughput of the network. The latter can be approximated by

$$p_n = 2k/((k-1)n + 2k/p) \tag{2}$$

(see [18]). Thus for a network with N PEs and N MMs, the throughput at each output port of the network is limited to O($1/logN$), holding k and p constant. The overall bandwidth of the network is then limited to O($N/logN$).

Thus while the unbuffered switch has a simple logic design and low hardware cost, it has a major drawback: its maximum bandwidth[4] (per PE) drops as the network size grows. It has been noted that by introducing buffering into the switches, the network bandwidth can be improved significantly [2]. Figure 2 shows the maximum bandwidth values for both unbuffered and buffered networks. The values for unbuffered networks are calculated according to equation (1) for 2×2 switches ($k = 2$). The corresponding values for buffered networks are taken from simulations of networks for Type A switches with effective queue size of 10 (see section 4 for definitions of Type A switches and effective queue size).

Equation (2) gives the throughput of an unbuffered network with a request rate p. However, in practice, the reverse problem may also be interesting. Namely, suppose we have a system in which each PE requires b (for bandwidth) shared memory accesses per network cycle. What request rate p need we impose on the network? What is the expected number of re-transmissions for each request?

From equation (2), we know that in order to sustain a throughput $p_n = b$, the request rate p must satisfy:

$$p = \frac{b}{1 - nb(k-1)/2k}. \tag{3}$$

Note that the term $1 - nb(k-1)/2k$ above must be greater than zero. This condition is automatically enforced by the condition that the request rate $p \le 1$, since (by (2)) the latter is equivalent to the following condition:

$$b \le \frac{2k}{n(k-1) + 2k}. \tag{4}$$

Equation (4) gives the maximum bandwidth achievable in an n stage unbuffered network.

Since for transmitting b messages per cycle we need to make p requests per cycle (including re-transmissions), each message is expected to be transmitted p/b times. Let r be the average retry number ($r = 1$ for no retries). Then we have,

[4]The maximum bandwidth defined as the network throughput at $p = 1$.

$$r = \frac{1}{1-nb(k-1)/2k}. \tag{5}$$

When a message of type r_i gets through the network without being killed during the conflicts on the forward path (it is easy to see that in unbuffered networks there are no conflicts on the return path) the round trip takes exactly $d_0 = 2n+1+a_i$ network cycles, where a_i is the memory access time for the request type r_i. Hence, if a request does not return after d_0 cycles, it must have been killed due to the conflicts, and must now be re-transmitted. The expected latency of a message, measured from the time a processor makes a request until it is satisfied, depends on the length of the timeout before re-transmission of a lost message. But since each message needs to be transmitted on average r times and each retransmission follows the previous transmission of the same message by at least d_0 cycles, the expected average delay $D \geq rd_0$. With equation (5) we have the expected average delay for a request type r_i as

$$D_i \geq \frac{2n+1+a_i}{1-nb(k-1)/2k}, \quad \text{for } i=1,2,\ldots,s. \tag{6}$$

At the maximum bandwidth from equation (4), the delay is thus

$$D_i \geq \frac{(2n+1+a_i)(n(k-1)+2k)}{2k}. \tag{7}$$

This O($\log^2 N$) growth in delay at the maximum bandwidth may in practice prevent the request rate from ever reaching the maximum bandwidth, and may thus be a more severe constraint than the O($N/logN$) bandwidth limitation.

4. A Taxonomy of Switch Types

Buffered switches are more complicated to design and analyze than unbuffered switches, and may differ in the location of the buffers, at input or output, and in the capabilities of the buffers, such as in the number of messages that can be accepted simultaneously. In previous papers, we have defined three types of buffered switches [24, 25] and compared the performance of single stage switches of each type [28]. In this section we review the definitions of the three switch types and compare their performance as used in a network with the Ultraswitch protocol. Although the basic classification of switch types is for $k \times k$ switches, the illustrations show 2×2 switches for simplicity.

4.1. Type A Switches—Output Queues Have *k* Inputs

For this switch structure, a hardware buffer (or queue) capable of accepting k inputs in one cycle is needed. Figure 3 shows such a structure for $k = 2$. This type of switch has been thoroughly analyzed in the literature, especially in [19] and [29]. In the following, we consider implementation questions.

When discussing buffered switches, we use the term *buffer size* or *queue size* to refer to the physical capacity of each queue at a switch. We define the *minimum queue size* S_0 to be the smallest queue size required for a network to be operational. The *effective queue size* s_e is the actual queue size s_q minus the minimum queue size S_0,

$$s_e = s_q - S_0. \tag{7}$$

In general, assume that there are *s* types of request messages, r_1, r_2, ..., r_s, where each type r_i request has m_i packets, for $i = 1,2,...,s$. Let $m = \max\{m_1, m_2, ..., m_s\}$ be the length of the longest request message. In our (message switched) protocol, once a message starts transmitting, it will continue until the end of the message. Therefore a queue must have at least *m* free slots in order to accept one message, since it does not know the message type in advance. Since the switch does not know in advance to which of the *k* output queues the possible *k* input messages will go, it must prepare for the worst case in which all *k* input messages, all with the maximum length *m*, are directed to the most nearly full output. This worst case will require *km* packets. In addition, there could be one packet from each input port representing an ongoing message transfer, which can result in another *k* packets to a single output queue at the next cycle. Taking all these packets into account, we have the minimum queue size S_0 for the network to be operational:

$$S_0 = k(m+1). \tag{8}$$

At the end of any given cycle, as long as each of the *k* queues has space for at least S_0 packets, all the *CTSo* signals will be set to true. Hence, at the next cycle, all sources connecting to the input ports of the switch may send messages. However, if any one of the *k* queues does not have enough space for S_0 packets, all the *CTSo* signals will be set to false, and all the sources to the switch will be blocked at the next cycle until the condition changes. This is necessary because we do not know to which queue the next input message will go, and must be prepared for the worst case in which there will be *k* maximally-sized input messages all directed to the queue with the least available space (Better protocols are described in section 5).

If the queue space is sufficient for the traffic load, there will be very little blockage. If there is no blocking, the switch can accept one packet of a message at each input port per cycle; and on each output port, one packet of a message is transmitted each cycle if the output queue is not empty. Therefore, we say that the Type A switch has 100% *input rate* and 100% *output rate*.

This configuration, using *k*-input buffers, has the disadvantage of being somewhat more difficult to realize in hardware than the alternatives using one-input buffers which follow.

4.2. Type B Switches—Output Queues Have One Input

This configuration (see Figure 4) is a way to use multiple instances of a simpler type of buffer to approximate the performance of the Type A switch discussed above. A packet leaves an output port whenever any of the *k* associated queues has data; if more than one has data, arbitration must occur. In [20], this configuration is called *buffers within the switches;* in [13], a network containing Type B switches is called a *split-buffered MIN.*

The analysis of each buffer involves a load dependent service time like that for the previous one-input buffer configuration. However, if we look at the *k* buffers associated with an output port as a single queueing system with a service rate of 1 per cycle, then the expected waiting time in the first stage is the same as that of the *k*-input queue. However, the service discipline within the queueing system is no longer first come, first served, so the variance of the waiting time may differ. This in turn may effect the result at later stages. The distributions of the waiting time in a single stage Type B switch with unbounded queues and with queues of size 1 are analyzed in [28]

The minimum queue size for the Type B switch is same as the Type C, as described in the section 4.3, i.e., $S_0 = m+1$. This size is independent of the switch size k. However, there are k^2 queues required in a single $k \times k$ switch module. A message at any input port may be routed to one of k queues, one from each of the k output ports. If one of these k queues does not have enough space, the input port is blocked. But, in contrast to the Type A switches, this does not affect the other input ports.

