Foundations

What is the meaning of shared memory when you have multiple access ports into global memory?

What if you have caches?

Sequential consistency: Final state (of memory) is as if all RDs and WRTs were executed in some fixed serial order (per processor order also maintained) → Lamport

[This notion borrows from similar notions of sequential consistency in transaction processing systems.]
Foundations

A hardware designer's physical perspective of sequential consistency

Key: Using fence to wait until flush is done is the key mechanism that guarantees sequential consistency

We will revisit this in more detail shortly

One other cache nasty to watch out for

Flush foo* from cache, wait till done

Does it always work?
One other cache nasty to watch out for

Cache line

Flush foo* from cache, wait till done

Correct final value: 🟢

Wrong final value: 💚

Problem called “False Sharing”
Leads to bugs with sw coherence
Leads to poor perf. with hw coherence

Solutions?
Pad shared data structures so multiple shared items do not fall into same cache line

Summary of New Multicore Instructions

• Send message
• Receive message
• Synchronization
  - Barrier
  - Test and set
  - F&A and relatives (e.g., F&Op, CmpXch)
• Flush cache line
• Memory fence
Outline

Memory architecture
Cache coherence in small multicores
Cache coherence in manycores

Recall, Shared Memory
Algorithmic Model

Shared Memory

P
P
P

... P
**Shared Memory Structures in Parallel Computers**

**Monolithic**
- Memory
- Network
- P
- C
- C
- C
- P
- P

**Distributed**
- Network
- M
- M
- M
- M

**Distributed - local**
- Network
- C
- M
- M
- C
- M
- C
- P

But, what about multicores chips?

**Shared-Memory Structure in Cutting Edge Multicores**

**Multicore Chip**
- Memory
- Ring

**Distributed**
- Memory
- Network
- P
- C
- C
- C
- P

Like legos, can move Ps, Cs and Ms around
Shared-Memory Structure in Cutting Edge Multicores

Tile processor
64 cores

Caches and Cache Coherence

Network
A World Without Caches

With Caches
How are Caches Different from Fast Local Memory (SRAM)?

Network

versus

Network

Discuss

Key insight
why use a cache when local mem exists

Anatomy of a common case LD operation

LD A

HW: 1 cycle
SW: 10 cycles

When done in HW, we call the store a cache!

If A replicated in local store
then fetch from local store

Else send message to get A from DRAM

HW: 100 cycles
SW: 110 cycles

Can do all of this in hardware too. This is what typical caches do
Cache Coherence Problem

Coherence problem

Solving the Coherence Problem

- Small multicores
  - Software coherence
  - Snooping caches
- Manycores
  - Software coherence
  - full map directories
  - limited pointers
  - chained pointers
    - singly linked
    - doubly linked
  - limitless schemes
  - Hierarchical methods

We will study
  Coherence structures
  Coherence protocols
  Cache side state diagrams
  Directory side state diagrams
Software Coherence
Saw this before

MEM

foo1
foo2
foo3
foo4

flush fence flush fence

GET_foo_LOCK

MUNGE

Flush foo* from cache
Fence: wait till changes that result from flush
are visible to everyone

RELEASE_foo_LOCK

Can stick the locking,
flushes and fences in library code
to provide clean abstractions

Hardware Cache Coherence
Snooping Caches

• Works for small multicores (mem off chip)
• Broadcast address on shared write
• Everyone listens (snoops) on bus/ring to see
  if any of their own addresses match
• Invalidate copy on match
• How do you know when to broadcast,
  invalidate
  - State associated with each cache line
  - Key benefit: no global state in main mem

Let's look at this in more detail next...
Hardware Cache Coherence
Invalidation versus Update Snooping Caches

- Broadcast address on shared write
- Everyone listens (snoops) on bus/ring to see if any of their own addresses match
- If address matches
  - Invalidate local copy (called invalidate or ownership protocol)
  - OR
  - Update local copy with new data from bus (writer must broadcast value along with address)

Only a cache side state machine needed

Discuss paper

Update versus Invalidate Protocols

Tradeoffs between
- Update protocols
- Ownership protocols

Update better when poor write locality
Invalidate better otherwise

Competitive snooping idea --
- Do write updates
- If more than a "few" updates, then use ownership
  "Few" ⇒ Switch mode when cost of all updates so far = cost of invalidation

The cost of this approach is no worse than twice the optimal (try to prove this)
"Competitive algorithms are cool"
State diagram for ownership protocols

For each address $a$

"Invalid"

Cache side state machine

Store state with cache tags

"Shared"

"Modified"

For each address $a$

Assume cache blocksize is one word for now; Let's deal with the cache block complexity later

"MSI"

Variants such as MESI, MOESI

Snooping Caches Definitions

Shared Memory

Bus or Ring

Ext. bus request

My bus response

Broadcast

My local request

My local response

Update

cache tags

cache

Match

Dual ported

Processor

Processor

write

snoop
State diagram for cache block in ownership protocols

In ownership protocol:
writer owns exclusive copy

State diagram for update protocols
Maintaining coherence in manycores

• Software coherence – saw this before

• Hardware coherence
  > full map directories
  > limited pointers
  > chained pointers
    • singly linked
    • doubly linked
  > limitless schemes
  > Hierarchical methods