****
Review -- 1 min
****

coherency – can you tell that memory system is playing tricks by looking at one address?
consistency – can you tell that the memory system is playing tricks by looking at multiple addresses?
Directory-based coherency

-- scalable

- challenges – out-of-order msgs, limited buffering, …

tools for cache design

example: used teapot yesterday

/*********************************
Lecture
*********************************/

Consistency

Coherency v. Consistency

QUESTION: what is the difference?

Reminder of motivation
Remember definition of coherency:

The problem of Coherency

Time	Event	Cache A	Cache B	Mem
0				1
1	A read X	1		1
2	B read X	1	1	1
3	A write 0 to X	0	1	0
4	B read X	0	1	0

Coherency:

- Any write must eventually be seen by a read
- All writes seen in order

Example (from above) violates first rule – absent coherency via snooping or directories – B could read X==1 indefinitely
Consistency goes to the question of what does "eventually" mean?

We could definite it in terms of time – "within 1 us" or whatever
We don't do that

Instead we assume

- communication between processors via memory
- causal model

    if event X precedes event Y at processor 1, we expect processor 2 to observe that X precedes Y

In our example – if, after P1 updates X, it sets a flag Y saying "I've finished updating X", if P2 reads Y and seems that X has been updated, it had better see the new value of X when it reads X

Simple Consistency Example

```
P1:   A = 0;                     P2:   B = 0;
      …                                  …
      A = 1;                           B = 1;
      if(B == 0) {                     if(A ==  0){
         printf("P1.");                   printf("P2.");
      }                                      }
```

Output   Legal ("Consistent")**?**
"P1."
"P2."
""

“P1.P2.”
“P2.P1.”

Why might this be a problem?
Write buffer.
In big MPP – read could go through network faster than write invalidates

Consistency Example

P1:

```
for(ii = 0; ii < 100; ii++) {
    A = ii;
    B = ii;
}
```

P2:

```
while(1){
    printf(“(%d, %d), ”,
        A, B);
}
```

Output: (*Where is inconsistecy*?)
(0,1), (1,1), (1,2), (4,8), (9,9), (9,10), (9,10), (10,10), (11,11), (12,12), …

slide: consistency example

Consistency
notice memory system playing tricks by looking at multiple locations
want: observe updates in consistent/causal order

Consistency – goal: present consistent (‘causal’) view of memory

Implementing consistency

Analysis of Examples

In first example – two writes by one processor must be observed in same order at another processor
(Fairly obvious definition of causality)

In second and third example – the order between writes and reads must be maintained
(Less obvious?)

→ consistency involves ordering both writes and reads

sequential consistency

reads and writes by a processor are observed in same order that they are executed by a processor

timesharing model – global pattern of reads and writes corresponds to some possible interleaving of sequential processor executions

simple implementation – delay each memory access until the previous one has completed

Problem: write buffers,
lockup free caches,
reads must wait for writes to complete,
can't pipeline memory system, …

→ SLOW

Solution: Weaker consistency models

- allow out-of-order memory accesses (sometimes)
- synchronization operations enforce order when needed

TSO (total store order aka processor consistency)

motivation: processors have write buffers

TSO Consistency model:

- allow reads to bypass writes
- writes complete in order
- write barrier – forces synchronous write flush

Partial store ordering – allows overlap/pipelining of writes
Weak ordering – allow reads and writes to get out of order

Programming Model : Synchronization

We want relaxed consistency models for performance, but how do we avoid confusion like examples.

Think about parallel programming model –

- shared variables protected by locks/monitors
  (even in sequential consistency environment)

- While I hold a lock, no other processor should be reading the things I'm writing anyhow!
- While I don't hold a lock, I should not be reading things that other nodes are writing.

data race free programs

Every write of a variable by one processor is separated from a read/write of that variable by another processor by a pair of synchronization operations – a **release** (unlock) by the first and an **acquire** (lock) by the second.

For TSO, PSO, weak ordering – add acquires and releases that act as read and write fences.