4.3. Type C Switches—Input Queues Have One Input

In the Type C architecture (Figure 5), each queue, now located at a switch input, is required to accept only a single packet per network cycle. Each input packet is first placed into a queue at the input port. The output end of the queue is connected to a switching mechanism that is similar to that of the unbuffered switch. Besides the simple queue structure, the other advantage of the Type C switch over the Type A is that its minimum queue size S_0 does not grow as the switch size (k) increases. Instead

$$S_0 = m+1, \tag{9}$$

where m the maximum size of all types of messages. Since each queue is only connected to a single message source, as long as the queue has enough space for a single message plus one packet extra for a possible ongoing message, the queue is ready to accept more input and sets the corresponding *CTSo* signal to true. Also in a Type A switch, once a single queue fills the whole switch is blocked, which is not the case for the Type C switch.

In a Type C switch, outputs of the buffers are multiplexed, and a buffer may be blocked on output by another buffer. Though the simplest to construct, this type of buffered switch is relatively difficult to analyze with conventional methods of queueing theory, due to the blocking that occurs even if ‘‘infinite buffers’’ are assumed.

Assume that the inputs generate uniform traffic. Suppose that α is the probability that a buffer is not empty; then, under the assumption that the queue lengths of the different buffers in a switch are independent and the system is memoryless, $b = 1-(1-\alpha/k)^k$ is the average number of items exiting from each output port each cycle (see the discussion of blocking in crossbars in [6] and [33]). Since α cannot be greater than one, the maximum bandwidth per port cannot be greater than $1-(1-1/k)^k$. At $k=2$, this is 0.75; as k gets large, this approaches $1-1/e \approx 0.63$. In [28] it is shown rigorously that 0.75 is the maximum bandwidth (the bandwidth with input rate $p=1$) for 2×2 switches. For switches of size greater than 2, our own simulations, as well as those in [34], indicate that the maximum bandwidth is actually less than $1-(1-1/k)^k$.

In [16] and [32], the authors used a 2×2 Type C switch as the basic unit in their Markov chain analyses of finite buffered banyan networks, without, however, seeming to be aware of the basic bandwidth limitation of 75 percent imposed by the switch structure.

4.4. Performance of the Three Basic Switch types

Figures 6,7 and 8 show the performance of the three basic switch types. The results are from a series of simulations with realistic configurations. The networks are 64×64 with 2×2 switches, and the traffic is uniform, parameterized by offered load. Sixty percent of the requests are for reads and the remainder are for writes. For each read request, the forward message has one address packet and zero data packets, and the return message has one address packet and one data packet. For a write type request, the forward message has one address packet and one data packet, and the return message has one address packet and zero data packets. The memory access time and the memory cycle time of the read requests are all two network cycles. For the write requests, the memory access time is zero, and the memory cycle time is two.

In an implementation that used Type B switches, the NYU Ultracomputer project, in order to simplify the internal logic of the switch, required that all messages have even length and that they be injected into the network only on even cycles [8, 31]. To model this hardware simplification, simulations reported here add a null packet to odd length messages.

Recall that the minimum queue sizes depend on the switch type. In our configuration, the minimum queue size is 6 packets for Type A switches and 3 packets for types 2 and 3. In order to make a fair cost comparison, we report the real queue size instead of the effective queue size. Also since the type 3 switch has 4 queues while there are only 2 queues in a type 1 or Type C switch, in the following figures, the buffer sizes shown for Type B switches are for two queues. The total delay is plotted, which includes queueing delays, stage delays, and memory delays. The *offered load* is the percentage of the time that a request is issued at an input port when the port is not busy. An input port is busy if either it is blocked due to unsufficient buffer space available at the switch or it is transmitting packets from a previous request. The offered load is different from the *request rate*, which is the average number of request messages issued per cycle at an input port. The *throughput* of the network, is the average number of messages transmitted through the network per cycle per input port. For the simulations here, each message is of two packets; the throughput of the network is therefore limited to 0.5 messages/cycle.

From the simulation results shown above, one can conclude that a Type B switch provides about the same performance as a Type A switch, while a Type C switch has only about three quarters of the maximum bandwidth of the other two types. However, due to its low hardware cost, the Type C switch may be a good choice for systems that require only modest communication bandwidth.

5. Multiple CTS Signal Switches

As we mentioned before, in a network with buffers at the output ports, an input port must block and refuse to accept further messages if any buffer it feeds is full. The ‘‘clear-to-send’’ (CTS) signal from the input port to an output port in the preceding stage must be lowered whenever the input port cannot accept a message.

This results in a loss of performance when not all the buffers in the later stage switch are full, since a message destined for a buffer that has space may nevertheless have to wait. One solution is to have multiple CTS signals, one associated with each of the buffers. Then a message will be blocked only if the output port to which it is destined in the next stage is actually full. The effectiveness of this method is further enhanced if the output buffers at the earlier stage have the capability to choose the message to be sent according to its output destination in the next stage; we will call such a buffer a *selective* buffer. A selective buffer can be implemented by sorting messages into separate buffers as they arrive or by using an associative buffer that can select messages for output by address bits. Either alternative requires additional hardware complexity.

In Figure 9, we show the performance enhancement of the 2-CTS signal switch over the ordinary 2×2 Type A switch. The simulation configuration is similar to those in section 4. The queue of each switch on the forward path has a size of 12 packets, and the return path queue has a size of 16 packets. Our simulations show a significant bandwidth increase can be obtained by the 2-CTS architecture, especially when equipped with selective buffers.

6. Combining Switches

When designing a buffered network, it is desirable to do something with the messages while they are waiting, if doing so either increases the overall throughput or performs some other useful function. Combining messages when they meet at a switch is one example of ‘‘doing something with the messages while they are waiting’’ in order to increase functionality. In particular, the Ultracomputer project has proposed combining fetch-and-add operations as well as loads and stores at the switches [15]. The fetch-and-add operation, useful as a synchronization primitive and in many parallel algorithms, is an indivisible add to memory; its format is F&A(X,e), where X is an integer variable and e is an integer expression. The operation is defined to return the (old) value of X and atomically replace X by the sum $X + e$.

Concurrent fetch-and-adds must satisfy the *serialization principle* [5], that is, their effect must be as if executed serially. As a result, fetch-and-add operations simultaneously directed at X cause it to be incremented by the sum of the expressions, and each such operation yields the intermediate value of X corresponding to some order of execution.

