X-TREE AND Y-COMPONENTS

C.H. Séquin and R.M. Fujimoto

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

ABSTRACT

The influence of VLSI technology on the construction of distributed computing systems composed of hundreds of computers is investigated. After a review of Project X-tree, the concept of a modular communications domain to carry the inter-processor message traffic is introduced and analyzed. A separation of the switching circuitry from the user processors permits communication and computation to take place concurrently. It also provides the flexibility to match the network topology to a particular application and facilitates the construction of heterogeneous networks.

A design is presented for a set of VLSI building blocks that permit the construction of high-bandwidth networks of arbitrary topology, providing the modularity needed for incremental expansibility. The proposed VLSI switching components support message-based, virtual-circuit communications over dedicated, time-multiplexed links. The simplest representative is the Y-component, a message switch with only three ports. Its usefulness for the construction of a variety of networks is shown with the analysis of simplified models. Some simulation results are presented that corroborate these findings.

Proc. Advanced Course on VLSI Architecture
University of Bristol, England, July 19-30, 1982;
To be published by Prentice Hall, 1983.

X-TREE AND Y-COMPONENTS

C.H. Séquin and R.M. Fujimoto

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

1. INTRODUCTION

The ever-increasing component reliability resulting from the use of very large scale integration (VLSI) will soon make systems with 10^8 transistors possible. While the research on RISC[Patt82, Séqu82] explores the influence of VLSI on the architecture of a single processor, this paper discusses ways in which VLSI affects the construction of distributed computer systems.

The processing power of a general purpose computing system can be increased in two ways. One approach, which has the advantage that old software can be re-used, is to increase the speed of the processor by technological means while maintaining its architecture. The huge investment represented by existing software fuels the effort to make GaAs technology[Long80] or Josephson junction processors[Ghee82] commercially viable.

A second approach to super-computers relies on a more general exploitation of parallelism, for instance by using a large pool of relatively inexpensive computers that operate in parallel on various subtasks. It is in this latter approach that VLSI can have a truly dramatic impact. We will discuss the constraints imposed by this technology and make design recommendations for the building blocks needed to construct such systems.

A key design parameter of multi-computer systems is processor *granularity*. At one end of the spectrum, some people propose to place an entire multiprocessor system on a single chip. Examples are the special purpose systolic array processors[Kung80] which are particularly suitable for high-throughput signal processing applications, or the 'tree-machine' developed at Caltech[Brow80] In both cases the unit to be replicated is small, and thus the granularity of the system is rather fine. The other extreme, using very large granules, is exemplified by such supercomputers as the S1 which employs a few very fast, pipelined processors interconnected with a number of memory modules through a crossbar switch.[Widd80] Commercially available multiprocessor systems built by IBM [Ensl74a] or UNIVAC [Ensl74b] also belong to this category.

In our earlier work on X-tree [Desp78, Séqu78] we have advocated an intermediate *granule* size equal to that of a single VLSI chip. For a general purpose system, some minimum complexity is needed in each node to enable them to cooperate productively. A "nano-processor," of which several dozens could fit on a single chip, is too small a building block for a general purpose computer. On the other hand, a very large granule size forces closely coupled components

such as a processor and its associated memory to be implemented on separate chips, thus increasing the performance penalties resulting from off-chip communications. An intermediate granule size equivalent to a RISC-like processor and its memory forms an entity with enough processing power for general purpose computing, but is still small enough to be implemented on a single chip.

Advances in VLSI technology are making general purpose computing systems composed of thousands of processors economically feasible. The processors, however, comprise only a portion of the system. The communications system that interconnects the processors is of equal importance. The performance of many multiprocessor systems has been limited by insufficient inter-processor input/output bandwidth. Furthermore, the communication system may dominate the hardware cost. In Cm*, for example, the k-maps responsible for setting up the communications paths were considerably more expensive than the processors.[Swan77a] It is clearly desirable to also exploit VLSI technology to reduce the cost of the switching hardware. This paper discusses issues in the design of universal VLSI components to be used as the building blocks for robust, high-bandwidth communications networks with enough flexibility to serve a wide variety of multiprocessor configurations and applications.

2. REVIEW OF PROJECT X-TREE

Project X-tree explored how the rapidly increasing computing power of LSI and VLSI processors can best be utilized for the construction of powerful general purpose computer systems.[Desp78, Séqu78] The special constraints of VLSI chips have a significant impact on the overall systems architecture. For the near future, transistor count and allowable power dissipation must still be viewed as limited resources for the implementation of single-chip computers.[Patt80] Furthermore the number of pins that provide access to such a chip and the total bandwidth that can be provided through these pins is also severely limited. System partitioning thus becomes crucial. Self-contained action within each individual chip must be emphasized.

Single-chip computers that combine processor and memory on the same chip have existed for many years, however they typically lack the necessary hardware modules for efficient inter-processor communication. Project X-tree thus focussed on a study of the communications issues in multiprocessor systems. We started with the study of a somewhat artificial model that relied on only a single VLSI component containing processor, memory and communication circuitry as a building block to construct large computing systems (Fig. 1).

Many topologies for point-to-point networks were investigated for their potential to provide high-bandwidth, low-latency communications between nodes in systems containing thousands of processors. For large systems a single bus or a ring network is clearly inadequate. The former becomes a bottleneck due to the interference of the communications between separate pairs of nodes; in the latter the average distance between pairs of nodes increases linearly with the size of the system. Lattice or tree networks are much more attractive since the average distance grows only as the root or the logarithm of the number of nodes.

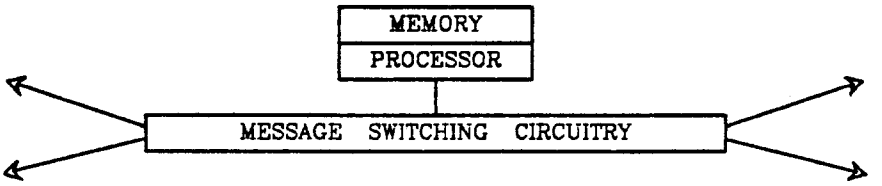

Figure 1. *A VLSI Building Block containing Processor, Memory, and Message Switching Circuitry.*

Emphasis was placed on a modular, incrementally expandable approach, in which the system could grow a few nodes at a time without the need for major reconfiguration of the already existing part. Almost from the beginning, our research concentrated on *dedicated links* rather than on a hierarchical system of busses. A great deal of research has been done in the latter area,[Swan77b] whereas the former area has been virtually neglected. Furthermore, the use of dedicated links puts all of the switching circuitry inside a single node; system configuration will thus have little effect on the electrical characteristics and the timing of these very localized, integrated switches. This approach is particularly suitable for the implementation of a multiprocessor system with single-chip VLSI nodes.

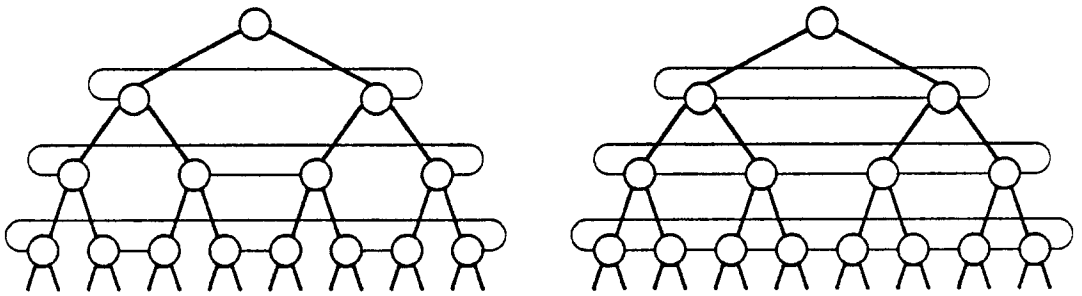

Figure 2. *Half-ring and Full-ring Binary Tree Topology.*

Comparisons of various network topologies[Desp80] and results of static and dynamic simulation[Suhl78, Gold79] have led to the selection of tree-structured network skeletons enhanced with horizontal links to form full-ring or half-ring binary trees (Fig. 2). The additional horizontal links shorten the average path length between pairs of nodes, distribute traffic more uniformly throughout the network, and provide the redundant paths necessary for fault-tolerant communication systems.

In X-tree, communication between nodes was to take place on byte-wide, half-duplex links. A TTL prototype of such a node was built, and the communication bandwidth of the parallel bidirectional link was investigated.[Laur79] Several important lessons were learned from project X-tree:

(1) Too much time is wasted in the low-level handshaking operation waiting for a byte to be checked and acknowledged. A unidirectional link sending larger contiguous blocks of data seems much more attractive.

(2) The restriction to a single type of building block is too artificial. The freedom to construct heterogeneous systems using special purpose processors with different instruction sets is important.

(3) Packing a suitable amount of processor, memory, and communication circuitry into the same chip will be a challenge for the near future. An alternative approach is to split the processor from the message switching circuitry and to define a suitable interface between the two.

(4) Considering only networks with the topology of a full-ring binary tree is too restrictive; more freedom to choose the topology most suitable for a particular application must be provided.

With these insights, a more generalized concept of a multicomputer system based on VLSI single-chip computers has emerged, which will be discussed in the remainder of this paper. It is based on general purpose VLSI switching components that permit the implementation of a large variety of network topologies. The next two sections present the envisioned multiprocessor system in a top down manner and derive the design constraints for the nodes of the network.

