Shared Memory Architectures

Software coherence

Fences

Intro to hardware coherence

Discuss paper on RP3

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: \(~1000\text{pJ}\)
- Add: \(~1\text{pJ}\)
- Register read: \(~1\text{pJ}\)
- Send 1mm distance: \(~10\text{pJ}\)

Ideas?

Caches!

- Cache read (small L1): \(~10\text{pJ}\)
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

\[ T_{ins} = 1 + lT \]

\[ T = 100 \text{ cycles} \]

With caching

\[ T_{ins} = 1 + lmT \]

\[ P(\text{load/st}) \approx 0.3 \]

\[ P(\text{miss}) \approx 0.05 \]

More energy efficient too

More energy efficient too

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?

Jacobi Example with Double Buffering

Network traffic dramatically reduced! Lower energy too
Caches – the good, the bad and the ugly

Jacobi Example

Shared Memory

\[
\begin{array}{c}
A \quad B \\
C_0 \quad C_1 \\
P_0 \quad P_1
\end{array}
\]

Network

Finish row. As before, use barrier to start next iteration
But is the barrier enough?

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:

```c
shared vars

GET_foo_LOCK
/* MUNGE WITH foos */
foo1 = X = foo2
foo3 = .
.
RELEASE_foo_LOCK
```

How do you make this work?

User Software Coherence

- 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

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

Got new value. Works!

Got old value! Yikes!!
Good Fences Make Good Neighbors

Use a fence

How to implement memory fences

Fence (generally, needs a new instruction)

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

Got new value!
Phew!!
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) \( \rightarrow \) Lamport

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

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

Key: Using fence to wait until flush is done is the key mechanism that guarantees sequential consistency.
One other cache nasty to watch out for

Flush foo* from cache, wait till done

Does it always work?

Correct final value: xxx

Wrong final value: 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