EECS 570

Designing Cache Coherence Protocol using Murphi

Winter 2017

Prof. Thomas Wenisch

http://www.eecs.umich.edu/courses/eecs570/

Slides developed in part by Profs. Adve, Falsafi, Hill, Lebeck, Martin, Narayanasamy, Nowatzyk, Reinhardt, Roth, Smith, Singh, and Wenisch.
Cache Coherence

• Why?
  - In presence of caches, orchestrate access to shared memory in a multi-core system

• What?
  - Read returns most recent value written
  - For a single location only

• How?
  - Well, many many flavors!
Cache Coherence – How?

- Interconnection network
  - Bus: Snoop-based protocols
  - Point-to-point: Directory-based protocols

- Stable states?
  - VI, MSI, MESI, MOSI, MOESI

- Optimizations employed – countless papers!!
  - 3-hop vs 4-hop
  - Self-downgrade (M->S)
  - Cruise-missile invalidations etc.
Deadlock!

• Protocol deadlock
  □ Wait for a message that is never sent
  □ **Solution:** Design your state machine correctly

• Network deadlock
  □ Coherence messages hold resources in circular manner
  □ **Solution:** Dedicated virtual networks for different messages
Virtual Networks

• Solve protocol dependent deadlocks
  ☑ Have separate VN for every message class

Source: A primer on memory consistency and cache coherence
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%
  - Will evaluate this by changing some cases and check if invariants fail
- 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?
Murphi Samples

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

- Twostate.m
  - A four-hop two-state valid-invalid coherence protocol
  - A good starting point for your project
How to begin?

- Download murphi_eecs570.tar.gz from the class website.

- cd Murphi3.1/src
  - make mu

- Compilation
  - cd Murphi3.1/eecs570_sample
  - ./mu twostate.m
  - make twostate
  - ./twostate
    - No error found.
    - State Space Explored: 259 states, 894 rules fired in 0.10s.
Murphi-Misc.

• Start early.
  □ Order of magnitude difficult than the 1st Assignment.

• One change at a time
  □ Start simple, add incrementally
  □ Compile at each step

• Memory
  □ You will soon run out of default memory allocated for Murphi
  □ Use: m<n>, n kilobytes while running executable

• Read User manual
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
3-Hop MSI protocol

How you think it should look like
3-Hop MSI protocol

How it really looks like

Transient states
Stable states

*My solution
MESI w/ self downgrade on 4 procs

What you end up implementing
My solutions

• 3-hop MSI NACK-free, 3 procs
  - 30161 states; 121984 rules in 1.48s

• MESI w/ self-downgrade, 4 procs
  - 1453082 states; 7670997 rules in 130.14s

*Number of states explored will be different for your implementation*
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

Talk to Prof. Wenisch or me if you want to do something else
Deliverables

- Waypoint report <uniqname>.pdf (Due on 3/13)
- Final submission (Due on 3/27)  
  <uniqname>.zip
  msi.m  msi_opt.m  msi.out  msi_opt.out  report.pdf

- When I say .zip, I mean .zip and NOT .tar or .7z
  - Stick to file names (lowercase) and directory structure

File descriptions

- msi.m: Baseline MSI, turn off optimization
- msi_opt.m: MSI protocol with optimization
- msi.out: Murphi output for baseline MSI
- msi_opt.out: Murphi output for MSI with optimization
- report.pdf: As per assignment specification. As always should not exceed 2 pages excluding protocol diagram
All the best!