Shared Memory Architectures

Software coherence
Fences
Intro to hardware coherence

Discuss paper on RP3

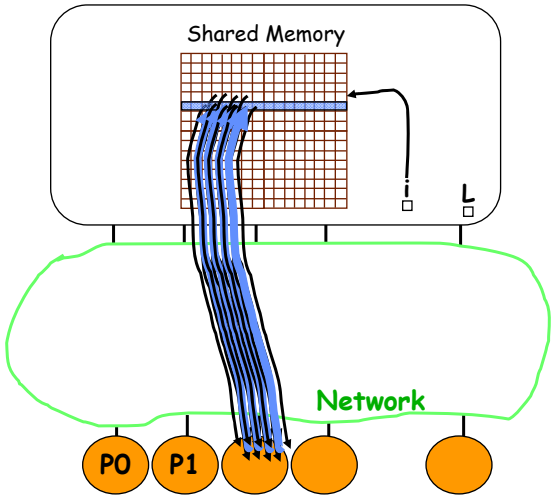
Agarwal

6.173
Fall 2010
L09

Today's Outline

- RP3 discussion
- How do caches change things
- Shared memory programming with caches
- Software coherence
- The meaning of shared memory
- Hardware cache coherence

Recall Shared Memory Jacobi



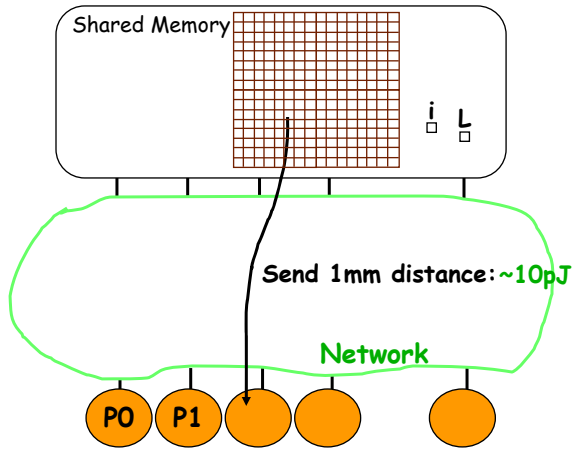
Lots of repeat accesses of data
Lots of communication over the network

And very energy inefficient

Pure Shared Memory

32-bit energy costs in 40nm

DRAM read: ~1000pJ



Add: ~ 1pJ

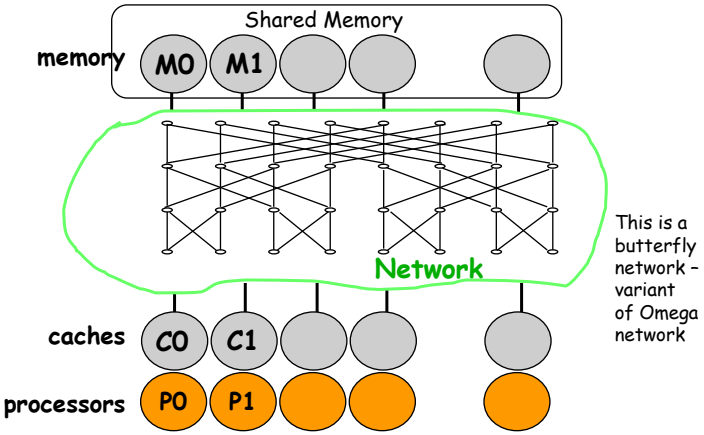
Register read: ~1pJ

Ideas?

Caches!

Cache read (small L1): ~10pJ

Ultracomputer Built as
RP3



- Indirect network – Omega network (details later in course)
- Shared memory machine with caches
- Each memory module placed physically close to processors
- Communication/synchronization through shared memory
- Hardware routing of memory requests
- SPMD FORTRAN programming (single program multiple data)
- No latency hiding – wait for memory request
- More complex hardware
- I could not find a picture of the RP3

What were the big ideas?

Trivia Question

Cosmic Cube – Chuck Seitz (Prof. Caltech)

RP3 – Greg Pfister (IBM,Yorktown, NY)

Ultracomputer – Allan Gottlieb (Prof. NYU)

These inventors have this in common:

