Design of a Bit-sliced network for a shared-memory multiprocessor system CSR-19-92

D.J. Rogers & R.N. Ibbett
Computer Systems Group
Department of Computer Science
University of Edinburgh

March 1992

Abstract

Packet switching crossbar switches with matching data widths and addressing range will provide the most efficient building blocks for a shared memory multi-processor network, providing the lowest latency consistent with high data throughput and error tolerance for a low device count.

A demonstration device for a 4*4 crossbar switch with 2-bit data paths has been implemented. Results from this show that the pad-bound assumption, inherent in the original argument would be true for a full custom implementation in a sub-micron process. Simulation for the arbitration method used is given, predicting that fairness is only slightly compromised against optimal for the sake of considerable reduction in complexity, if priority is based solely on queue length.

Contents

1 Introduction

As the degree of parallelism in multiprocessor (shared memory) and multi-computer (message passing) systems continues to increase, the interconnection problem becomes increasingly severe. In multicomputer systems this problem is usually solved by using a sparse interconnection network such as the hypercube (or, more formally, the binary k-cube [9]) and routing packets through the network from source to destination in a series of hops. Such networks are not normally appropriate for shared memory systems, however, where each processor requires direct access to all of the available memory, itself typically made up of a number of discrete units. The two most obvious techniques to use in this situation are a common bus or a cross-bar switch.

The shortcomings of the common bus are self evident; the number of processors which can be connected to a bus is restricted not only by the physical and electrical properties of the bus, but also by the finite bandwidth of the bus which constrains the data transfer capacity between the processors and memory. This latter problem can be ameliorated by the use of cache memories within each processor, though this then introduces the further problem of cache coherency [14]. Systems with a few tens of processors are nevertheless cost-effective, as witnessed by the success of commercial machines such as the Sequent Balance and the Encore Multimax [16].

The simplest way to implement full connectivity between m source units and n destination units is to use a cross-bar switch The cross-bar switch is capable of realising any one-to-one, or one-to-many, set of connections and such a switch was used in one of the earliest multiprocessor systems, the 16-processor C.mmp machine started at Carnegie Mellon University in 1971 [17]. However, the hardware cost is proportional to $m.n$, and as m is normally similar in magnitude to n this equates to approximately n^2. This makes such interconnection structures impractical for highly parallel systems, where n and m are typically in the range 2^8 to 2^{16}.

However, an $N \times N$ cross-bar switch can be reduced to two $N/2 \times N/2$ cross-bar switches and two N-input exchange switches using a method devised by beneš [1]. The resulting $N/2 \times N/2$ cross-bar switches can be similarly reduced, and through this recursive trade-off between complexity and network latency, a full connection network can be produced at a significantly lower cost than a full cross-bar switch. The network shown in figure 1, for example, is constructed entirely from 2-input 2-output switch-nodes, arranged in layers and suitably connected. The particular interconnection shown is but one of a family of interconnects [13].

A crossbar based network may either be circuit switched or packet switched. Most interconnection networks used in shared memory multiprocessor systems have been circuit switched. The network used in the BBN Butterfly, for example [12], although described as being packet switched, is in fact circuit switched. When a processor makes a non-local memory request a circuit to memory is

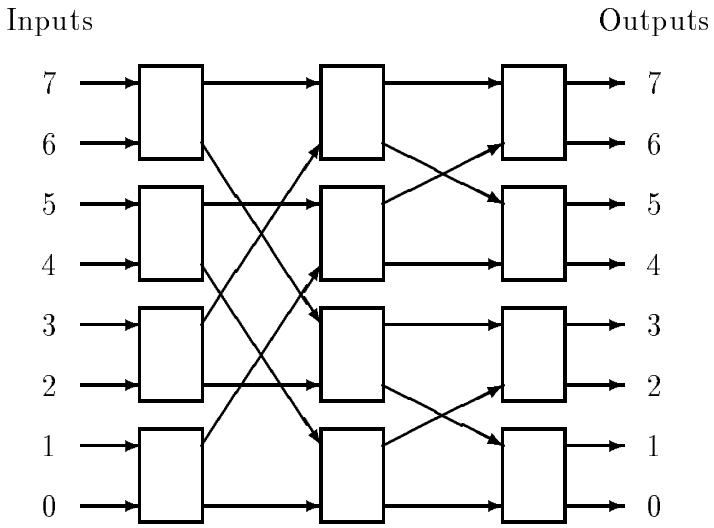

Figure 1: An 8-way 3 layer multi stage network using 2 × 2 cross-bar switches

opened through the switch and held open until the request has been satisfied. The path through the switch is, in effect, an extension of the processor bus.

An early example of a genuinely packet switched interconnection network was the MU5 Exchange [8]. This was a cross-point switch used to interconnect a small number of heterogeneous computers and their shared random access and disc backing stores. Although the majority of backing store accesses involved block transfers, the main MU5 processor could nevertheless make single-word accesses to the shared random access memory. Once such a request had been sent across the Exchange, the Exchange became available for use by other attached devices (each of which was connected as both a source and a destination) and when the memory had been accessed and was ready to respond to the request, it initiated a return packet transfer across the Exchange.

Xbar, the VLSI circuit described here, was originally designed for use in a Context Flow shared memory multiprocessor [15], but could be used in a variety of other applications. Context Flow processors use cache memory to reduce the number of external memory requests but csn also switch contexts within their pipelines when an external memory request is made. This is essential since interprocessor communication is via a synchronisation mechanism incorporated into the external shared memory, and a request may be delayed for more than one memory access time. Under these circumstances packet switching is the appropriate paradigm for the interconnection network. Processors, memories and I/O are attached to the network as both sources and destinations so that the network can provide all the connections required for a multi-processor computer.

2 Network Architecture

The network shown in figure 1 provides a single bit interconnection path between each source and destination. To transfer a w-bit packet through the network the plane must be duplicated w times thus giving low latency and high bandwidth but also increased cost, or the packet must be transferred serially through a single plane thus minimising cost but at reduced bandwidth and increased latency. Alternately some intermediate solution may be adopted in which the w-bit packet is transferred as a sequence of k w/k-bit parcels. Thus for a given network size (in terms of number of attached devices) there is a wide range of possible network architectures, and similarly a wide range of possibilities for the partitioning of the network into VLSI components. Indeed there is potential for a whole family of such components, but in order to design an integrated circuit for a specific application, its effective bandwidth in a network and its contribution to packet latency should be optimised. Although the Xbar circuit has been optimised for a particular network configuration, it could nevertheless be used in other configurations or network topologies.

