Chapter One
Introduction

A computer program can be viewed as a set of instructions or operations which must be executed in order to complete a task. A primary goal of computer architecture is to design computers which will execute programs in the shortest time possible. Two possibilities for increasing the speed of a computer are to increase the speed with which each instruction is executed or to increase the number of instructions which can be executed simultaneously. Recent trends in VLSI technology have shifted the focus of high performance computing from the former possibility to the latter. A large scale distributed computer, consisting of hundreds or thousands of independent processing nodes, has the potential to bring many times the processing power of a supercomputer to bear upon the execution of a program for a fraction of the supercomputer's cost.

A distributed program is a program implemented as set of processes, where each process can be executed in parallel on the separate nodes of a distributed computer. For these processes to cooperate in the solution of a problem they must communicate with each other, either by transmitting data from one processor to another via messages or by accessing shared memory. Distributed systems which use the former communication model are often referred to as *multicomputers*, while shared memory machines are *multiprocessors*. The more processes a task can be divided into, the more steps toward the task's completion that can potentially be executed simultaneously, and, in theory, the faster the task will complete. However, as the process granularity becomes very fine, the latency of the process-

to-process or process-to-memory (depending upon the architecture) communication begins to contribute significantly to the task's execution time. In addition, the available bandwidth of the communication network may be less than the desired throughput of communication. These two issues — high communication latencies and finite network bandwidth — limit the degree to which a task can be profitably distributed. Implementing high bandwidth, low latency communication networks is thus critical to the ability of multiprocessors and multicomputers to achieve high performance by exploiting parallelism.

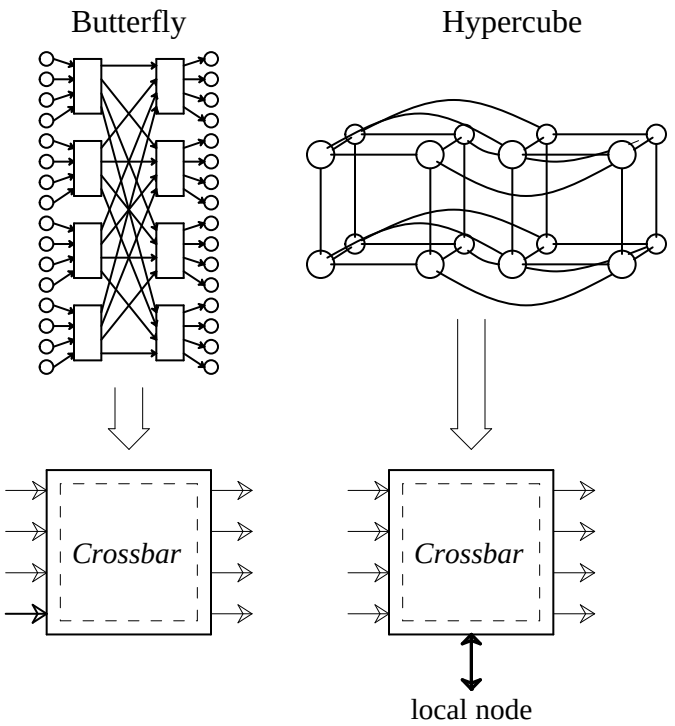

Figure 1.1: *Switches in Multiprocessors and Multicomputers.* Small $n \times n$ switches are the central component of the communication networks of both multiprocessors and multicomputers.

Multiprocessors with a large number of nodes (e.g. greater than 64) use multistage interconnection networks to connect processors to memory [Crow85,

Gott83, Rett90, Pfis85b]. These networks are typically composed of a large number of small $n \times n$ switches (typically, $2 \le n \le 10$) (Fig. 1.1). Memory references are formed into packets to be transmitted across these networks. Multicomputers, on the other hand, use point-to-point links dedicated to nearest neighbor communication [Seit85, Whit85, Dall87b]. If a process wishes to send a message to a node which is not an immediate neighbor, the packet(s) which constitute the message must pass through intermediate nodes to reach their destination. To accelerate its performance, each computing node relies on a communication coprocessor with a small number of ports [Dall86, Stev86] that function as a small $n \times n$ switches with $n-1$ ports connected to other nodes, and one bidirectional port connected to the local application processor (Fig. 1.1). The design of high performance small $n \times n$ switches is thus of critical importance to the development of multiprocessor and multicomputer systems. Since many of these $n \times n$ switches are needed in a large system, there is strong motivation to implement each switch as a single VLSI chip.