(a) None finished their Bachelor's degrees

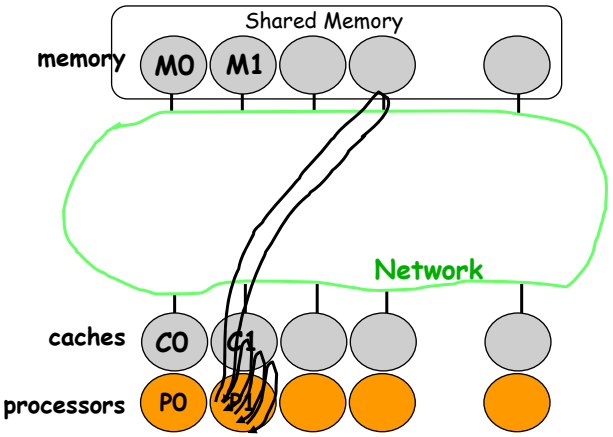
(b) They were all Geminis

(c) None of them is retired even today

(d) They are all Yankees fans

(e) None of the above

Caches – the good, the bad and the ugly



Average instruction time, no caching

$$T_{ins} = 1 + lT$$

T=100 cycles

P(load/st) ~0.3

With caching

$$T_{ins} = 1 + lmT$$

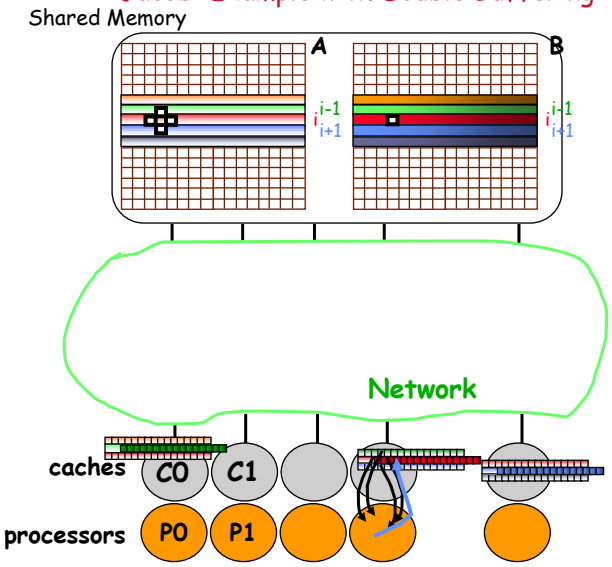
P(miss) ~0.05

More energy efficient too

- 7 -

Caches – the good, the bad and the ugly

Jacobi Example with Double Buffering



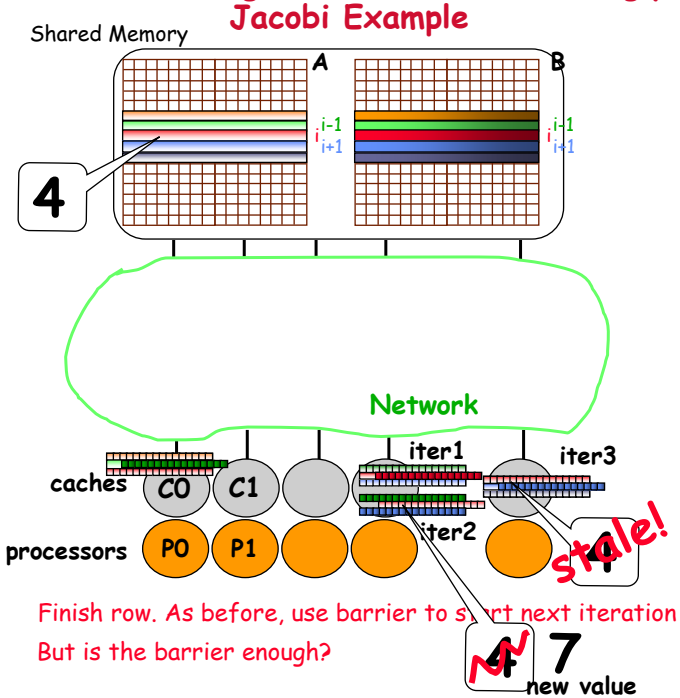
Caches exploit spatial locality here (fetch cache line)
Temporal locality too (discuss)
Network traffic dramatically reduced! Lower energy too

Finish row. As before, use barrier to start next iteration

But is the barrier enough?

- 8 -

Caches – the good, the bad and the ugly
Jacobi Example



Cache coherence problem! What do we do?

Maintaining coherence in manycores
Major approaches

- User-software managed coherence
  - RP3
  - Beehive
- System-software managed coherence
- Hardware managed coherence (next week)

User-software managed coherence in manycores

Typically yields weak coherence

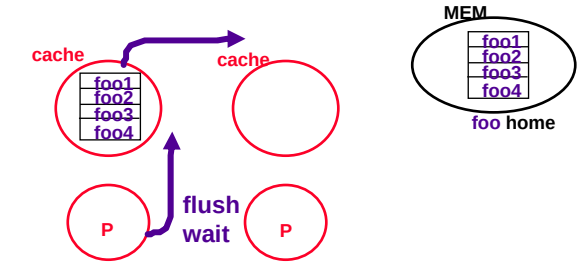
i.e. Coherence at sync points (or fence pts)

E.g.: When using locks for shared object accesses

Code:

How do you make this work?

