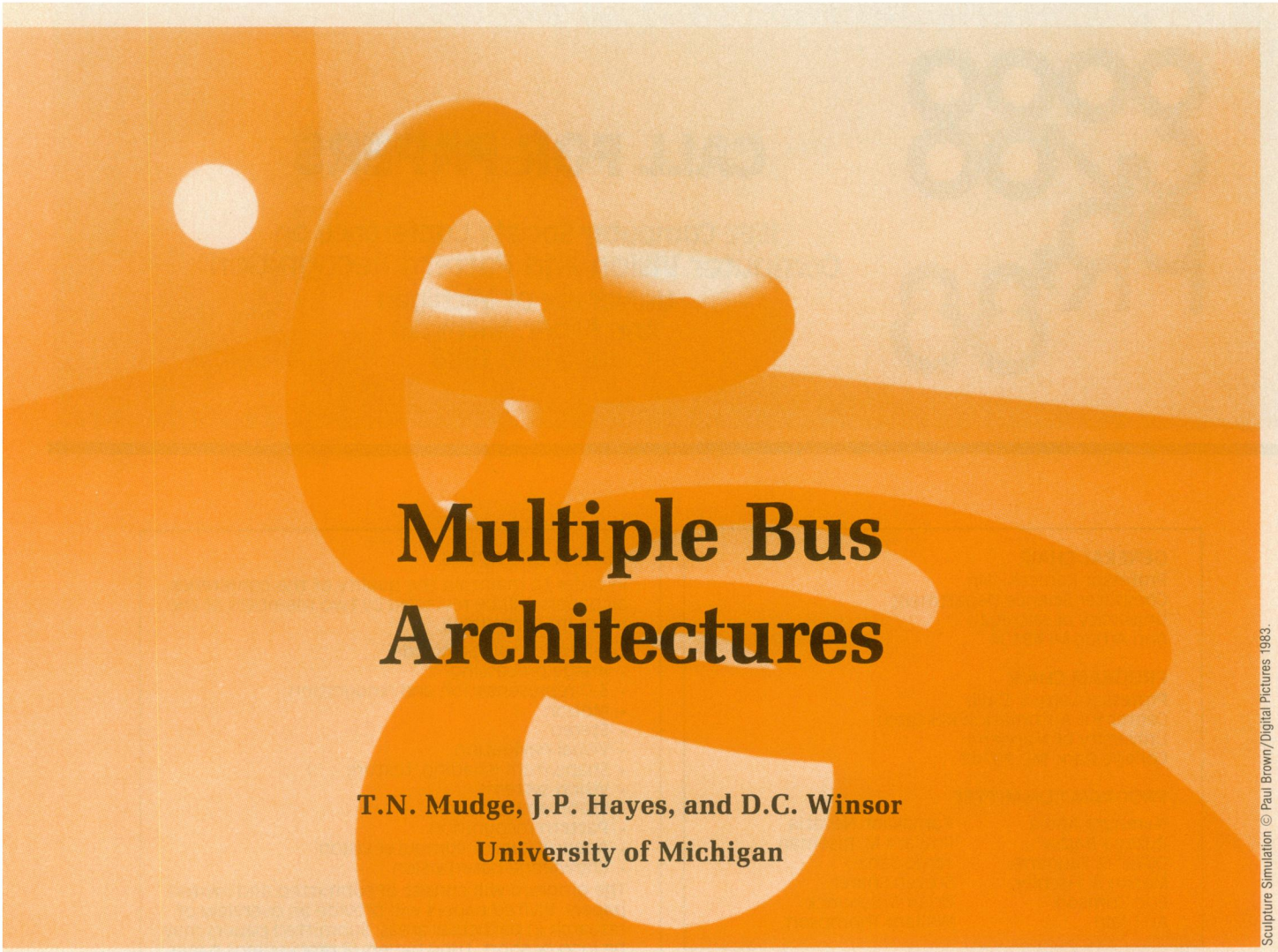


Multiple Bus Architectures

T.N. Mudge, J.P. Hayes, and D.C. Winsor
University of Michigan

Sculpture Simulation © Paul Brown/Digital Pictures 1983.

By replacing the single shared bus of conventional multiprocessor architectures by a set of buses, the number of processors can be increased to a few hundred.

A recent study[1] noted that for shared memory multiprocessors the single system bus typically used to connect the processors to the memory ". . . is by far the most limiting resource, and system performance can be increased considerably by increasing the capacity of the bus." One way of increasing the bus capacity, and also the system's reliability and fault tolerance, is to increase the number of buses. In this article we discuss using multiple buses to provide high-bandwidth connections between the processors and the shared memory, thereby allowing the construction of larger and more powerful systems than currently possible.

The architecture we have in mind is shown in Figure 1. It contains N processors, $P_1, P_2, \ldots, P_N$, each having its own private cache, and all connected to a shared memory by B buses, $B_1, B_2, \ldots, B_B$. The shared memory consists of M interleaved banks, $M_1, M_2, \ldots, M_M$ to allow simultaneous memory requests concurrent access to the shared memory. This avoids the loss in performance that occurs if those accesses must be serialized, which is the case with only one memory bank.