3. A MODULAR HIGH-BANDWIDTH COMMUNICATIONS DOMAIN

Key aspects and the main elements of a communications domain constructed from modular VLSI building blocks are introduced in this section. To make the example of such a system more concrete, we use specific numbers for some of the design parameters, even though no data exist to prove that the selected values are optimal.

3.1. The Concept

We envision a collection of VLSI communications components that can be combined into networks of high bandwidth and arbitrary topology. Any processor with the proper interface can be attached to this communication system. Only a few types of VLSI building blocks are required, providing modularity and incremental expansibility. The idea is to develop components that plug together easily, and completely hide from the user the details of the information transfer between nodes. Just as suitably designed logic gates have freed the electronics engineer from considering the analog behavior of the devices of which the gate is constructed, these new communications modules should produce an abstraction above the bit level. They should have built-in facilities for low-level functions such as handshaking, buffering, and flow control, so that the information packet or the block of data to be transmitted is the lowest primitive the systems builder needs to be concerned about. For the user of the final system the network provides end-to-end communication much like the telephone system.

Figure 3 gives a conceptual view of such a system, divided into a communications domain (C) using these VLSI communications components, and a processor domain (P) dedicated primarily to the user's computations.

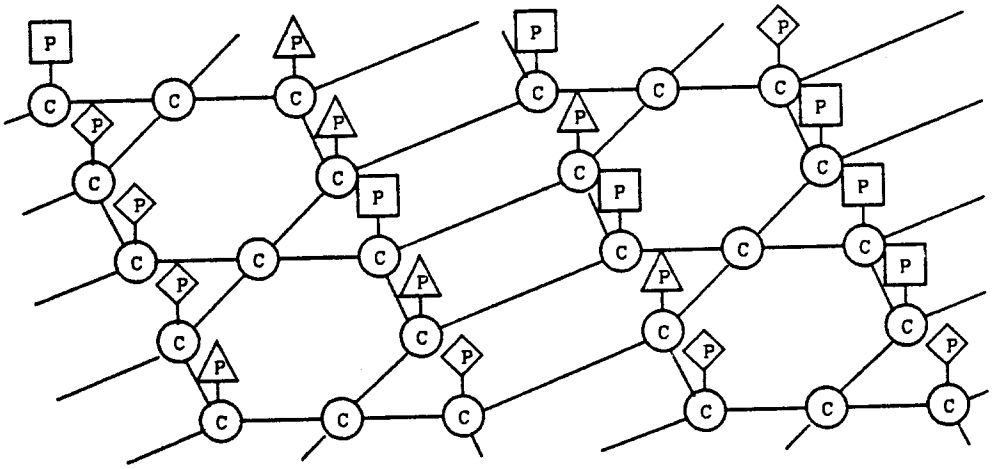

Figure 3. *Separation of a Multiprocessor into a Communications Domain (C) and a Processor Domain (P).*

3.2. Design Considerations

Required properties of the communications domain include unrestricted network topology, modularity, incremental expansibility, decentralized control, and the ability to recover from certain classes of failures. Some applications envisioned for multiprocessors (e.g. circuit simulation and signal processing) require more bandwidth than traditional local network hardware (e.g. rings or ethernet-like bus structures) can provide[Fuji82] Because of such high-bandwidth requirements, our research has focused on local networks using a large number of dedicated links. The proposed communications domain is designed to support high-bandwidth, low-latency communications among a large number, possibly thousands, of processors.

The types of processors used in the processor domain may vary depending on the application, but the interface to the communications system is standardized. This separation of the communications domain and the computation domain relieves the processors of much of the overhead associated with the forwarding of messages destined for different nodes. It makes possible the development of general purpose communications hardware that is suitable for a wide range of a applications and also provides the flexibility to construct heterogeneous systems containing many different types of specialized processors.

One may note the similarity between the components described here and an ARPANET IMP[Hear70] Indeed, many problems associated with loosely coupled computer networks (e.g. routing, buffering, flow control) also appear in this context. However, our design is not merely a scaled down version of the ARPANET. The key differences arise from the aim at higher bandwidth and lower latency, intrinsically lower error and failure rates within the communications hardware, and the envisioned implementation in VLSI.

3.3. Message-Based Communication

The communications domain studied in this paper is a message based network using a *virtual circuit* transport mechanism. Each processor has a fixed number of input and output circuits for receiving and sending data respectively. Sending a message is a three step process. First, a virtual circuit (i.e. a path of time-multiplexed links) to the destination processor is established by sending a message header with routing information through the network. Once a circuit is set up, an arbitrary amount of data, which may consist of several logical messages, can be sent along this circuit. Data can follow the message header immediately without an end-to-end handshake and need not be transmitted continuously for the circuit to remain intact. This approach reduces the routing overhead on all packets except the message header. When the circuit is no longer needed, it is torn down by sending a tagged message trailer.

The communications system provides only a data transport facility. Except for the header and trailer information, all data passes uninterpreted through intermediate nodes. Error checking and retransmission are left to an end-to-end protocol so that forwarding of data packets can begin immediately if the proper outgoing link is idle (*virtual cut-through*[Kerm79]) and need not be delayed until the entire packet has arrived in the buffer. VLSI components and local area networks promise very low error rates[Shoc80] making this approach possible.

3.4. Virtual Circuits

The communications domain can be viewed as a simple, connected graph. Nodes and edges represent communications components and links, respectively. A circuit from one processor to another corresponds to a path in this graph. Two distinct paths (say from node A to B and from C to D in figure 4) may use a common edge (from X to Y). Thus, the link associated with that edge must be multiplexed between the two paths, and provisions must be made to ensure that data from A is sent to B, and not to D.

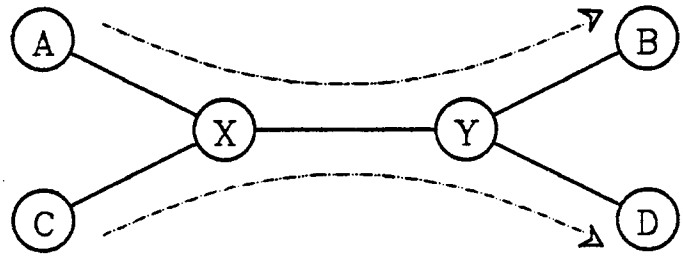

Figure 4. *Two Paths Multiplexed through the Same Link.*

Each physical link is divided into some fixed number of unidirectional *channels*. Each channel can carry data for one circuit (i.e. one path). Thus, a circuit from one node to another consists of a sequence of channels on the links in the path between the two nodes. The circuit from A to B in figure 4, for example, might use channel #3 to get to X, then #5 to get to Y, and finally #0 to get to B.

When node X sends data to node Y, the latter must determine which circuit this data belongs to. Two commonly used techniques for providing this information are, among others:

(1) Divide the link into a fixed number of *time slots*, and statically assign each time slot to a channel (e.g. the first time slot might be assigned to channel #0, the second to channel #1, etc.). The time slot on which the data arrives identifies the channel that sent it.

(2) Precede the data with a tag that identifies the channel it is being sent on. In this scheme, the available bandwidth on the link is allocated to the various channels by some demand-driven scheduling algorithm.

In the first scheme, the link is effectively divided into a number of lower bandwidth links, with the sum of these bandwidths equal to that of the physical link. If a channel does not send any data, its allocated bandwidth is wasted. In addition, latency is increased, since each channel has to wait for its turn to send a unit of data. In the second scheme, the entire bandwidth of the link can be allocated to channels upon demand, i.e. when they have data to send. For these reasons, demand-driven time-multiplexing appears superior.

The time-sharing of the physical links also increases hardware utilization. Since not all channels will always have data to send, one would like to have enough channels to keep the link utilized as effectively as possible. On the other hand, the delay seen by each channel increases as more channels compete for the use of the link. The number of channels on each link is thus chosen as a compromise between a set large enough to permit frequently used virtual circuits to exist almost indefinitely but not so large that the latency due to the time-multiplexing becomes excessive. Initial studies indicate that 16 channels might be a reasonable compromise between these conflicting goals. More studies aimed at specific applications are in progress to verify this result.

In order to "link together" the subsequent channels corresponding to a circuit, each node maintains a set of *translation tables*. There is one translation table for each input port of a node. Each entry of the translation table contains two fields: an output port, and the number of a channel on that port. When data arrives on an input channel, say channel #3, entry 3 of the translation table for that port is read to yield the output port and the number of the channel the data is to be forwarded on.

3.5. Routing Hardware

The translation tables logically link incoming and outgoing channels, and thus establish the various virtual circuits through the node. Setting up these circuits involves allocating channels and updating translation tables along each path from source to destination. This task is performed by a *routing controller* residing in each communications node.

Initially, all translation tables specify that data is to be sent to the local routing controller. When a message header setting up a new circuit arrives at a node, the routing controller analyzes the destination address in the header, determines the proper output port with the use of some routing algorithm, allocates a free output channel, and updates the translation table at the input port.

Measurements on a TTL prototype of such a routing controller[Fuji80] show that this entire operation can be done in 4-5μsec if a free channel is available on the selected link. Subsequent data is then forwarded without intervention by the routing controller. Similarly, when the circuit is torn down, the channel is released, and the corresponding translation table entry is reset to point to the routing controller.

