Advanced Computer Architecture

Homework # 4

Submitted by: Maliha Arif
PID: 4506817
Submitted to: Dr. Dan C. Marinescu
Submitted on: 15th November 2018

[Question 5.25]

One proposed solution for the problem of false sharing is to add a valid bit per word. This would allow the protocol to invalidate a word without removing the entire block, letting a processor keep a portion of a block in its cache while another processor writes a different portion of the block. What extra complications are introduced into the basic snooping cache coherence protocol (Figure 5.6) by this addition? Consider all possible protocol actions.

Answer:

False sharing is a problem which arises from the use of invalidation based coherence protocol where there is a single valid bit per cache block and not per word. It occurs when a block is invalidated because some word in the block is being written into, other than the one being read and a miss occurs.
So, the proposed solution where a valid bit is added per word would work but subjected to different complications. For example, when monitoring or snooping the bus, it would be important not to only match the tag of the cache block but also the offset within the block which tracks the word by adding few more bits.
At the same time, we would have to equip the cache to support write-back of partial cache blocks, this would ensure only valid words are written to memory because invalid words are probably not coherent with the system.
Lastly, we have a finite state machine defined for a cache block. And we know how it works, but somehow, we need to modify it so that it can work with reference to words inside a cache block and not just for the cache block. The easiest way to do this would be to provide the state information for each word in the block. This would require much more than one valid bit per word. If not then we have to change the coherence protocol slightly.

[Question 5.26]

This exercise studies the impact of aggressive techniques to exploit instruction-level parallelism in the processor when used in the design of shared memory multiprocessor systems. Consider two systems identical except for the processor. System A uses a processor with a simple single-issue, in-order pipeline, and system B uses a processor with four-way issue, out-of-order execution and a reorder buffer with 64 entries.

a. [15] <5.3> Following the convention of Figure 5.11, let us divide the execution time into instruction execution, cache access, memory access, and other stalls. How would you expect each of these components to differ between system A and system B?

Answer:

Since both systems have different types of processors, one single issue, in order, the other out of order, 4-way issue, there is going to be difference in instruction execution time, cache excess time, memory access etc.

Cache access would speed up in system B but since cache accesses take longer than functional units latencies, they would need more instructions to be issued in parallel, this would overlap and give less latency overall.

Instruction execution component on the other hand would be much faster as compared to caches access in System B as there is out of order execution and multiple instructions can be issued simultaneously.

Memory access time will improve also in system B but as memory comprises local and remote memory together with cache to cache transfers, the increase won't be as much as with instruction and cache access. The 64-entry instruction window in this example is not likely to allow enough instructions to overlap with such long latencies. There is, however, one case when large latencies can be overlapped: when they are hidden under other long latency operations. This leads to a technique called miss-clustering that has been the subject of some compiler optimizations.

Lastly, other stalls like resource stalls, branch mispredictions will also improve.

b. Based on the discussion of the behavior of OLTP workload in Section 5.3, what is the important difference between the OLTP workload and other benchmarks that limit benefit from a more aggressive processor design?

Answer:

OLTP is Online Transaction Processing (OLTP) workload which consists of a set of client processes that generate requests and a set of servers that handle them. The server processes consume 85% of the user time, with the remaining going to the clients. Although the I/O latency is hidden by careful tuning and enough requests to keep the processor busy, the server processes typically block for I/O after about 25,000 instructions. Overall, 71% of the execution time is spent in user mode, 18% in the operating system, and 11% idle, primarily waiting for I/O. Of the commercial applications studied, the OLTP application stresses the memory system the hardest and shows significant challenges even when evaluated with much larger L3 caches. Hence, we can say the instruction miss and memory stall time are significant in OLTP workload compared to other benchmarks as both are not that well addressed by out of order processors, there for system B does not work well for it and give slower speed up.

[Question 5.29.]

Assume a directory-based cache coherence protocol. The directory currently has information that indicates that processor P1 has the data in "exclusive" mode. If the directory now gets a request for the same cache block from processor P1, what could this mean? What should the directory controller do? (Such cases are called "race conditions" and are the reason why coherence protocols are so hard to design and verify.)

Answer:

Since we know the question is pointing to a race condition scenario, we will look at it with that perspective and come up with a solution that does not result in deadlock as usually that is one of the consequence of race condition scenarios. Since P1 already has the data in exclusive mode but then requests for it again, it means that it evicted the cache block but requested it immediately in the next instruction/request. Now we know, writeback messages take longer than request messages and most likely, P1 evicted the block as it begun the writeback. Now an important thing to note is, that since we are dealing with out of order networks, the new request can arrive before the write back arrives at the directory hence one way to solve this is to simply wait for writeback to finish and then execute the read request or another option can be, the directory can just issue a negative acknowledgment and don't respond to the request at all.

[Question 5.30]

A directory controller can send invalidates for lines that have been replaced by the local cache controller. To avoid such messages, and to keep the directory consistent, replacement hints are used. Such messages tell the controller that a block has been replaced. Modify the directory coherence protocol of Section 5.4 to use such replacement hints.

Answer:

If we modify the directory coherence protocol and use replacement hints to update the directory controller whenever a block has been replaced, it would certainly help in reducing the invalidates being sent by the directory controller. This is how it can be accomplished. The replacement hints sent would help the home directory to remove that CPU or local cache copy from the sharing list of the block. So, when the block is to be written by some other CPU, there is no conflict and incorrect information in the directory controller.
This technique helps but we should keep in mind that overall protocol traffic is not reduced as replacement hints and invalidates consume same amount of bandwidth.