Each processor is connected to every bus and so is each memory bank. When a processor needs to access a particular bank, it has B buses from which to choose. Thus, each processor-memory pair is connected by several redundant paths, which implies that the failure of one or more paths can, in principle, be tolerated at the cost of some degradation in system performance.

In a multiple bus system several processors may attempt to access the shared memory simultaneously. To deal with this, a policy must be implemented that allocates the available buses to the processors making requests to memory. In particular, the policy must deal with the case when the number of processors exceeds B. For performance reasons this allocation must be carried out by hardware arbiters which, as we shall see, add significantly to the complexity of the multiple bus interconnection network.

0018-9162/87/0600-0042$01.00©1987IEEE

Techniques for interconnecting multiple processors and, in particular, connecting multiple processors to a shared memory, have been the object of considerable study during the past decade. Perhaps the most studied type of interconnection network is the multistage network. Many have been proposed, and we refer the reader to Siegel[2] for a thorough discussion of the subject.

The rationale for most multistage networks is that they have fewer components than a crossbar switch while retaining many of the desirable connectivity properties of the crossbar. This reduction in interconnection complexity is important because the systems for which multistage networks are targeted are envisioned to have hundreds or thousands of processors. A few commercial machines based on multistage interconnection networks have been produced, such as the BBN Butterfly. These networks have also been proposed for a number of experimental machines, such as the Illinois Cedar project, the IBM RP3, and the Purdue PASM.

In contrast, the shared memory multiprocessors emerging as commercial products have, with few exceptions, avoided multistage networks as a means of interconnecting the processors to the shared memory. Instead they employ interconnection architectures based on conventional buses. Representative examples are the Encore Multimax and Sequent Balance 8000, which use a single shared bus.[1] One consequence of this evolutionary approach to interconnection design is that the commercial systems have been limited to a few dozen processors rather than thousands of processors. The multiple-bus interconnection network retains the well-understood features of the single bus, but may allow construction of larger systems that can compete with the multistage systems in computing power.

A typical shared system bus contains 50 to 100 wires and is physically realized by printed lines on a circuit board or by discrete (backplane) wiring. Additional costs are incurred by interface electronics, such as line drivers, receivers, and connectors. Concern about wiring and interface costs is probably a major reason why multiple buses have seen little use so far in multiprocessor designs.

In this article we explore some of the practicalities of extending a single shared bus system that can support a few dozen processors to a multiple bus system that can support two or three times as many processors. To illustrate the issues involved, we first describe the operation of a hypothetical multiple bus system in more detail.

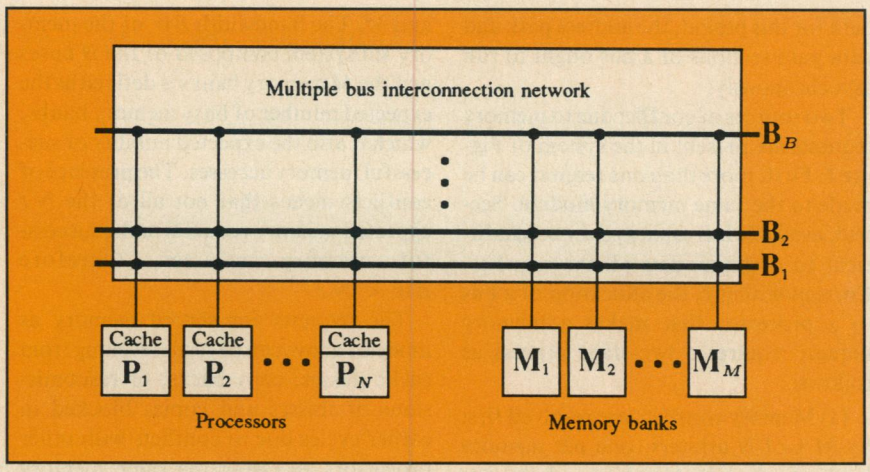


Figure 1. Multiple-bus multiprocessor.

System operation

The importance of the caches in the multiple-bus system of Figure 1 is that they reduce the number of processor-memory requests (the bus traffic) that need to refer back to the shared memory, allowing the construction of more powerful systems without increasing the number of buses. This is important because, as noted above, the buses and their associated interface and control logic form a significant fraction of the system cost. Unlike the uniprocessor case, the cache of a multiprocessor does not have to be constructed from very fast (and expensive) memory, as its primary function is to reduce the traffic with the shared memory, and thus to avoid the potential performance degradation associated with having to wait for a shared resource. The extent to which references have to be made to shared memory depends a lot on the caching algorithm used. A key aspect of this algorithm is the policy for updating the shared memory when a request is addressed to the cache. In this article we assume a simple *copy back* or *write back* updating technique, where an altered cache block is copied to shared memory only when the block is deallocated from the cache.

Copy back divides the references to shared memory into two categories: those due to cache misses and those due to the copy back process. A cache miss occurs when a processor generates a read or write request for data or instructions that cannot be satisfied by the cache. A cache block containing the requested word is read from the shared memory or one of the other caches into the cache where the miss occurred. This replaces a block currently in the cache.

