Announcements

• Assignment 2
  - Waypoint on 3/11
  - Due on 3/28
Assignment 2 Objectives

• Learn to design a CC protocol
  ❑ Come up with a state transition diagram.

• Learn formal verification language (Murphi)

• Describe your CC protocol in Murphi and verify it

• Requirements
  ❑ Verify with at least 3 processors, 1 memory location
  ❑ Connected via an arbitrary interconnect
    ○ Network can reorder messages arbitrarily
    ○ Infinite buffers for this assignment
    ○ Multiple lanes (as many as you decide you need)

• Directory based memory unit (Directory collocated with memory).
Grading

- Waypoint – 10%
- Correctness – 60%
- “Quality” of invariants & base protocol – 10%
- Optimization correctness – 10%
- Optimization difficulty – 10%
Murphi

• "Protocol Verification as a Hardware Design Aid," David L. Dill, Andreas J. Drexler, Alan J. Hu and C. Han Yang, 1992

• Formal verification of finite state machines
  □ State space exploration – explores all reachable states
  □ Tracks queue of “to-be-explored” states
  □ Keeps giant table of all previously visited states
  □ Canonical representations & hashing make it efficient
  □ Exploits symmetry to canonicalize redundant states
Murphi Language

• Looks sort of like Pascal
• User-defined data types & structures
• “Rules” indicate non-deterministic steps between states
• “Invariants” and “asserts” confirm protocol correctness
• “Scalarsets” and “multisets” data types capture symmetry
State Space Exploration

- Identify states.
  - Stable and transient both

- Actions:
  - Identify actions
  - Prerequisite for an action to happen?
  - What is the outcome?

- Invariants:
  - Why we need invariants?
How to begin?

- Download Murphi.tar from the class website.
- Follow Readme on instructions to compile and run.
- Make sure sample protocol twostate.m works.
- Play with the toy programs and learn Murphi language.
- Read UserManual
Designing a CC Protocol

- MSI Base Protocol
- Figure out different message types needed.
- Nack-free → More difficult
- Allow silent drop of clean data or maintain precise sharing?
  - What are the implications.
- How many protocol lanes needed?
- Figure out all the transient states required for processors and directory
- One optimization over your base protocol
Misc.

• Read the assignment well to make sure you have understood everything in the specification

• Start early.
  - Order of magnitude harder than the 1st Assignment.

• Keep the protocol as simple as possible
  - Easy to debug
  - Avoid running into memory and runtime problems

• Compile at every addition you do in the code
Murphi Samples

• Pingpong.m
  □ A two-player ping-pong game

• Twostate.m
  □ A four-hop two-state valid-invalid coherence protocol
  □ (Also a good starting point for your project)
My solutions

2-state VI protocol w/ 3 procs, 2 values (444 lines of code)
• This sample is included in the distribution
• 259 states; 0.1 s to verify on this Mac

4-hop MSI NACK-free w/ 3 procs, 2 values (615 lines of code)
• 5034 states; 0.4 s to verify on this Mac

3-hop MSI NACK-free w/ 3 procs, 2 values (722 lines of code)
• 72838 states; 4.1 s to verify on this Mac
• Needs 256MB state hash table
Optimizations (easy to hard)

- Self-downgrade (spontaneous M→S)
- MESI, directory may provide E in response to reads
- Migratory sharing optimization
- Add an owned state
- Cruise missile invalidations
- 2-hop speculative requests
- Occupancy-free directory
- 2 directories with directory migration / delegation
- SCI-style distributed sharer lists