1.1. Latency and Output Port Contention

A switch's job is to take packets arriving at its input ports and route them to its output ports. As long as only one packet at a time arrives for a given output port, there will be no conflict, and the packets are routed with a minimum latency. Unfortunately, as the throughput goes up, so does the probability of conflict. When two packets destined for the same output port arrive at different input ports of a switch at approximately the same time, they cannot both be forwarded immediately. Only one packet can be transmitted from an output port at a time,

and hence one of the two packets must be stored at the node for later transmission. The maximum throughput at which the switch can operate depends directly on how efficiently the switch can store the conflicting packets and forward them when the appropriate output port is no longer busy.

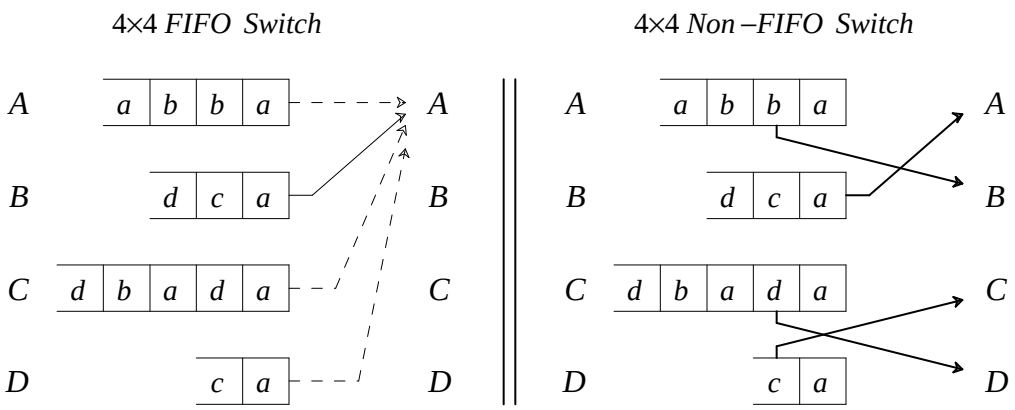

Figure 1.2: *Output port contention.* In the FIFO switch, packets destined for output port A block packets destined for the other output ports. In the other switch, non-FIFO buffer access improves buffer and output port utilization.

Communication switches which use first in, first out (FIFO) buffers located at the input ports to store packets [Rimo87] do so because it is a simple scheme to implement. Space allocation in a FIFO buffer is simple, even for variable size packets, and only two registers are required to maintain buffer organization (pointers to the front and rear of the queue, with a ‘‘circular’’ memory array). This simplicity, however, has a price — FIFO buffers are very inefficient. Their inefficiency stems from the fact that, in a FIFO buffer, only the ‘‘oldest’’ packet in the buffer is available for transmission at any given point in time. This makes the switch susceptible to *output port contention.* Output port contention is the situation in which two or more buffers have packets destined for the same output port

(Figure 1.2). Since each output port can only transmit one packet at a time, one or more of the buffers cannot transmit its packet, and because in a FIFO buffer only the first packet in the queue is available for transmission, those buffers must remain idle while the other transmits its packet. In addition, the buffers which are idle may have within them packets destined for other output ports, which may also be idle. In this situation, the FIFO property of the buffers is limiting the throughput of the switch; available bandwidth could potentially be better utilized by using non-FIFO buffers.

The significance of this is magnified when dealing with real-time systems. Multiprocessor and multicomputer systems that are connected to input / output devices which interact with the outside world occasionally require particularly fast communication with different parts of the system. The broadcasting of certain types of system-wide events, such as the initiation of global system rollback, may also require preferential handling by the network. Real-time requirements due to interaction with I / O devices and system-wide exception handling thus lead to the need to support special purpose *high-priority* traffic whose maximum latency is significantly lower than the maximum latency for general traffic. In a switch composed of FIFO buffers it is likely that, upon entering a switch, a high-priority packet will be trapped behind a ‘‘normal’’ packet, and will thus be delayed. A switch composed of non-FIFO buffers, on the other hand, would be able to transmit the high-priority packet as soon as its destination output port is free, allowing the high-priority packet to bypass the normal packets.