2.1 Bit-slicing the network

The efficient implementation of silicon requires that an individual IC takes a data width d for a given pin count p and number of ways m. To provide support for a D-bit wide parcel the network must be made up of D/d bit-slices. Each bit-slice must transfer its packet element to the correct address. This may be achieved either by having a master slice providing global address information or for each bit-slice to be autonomous, with each packet element containing its full destination address. The former mechanism creates a single point of failure mechanism for the network, and it can be shown that there is no significant difference between the two mechanisms on pin utilisation. The Xbar circuit has therefore been optimised for implementation in autonomous bit-sliced configurations.

Having chosen to implement the network in this way it is necessary to consider the mechanism for reassembling the packet elements. For all of the bit-sliced elements of the original packet to be reassembled, either each element must carry packet identification or each bit slice must be deterministic so that all elements arrive together to be recombined. Clearly the addition of a unique packet identifier to each element would increase the packet size dramatically, and create considerable problems in the recombining logic. For the bit-slices to be deterministic they must be fully synchronous, being reset to the same state and sharing a common clock.

Providing a distributed clock to all devices with an acceptable maximum clock skew is a non-trivial task in a large network operating at upwards of $50MHz$. The uni-directional nature of the network aids this though, as the clock can propagate with the data from one layer to the next. Matching wire lengths and

using transmission line techniques is required if there is any significant distance between the cross-bar components in the network. There is a problem with the returning overflow handshake line, but this can be delayed through latches to provide correct synchronisation.

2.2 Fault tolerance

For a network of the size for which Xbar is intended, fault tolerance is a requirement. Mechanisms must be provided to detect and correct errors, as well as to allow the network to function with at least one faulty device. To provide error free data transfers within a message passing network normally requires checksums, acknowledgement protocols and time-out mechanisms to guarantee correct packet transfer. In order that faulty devices may be routed around, redundancy, usually using extra layers in the network to provide multiple paths between each node, is required. However, the autonomous mechanism of the bit-sliced network used by Xbar is readily exploitable to provide automatic error detection and correction across the packet elements. This obviates the need for a communication protocol mechanism. Also the additional latency introduced by the presence of additional layers in the network, required to provide redundant paths, will be obviated.

If an additional layer is provided to contain the exclusive-or checksum of all the bit-slice layers, then if any one layer is known to be faulty, its contents can be recovered. By also providing a one bit error detection and correction code, for each bit of the bit-slice, either a one-bit fault or an additional faulty bit-slice can be recovered.

Thus using Hamming code checking with the additional exclusive checksum across the bit-slices, one non-operable plane can be tolerated along with errors affecting one or more bits in any other bit-slice. The overhead for this is dependent only on the number of planes utilised, which is in turn a function of the data word width D.

Overhead cost of fault and error tolerance				
Checking bit-slices	3	4	5	6
Maximum data bit-slices	1	4	11	26
Minimum overhead	300%	100%	45%	23%

The table above illustrates how, as the number of bit-slices used to carry data increases, so the percentage of additional bit-slices required to provide the checksum is reduced. A network of 4-bit Xbar devices supporting between 32 and 44-bit wide data packets would incur an overhead of about 50%.

3 Network protocol

There has to be a strategy for handling the case where two or more packets contend for the same output within a single crossbar switch. There are four possible mechanisms:

- Additional packets are lost and the acknowledgement protocol causes them to be resent.
- All but one of the packets are redirected and retransmitted on arrival at the wrong address.
- Transmission is halted from previous devices for all but the successful packet.
- Packets are stored in queues until the output becomes available.

Loosing packets is both inefficient and liable to excessive loss of bandwidth of the system. Unless acknowledgement messages are given a higher priority than other packets, the network will tend to saturate suddenly with relatively low traffic densities, due to retransmissions. The only advantage is that the transmission time is predictable and therefore the time-out requirement for the acknowledgement protocol mechanism can be short. If fault tolerance is to be provided through the bit-slice mechanism described above, then this mechanism would not work.

Re-direction of packets relies on every possible destination address having an active device attached to it, to receive and re-transmit the mis-directed packet. This implies that if any of these are faulty, packets will be lost. Therefore an acknowledgement protocol must be used, or the mechanism for re-transmission of a packet must be provided within each bit-slice.

Stopping the transmission of data requires that a blocking signal can propagate back to the source before the next parcel is transmitted. This prevents the use of pipelining to maximise the data throughput for a large network.

Within Xbar we have chosen to use on-chip queues in order that a pipelined scheme may be used with high data throughput, and so that the bit-slice fault tolerant mechanism may be used. In this way packets are queued as they arrive, preventing data loss. Queues are finite in length though, so to prevent overflow and thus data loss, under extreme conditions it must be possible to halt transmission from a previous device. This requires a handshake line travelling in the opposite direction to the data indicating that the queue is above some predetermined level.

3.1 Handshake lines

In order that packets are identified, at least one handshake line must accompany the data, to indicate that its value is currently valid. Some method is required to indicate unambiguously the start of any packet. In order to achieve this,

either the packets must be separated by one or more cycles of null data, or the handshake line must go low for the final clock cycle of the packet transmission. This mechanism is illustrated in figure 2. One consequence of this is that packets must occupy at least two clock cycles in order for the handshake line to identify a valid packet.

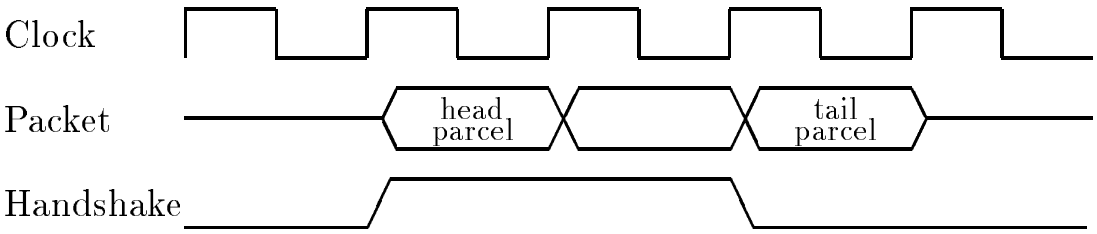

Figure 2: Packet and handshake line operation for Xbar

A second consequence is that a packet may not be halted in mid-transmission. The major effect of this is that a crossbar switch must request a halt in transmission while there is still sufficient queue space to handle the maximum packet size. Large packets can block the network for critical small packets, so the imposition of a maximum packet size is acceptable. The effective queue length is severely reduced by the need to prevent transmissions.

