Proc. of the 1987 Int. Conf. Parallel Processing, Aug. 1987, pp. 266-269.

Crosspoint Cache Architectures *

Donald C. Winsor and Trevor N. Mudge
Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2212

Abstract: We propose a new architecture for shared memory multiprocessors—the crosspoint cache architecture. This architecture consists of a crossbar interconnection network with a cache memory at each crosspoint switch. It assures cache coherence in hardware while avoiding the performance bottlenecks associated with previous hardware cache coherence solutions. We show this architecture is feasible for a 64 processor system. We also consider a two-level cache architecture in which caches on the processor chips are used in addition to the caches in the crosspoints. This two-level cache organization achieves the goals of fast memory access and low bus traffic in a cost effective way.

Introduction

Advances in VLSI technology have made available high performance single-chip 32-bit processors. The excellent cost/performance ratio of these microprocessors has generated considerable interest in using them to build high performance multiprocessor systems. Architectures in which all of the processors share a single memory have significant advantages in flexibility and ease of programming over other multiprocessor architectures. However, the maximum performance of these shared memory systems is extremely sensitive to both memory bandwidth and memory access time; thus, cache memories are an essential component of this class of multiprocessor.

When using a private cache memory for each processor in a multiprocessor system, it is necessary to ensure that all valid copies of a given cache block are the same. This requirement is called the *multicache consistency* or *cache coherence* problem. Solutions to the cache coherence problem based on hardware, software, and combinations of both have been proposed. Hardware solutions have the advantage of being completely transparent to the software; existing programs need not be modified. However, none of the proposed solutions is completely satisfactory for large numbers of high performance processors. They all become prohibitively expensive, slow, or both, for more than a few processors. Software solutions avoid the cost of special hardware, but they require the operating system to identify all shared regions of memory and to issue appropriate commands to the caches to ensure consistency. These requirements significantly complicate the operating system. Furthermore, the high memory traffic generated by the software solutions may be prohibitive for large numbers of processors. For a more detailed survey of cache coherence schemes, the reader is referred to [1].

In this paper, we propose a new cache architecture that assures cache coherence in hardware while avoiding the performance bottlenecks associated with previous hardware solutions. Our proposed architecture is a crossbar interconnection network with a cache memory at each crosspoint. Crossbars have traditionally been avoided because of their complexity. However, for the "non-square" systems that we are focusing on with 16 or 32 processors per memory bank and no more than 4 to 8 memory banks, the number of crosspoint switches required is not excessive. The simplicity and regular structure of the crossbar architecture greatly outweigh any disadvantages due to complexity. We will show that our architecture allows the use of straightforward and efficient bus oriented cache coherence schemes while overcoming their bus traffic limitations.

Snooping caches

The most promising solutions to the cache coherence problem require that all processors share a common main memory bus. Each cache monitors all bus activity to identify references to its lines by other caches in the system. This monitoring activity is called "snooping" on the bus. Along with the "valid" and "dirty" tag bits for each line, there may be one or more additional bits to record whether the line is shared or exclusive. Thus, data is dynamically classified as shared or exclusive, as in the centralized directory approach, but the directory information and control logic is distributed among the caches. This avoids the bottleneck of the central directory and controller.

Although this approach appears to be similar to broadcasting writes, its performance is much better as long as the cache policy is not write through. Since the caches record the shared or exclusive status of each line, it is only necessary to broadcast writes to shared lines on the bus; bus activity for exclusive lines is avoided. Thus, the cache bandwidth problem is much less severe than for the broadcast writes scheme.

One of the first snooping cache schemes was the "write once" technique proposed in [2]. Numerous variations and improvements have followed. These methods differ primarily in the details of the cache and bus protocols used to ensure cache coherence.

The major limitation of the snooping cache schemes is that they require all processors to share a common bus. The bandwidth of a single bus is typically insufficient for even a few dozen processors. Higher bandwidth interconnection networks such as crossbars and multistage networks cannot be used with snooping cache schemes, since there is no simple way for every cache to monitor the memory references of all the other processors. Multiple bus systems similar to those in [3] could be used, but each cache would have to monitor every cycle on every bus. This would be impractical for more than a few buses, since it would require extremely high bandwidth for the cache address tags. Multiple buses also cause difficult synchronization problems. If two processors reference the same line (using two different buses), they could each modify a copy of the line in the other's cache, thus leaving that line in an inconsistent state.