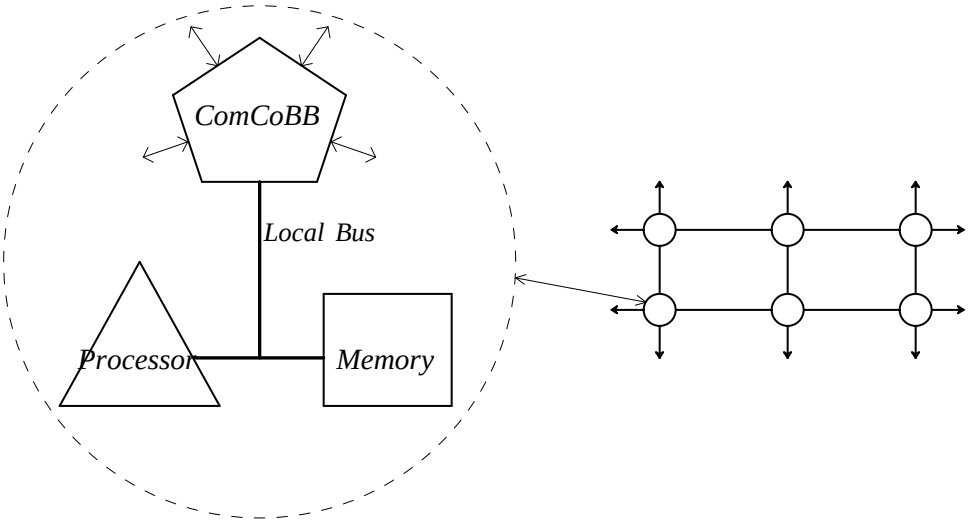

Figure 1.3: *The ComCoBB Chip.* The ComCoBB Chip is a communication coprocessor whose intent is to ‘‘hide’’ the communication network from the processing nodes of a multicomputer.

1.2. The Dynamically Allocated, Multi-Queue Buffer

The results reported in this dissertation were produced as part of the UCLA ComCoBB project. The goal of the ComCoBB (**Com**munication **Co**processor **B**uilding-**B**lock) project is to design and implement a single-chip high-performance communication coprocessor for use in VLSI multicomputer systems. In a multicomputer node consisting of a CPU, local memory and various co-processors, the ComCoBB chip handles forwarding all packets which do not originate from or are not destined to the node, as well as handling the network interface for all packets / messages which are to or from the node (Figure 1.3). Thus, the ComCoBB chip is, in part, a small $n \times n$ switch. It quickly became evident that one of the keys to designing a high performance packet switch was to design an efficient buffering scheme for the switch. The focus of this dissertation is the

design and implementation of the *dynamically allocated, multi-queue* (DAMQ) buffer, the internal buffer of a small $n \times n$ VLSI communication switch.

The key to the high performance of the DAMQ buffer is its non-FIFO behavior. The DAMQ buffer maintains an internal organization of packets via linked lists. The lists correspond to the destination ports of the packets and the priority with which the packets are forwarded. This allows the buffer to be accessed ‘‘randomly’’, based upon the destination and priority of the packets within it. The use of linked lists, however, maintains the FIFO ordering of the packets of the same priority between each input port / output port pair, thus supporting routing schemes such as virtual circuits which depend upon packet ordering.

The DAMQ buffer supports features other than non-FIFO switching which contribute to its high performance. With *shift register addressing,* [Tami88a] the DAMQ buffer can transfer data into and out of a large buffer memory at very high bandwidths. This addressing scheme also efficiently supports implementation of a dual ported buffer memory, which is in turn needed to support *virtual cut-through* [Kerm79]. Virtual cut-through is the ability to begin to forward a packet as soon as it has been routed, i.e. before it has been completely received. This has the potential to significantly reduce the latency of packets traveling across a lightly loaded network. Finally, the DAMQ buffer can support variable size packets; the design discussed in this thesis supports packets of one to thirty-two bytes in length. Supporting variable length packets reduces the average packet latency, increases the system throughput and is essential in a high-performance system running fine-grained distributed tasks.

1.3. The Scope of this Work

In this dissertation we present the architecture of new buffer, the dynamically allocated multi-queue buffer. We demonstrate how the effect of output port contention on switch throughput is reduced by organizing the packets within a buffer into multiple queues and associating the queues with each of the switch's output ports, thus allowing the packets to be accessed based upon where they are routed. This scheme is extended by associating the queues not only with output ports but also with packet priority levels. This extension allows the DAMQ buffer to support distributed real-time computing by providing ''special'' service to high-priority packets.

A comprehensive analysis of the DAMQ buffer is provided by (a) exploring the implementation considerations of the DAMQ buffer versus both FIFO buffers and alternate non-FIFO buffer architectures and (b) by extensively analyzing and simulating switches implemented with these buffers. The simulations are in the context of a multistage interconnection network, and demonstrate the efficacy of the DAMQ buffer for both a general purpose multicomputer, where the average latency per throughput and the maximum network throughput are the key performance measurements, and for a distributed real-time system, where the maximum latency of high-priority (real-time) packets for a given network throughput is the key measurement.

