Chapter Two
Previous Work

In this chapter, research related to this work is surveyed. Section 2.1 describes several arbitration problems discovered in various applications. In many computer systems, resources (e.g. buses) are shared among users (e.g. processors). Conflicts occur when users compete for these limited resources at the same time. We describe previous research on the arbitration schemes which are used to resolve the conflicts that appear in different applications. In Section 2.2, we give an overview of the related work on buffered communication switch design. The effect of adding buffers in the communication switch on the network performance is discussed. Several experimental switch designs proposed recently by researchers are summarized. Section 2.2 describes other design issues in our ComCoBB project.

2.1. Arbitration Problems in Computer Systems

Several arbitration mechanisms used in different computer applications are presented in this section. Section 2.1.1 describes the most common arbitration problem — conventional bus arbitration. In conventional bus arbitration, more than one processor competes for a single bus, and only one processor can get the bus grant at a time. Section 2.1.2 describes multiple bus arbitration where two or more buses are used to connect processors and memory modules. In this application, more than one processor competes for a pool of resources and these resources are interchangeable. Another application of arbitration is found in the segmented ring bus architecture, which is presented in Section 2.1.3. Similar to multiple bus arbitration, segmented ring bus arbitration has also more than one user and more than one resource. However, unlike the multiple bus arbitration, the resources are not interchangeable in segmented ring bus arbitration.

2.1.1. Conventional Bus Arbitration

In many multiprocessor systems, processors and memory modules are connected by a shared bus. Figure 2.1 shows the configuration of a typical shared-bus multiprocessor. Since there is only one bus, a bus arbiter is used to arbitrate the memory access requests issued by processors. This conventional bus arbiter receives up to n requests for use of the bus and grants one of these requests. Figure 2.2 shows a user-resource model of shared bus arbitration. There are n users (processors) and one resource (shared bus) in the model. The design of such n-user 1-server (or 1-of-n) arbiters has been investigated by many researchers and system developers [Gust84, Taub84].

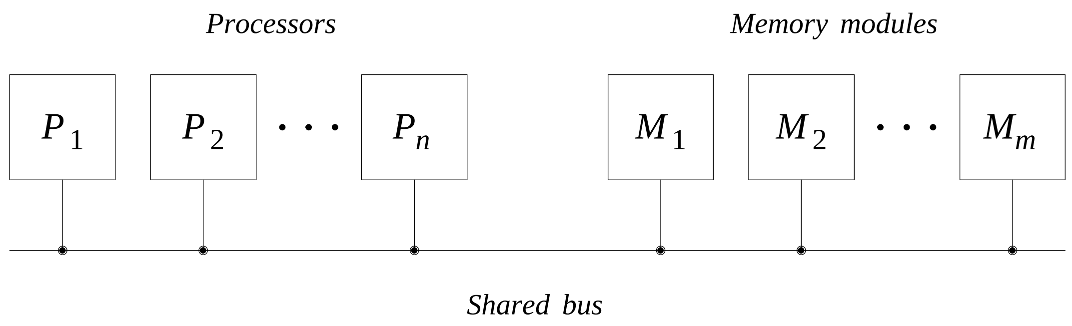


Figure 2.1: A shared-bus multiprocessor.

One of the most important issues regarding arbiter design is the selection policy for the arbitration. In most systems, there is a FIFO queue at each processor

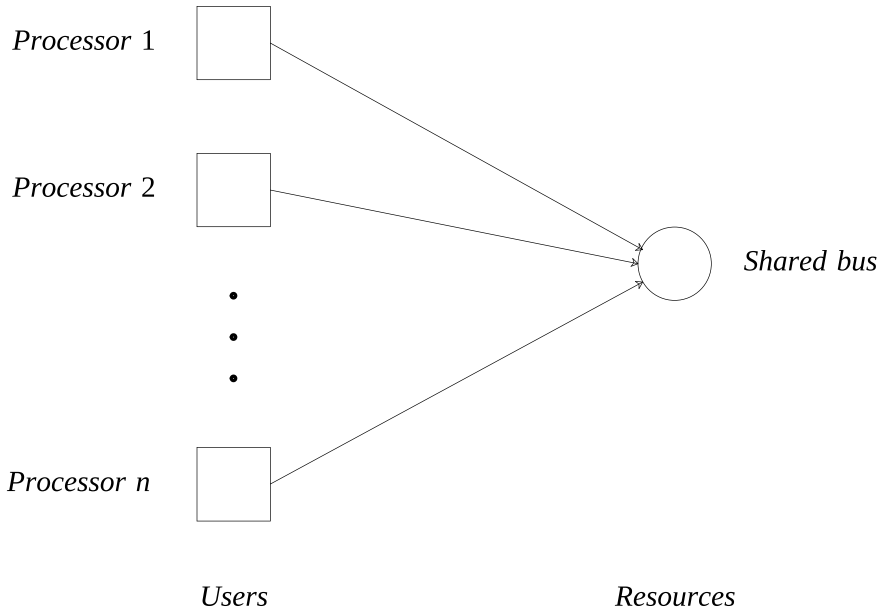


Figure 2.2: A user-resource model of shared bus arbitration.

to buffer the requests to the shared bus. The selection policy of the arbiter decides the priority of the requests at the heads of the FIFO queues. Fixed priority arbiters are relatively simple and fast, but they have the disadvantages that they are not fair; lower priority processors can be forced to wait indefinitely if higher priority processors keep the memory busy. This fairness problem can be solved by giving priority to the processors on a rotating basis, with the lowest priority given to the processor that most recently used the bus [Pear75].

Priority rotation can be implemented in a ring-structured arbiter. However, the arbitration time grows linearly with the number of processors. If we arrange the arbiter as a binary tree of depth $\log_2 N$ constructed from 1-of-2 arbiter modules, the arbitration time grows as $O(\log_2 N)$ instead of $O(N)$. Fairness can be assured by placing a flip-flop in each 1-of-2 arbiter module, toggled automatically to alternate priorities when the arbiter module receives simultaneous requests [Pear75].

