Multiprocessor Memory Consistency

Kai Shen

Multiprocessor Cache Coherence

- Coherence means the system semantics is the same as that of a system without processor-local caches
- Multiprocessor cache coherent if there exists a hypothetical sequential order of all operations for each data location:
  - returned value in the read operation is that written by last write in the sequential order
  - the sequential order matches the order of operations from each processor
- Individual ordering on each location may not be sufficient for supporting parallel execution semantics

Multithread Application Semantics

- Initially

```
flag1 = flag2 = 0;
```

- P1

```
flag1 = 1;
if (flag2==0) {
   /* critical section */
}
```

- P2

```
flag2 = 1;
if (flag1==0) {
   /* critical section */
}
```

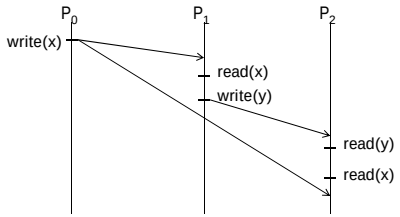
- Semantics – only one CS may run (or neither)
- Uni-processor
  - Assume interrupts are precise (instruction is either finished or has not started at interrupt)
- Multi-processor
  - Is cache coherence sufficient?
  - Memory consistency model: specification of memory access behaviors (with or without cache, over all accesses)

Sequential Memory Consistency

- It means the memory access semantics is the same as that of a uni-processor system with precise interrupts
- Sequential consistent if there exists a hypothetical sequential order of all operations on all locations:
  - returned value in the read operation is that written by last write in the sequential order
  - the sequential order matches the order of operations from each processor

Sequential Ordering on One-Location, All-Locations

- Does the follow execution satisfy sequential ordering on one-location, all-locations?



Support Sequential Consistency

- A multiprocessor without processor-local caches, using a shared bus connected to memory
  - All memory operations serially issued in program order by each processor
- A multiprocessor with processor-local caches, using a shared bus connected to memory
- Bus snooping protocol for write-through cache (originally proposed for cache coherence)
  - Each processor monitors the bus for writes
  - If there is a local cached copy of the write target, it is invalidated or updated.
  - Does it guarantee sequential consistency?

Complications

- NUMA
- Instruction-level parallelism (SuperScalar processor):
  - Read is issued while writes to other locations are outstanding.
  - Overlapping writes to different locations
  - Non-blocking reads (issuing next read, to a different location, while waiting for the current one)
- Compiler reordering

Support Sequential Consistency

- Program order
  - A processor ensures the previous memory operation completed before issuing next operation in program order
  - For a write, completion means the result is visible to all
- Write atomicity
  - Writes to the same location are serialized
  - A write is not returned by a read until the result is visible to all

⇨ Not helpful to instruction-level parallelism

Optimizations satisfying Sequential Consistency

- Prefetch write ownership (ReadEx)
  - What if the ReadEx shouldn't be done early after all?
- Speculative read
  - What if the speculation is wrong?

Relaxed Consistency Models

- Relax write-to-read ordering
  - Optimization: a read can be issued while a write is ongoing if they are to different locations
- Relax write-to-write ordering
  - Optimization: addresses of outstanding writes are buffered; new write can be issued as long as its address doesn't appear in the buffer
- Safety backup
  - serialization instruction
  - waits for effects of write complete

Specific Relaxed Consistency Models

- IBM 370
- SPARC V8 total ordering model (TSO)
- Processor consistency
- Weak ordering
- Release consistency