An implementation of a DAMQ buffer is presented, and it is shown that the complex control required for a DAMQ buffer does not reduce its raw bandwidth relative to a simpler FIFO buffer. Further, the implementation reveals that the DAMQ buffer control logic does not require significantly more silicon area to

implement than the FIFO buffer control logic.

Finally, this dissertation explores communication flow control mechanisms for scalable multicomputers. We argue the necessity of implementing hop-level flow control for large scale multicomputers. Several hop-level flow control mechanisms are evaluated, both on the basis of their cost of implementation and on the basis of the degree to which they impact communication network performance under a variety of data traffic conditions. We found that no single hop-level flow control mechanism that we explored out-performed all of the other mechanisms in every traffic pattern that we evaluated. However, it is demonstrated that, due to the fact that it dynamically allocates its buffer space on a per-packet basis and maintains its internal organization via linked lists, the DAMQ buffer can support a wide variety of flow control mechanisms.

In the next chapter, we review previously published work in the areas of communication buffer design, switch design, flow control and support for real-time multicomputer / multiprocessor communication. In Ch. 3, the architecture of the DAMQ buffer is presented. The body of the chapter examines switch and buffer architecture tradeoffs, evaluating them on the basis of their ability to promote high performance communication. The tradeoff evaluations result in the specification of several non-FIFO buffer architectures, including the DAMQ buffer. Finally, details of the DAMQ buffer architecture such as the implementation of multiple queues of packets within the buffer as linked lists and support for variable length packets are presented.

In Ch. 4, the DAMQ switch is evaluated in the context of a multi-stage interconnection network by comparing it to alternative switch architectures,

including a switch composed of FIFO buffers. The comparisons are made with the assumption that the ‘‘raw bandwidth’’ (the instantaneous bandwidth for each input and output port) is limited by the rate at which the link can be driven, as opposed to the rate at which a buffer can be accessed, and thus the raw bandwidth is equal for switches using the different buffer architectures. The comparisons were performed using Markov analysis and event-driven simulations of a synchronous communication network. These evaluations demonstrate the performance gains achievable by using the DAMQ buffer.

Chapter 5 examines the idea of utilizing the non-FIFO property of the DAMQ buffer to support multiple classes of packets. By dedicating a packet queue to ‘‘high-priority’’ packets, a network can guarantee a low maximum latency for critical network traffic. The simulator presented in Ch. 4 is used to evaluate the high-priority queue.

Chapter 6 evaluates the performance of the DAMQ buffer in the context of an asynchronous communication network. Since the DAMQ buffer design discussed in this dissertation has features which cannot be used in a synchronous protocol — support for variable-length packets and virtual cut-through — it is important to evaluate the performance in an asynchronous environment. The omega network simulation results in this chapter support the results and conclusions of Ch. 4. Also in this chapter, it is shown that the DAMQ buffer can be used to implement deadlock-free dimensional routing for the torus network topology, similar to that presented in [Dall86]. An enhancement to this routing mechanism is proposed and evaluated via simulation of a two-dimensional square torus network.

A VLSI implementation of the DAMQ buffer is described in Ch. 7. The

DAMQ buffer is implemented as a functional unit independent of any single switch architecture. This single-chip DAMQ buffer (the *DAMQ Chip*) is designed for use as the communication interface for a multicomputer node. A comparison of the silicon area required to implement the DAMQ buffer versus a comparable FIFO buffer is presented, and circuit analysis demonstrates that the complex control logic of the DAMQ buffer does not limit the clock frequency or reduce the buffer's raw bandwidth.

The evaluations of communication network performance presented in Chs. 4-6 focus on a buffer's ability to forward packets as the key to network performance, keeping other network features constant. Yet, there are other communication network features which not only affect network performance, but also interact with the buffers in such a way as to enhance the performance of some buffer architectures over others. Of particular interest is the flow control mechanism employed by the network. Ch. 8 discusses previous work in the area of flow control for packet switching communication networks. It establishes hop-level flow control as an integral component of a communication network for a scalable distributed computer, and presents some alternative hop-level flow control mechanisms. The relationship between the flow control mechanisms and the buffer architecture is discussed, and their impact upon communication network performance is measured via simulation. Then the complexity / cost of implementing these mechanisms is examined, and the chapter ends with a summary of hop-level flow control.

Chapter 9 summarizes the dissertation. It reprises the contributions claimed by this dissertation and reviews the analyses and conclusions of each chapter. It is

argued that the DAMQ buffer is an appropriate choice when implementing a communication switch intended for large scale multicomputer and multiprocessor packet switching networks. The chapter concludes by presenting potential topics of future research.