The arbiter for shared-bus multiprocessors can be applied to the design of crossbar switches. In an $n \times n$ crossbar switch, n inputs are connected to n outputs through an $n \times n$ crossbar. With FIFO input buffers, there is at most one packet at each buffer that is ready for transmission through the crossbar. Based on the destination of the packet at the head of its FIFO queue, each non-empty input buffer contends for the internal bus connected to one of the output ports. Access to each internal output port bus can thus be arbitrated independently of arbitration of access to other internal buses. Hence, a crossbar arbiter for an $n \times n$ FIFO switch (i.e., a switch with FIFO input port buffers) can be constructed out of n independent conventional bus arbiters. Such crossbar arbitration with n 1-of-n arbiters is used in the Torus Routing Chip [Dall86], where there is a fixed priority order of the inputs to each of the arbiters. Bhuyan [Bhuy87] describes crossbar arbitration where there is an attempt to maintain fairness by rotating the priority order in each 1-of-n arbiter such that the top priority is given to the user following the one who was last serviced.

2.1.2. Multiple Bus Arbitration

In order to increase the bandwidth between processors and memory modules, systems with multiple buses have been proposed [Lang82a, Mudg87]. In these systems n processors are connected to m memory modules through b buses. Figure 2.3 shows the configuration of a multiple-bus multiprocessor. In order to complete, a request from a processor must win the arbitration for two types of resources: a memory module and a bus. Since each processor requests one particular memory module and a memory module can handle only one request at a time, this arbitration can be performed by m independent conventional 1-of-n arbiters [Bhuy87, Lang82b]. All the memory modules for which there are requests must then contend for the available b buses. Figure 2.4 shows a user-resource model of multiple bus arbitration. Since a memory module can use any of the b buses, a different type of arbiter must be used.

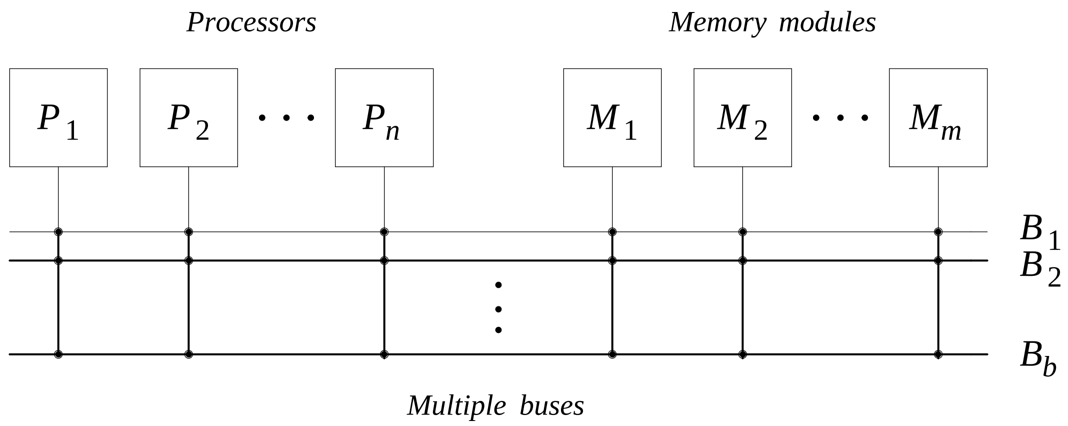


Figure 2.3: A multiple-bus multiprocessor.

Lang and Valero [Lang82b] describe the design and implementation of such an m-user b-server (b-of-m) arbiter. The basic arbiter consists of an iterative ring of m arbiter modules that compute the bus assignments, and a state register to store the arbiter state after each arbitration. The storage of the state is used to make the arbiter fair by taking into account previous bus assignments. After each arbitration, the highest priority is given to the module immediately following the last one serviced.

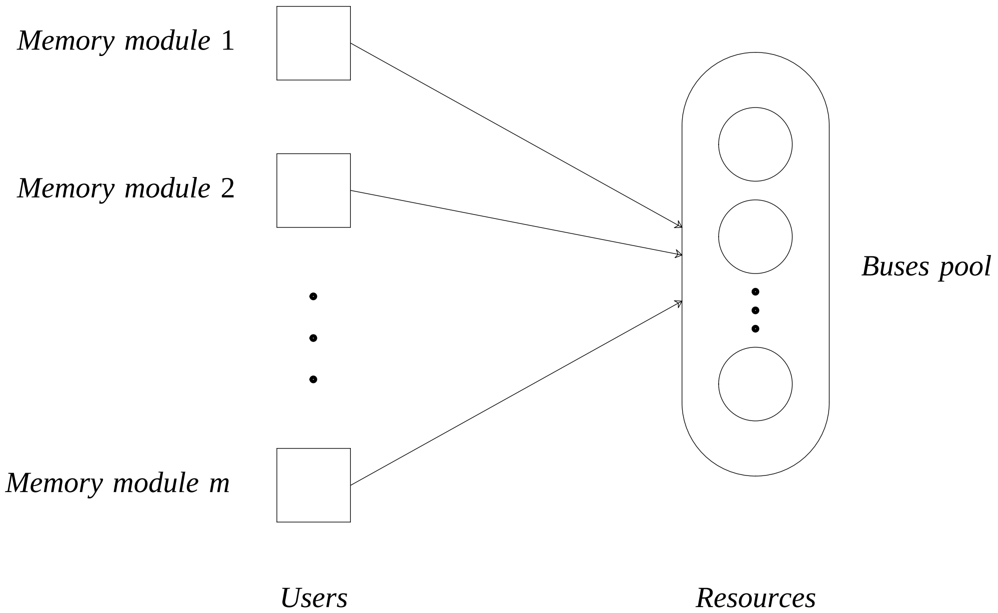


Figure 2.4: A user-resource model of multiple bus arbitration.

2.1.3. Segmented Ring Bus Arbitration

In the Concert multiprocessor system [Hals86], a segmented ring bus is used to interconnect processor clusters. Figure 2.5 shows a multiprocessor based on the segmented ring bus. Each one of the n processor clusters (users) can request up to $n/2$ bus segments (servers) in order to complete an access. Since each bus segment can be used by only one requests at a time, requests from different processor clusters can conflict and arbitration is needed. Each cluster request is translated into a request for a particular set of bus segments. If any one of the requested bus segments is busy, the cluster request cannot be serviced. If a cluster request is denied because one of its required bus segments is busy, any other bus segments already ‘‘given’’ to the cluster become free for use by another cluster. Hence, for maximum utilization of the ring bus, the arbitration for each bus segment cannot be performed independently of the arbitration of other bus segments. Thus, conventional 1-of-n arbiters cannot be used. The required arbitration is also different from the b-of-m arbitration used for multiple-bus systems since the resources (bus segments) are not interchangeable and a request may require more than one resource. Figure 2.6 shows a user-resource model of segmented ring bus arbitration.

