Synchronization

Sequential Memory Consistency
- It means the memory semantics is the same as that of a uni-processor system
- 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 program order of operations at each processor
- Support
  - Assume each processor issues all memory operations in program order without instruction-level parallelism
  - Bus architecture with snooping-based coherence

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 instructions (memory barrier, store barrier)

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 Scheduling

- Options:
  - Busy-waiting synchronization
  - Waste CPU on waiting
  - Blocking (yielding CPU) synchronization
  - Overhead of context switch
- Parallel programs with dedicated processors?

Synchronization and Instruction-Level Parallelism

- Mutually exclusive and deadlock free?

```plaintext
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
```

- What if the hardware allows write-to-write reordering on the same processor?
- What if write-to-read reordering is allowed?
Synchronization Performance

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

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

Synchronization Scalability

- In a large system of many threads
- Costs of busy waiting to each other
- Costs of settling competition when critical section becomes available

Blocking Synchronization

- Benefit:
  - Useful when number of processors fewer than number of threads
  - Critically important when one thread is switched out while it is inside a critical section
- Cost:
  - Context switch overhead (cache warmup cost)

- When a thread must wait for synchronization from some other thread, should it spin (busy-wait) or block?
  - Spin for the time equal to the context-switch overhead. If not successful, then block.