CSE 400 – Related Work: Instructions & Example

The "Related Work" section should **synthesize** your literature review. You must cite and discuss references gained from at least 3 different sources. These could include technical or scholarly journals, conference papers, books, textbooks, patents, technical reports, theses/dissertations or websites. Feel free to consult the librarians in the engineering library, who will be happy to assist you in your research.

Your review should:

- Summarize existing research, products and systems
- Talk about trends in the field
- Discuss research themes that emerged from your review
- Place your work in the context of cited work
- Explain why the proposed project is better than or different from what already exists.

Sample Related Work Section

Note: The in-text references should follow APA Style. See class Blackboard site for information on APA Style.

From "Screwdriver:" An HDL implementation of an advanced cache coherence protocol
Presented here by permission of the author, Peter Hornyack. © 2007.

2.1 The cache coherence problem

The cache coherence problem and proposed solutions to it have existed as long as multiprocessors themselves have. Researchers quickly realized that the full potential of multiprocessors could only be realized if each processor core had its own local memory. Configurations in which multiple processors accessed one common memory or memory hierarchy had limited scalability, with the shared memory bus between the processors and the memory being inundated with requests and becoming a system bottleneck (Papamarcos, Patel, 1984). While the use of local caches decreased the number of requests to shared memory, it introduced the problem of cache coherence.

The simplest kind of protocol for cache coherence is a broadcast-invalidate protocol, in which the address of a written cache block is broadcast to all processors. If a processor has that block in its cache, it invalidates the block so it cannot be used until it is updated with the current, correct data (Papamarcos, Patel, 1984). This kind of protocol is also known as a "snooping" protocol, because each processor has to "listen" at all times for addresses that are being written to. The most basic broadcast-invalidate protocol is the modified-invalid, or MI, protocol, in which a cache block can have one of two states, either modified (up-to-date) or invalid (out-of-date). Only one processor can have a copy of a block in the M state at any time, ensuring that no processor uses an out-of-date cache block. A more efficient protocol is the modified-shared-invalid, or MSI, protocol. The MSI protocol's modified and invalid states are the same as the MI protocol, but an additional "shared" state allows multiple caches to have read-only access to a cache block at the same time (Martin, Roth, 2005).

2.2 Traditional cache coherence protocols

A “coherence message” is a processor’s request to either read or write a block that is not currently in its cache (or a block that is in its cache, but that it does not have the proper permissions for). In a multiprocessor system that uses cache coherence, other processors must receive these messages and invalidate or otherwise change the state of their own cache blocks as necessary. Various forms of broadcast-invalidate protocols transmit their coherence messages
through the multiprocessor system in different ways. Perhaps the most obvious way is using a full interconnect network, with wires directly between each pair of processors (and main memory) for broadcasting invalidate addresses. The problem with a full interconnect network is that it creates race conditions (Martin, Hill, Wood, 2003). For example, two processors may broadcast coherence
messages one after the other, but due to interconnect delays and contention for resources (i.e. main memory), the later message could reach another processor before the earlier message and lead to incorrect data use.

Two primary methods have been proposed to solve the problem of coherence message races with a broadcast-invalidate protocol. The first way is to broadcast coherence messages on a shared bus, rather than an interconnect network, with the shared bus providing “total ordering” for coherence messages; that is, the shared bus ensures that all processors and memory see the same coherence messages in the same order. A shared-bus protocol solves the problem of races,
but providing total ordering can be complex to implement, and the shared bus has lower bandwidth and more limited scalability than an interconnect network (Martin et al., 2003).

A second way of avoiding coherence message races is to use a “directory” protocol. In a directory protocol, a high-bandwidth, easily scalable interconnect network is still used, but each cache block has a “home” processor that knows which other processors are holding the cache block and must be sent invalidation messages. In addition to reducing the amount of bandwidth required for sending coherence messages (because the messages are sent only to the processors
holding the block, and not to every processor), the home processors in a directory protocol also provide the ordering needed to eliminate coherence message races. However, directory protocols increase memory access latency because an extra level of indirection, the visit to the home processor, is required (martin, Roth, 2005). The ordering provided by the home processors can also be complex to implement.

2.3 Non-traditional coherence protocols

Recently, more advanced coherence protocols have been proposed that attempt to incorporate the advantages of the traditional methods while improving upon their limitations. One such method is "token" cache coherence. Token coherence uses a direct interconnect network that provides greater scalability than a shared bus, but avoids the high-latency level of indirection required by directory protocols while still avoiding races. The protocol is called token coherence because "tokens" that are associated with each cache block are passed between processors and memory and counted to ensure coherence. Each cache block is initialized in main memory to have T tokens, where T is equal to the number of processors in the system. If a processor has a cache block with zero tokens, then the block is invalid and can be neither read nor written. A cache block with one or more tokens can be read by that processor. For write access to a cache block, the processor must have all T tokens associated with that block. These three conditions correspond to the three states of the MSI protocol, invalid, shared and modified. Enforcing these token-counting conditions eliminates the need for total ordering, so races cannot occur and coherence is ensured (Martin et al., 2003).

Another advanced coherence protocol is used in Opteron (a.k.a. Hammer) chips produced by AMD. The Hammer coherence protocol also uses a direct interconnect network, but each processor has its own integrated "memory controller" module. When a processor needs access to a cache block, its memory controller directly broadcasts a coherence message to every other
memory controller (including main memory), then waits for each memory controller to acknowledge its request, providing current data if necessary, before proceeding (Keltcher, McGrath, Ahmed, Conway, 2003). In this approach, ordering is provided by each memory controller, while still using a highly scalable, low-latency interconnect network.

2.4 "Screwdriver" protocol implementation using HDL for FPGAs

The goal of this project was to implement an advanced cache coherence protocol in a hardware description language (HDL) suitable for use on an FPGA. The protocol, named the "Screwdriver" protocol (inspired by AMD's Hammer, among other factors), was designed to be an improvement over shared-bus and directory protocols, incorporating aspects of the nontraditional protocols described above, while being easily scalable and avoiding races. In the
Screwdriver protocol, a single memory controller sits at the center of an interconnect network between all of the processors in a multiprocessor system. The role of the memory controller is very similar to that of the memory controllers in the Hammer protocol: it receives cache block requests from all of the processors, sends coherence messages to all of the other processors, and
upon their return, completes the original processor's request. Having just one memory controller in the system, instead of one for each processor, limits the bandwidth of the memory hierarchy but reduces the number of possible race conditions. The Screwdriver protocol also avoids having to keep track of a lot of state information, unlike the token coherence protocol, which must count
the number of tokens with every cache block, including those residing in main memory. Finally, using an interconnect network instead of a shared bus allows the protocol to be easily scalable to a large number of processors.

The multiprocessor design and Screwdriver coherence protocol were implemented in Verilog HDL to allow for realistic simulation, as well as the possibility of the design being downloaded onto an FPGA to physically instantiate the protocol. The HDL code was

written to easily allow different numbers of processors to be used in the multiprocessor system, so the objective
performance and limitations of the Screwdriver protocol implementation could be evaluated for systems of up to 16 processor cores.