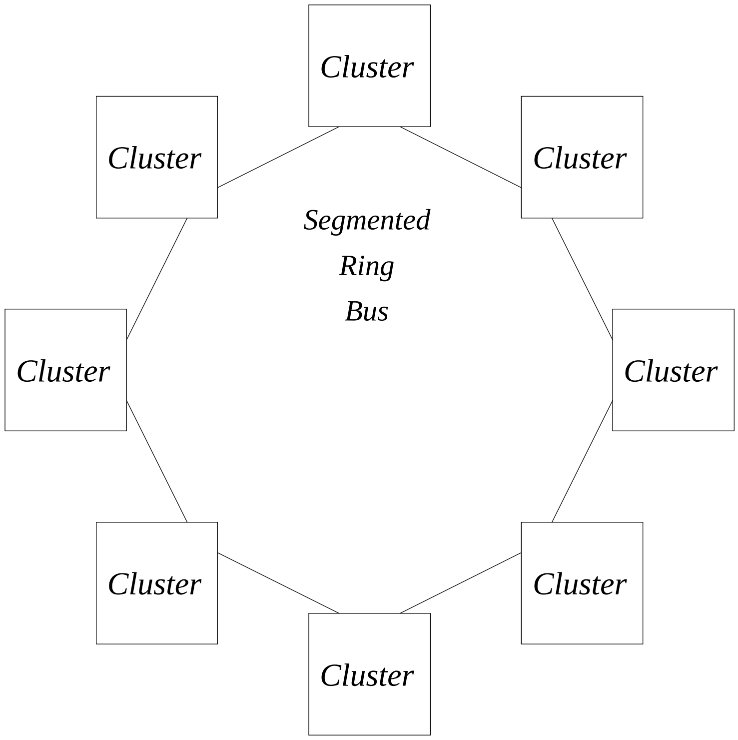


Figure 2.5: A multiprocessor based on the segmented ring bus.

In the Concert multiprocessor, priority rotation is used in the arbiter for the priority selection policy [Hals86]. For each arbitration cycle, the Concert arbiter loops through the requests in order of decreasing priority and make decisions as to whether or not to grant each request. The arbiter grants a request if it does not conflict with either a request already granted or an ungranted higher priority request. Since a request cannot be granted if it conflicts with an ungranted higher

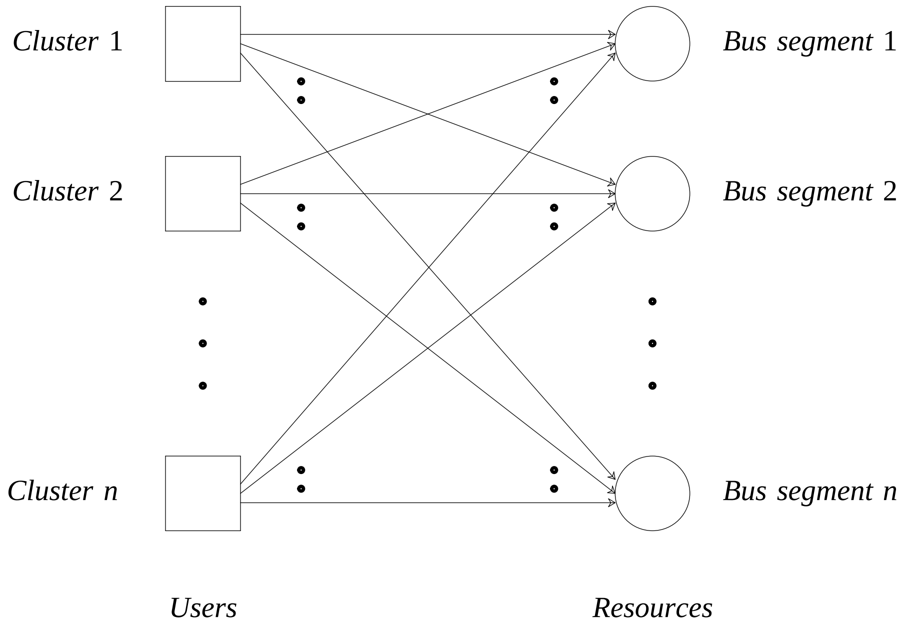


Figure 2.6: A user-resource model of segmented ring bus arbitration.

priority request, it is guaranteed that no request can be locked out forever with the priority rotating scheme. Hence, fairness is ensured.

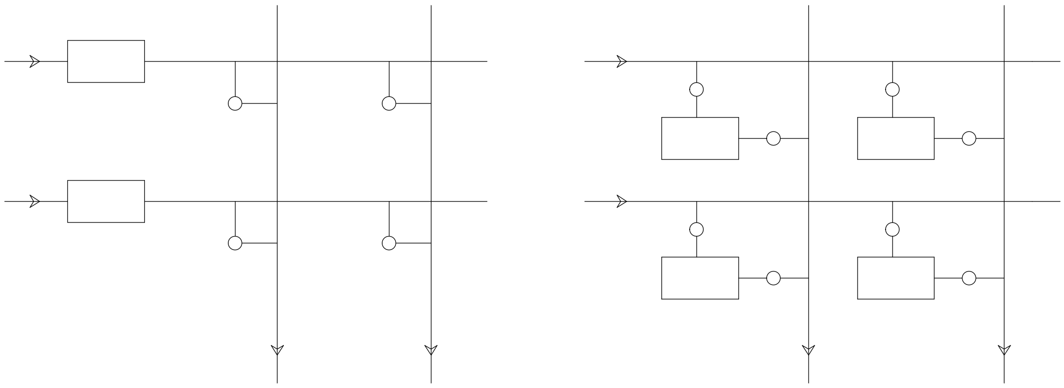
2.2. Buffered Communication Switches

Communication switches are used to construct interconnection networks for supporting interprocessor communication in multiprocessors and multicomputers. Many researchers have studied various design issues of communication switch architectures. It has been shown that adding buffers in communication switches generally can increase the throughput of the interconnection network [Dias81a, Dias81b]. Hence, buffers are often included in many communication switch designs. This section surveys previous research on the design of buffered communication switches. Section 2.2.1 describes the related work on adding buffers in an interconnection network. The performance impact of different buffer organizations is discussed. In Section 2.2.2, we describe several designs of communication switches proposed by other researchers. The important features of each design are introduced.

