Quad Core MESI Protocol

The quad core MESI protocol provides a coherent view of the memory shared among the four processors and their L1 caches. The states of these processors and their caches are all replicated in my implementation. I was able to try my protocol on several examples focused on coherence.

*Coherence Correctness*: My first example, *testcoher,* involves two CPUs both attempting to load and store to the same two addresses. This test checks that the writes are performed exclusively and that the modified value is propagated.

CPU 0 loads the address X2+0x00 followed by a store of value 3 to the address X2+0x90. CPU 1 performs a load from X2+0x90 followed by a store of value 4 to 0x00. In the test, CPU 0 performs the load to 0x00 first transitioning from I to E and initiates the store to X2+0x90. Then, the load of CPU1 to X2+0x90 is stalled to maintain write exclusion, and retrieves the correct value of 4 when it completes indicating write propagation. The same write propagation can be observed in the other direction. This example though simple in comparison to the one provided with 4 CPUs illustrates both the key requirements for maintaining coherence.

*Coherence Overhead:* My second example, *falseshare*, illustrates the overhead that may occur due to coherent misses. This example, also involves two CPUs only. False sharing occurs (as explained in class) when two CPUs though always modifying disjoint parts of a cache line are apparently forced to endure coherence misses due to architectural limitations. Specifically, sharing is performed at the granularity of cache line since maintaining valid and dirty bits for each addressable unit in a cache line may impose too much control overheads. And, this will also complicate the logic. In this example, the CPU 1 stores at X2+0x80 to X2+0x84 whereas the CPU 2 stores at X2+0x88 to X2+0x90. The common shared address range between the two CPUs is X2+0x80 – X2+0x90. To amplify the coherence penalty in the test case, I execute both these access in a loops. The test output shows CPU 1 and CPU alternatively getting invalidated and then transitioning to M to perform the store.