4. KEY FUNCTIONS OF THE COMMUNICATIONS COMPONENTS

In this section three key functions of the nodes in such a communications network will be discussed: routing, buffering, and flow control.

4.1. Routing

In a multiprocessor network, some routing algorithm is necessary to build a path between communicating nodes. A great deal of research has been done in the area of routing in loosely coupled computer networks, and much of this work is applicable here. In the context of the proposed communications domain we will only consider totally distributed routing that does not rely on any centralized authority. For this discussion it is also appropriate to distinguish between regular networks with a predefined topology such as arrays or binary trees, and irregular networks of arbitrary connectivity.

In regular networks, routing can be performed in each node by a state machine, which performs a fixed algorithm based on the difference between the local address and the destination address. In square lattices, for instance, the routing controller could forward the message header in a direction that would reduce the difference of the x- and y- coordinates of the current and the destination nodes. Routing algorithms for binary half-ring or full-ring trees have been discussed elsewhere[Séqu78] Another interesting class of fault-tolerant networks with distributed routing algorithms has been presented recently[Prad82]

For a general purpose communications component, the routing algorithm must not be frozen in hardware. A routing controller with a writable program memory is more appropriate and guarantees that the same component can serve many different network topologies. A routing algorithm suitable for the particular network structure could be broadcast at system initialization.

For irregular networks, routing may be based on suitable look-up tables. In a decentralized system each node i has entries of the form:

$$NN = R_i(DN),$$

implying that messages destined for node DN are forwarded by node i to neighbor node NN. This look-up table, commonly called a ***routing table***, can be defined statically or it can be maintained dynamically using information exchanged between neighboring nodes. The latter approach also allows the network to automatically reconfigure itself should the topology change due to node failure or network expansion. Many interesting techniques have been developed for initializing, maintaining, and reducing the size of the routing tables[Gerl81]

4.2. Buffering

Each message passed into the communications domain must be subdivided by the sender into some number of fixed-length packets. These packets form the unit of data transmitted across the links of the communications domain. Due to conflicts that arise when several packets simultaneously require the use of the same link, buffering is required in each node.

A scheme is necessary to allocate a node's buffers among the virtual circuits using the node. A straight-forward solution is to give each channel on each link a separate buffer. This is inefficient, however, since much of the buffer space will be unused most of the time. By allowing several channels to share the same buffers, the fluctuations in the need for buffer space can be averaged over a larger number of communications paths, thus achieving better utilization of the precious resources on the chip. A mapping is then required linking each channel to the buffers holding its packets so that the latter can be found when it is time to forward them. Furthermore, when a new packet arrives, an empty buffer must be found. From this perspective, buffer management is similar to the management of a cache memory: a program (here: a channel) needs to fit blocks from main memory (packets) into cache pages (buffers).

As in cache memory design, there are three well known schemes for performing this mapping: direct mapping, set associative mapping, and fully associative mapping. In turn, these three schemes offer an increasing degree of buffer sharing, and thus improved memory utilization, but at the cost of increased complexity of the control circuitry. They are distinguished by restrictions on where a channel's packets can be placed. In the direct mapping scheme providing minimal sharing, each channel has a set of buffers dedicated to it, e.g. its own FIFO queue. The set-associative scheme (moderate sharing) allows each channel to use a larger set of buffers, but it is no longer given sole access to them. It might be realized by letting all channels of a single port share a pool of buffers dedicated to this port. In the fully associative scheme (maximal sharing), each node has a centralized pool of buffers which all channels share. Possible implementations of a set-associative and fully associative buffer management schemes will be discussed in a later section. An implementation of ports using the direct mapping scheme has been described previously[Laur79]

4.3. Flow Control

Flow control refers to the mechanism which regulates the transmission of data packets along the virtual circuits, and ensures that buffers do not overflow in heavily utilized nodes. In particular, one must anticipate the case in which a processor is sent more messages than it can immediately receive.

Flow control can be achieved by controlling buffer allocation within the communications network. If a node is inundated with data, packets will "back up" along the virtual circuits leading up to it, much like the way cars back up on a congested freeway. By controlling the maximum number of buffers each channel can use, "buffer hogging" (i.e. one channel using more than it's share of buffers) is avoided. Buffer hogging could potentially impede other traffic using the node, and lead to deadlock situations. The direct mapping scheme, and to a lesser extent the set-associative scheme, automatically provide protection

against buffer hogging, since they inherently restrict buffer sharing. All three schemes however, need some mechanism to ensure that data is not lost if no free buffers are available.

The simplest flow control scheme uses a send/acknowledge protocol to transmit data over the link. If the receiver cannot allocate a buffer to the incoming packet, it is discarded and a negative acknowledge is returned. The sending node must later retransmit the packet over the link. Such a negative acknowledgement entails wasted link bandwidth which grows in proportion to the packet size. Furthermore, one would like to be able to send a second packet on a channel without waiting for an acknowledgment of the first. Allowing multiple unacknowledged packets to be pending for the same link, however, adds a considerable amount of complexity to the port.[Pouz78]

An alternative approach is to implement the buffer allocation policy for a node in its neighboring nodes, i.e. by controlling the flow of information from the sender end of each link. For example, each output port could maintain a table remembering how many buffers in the neighboring node are allocated to each channel of the link in between. With this information, the sender can decide which channel to serve next, and packets can be forwarded without the possibility of overflowing the buffer space in the receiver. Maintaining this remote status information requires some overhead. The fact that a buffer has been freed up must be reported back to the transmitter. Still, for heavy network traffic, this overhead will be less than the bandwidth lost due to the negative acknowledgments in the ACK/NACK scheme.

5. IMPLEMENTATION OF COMMUNICATIONS COMPONENTS

This section discusses some key issues in the single-chip implementation of a node of the communications domain. After a discussion concerning the links and the number of ports, we present two possible designs for a VLSI communications component: the first one with minimal hardware complexity, and a second one demonstrating techniques for improving performance.

5.1. Constraints Resulting from Single-Chip Implementation

A generally usable VLSI communications component is a very attractive building block since it can expect a large market. A single-chip implementation also minimizes the propagation delay penalties associated with signal transmission through the chip periphery, but imposes certain constraints that have a marked impact on the design of the overall communications system.

First of all, a VLSI node must be constructed within the resource limitations of integrated circuits chips. For the near term this will mean a limited number of pins (say 80) and transistors (say 100,000). The former implies restrictions on the number of ports for each component, while the latter restricts the amount of internal buffer space and the complexity of the routing and error recovery functions that can be implemented. Furthermore, power dissipation of the packages typically used is limited to a few watts and implies that the total I/O bandwidth of each chip is strictly bounded. This maximum bandwidth of data transfer through the chip periphery must be shared among all communications ports and among all wires within a link.

5.2. Implementation Trade-offs for the Links

Because the major bandwidth limitation results from power considerations, a *parallel* link will not necessarily provide higher bandwidth than a *serial* link. The chip area and power budget allocated to the drivers of the individual strands of a parallel link can just as well be combined into a single more powerful driver for a serial link. A serial link has the advantage that the data rate will be limited only by the dispersion on the serial line rather than by data skew and synchronization problems between parallel bits.

The bi-directional link required between neighboring nodes can be implemented as two individual unidirectional lines, in place of the link with alternating direction of transmission that was used on the TTL prototype of the X-tree node.[Lee79] The use of unidirectional lines avoids the long delays resulting from the full turn-around and acknowledgement of each packet on the X-tree link. Data can now be sent in a pipelined fashion limited only by the available transmission bandwidth of each line.

However, the forward and return path on a link are now much less tightly coupled, and typical ACK/NACK schemes for flow control are unsuitable. Much of the advantage of the unidirectional link is lost if large packets are forwarded without the guarantee that there is enough buffer space at the other end. Thus forward-looking flow control, as discussed at the end of section 4.3, should be used.

5.3. Optimum Number of Ports

As mentioned above, the maximum transmission rate from each node is limited to some fixed value, say B units of data per second, given by the allowable power dissipation in each node. It will be assumed that this total bandwidth, B, is equally divided among the p ports of the node, i.e. each port can emit only B/p data units per second. A unit of data is defined as the amount of data which a node must buffer (e.g. a byte in X-node[Fuji80]) before it can begin forwarding that data to the next node. Ignoring for the moment queuing delays and the "speed-of-light" latency associated with transmitting the first bit of data across the link, the latency associated with forwarding one data unit over a link is inversely proportional to the bandwidth of the link, B/p, and thus directly proportional to p. Data traveling H hops through an idle network will then arrive approximately $\frac{H \times p}{B}$ seconds after it is sent. In the construction of a communications domain there is thus a trade-off between using nodes with many low-bandwidth links and using nodes with fewer but higher-bandwidth ports. Networks constructed with the latter generally require more "hops" to reach a particular destination node (Fig. 5).

An analysis of a simplified model of various network topologies shows that in most cases, reducing the hop count by increasing the number of ports does not adequately offset the lost bandwidth per port.[Séqu81] Specifically, the networks in figure 5 can be analyzed fairly easily for two special cases: 1) average latency of a message header between any pair of nodes and 2) average bandwidth available to an individual virtual circuit through the time-multiplexed links of a heavily utilized network.

