Multiprocessors

We can think of multiprocessors as a way to increase the parallelism of the uniprocessor systems we have been studying. Scalar processors have a single pipeline. Superscalar processors have multiple parallel execution pipelines, but share common instruction fetch and decode pipelines. Multiprocessors have complete multiple parallel pipelines where each pipeline has its own fetch and decode stages.

Anyone that has used the internet has experienced a system with multiple processors. The internet consists of massive numbers of independent processors that are loosely coupled through their network connection. To each processor, the network connection looks like another relatively slow I/O device. In particular, each processor has its own memory which is not shared with other processors.

Multiprocessors are much more tightly coupled. Memory (and I/O devices) are shared via a local interconnection network. Each processor has access to its own memory and all the memory of all the other processors. This requires a much higher speed interconnection that is capable of keeping up with memory data transfer rates. The two basic organizations of shared memory multiprocessors are shown in fig. 9-4, p. 424.

Memory becomes a common resource which must be shared between execution threads running simultaneously (really simultaneously, not time shared) on different processors in the multiprocessor system. The use of memory by different threads running on the same or different processors must be properly synchronized using the techniques of the previous section. In this manner, the design goals (middle of p. 423) for multiprocessors can be achieved.

Cache Coherence. The unit latency requirement is usually satisfied by providing each processor with its own local cache as seen in fig. 9-4, p. 424. As long as cache miss rates are low, the unit latency goal will be (almost) met.

Separate local caches makes it difficult for each processor to have a coherent view of memory which includes the effects of its own memory writes and the memory writes of other processors. Coherence protocols are necessary as shown in fig. 9-5, p. 425.

An update protocol requires the transfer of write data (and write address) to all the processor caches that have a copy of the data. Since the data is transferred through the interconnection network, the data might as well be stored in main memory as well. This is very similar to a write-through strategy for uniprocessor cache. Write-through (and therefore update protocols) is known to have high memory bandwidth requirements since each write (by every processor) must be transmitted on the interconnection network. For this reason, update protocols are not used in modern designs.

An invalidate protocol only requires the transfer of the write address to all processor caches that have a copy of the data. Furthermore, once the copies of the data in other caches have been invalidated (removed from the cache), the original processor has an exclusive copy of the data. It can continue to read or write to this location without notifying the other processors. This is very similar to a write-back strategy for uniprocessor cache. Write-back (and therefore invalidate protocols) is known to have lower memory bandwidth requirements since most data transfers take place to local cache without using the interconnection network at all. Once in the exclusive state, the interconnect network is used only when

1. the local cache must write back (evict) the data to make room for new data locations in cache,
2. other processors want to read or write to the exclusive location.

The invalidate protocol is usually implemented with the four states shown in fig. 9-6, p. 428. Note:

1. Each cache block has its own state which is maintained by the local cache controller in response to both local processor memory operations and bus operations from remote processors.
2. Bus read, bus write and bus upgrade correspond to remote read, remote write and remote upgrade (invalidate) respectively.
3. Bus upgrade is usually called bus invalidate since this is the message to invalidate caches in the shared state.
4. A cache miss (tag mismatch) on read or write is treated the same as a local read or write to an invalid state.
5. The Modified state is also exclusive. Only one cache can be in the exclusive or modified state for a particular memory block.
6. Local evict does not have an effect on other caches.

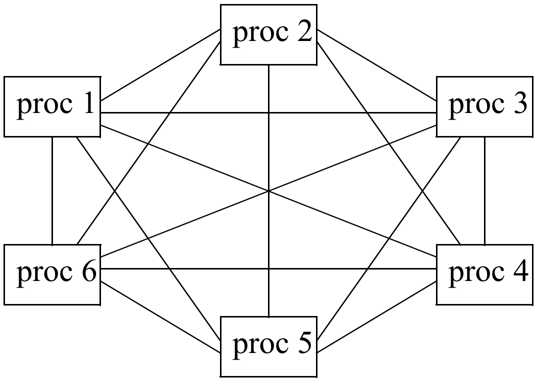
Snooping. The simplest way to implement cache coherence invalidate protocols is to have all processors share a common address bus. The common address bus can be a single physical bus or it might be a collection of physical busses in a more complicated interconnection topology. Each processors monitors (snoops) addresses on the bus from other processors. When an address on the address bus matches a location in the local cache, the appropriate action is taken according to the coherence protocol.

State transitions in fig. 9-6 do not occur until the local cache controller gains access to the shared address bus. This prevents such problems as two cache controllers in the shared state from invalidating each other. This means that the real cache controller has intermediate states to wait for bus access.