As the optimisation for Xbar is based on external pin count, a single handshake line in each direction is required. Maximum packet length has therefore to be defined prior to implementation.

3.2 Address header

Because the network is divided into fully autonomous bit-slices, while the data may be spread over all planes, the address must be replicated, as shown in figure 3. This address forms a substantial part of a packet, increasing latency and reducing the effective bandwidth. By removing that part of the address that has already been used, the packet shrinks as it goes through the network. This reduces the traffic level and hence reduces conflict induced latency.

4 Optimisation

In VLSI terms Xbar is an I/O intensive device and it was reasonable to assume that the design would be pad bound, i.e. the number of input and output connections would determine the area of silicon required for its implementation. Optimisation has therefore been based solely on pin count, and the area of silicon defined by the pad ring is utilised to provide the best overall performance for a general purpose network.

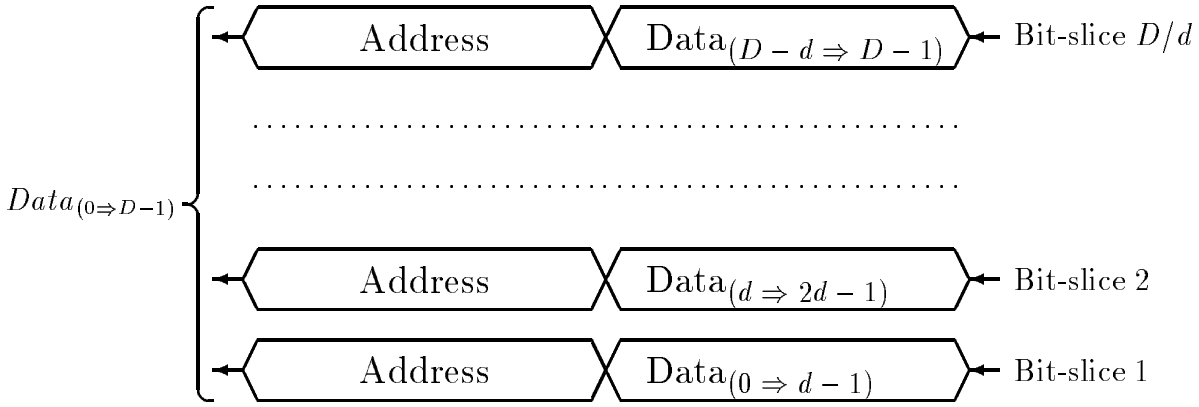

Figure 3: Bit-slicing the packet

The network size, in terms of number of attached devices and data width, is taken to be 256 devices each with 32 data bits. In practice, apart from the overhead of providing fault tolerance, neither of these factors affects the optimisation which has been based on a simple premise that the bandwidth and latency between an input and output device on an otherwise unused network is adequate.

In optimising the design there are three overheads that reduce the effectiveness of the IC in a network. These are the inter-IC handshake lines, the packet address header, and the redundant planes for error detection and correction.

All effects can be expressed as functions of the number of pins p available for data input and output. To this number must be added the control clock and test connections, along with power and ground connections. Each IC in the family will then be configured as an m-way crossbar switch, each way having d data bits and h handshake lines so:-

$$p = 2m(d + h) \tag{1}$$

If only the overhead of the handshake lines is considered, then the maximum bandwidth B_c, provided per chip within a network with n ports, where each IC has p pins and provides an m-way crossbar switch is:-

$$B_c = \frac{d * n}{n/m * \log_m(n)} \tag{2}$$

Substituting for d from equation 1 in equation 2, substituting $h = 2$ to take account of the assumption that Xbar would use a 2 wire handshake, and simplifying:-

$$B_c = \frac{(p - 4\,m)\ln(m)}{2\,\ln(n)} \tag{3}$$

This function has a minimum which is fortuitously not dependent on the size of the network n. As there is no analytic solution for p in terms of m, this is achieved by differentiating equation 3 with respect to m and solving for p.

$$p = 4\,m\,(\ln(m) + 1) \tag{4}$$

This gives an optimum value of crossbar ways for any pin count in order to minimise the number of chips required to provide the desired data bandwidth.

The latency caused by the transit time through the network is the product of the number of layers in the network and the transit time through each layer. Therefore, provided the transit time of a crossbar switch is independent of its size, the more ways it has, the fewer layers are required and thus the lower is the transit induced latency. This simple relationship is distorted, however, by the additional delay introduced by the inclusion of the address header that has to prefix the packet information.

The transit time T expressed as clock cycles through an IC includes the clock cycle required for the information to be read in and 1 or more clock cycles to pass through. Thus it takes a minimum of $T\log_m(n)$ clock cycles for data to pass through the complete network. Additional delays do not vary with configuration and so can be ignored. The address header occupies $\log_2(n)/d$ words where d is the data width of the IC and will take this number of cycles. Therefore the total network latency L_n caused by the minimum delay and addressing overhead, substituting for d from equation 1 and simplifying is:-

$$L_n = \frac{T\,\ln(n)}{\ln(m)} + \frac{2\,m\,\ln(n)}{\ln(2)\,(p - 4\,m)} \tag{5}$$

Minimising equation 5 for L_n with respect to m, again the total size of the network becomes unimportant and has only a solution for p in terms of m.

$$p = \frac{m\left(8\,T\ln(2) + 2\,\ln(m)^2 + 2\ln(m)\sqrt{8\,T\ln(2) + \ln(m)^2}\right)}{2\,T\ln(2)} \tag{6}$$

Equations 4 and 6 are plotted in figure 4 to show the maximum throughput and minimum latency for a transit time T of 2 and 3 cycles. An increase in the crossbar transit time clearly shows an increase in the optimum number of ways for a given number of pins. Also the optimum configuration to maximise throughput or minimise latency is similar.

These graphs are based on the assumption that the equations are strictly linear, whereas in fact they are subject to quantisation. This is done in order to illustrate more clearly a comparison between the requirements. Also these values do not take into account the overhead of the packet address size or error checking overhead on the maximum data throughput. No account is taken either, of packet collision on minimum latency.