The case where a cache is the source of a requested block occurs when the most recently modified version of the block is in some cache rather than in the shared memory; we refer to this as a cross-cache hit. Caches check for cross-cache hits addressed to themselves by monitoring memory addresses (snooping) on the system buses. Congestion can occur at a cache when it receives several simultaneous cross-cache hits from the buses.

Replaced cache blocks can be discarded (overwritten) if not modified while in the cache. On the other hand, if modified they must be copied back to the shared memory. A processor need not wait for a copy back operation to be completed if a write buffer is used at the interface to the buses.

Ensuring the consistency (coherence) of the data stored in multiple caches can get quite complicated. We refer the reader to Archibald and Baer[1] for a detailed discussion of this issue.

Since both read misses and copy back operations require a cache block to be transferred on a bus, we assume that the data path width of the buses matches the block size of the cache. In the case of reads, the memory address information will precede the data transfer by the memory access time. To avoid idling the data

path for this period, the address path and data path sections of a bus ought to run asynchronously.

Two sources of conflict due to memory requests are present in the system of Figure 1. First, more than one request can be made to the same memory module. Second, available bus capacity may be insufficient to accommodate all the requests. Correspondingly, the allocation of a bus to a processor that makes a memory request requires a two-stage process as follows:

(1) Memory conflicts are resolved first by M 1-of-N arbiters, one per memory bank. Each 1-of-N arbiter selects one request from up to N requests to get access to the memory bank.

(2) Memory requests selected by the memory arbiters are then allocated a bus by a B-of-M arbiter. The B-of-M arbiter selects up to B requests from the M memory arbiters.

The assumption that the address and data paths operate asynchronously allows arbitration to overlap with data transfers.

At first glance it may seem that there should be one bus per memory bank. This obviates the need for the second level of arbitration, yielding a double saving—we can drop the B-of-M arbiter and use a faster bus cycle. However, the optimal configuration may require significantly fewer buses than memory banks. To illustrate this, we next develop a simple performance model of multiple-bus systems.

Performance modeling

If we model the requests made by a processor for memory as a sequence of independent Bernoulli trials,* we can develop expressions for an "average" number of memory requests made by the N processors at the start of each memory cycle and the bandwidth of the interleaved memory. Let r be the probability that an arbitrary processor P_i requests access to shared memory at the start of a memory cycle (this is the Bernoulli trial), then the expected number of requests for shared memory is given by $N r$.

Some requests will always be blocked due to the two types of conflict mentioned earlier, no matter how large we make B and M. The bandwidth BW of the memory subsystem composed of the B buses and the M memory banks is defined as the expected number of busy memory banks, which is also the expected number of successful memory accesses. The presence of conflicts means that not all of the $N r$ expected memory requests make successful memory accesses, therefore $BW < N r$.

The requests for shared memory as modeled above can be reads resulting from cache misses; copy backs; or resubmissions of misses and copies blocked in earlier cycles due to conflicts with other processors accessing the same memory bank. When two or more processors attempt to access memory, an arbiter is invoked to give access to just one of them.

If we assume that the memory banks interleave on the low-order bits of the cache block address and that the instruction fetches and data accesses intermix, then some empirical evidence suggests that requests to all memory banks are equally likely.[3] In other words, the probability that the request from processor P_i is for a particular memory bank M_j is r/M, independent of i or j. Following Mudge et al.,[4] we split the development of an expression for BW into two steps corresponding to the two levels of arbitration.

(1) *Memory arbitration.* The probability that there are no requests from P_i to M_j is $1 - r/M$, and therefore the probability that none of the processors request M_j at the start of a memory cycle is $(1 - r/M)^N$. Let E_j be the event that there is at least one request for M_j; in other words, that the 1-of-N arbiter for M_j outputs a request. Then

$$Pr[E_j] = 1 - \left(1 - \frac{r}{M}\right)^N \qquad (1)$$

for any j. If the events E_j, $j = 1, \ldots, M$, are assumed independent and there are always a sufficient number of buses, i.e., $B \geq M$, then the expected number of busy memory banks is

$$BW_s = \sum_{j=1}^{M} Pr[E_j] = M\left[1 - \left(1 - \frac{r}{M}\right)^N\right]$$

where the subscript S denotes sufficient buses. In the case of large N this expression has an approximate lower bound given by

$$BW_s \approx M\left(1 - e^{-\frac{Nr}{M}}\right) \qquad (2)$$

(In the example given later, this approximation results in an error of less than 0.2 percent.) This expression, or variations on it, has appeared in the literature on numerous occasions. One of the earliest derivations came from Strecker.[5] In specific cases, Equation (2) can be evaluated by estimating r from the miss ratio, the cache block size, the ratio of memory cycle time to processor cycle time α, and the caching algorithm for the processors.

In the most general case of interest, $B < M$. This leads to the next step of the analysis.