When two fetch-and-adds referencing the same shared variable, say F&A(X, e) and F&A(X, f), meet at a switch, the switch forms the sum $e + f$, transmits the combined request F&A(X, $e + f$), and stores the value e in its local memory (see Figure 10). When the value Y is returned to the switch in response to F&A(X, $e + f$), the switch returns Y to satisfy one request, F&A(X, e), and $Y + e$ to satisfy the other, F&A(X, f). Assuming that the combined request was not further combined with yet another request, memory location X is properly incremented, becoming $X + e + f$. If other fetch-and-add operations updating X are encountered, the combined requests are themselves combined. The associativity of addition guarantees that this procedure gives a result consistent with the serialization principle. Other associative operations, and even some non-associative operations, can be combined in a similar manner [17].

Since combined requests can themselves be combined, any number of concurrent memory references to the same location can be satisfied in the time required for one shared memory access from a single PE. This property permits the bottleneck-free implementation of coordination protocols in which many PE’s access the same variables (‘‘hot spots’’).

For buffered networks, even moderate hot-spot traffic can severely degrade all memory access, not just access to shared coordination locations [23, 30]. In [30], combining was shown to be an effective way to cope with the performance degradation caused by hot-spots. However, the effectiveness of combining for large networks with heavy hot spot traffic is affected by the number of requests that can be combined to a single request at a single stage [22]. In [11], recurrence relations were constructed for the queue length probability distribution of several combining switch architectures; these can be used to compute average queue length and output rates, as well as the queue-length-dependent probability that a combine may occur.

The combining feature can be added to any one of the three basic types of switches. Figure 11 shows a 2×2 bidirectional combining switch structure based on the Type A switch model. It contains two 2×2 switches, one for the forward network (from left to right) and one for the return network (from right to left). As shown in the picture, $FQ0$ and $FQ1$ are output queues on the forward path, $RQ0$ and $RQ1$ are output queues on the return path, and WB is a wait buffer, where the data needed to decombine messages that were combined on the forward path waits for the response to the combined request to return from memory. The combining switch operates as follows:

On the forward path, messages in the same queue are compared. If two messages have a common destination address and compatible request types (the definition of the compatibility depending on the individual system design), they can be combined. When the two messages are combined, a ‘‘combined message’’ and a ‘‘wait message’’ are produced. Only the combined message is sent to the next stage; the wait message is put into the wait buffer. When a request returns, it is compared (associatively) with wait messages in the wait buffer. When a match is found the return request gets de-combined into two return messages, corresponding to the two original combined requests.

One important parameter of a combining switch is the *combining multiplicity*, which is the maximum number of messages allowed to combine into a single request at a single switch. A switch with combining multiplicity w is called a w-way combining switch. For a $k \times k$ Type A based combining switch with a combining multiplicity w, one return message may possibly get de-combined into up to w messages, all destined for the same output queue. Were all these messages to be sent to the output queue simultaneously, the minimum queue size for the return switch would be

$$S_0 = k(mw + 1), \tag{10}$$

where m is the maximum size of a message. If instead, the wait buffer uses the same flow control protocol as is used between queues in different stages, the minimum queue size is reduced to

$$S_0 = (k + 1)(m + 1). \tag{11}$$

For switches with one-input queues, such as the Type B and the Type C switches, messages from different input ports are sent to different queues of the switch. Therefore, combining switches based on those two types usually only allow messages from the same input port to be combined at the switch. In Figure 12, we show a one-input two-way combining queue structure developed at the NYU Ultracomputer project. Type B switches using this design have been

Network size	Performance (throughput)						
	NC	CS2	CS3	CD2	C2.5	CD3	U
64	0.086	0.177	0.182	0.203	0.207	0.208	0.201
128	0.051	0.167	0.177	0.198	0.203	0.204	0.201
256	0.028	0.158	0.173	0.195	0.205	0.205	0.200
512	0.015	0.151	0.168	0.192	0.203	0.206	0.198
1024	0.008	0.142	0.165	0.186	0.200	0.201	0.198

Network size	Performance (latency)						
	NC	CS2	CS3	CD2	C2.5	CD3	U
64	72.2	48.0	48.7	50.0	50.8	51.2	59.0
128	111.7	50.5	50.4	51.6	54.1	54.5	63.9
256	193.1	53.0	53.5	54.9	57.8	57.6	68.3
512	339.2	56.4	55.5	56.6	60.5	62.4	71.3
1024	629.4	59.9	58.5	59.3	63.0	63.5	74.9

Table 1. Performance vs. Combining Strategies
(3% Hot-Spot, 100% Offered Load)

fabricated by MOSIS and are in routine use in the NYU Ultra III prototype [7].

In the systolic design shown in Figure 12, a message is taken in at the end of the IN column. The input message moves up until a corresponding empty slot is found at the OUT column. It then shifted to the empty slot in the OUT column and moves down the OUT column on the way out. Comparators are built between the IN column and the OUT column. When a match between a message in the IN column and a message in the OUT column is found, and if the corresponding slot in the CHUTE column is not occupied, the message in the IN column is shifted to the CHUTE column. The message in the CHUTE column moves down together with the corresponding message in the OUT column to the Combine Logic, where they get combined.

The queue structure shown in Figure 13 is similar to that of Figure 12 except that there are two columns of CHUTE structures so that up to three messages can be combined into one.

To overcome the restriction in Type B switches that only messages from the same switch input can be combined, we have proposed an extension to the systolic queue structure. As shown in Figure 14, this structure merges the OUT columns of two Type B queues. A message proceeding along the common OUT column is compared with the messages in the two IN columns simultaneously. This allows two other messages, one from each input port, to combine with the outgoing message. This mechanism is called two-and-a-half-way combining. To extend this design to full three-way combining would require connections from both IN columns to both CHUTEs, and an arbitration mechanism for the CHUTE columns as well as the OUT

Network size	Performance (throughput)						
	NC	CS2	CS3	CD2	C2.5	CD3	U
64	0.060	0.171	0.180	0.204	0.211	0.211	0.201
128	0.034	0.164	0.177	0.198	0.207	0.207	0.201
256	0.018	0.155	0.177	0.196	0.207	0.208	0.200
512	0.010	0.147	0.171	0.194	0.206	0.207	0.198
1024	0.005	0.139	0.166	0.186	0.204	0.206	0.198

Network size	Performance (latency)						
	NC	CS2	CS3	CD2	C2.5	CD3	U
64	95.4	45.2	45.6	46.2	48.5	48.2	59.0
128	157.9	47.9	47.6	48.5	51.2	50.7	63.9
256	283.9	49.9	50.2	50.0	53.8	55.5	68.3
512	531.4	53.6	52.7	52.6	58.9	59.0	71.3
1024	1031.6	57.0	55.4	55.6	59.2	60.3	74.9

Table 2. Performance vs. Combining Strategies
(5% Hot-Spot, 100% Offered Load)

column. Further discussion about implementation of these designs can be found in [9, 12].

To model the different combining strategies discussed above, we simulated networks composed of Type A switches with different combining strategies. Each switch was equipped with 13-packet forward path queues, 22-packet return queues, and a 10-message wait buffer for combining. Requests were generated with 100% offered load. All requests were F&A, i.e., both the forward message and the return message had one address packet and one data packet, and all requests had a two cycle memory access time and a four cycle memory cycle time. When its wait buffer filled, a switch stopped combining until there was wait buffer space available.