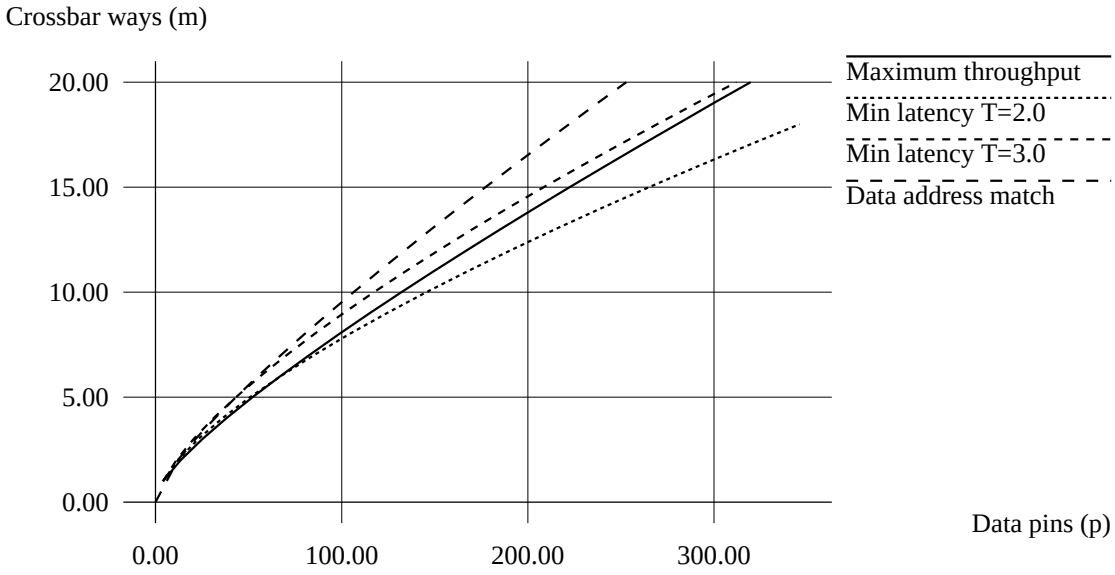

Figure 4: Relation between crossbar ways and data pins for various suppositions

4.1 Practical implementations

A number of network configurations will invoke no loss of overall efficiency due to quantisation. Of these there is a particularly interesting subset where the address required for each crossbar switch exactly matches the number of bits associated with its data path (eqn 7).

$$d = \log_2(m) \tag{7}$$

This relationship is plotted in figure 4 to show the degree of matching with the optimum cases for minimum latency and maximum bandwidth. There are 5 possible implementations for ICs with less than 500 pins with this configuration. These have 1, 2, 3, 4 and 5 data bits providing 2, 4, 8, 16 and 32 way crossbar switch capability respectively. While these points are near to optimal for data throughput, they appear to provide lower than optimal latency. This is slightly misleading, however, as the ability to use the first word of a packet solely for internal addressing allows the IC to have a lower latency than could be achieved if this word had to be re-transmitted. The special case where the address size and word width are equal can therefore can be shown to provide both the maximum data throughput and the minimum overall latency.

Furthermore, solutions other than these constrain the design of the network in some way if optimal performance is to be achieved. Because of the near optimal capability of networks based on ICs with these configurations as well as their appropriateness for integration into bit-sliced networks, only designs conforming

to this simple relationship were considered for practical implementation.

Xbar Ways	device Pins	data bits	bandwidth for k=0			bandwidth for k=4			relative latency
			relative	per pin	per pin^2	relative	per pin	per pin^2	
2	12	1	1	1	1	0.25	0.25	0.25	1
4	32	2	8	3	1.13	2.67	1	0.38	0.5
8	80	3	36	5.4	0.81	15.4	2.3	0.35	0.33
16	192	4	128	8	0.5	64	4	0.25	0.25
32	448	5	400	10.7	0.29	222.2	6.0	0.16	0.2

Table 1: Optimal crossbar switch configurations

Table 1 lists the values for the four previously mentioned configurations and a fifth 2-way device for comparison. We can note from this that as the number of pins utilised rises, the equivalent bandwidth rises dramatically and the latency falls significantly. If cost is associated with the area of either the chip or the package, which is roughly proportional to the square of the number of pins in both cases, we see a slight decline in device efficiency for long packets (k=0). The improvement in latency is also not significant compared to the packet transmission time for this case. If the network is to be used for memory access however, packets will be short and the addressing cost may rise as high as 4 for critical cases, in which case the cost remains roughly constant with pin^2 and the latency improvement becomes critical to device efficiency.

5 The Xbar protocol

The sum of all the previously mentioned decisions is a protocol that can be implemented in silicon. Central to that protocol is the data-address size matching coupled to the 2 wire handshake. Latency is minimised in the network by packets passing through a switch with minimum delay, only being stored if there is contention for the output resource. This set of requirements simplifies the packet structure, with one extra address parcel added for each layer of the network. This parcel can be stripped from the packet on transfer, increasing the effective minimum delay per layer from 2 cycles, 1 for transferring between devices, and 1 for passing through it, to 3 including the address transmission time..

The timing for a minimum delay case using this protocol is shown in figure 5. The incoming packet contains 2 or more data parcels preceded by an address parcel. On entering the device, the header address parcel is used and discarded. All following parcels are routed directly to the output unless there is contention, with a total delay of 3 clock cycles per layer.

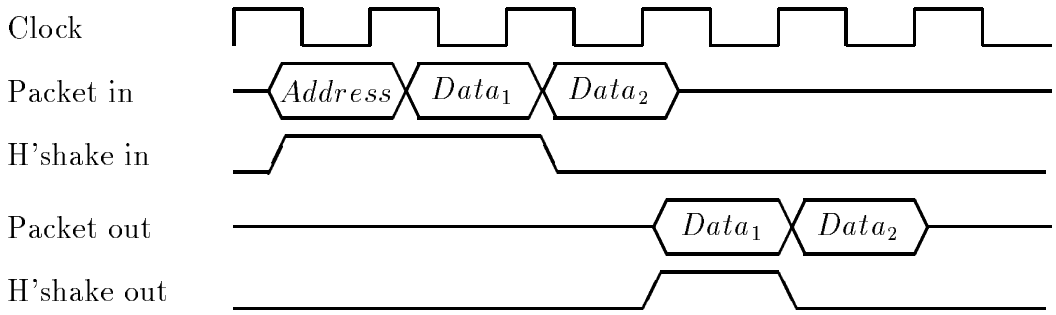

Figure 5: Packet timing for demonstration device

6 Queues

The main call on the area of silicon bounded by the pad ring is the provision of queues. These have been chosen as the main mechanism for handling contention between packets seeking the same resource. Queues are normally implemented with a circular buffer, consisting of random access memory (RAM) that can be read from and written to concurrently. Counters are used to point to the head and tail of the queue, with some logic to detect impending overflow.