(2) *Bus arbitration.* The assumption that the E_j's are independent allows us to express $f(i)$, the probability that exactly i of the memory arbiters output a request at the start of a memory cycle, as follows:

$$f(i) = \binom{M}{i} Pr[E_j]^i (1 - Pr[E_j])^{M-i}$$

In the case where $i \leq B$, there are sufficient buses to handle the memory requests and the B-of-M arbiter does not have to block any requests. In the case where $i > B$, all the B buses are in use and the B-of-M arbiter blocks $i - B$ of the requests. With these two cases in mind, we can write the expression for the expected number of memory banks in use as

$$BW = \sum_{i=1}^{B} if(i) + \sum_{i=B+1}^{M} Bf(i) \qquad (3)$$

where the two terms on the righthand side correspond to the two cases, $i \leq B$ and $i > B$. It is easy to show that when $B = M$, $BW = BW_S$; however, in general $N r > BW_S > BW$, when $r > 0$. These inequalities correspond to conflicts that cause memory requests to be blocked during memory arbitration ($N r > BW_S$) and then during bus arbitration ($BW_S > BW$). Goyal and Agerwala[6] first derived Equation (3) for BW, about the same time that Mudge et al.[4] extended it to the partial bus case. Das and Bhuyan[7] later used it in a reliability study.

Mudge et al.[8] observed that the derivation leading to Equation (3) relies on two assumptions: temporal independence and spatial independence. Temporal independence requires that successive memory requests by a processor be independent, which is clearly not valid for resubmitted blocked requests. Spatial independence corresponds to independent E_j's, an assumption that also has limited validity.

Consider a system with two processors, two buses, and two memory banks. The condition for spatial independence is

*Repeated (statistically) independent trials are called Bernoulli trials if there exist only two possible outcomes for each trial and their probabilities remain the same throughout. An example of this is a sequence of coin flips.

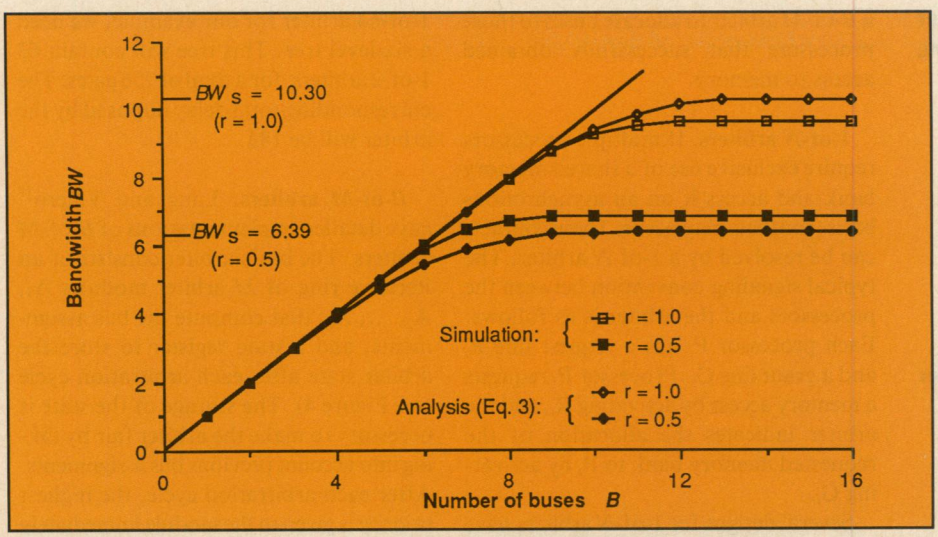


Figure 2. Graphs of BW versus B for $N = M = 16$.

$$Pr[E_1E_2] = Pr[E_1]Pr[E_2] \quad (4)$$

However, $Pr[E_1] = Pr[E_2] = r(1 - (r/4))$ but $Pr[E_1E_2] = r^2/2$. In other words, Equation (4) is not satisfied for $1 \geq r > 0$. The inaccuracy introduced by the two independence assumptions is illustrated in Figure 2, which compares the values of BW calculated from Equation (3) to those obtained by a simulation in which neither temporal nor spatial independence was assumed. In view of the simplifying assumptions in the underlying model, deviation of the computed results from the simulation data is quite small. Note that when the number of buses B is less than BW_S, the buses are the limiting factor, i.e., $BW \approx B$. When the number of buses exceeds BW_S, then the bandwidth can approach its maximum (bus-sufficient) value.

Other models address the accuracy of the independence assumptions. For example, Valero et al.[9] and Bhuyan[10] present several equivalent models in which only temporal independence is assumed. Although more complicated to develop, their models are only slightly more accurate than the model discussed above leading to Equation (3). On the other hand, iterative improvement techniques allow for temporal dependence and yield somewhat more accurate results when applied to Equation (3) and the other models cited above.[8] Towsley[11] developed a model that gives bandwidth predictions generally within one percent of simulation, at the cost of considerable computation. Using semi-Markov processes, Mudge and Al-Sadoun[12] obtained a model that extends to multicycle memory accesses. This model is useful when modeling systems where the block size exceeds the bus data path.

Finally, another class of multiple bus memory interference models, based on the work of Marsan and Gerla, employ traditional Markovian queueing network techniques.[13,14] Their distinguishing feature is that memory access time is treated as an exponentially distributed random variable. However, the fixed access times incorporated into the earlier models are the norm in real memories.

The remainder of our discussion illustrates the use of the performance model leading to Equation (3) and its implications for the design of the multiple bus subsystem.

