CSC469 Tutorial 4
- Shared Memory Consistency -

Alexey Khrabrov
Announcement: A1 Office Hours

• Tuesday/Wednesday afternoon next week

• Piazza poll for times - please fill out if you’re interested in office hours
Why do we have to know this?

• To write correct programs
  – If you make incorrect assumptions about memory consistency, your program could crash

• To understand performance
  – If you evaluate your algorithm under incorrect memory consistency assumptions, you're in for a shock
Defn: Memory consistency model

- The order in which *memory operations* appear to execute
  - read (load)
  - write (store)
- What does it affect?
  - Ease of programming
  - Performance
Sequential Consistency

• [Lamport, 1979] - definition:
  – “A multiprocessor system is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.“

• Parallel schedule of instructions is equivalent to a legal serial schedule; ie.
  – Machine instructions are not reordered.
  – Memory references are globally ordered.
    • ie. All processors see all writes at the same time.
    • (ie. everyone sees the same schedule)

• So, 2 aspects:
  – maintain program order among operations from individual processors
  – maintain a single sequential order among operations from all processors (implies memory ops are atomic/instantaneous w.r.t other memory operations)
Questions

• *Can* we implement a sequentially-consistent machine?

• Do we *want* to?
Answers

• *Can we implement a sequentially-consistent machine?*
  – **YES**
  – **Example:** MIPS R10000

• *Do we want to?*
  – **NO**
  – Too restrictive memory consistency model
  – Wastes opportunity for parallelism
  – H/W & S/W issues to consider in multiprocessors
Why not?

- Hardware optimizations
  - Write buffers
  - Overlapping writes
  - Non-blocking reads
- Architectures with caches
- Software
  - Optimizing compilers
Why not? Write buffers

• Common optimization
• Processor that wants to write to memory
  – enqueues write on a write buffer
  – continues
• Reads can bypass the write buffer
Why not? Write buffers

Initially, Flag1 = Flag2 = 0

**CPU1:**

Flag1 = 1

if (Flag2 == 0)

    critical section

**CPU2:**

Flag2 = 1

if (Flag1 == 0)

    critical section
Why not? Write buffers

Initially, Flag1 = Flag2 = 0

CPU1:
Flag1 = 1
if (Flag2 == 0)
critical section

CPU2:
Flag2 = 1
if (Flag1 == 0)
critical section

CPU1 buffers write (Flag1 = 1)
CPU1 enters critical section
CPU2 reads Flag1 ==0 b/c write still in buffer
CPU2 enters critical section
Why not? Overlapping writes

- Interconnection network
  - connects the processors to memory
- IA-32: use a *bus* (linear)
  - one write to memory at a time
- More general interconnection:
  - multiple writes can be happening simultaneously
  - two writes *by the same CPU* can be reordered
Why not? Overlapping writes

Initially, data = head = 0

CPU1:
\[
\begin{align*}
\text{data} &= 2000; \\
\text{head} &= 1;
\end{align*}
\]

CPU2:
\[
\begin{align*}
\text{while (!head)} &; \\
x &= \text{data};
\end{align*}
\]

Can CPU2 see x=0 when finished?
Why not? Overlapping writes

Initially, data = head = 0

CPU1:
4 data = 2000;
1 head = 1;

CPU2:
2 while (!head)
3 x = data;

Initially, data = head = 0

CPU1:
4 data = 2000;
1 head = 1;

CPU2:
2 while (!head)
3 x = data;
Why not? Non-blocking reads

• Read analogue of overlapping writes
  – Happens with a general interconnect

• CPU can be reading multiple values from memory simultaneously

• Order read != Order written
Why not? Non-blocking reads

Initially, data = head = 0

CPU1:
data = 2000;
head = 1;

CPU2:
while (!head)
    ;
x = data;

Can CPU2 see x=0 when finished?
Why not? Non-blocking reads

Initially, data = head = 0

CPU1:
2 data = 2000;
3 head = 1;

CPU2:
4 while (!head)  

1 x = data;

Initially, data = head = 0
Why not? Caches

• Assume each CPU has a cache
• Write to a variable in a cache
  – Must *invalidate* it in other processors' caches
  – Send them the updated value
• Reads could return cached values instead of updated values
Why not? Caches

Initially, data = head = 0

CPU1:
- data = 2000;
- head = 1;