We simulated five different combining strategies: (1) two-way combining, restricted to allow only messages from the same input port to combine (CS2); (2) three-way combining, restricted to allow only messages from the same input port to combine (CS3); (3) two-way combining, two messages may combine into one regardless of which input port they entered (CD2); (4) three-way combining, three messages may combine regardless of input port (CD3); and (5)‘‘two-and-a-half-way’’ combining, in which up to three messages may be combined into one, but the last two messages must be from different input ports (C2.5). The simulation results are shown in Table 1 (for 3% hot-spot requests) and Table 2 (for 5% hot-spot requests). We also show for comparison results without combining (NC) and no hot-spot traffic (U).

The results show that although CS2 and CS3 provide significant performance improvement over non-combining networks, they are not as efficient as the other three combining strategies in almost all cases. For small networks (less than 512), CD2 has for hot spot traffic has a throughput close to that for uniform traffic. C2.5 and CD3 have a throughput equal to the throughput of uniform traffic for networks with size up to 1024. The performance of C2.5 is almost indistinguishable from that of CD3.

7. Conclusion

We have studied several ways in which additional hardware added to switches in an interconnection network can improve performance. Although the results are unsurprising in the sense that increased cost is associated with increased benefit, the quantitative results are important, since they can be used, together with detailed implementation information, for optimal system design. For smaller systems or systems with low bandwidth requirements, the less expensive switches can provide adequate performance. For larger systems with high bandwidth requirements, additional switch hardware, though it makes the individual switching components more expensive, may pay for itself by increasing the utilization of other resources, like memories, processing elements and especially wires, so that the aggregate cost for performing computations is less than for systems with less expensive switches. In addition, the relative benefit of buffering and combining messages increases as the system size increases. An important result of this research is that relatively inexpensive combining schemes, combining no more than three messages, give good performance in controlling hot spot traffic for systems with thousands of processors.

References

[1] Chuan-lin Wu and Tse-yun Feng, ''On a Class of Multistage Interconnection Networks'', *IEEE Transactions on Computers* **C-29** (8), pp. 694-702 (August 1980).

[2] Daniel M. Dias and J. Robert Jump, ''Analysis and Simulation of Buffered Delta Networks'', *IEEE Transactions on Computers* **C-30** (4), pp. 273-281 (April 1981).

[3] P. Kermani and Leonard Kleinrock, ''Virtual Cut-through: A New Computer Communication Switching Technique'', *Computer Networks* **3**, pp. 267-286 (1979).

[4] William J. Dally and Charles L. Seitz, ''Deadlock-Free Message Routing in Multiprocessor Interconnection Networks'', *IEEE Transactions on Computers* **C-36** (5), pp. 547-553 (May 1987).

[5] K. P. Eswaran et. al., ''The Notion of Consistency and Predicate Locks in a Database System'', *Communications of the ACM*, pp. 624-633 (November 1976).

[6] D. P. Bhandarkar, ''Analysis of memory interference in multiprocessors'', *IEEE Transactions on Computers* **C-24** (9) (September 1975).

[7] Ronald Bianchini, Susan R. Dickey, Jan Edler, Gabriel Goodman, Allan Gottlieb, Richard Kenner, and Jiarui Wang, ''The Ultra III Prototype'', *Proc. Seventh International Symposium on Parallel Processing Parallel Systems Fair*, pp. 2-9 (April 1993).

[8] Susan R. Dickey and Richard Kenner, ''A Combining Switch for the NYU Ultracomputer'', Ultracomputer Note No. 178, New York University (February 1992).

[9] Susan R. Dickey and Richard Kenner, ''Combining Switches for the NYU Ultracomputer'', *Proc. Fourth Annual Conference on the Frontiers of Massively Parallel Computation*, pp. 521-523 (October 1992).

[10] Susan R. Dickey, Richard Kenner, Marc Snir, and Jon Solworth, ''A VLSI Combining Network for the NYU Ultracomputer'', *Proc. International Conference on Computer Design*, pp. 110-113 (1985).

[11] Susan R. Dickey and Ora E. Percus, ''Performance Differences among Combining Switch Architectures'', *Proc. 21st Annual ICPP* (August 1992).

[12] Susan R. Dickey, ''Systolic Combining Switch Designs'', Ph. D. dissertation in preparation, New York University ().

[13] Jianxun Ding and Laxmi N. Bhuyan, ''Performance Evaluation of Multistage Interconnection Networks with Finite Buffers'', *International Conference on Parallel Processing* (August 1991).

[14] Allan Gottlieb, ‘‘An Overview of the NYU Ultracomputer Project’’, pp. 25-95 in *Experimental Parallel Computing Architectures*, ed. Jack J. Dongarra, North Holland (1987). An updated version is available as Ultracomputer Note 100, Courant Institute, NYU.

[15] Allan Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Larry Rudolph, and Mark Snir, ‘‘The NYU Ultracomputer—Designing a MIMD, Shared-Memory Parallel Machine’’, *IEEE Transactions on Computers* **32** (2), pp. 175-189 (February 1983).

[16] Yih-chyun Jeng, ‘‘Performance Analysis of a Packet Switch based on Single-buffered {B}anyan Network’’, *IEEE Journal on Selected Areas in Communications* **SAC-1** (6), pp. 1014-1021 (December 1983).

[17] Clyde P. Kruskal, Larry Rudolph, and Marc Snir, ‘‘Efficient Synchronization on Multiprocessors with Shared Memory’’, *ACM Transactions on Programming Languages and Systems* **10** (4), pp. 579-601 (October 1988).

[18] Clyde P. Kruskal and Marc Snir, ‘‘The Performance of Multistage Interconnection Networks for Multiprocessors’’, *IEEE Transactions on Computers* **32**, pp. 1091-1098 (1983).

[19] Clyde P. Kruskal, Marc Snir, and Allan Weiss, ‘‘The Distribution of Waiting Times in Clocked Multistage Interconnection Networks’’, *IEEE Transactions on Computers* **37** (11) (November 1986).

[20] Manoj Kumar and J. R. Jump, ‘‘Performance Enhancement in Buffered Delta Networks Using Crossbar Switches and Multiple Links’’, *Journal of Parallel and Distributed Computing* **1**, pp. 81-103 (1984).

[21] Duncan H. Lawrie, ‘‘Access and Alignment of Data in an Array Processor’’, *IEEE Transactions on Computers* **24** (12), pp. 1145-1155 (December 1975).

[22] Gyung Ho Lee, Clyde P. Kruskal, and David J. Kuck, ‘‘The Effectiveness of Combining in Shared Memory Parallel Computers in the Presence of ‘Hot Spots’’’, *International Conference on Parallel Processing*, pp. 35-41 (August 1986).

[23] Yue-Sheng Liu, ‘‘Delta Network Performance and ‘Hot Spot’ Traffic’’, Ultracomputer Note 132, Courant Institute, NYU (January 1988).

[24] Yue-Sheng Liu, ‘‘Architecture and Performance of Processor-Memory Interconnection Networks for MIMD Shared Memory Parallel Processing Systems’’, Ph. D. Dissertation, New York University, Department of Computer Science (September 1990).