Design example

Consider the design of the multiple bus architecture of Figure 1, where M and B are to be determined. Suppose that from performance studies of the candidate processors on the anticipated workload, it has been determined that $N = 64$ processors are needed, and that requests for memory from these processors should be successful 90 percent of the time. We make the following three assumptions:

(1) The processor cycle time is half the cycle time of the shared memory, i.e., $\alpha = 2.0$.

(2) The cache has a capacity of 64 kilobytes with a block size of 16 bytes. For such caches the observed miss ratio m has been in the neighborhood of 0.01.[15]

(3) Misses are frequently followed by copy back operations. Statistics published in Smith[15] indicate that the resulting double bus cycles occur about half of the time.

From these three assumptions we obtain an estimate for r of $1.5m\alpha = 0.03$. This leads to an expected number of $Nr = 1.92$ requests for shared memory at the start of each memory cycle.

As we saw in the earlier analysis, it is impossible to satisfy all requests because of conflicts due to memory reference patterns and insufficient bus capacity. The mismatch between memory bandwidth and memory requests can be conveniently characterized by p, defined by

$$p = \frac{BW}{Nr} \quad (5)$$

and corresponds to the probability of a successful request, which we require to be 90 percent. The memory subsystem design problem can now be viewed as choosing values of B and M, for a given N and r so that p reaches a minimum acceptable value. We solve this in two steps. First, we estimate lower bounds on B and M assuming a sufficient number of buses are available. These bounds are used to restrict the search space when solving Equation (3) for values of B and M that satisfy the constraint on p.

It follows from Equation (2) and Equation (5) that the probability of a successful request p_S if there are sufficient buses is given by

$$p_s = \frac{BW_s}{Nr} \quad (6)$$

This is inherent in the memory request patterns and depends only on M. Clearly,

Table 1. Success probability p_S for representative values of M assuming sufficient buses.

M	p_s
4	0.80
8	0.89
16	0.94
32	0.97

Table 2. Success probability p for representative values of M and B.

M	$B=2$	$B=4$	$B=8$
16	0.73	0.92	0.94
32	0.74	0.95	0.97
64	0.74	0.96	0.99

$p_S > p$, as (insufficient) buses can only make things worse. Our design specification calls for processor requests for memory to be successful 90 percent of the time, i.e., $p > 0.9$. The quantity p_S must also satisfy this bound, therefore we can use Equation (2) and Equation (6) to estimate M, the number of memory banks required. Table 1 below shows values of p_S for selected values of M, which are powers of 2 for efficient address interleaving.

From Table 1 we can see that at least 16 memory banks are needed to satisfy the requirement $p_S > p > 0.9$ and to make M a power of 2. Also, since $Nr = 1.92$, we can conclude that at least two buses are needed. Therefore, we can restrict our attention to values of $M \geq 16$ and $B \geq 2$, when using Equation (3) to obtain combinations of B and M that meet the bounds on p. This leads to Table 2, from which we see that $B = 4$ and $M = 16$ yields $p = 0.92$. Hence the system configuration with 4 buses and 16 memory banks is the desired solution to our example problem.

Lang et al.[16] were among the first to recognize in their study of multiple bus systems that the number of buses could be significantly less than the number of memory banks. The savings in the number of buses can be substantial, as we can see from our example, but come at the cost of requiring a B-of-M arbiter. In the next section we will look at this cost in detail.

Arbiter design

As we have seen, a general multiple bus system calls for two types of arbiters: 1-of-N arbiters to select among processors and a B-of-M arbiter to allocate buses to those processors that successfully obtained access to memory.

1-of-N arbiters. If multiple processors require exclusive use of a shared memory bank and access it on an asynchronous basis, conflicts may occur. These conflicts can be resolved by a 1-of-N arbiter. The typical signaling convention between the processors and the arbiter is as follows: Each processor P_i has a request line R_i and a grant line G_i. Processor P_i requests a memory access by activating R_i, and the arbiter indicates the allocation of the requested memory bank to P_i by activating G_i.

Several designs for 1-of-N arbiters have been published.[17] In general, these designs can be grouped into three categories: fixed priority schemes, rings, and trees.

Fixed priority arbiters are relatively simple and fast, but they have the disadvantage that they are not fair: lower priority processors can be forced to wait indefinitely if higher priority processors keep the memory busy.

A ring-structured arbiter gives priority to the processors on a rotating basis, with the lowest priority given to the processor that most recently used the memory bank requested. This has the advantage of being fair, because it guarantees that all processors will access memory in a finite amount of time, but the arbitration time grows linearly with the number of processors.

A tree-structured 1-of-N arbiter is generally a binary tree of depth $\log_2 N$ constructed from 1-of-2 arbiter modules (see Figure 3). Each 1-of-2 arbiter module in the tree has two request input lines, each with a corresponding grant output line, and a cascaded request output and a cascaded grant input for connection to the next arbitration stage. Tree-structured arbiters are faster than ring arbiters because the arbitration time grows as $O(\log_2 N)$ instead of $O(N)$. Fairness can be assured by placing a flip-flop in each 1-of-2 arbiter, toggled automatically to alternate priorities when the arbiter receives simultaneous requests.