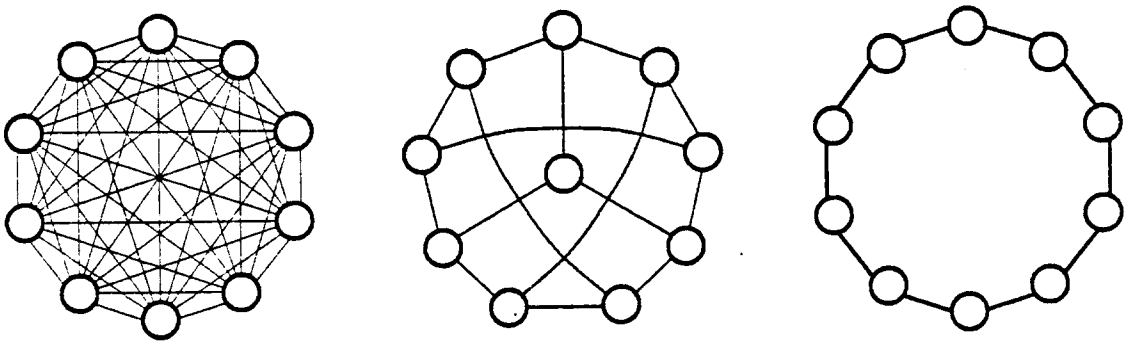

Figure 5. *Trade-offs in Network Construction using Nodes with Many Slow Links (left) and with Fewer Faster Links (right).*

Analysis of the first case shows[Séqu83] that in networks with less than a few hundred nodes, components with only three ports minimize average latency. In larger networks of up to several tens of thousands of nodes, a building block with four or five ports will give the best performance. This corroborates earlier results reported by Despain[Desp80] With respect to the average bandwidth between any pair of nodes, the use of components with three or four ports gives significantly better performance than a two-port component; however the performance continues to improve slowly with the number of ports until one has realized a fully interconnected network[Séqu83]

These results were obtained under the assumption that each node communicates an equal amount of information with every other node. In practice, one would try to map a problem onto a multiprocessor in such a way as to maximize locality of communications. Thus traffic between neighboring nodes (low hop count) should be weighted more heavily. If this is taken into account, the desirability of nodes with few, high-bandwidth links is increased. This result is corroborated by the simulation studies presented in chapter 6. In addition, we will see in the next two sections that the hardware complexity of a node increases more than linearly with the number of ports. This makes a strong case for VLSI communications components with relatively few ports, say from 3 to 6.

5.4. A Simple Switch: the Y-Component

The simplest useful communications component, which we call the "Y-component", has only three ports, and thus requires only a limited amount of buffer space and control logic. Due to the small number of ports, it is normally not possible to attach a computation processor to each component, since the resulting topology would then be constrained to a ring. If a "node" with a processor and more than two neighbors is desired, it must be constructed from several Y-components. In general, a "node" with p ports (one of which leads to a computation processor) can be constructed from p-2 Y-components (Fig. 6). More hops are typically required to transmit a message through such a *cluster node* of the network, but this is offset by the ability to use higher bandwidth links, as discussed above. Actually this is a way to circumvent the chip I/O

bandwidth limitation, since it permits one to increase the overall power consumption tolerable in a "node" of the network.

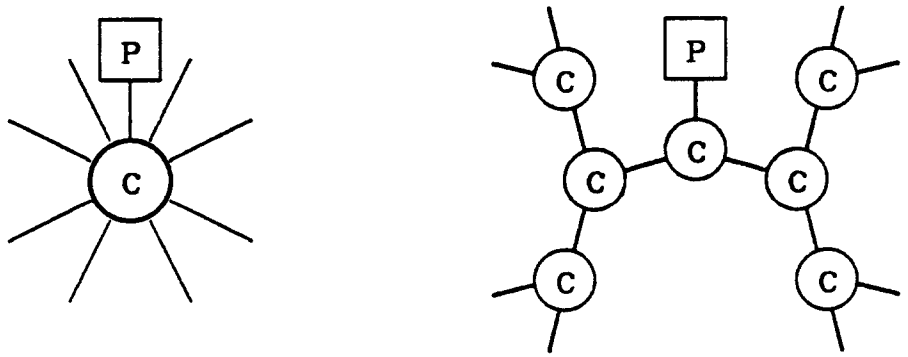

Figure 6. *Simple Node and Cluster Node. Composing a Node with 9 Links from a Cluster of 7 Y-components.*

The effectiveness of such a cluster node built from Y-components can again be evaluated for two simple situations: 1) average latency of a message header through a lightly utilized node and 2) average bandwidth available to an individual virtual circuit through the node in a heavily utilized network. In both cases we assume that a message is equally likely to arrive on any one of the p ports, and is equally likely to exit on any of the other $p-1$ ports. Average latency and throughput bandwidth through a switching node are listed in table 1 for the two basic approaches introduced in figure 6.

Table 1.
Average Latency and Bandwidth through a Switching Node

# of branches to neighbors	*# of levels in cluster*	*Latency*		*Bandwidth*	
		Simple	*Cluster*	*Simple*	*Cluster*
2	*1*	*3*	*3*	*1/3*	*1/3*
4	*2*	*5*	*7*	*1/15*	*1/11*
8	*3*	*9*	*11.6*	*1/63*	*1/38*
16	*4*	*17*	*16.6*	*1/255*	*1/130*
32	*5*	*33*	*22*	*1/1023*	*1/453*
64	*6*	*65*	*27.6*	*1/4095*	*1/1593*
128	*7*	*129*	*33.3*	*1/16383*	*1/5671*

When the number of branches emerging from a node has to be large, the cluster approach is clearly advantageous. The latency increases only proportional to the number of levels in the cluster, i.e. logarithmically with the number of branches, whereas in the simple node the latency increases proportional to the number of ports. In terms of average throughput bandwidth, the cluster node always shows better performance; and the ratio between the cluster and the simple node increases slowly with the number of branches.

In many network topologies the performance of the cluster node can be improved by using a routing algorithm that ensures that most of the traffic through the "node" will use the paths with fewest hops. By putting a higher weighting factor on the shorter paths through the cluster, latency is reduced. In a tree-structured cluster node the links near the root have the highest average utilization. By avoiding these more congested links as much as possible, the average bandwidth through the cluster is also improved.

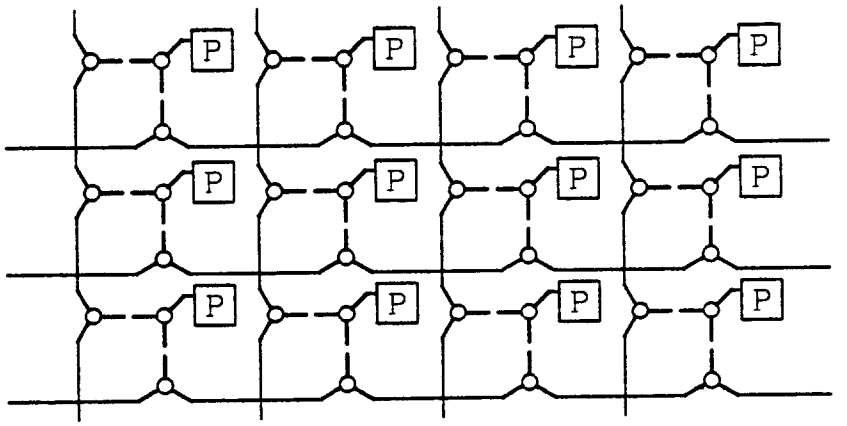

Figure 7. *Implementation of a Square Array with Y-components.*

The limited number of ports in the Y-component is not very restrictive in the realization of networks with a higher degree of branching at each node. Consider the realization of the square lattice shown in figure 7. Each node is composed of three Y-components. The processor is attached to the node in the middle, and the other two Y-components form chains running in the x- and y- coordinate directions, respectively. When routing messages to their destination, transfers between the x- and y- rails should be minimized since they require two extra hops. This can be achieved by using a routing algorithm that first routes the message to the proper x-coordinate and only then adjusts the y-coordinate. With this approach, there is a minimal penalty due to multiple hops within the "nodes" of the square lattice, and the higher bandwidth of the links of the Y-components can be taken full advantage of.

5.5. Design of a Y-Component

A block diagram for one design of a Y-component is shown in figure 8. The component consists of a routing controller, three input ports, three output ports, and one buffer module associated with each input port. In order to simplify the discussion which follows, it will be assumed that there are 16 input and 16 output channels and that each buffer module comprises 16 data buffers.

A set-associative buffer management scheme is used. Of the 16 buffers assigned to each input port, each channel is statically assigned, say, 4 buffers by means of some algorithm for mapping channel numbers to buffer addresses, e.g. channel i might be able to address buffers i, $i+1$, $i+2$, and $i+3$, where all sums are taken modulo 16. Thus, four channels share the use of each buffer.