The area taken by the RAM is roughly proportional to the product of the data and address size. The data width will usually include at least one extra bit to identify separate packets. The address size will normally equal the effective length of the queue, the size of which must be sufficient to prevent excessive network blocking under normal circumstances. Kumar and Jump [7] have shown that buffers need to hold about 4 packets to achieve throughputs of over 80% of the theoretical limit. This needs to be raised to 8 to achieve 90% throughput. The queue must also have sufficient size after signalling an impending overflow to absorb any incoming information, before the signal prevents further information transfer without the queue overflowing.

6.1 Queue position

Queues can be positioned between switches, thus buffering each data path, or within each switch buffering each input-output pair. Clearly, for each crossbar switch, only m queues will be required for the first case but m^2 for the second case. The probability P_{max} of transmitting a packet with external queues in a fully loaded network with random traffic is calculable using the relationship between the probability of input and output (eqn 8).

$$P_{max} = 1 - (1 - 1/m)^m \tag{8}$$

As $m \to \infty$ this simplifies to $(1 - e^{-1})$. So for a network of crossbar switches with queues between switches, the maximum data throughput for random traffic, assuming no queue overflow is given below. With internal queues, however, and

Maximum network utilisation with external queues				
Crossbar size m	2	4	16	∞
Output probability P_{max}	75%	68%	64%	63%

infinite queue length then, $P_{max} = 1$. Thus internal queues are to be preferred, as with a large number of ways the throughput is otherwise significantly reduced.

Providing both internal and external queues may be an efficient mechanism for use in Xbar as one side effect of the 2-wire handshake protocol is that a packet cannot be interrupted in the middle of transmission between switches. It is therefore necessary to store one complete packet after a queue has indicated an overflow condition. As the n queues associated with each input each have to reserve this area, which would normally be a significant part of that memory and associated control circuit. By implementing separate overflow queues associated with each input and allowing more control signals between the demultiplexer and queues, however, this overhead could be concentrated into a single resource. Speed and latency need not be compromised as the overflow queue would be bypassed most of the time. This feature would improve effective queue length for systems with a large number of crossbar ways.

6.2 Priority for internal queues

When a number of queues are competing for the same resource, an arbiter must select one of them to gain control in order to output one packet. This requires the queues to present to the arbitor a priority value. The mechanism employed does not affect the mean latency of a packet but only the fairness of handling each of the packets. Fairness may be judged using the standard deviation of delay. To minimise this, the oldest packet must be transmitted first, using first-come first-served arbitration. If queue overflows are to be minimised, then it is more important to prioritise on queue length, using fullest queue first arbitration. In a synchronous circuit, more than one packet can have the same priority, so if one route is not to be favoured, round robin arbitration, where the queues compete with changing rules of precedence must be used.

To support first-come first-served arbitration, the queues must provide the time at which a packet at its head entered the queue. To do this the time associated with the packet must be stored, normally alongside the data in the RAM. This requires several bits of information to prevent overflow creating ambiguity. Implementing fullest queue first arbitration requires only logic to subtract the value of the head counter from the tail counter. Implementing round robin arbitration only requires the queue to signal that it has a packet waiting for transmission, thus simplifying the logic and speeding up arbitration. Such a

mechanism is not efficient however, not only because it creates unfairness, but because the use of the queues is uneven, requiring larger RAMs to contain them.

7 Demonstration device

A demonstration packet switching crossbar switch with internal queues was designed using ES2 SOLO1400 software for fabrication with the complimentary ES2 $1.5\mu m$ process. This was designed as a final year undergraduate project [5], specification details for which are given in table 2. Due to the limitations of silicon availability, the design was limited to a 4*4 arrangement with 16 internal 32-word long queues. Longest queue arbitration was chosen as easiest to implement, reverting to round robin when arbitration levels are equal. The round robin counter is resettable to force deterministic behaviour for the device. In order to prevent the delay through the arbitrator limiting the operating speed of the device, the 5-bit queue length and 1-bit active loading information had to be compressed to 3 bits. Also, in order to support different packet lengths, the queue overflow position is selectable at 16 or 24.

The queues were constructed using a generated dual port RAM with 5-bit addresses. These are synchronous in operation and require the address to be presented prior to the clock edge that reads or writes data. In order for data to pass through this RAM within one clock cycle it is necessary to write on the rising clock edge and read on the following falling edge, which creates a timing bottleneck if data is to pass through within one clock cycle. In order to maximise the operating frequency of the IC it was therefore necessary to allow data to appear on the falling clock edge. This mode is settable by a logical input. In this mode the delay through a layer is limited to 2.5 clock cycles rather than the 2 clock-cycle theoretical minimum that is otherwise achieved. The device simulation will operate with typical delay parameters at a clock frequency of $31MHz$.

The output pads used under normal operating conditions can be expected to sink or source $60mA$. This is adequate for transmission line driving. The inputs are all CMOS compatible so the noise margin is large. Because of the presence of high power drivers, it is necessary for there to be a large number of power and ground pads to prevent power starvation.

The active area used by the queues, prioritisor and all associated logic and routing occupied $41.2mm^2$. The pads used would have required a minimum circumference of $9.2mm$ so a pad bound design would have had $5.4mm^2$ of silicon available for this circuitry. It is known that the ES2 $1.6\mu m$ process is not area efficient, as the metal 1 to metal 2 via size is larger than would be expected with transistors of this size. By using the $1.2\mu m$ process we could expect therefore an area reduction of 50%. Further reductions would be possible by using more powerful design tools or the use of finer geometry design rules. It is likely therefore

Detail	Specification
Number of ways	4 by 4
Data width	2 bits
Handshake lines	2 (1 in 1 out)
Bond pads	20 input, 16 output, 8 power, 8 ground
Address handling	Prefix address removed in handling
Queue position	internal (16 total)
Queue length	32 with overflow warning on 24 or 16 (selectable)
Clocking	Synchronous data in on rising edge, output selectable
Minimum latency	2 or 2.5 cycles (dependent on output timing)
Prioritisation	longest queue with compression
Silicon area	Die size $60.99mm^2$, array area $41.29mm^2$

Table 2: Demonstration device condensed specification

that this design could be made close to pad bound using current technology.