[25] Yue-Sheng Liu and Susan R. Dickey, ''Simulation and Analysis of Enhanced Switch Architectures for Interconnection Networks in Massively Parallel Shared Memory Machines'', *Proc. 2nd Symposium on the Frontiers of Massively Parallel Computation* (October 1988).

[26] Janak H. Patel, ''Processor-Memory Interconnections for Multiprocessors'', *Proc. Sixth Annual Symposium on Computer Architecture* , pp. 168-177 (April 1979).

[27] Janak H. Patel, ''Performance of Processor-Memory Interconnections for Multiprocessors'', *IEEE Transactions on Computers* **C-30** (10), pp. 771-780 (October 1981).

[28] Ora E. Percus and Susan R. Dickey, ''Performance Analysis of Clock-Regulated Queues with Output Multiplexing in Three Different 2 by 2 Crossbar Switch Architectures'', *Journal of Parallel and Distributed Computing* **16** (1) (September 1992).

[29] Ora E. Percus and Jerome K. Percus, ''Elementary Properties of Clock-regulated Queues'', *SIAM Journal on Applied Mathematics* **50**, pp. 1166-1175 (1990).

[30] Gregory F. Pfister and V. Alan Norton, '''Hot Spot' Contention and Combining in Multistage Interconnection Networks'', *IEEE Transactions on Computers* **34** (10), pp. 943-948 (October 1985).

[31] Marc Snir and Jon Solworth, ''The Ultraswitch - A VLSI Network Node for Parallel Processing'', Ultracomputer Note 39, New York University (January 1984).

[32] Thomas H. Theimer, Erwin P. Rathgeb, and Manfred N. Huber, ''Performance Analysis of Buffered Banyan Networks'', *IEEE Transations on Communications* **39** (2), pp. 269-277 (February 1991).

[33] David W. L. Yen, Janak H. Patel, and Edward S. Davidson, ''Memory Interference in Synchronous Multiprocessor Systems'', *IEEE Transactions on Computers* **C-31** (11) (November 1982).

[34] Hyunsoo Yoon, Kyungsook Y. Lee, and Ming T. Liu, ''Performance Analysis of Multibuffered Packet-Switching Networks in Multiprocessor Systems'', *IEEE Transactions on Computers* (March 1990).

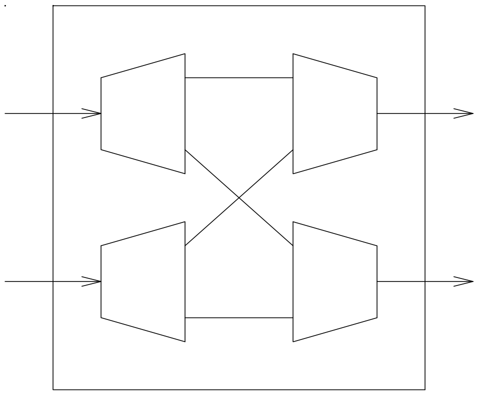

Figure 1 An Unbuffered 2×2 Switch (Type 0)

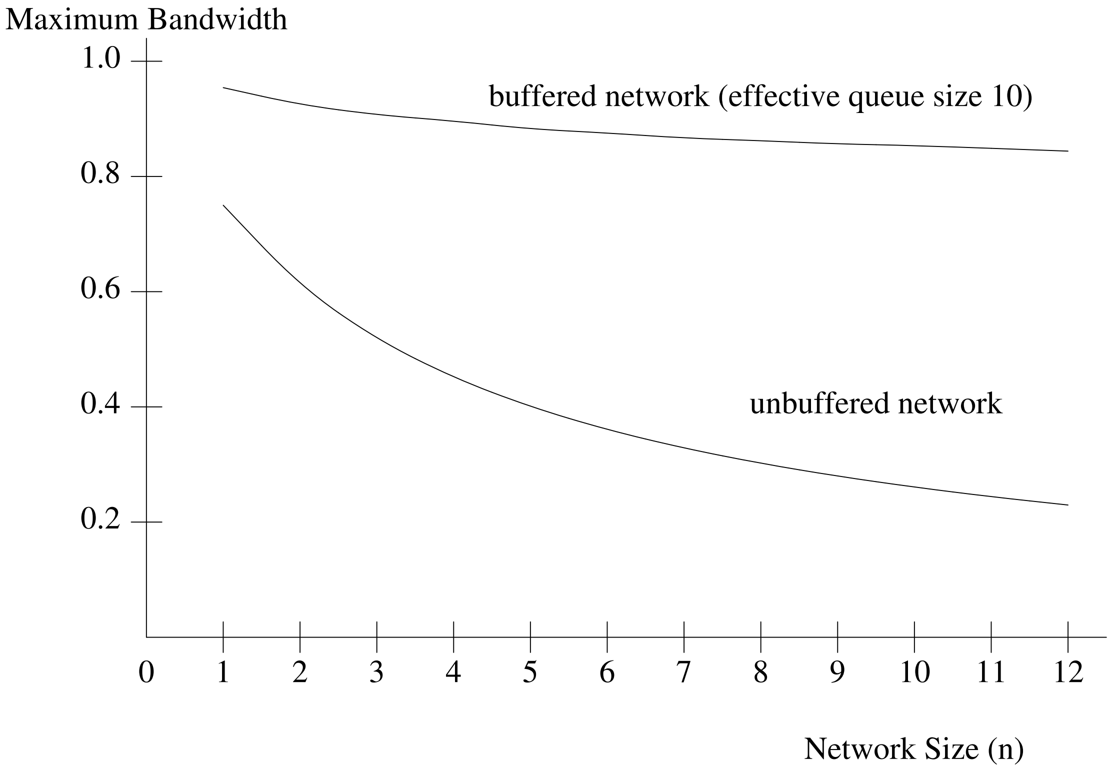

Figure 2. Maximum Bandwidth of Buffered and Unbuffered Networks

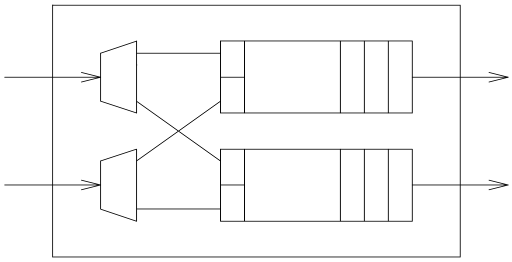

Figure 3. A 2-input Output-Queue Switch (Type A)

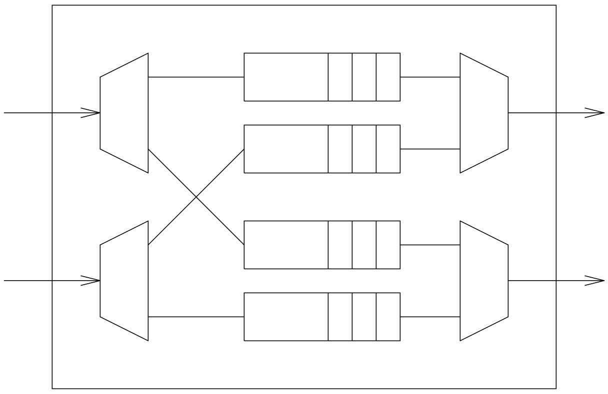

Figure 4. A One-Input Output-Queue Switch (Type B)