CPU2:
- while (!head);
- x = data;
Why not? Caches

Initially, \( \text{data} = \text{head} = 0 \)

**CPU1:**

\[
\begin{align*}
\text{data} &= 2000; \\
\text{head} &= 1;
\end{align*}
\]

**CPU2:**

\[
\begin{align*}
\text{while} &\quad (\text{!head}) \\
\text{x} &= \text{data};
\end{align*}
\]

* data already in CPU2’s cache
* CPU1 executes write (\( \text{data} = 2000 \))
* CPU1 tells CPU2 to invalidate its cache, but CPU2 does not get the message yet
* CPU2 reads stale data from its cache (\( \text{x} = \text{data} \))
Why not? Optimizing Compilers

• Compilers do many optimizations
  – constant folding
  – dead code elimination
  – moving code outside of loops
  – reordering instructions to reduce pipeline stalls
• Effects are similar to hardware optimizations
• Compiler considers only one thread of execution when compiling a function
Initially, data = head = 0

CPU1:
data = 2000;
head = 1;

CPU2:
while (!head)
;
x = data;
Why not? Optimizing Compilers

Initially, data = head = 0

CPU1:
data = 2000;
head = 1;

CPU2:
while (!head)
  
x = data;

* Reverses the order of CPU1’s instructions (independent)
What guarantees do we have?

• Memory consistency model of shared-memory multiprocessors provides formal specification of how the memory system will appear to the programmer
  – Behaviour expected by the programmer
  – Actual behaviour supported by the system

• Places restrictions on what values can be returned by a read in a shared memory program

• Intuitively, the value of the “last” write to the same location
  – uniprocessors: clearly defined by program order
  – not the case with multiprocessors
Relaxing Sequential Consistency

- Causal Consistency (CC) – weaker than SC
  - B control dependent on A
    - IF X THEN Y, A = IF X, B = Y
  - “A < B” iff A & B causally-related
    - B depends on A
  - Memory operations that are potentially causally-related must be seen by all CPUs in the same order

- “A < B” means all processors see A before B
Relaxing Sequential Consistency

• Processor/PRAM Consistency: - weaker than CC
  – Relaxes causality across CPUs
  – Instr A & B write to word W
  – “A < B” iff A & B on the same CPU
    • (and A is before B according to program order)
  – All CPUs see writes from one CPU in the order they were issued from this CPU
  – However, writes from different CPUs may be seen in a different order by different CPUs

• Sync ops: Weak Consistency, Release Consistency, etc. ...
Fence Instructions

• What do we do when we want sequential consistency?
• Use a fence instruction
  – aka memory barrier

\[
\text{fence} \quad \implies \quad A < B
\]
Read Fences, Write Fences

- **write fence** – each write *before* the fence must happen before any write *after* the fence
  - very common
- **read fence** – each read *before* the fence must happen before any read *after* the fence
- Various atomic instructions such as Compare-And-Swap (usually) generate fences
  - e.g., that’s why you can be sure you read fresh values after acquiring a mutex
What fences are needed?

**CPU1:**
data = 2000;
head = 1;

**CPU2:**
while (!head)
    ;
x = data;
What fences are needed?

CPU1:

data = 2000;
write_fence();
head = 1;

CPU2:

while (!head)
  ;
read_fence();
x = data;
References


Assignment 1

• Should determine threshold experimentally
  – Part of the tool, not just a hard-coded value in your program

• To find CPU frequency, use something like k-best measurement scheme from lecture slides
  – Combine with sleep or nanosleep calls
  – Check man pages to see if both over and under estimates are possible, or if error is in only one direction (nanosleep has an advantage over sleep)
Assignment 1: some tips

• Experimental data representation (graphs)
  – Don’t even try to use MS Excel
  – Use gnuplot or alternatives, e.g. matplotlib
  – Make sure your graphs are readable
    • tip: show them to someone not taking this course

• Find some nice way to represent Part 2 results
  – Not just a huge table with memory bandwidth values

• Collect data for Part 2 in advance; the server might be under high load closer to the deadline

• Try to minimize system resources usage when running Part 1 experiments
A1: Report tips/expectations

• The important part of the assignment
  – code is the instrument to get data for the report

• We expect a proper technical report, not just an assignment write-up

• Clear description of methodology
  – understandable by someone not familiar with the assignment

• Presenting experimental data
  – measurement errors, number of runs, ...
  – think of this like a physics lab report