7.1 Simulation

A register level simulation of the design was written to allow observation of alterations to the arbitration protocol and the significance of queue length. The traffic generator used created constant length packets with a fixed probability of transmission in every available clock cycle, each packet with a randomly generated address.

Results are given in table 3 showing operation at various traffic densities for the implemented longest queue priority scheme, a first come first served priority mechanism and predicted results based on queueing theory. This was derived from the equation given by Kruskal and Snir in their article [6] and is given in equation 9 where m is the number of ways, in this case 4, k is the packet length and p_{out} is the input probability.

$$delay = \frac{(1-1/m)P_{in} * (k-1)^2}{2l(1-P_{in}(k-1)/k)} \; cycles \tag{9}$$

The difference between theory and practice for mean packet delay is probably caused by the approximation made in the above equation in assuming that packets arrive on the same clock cycle. The correlation is nevertheless acceptable for use in design studies.

Providing overflow does not occur, the mean packet delay and mean queue length are independent of arbitration mechanism. What is noticeable is that the standard deviation and worst case delay is slightly better using the first-come first-served prioritisation while the implemented longest queue arbitration gives

Theoretical prediction if no overflow occurs												
words per packet	4 in 3 out				6 in 5 out				8 in 7 out			
Busyness factor	25%	50%	75%	99%	25%	50%	75%	99%	25%	50%	75%	99%
mean packet delay	2.26	2.67	3.45	5.24	2.49	3.34	5.12	10.84	2.73	4.04	7.01	19.00
Longest queue priority scheme												
words per packet	4 in 3 out				6 in 5 out				8 in 7 out			
Input prob. P_{in}	25%	50%	75%	99%	25%	50%	75%	99%	25%	50%	75%	99%
mean packet delay	2.26	2.60	3.36	4.71	2.51	3.19	4.96	9.38	2.83	3.82	6.66	n/a
standard deviation	0.77	1.26	2.41	3.94	1.42	2.29	4.80	10.1	2.49	3.41	7.10	n/a
worst case delay	10	13	44	46	18	24	57	113	51	37	91	n/a
mean queue length	0.15	0.36	0.71	1.31	0.20	0.50	1.20	3.03	0.24	0.64	1.73	n/a
standard deviation	0.55	0.90	1.50	2.40	0.72	1.32	2.66	5.59	0.97	1.78	3.87	n/a
worst case length	3	6	9	9	10	10	11	20	11	14	20	n/a
First come first served priority scheme												
words per packet	4 in 3 out				6 in 5 out				8 in 7 out			
Input probability P_{in}	25%	50%	75%	99%	25%	50%	75%	99%	25%	50%	75%	99%
mean packet delay	2.26	2.60	3.35	4.71	2.51	3.19	4.97	9.40	2.83	3.82	6.67	n/a
standard deviation	0.75	1.21	2.19	3.37	1.41	2.20	4.35	8.32	2.26	3.35	6.47	n/a
worst case delay	9	13	29	30	16	22	37	69	26	32	50	n/a
mean queue length	0.15	0.36	0.70	1.31	0.20	0.50	1.20	3.03	0.24	0.64	1.73	n/a
standard deviation	0.55	0.89	1.46	2.29	0.72	1.30	2.56	5.18	0.94	1.77	3.72	n/a
worst case length	3	5	12	13	10	10	15	27	10	14	21	n/a

Table 3: Simulation results for 4*4 crossbar switch

shorter queue lengths. This result is consistent with common sense, though a larger difference might have been expected.

Results are not given for the worst case of 99% utilisation with 8-word packets as this causes queue overflow and such results are of limited value. Other results show clearly that at relatively high busyness factors of 50% or 75%, the delays due to collision are only of the same order as for the delay through the switch. Having queues which are 32 words long should also not compromise performance unless the network is fully loaded with long packets.

8 Very large Xbar devices

Crossbar switches with 16 or more ways are extremely attractive, as they offer the possibility of providing networks with only one layer and massive ones with 2. This will reduce the latency dramatically for such designs, and provide a natural segmentation for modular construction.

Present packaging technology would support the 448 I/O pins required for a 32 way device but not yet the 1024 I/O pins for a 64 way device. Providing the bond pads on the silicon device would also be a problem, as if a limit of 15mm square is taken for the silicon chip, then each pad will need to be less than $50\mu m$

wide, compared to the current minimum of about $100\mu m$. Future developments in silicon and packaging technology are likely to allow this, so it is reasonable to consider the design of Xbar devices with up to 64 ways.

8.1 Internal queue utilisation

As the number of inputs competing for each output rises, the probability that any one of them will require that output reduces. Implementing one queue for each input-output pair is therefore inefficient, requiring space to be allocated that will normally not be used. Statistical analysis is helpful in understanding the nature of the requirement, although it assumes random addressing and unitary length packets.

Given the input probability P_{in} of a packet arriving at any of the m inputs of a crossbar switch, then for any of the m outputs of that switch, the probability P_{out} of i packets arriving concurrently may be found. This is given in equation 10 where the vector form is used to represent the binomial function. Finding the probability that a given number of queues would or would not be adequate can then be found through summation.

$$P_{out} = \left(\frac{P_{in}}{m}\right)^{i} \left(1 - \frac{P_{in}}{m}\right)^{m-i} \begin{pmatrix} m \\ i \end{pmatrix} \tag{10}$$

As $m \to \infty$ then equation 10 may be simplified. As $(m - k) \to m$ where k is any non infinite value, the binomial function simplifies to $m^i/i!$ while the second term of equation 10 expands to the exponential series $e^{P_{in}}$. The final form for this is given in equation 11.

$$P_{out} = \frac{(P_{in})^i e^{-P_{out}}}{i!} \tag{11}$$

Table 4 lists the probability that more than a given number of packets would be competing for a given output, and hence how often that number of queues would be inadequate to handle the incoming data, for both a range of crossbar switch sizes with maximum input probability, and a range of input probabilities for an infinitely large crossbar switch.

queues would be required for each output for a selection of different crossbar ways is given, for maximum traffic conditions, and for different input traffic densities for an infinitely large crossbar switch. This assumes that packets are entering all the inputs simultaneously. These figures would suggest that there is no point in providing more than 5 queues for each output, if they can be used as a shared resource by all the inputs and when the traffic density is only 50% of peak, the demand for 3 or more queues is reduced to below 0.2% for all cases, suggesting that 2 or 3 queues would be sufficient for most practical implementations.