2.2.1. Performance Impact of Switch Buffers

In an interconnection network, two or more packets arriving in a communication switch may be destined to the same output. When the output conflict occurs, only one packet can be forwarded to the next switch through this output. In an *unbuffered* switch, packets which cannot be forwarded immediately have to be dropped or misrouted. Patel [Pate79] analyzed the performance of unbuffered interconnection networks. Dias and Jump [Dias81a, Dias81b] extended Patel's results and investigated how addition of buffers to the links between stages affects the performance of an interconnection network by analysis and simulations. The performance of buffered interconnection networks is characterized by their throughput (number of packets transferred through the network per unit time) and the latency (average time spent by the packet in the network). The insertion of buffers between the stages of an interconnection network considerably improves its throughput. However, if the buffer size (the capacity of the buffer) is large, the average latency for a packet traversing the network is also significantly increased when the traffic load is high.

Kumar and Jump [Kuma84] investigated the effect of two different buffer designs on the performance of communication switches. One design is that there is one buffer for each input (Figure 2.7-a). Hence, there are n buffers in an $n \times n$ switch with this approach. The other design is that there is one buffer for each crosspoint. (Figure 2.7-b). That is, for each input, the arriving packets destined to the same output are placed in the same buffer. Hence, there are n^2 buffers in an $n \times n$ switch with this approach. Kumar and Jump compared the network performance of the communication switches for these two designs with the same total buffer capacity in the network. It was observed that unless the buffer size is very small, the design with one buffer for each crosspoint achieves superior performance, providing significantly higher throughput. When the buffer size is very small (e.g. only one packet can be accommodated in each buffer), buffer blocking limits the throughput for the design with one buffer for each crosspoint. However, when the buffer size is sufficiently large (e.g. two or more packets can be accommodated in each buffer), this buffer design utilizes the output links more efficiently.



a) One buffer for each input.　　　b) One buffer for each crosspoint.

Figure 2.7: Two different buffer designs.

2.2.2. Communication Switch Designs

There have been a few experimental switch designs proposed by researchers in the past. Gottlieb et al. described a switch design for the multistage interconnection network used in their NYU *Ultracomputer* [Gott83]. The switch is a 2×2 bidirectional routing element, which can transmit a message from the input port where the message arrives to the appropriate output port. An arriving message in the switch is routed based on a single bit in the destination address. A queue is associated with each output port, where messages waiting for transmission to the adjacent stage are stored. The distinguishing feature of the switch is the capability of combining messages. Memory requests are messages that are sent from the processors to memory modules by traversing several stages of switches. When concurrent memory requests are directed at the same memory location and meet at a switch, they can be combined. Combining requests reduces communication traffic and thus decreases the lengths of the queues, leading to lower network latency.

Stevens, Robison, and Davis proposed a switch, called the *Post Office*, designed for their multicomputer system with hexagonal meshes topology [Stev86]. The switch is coupled with a processing element to form a six-ported node. The Post Office uses *virtual cut-through* for the switching mechanism, in which the packet header can be examined as soon as it arrives to determine the output. If the destined output is free, the packet can be transmitted to the next switch immediately without waiting for the whole packet arrives. All ports in a Post Office share the same buffer pool from which packets are retrieved for transmission and stored on arrival. This shared buffer organization minimizes the number of internal buses, wires and controllers, and reduces the total buffer size needed in the switch owing to resource sharing. However, it also results in performance degradation because of the bottleneck of buffer contention. The Post Office supports adaptive routing. The packet buffer controller in the switch determines when packets are stagnating in the buffers. When this occurs, a routing mechanism which dynamically re-routes packets via alternative links is invoked. The buffer controller also contains the deadlock avoidance mechanism.

Dally and Seitz designed a VLSI switch chip, called the *torus routing chip* (TRC), which is used in the k-ary n-cube networks for multicomputers. The TRC provides four bidirectional ports connected to its neighboring routing chips, and one bidirectional port connected to its local processing node. The TRC is entirely self-timed, thus permitting each processing node to operate at its own rate with no need for global synchronization. Similar to the Post Office, the TRC uses virtual cut-through to reduce the latency for packets traversing the network. *Dimension-order routing* is used in the TRC, in which the network routes packets first in the X direction, then in the Y direction. Packets are routed in the direction of decreasing address, decrementing the relative address at each switch. When the relative X or Y address is decremented to zero, the packet has reached the correct X or Y coordinate. The TRC uses *virtual channels* to perform deadlock-free routing in torus networks. By splitting each physical channel into two virtual channels and making routing dependent on the virtual channel on which a packet arrives, the TRC breaks the cycle of channel dependencies. Both virtual channels share a physical communication channel. However, a queue is associated with each virtual channel.

Fujimoto described the architecture of a VLSI communication component proposed for multicomputer networks [Reed87, Fuji83]. In the design, *virtual circuits* is used for the transport mechanism. Each physical link is divided into some fixed number of unidirectional *virtual channels*. The bandwidth is allocated to channels when they have data to send, such that the bandwidth can be efficiently utilized. A translation table is associated with each input port to generate the outgoing link and the channel number. This design employs lookup tables to implement the routing scheme, thus the switch can be used in irregular networks as well as regular networks. To reduce the size of the routing table, hierarchical names are used and there are multiple routing tables per node. There is a single buffer pool shared by all channels of the switch. To avoid performance bottleneck of accessing the buffer by several ports, the buffer consists of 16 memory modules and is interleaved by 16 ways. Strategies that limit buffer usage by individual circuits are used to avoid buffer hogging.