Crosspoint cache architecture description

In this section, we show how two existing architectures, the single bus with snooping caches and the crossbar network, may be combined to form a new architecture, the *crosspoint cache architecture.*

*This work was supported in part by Army Research Office grant number DAAG29-84-K-0070

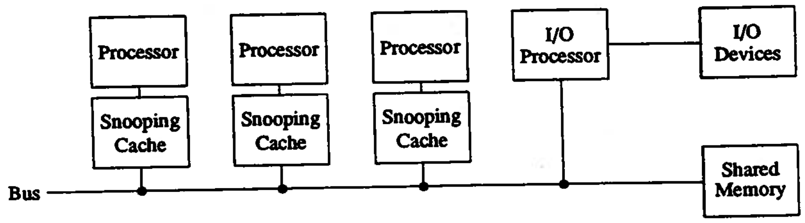

Figure 1: Single bus with snooping caches

Single bus architecture

Figure 1 shows the architecture of a single bus multiprocessor with snooping caches. Each processor has a private cache memory. The caches all service their misses by going to main memory over the shared bus. Cache coherence is ensured by using a snooping cache protocol in which each cache monitors all bus addresses.

Cache coherence problems due to input/output operations are solved by connecting the I/O processors to the shared bus and having them observe the bus cache coherence protocol. Since the only function of the I/O processor is to transfer data to and from main memory for use by other processors, there is little or no advantage in using cache memory with it. It may be desirable to cache disk blocks in main memory, but this is a software issue unrelated to the use of a hardware cache between the I/O processor and the bus.

Although this architecture is simple and inexpensive, the bandwidth of the shared bus severely limits its maximum performance. Furthermore, since the bus is shared by all the processors, arbitration logic is needed to control access to the bus. Logic delays in the arbitration circuitry may impose additional performance penalties.

Crossbar architecture

Figure 2 shows the architecture of a crossbar network. Each processor has its own bus, as does each memory bank. The processor and memory buses are oriented (at least conceptually) at right angles to each other, forming a two-dimensional grid. A crosspoint switch is placed at each intersection of a processor bus and a memory bus. Each crosspoint switch consists of a bidirectional bus transceiver and the control logic needed to enable the transceiver at the appropriate times. This array of crosspoint switches allows any processor to be connected to any memory bank through a single switching element. Arbitration is still needed on the memory buses, since each is shared by all processors. Thus, this architecture does not eliminate arbitration delay.

The crossbar architecture is more expensive than a single bus. However, it avoids the performance bottleneck of the single bus, since several memory requests may be serviced simultaneously. Unfortunately, if a cache were associated with each processor in this architecture, cache coherence would be difficult to achieve. The snooping cache schemes would not work, since there is no reasonable way for every processor to monitor all the memory references of every other processor. To overcome this problem, we propose the crosspoint cache architecture.

Crosspoint cache architecture

In the crosspoint cache architecture the general structure is similar to that of the crossbar network shown in Figure 2, with the addition of a cache memory in each crosspoint.

For each processor, the multiple crosspoint cache memories that serve it (those attached to its processor bus) behave similarly to a larger single cache memory. For example, in a system with four memory banks and a 16K byte direct mapped cache with a 16 byte line size at each crosspoint, each processor would "see" a single 64K byte direct mapped cache with a 16 byte line size. Note that this use of multiple caches with each processor increases the total cache size, but it does not affect the line size or the degree of set associativity. This approach is, in effect, an interleaving of the entire memory subsystem, including both the caches and the main memory.

To explain the detailed functioning of this system, we consider processor bus activity and memory bus activity separately.

Processor bus activity

Each processor has the exclusive use of its processor bus and all the caches connected to it. There is only one cache in which a memory reference of a particular processor to a particular memory bank may be cached. This is the cache at the intersection of the corresponding processor and memory buses.

The processor bus bandwidth requirement is low, since each bus needs only enough bandwidth to service the memory requests of a single processor. The cache bandwidth requirement is even lower, since each cache only handles requests from a single processor, and it only services those requests directed to a particular memory bank.

Note that this is not a shared cache system. Since each processor bus and the caches on it are dedicated to a single processor, arbitration is not needed for a processor bus or its caches. Furthermore, bus interference and cache interference cannot occur. Thus, the principal delays associated with shared cache systems are avoided.

Memory bus activity

When a cache miss occurs a memory bus transaction is necessary. The cache that missed places the requested memory address on the bus and waits for main memory (or sometimes another cache) to supply the data. Since all the caches on a particular memory bus may generate bus requests, bus arbitration is necessary on the memory buses. Also, since data from a particular memory bank may be cached in any of the caches connected to the corresponding memory bus, it is necessary to observe a cache coherence protocol along the memory buses. The cache coherence protocol will make memory bus operations necessary for write hits to shared lines as well as for cache misses.