An alternative approach to this problem is to look at the use of multiple input queues to provide buffering of the blocked packet in such a way as to allow the

crossbar	Input probability = 1.0					
ways	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
2*2	75.0%	25.0%	N/A	N/A	N/A	N/A
4*4	68.4%	26.2%	5.1%	0.4 %	N/A	N/A
8*8	65.6%	26.4%	6.7%	1.1%	0.1%	0.0%
16*16	64.4%	26.4%	7.4%	1.5%	0.2%	0.0%
32*32	63.8%	26.4%	7.7%	1.7%	0.3%	0.0%
$\infty * \infty$	63.2%	26.4%	8.0%	1.9%	0.4%	0.1%
Input	Crossbar switch size = $\infty * \infty$					
probability	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
0.2	18.1%	1.8%	0.1%	0.0%	0.0%	0.0%
0.4	33.0%	6.2%	0.8%	0.1%	0.0%	0.0%
0.6	45.1%	12.2%	2.3%	0.3%	0.0%	0.0%
0.8	55.1%	19.1%	4.7%	0.9%	0.1%	0.0%
1.0	63.2%	26.4%	8.0%	1.9%	0.4%	0.1%

Table 4: Probability that k output queues would not be adequate

following packet to overtake. While the statistical analysis of such a scheme is difficult to evaluate exactly, an acceptable approximation for the case where a single packet can be held at the input without causing blocking is close to the cube of the probability of an output not being available. Therefore for the $\infty * \infty$ case with 2 queues and input packet buffering, the probability of a blockage should be below 0.1%. Simulation of an actual system is required to evaluate this further.

Because of the need for input overflow queues to buffer the rare collision cases, and the additional level of arbitration for an input to gain control of a queue, sharing queues would only be worthwhile for crossbar switches with 16 ways or above.

8.2 Possible configurations

A working design will require between 2 and 5 shared output queues as each output will require at least 2 queues to improve upon the external queue limit, and 5 will prevent blocking almost all of the time. Each input requires access to all of the available queues either through the use of multiple input busses or de-multiplexers. There is a finite probability that a packet will be blocked on entering the switch, requiring it to be buffered at the input.

Figure 6 shows the arrangement for a 4*4 way device with a single input queue per input connected to 2 queues per output via an 4*8 crossbar switch. Arbitration is required to select the route for incoming packets to an appropriate output queue, with the additional task of blocking un-routable packets, causing

them to be queued at the input. This is in addition to the arbitration required to select between output queues competing for the output.

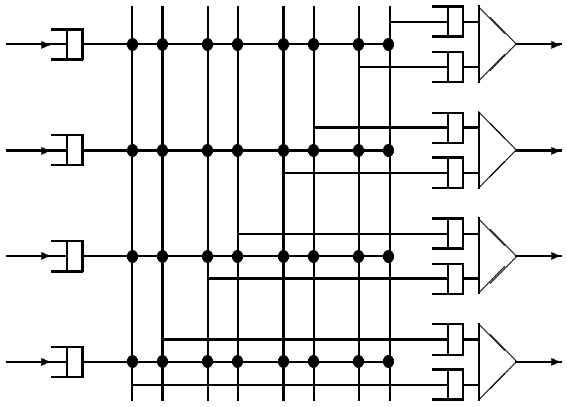

Figure 6: Possible shared queue configuration

The first level of arbitration needs to resolve between a large number of competitors for a multiple resource. However the nature of the arbitration is not thought to be significant so the decision time can be kept short. The second level arbitration in contrast needs to arbitrate between only a small number of competitors. The probable requirement of a strict first come first served protocol will though require a more complex arbiter design. While it should be possible to perform both arbitrations within one clock cycle, it may be difficult to maintain the minimum latency that has been achieved for the 4*4 prototype device.

8.3 Prioritisation with shared queues

This network provides a unique route between each device, simplifying the protocols to control it. Equally, the strict pipelining would allow the reduction of write latency for shared data in a cache-based multi processor system [2]. The presence of shared queues requires multiple routes within the IC allowing packets to overtake each other. As this would complicate the protocols and therefore compromise the performance of this network for use when supporting shared caching, it is necessary to prevent this.

There are several mechanisms that can be used to prevent this where there are shared output queues, but all of them compromise the effectiveness of a shared queue implementation except using strict first come first served arbitration on the output of the queues. If multiple inputs are also used then further protection will also be required to prevent overtaking at that point by not allowing a packet to one address to be stalled behind one to another address. If more than one packet is contending for the same set of output queues at the same time, either

the age of a packet must be transferred with it, or only the oldest packet must be allowed to succeed. This second solution simplifies the arbitration at both levels.

The provision of first come first served prioritisation requires a counter that is incremented on every clock cycle that a new packet enters one of the queues. In this way, the counter size can be restricted, reducing the overhead for each queue. If there is no filling order for the queues then the maximum counter value required would equal $(k-1) * l/P_{min}$ For the case where the number of queues $k = 5$, their length $l = 64$ and the minimum packet length $P_{min} = 2$ this is 128, requiring 7 bits. Most practical input arbitration mechanisms reduce this further, limiting the overhead to between 4 and 6 bits for most cases.

9 Additional features

Several features could be included in the crossbar switch to improve the overall network efficiency. These mechanisms compete with the queues for space, so any improvement must be demonstrably more useful than the displaced queue length or any degradation in latency or speed. Deciding between these features can often only be done by empirical argument or simulation using typical traffic.

9.1 Priority mechanism

There are two main mechanisms for priority handling, first come first served, and longest queue. It is generally recognised that first come first served provides the fairest mechanism, while longest queue minimises the probability of queue overflow. Both mechanisms are liable to show equal priority to two or more contenders at the same time. Order of preference should then be on a past service basis or pseudo-random. The implementation of first come first served requires each packet to be order stamped on entering its queue. To implement this can double the area of silicon used for the queues. Queue length however can be derived directly from the head and tail counters used to implement the queue.

