Multithreading and Cache Coherence

Ryan Lee
6.823 Fall 2021
Adapted from prior course offerings
Goal of Multithreading

» Hide the cost of long-latency operations...
   - by doing something else!

» Fine-grained
» Coarse-grained
» SMT (Simultaneous Multithreading)
A) Conventional Processor

B) Coarse-grained Multithreaded (CMT)

C) Fine-grained Multithreaded (FMT)

D) Simultaneous Multithreaded (SMT)

Execution Units → Time

Interrupt, exception, or OS call

return from exception

Cache miss

Cache miss

Cache miss
Why ICount Policy?

» Recall: Original SMT implementation didn’t perform very well

» ICount: Fetch from the least number of instructions in flight
Consider the following toy problem...

» Out of order machine
  - 20 ROB Entries
  - Two Threads: t1 and t2
  - Want to maximize commit throughput

» Threads have different average instruction latencies
  - $T = \text{Throughput}, N = \text{Number of instructions in flight}$
  - $T_{t1} = 0.1 \times N_{t1}$
  - $T_{t2} = 0.8 \times \sqrt{N_{t2}}$
\[ T_1: T_{t1} = 0.1 \times N_{t1} \]
\[ T_2: T_{t2} = 0.8 \times \sqrt{N_{t2}} \]

» Which thread would have more instructions in flight with *round-robin* policy? **T1**
$$T_1: T_{t1} = 0.1 \times N_{t1}$$
$$T_2: T_{t2} = 0.8 \times \sqrt{N_{t2}}$$

» Which thread would have more instructions in flight with ICount policy?  
   Same

» Which policy has better overall throughput?  
   ICount
T1: $T_{t1} = 0.1 \times N_{t1}$
T2: $T_{t2} = 0.8 \times \sqrt{N_{t2}}$

» Extra question: Is there a policy that always maximizes overall throughput?
Goals of caches

- Small memories that provide quick access to recently accessed data.

- Transparently managed by hardware (and OS)
  - Program output should appear as if the caches did not exist and applications directly accessed main memory.
  - In contrast with scratchpads (explicitly managed)
Goals of shared memory

» Multiple concurrently executing threads can read and write data in a single address space.

» Transpareently managed by hardware (and OS)
  - Program output should appear as if the caches did not exist and applications directly accessed single memory.
  - In contrast with message passing (explicitly manage shared data)
Caches in parallel systems

» Caches give quick access to data:
  - Small **private caches** may hold copies of data.

» Transparent management: How to ensure cache accesses don’t act on stale data?
  - No shared writeable address space: Pure message passing, or
  - Cache coherence
Cache coherence

Processor X:
Ld 0xA → 0
Ld 0xA → 0
Ld 0xA → 0
...
Ld 0xA → 42
Ld 0xA → 42

Processor Y:
St 42 → 0xA

Q. Do you think this system is coherent?
Cache Coherence

» Two Rules:

1. Write propagation: Writes eventually become visible to other processors
2. Write serialization: All processors observe writes to one location appear to happen in a consistent order

» Strategies for propagation:

- A write invalidates copies in other private caches
- A write updates copies in other private caches
- Tradeoffs?
Serialization strategies

» **Snoopy** coherence protocol
  On a miss, private caches broadcast their actions through a bus-like interconnect, other caches observe ("snoop") and perform updates or invalidations.

» **Directory-based** coherence protocol
  On a miss, private caches send unicast message to the directory, which serializes requests and sends unicast messages to other caches to perform updates or invalidations.

Tradeoffs?
Do write-through caches need coherence?

» Yes.
  - Writes must propagate: update or invalidate copies in other private caches.
  - Write serialization is trivial (where is the serialization point?)

» A protocol with two stable states is sufficient:
  - Invalid
  - Shared

» Do you need transient cache states?
  - Yes!