Since each memory bus services only a fraction of each processor's cache misses, this architecture can support more processors than a single bus system before reaching the upper bound on performance imposed by the memory bus bandwidth. For example, if main memory were divided into four banks, each with its own memory bus, then each memory bus would only service an average of one fourth of all the cache misses in the system. So, the memory bus bandwidth would allow four times as many processors as a single bus snooping cache system.

Memory addressing example

To better illustrate the memory addressing in the crosspoint cache architecture, we consider a system with the following parameters: 64 processors, 4 memory banks, 256 crosspoint caches, 32-bit byte addressable address space, 32-bit word size, 32-bit bus width, 4 word (16 byte) crosspoint cache line size, 16K byte (1024 lines) crosspoint cache size, crosspoint caches direct mapped.

When a processor issues a memory request, the 32 bits of the memory address are used as follows: The two least significant bits select one of the four bytes in a word. The next two bits select one of the four words in a cache line. The next two bits select one of the four memory banks,

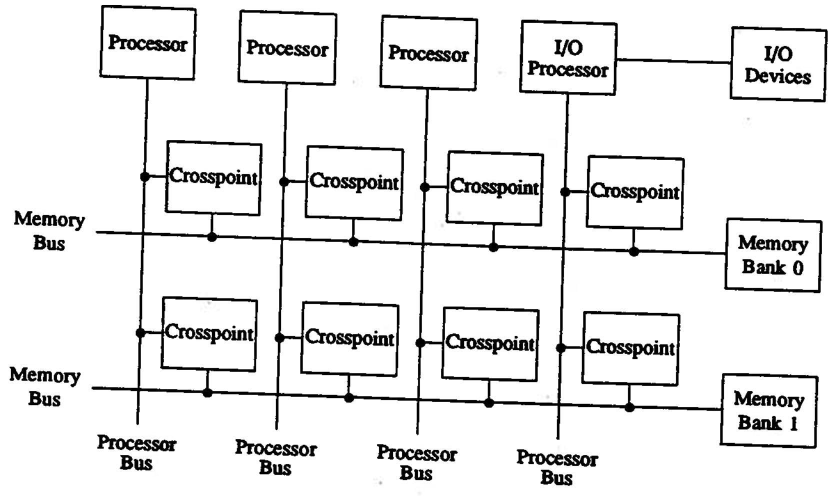

Figure 2: Crossbar network

and thus one of the four crosspoint caches associated with the requesting processor. The next ten bits select one of the 1024 lines in a particular crosspoint cache. The remaining bits (the most significant 16) are the ones compared with the address tag in the crosspoint cache to see if a cache hit occurs.

Performance estimate

Because of the importance of a good snooping cache scheme to our proposed design, we select a particular coherence protocol before proceeding with our performance analysis. Six snooping cache protocols, called *Write-once, Synapse, Berkeley, Illinois, Firefly*, and *Dragon*, are evaluated in [4] using a simulation model. The results indicate that the Dragon protocol gives the best performance over a wide range of workloads. (Dragon is a protocol used in a multiprocessor of the same name being designed at Xerox Palo Alto Research Center.) The reader is referred to [4] for a detailed explanation of the Dragon protocol.

The simulation results of [4] show that the Dragon protocol can support up to 15 processors without saturating the bus. Results were not shown for more than 15 processors. The assumptions underlying this result include: 16K word cache size, 4 word line size, one word bus width, bus cycle time equal to cache cycle time, main memory cycle time equal to four cache cycles, 98% cache hit ratio for private blocks, 85% read operations, 16 to 1024 shared lines, 5% of references to shared lines.

Since saturation was not reached even with 15 processors, we will assume that placing 16 processors on a single bus is feasible. Now consider a crosspoint cache system with 64 processors, 4 memory banks, and 4K words per crosspoint cache as in our previous example. Since four crosspoint caches of 4K words behave similarly to a single cache of 16K words, we would expect the simulation results for a single 16K word cache to be valid for the four 4K word crosspoint caches also. Since the total bus and memory bandwidth of the crosspoint cache system is four times that of a single bus, single memory system, we expect that this bandwidth will adequately support 64 processors without bus saturation.