User Software Coherence



GET_foo_LOCK

MUNGE

Flush foo* from cache, wait till done

RELEASE_foo_LOCK

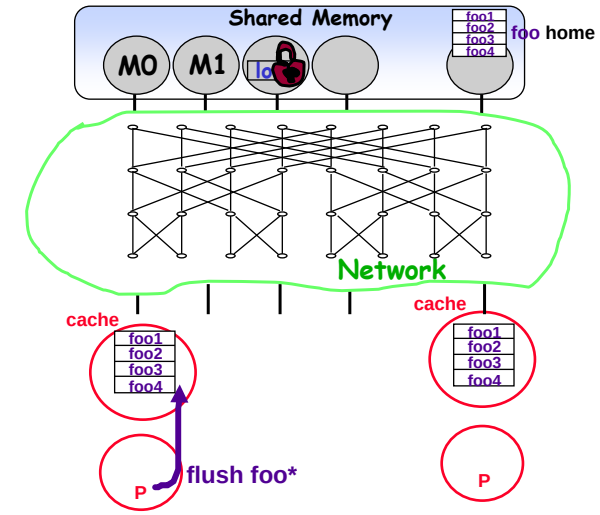
Issues …the ugly

- Need special processor instructions for flush (e.g., beehive)
- Also need a "memory fence" (wait till all flushed local values are reflected in global store)… next
- Can you cache the lock?
- Must be conservative; when in doubt, flush
  - Lose some locality
- But, can exploit application characteristics to allow some inconsistency

  e.g. TSP – bound does not have to be accurate

  Chaotic relaxation

Good Fences Make Good Neighbors



GET_foo_LOCK

read foo*

MUNGE

Flush foo* from cache

RELEASE_foo_LOCK

GET_foo_LOCK

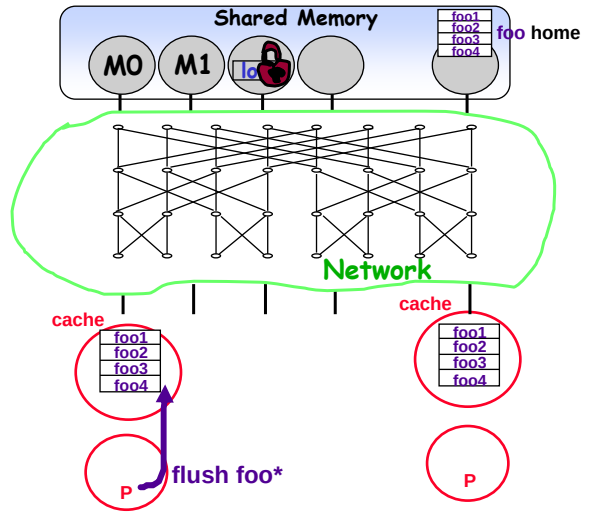
read foo*

Got new value. Works!

- Need special processor instructions for flush (e.g., beehive)

Good Fences Make Good Neighbors

But what about this?



GET_foo_LOCK

read foo*

MUNGE

Flush foo* from cache

RELEASE_foo_LOCK

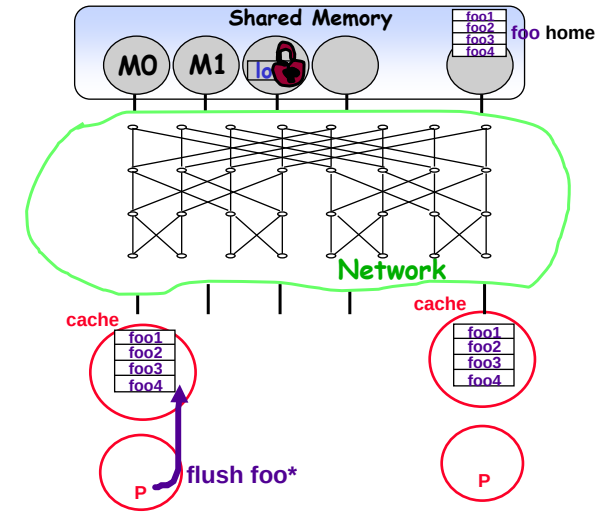
GET_foo_LOCK

read foo*

Got old value! Yikes!!