Joerg presented a communication switch design, called the *Packet Switched Routing Chip* (PaRC), which is used as the building block for the indirect n-cube network in the Monsoon multiprocessor dataflow computer [Joer87]. The PaRC is a 4×4 VLSI routing element implemented in a VLSI gate array chip. Instead of a FIFO buffer at each input port, there are four separate buffers there. The capacity of a buffer is for accommodating one packet, and a packet destined to any output port can be stored in any buffer. Through a 16×4 crossbar, each output port can read from any of the 16 buffers at the input ports. The reason for this buffer arrangement is to maximize the throughput of the switch. If only a large FIFO buffer is used at each input port, the blocked front packet in the queue can prohibit the packets behind it from transmitting to free destined output ports. Because of this buffer organization, the PaRC has a scheduling circuit at each output port to guarantee that the order of the packets is preserved if these packets have the same input and the same output. As a result, packets sent from the same source node to the same destination node in the network arrive in order. A facility for sending circuit switch packets has also been included. This allows a processor to ensure that its message has been received before it continues.

Hsu and Banerjee described their routing controller, called *virtual channel router* (VCR), used in the hypercube or other multicomputer networks [Hsu90]. The VCR is a switch designed for circuit switched networks with blocking when contention occurs. Hsu and Banerjee observed that the communications in multicomputer networks exhibit high locality in both message destination and length. The switch can keep the circuit (from source to destination) connected even after the message transmission is completed in order to exploit the locality of communication. This reserved circuit is called the *cached circuit*. When a new message generated at a node has the same destination as the last message, this new message can be sent immediately along the cached circuit, thus saving the circuit set-up overhead. In case of contention, cached circuits can be torn down for other new messages. To improve the connectivity, each physical link in the VCR is divided into two virtual channels.

Konstantinidou and Snyder proposed the *chaos router*, a self-timed adaptive routing switch, for the k-ary n-cube multicomputer networks [Kons90a]. The chaos router is an adaptive router which can *misroute* messages. That is, messages can be sent further from their destination in the presence of congestion. There is a shared queue in the chaos router, which is used to store messages waiting for transmission. The hardware algorithm implemented in the router can randomly select a message in the queue to misroute. A hardware circuit in the router provides the source of randomness for selecting the message in the queue. The chaos router operates fully asynchronously, and the whole router is a self-timed circuit. The hardware algorithm implemented in the chaos router guarantees no message is stalled forever (deadlock freedom). It is also shown that every message injected into the network is eventually delivered (livelock freedom) and every node is able to inject a message eventually (starvation freedom).

2.3. The ComCoBB Project

Small $n \times n$ switches are the key components of the interconnection network used in large-scale multiprocessors and multicomputers. The architecture of these small switches is critical for achieving high-bandwidth low-latency interprocessor communication for multiprocessors and multicomputers. The UCLA ComCoBB project is aimed at investigating the design and implementation issues of VLSI communication coprocessors for multicomputer networks. In this section, a survey of design issues which have been studied by the ComCoBB project is presented. In Section 2.3.1, various buffer organizations are compared based on their performance. Among them a particular buffer design is shown to achieve higher performance than the other practical designs. Section 2.3.2 discusses switch designs supporting high-priority traffic. It is shown that the buffer organization suggested in Section 2.3.1 can be modified to provide superior support for high-priority traffic at a relatively low cost compared to other options. In Section 2.3.3, a message transport mechanism is introduced. This message transport mechanism can be used to efficiently support adaptive routing.

2.3.1. High-Performance Multi-Queue Buffers

It has been shown that adding buffers in communication switches can significantly improve the performance of interconnection networks [Dias81b, Dias81a]. In [Tami92a], alternative buffer designs are studied for small $n \times n$ switches. The goals are to maximize the network throughput and minimize the average latency for the packets traversing the network. Two key design decisions are considered: the location of the buffers and the organization of the buffers.

There are three approaches regarding the location of the buffers in the switch: a central buffer pool, buffers at the input ports, and buffers at the output ports. A central buffer pool provides complete sharing of the available buffer space by all communication ports in a switch, thus can achieve excellent storage utilization. However, there are fundamental difficulties in implementing a high-performance switch with a central buffer pool. Specifically, the buffer will be slow and consume a great deal of chip area, if it needs to support simultaneous reads and writes from several input and output ports. Similar to the problems with the central buffer pool, the option of placing buffers at the output ports is also undesirable, because it also needs to support simultaneous writes from several input ports. These considerations lead to choosing the option of placing the buffers at the input ports, which can achieve high performance with reasonable implementation cost in switch design.

Given that buffers are placed at the input ports, there are two design choices for the buffer organization: FIFO buffers and non-FIFO buffers. In most switch designs, the input port buffers are organized as FIFO queues. However, as discussed in Section 1.1, the performance of FIFO input buffers is limited because of unnecessary blocking. Specifically, when the packet at the head of the queue is blocked, all other packets in the buffer, even those destined to idle output ports, are also blocked. This poor performance of FIFO input buffers can be avoided by allowing handling of packets in non-FIFO order. Three different non-FIFO buffers are studied: statically allocated multi-queue (SAMQ) buffers, statically allocated fully connected (SAFC) buffers, and dynamically allocated multi-queue (DAMQ) buffers. Figure 2.8 shows the organizations of the FIFO buffers and the three non-FIFO buffers.

In an $n \times n$ switch with SAMQ buffers or SAFC buffers, the storage allocated for each input port is statically partitioned into n queues, and each queue stores the packets destined to the same output port. SAMQ buffers and SAFC buffers provide non-FIFO order of handling packets, because now the packets at the head of any one of the n queues can be accessed. Hence, when a packet at a particular input port is blocked, it is still possible to transmit another packet from the same input port to a free different output port. The difference between SAMQ buffers and SAFC buffers is that SAFC buffers allow more than one packet to be transmitted to the outputs for each input port, while SAMQ buffers allow only one for each input port.