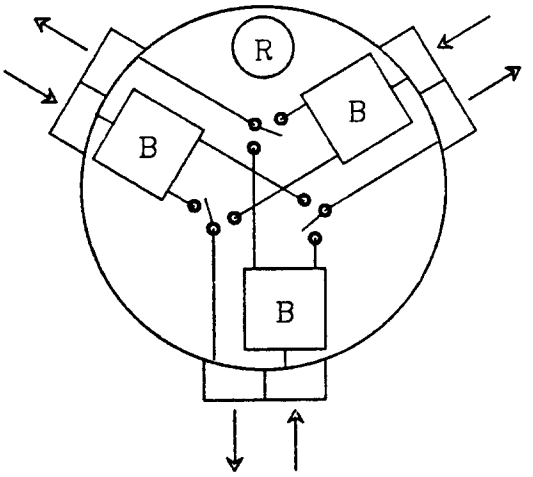

Figure 8. *Block Diagram of Y-component.*

A remote buffer management scheme like that described in section 4.3 is used for flow control. Each Y-component's buffers are managed by its neighboring nodes. When a component sends a packet, it not only specifies the number of the channel the packet is being sent on, but also the buffer that the receiving component is to use. After having forwarded the packet, the receiver will report back to the sender that the buffer is again available.

When a packet arrives at an input port, it is placed in the buffer specified by the sender inside the buffer module associated with this link. The other two output ports actively search through this buffer module. They forward the packet onto the link they control if it carries the proper output port tag. The buffer is then marked empty so that the neighbor which sent the packet can be notified. Some additional control logic ensures that packets on each channel are forwarded in the order in which they arrived.

The design of such a Y-component was carried out to the gate level to explore some of the issues presented above and to obtain a realistic estimate of its complexity.[Wong81] It is estimated that such a component would require roughly 100,000 transistors, of which approximately 40,000 are used in the data buffers.

5.6. Increasing the Performance of the Communications Component

A design for a communications component yielding somewhat higher bandwidth, lower latency, and more efficient buffer utilization than the switch described above has also been studied. It has been structured in such a manner that the number of I/O ports can be increased without adding unduly to it's complexity. The most distinguishing feature of this design is a single pool of buffers shared by all channels of the node (Fig. 9).

Since all packets traveling through a node must use this pool of buffers, it must have enough bandwidth to avoid becoming a bottleneck. This is achieved by interleaving the memory 16 ways, assuming packets consist of 16 bytes. Byte i of each packet is always stored in memory module i (MM_i). Each of the p ports can simultaneously load a packet into a buffer, provided no two use the

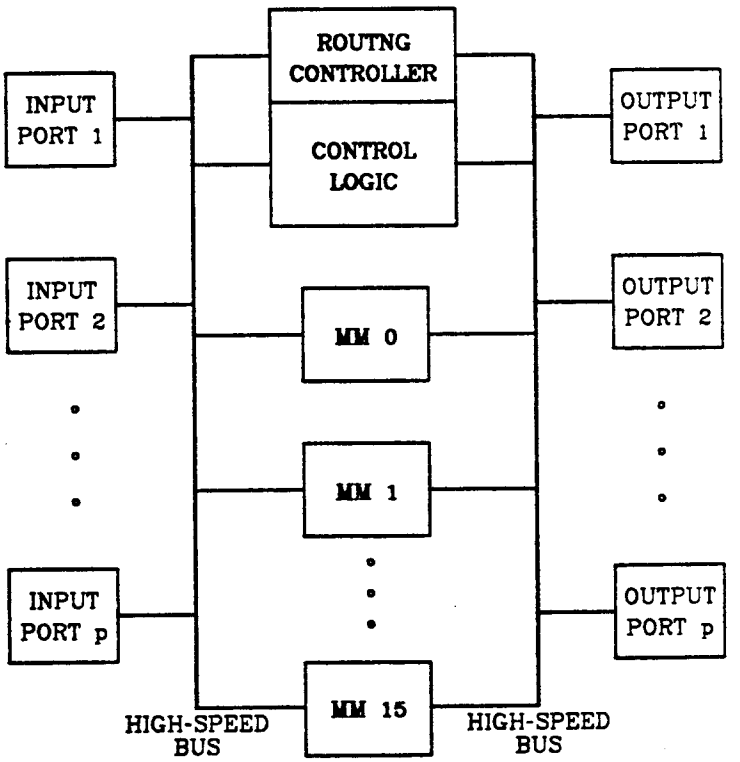

Figure 9. *Block Diagram of Higher Performance Component.*

same memory module at the same time. In the worst case, p packets simultaneously arrive at a node. Since only one port can be granted access to MM_0, additional registers are required to temporarily buffer the arriving data bytes until they can be stored in MM_0. On the next clock cycle, when the second byte of each packet arrives, one of these newly arriving bytes will be loaded into MM_1, and one of the temporarily buffered bytes can now be written into MM_0. Similarly, three accesses to the buffer pool will occur on the third clock, and so on. Eventually, each port will be able to access a different memory module on each clock cycle.

If the links can transmit one data unit (e.g. 1 byte) per clock cycle, then the communications component must be able to transport p data units from the input ports to the memory modules in each clock cycle. A high speed, time multiplexed bus performs this function. Since this bus remains entirely within the chip, it can run approximately an order of magnitude faster than the I/O links, which require off-chip communications.[Séqu78] A second high speed bus carries bytes from the memory modules to the output ports.

Since buffers are dynamically assigned to channels upon demand, a mechanism is required to keep track of which buffers are assigned to which channels at any given time. This is accomplished by maintaining a linked list for each output channel which lists the buffers waiting to be forwarded by that channel. When a packet arrives, it is placed at the end of the linked list corresponding to the channel the packet is to be forwarded on. It is removed from the list after it has been sent to the next node. These mechanisms are

implemented in hardware so that packet forwarding can proceed as quickly as possible. The linked lists ensure that packets are forwarded in the same order in which they arrive. Also, as with the previous design (Sect. 5.5), an output port can begin forwarding a packet before all of it has arrived, improving the latency associated with transmitting a packet significantly. It is assumed that all communications links operate at the same speed.

As with the Y-component (Fig. 8), a remote buffer management scheme is used for flow control. Here however, each output channel maintains a *count* of the number of buffers it is using, rather than keeping track of the actual buffers used. Sending a packet increments this counter. The counter is decremented when a control byte is received indicating that a packet earlier sent by that channel has now been forwarded. A channel is not allowed to send a packet if doing so would cause its counter to exceed some previously set *channel limit*. This scheme avoids transmitting buffer numbers over the link, and thus reduces the overhead associated with packet forwarding.

In addition to the counters for each channel described above, each output port maintains a single counter indicating the total number of buffers it is using in the remote node. This counter is not allowed to exceed a certain *port limit* that represents the number of remote buffers allocated to that port. As long as the count is below the port limit, a buffer exists in the receiving node to receive the next packet. By allowing the sum of the channel limits of a port to exceed its port limit, the efficiency of buffer utilization can be improved; an underutilized channel's buffers can be allocated to more highly utilized channels. In addition, a higher level protocol can change the quota of buffers allocated to each port to accommodate heavy traffic loads on some ports at the expense of less heavily utilized ports.

The performance improvement expected from this component over that of the Y-component arises from the reduced overhead associated with transmitting a packet across a link and the avoidance of polling translation tables to locate packets to be forwarded. This design also achieves a more efficient utilization of the buffer space due to the fully associative buffer management scheme. It does, however, require significantly more control circuitry than the Y-component. Common to both designs is the routing controller which is responsible for setting up paths through a node (i. e. updating translation tables and allocating output channels) and for implementing any protocols above the link level. One possible design of such a routing controller is described elsewhere.[Fuji80]

6. SIMULATION STUDIES

The study of the idealized network topologies that was used in the determination of the optimal number of ports[Séqu83] and in the analysis of the usefulness of the Y-component made some strong simplifications; in particular, it assumed a uniform traffic distribution between all pairs of nodes. To gain a more thorough understanding of the network utilization and the interaction of different messages, the execution of realistic applications on such a multiprocessor system must be studied. For this purpose a discrete-time, event-driven

simulation program has been written[Fuji82] in which the traffic in the communications domain is generated by real application programs executing some parallel algorithm.

6.1. The Simulator

The simulation program consists of three main components: the *application program*, the *simulator base*, and the *switch model* (Fig. 10). The application program is formulated as a number of tasks (or processes) which execute in parallel and communicate by exchanging messages. The simulator base time-multiplexes the execution of the individual tasks on the host computer, in this case a VAX-11/780. It ensures that interactions among tasks (e.g. message transmissions) are simulated in the proper time sequence by keeping track of time for each task by virtue of a clock which advances as the task executes. The software modeling the interconnection network is contained in a separate module, the *switch model*, permitting convenient comparison of different switching components. Each switch model provides a fixed virtual circuit interface for the tasks and simulates message passing between processors.