Pearce, Field, and Little[17] give an implementation of a 1-of-2 arbiter module constructed from 12 gates. The delay from the request inputs to the cascaded request output is 2Δ, where Δ denotes the nominal gate delay, and the delay from the cascaded grant input to the grant outputs is Δ. Thus, the total delay for a 1-of-N arbiter tree is $3\Delta\log_2 N$. Therefore, to construct a 1-of-64 arbiter for our example, we need a six-level tree. This tree will contain 63 1-of-2 arbiters, for a total of 756 gates. The corresponding total delay imposed by the arbiter will be 18Δ.

***B*-of-*M* arbiters.** Lang and Valero[18] gave detailed implementations of B-of-M arbiters. The basic arbiter consists of an iterative ring of M arbiter modules A_1, A_2, ..., A_M that compute the bus assignments, and a state register to store the arbiter state after each arbitration cycle (see Figure 4). The storage of the state is necessary to make the arbiter fair by taking into account previous bus assignments. After each arbitration cycle, the highest priority is given to the module immediately following the last one serviced—a standard round-robin policy.

An arbitration cycle starts with all the buses available. The state register identifies the highest priority arbiter module, A_i, by asserting signal e_i to that module. Arbitration begins with this module and proceeds around the ring from left to right. At each arbiter module, the R_i input is examined to see if the corresponding memory bank M_i is requesting a bus. If a request is present and a bus is available, the address of the first available bus is placed on the BA_i output and the G_i signal is asserted. BA_i is also passed to the next module, to indicate the highest numbered bus that has been assigned. If a module does not grant a bus, its BA_i output is equal to its BA_{i-1} input. If a module does grant a bus, its BA_i output is set to $BA_i + 1$. When $BA_i = B$ all the buses have been used and the assignment process stops.

The highest priority module A_i, as indicated by the e_i signal, ignores its BA_{i-1} input and begins bus assignment with the first bus by setting $BA_i = 1$. Each module's C_{i-1} input is a signal from the previous module that indicates that the previous module has completed its bus assignment. Arbitration proceeds sequentially through the modules until all of the buses have been assigned, or all the requests have been satisfied. The last module to assign a bus asserts its s_i signal. This is recorded in the state register, which uses it to select the next e_i output so that the next arbitration cycle will begin with the module immediately after the one that assigned the last bus.

Turning to the performance of B-of-M arbiters, we observe that the simple iterative design of Figure 4 must have a delay proportional to M, the number of arbiter

modules. By combining g of these modules into a single module (the "lookahead" design of Lang and Valero[18]), the delay is reduced by a factor of g.

If the enlarged modules are implemented by PLAs with a delay of 3Δ, the resulting delay of the arbiter is about $(3M/g)\Delta$. For our example, where $M = 16$ and $g = 4$, the arbiter delay is about 12Δ. This allows the complete arbitration process for this example to be implemented with delay 30Δ, 18Δ for the 1-of-N arbiter, and 12Δ for the B-of-M arbiter. Since arbitration can be overlapped with bus accesses, the memory bus cycle time must be at least 30Δ.