A potential performance problem with SAMQ and SAFC buffers is that the storage space in the whole switch is statically partitioned into n^2 queues. When the available storage space is limited in the switch, each queue can be too small to be

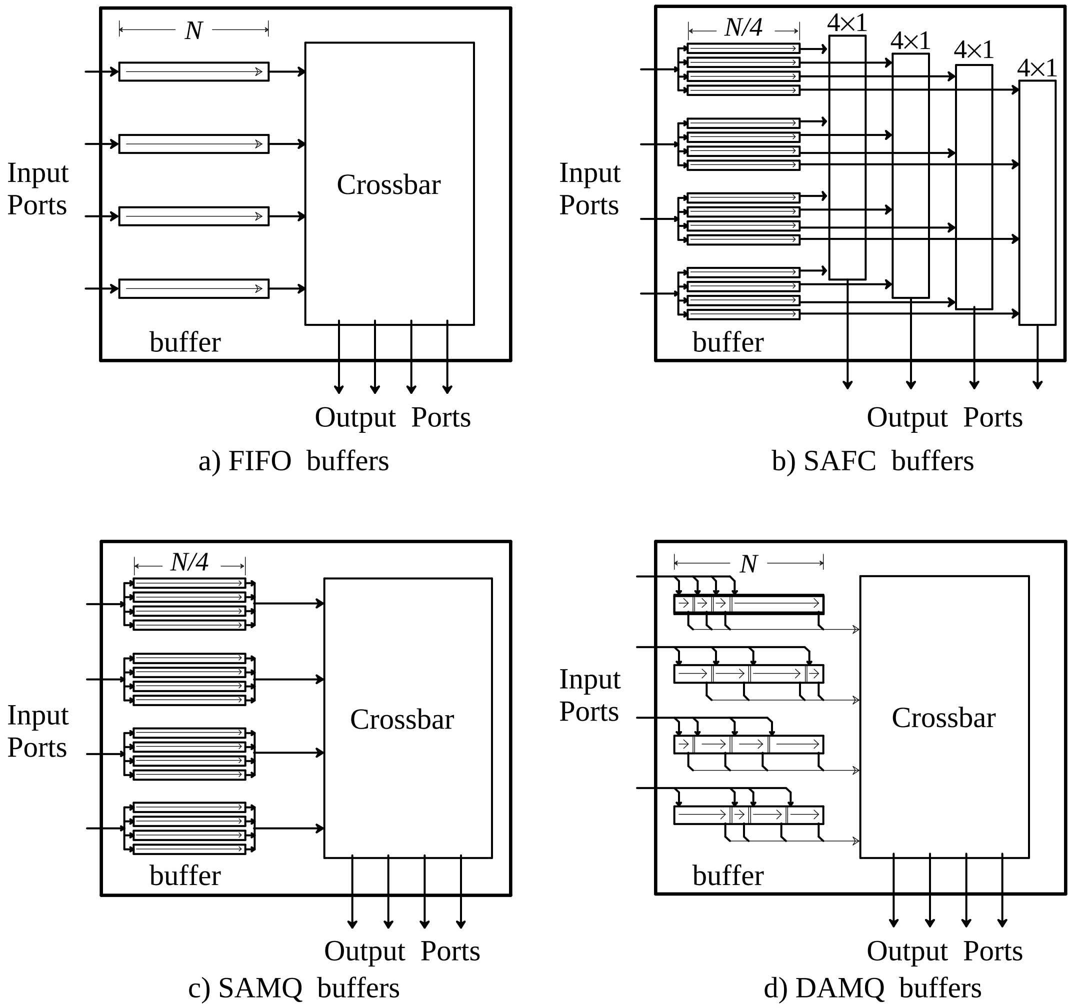


Figure 2.8: Four alternative buffer designs for communication switches.

effective. In [Tami88a, Tami92a, Fraz89], the DAMQ buffer is proposed. In this buffer, the storage at each input port is partitioned into n queues *dynamically*, one for each output port. The DAMQ buffer at an input port can achieve better utilization of the buffer space owing to the effect of resource sharing, while providing non-FIFO order of handling packets. The DAMQ buffer can be managed by using hardware linked lists. The control of the linked lists is provided by a dedicated finite state machine. This additional hardware requires about the same chip area as 32 bytes of storage in the buffer, which is equal to the buffer space to accommodate a packet in the implementation of the ComCoBB switch.

Simulations for a 64×64 Omega network constructed from 4×4 switches have been conducted for the performance comparison of the four different buffer designs. It is shown that the DAMQ buffer achieve superior performance to the other alternatives. For uniform traffic, the DAMQ buffer results in significantly lower latency and higher maximum throughput than the others with the same total buffer storage capacity. For instance, with four packet slots per input port, the networks with FIFO, SAMQ, and SAFC buffers saturate at approximately 0.50. However, the network with DAMQ buffers can achieve a throughput of 0.71. Even the additional chip area is accounted for the more complex control, DAMQ buffers still considerably outperforms the others.

2.3.2. Support for High-Priority Traffic

Large-scale multiprocessors and multicomputers can be used for real-time applications. For these systems, it is critical that high-priority traffic generated by such as I/O devices or system-wide exception handling be transmitted through the network with low latency. To meet this requirement, communication switches used in the interconnection network may have to provide special support for such high-priority traffic. In [Tami88b, Tami92b], communication switches supporting high-priority traffic are discussed and evaluated. The focus is on the design of the input port buffers in the communication switch. Simulations for a 64×64 Omega network are conducted to evaluate the various designs. As a measure of ‘‘worst-case’’ performance, the 99^{th} percentile latency is used. The 99^{th} percentile latency is the minimum of the latencies of the 1% of the packets that received the poorest service (longest latencies) from the network.

From the network simulations, the 99th percentile latency may be more than twice the average latency, and the situation deteriorates when the network load increases. This phenomenon does not change with the different buffer organizations described in Section 2.3.1. Hence, systems for real-time applications have to identify the packets in need of fast service and provide special support for these high-priority packets. Four approaches to supporting high-priority packets in a communication switch are discussed in [Tami88b, Tami92b]:

A. Preferential Arbitration for High-Priority Traffic

The crossbar arbiter is modified in such a way that the requests from the queues which have a high-priority packet at the head are given higher priority. At each cycle, the crossbar arbiter attempts to connect queues with high-priority packets at their heads to the destined output ports first, and then connect queues with normal packets to any remaining available output ports. From the simulation results, it shows that this approach does not provide sufficient support for the high-priority traffic. The 99th percentile latency of the high-priority packets is not reduced to the level of average normal packet latency, even with moderate network load. This conclusion applies to both FIFO and non-FIFO buffers described in Section 2.3.1.

