CSE 471: Computer Design & Organization
Assignment 4
Due: Thursday, June 3 at midnight
(There will be milestones before that.)

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? Is so, what are they? 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 mini-report, represent this FSM as nodes and arcs like we did in class. The mini-report should also contain your hypotheses.

This is your first milestone, due **Thursday**, **May 6**.

Part II: Cache coherency protocol implementation

Implement your new protocol with its fourth state. For this work you will use MultiCacheSim, a multiprocessor cache simulator built by Brandon Lucia, a graduate student in CSE. MultiCacheSim has been designed to do research studies in multiprocessor caches and cache coherency, similar to the one you are about to do. It therefore models the cache subsystem of a CMP in detail and for the sake of short simulations times omits modeling the rest of the processor. MultiCacheSim provides processor and memory subsystem configuration parameters like SimpleScalar (block size, associativity in number of sets, cache size), as well as multiprocessor specific parameters, such as the number of processors and the choice of cache coherency protocol. The default value for this is the three-state protocol we will study in class; you should provide a different value for this flag to indicate that the simulator should run your protocol. The 3-state protocol that you will use as your base has already been defined. In the simulator's README file there is detail on how to configure your new protocol. Fran will discuss MultiCacheSim and give you a tour through the code in section on **April 29**.

Evaluate your new protocol relative to the three-state protocol already implemented in MultiCacheSim. Don't just report the bottomline performance -- design and implement metrics that show ***why*** one protocol is better than the other. What aspects of performance are improved or worsened? MultiCacheSim already provides basic read and write metrics, such as the number of read/write hits and misses, the number of cache-to-cache transfers, and the number of writes to data in the *shared* state; you may need to add to this list to come up with a robust analysis. Devising a good set of component metrics that you can use to explain the bottom-line performance is a component of your project grade.

The first time I implemented a coherency protocol, I found it useful to begin the simulations with just one processor, then two, then three, then the sky.

MultiCacheSim was built with the PIN binary rewriting tool and can simulate any parallel x86 applications. There are two research benchmark suites for parallel applications, which you can use to show off your new coherency protocol: PARSEC and SPLASH2. Both are available on the web, but Fran will put them in the class directory. I would suggest using PARSEC, as it is more modern and includes a script that will let you run the entire suite in one fell swoop. 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 also be a component of your project grade. To further help with debugging, Fran will provide a global monitor that tracks the effects on all caches of changes brought about by each read and write to shared data. The monitor ensures that each read and write results in a correct coherency state for all caches, thereby catching mistakes in your protocol immediately, saving you a lifetime of debugging.

This is your second milestone, due **Thursday**, **May 27**.

Part III: The report

Turn in the fruits of your labor in the form of a report. The report should be in the format of the sample report on our web site. All code and configuration files should be submitted to Fran through a drop-box.