If the lookahead design approach of Lang and Valero[18] is followed, the

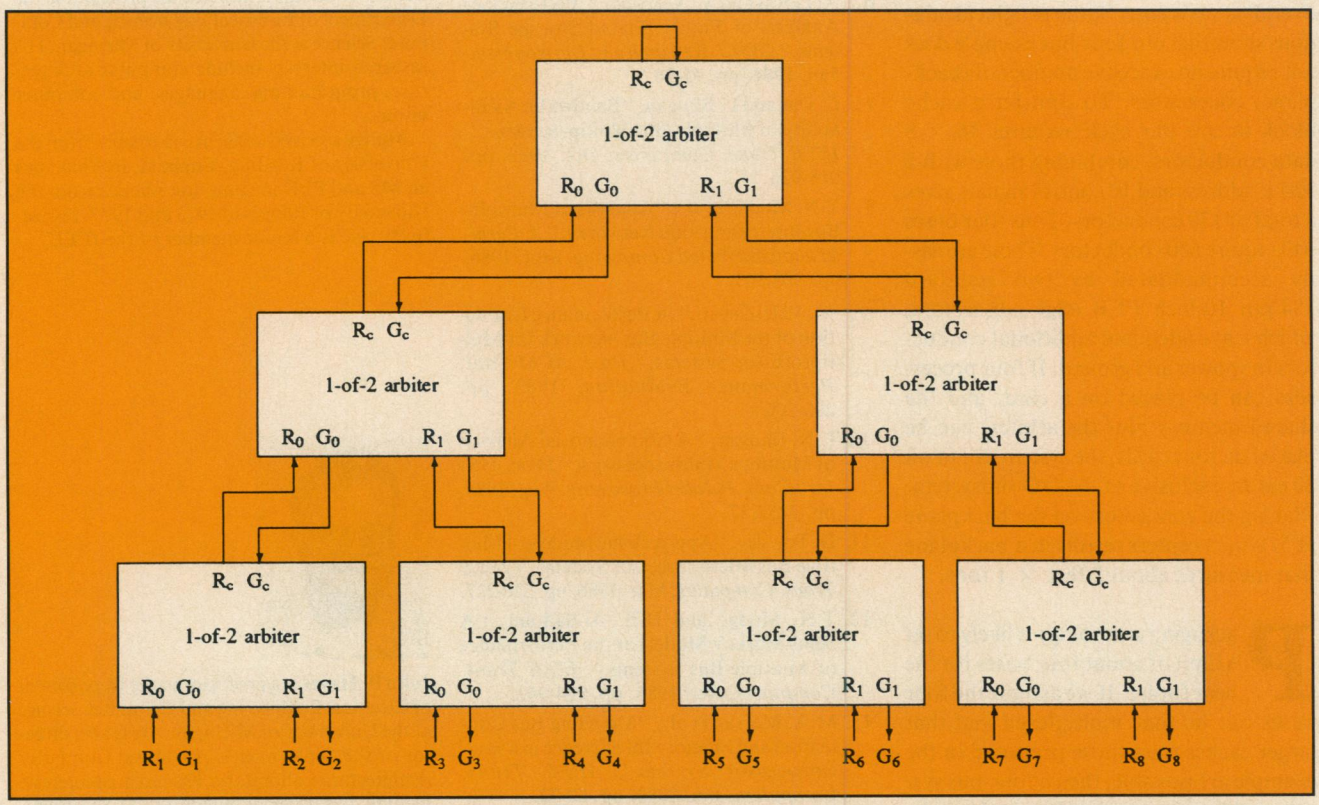


Figure 3. 1-of-8 arbiter constructed from a tree of 1-of-2 arbiters.

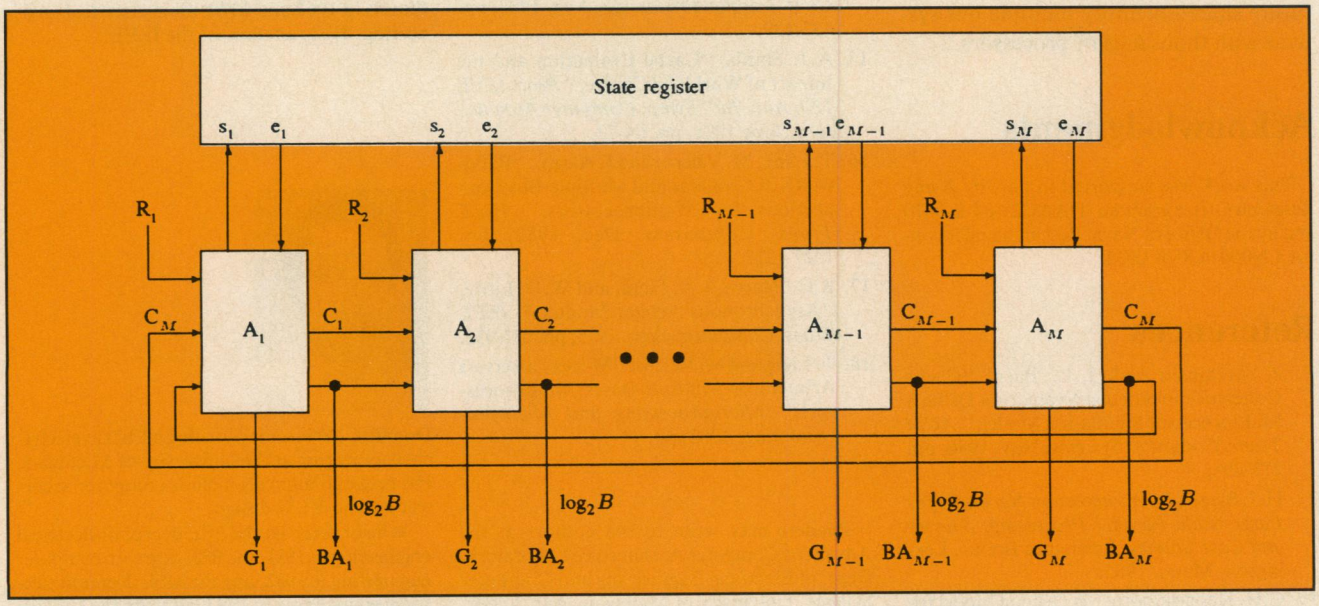


Figure 4. Iterative design for a B-of-M arbiter.

arbitration time of B-of-M arbiters grows at a rate greater than $O(\log_2 M)$ but less than $O(\log_2^2 M)$. Thus the delay of the B-of-M arbiter could become the dominant performance limitation for large M. This, however, is not a problem in our example with $M = 16$.

Like the arbiters, the bus conductors make a significant contribution to the overall system cost. Some rough calculations show that our four-bus example does not require an excessive number of (backplane) conductors. To transfer a cache block in one bus cycle requires 16×8 data conductors; combining these with a 32-bit address and 10 control signals gives a total of 170 conductors. Thus four buses will require 680 conductors. These are easily accommodated by two standard 384-pin 10-inch PCB edge connectors, which leave adequate additional conductors for power and ground. If four processors can be placed on a card, and the shared memory plus the arbiters can be placed on four cards, the system will fit on 20 cards, exclusive of the I/O subsystem. Placing the connectors on the backplane at 5/8-inch centers results in a backplane that measures about 2 feet $\times$ 1 foot.