Bus loading on the memory buses may limit the maximum size of crosspoint cache systems, since all the processors must be connected to each memory bus. As the number of processors becomes large, bus propagation delays and capacitive loading will reduce the maximum speed and bandwidth of the bus. A slow bus would be intolerable in the absence of cache memories. However, with cache memories a slow bus will not seriously reduce performance, since most memory references will be satisfied by the caches and will not involve the bus at all. Similarly, the bus traffic reduction obtained from the caches will offset the reduced bandwidth of a slower bus.

Two-level caches

The performance of the crosspoint cache architecture may be further improved by adding a local cache between each processor and its processor bus. To see why this is so, we examine some of the tradeoffs in designing a crosspoint cache system.

Since memory bus bandwidth is a critical resource, large crosspoint caches are desirable to maximize the cache hit rate, thus minimizing the memory bus traffic. Cache speed is one of the most important factors influencing a processor's average memory access time. Thus, the speed of the crosspoint caches should be as fast as possible to maximize the performance of each individual processor. Simultaneously achieving the goals of low bus traffic and fast memory access would be expensive, since large amounts of fast memory would be necessary.

Processor bus delay also limits the speed of the individual processors. Although the crosspoint cache architecture eliminates processor bus arbitration delays and largely eliminates delays from cache interference, there is still the delay of the processor bus itself. Bus propagation delays, capacitive loading, and delays from the bus interface logic limit the maximum feasible bus speed.

By placing a fast cache between each processor and its processor bus, the effect of the processor bus and crosspoint cache delays can be greatly reduced. When speed is the primary consideration, the best possible location for a processor's cache is on the processor chip itself. On-chip caches can be extremely fast, since they avoid the delays due to IC packaging and circuit board wiring. They are limited to a small size, however, since the limited area of a microprocessor chip must be allocated to the processor itself, the cache, and any other special features or performance enhancements desired. By combining a fast but small on-chip cache with large but slow crosspoint caches, the benefits of both can be realized. The fast on-chip cache will serve primarily to keep average memory access time small, while the large crosspoint caches will keep memory bus traffic low.

Cache coherence with two-level caches

Using a two-level cache scheme introduces additional cache coherence problems. Fortunately, a simple solution is possible.

In our cache coherence solution, the Dragon protocol is used on the memory buses to ensure coherence between the crosspoint caches. The on-chip caches use write through to ensure that the crosspoint caches always have current data. The high traffic of write through caches is not a problem, since the processor buses are only used by a single processor. Since write through ensures that lines in the crosspoint caches are always current, the crosspoint caches can service any references to shared lines that they contain without interfering with the processor or its on-chip cache.

Special attention must be given to the case in which a processor writes to a shared line that is present in another processor's on-chip cache. It is undesirable to send all shared writes to the on-chip caches, since this would reduce the bandwidth of the on-chip caches that is available to their processors.

If each crosspoint cache can always determine whether one of its lines is also present in its associated on-chip cache, then it can restrict accesses to the on-chip cache to only those that are absolutely necessary. When a write to a shared line hits on the crosspoint cache, the crosspoint cache can send an invalidation request for the line to the on-chip cache only if the on-chip cache really has the line.

With suitable cache design, the crosspoint cache can determine whether one of its lines is currently in the on-chip cache, since the on-chip cache must go through the crosspoint cache to obtain all its lines. To see how this can be done, we consider the simplest case. This occurs when both the on-chip and crosspoint caches are direct mapped and have equal line sizes.

In a direct mapped cache, there is only a single location in which a particular line from main memory may be placed. In most designs, this location is selected by the bits of the memory address just above the bits used to select a particular word in the line. If the total size of the crosspoint caches is larger than that of their associated on-chip cache and all line sizes are equal, then the address bits that are used to select a particular crosspoint cache entry will be a superset of those bits used to select the on-chip cache entry. Consider those address bits that are used to select the crosspoint cache entry but not the on-chip cache entry. Out of a group of all crosspoint cache entries that differ only in these address bits, exactly one will be in the on-chip cache at any given time. If the value of these address bits are recorded in a special memory in the crosspoint caches for each line obtained by the on-chip cache, a complete record of which lines are in the on-chip cache will be available.

To determine if a particular line is in the on-chip cache, the crosspoint cache uses the same address bits used to select the on-chip cache entry to select an entry in this special memory. It then compares the additional address bits used to select the crosspoint cache entry with the value of those bits that is stored in the special memory. These bits will be equal if and only if the line is in the on-chip cache.