The cache must respond to the local processor as any normal write-back cache would. In addition, snooping requires a cache operation (tag match) every time any processor uses the shared address bus. This normally requires multiport cache to avoid slowing down the local processor.

Snooping has been used successfully in commercial systems with up to 64 processors. For larger multiprocessor systems, the bandwidth requirements on the shared address bus will make snooping impractical. Each processor puts addresses on the shared address bus at a rate shown in eq. 9-1, p. 429. The total bus traffic, eq. 9-2, is proportional to the number of processors. Since only one processor can put an address on the shared bus at any given time, it takes longer and longer for messages on the bus to get through as most processors spend time waiting to get control of the bus. The waiting time appears as increased memory latency which increases the effective cache miss penalty.

Limitations of Snooping. The bandwidth of a single shared address bus can limit the performance of a multiprocessor system with large numbers of processors. Suppose we add more interprocessor communication busses. The best we can do is a dedicated bus between each pair of processors.



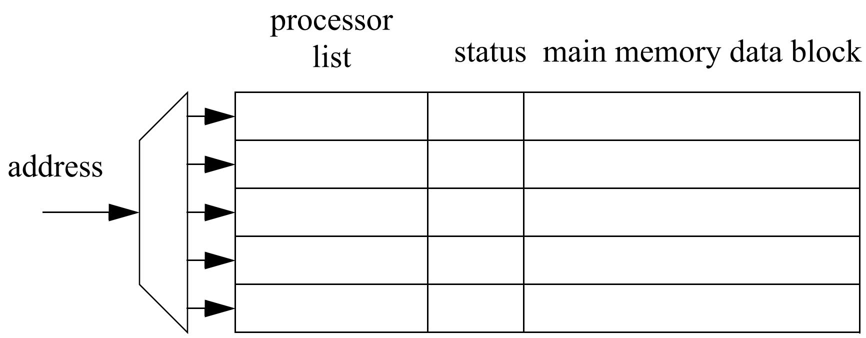
This is called a "fully connected" network. If there are N processors, then the total number of busses is

$$(N-1) + (N - 2) + ... + 2 + 1 + 0 = N(N - 1)/2$$

The best we can do is ~ $N^2/2$ times the communication bandwidth of a single bus.

If cache coherence is implemented with snooping, the bus read, bus write and bus upgrade (invalidate) messages must be broadcast to all processors. Each of the N processors will send N - 1 messages (one to each of the other processors) resulting in a total average message count ~ N^2. Clearly, adding more busses does not help if a cache coherence protocol is used which requires broadcasting messages to all processors. We need a coherence scheme that takes advantage of invalidate protocols not needing to broadcast messages.

Directory Based Cache Coherence. Suppose we add extra bits (the directory) to main memory to keep track of which processor caches have copies of each memory block. We can also keep track of the status of each main memory block with a directory controller similar to the cache controller for each processor. The organization of main memory with directory is as follows.



The processor list must be compact to keep from increasing the memory size too much. Usually, a single bit per processor is used in the list to indicate whether or not a processor has a copy. Even so, the number of bits in the list can get excessive when there are hundreds of processors.

The status bits of the directory represent one of the following four states.

- Uncached (U) - none of the processor caches has a copy. The main memory data block is valid.
- Shared (S) - each processor in the list has a copy in the shared state. The main memory data block is valid.
- Exclusive (E) - one and only one processor has a copy in the exclusive (unmodified) state. The main memory block is valid.
- Modified (M) - one and only one processor has a copy in the modified state. The main memory block is invalid.

Observe that:

1. The cache controller responds in the same way as in fig. 9-6, p. 428. The difference is that the bus read, bus write and bus upgrade (invalidate) messages are sent to and come from the directory controller, not other cache controllers.
2. A bus read request to the directory for a block in the M state requires the directory to send a message to the single processor cache that has a valid copy of the data. This extra "dirty miss" latency can be a problem for some data base applications.
3. Other bus read requests can be handled by the directory (rather than loading down the processor cache controller) since the main memory data block is valid.
4. A bus write or bus upgrade (invalidate) request requires messages to be sent only to processors in the list (no broadcasting).

Since all bus read, write and invalidate operations must be handled by the directory, contention for the directory can easily become a problem. For this reason, most directory schemes spread the main memory locations between the processor nodes (the NUMA architecture fig. 9-4, p. 424) with a separate directory controller for each processor.