Packaging technology is likely to set a limit of about four buses for the near future. If we assume the four buses can be time multiplexed and that larger caches than those proposed in the example are possible, then a four-bus system could support several hundred processors. If optical backplanes become a reality, the number of buses that could be supported will greatly increase, allowing future shared-memory multiple bus systems with thousands of processors. □

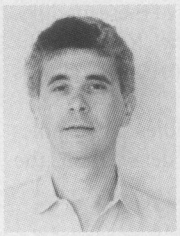
Acknowledgments

This work was supported in part by Army Research Office grant no. DAAG29-84-K-0070 and by the Office of Naval Research under contract N00014 85 K 0531.

References

1. J. Archibald and J.-L. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," *ACM Trans. Computer Systems*, Nov. 1986, pp. 273-298.
2. H.J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, Lexington Books, Lexington, Mass., 1985.
3. C.H. Hoogendoorn, "A General Model for Memory Interference in Multiprocessors," *IEEE Trans. Computers*, Oct. 1977, pp. 998-1005.
4. T.N. Mudge et al., "Analysis of Multiple-Bus Interconnection Networks," *Proc. 1984 Int'l Conf. Parallel Processing*, Aug. 1984, pp. 228-232.
5. W.D. Strecker, *Analysis of the Instruction Execution Rate in Certain Computer Structures*, PhD thesis, Carnegie Mellon University, Pittsburgh, Penn., 1970.
6. A. Goyal and T. Agerwala, "Performance Analysis of Future Shared Storage Systems," *IBM J. Research and Development*, Jan. 1984, pp. 95-98.
7. C. Das and L. Bhuyan, "Bandwidth Availability of Multiple-Bus Multiprocessors," *IEEE Trans. Computers*, Oct. 1985, pp. 918-926.
8. T.N. Mudge et al., "Analysis of Multiple-Bus Interconnection Networks," *J. Parallel and Distributed Computing*, Sept. 1986, pp. 328-343.
9. M. Valero et al., "A Performance Evaluation of the Multiple Bus Network for Multiprocessor Systems," *Proc. ACM Conf. Performance Evaluation*, 1983, pp. 200-206.
10. L.N. Bhuyan, "A Combinatorial Analysis of Multibus Multiprocessors," *Proc. 1984 Int'l Conf. Parallel Processing*, Aug. 1984, pp. 225-227.
11. D. Towsley, "Approximate Models of Multiple Bus Microprocessor Systems," *IEEE Trans. Computers*, Mar. 1986, pp. 220-227.
12. T.N. Mudge and H.B. Al-Sadoun, "A Semi-Markov Model for the Performance of Multiple-Bus Systems," *IEEE Trans. Computers*, Oct. 1985, pp. 934-942.
13. M.A. Marsan et al., "Modeling Bus Contention and Memory Interference in a Multiprocessor System," *IEEE Trans. Computers*, Jan. 1983, pp. 60-72.
14. I.H. Önyüksel and K.B. Irani, "A Markovian Queueing Network Model for Performance Evaluation of Bus-Deficient Multiprocessor Systems," *Proc. 1983 Int'l Conf. Parallel Processing*, Aug. 1983, pp. 437-439.
15. A.J. Smith, "Cache Evaluation and the Impact of Workload Choice," *Proc. IEEE 12th Ann. Int'l Symp. Computer Architecture*, June 1985, pp. 64-73.
16. T. Lang, M. Valero, and I. Alegre, "Bandwidth of Crossbar and Multiple-Bus Connections for Multiprocessors," *IEEE Trans. Computers*, Dec. 1982, pp. 1227-1233.
17. R.C. Pearce, J.A. Field, and W.D. Little, "Asynchronous Arbiter Module," *IEEE Trans. Computers*, Sept. 1975, pp. 931-932.
18. T. Lang and M. Valero, "M-users B-servers Arbiter for Multiple-Buses Multiprocessors," *Microprocessing and Microprogramming*, 1982, pp. 11-18.

Readers may write to the authors at the Advanced Computer Architecture Laboratory, Dept. of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109.

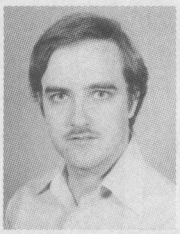


Trevor N. Mudge is an associate professor in the Department of Electrical Engineering and Computer Science at the University of Michigan. His research interests include computer architecture, programming languages, and computer vision.

Mudge received a BSc in cybernetics from the University of Reading, England, in 1969, and an MS and PhD in computer science from the University of Illinois in 1973 and 1977, respectively. He is a senior member of the IEEE.

John P. Hayes is a professor in the Department of Electrical Engineering and Computer Science at the University of Michigan. He is also director of the department's Advanced Computer Architecture Laboratory. His research interests include computer architecture, VLSI design, digital system testing, and switching theory.

Hayes received the BE degree from the National University of Ireland in 1965, and the MS and PhD degrees from the University of Illinois in 1967 and 1970, all in electrical engineering. He is a fellow of the IEEE.



Donald C. Winsor is pursuing a PhD in electrical engineering at the University of Michigan. His research interests include computer architecture and VLSI design.

Winsor received the BSE degree in electrical engineering in 1981, the BSE degree in computer engineering in 1982, and the MSE degree in electrical engineering in 1983, all from the University of Michigan.