Good Fences Make Good Neighbors

Use a fence



GET_foo_LOCK

read foo*

MUNGE

Flush foo* from cache

Fence

RELEASE_foo_LOCK

GET_foo_LOCK

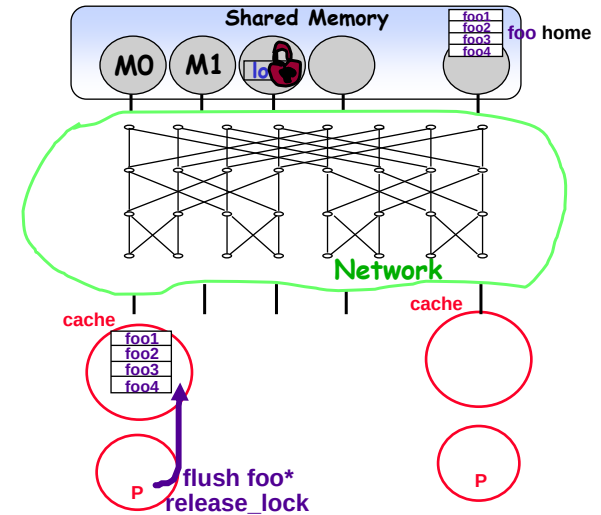
read foo*

Got new value! Phew!!

How to implement a fence?

Good Fences Make Good Neighbors

How to implement memory fences



GET_foo_LOCK

read foo*

MUNGE

Flush foo* from cache

Fence (generally, needs a new instruction)

RELEASE_foo_LOCK

Hmmm… why does flush work in Beehive?

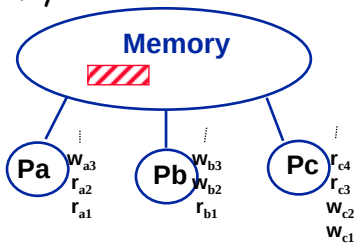
Put lock on same memory port as data it protects and make sure network maintains order

Fence instruction can wait for acks on previous memory references to make sure all have completed before returning

Foundations

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

What if you have caches?

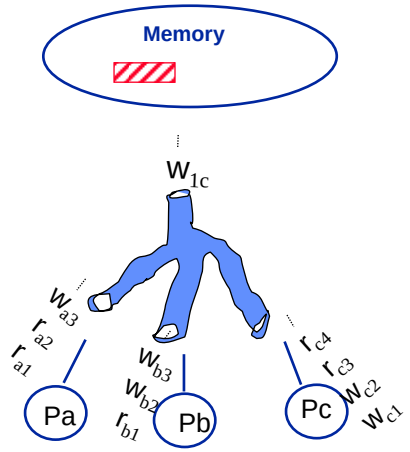


Sequential consistency: Final state (of memory) is as if all RDs and WRTs were executed in some fixed serial order (per processor order also maintained) → Lamport

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

Foundations

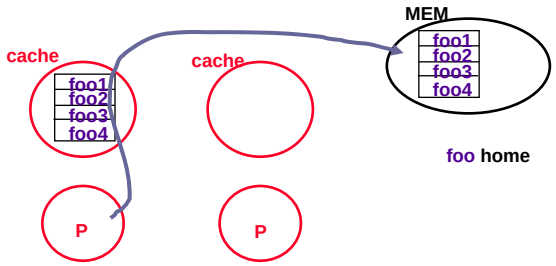
A hardware designers physical perspective of sequential consistency



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

We will revisit this in more detail in a couple of weeks

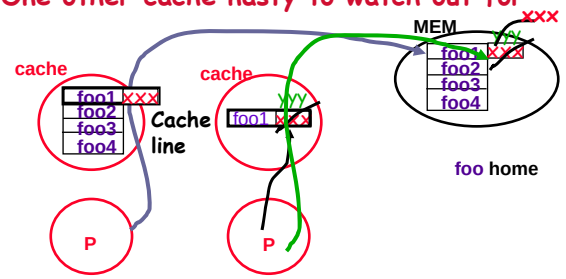
One other cache nasty to watch out for



Flush foo* from cache, wait till done

Does it always work?

One other cache nasty to watch out for



Flush yyy from cache, wait till done

Flush foo* from cache, wait till done

Correct final value: foo1 yyy

Wrong final value: foo1 xxx

Problem called "False Sharing"

Leads to bugs with sw coherence

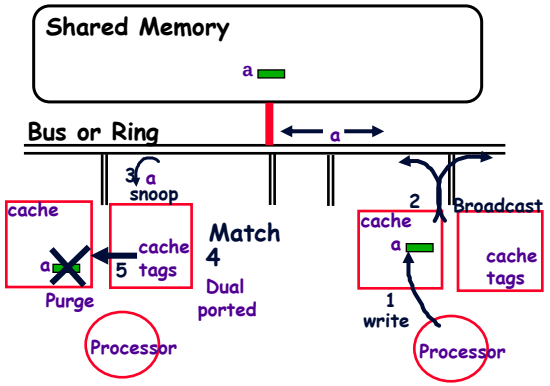
Leads to poor perf. with hw coherence

Solutions?

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

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
- How do you know when to broadcast, invalidate
  - State associated with each cache line
  - Key benefit: no global state in main mem

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