B. Dedicated Queues for High-Priority Traffic

The buffer organization is modified in this approach. At each input port, an additional queue is dedicated to high-priority packets. Since there is a dedicated queue at each input port, it will not happen that the high-priority packets are blocked behind the normal packets as in the preferential arbitration case. Hence, the performance for high-priority packets is greatly improved. The simulation results indicate that as long as the proportion of high-priority traffic is low, dedicated queues for high-priority traffic reduce the 99th percentile latency of the high-priority packets to the level of the average latency for normal traffic under a wide range of network throughputs. The DAMQ buffer is particularly amenable to this approach in terms of implementation since no additional buffer is required. Furthermore, the modified DAMQ buffers achieve better performance than the other modified non-FIFO buffers. Hence, only designs based on DAMQ buffers are considered in the following.

C. Multiple Dedicated Queues for High-Priority Traffic

In this approach, at each input port there is a dedicated queue for the high-priority packets destined to each output port. With a DAMQ buffer at each input port, there are $2n$ queues: n queues for normal packets and n queues for high-priority packets. The simulations results show that when a large proportion of network traffic is high-priority and the network is heavily loaded, this approach performs better than the approach of using dedicated queues for high-priority packets described in the above. However, this situation is less likely, and the performance advantage is small while the associated implementation cost is high.

D. Dedicated Buffers for High-Priority Traffic

In this approach, there are two independent buffers at each input port: one for normal packets and one for high-priority packets. The buffer for normal packets is organized as a DAMQ buffer. The buffer for high-priority packets can be either a

FIFO queue or a DAMQ queue. The simulation results show that dedicated buffers for the high-priority traffic can provide the same performance advantage as dedicated queues in shared DAMQ buffers. However, the implementation cost for dedicated buffers is much higher than for dedicated queues in shared DAMQ buffers. Thus, the approach of using dedicated buffers for high-priority traffic is not cost effective in comparison.

From the above discussion, the approach of using dedicated queue for high-priority traffic is clearly the choice. It provides superior support for high-priority traffic while the implementation cost is relatively low.

2.3.3. Dynamic Virtual Circuits

Adaptive routing can be used to improve network performance and provide fault tolerance in multicomputer systems [Gall77, Bell86, Ngai89, Kim88, Chie86, Chow88, Chen90a, Chen90b, Kons90b, Kons90a, Kons91, Dall93]. With adaptive routing, multicomputers achieves better link utilization and fault tolerance by routing packets around congested or faulty areas in the network. In [Tami91], a *message transport* mechanism which can adapt to congested or faulty network areas is proposed. This message transport mechanism is called *dynamic virtual circuits*.

Traditional virtual circuits scheme is a message transport mechanism which combines the desirable features from both circuit switching and packet switching [Bert87, Reed87]. With conventional virtual circuits, a logical path is set up from the source node to the destination node before the communication starts. Packets are then sent through pre-established logical paths, thus minimizing packet routing time and overhead. On each virtual circuit, packets arrive in the destination node in first-in-first-out (FIFO) order. To improve the utilization of the links in the network, each physical link along the path is logically divided into multiple virtual channels and time-shared among them. Virtual circuits are paths which consist of a sequence of virtual channels.

To establish a virtual circuit, the source node sends a *circuit establishment packet* (CEP) through the network to the destination node. In the intermediate nodes, the CEP arrives on an unused input channel of an input port and then is routed to a chosen output port according to its destination address, using one of the unused channels of the output port. The mapping table is also set to route future packets arriving on the same input channel to the chosen output channel. After the circuit is established, the source node can then transmit packets along the way to the destination node. Once the circuit is no longer needed, a packet called *circuit destruction packet* (CDP) is sent by the source node to delete the circuit from the network. Figure 2.9 shows an established virtual circuit. A mapping table is associated with each input port. The packets arriving in the input port uses the input channel number as the index of the table and extracts the destined output port and the output channel number.

The problem with conventional virtual circuits is that they cannot be changed in response to congestion or failure since the paths are static. Dynamic virtual circuits proposed in [Tami91] can overcome this problem by allowing existing circuits to be torn down on demand. With dynamic virtual circuits, the establishment of a new virtual circuit is guaranteed to succeed even when there are no free virtual channels on a desired link. When a CEP arrives in a node where all