The speed and size of the prioritisor is a function of the number of priority levels for each contender. In practice it is only necessary to differentiate finely between short queues; this also tends to reflect their time of arrival as the data is still entering the queue as the decision is made. Fairness may also be better served for long queues by randomness, although queues approaching overflow must be given access first anyway to prevent the network stalling.

9.2 Broadcasting and packet combining

The need to broadcast to all (or a subset of) addresses could prove useful, especially for network management tasks. In order that the overhead incurred by

non-broadcast packets is not too great, the only practical mechanism for implementing a general broadcast is for one address to be designated for broadcast, reducing a 16-way crossbar switch to 15 etc. It is thought that providing mechanisms for transmitting information to all devices would provide a more practical solution for most cases. All limited broadcast protocols would not reduce the traffic on the network except for very long packets. Broadcast was therefore not implemented in the demonstrator device.

Hot-spots have been shown to seriously degrade the performance of networks of this type [11] and message combining has been used as a mechanism to overcome this in both the RP3 [10] for its low speed network and the NYU-ultracomputer [4]. To implement message combining, the crossbar switch must identify identical packets un-ambiguously. However bit-slicing the network prevents this as identical bit-sliced parts do not necessarily indicate identical packets. If hot-spots are caused by spin-lock within a processor, then this can be resolved by the use of caches with a cache consistency protocol. This ensures that a processor spins on its own cache, generating no network traffic beyond fetching the cache block, and the subsequent cache update. The performance of this network may therefore depend on using a suitable cache-consistency protocol.

9.3 Packet format

The 2-wire handshake protocol demands that for abutting packets, a minimum size of 2 words be allowed. It may be desirable to transmit 1-word packets without incurring the bandwidth overhead of a dummy word. This problem only occurs for data leaving the network, as prior to that the address header adds extra words. This could be obviated by automatic padding with an easily identified dummy word by the crossbar switch on detecting a packet stripped to a single word.

While that part of the destination address that applies to the current crossbar switch is removed from the front of the packet, this could be replaced by the source address, either in its place or more likely appended to the packet. This would allow easy configuration of the network and simplify the return of a packet. This would also allow single word packets to be sent. However this would increase latency, as now a packet would not shorten as it passed each crossbar switch, and the density of traffic would not reduce as would happen with a policy of simple address stripping.

This feature is not essential since if the return address is placed within the body of the packet, it will place far less of a burden on the network. Discovering the network topology at start up would prove more complex, but not impossible, and should not pose any serious problems.

10 Summary and Conclusion

In this paper we have shown that a message passing crossbar switch designed to be used in a bit-sliced network may be efficiently implemented. A 4*4 version of this switch has been fabricated and simulations show that the design is appropriate for normal use [3].

Significantly better performance could be obtained using a 16 or 32 way crossbar. Such a device would benefit from a reduction of the number of queues from n^2 using some form of sharing mechanism. Additional features, to improve the overall performance of the network, could also be added, provided they can be shown to be demonstrably more useful than the displaced queue length.

Networks formed from Xbar devices would be suitable for replacing bus-based networks in multiprocessor systems where an increased data capacity and larger number of interconnected devices are required.

References

[1] V. Beneš. Optimal Rearrangeable Multistage Connecting Networks. Bell System Technical Journal, 43(4):1646–1656, 1964.

[2] Frederik Dahlgren and Per Stenström. Reducing Write Latencies for Shared Data in a Multiprocessor with a Multistage Network. In Proceedings of the Hawaii International Conference on Systems Sciences, pages I–449–456, 1992.

[3] D.J. Rogers & R.N. Ibbett. Xbar: a VLSI Circuit for Bit-sliced Packet Switching Networks. In Proc. of IFIP congress '92, 1992.

[4] A. Gottleib, R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolph, and M. Snir. The NYU Ultracomputer — Designing a MIMD Shared Memory Parallel Machine. IEEE Transactions on Computers, C-32(2):175–189, 1983.

[5] Ralph Hole. Interconnection network crossbar switch. B.Eng Honours Project Report, Department of Computer Science, University of Edinburgh, May 1991.

[6] C. P. Kruskal and M. Snir. The Performance of Multistage Interconection Networks for Multiprocessors. IEEE Transactions on Computers, C-32(12):1091–1098, 1983.

[7] M. Kumar and J. R. Jump. Performance Enhancement in Buffered Delta Networks Using Crossbar Switches and Multiple Links. Journal of Parallel and Distributed computing, (1):81–103, 1984.

[8] D. Morris and R.N. Ibbett. The MU5 Computer System. Macmillan, London, 1979.

[9] M.C. Pease. The Indirect Binary n-Cube Microprocessor Array. IEEE Transactions on Computers, C-26(5):458–473, May 1977.

[10] G.F. Pfister, W.C. Brantley, D.A. George, S.L. Harvey, W.J. Kleinfelder, K.P. McAuliffe, E.A. Melton, V.A. Norton, and J. Weiss. The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture. In Proc. International Conference on Parallel Processing, pages 764–771, 1985.

[11] G.F. Pfister and V.A. Norton. "Hot-Spot" Contention and Combining in Multistage Interconnection Networks. In Proc. International Conference on Parallel Processing, pages 790–795, 1985.

[12] R. Rettberg and R. Thomas. Contention is no Obstacle to Shared-Memory Multiprocessing. Communications of the ACM, Vol. 29(12):1202–12, December, 1986.

[13] Howard J. Siegel. Inter-connection networks for Large Scale Parallel Processing (second edition). McGraw-Hill, 1990.

[14] P. Stenström. A Survey of Cache Coherence Schemes for Multiprocessors. IEEECOMP, 24:12–24, 1990.

[15] N.P. Topham, A. Omondi, and R.N. Ibbett. Context Flow: An Alternative to Conventional Pipelined Architectures. Journal of Supercomputing, 2:29–53, September 1988.

[16] A. Trew and G.V. Wilson. Past, Present, Parallel: A Survey of Available Parallel Computing Systems. Springer-Verlag, London, 1991.

[17] W.A. Wulf and C.G. Bell. C.mmp - A multi-mini-processor. Proc. AFIPS Fall Joint Comp. Conf., 41:765–777, 1972.