CSE 471: Computer Design & Organization
Assignment 4
Due: Tuesday, June 3

This assignment gives you the opportunity to design and evaluate your own cache coherency protocol.

For this assignment you need to work in teams of two – no singletons.

Part I: Cache coherency protocol design

In class we studied a simple 3-state snooping cache coherency protocol. Although two bits are needed to encode the coherency state in this protocol, only three of the four possible values are used: invalid, shared, and exclusive. This begs for a coherency protocol which utilizes the fourth value to implement a fourth state that improves multiprocessor performance in some way.

In this project you are to devise a fourth state and present hypotheses that support your choice. For example, in what situations should it improve performance? Why do you think those situations will occur often enough to make a performance difference (Amdahl's Law applies here again)? Does your new coherency state have any downsides? And so forth. Design a finite state machine that carries out the functions of the protocol with the new fourth coherency state, both from the CPU and the snoop points of view. In your report, represent this FSM as nodes and arcs like we did in class.

Part II: Cache coherency protocol implementation

Implement your new protocol with its fourth state. For this work you will use SESC, a multiprocessor simulator from the University of Illinois (Luis Ceze, our latest architecture hire, and Karin Strauss, an affiliate faculty member working at the Seattle AMD lab were authors). SESC has been built to do research studies similar to the one you are about to do. SESC models an SMP built around the MIPS Instruction Set Architecture. This simulator provides processor and memory subsystem configuration parameters like SimpleScalar, as well as multiprocessor specific parameters, such as the number of processors and control over the directory that handles cache coherency. Andrei will discuss SESC and give you a tour through the code.

Evaluate your new protocol relative to the three-state protocol that is already implemented in SESC. Don't just say the performance difference is X -- design and implement metrics that show ***why*** one protocol is better than the other. What aspects of performance are improved or worsened?

We have provided two programs to evaluate the protocols: crafty (a chess simulator benchmark from SPEC2000) and MCF (single-depot vehicle scheduling optimizer, also from SPEC 2000). In addition, you may need to write small test cases to test and debug particular aspects of your protocol, to make sure they do what they are supposed to and don't do what they are not supposed to do. The completeness of these tests will be a component of your project grade.

In addition to the protocol, you should implement a "global coherency state monitor" that tracks the effects on the entire multiprocessor of changes brought about by reads and writes to shared data. This monitor guarantees that mistakes in your protocol are caught immediately, saving you a lifetime of debugging. I'll explain the monitor in more detail after Andrei's SESC discussion. In addition to the global monitor, you should use the diagnostic output already provided by SESC, such as coherency traffic on the bus and changes to the cache line state, etc.

Part III: The report

Turn in the fruits of your labor in the form of a report. All code and configuration files should be submitted to Andrei via email. (Please do NOT modify the same configuration file for multiple tests.)