Relaxed Memory Consistency

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Topics

- Motivation of relaxed consistency models
- TSO and PC models
- Weak consistency models
Memory Consistency Model

- Specifying constraints on the order in which memory operations can appear to execute with respect to one another
- Enabling programmers to reason about the behavior and correctness of their programs

- Fewer possible reorderings $\rightarrow$ more intuitive
- More possible reorderings $\rightarrow$ more performance optimization
  - Fast but possible wrong(?)
Sequential consistency: execution should be the same result as if ...

- Some serial order among operations from multiple processors
- Program order among operations in each individual processor
Memory Operation Ordering

- Four types of memory operation orderings
  - \( W \rightarrow R \): writes must complete before subsequent reads
  - \( R \rightarrow R \): reads must complete before subsequent reads
  - \( R \rightarrow W \): reads must complete before subsequent writes
  - \( W \rightarrow W \): writes must complete before subsequent writes

- Sequential consistency maintains all of the orderings
- Relaxed memory consistency models allow some of the orderings to be violated
Motivation

- Sequential consistency is intuitive to programmers
  - Easy to program
- But, is sequential consistency essential for all memory operations?
  - No
  - Strict ordering of memory operations restricts many performance optimization chances
    - Reordering of instructions in compiler
    - Out-of-order execution
    - Write buffers in processor
    - Cache miss vs. cache hit
Motivation: Hiding Latency

- Why we are interested in relaxing memory consistency model
  - Performance
  - Specifically, hiding memory latency
    - Overlap memory accesses with other operations
  - Memory access in a cache coherent system is more complex

![Diagram showing memory access overlap]

Write A

Read B

vs.

Write A

Read B
Intuition Behind Relaxed Memory Consistency Models

Program order

Thread 1 on P1
A = 1;
B = 1;
flag = 1;

while (flags == 0)
    u = A;
    v = B;

Sufficient order

Thread 1 on P1
A = 1;
B = 1;
flag = 1;

while (flags == 0)
    u = A;
    v = B;
Allowing Reads to Move Ahead of Writes

- Four types of memory operation orderings
  - \( W \rightarrow R \): writes must complete before subsequent reads
  - \( R \rightarrow R \): reads must complete before subsequent reads
  - \( R \rightarrow W \): reads must complete before subsequent writes
  - \( W \rightarrow W \): writes must complete before subsequent writes

- Motivation
  - Hiding write latency
    - Taking advantage of write buffers in a processor

- Models
  - Total Store Ordering (TSO)
  - Processor Consistency (PC)
Allowing Reads to Move Ahead of Writes

- **Total store ordering (TSO)**
  - Processor P can read B before it’s write to A is seen by all processors
    - Processor can move its own reads in front of its own writes
  - Read by other processors cannot return new value of A until the write to A is observed by all processors $\rightarrow$ write atomicity

- **Processor consistency (PC)**
  - Any processor can read new value of A before the write is observed by all processors
  - Does not guarantee write atomicity

- **In TSO and PC, W→W constraint still exists**
  - Writes by the same thread are not reordered
  - They occur in program order
# Example Codes

<table>
<thead>
<tr>
<th>Code 1</th>
<th>Code 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P1</strong></td>
<td><strong>P1</strong></td>
</tr>
<tr>
<td>$A = 1$;</td>
<td>$A = 1$;</td>
</tr>
<tr>
<td>$Flag = 1$;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code 3</th>
<th>Code 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P1</strong></td>
<td><strong>P1</strong></td>
</tr>
<tr>
<td>$A = 1$;</td>
<td>$A = 1$;</td>
</tr>
<tr>
<td>B = 1;</td>
<td>print B;</td>
</tr>
</tbody>
</table>

### Execution matches sequential consistency (SC)

<table>
<thead>
<tr>
<th></th>
<th>Code 1</th>
<th>Code 2</th>
<th>Code 3</th>
<th>Code 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Store Ordering (TSO)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Processor Consistency (PC)</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
Allowing Writes to be Reordered

- **Four types of memory operation orderings**
  - $W \rightarrow R$: writes must complete before subsequent reads
  - $R \rightarrow R$: reads must complete before subsequent reads
  - $R \rightarrow W$: reads must complete before subsequent writes
  - $W \rightarrow W$: writes must complete before subsequent writes

- **Motivation**
  - Hiding write latency more (write merging in write buffer)
  - Making writes visible to other processors sooner

- **Partial Store Ordering (PSO)**
  - Execution may not match sequential consistency
    - P2 can observe A is 0
      - Thread 1 on P1
        - $A = 1$
        - flag = 1
        - print A
      - Thread 1 on P1
        - while ( flags == 0)
        - print A
Allowing All Reorderings

- Four types of memory operation orderings
  - $W \rightarrow R$: writes must complete before subsequent reads
  - $R \rightarrow R$: reads must complete before subsequent reads
  - $R \rightarrow W$: reads must complete before subsequent writes
  - $W \rightarrow W$: writes must complete before subsequent writes

- Examples
  - Weak ordering (WO)
  - Release Consistency (RC)
    - Processor supports special synchronization operations
    - Memory accesses before sync must complete before sync issues
    - Memory accesses after sync cannot begin until sync complete
Example: Expressing Synchronization in Relaxed Models

- **Intel x86**
  - Total Store Ordering (TSO) model
  - Provides sync instructions if software requires a specific instruction ordering not guaranteed by the consistency model
    - lfence ("load fence")
    - sfence ("store fence")
    - mfence ("mem fence")

- **ARM**
  - More relaxed consistency
Conflicting Data Accesses

- Two memory accesses by different processors conflict if
  - They access the same memory location
  - At least one is a write

- Unsynchronized program
  - Conflicting accesses not ordered by synchronization

- Synchronized programs yield SC results on non-SC systems
Relaxed Consistency Performance

Base: Sequentially consistent execution
W-R: relaxed W-R ordering constraint
W-W: relaxed W-W ordering constraint
Summary

- **Goal**
  - Obtain higher performance by relaxing memory consistency
    - By reordering memory operations to hide latency

- **One cost is software complexity**
  - Compiler or programmer must correctly insert synchronization to ensure certain specific ordering
    - In practice, complexities are encapsulated in libraries that provide synchronization primitives (e.g., lock, unlock, barrier)

- **Relaxed consistency models differ in which memory ordering constraints they ignore**