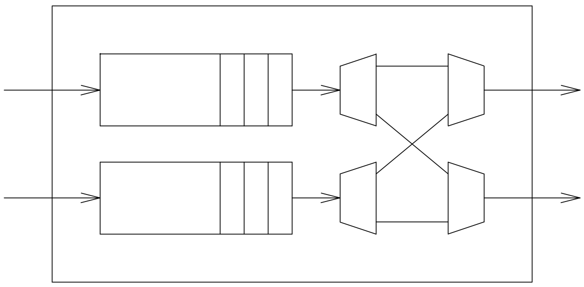

Figure 5. A One-Input Input-Queue Switch (Type C)

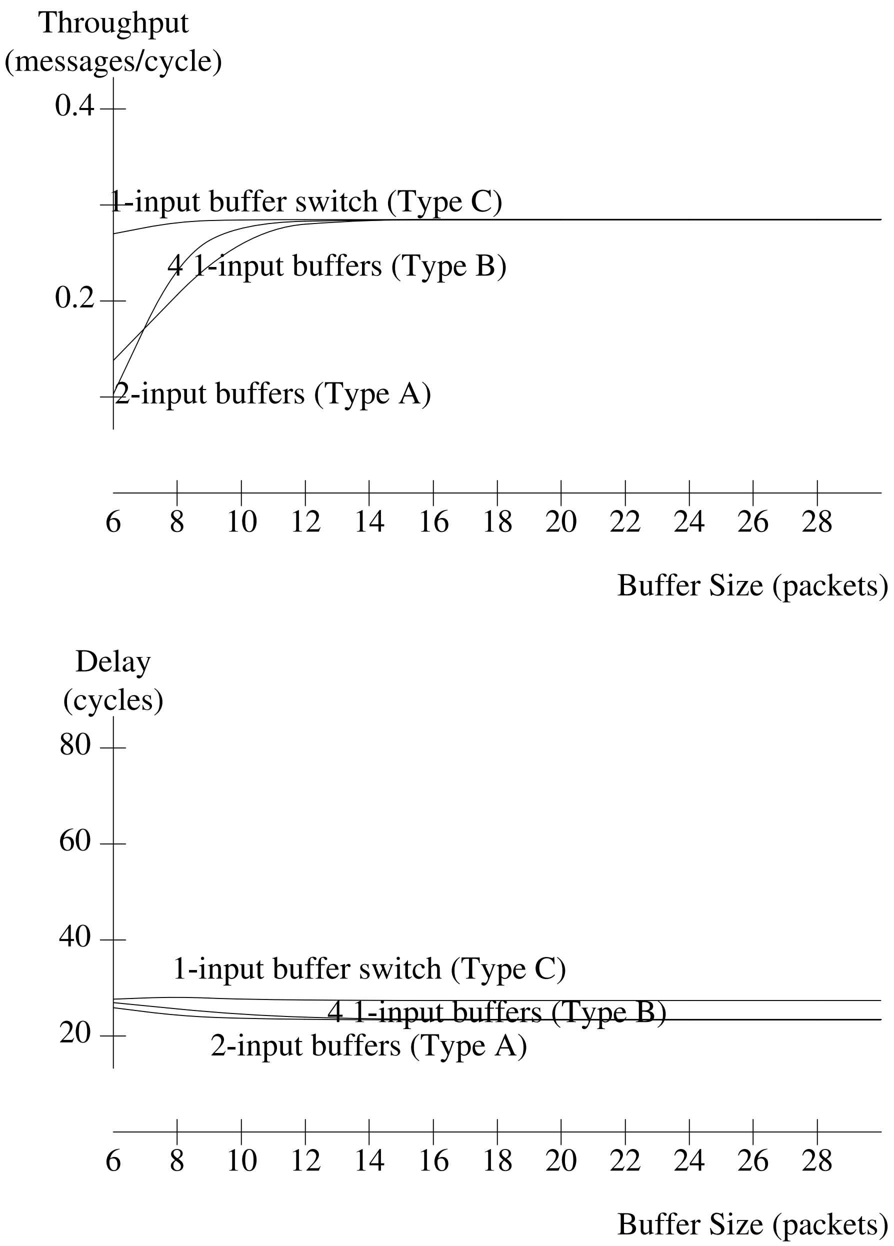

Figure 6. Throughput and Delay vs. Queue Size at 40% Offered Load

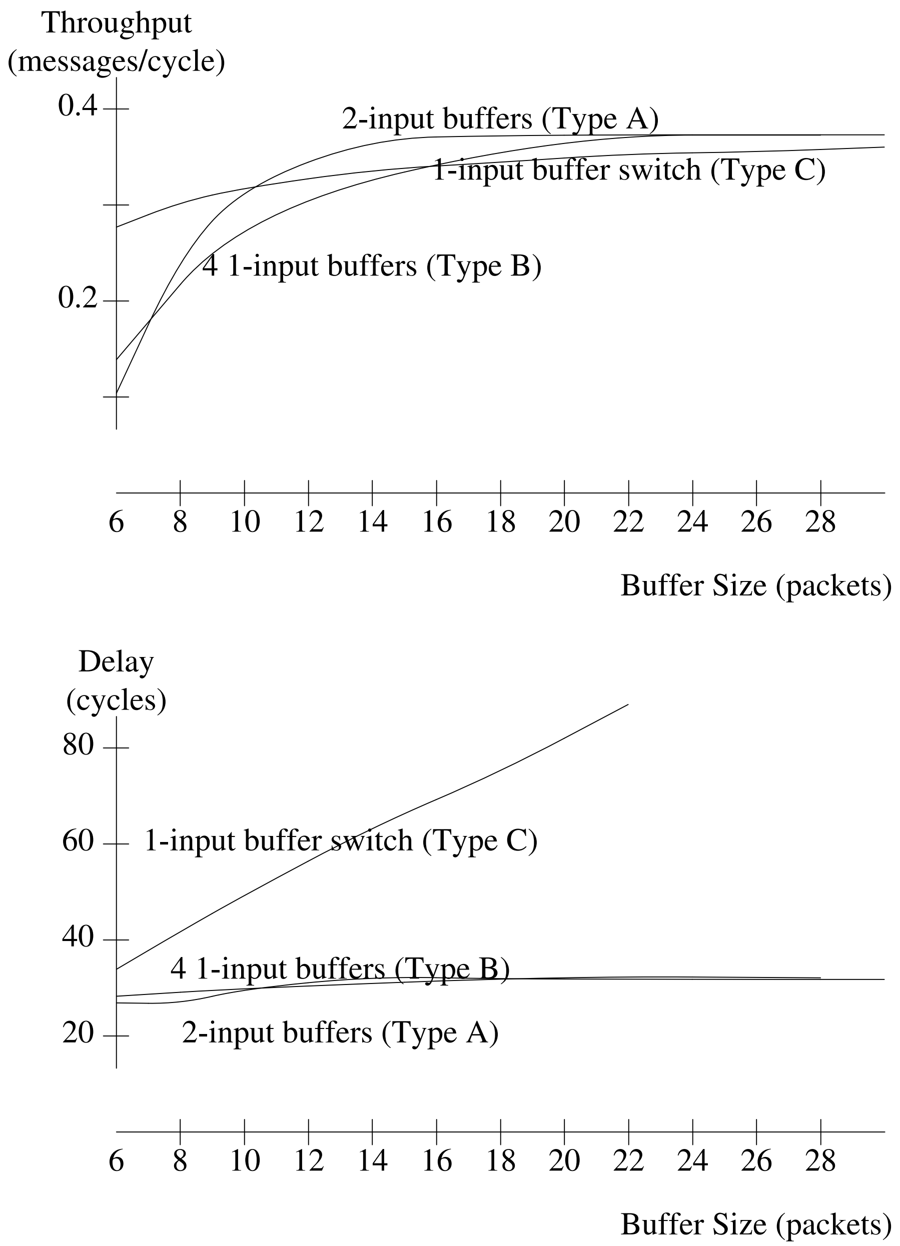

Figure 7. Throughput and Delay vs. Queue Size at 60% Offered Load