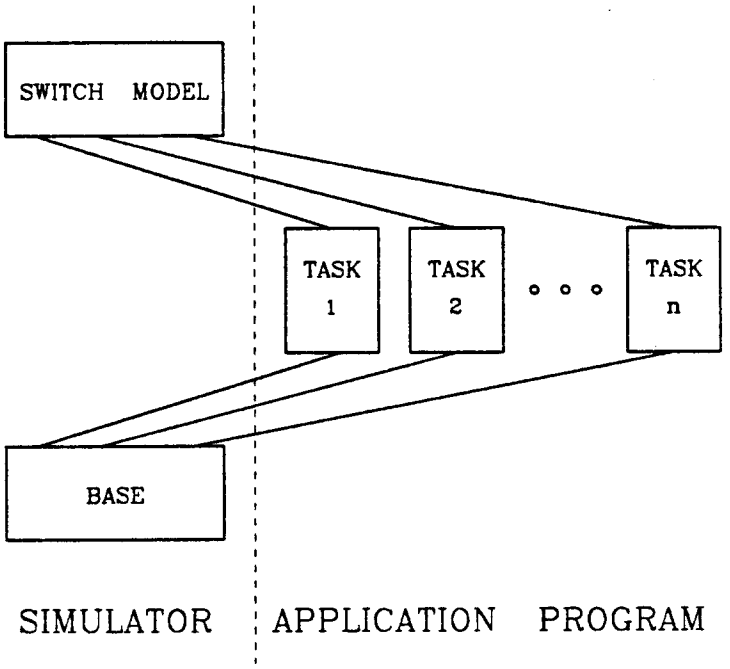

Figure 10. *Block Diagram of Simulation Program.*

A number of assumptions are made in the simulation experiments reported in this paper. First, the time overhead for invoking intertask communications, i.e. starting the transmission of a message, only takes the time of a single subroutine call. Although unrealistic (at least in current technology), this allows separation of the penalty due to operating system overhead from that inherent in the communications switch. This paper focuses on the latter. Studies analyzing the effect of operating system overhead are forthcoming[Fuji83]

The results presented here also assume that each task executes on a separate processor, each with about the computing power of a VAX-11/780. The switch models assume that messages are divided into fixed-length packets consisting of a one-byte header (e.g. a channel number) and 16 data bytes. If the output link selected by the routing algorithm is free, a node can begin forwarding a packet immediately after the first byte has arrived, and need not wait until the entire packet has been received.

Shortest path routing is assumed in all networks. The routing tables and all virtual circuits are set up before the simulation begins. As mentioned in section 3.5, the routing of a new message header can be performed fairly quickly; the corresponding overhead thus becomes negligible for virtual circuits that stay in place for a few thousand clock ticks. For the signal processing applications discussed in this section, all virtual circuits remain fixed throughout the entire simulation.

6.2. Issues under Investigation

So far, the simulator has been used to study the following three issues:

(1) What is the performance of a multiprocessor built from separate single-chip processors and communications components relative to one in which the communications circuitry is integrated directly into the processor chip.

(2) Given the use of single-chip VLSI communications components, what performance is achieved when Y-components are used instead of components with a larger number of ports?

(3) How much performance improvement is gained by building a multicast mechanism into the communications components?

The first comparison is included to provide a benchmark for the multiprocessors built with separate communications components. The comparison is not fair, since the integration of processor *and* switching hardware onto the same chip requires a more advanced integration technology. But this is the model typically used in most other studies of multiprocessor networks.[Witt81]

The second trade-off concerns the use of simple switching components with three ports to construct cluster nodes with the desired number of branches. This approach is compared to the situation in which there are "as many ports as necessary" for the topology, i.e. 3 ports for rings, 6 for full-ring trees, and as many as there are tasks for fully connected networks.

In many algorithms the same data is transmitted to several destinations. A *multicast* mechanism that distributes *multiple-destination packets* efficiently is expected to improve performance. Without a multicast mechanism, several single-destination packets are generated at the source node, and each one is routed separately through the network to its destination. Many of these packets will follow each other for a certain distance through the network, leading to inefficient use of the available bandwidth. The multicast mechanism combines these sequences of identical packets into a single "multicast packet." A new copy is not generated until the paths of the single-destination packets incorporated into the multicast packet separate.

Implementation of multicasting requires a more complicated header to carry the list of destination nodes when the virtual circuit is set up. The routing tables must permit multiple entries per incoming channel, and the switching circuitry needs some extra hardware to duplicate message packets. For some applications, the resulting performance improvement is well worth the extra hardware complexity.

6.3. An Application Program

To demonstrate the kind of results that can be obtained with this simulator and to further document our previous claims concerning the optimum number of ports for the switching components, we use a parallel signal processing program that relies on a technique developed by Barnwell.[Barn78, Barn79, Barn82] This particular application involves twelve separate processors that communicate as indicated schematically in figure 11. An "input processor" starts at time 0 to distributes a total of 400 samples of the input signal waveform to the ten "computation processors" which execute the actual filtering algorithm. Each data point received by one of these processors is combined with data generated by other processors to form a new intermediate result. This value is then broadcast to 6 other processors who use it in their computations. If the computation processors are numbered 0 through 9, processor i broadcasts to processors $i-1$, $i-2$, $i-3$, ... , $i-6$ (modulo 10). Finally, an "output processor" collects the results from the ten intermediate processors and reassembles them into the output waveform also containing 400 samples. It is assumed that the input samples arrive fast enough compared to the rate at which data points can be processed so that the execution time is not limited by the input data rate.

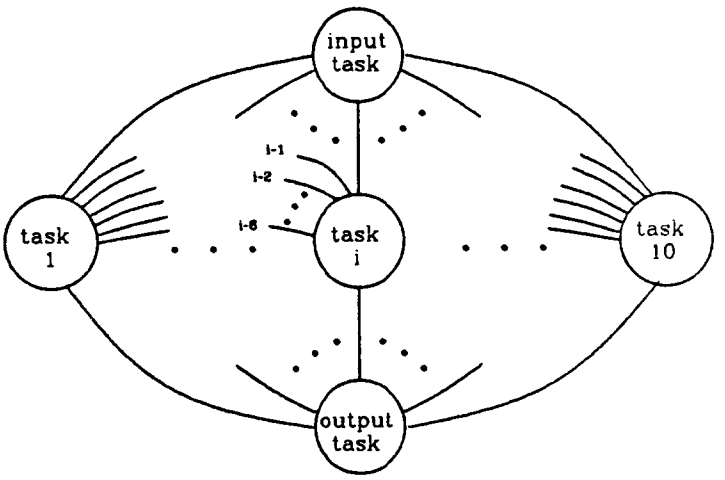

Figure 11. *Data Paths in Barnwell's Filter Program.*

This particular application is typical of many signal processing algorithms in that the computation consist of a few floating point operations generating a result which is then passed to other processors for further processing. An FFT computation, for example, exhibits similar behavior. Barnwell's algorithm does, however, exhibit a great amount of data sharing, and thus the benefits resulting

from multicasting are greater than in many other applications. Also, the number of tasks that are simultaneously executed is relatively small, making this example less useful for evaluating large multiprocessor networks. Nevertheless, it is a useful benchmark for exercising the simulator, and it provides one data point in the evaluation of distributed signal processing algorithms.

6.4. Simulation Results

The above application was simulated for multiprocessors with the following three network topologies:

(1) a 12-node fully connected structure,

(2) a full-ring binary tree with 4 levels,

(3) a bidirectional ring structure with 12 nodes.

In these structures, the number of ports per switching component ranges from two to twelve. In some simulations a node with b branches was emulated with a cluster of $b-2$ Y-components; in other cases the user processor and the switching circuitry were assumed to be on the same chip.

Figures 12-14 plot the speed-up of Barnwell's filtering algorithm relative to the total execution time obtained on a uniprocessor as a function of the total chip bandwidth. This chip bandwidth, B, ranging from 0 to 300MBaud/chip, is divided equally among the chip's communication links. Thus in the ring network of Y-components, for example, B equal to 300Mbits/chip/sec corresponds to 100MBaud links. The speed-up obtainable on a multiprocessor with an infinite-bandwidth, zero-delay interconnection system (i.e. a "perfect switch") is also included as a reference point to visualize the cost of communications.

For the fully connected configuration (Fig. 12), the multicast/non-multicast curves are identical if no communications components are used, since each node has a direct connection to every other node, and thus the separation of individual message packets occurs at the source node. The curves demonstrate that, under the assumption of a limited total chip bandwidth, the reduction in hop count achieved by using nodes with more branches is not sufficient to compensate for the lost port bandwidth. This is partially attributed to the use of *cut-through*, i.e. packet forwarding within each node begins as soon as the first byte of the packet arrives rather than after the entire packet has been received. This immediate forwarding reduces the delay on messages proportional to the number of hops.[Kerm79] For the chosen parameter values, the reduction in delay due to cut-through can be up to an order of magnitude.

An important observation is that for the network topologies studied, an implementation with Y-components yields better performance than other realizations. This is true even for the fully interconnected case with nodes in which the communications circuitry is included on the same chip as the processor, – a situation that requires only a single hop between any pair of nodes. The reason lies in the higher-bandwidth links in the cluster nodes built from Y-components. The total I/O chip bandwidth in such a cluster is much higher than the I/O bandwidth of a single-chip node, since 10 times as many chips are used. Thus, the increased performance does not come for free.