The size in bits of the special memory for each crosspoint cache is given by:

$$\left(\frac{L_{oc}}{M}\right) \log_2 \left(M \frac{L_{xp}}{L_{oc}}\right)$$

where L_{oc} is the number of lines in the on-chip cache, L_{xp} is the number of lines in each crosspoint cache, and M is the number of memory banks. This is not a large amount of memory. In our example system with four memory banks, a line size of 16 bytes, and a crosspoint cache size of 16K bytes, we have $M = 4$ and $L_{xp} = 1024$. If we assume an on-chip cache size of 1K byte, we have $L_{oc} = 64$, so only 96 bits per crosspoint cache are needed to keep track of the lines in the on-chip cache.

This approach is more difficult to use with set associative on-chip caches, since additional signals must be provided on the microprocessor to allow the on-chip cache to inform the crosspoint caches of the location (which element in the set) of each line it loads.

A disadvantage of this two-level cache coherence approach is that it requires arbitration on the processor buses, since the crosspoint caches use these buses to issue the invalidation requests. This will decrease the effective speed of the processor buses, so the time required to service a miss for the on-chip cache will be slightly greater than the memory access time of a similar system without the on-chip caches.

VLSI considerations

Using VLSI technology to build a crossbar network requires an extremely large number of pin connections. For example, a crossbar network with 64 processors, 4 memory banks, and a 32 bit multiplexed address and data path requires at least 2208 connections. Present VLSI packaging technology is limited to a maximum of several hundred pins. Thus, a crossbar of this size must be partitioned across multiple packages.

The most straightforward partitioning of a crossbar network is to use one package per crosspoint. This results in a design that is simple and easy to expand to any desired size. The complexity of a single crosspoint is roughly equivalent to an MSI TTL package, so the ratio of pins to gates is high. This approach leads to MSI circuitry in VLSI packages, so it does not fully exploit the capabilities of VLSI. A much better pin to gate ratio is obtained by using a bit sliced partitioning in which each package contains a single bit of the data path of the entire network. The bit sliced approach, however, is difficult to expand since the network size is locked into the IC design.

The crosspoint cache architecture, on the other hand, permits the construction of a single VLSI component containing the crosspoint cache and its bus interfaces that is efficient in systems spanning a wide performance range. If each package contains a single crosspoint cache, the number of pins required is reasonable, and the cache size may be made as large as necessary to take full advantage of the available silicon area. It also allows the same chip to be used both in small systems with just a few processors and a single memory bank and in large systems with a hundred or more processors and eight or sixteen memory banks.

In the example given, each crosspoint cache contains 128K bits of data storage, approximately 20K bits of tag storage, and some fairly simple switch and control logic. Since static RAMs as large as 256K bits are widely available, it should be feasible to construct such a crosspoint cache on a single chip with present VLSI technology.

Summary and future research

To overcome the performance limitations of shared memory systems with a single bus while retaining many of their advantages, we have proposed the crosspoint cache architecture. We have shown that this architecture should permit shared memory multiprocessor systems to be constructed with more processors than present systems while avoiding the need for software enforcement of cache coherence.

We have also described a two-level cache architecture in which both crosspoint caches and caches on the processor chips are used. This architecture uses small but fast on-chip caches and large but slow crosspoint caches to achieve the goals of fast memory access and low bus traffic in a cost effective way. Further investigation of the protocols for a two-level cache would be a useful topic for future research.

Finally, run-time measurements of real multiprocessor programs are needed. Measurements of the quantity of shared data and the frequency with which this data is referenced would be extremely valuable, as they would allow more realistic estimates to be made of the performance that can be obtained with the crosspoint cache architecture.

References

[1] Alan Jay Smith, 'CPU Cache Consistency with Software Support and Using "One Time Identifiers"', *Proceedings of the Pacific Computer Communications Symposium*, Seoul, Republic of Korea, October 21-25, 1985, pages 142-150.

[2] James R. Goodman, "Using Cache Memory to Reduce Processor–Memory Traffic", *Proceedings of the 10th Annual International Symposium on Computer Architecture*, Stockholm, Sweden, volume 11, number 3, June 13-17, 1983, pages 124-131.

[3] T. N. Mudge, J. P. Hayes, G. D. Buzzard and D. C. Winsor, "Analysis of Multiple-Bus Interconnection Networks", *Journal of Parallel and Distributed Computing*, volume 3, number 3, September 1986, pages 328-343.

[4] James Archibald and Jean-Loup Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model", *ACM Transactions on Computer Systems*, volume 4, number 4, November 1986, pages 273-298.