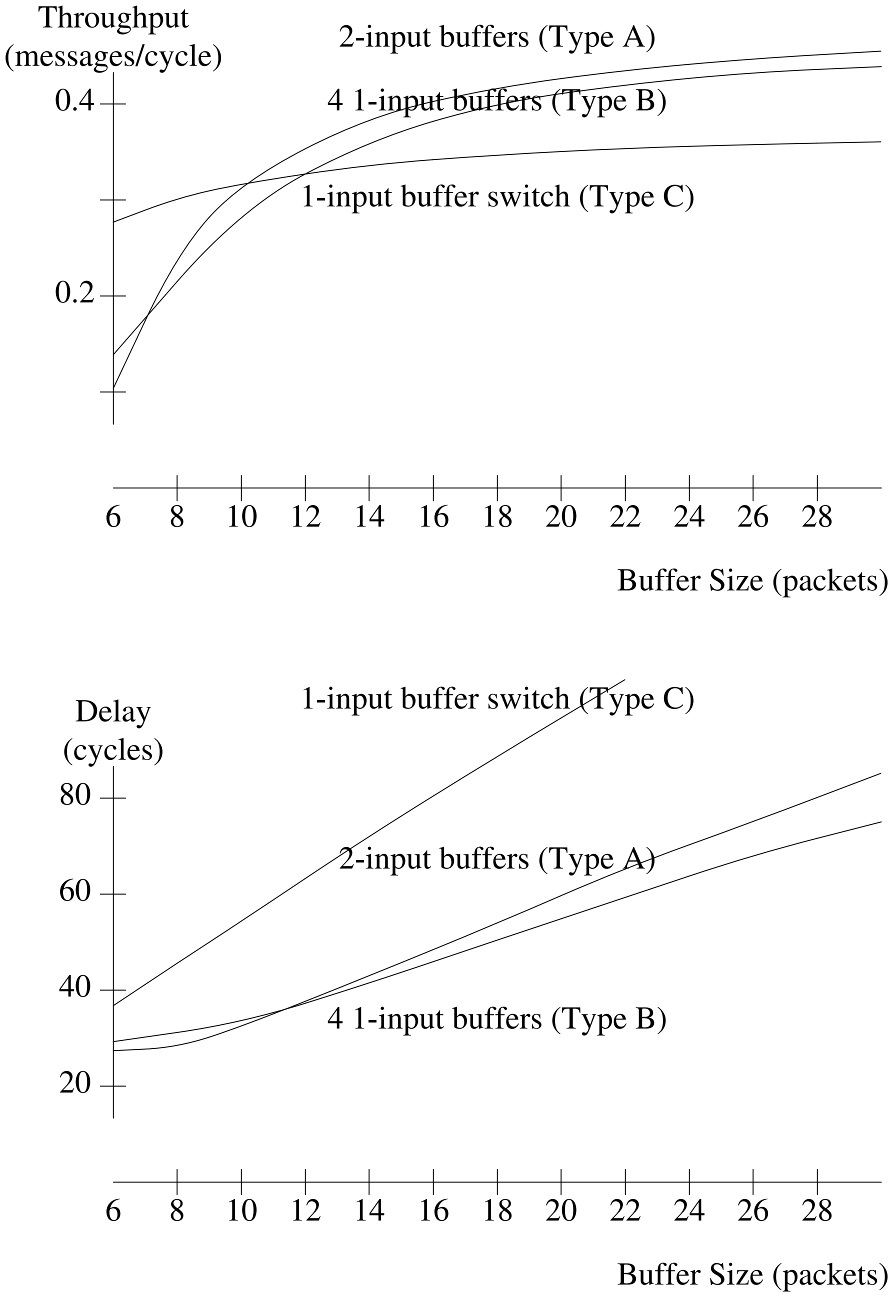

Figure 8. Throughput and Delay vs. Queue Size at 100% Offered Load

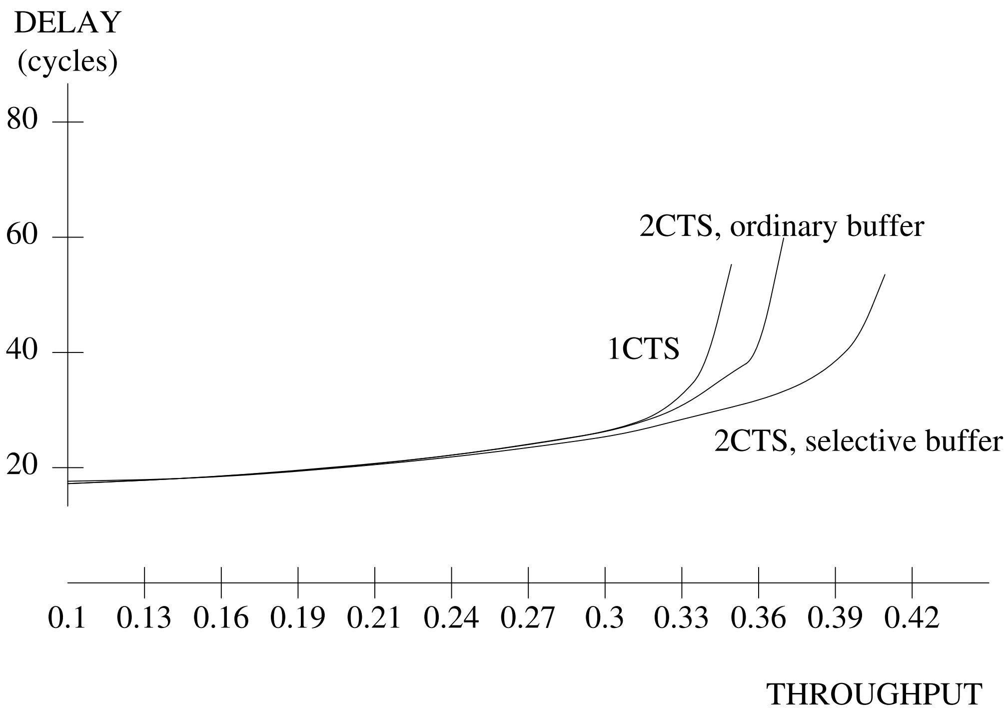

Figure 9. The Performance Enhancement of 2-CTS Signals

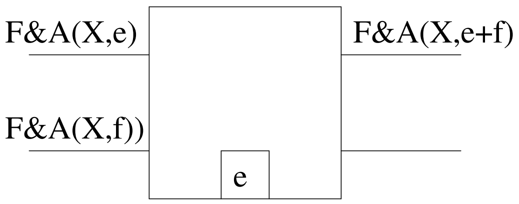

Figure 10. Combining Fetch-And-Add Requests

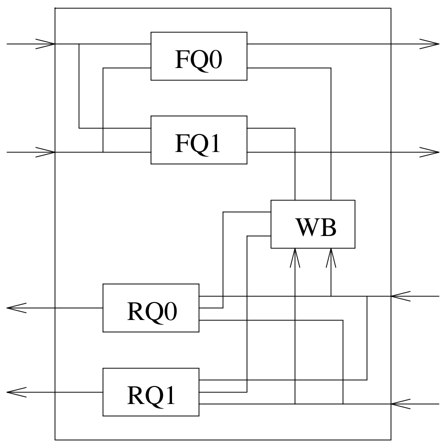