Write fence

- all writes by P that occur before P executed the write fence complete before the write fence completes

- no writes by P that occur after the fence are initiated before the fence completes

Read fence similar

Memory fence == both

Release Consistency

TSO, PSO, weak ordering – treat each synchronization as a memory fence

Release consistency distinguishes acquire from release

FIGURE 8.40

Sa → W
Sa → R
R → Sr /* Error in text */
W → Sr

A range of consistency models implemented in hardware:

FIGURE 8.39

Performance: figure 8.41

Beyond release consistency

    Lazy release consistency (figure 8.40)

    specific locks w. specific data

Implementation Complexities

    Atomic lock operations:
    - test and set
    - load-linked and store conditional

    Efficiency
        spin locking
        exponential back off
        queue locks

/*****************************
Admin
*****************************/
M: project office hours
Q&A: Wednesday class
Exam: Wednesday evening 6-9 PM TAY 3.144
F: advice on technical writing and speaking; course evals
M-W: project presentations

****
Lecture - 24 min
****

The future: Large-scale commercial machines

Limits of bus-based SMP

    SGI slide – mem size increasing faster than system BW

    Papadopolous slide -- # nodes per SMP will fall

Solutions
----

1) Better busses
    - wider busses – more parallelism
        - today 256 bits data, separate address, data busses
        - DA: cost – more pins
            - note: cost scales with size of biggest system you want to build

    - more busses
        - busses not ***inherently*** unscalable
        - e.g. each processor snoops 2 or 4 busses
        - DA: cost – more pins, more tags, more controllers

    - fundamental limitation – max length f(speed of light)
        - → hard to increase frequency

2) Crossbar

<picture>

e.g. mainframes, Cray T90, Sun E10000

e.g. Sun E10000

Sun E5000 – 2 GB/s bus – 2-16 processors

Sun E10000 – 12 GB/s crossbar – 2-64 processors

16x16 crossbar; 16-bytes wide (128 bits)

Question – how do you maintain consistency

A: (in Sun E10K) – crossbar for data, 4 busses for addresses

each address bus goes to 1 of 4 banks of memory

all address busses snooped

DA: cost – typically crossbar is a unit
e.g. Sun E10000 - crossbar is $375K (list) (1997)

nprocessors	list	list/processors
4	500K	125K
8	625K	78K
16	875K	55K
32	1375K	43K
64	2375K	31K

3) CC-NUMA

Sun E5000 hardware looks like NUMA – they just don't quite go all the way
(good bet next generation will)

A few machines do CC-NUMA
Convex Exemplar, DASH, FLASH, Alewife,

SGI Origin 2000 --
Directory-based coherency
8-128 processors

Main components

- processor board – dual MIPS 195MHz R10000
- 2 processors connected by bus
- hub – connects processor bus to memory/other processors
    - coherence controller
    - up to 12 outstanding mem requests per processor
- IO Crossbar "Xbow" – connects to 6 IO interfaces and 2 nodes
    - up to 480 MB/s per Xbow
- Router – 6-way crossbar @ 800 MB/s per link
    - 2 local nodes + up to 4 other routers
        - 2 routers → 8 other routers → max 64 nodes
        - (128 nodes is a hack)
    - Hypercube toplogy
    - Bisection bandwidth

- 8 nodes – 1.6 GB/s
- 64 nodes – 12.8 GB/s??

Key technologies
directory-based coherence
OS support for locality
cheap ASICS – scalable interconnect
scales DOWN as well as UP
(only scales UP to a max size)

4) Clusters of SMPs

SGI Origin 2000 was meant to be most scalable architecture. Still only scale to 128 nodes (why might they do that?)

What if you want more than 128 nodes –
Option: engineer a more scalable version of O2000 architecture (expensive, low volume)
Option: connect together a bunch of O2000's with high-performance networks (they do the latter)

Trade-offs:
Cheap, scalable
More difficult programming model

****
Summary - 1 min
****
Consistency – want strong semantics, weak performance
Release Consistency
Trends – economical scaling