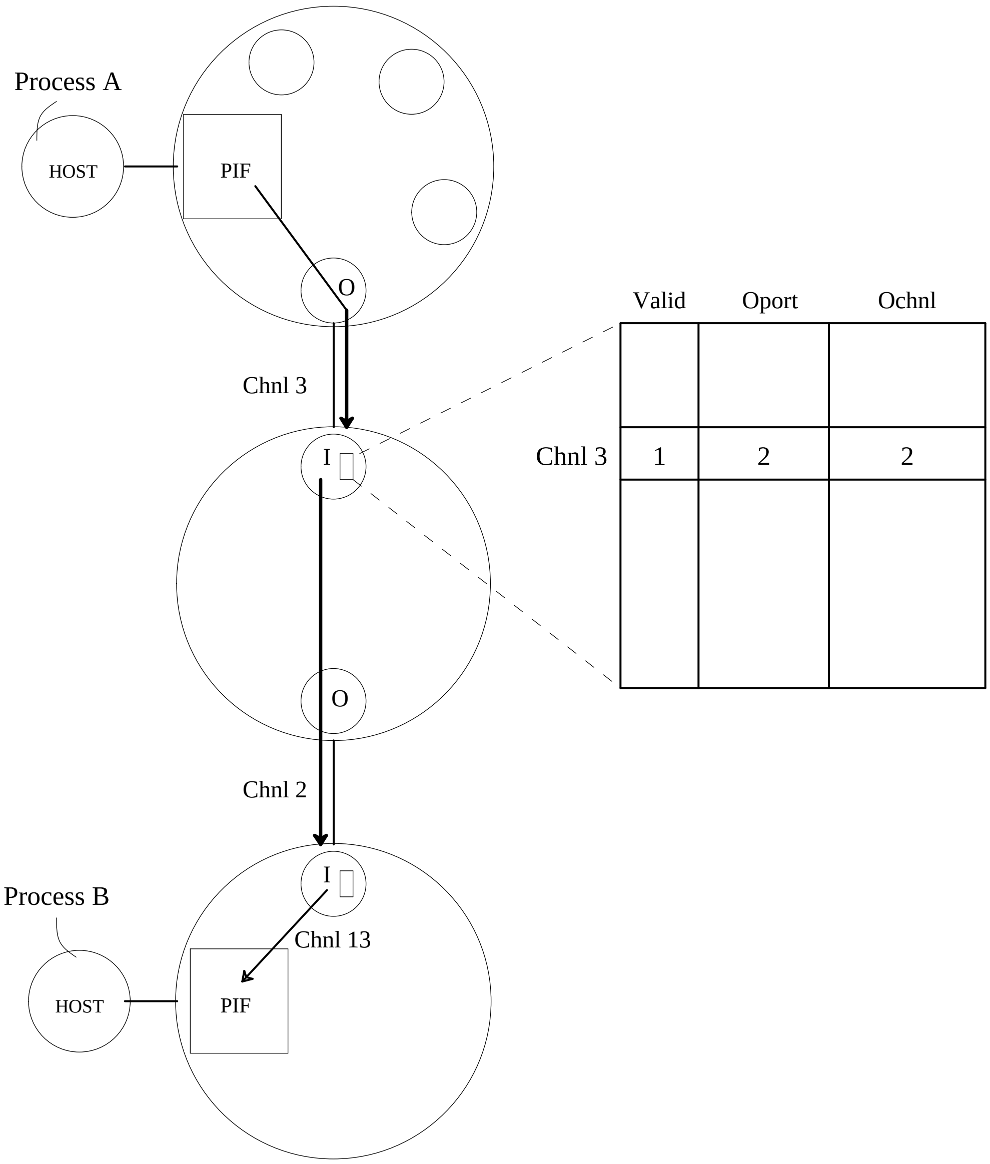


Figure 2.9: A virtual circuit from process A to process B. The circuit uses channel 3 of the first link and channel 2 of the second. PIF is the processor interface.

the channels of the needed link have been allocated, a victim channel is selected and the circuit is disestablished from the intermediate node. The decision is made locally at this intermediate node. The part of the circuit from the source node to the destination node remains intact so that the circuit can be reestablished later.

Tearing down a circuit temporarily is achieved by sending a CDP along the path from the intermediate node. After the victim channel is chosen and deallocated, the node generates a CDP and sends it along the victim channel. Then the CEP can be sent through the output port, continuing to establish the new circuit. The generated CDP is marked *terminal* so that the destination node knows that the circuit is torn down temporarily from an intermediate node, and it will be reestablished later.

To reestablish the cut circuit, the information regarding the ultimate destination node has to be provided. This information is kept at each intermediate node when the circuit is established originally. The reestablishment of the cut circuit is triggered by its next arriving packet at the node where the circit is previously cut. The node chooses an output port, creates a CEP, updates the mapping tables, and sends the CEP. The arriving packet and the following ones on the same circuit can then be sent along the new path.

Since a circuit can be torn down temporarily and reestablished later, physical FIFO packet arrival at the destination node is not guaranteed. It is possible that the reestablishment CEP follows a different path and arrives at the destination node before the CDP generated at the intermediate node arrives. As a result, the packets on the reestablished branch can arrive before the packets on the torn-down branch. To ensure proper packet order, a mechanism based on logical timestamps can be employed. The mechanism achieves proper packet order by providing a kind of logical timestamps in CEPs and CDPs so that the destination node can identify and order the arriving branches from the same circuit. In each node, a count of the number of nonterminal circuit destructions the node has initiated is maintained. This count should be large enough and can thus serve as a logical timestamp on the node. A circuit destruction event can be uniquely identified by combining the identifier of the node and the timestamp. When the node sends out the reestablishment CEP, the node identifier and the stored destruction timestamp are included in this packet. At the destination node, the reestablishment CEP is matched with the corresponding CDP by checking the values of the node identifier and the timestamp.

A multicomputer node supporting dynamic virtual circuits consists of an application processor, local memory associated with the application processor, a crossbar switch, a special routing processor with its memory. Figure 2.10 shows the organization of such a multicomputer node. The routing processor is a general-purpose processor which is used as a dedicated controller to perform some of the infrequent but complex operations that are needed to support dynamic virtual circuits. These infrequent operations include initiating circuit destruction, reestablishing a circuit, updating of global routing tables, and resolution of deadlocks. Those frequent operations such as routing and forwarding of a packet on an established circuit are handled by the crossbar switch.

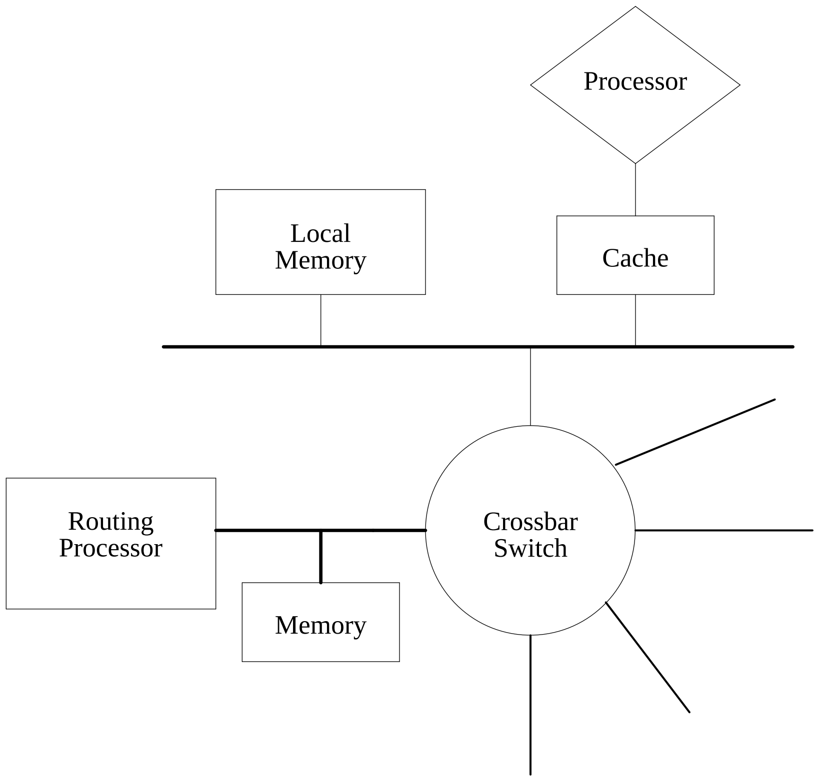


Figure 2.10: A multicomputer node.