Figure 11. A Combining Switch

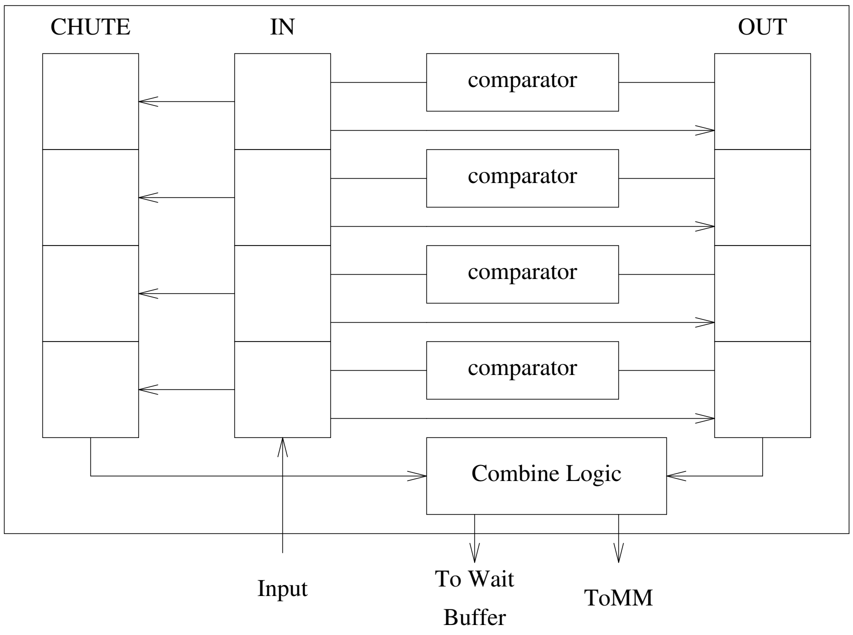

Figure 12. A Type B Two-Way Combining Queue

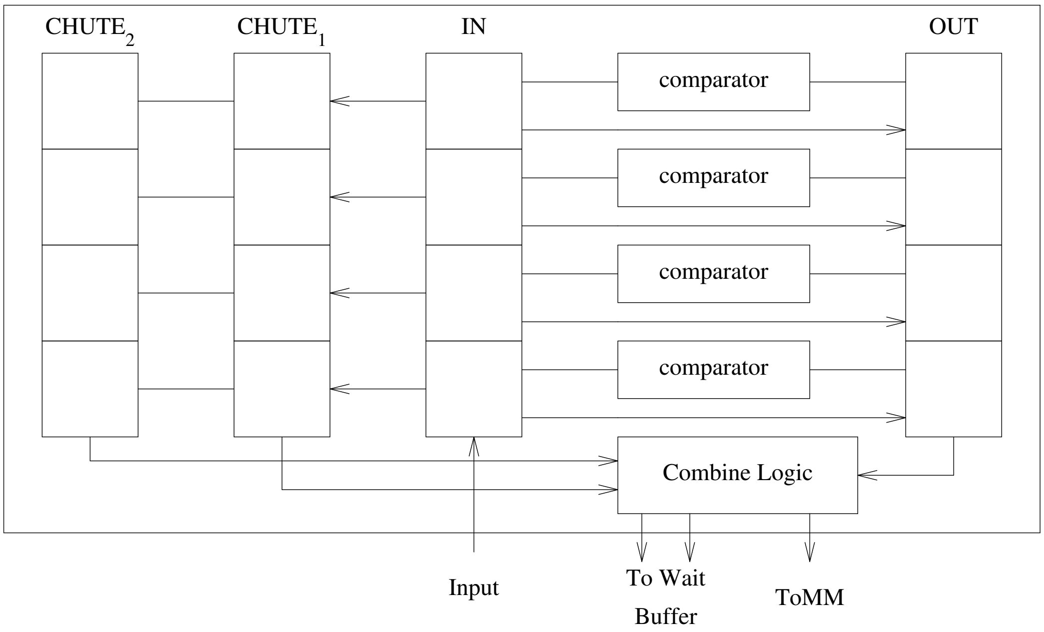

Figure 13. A Type B Three-Way Combining Queue

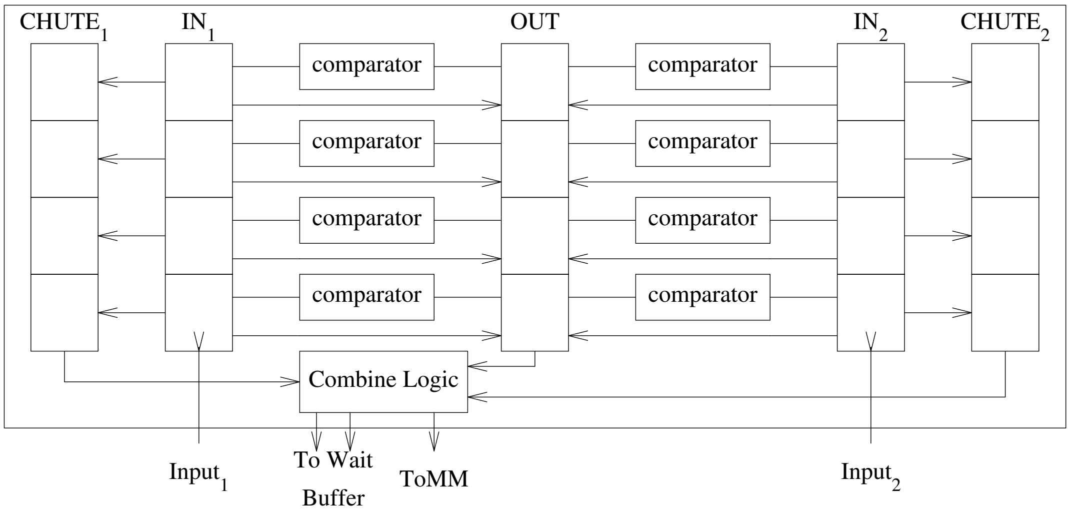

Figure 14. A Two-and-a-Half-Way Combining Queue