548

Lecture 10 - Consistency

Main points, as I see them

- Consistency is a part of the ISA, coherence the implementation
- You are most likely to program to TSO (or processor consistency)
  - => My brain thinks this way about MP coding. SC is actually *hard* for me to reason about :-)
- The compiler is not your friend
- More relaxed consistency models exist => ARM
- Only access aligned items

I want determinism?

- Where does the nondeterminism come from?
  - external events, interrupts, other code, clock speed, resource competition non deterministic; the cache has a random initial state; speculation state (BP bits)
- Global clock: all processors execute instructions at the same rate; deterministic ordering of how of processors; need consistency model
  - deterministic only needs causal ordering;
  - don't need same rate; clocks only need to be deterministic

Why do you want DMP?

- Easier to debug
- Easier to run a test suite.. or not.
- Predictable performance
  - Slows you down :=)
- Replayable

Your Questions 1

- In SC why can't memops to different address happen between the load and store of an atomic RMW?
- Should we distinguish 'allowed executions' by a memory model from whether operations can happen in parallel? Where do these concepts intersect?
- In practice, how long is something typically sitting in the write buffer? Is it usually just other cores' atomic ops holding up the obtaining of Exclusive state on the cache line?

- Is it possible to guarantee deterministic execution and still give substantial speed-up from using multiple processors?
- Are there major latency issues with multicore systems and maintaining cache coherence?
- What are granularities of coherency have actually been implemented?
- Are there other invariants used to define cache coherence? If so, what?

Your questions 2

- Do CMPs enforce consistency constraints even with single threaded code?
- The text specifically identifies that they've not addressed instruction caches, multiple-level caches, shared caches, virtual memory, DMA... The list goes on. How much more complicated does this get, and how realistic is this "primer" because of that?
- X86 does TSO (we think). Does anyone actually use SC?
- How much would it realistically hurt the performance to disable the write buffers?
- SC is used in some databases, is TSO?
- Any modals between SC and TSO?
- Why are write buffers so necessary for performance? i.e. Are they that much more expensive than loads? Why not a load/store buffer that buffers both?
- Are there other methods to achieve cache coherency beyond the broadcast mechanisms that are more scalable?
- Table 3.3 (r1, r2) = (0,0) is allowed in today's real hardware! If they don't

Your questions 3

- Does x86 actually use TSO, or something slightly different? How are fences usually implemented in hardware?
- It seems like it would be beneficial in some cases to have a dedicated memory bank for each core as well as a shared memory; this way cores could use their own memory without having to worry about multicore coherence and consistency issues. Has this been done?
- Coherence and consistency models implicitly assume that arbitrary code could be running on the other cores but in practice this isn't true since we know what code is running on all cores. Is it possible to take advantage of this (perhaps by using hints such as FENCES inserted by the compiler) in order to reduce the amount of work necessary to preserve coherence and consistency?
- I've sometimes wondered how much more powerful a processor could be made if all of the issues it deals which are not strictly necessary (e.g. out of order execution, deciding which functional units should process which data, memory coherence and consistency issues, etc.) were dealt with as little as possible so the compiler would have to resolve all of these things (e.g. it would have to decide how instructions would be sent to different functional units and how to handle memory issues). It seems like the processor could be made much more powerful in this case though obviously it would be a lot harder to write good compilers for it (and the potential for nasty bugs would be much greater).
- Overhead of coherence traffic?
-