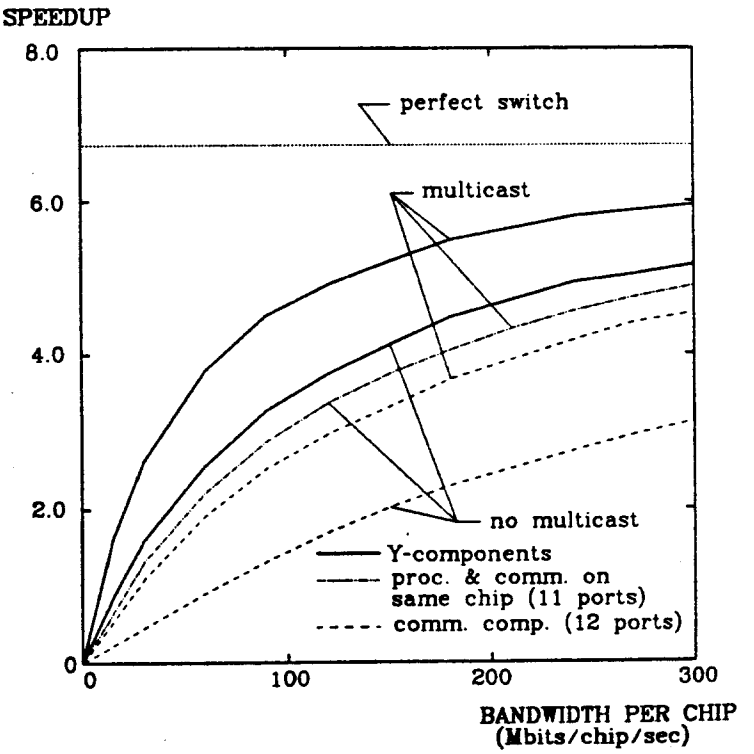

Figure 12. *Performance of Fully Connected Topology.*

Multicasting also leads to improved speed-up, which is most significant when the performance is limited by the network bandwidth. If multicasting is not used, the link from the processor to the fully connected communications domain becomes a bottleneck when broadcasting data, since a queue appears instantly in front of this link on each broadcast. This causes increased message delays and longer execution times.

Regardless of the question of multicasting, the performance of the fully connected network of switching components can be improved if the total chip bandwidth is assigned differently to the individual ports. Half the total chip bandwidth should be given to the link to the user processor since every message going through the switching node must also go through this critical link. However, the discussion of such asymmetrical switching components is beyond the scope of this paper.

The results of the study of the full-ring binary tree network are shown in figure 13. The input task, the ten intermediate computation tasks, and the output task have been assigned to the various nodes in top-down, left-to-right order in the diagram of figure 2. Again, the network constructed from Y-components using a multicast mechanism yields the best performance. Also, the integrated nodes combining switching circuitry and processor on the same chip perform better than the network of separate switching components since they exhibit reduced hop count and increased port bandwidth. In the absence of multicasting, the network of 6-port switching components also suffers from a bottleneck on the link between a user processor and the communications domain.

X - TREE & Y - COMPONENTS

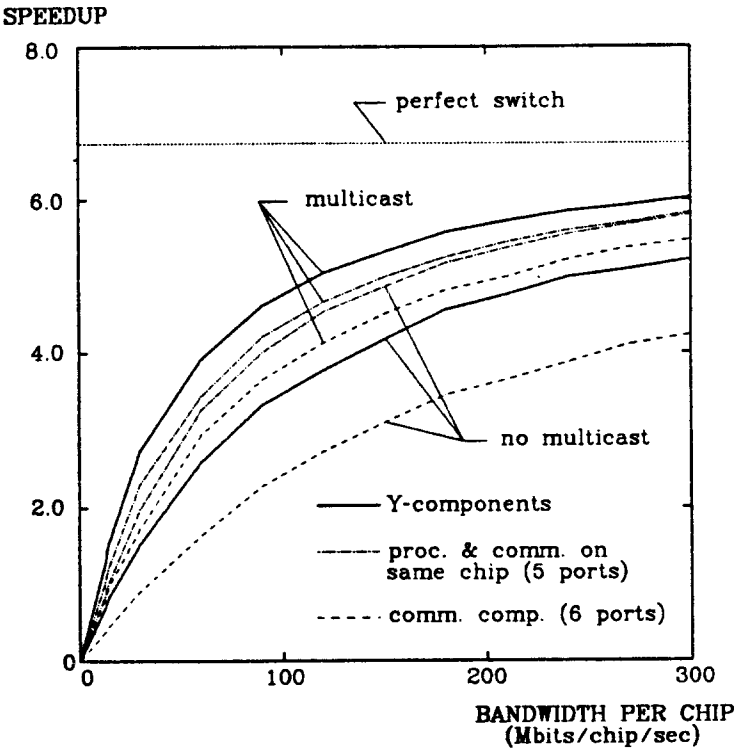

Figure 13. *Performance of Full-ring Tree Topology.*

Ring topologies are analyzed in figure 14. A separation of processor and switching circuitry automatically results in a network of Y-components; thus only four curves are shown. For ring networks, it would be particularly advantageous to have the switching circuitry on the processor chip. In addition to reducing the hop count, this would permit an increase in the port bandwidth by up to 50%, -- if one assumes that the presence of a processor does not reduce the power that can be allocated to the communications ports. In such a configuration using 150MBaud links, the execution time is almost entirely limited by the actual computation rather than by communication.

When comparing the three implementations yielding the highest performance for the three topologies, the differences are rather small. In all three cases the best performance is obtained with components with only two or three ports. The variations are due primarily to hop count differences among the topologies.

Overall the ring achieves the highest performance, while the fully connected and full-ring topologies yield similar, but somewhat lower speed-ups. The high performance of the ring relative to the other two topologies must be considered atypical. For this small multiprocessor system, it can be attributed to its use of higher-bandwidth links. Also it is a topology which is well suited for broadcast communications, and thus well matched to the needs of Barnwell's filter algorithm. As the number of processors in the system increases, however, and for applications which cannot make much use of multicasting, the performance of ring networks is limited by the linear increase in average hop count and by

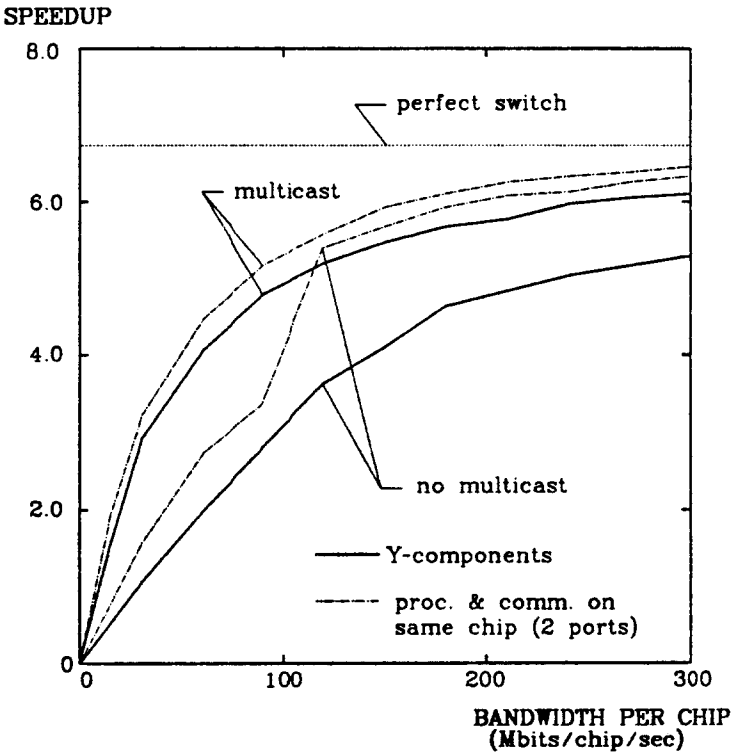

Figure 14. *Performance of Ring Topology.*

congestions on links shared by many virtual circuits.

For all networks, performance degrades dramatically as message transmission delays increase. Although delay in these simulations result from reducing the bandwidth of the communications links, it could just as easily arise from the use of software rather than hardware to store and forward packets in the communications domain, or from operating system overhead at the sending or receiving processor. Other simulations, in particular the execution of a pipelined 16-point FFT algorithm, show a distinct knee in the corresponding speedup curves. In the region below the knee, the network cannot provide the throughput required by the intertask communications, and long queues develop. Above the knee, the cost of communication is simply an additive factor representing the sum total of the net electrical transmission delays. It is clearly desirable to operate a multiprocessor system in this latter domain.

6.5. Future Work

The simulation results used as an example in this paper are for one specific application on a relatively small number of processors. Simulation studies of more complex applications, e.g. circuit simulation or speech recognition, are currently in progress. The investigation needs to be broadened to include the effect of the operating system overhead, for instance the effect of the extra paging traffic caused by the limited amount of memory available in each node.

Another area that warrants careful study is congestion control. The idea is to dynamically redirect messages over less heavily utilized links to avoid the

development of bottlenecks. Alternate paths for routing messages to destination nodes as well as means to monitor traffic and to detect congested links are necessary. The potential improvement in performance, must be weighed against the extra hardware or software complexity required by this mechanism, -- particularly in view of the constraints imposed by a single-chip VLSI implementation.

In loosely coupled networks, a great deal of attention is paid to the integrity of the transmitted packets or messages. In closely coupled networks, extremely low error rates can be expected, and the error recovery mechanism can thus be left to the end-to-end communications protocol. This reduces the complexity of the communications component significantly, and improves the effective throughput in the error-free case. The delay associated with transmitting a message is reduced significantly if forwarding can begin as soon as the data arrives. Such a scheme makes recovery from transmission errors at the link level almost impossible, since part of the erroneous packet has already been forwarded to another node when the error is detected. Hardware support should be employed in the computation processor to keep these end-to-end checks from degrading performance.

An issue that has yet to be addressed concerns fault tolerance. If a communications component fails, routing tables need to be updated, and broken message paths need to be restored. The rerouting must be done in a manner that ensures that no loops are introduced. Much of the work in rerouting strategies in computer networks is directly applicable here.[McQu74, Taji77, Sega81] One must also safeguard the network against stale messages addressed to nonexistent or unreachable nodes. These issues cannot be ignored, -- not even in first generation VLSI communications components.

