Multiprocessor Memory Consistency

Kai Shen

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 (instructions before interrupt are completed; instructions after interrupt hasn't started)
- Multi-processor with local caches
  - Would cache coherence help?

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 memory accesses:
  - returned value in the read access is that written by last write in the sequential order
  - the sequential order matches the order of memory accesses from each processor
- At the program level, ordering only effective on individual data location since cross-data-location ordering is affected by optimizations (what optimizations?)

Complications

- Ordering only effective on individual data location since cross-data-location ordering is affected by
  - 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 (common sub-expression elimination, register allocation, …)
- But 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 */
}
```

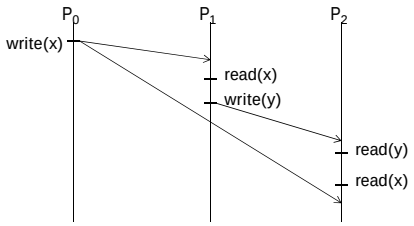
- Semantics – only one CS may run (or neither)
- Multi-processor
  - Is cache coherence sufficient?
  - Memory consistency model: specification of memory access behaviors (at program level, at the presence of instruction-level parallelism)

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 memory accesses on all locations:
  - returned value in the read access is that written by last write in the sequential order
  - the sequential order matches the order of memory accesses from program on each processor

Sequential Ordering on One-Location, All-Locations

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



- So this can happen on a cache-coherent machine that allows unrestricted instruction-level parallelism. How?

Support Sequential Consistency

- Naïve approach
  - Processor-level serialization: A multiprocessor without processor-local caches, using a shared bus connected to memory
  - Program-level serialization: All memory operations serially issued in program order by each processor
- A multiprocessor with processor-local caches
  - Use cache coherence to ensure processor-level serialization

Support Sequential Consistency

- Program order
  - A processor ensures the previous memory access completed before issuing next memory access in program order
- 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
- ⇨ Determining the completion of a write typically requires explicit acknowledgement memory from memory (and other caches)

Optimizations satisfying Sequential Consistency

- Prefetch write ownership (ReadEx)
- Speculative read
  - Can the speculation go wrong?
  - What do we do when it goes wrong?

Relaxed Consistency Models

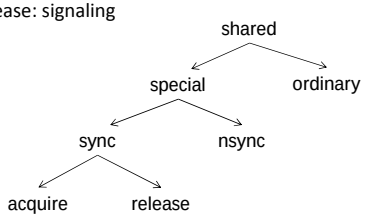
- Relax write-to-read ordering
  - Optimization: a read can be issued while a write is ongoing if they are to different locations
- Relax write-to-write ordering
  - Optimization: addresses of outstanding writes are buffered; new write can be issued as long as its address doesn't appear in the buffer
- Safety backup
  - Serialization instruction (memory fence)
  - Example: http://en.wikipedia.org/wiki/Memory_barrier
- Collaboration between hardware and software

Specific Relaxed Consistency Models

- IBM 370
- SPARC V8 total ordering model (TSO)
- Processor consistency
  - Relaxing write-to-read ordering with safety nets
- Weak ordering
  - Relaxing all types of ordering
  - Allow a special category of synchronization memory operations: a synchronization operation preserves order with all previous and following operations

Specific Relaxed Consistency Models

- Release consistency
  - Acquire: reading a critical variable (flag) signaled by others
  - Release: signaling



- Ordering: acquire → all, all → release, and special → special.