Synchronization

Kai Shen

Sequential Memory Consistency

- It means the memory 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
- Support
  - Multiprocessor cache-coherence
  - Each processor issues all memory accesses in program order without instruction-level parallelism; no compiler re-ordering either

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
- Release consistency
- … …
- Software manipulation
  - Software understands the relaxed consistency model and constructs programs accordingly
  - Use serialization instructions (memory fence, acquire→all, all→release) when needed

Synchronization

- What is synchronization?
  - Synchronize/coordinate the progress of parallel/concurrent processes to satisfy certain high-level semantics
- What are desired semantics for synchronization?
  - Mutex locks
  - Condition variable
  - Barrier
  - …
- Avoid race conditions
  - Race condition – output/result of a parallel execution depends on the relative progress of concurrent processes

Synchronization and Instruction-Level Parallelism

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

P1
```
flag1 = 1;
if (flag2==0) {
   /* critical section */
}
```

P2
```
flag2 = 1;
if (flag1==0) {
   /* critical section */
}
```

- Assume sequential memory consistency on multiprocessor
- Show that at most one of the critical sections runs?
- What if each processor allows instruction-level parallelism that relaxes write-to-write reordering (on different locations)?
- What if write-to-read reordering is relaxed?
  - Where to put the fence instruction?

Synchronization and Instruction-Level Parallelism

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

P1
```
flag1 = 1;
turn = 2;
while (flag2 && turn==2) ;
/* critical section */
flag1 = 0;
```

P2
```
flag2 = 1;
turn = 1;
while (flag1 && turn==1) ;
/* critical section */
flag2 = 0;
```

- Assume sequential memory consistency on multiprocessor
- Mutually exclusive?
- Deadlock free?
- What if the each processor relaxes write-to-write reordering?
- What if write-to-read reordering is relaxed?

Synchronization Using Special Instruction: TSL (test-and-set)

```
entry_section:
   TSL R1, LOCK          | copy lock to R1 and set lock to 1
   CMP R1, #0            | was lock zero?
   JNE entry_section     | if it wasn't zero, lock was set, so loop
   RET                   | return; critical section entered

exit_section:
   MOV LOCK, #0          | store 0 into lock
   RET                   | return; out of critical section
```

- Mutually exclusive and deadlock free (support many processes, only 2 in the previous software case).
- What if the superscalar processor may reorder memory accesses to different locations?

Synchronization Performance

P1
```
flag1 = 1;
turn = 2;
while (flag2 && turn==2) ;
/* critical section */
```

- Costs of busy waiting on others?
  - No processor-local cache
  - Cache-coherent local cache

Synchronization Performance

```
entry_section:
   TSL R1, LOCK          | copy lock to R1 and set lock to 1
   CMP R1, #0            | was lock zero?
   JNE entry_section     | if it wasn't zero, lock was set, so loop
   RET                   | return; critical section entered
```

- Costs of busy waiting on others (cache-coherent local cache)?
  - Write-through cache, write-back cache
  - One is waiting, many are waiting

```
exit_section:
   MOV LOCK, #0          | store 0 into lock
   RET                   | return; out of critical section
```

- Costs of settling the new critical section holder?

Synchronization Performance Summary

- Synchronization performance
  - Bus/memory traffic while waiting
  - Bus/memory traffic while the critical section becomes available for competition
- Scalability in a large system of many threads
  - Costs of busy waiting to each other
  - Costs of settling competition when critical section becomes available