7. CONCLUSIONS

VLSI technology can provide us with a novel set of building blocks for the construction of high-performance point-to-point networks for closely coupled multi-computer systems. For systems with thousands of processors, the described approach based on dedicated links between individual switching nodes is well matched to the evolving VLSI MOS technology.

"Plug-compatible" VLSI communications components with 3 to 6 ports make particularly attractive building blocks. Their modularity permits the incremental growth of a multicomputer system with a corresponding growth of the total bandwidth of the communications domain. The overall performance of the network depends critically on the total chip bandwidth of these components, which is determined to a large degree by packaging technology. It is also influenced by the buffering and forwarding policies employed, which depend themselves on the amount of buffer space and the complexity of the control logic in the switching components. Our simulation studies show the advantage of providing a multicast mechanism for applications in which the same data must be sent to many different processors.

For the near future, the limited number of devices that can be fabricated economically on a single chip will encourage the development of separate switching components. However, towards the end of this decade, the preferred

building block may well consist of a powerful processor, a substantial amount of on-chip memory, and all the switching circuitry that is needed so that these components can be readily plugged together into a working multi-computer system.

ACKNOWLEDGEMENT

The bulk of the work on project X-tree was carried out jointly with Alvin M. Despain and David A. Patterson and several graduate students. Mark Laurent and Benjamin Lee worked on the ports and links of X-tree. Roney Wong helped to analyze usage of serial links and of the Y-components. Paul Suhler and Bill Goldberg both wrote early simulators for the traffic in tree-structured networks. Tim Lu prepared the code for Barnwell's algorithm. We would also like to thank Manolis Katevenis and Yuval Tamir for a critical review of the manuscript.

Fellowship support for this work was provided by the National Science Foundation and IBM and is gratefully acknowledged.

References

Barn78. Barnwell, T.P., Gaglio, S., and Price, R.M., "A Multi-microprocessor Architecture for Digital Signal Processing," *Proc. Intl. Conf. on Parallel Processing*, (Aug. 1978).

Barn79. Barnwell, T.P., Hodges, C.J.M., and Gaglio, S., "Efficient Implementations of One and Two Dimensional Digital Signal Processing Algorithms on a Multiprocessor Architecture," *1979 Intl. Conf. on ASSP*, Washington, D.C., (April 1979).

Barn82. Barnwell, T.P., "Optimal Implementations of Recursive Signal Flow Graphs on Synchronous Multiprocessor Architectures in SSIMD Mode," Report (in preparation 1982).

Brow80. Browning, S., "Communications in a Tree Machine," Bell Laboratories Internal Memo (Jan. 1980).

Desp78. Despain, A.M. and Patterson, D.A., "X-Tree: A Tree Structured Multiprocessor Computer Architecture," *5th Annual Architecture Conference*, Palo Alto, (April 1978).

Desp80. Despain, A.M., "X-Tree: A Multiple Microcomputer System," *Proc. Spring COMPCON*, San Francisco, (Feb. 1980).

Ensl74a. Enslow, P.H., "Appendices I & J: IBM System 360 & 370," pp. 238-256 in *Multiprocessors and Parallel Processing*, John Wiley & Sons (1974).

Ensl74b. Enslow, P.H., "Appendices N, O, & P: UNIVAC 1108, 1110, and AN/UYK-7," pp. 290-327 in *Multiprocessors and Parallel Processing*, John Wiley & Sons (1974).

Fuji80. Fujimoto, R., "Routing Controller for X-Tree," Master's Report, U.C. Berkeley, (Dec. 1980).

Fuji82. Fujimoto, R.M. and Séquin, C.H., "The Impact of VLSI on Communications in Closely Coupled Multiprocessor Networks," *Proc. COMPSAC 82*, Chicago, (Nov. 1982).

Fuji83. Fujimoto, R.M., "Multiprocessor Networks Using VLSI Communications Components," Ph.D. Dissertation (in preparation 1983).

Gerl81. Gerla, M., "Routing and Flow Control," pp. 122-174 in *Protocols and Techniques for Data Communication Networks*, ed. F.F. Kuo, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1981).

Ghee82. Gheewala,, T., "The Josephson Technology," *Proc. IEEE* **70**(1) pp. 26-34 (Jan. 1982).

Gold79. Goldberg, W.S., "Communication in X-tree: A Simulator Supported Analysis," Master's Report, U.C. Berkeley, (July 1979).

Hear70. Heart, F., Kahn, R., Ornstein, S., Crother, W., and Walden, D., "The Interface Message Processor for the ARPA Computer Network," *Proc. AFIPS Joint Computer Conference* **36** pp. 551-567 (May 1970).

Kerm79. Kermani, P. and Kleinrock, L., "Virtual Cut Through: A New Computer Communication Switching Technique," *Computer Networks* **3**(4) pp. 267-286 (Sept. 1979).

Kung80. Kung, H.T. and Leiserson, C.E., "Algorithms for VLSI Processor Arrays," pp. 271-292 in *Introduction to VLSI Systems*, ed. C.A. Mead and L.A. Conway, Addison-Wesley, Reading, Mass. (1980).

Laur79. Laurent, M., "Input-Output Ports for the X-Tree Nodes," Master's Report, U.C. Berkeley, (Dec. 1979).

Lee79. Lee, S., "The Communication Link in X-Tree," Master's Report, U.C. Berkeley, (Nov. 1979).

Long80. Long, S.I., Lee, F.S., Zucca, R., Welch, B.M., and Eden, R.C., "MSI High-speed Low-power GaAs Integrated Circuits using Schottky Diode FET Logic," *IEEE Trans. Microwave Theory Tech.* **MTT-28**(5) pp. 466-472 (May 1980).

McQu74. McQuillan, J., "Adaptive Routing Algorithms for Distributed Computer Networks," NTIS Report (AD-781 467), U. S. Department of Commerce (May 1974).

Patt80. Patterson, D.A. and Séquin, C.H., "Design Considerations for Single-Chip Computers of the Future," *IEEE Trans. on Computers* **C-29**(2) pp. 108-116 (Feb. 1980). Joint Spec. Issue w. IEEE Jour. Solid-State Circuits.

Patt82. Patterson, D.A. and Séquin, C.H., "A VLSI RISC," *Computer* **15**(9) pp. 8-21 (Sept. 1982).

Pouz78. Pouzin, L. and Zimmerman, H., "A Tutorial on Protocols," *Proc. IEEE* **66**(11) pp. 1346-1370 (Nov. 1978).

Prad82. Pradhan, D.K. and Reddy, S.M., "A Fault-Tolerant Communication Architecture for Distributed Systems," *IEEE Trans. on Computers* **C-31**(9) pp. 863-870 (Sept. 1982).

Sega81. Segall, A., "Advances in Verifiable Fail-Safe Routing Procedures," *IEEE Trans. Communications* **COM-29**(4) pp. 491-497 (April 1981).

Séqu78. Séquin, C.H., Despain, A.M., and Patterson, D.A., "Communications in X-Tree, A Modular Multiprocessor System," *Conf. Proc., ACM78*, Washington D.C., (Dec. 1978).

Séqu81. Séquin, C.H. and Fujimoto, R.M., "Communication Components for Multiprocessor Networks," *Intnl. Seminar on the Teaching of Computing Science: Very Large Scale Integration*, Newcastle upon Tyne, England, (Sept. 1981).

Séqu82. Séquin, C.H. and Patterson, D.A., "Design and Implementation of RISC I," in *Proc. Advanced Course on VLSI Architecture*, Univ. of Bristol, England, ed. P.C. Treleaven, Prentice Hall, Englewood Cliffs, New Jersey (1982).

Séqu83. Séquin, C.H. and Fujimoto, R.M., "Optimum Number of Ports for Switching Components," Internal Report (in preparation 1983).

Shoc80. Shoch, J. and Hupp, J., "Measured Performance of an Ethernet Local Network," *Comm. ACM* **23**(10) pp. 711-721 (Dec. 1980).

Suhl78. Suhler, P.A., "X-SIM, A Dynamic Communications Simulator for the X-tree Network," Master's Report, U.C. Berkeley, (June 1978).

Swan77a. Swan, R.J., Bechtolsheim, A., Lai, K., and Ousterhout, J.K., "The Implementation of the CM* Multi-microprocessor," *Proc. Natnl. Computer Conf.*, Dallas, Texas, pp. 645-655 (June 1977).

Swan77b. Swan, R.J., Fuller, S.H., and Siewiorek, D.P., "CM*—A Modular, Multi-microprocessor," *Proc. Natnl. Computer Conf.*, Dallas, Texas, pp. 637-643 (June 1977).

Taji77. Tajibnapis, W., "A Correctness Proof of a Topology Maintenance Protocol for a Distributed Computer Network," *Comm. ACM* **20**(7) pp. 477-485 (July 1977).

Widd80. Widdoes, L.C., "The S-1 Project: Developing High-Performance Digital Computers," *Proc. COMPCON*, San Francisco, pp. 282-291 (Feb. 1980).

Witt81. Wittie, L.D., "Communications Structures for Large Networks of Microcomputers," *IEEE Trans. Computers* **C-30**(4) pp. 264-273 (April, 1981).

Wong81. Wong, R., "Serial Communications in X-Tree," Master's Report, U.C. Berkeley, (June, 1981).