Elec/Comp 526
Spring 2018
Project 6: MOESI Snooping Cache Coherence Protocol

This assignment deals with the implementation and evaluation of the MOESI snooping cache coherence protocol. It is similar to project 5 where you designed the BusSnooper for the MSI protocol. You are required to design the Bus Snooper process to include the O and E state. The LookupCache and HandleCacheStateChange functions in the file mem.c have *already been extended to include the O and E state*. The project files are available on CANVAS in the tar file: project6.tar.gz

The lecture notes provide an overview of the MOESI protocol. The structure of the files are the same as Project 5. You will only need to add the BusSnooper function in bus.c and to change parameters in global.h. You will experiment with two different workloads: the traces will be generated by two different trace file generators: arrayupdate.c and sharedarray.c

All the code for the cache backend controller except that for BusSnooper process is provided to you. Go through the files and understand how the system works. Note that the implementation of HandleCacheStateChange has been slightly changed from the implementation of MSI. *In particular, the updates of the cache state must be done by you in your BusSnooper process.* In the MSI implementation, the new state of a block in a processor's cache following its own bus transaction, was handled by HandleCacheStateChange.

The MOESI protocol needs to make decisions based on a global signal that wire-OR's the "present" bit set by processors to indicate whether or not the block referred to in a bus transaction is present in their cache. This signal called P is computed by a function wireOR provided in bus.c.

I. Implement the BusSnooper function to implement the backend functions for the MOESI protocol.

Experiment and Report

Once you have a working BusSnooper implementation, you will use the simulator to evaluate two workloads.

Experiment 1

Use the workload obtained by compiling and executing arrayupdate.c. Note that for this workload, the array of size TOTALSIZE is divided equally among the processors.

Parameters: TOTALSIZE: 8192, MINDELAY: 0, Chunk Partitioning, CACHE SIZE: 1 Block, MEM_CYCLE_TIME: 100.0 cycles, CACHE TRANSFER TIME: 10.0 cycles

I. For NUM_PROCESSORS = 1 and 2

Vary MAX_DELAY between 0 and 20 (in steps of 2) and record the total execution time of the MOESI protocol. (Note: Don't forget to recompile and re-execute arrayupdate.c and your simulation program files for each value simulated).

II. Repeat step I for the MSI protocol (use your setup of Project 5) with the same parameters as above.

III. Graph the Total Execution Time vs Delay with NUM_PROCESSORS = 2 for both the the MSI and MOESI protocols on the same plot. (PLOT 1A)

IV. Graph the Speedup (for two processors) vs Delay for both protocols on the same plot. Note that speedup using p processors is defined as the time required using 1 processor divided by time with p processors. (PLOT 1B)

V. Explain your results. Why does one protocol seem to do so much better for this workload.

Experiment 2

Use the workload obtained by compiling and executing sharedarray.c. Note that for this workload every process has a trace of size TOTALSIZE.

Parameters: TOTALSIZE: 8192, MINDELAY: 0 and MAXDELAY: 10, Chunk Partitioning, CACHE SIZE: 1 Block, MEM_CYCLE_TIME = 100.0 cycles, CACHE TRANSFER TIME =10.0 cycles

I. Vary processors NUM_PROCESSORS for the values: 1, 2, 4, 8, and 16 and record the total execution time of the MOESI protocol. (Note: Don't forget to recompile and re-execute both sharedarray.c and your simulation files for each value simulated).

II. Uncomment the statement "if (mode=0) NUM_PROCSSORS -1" in sharedarray.c, so that the trace of the last processor is 100% writes.

III. Repeat step I with the modified trace.

IV. Graph the Total Execution Time vs NUM_PROCESSORS for both the experiments of steps I and II on the same plot. (PLOT 2)

V. Explain your results in PLOT 2.