Memory Consistency

Instructor: Josep Torrellas
CS533
Hiding Memory Latency

- Overlap memory accesses with other accesses and with computation:

\[ \text{Wr A} \quad \text{Rd B} \quad \text{Wr A} \quad \text{Rd B} \]

- Simple in uniprocessors
- Can affect correctness in MPs
- Memory Model: specifies the ordering constraints among accesses
Uniprocessor Memory Model

• Memory accesses atomic and in program order

Write A
Write B
Read A
Read B

• Not necessary to maintain sequential order for correctness
  – Hardware: buffering, pipelining
  – Software: register allocation, code motion

• Simple for programmers
• Allows for high performance
Shared Memory Multiprocessors

- Order between accesses to different locations becomes important

```
P1
A = 1;
Flag = 1; wait (Flag == 1);
.. = A;
```

- Unsafe reorder can happen: accesses issued in order may be observed out of order (even without caches):
  - Flag is in the local memory module of P2
Caches Complicate Things More

• Multiple copies of the same location

```plaintext
P1
A = 1;

P2
wait (A == 1);
B = 1;

P3
wait (B == 1);
.. = A;
```

• P3 had A=B=0 in its cache, invalidations for B have arrived before the invalidations for A. P3 reads 0
Sequential Consistency

- Formalized by Lamport
  - “Execution of parallel program appear as some interleaving of the parallel processes on a sequential machine”

- Intuitive orders assumed by programmer are typically maintained
Example

- Initially: all vars are 0

  \[\begin{align*}
  &\text{P1} & \text{P2} \\
  &A = 1 & x = \text{Flag} \\
  &\text{Flag} = 1 & y = A
  \end{align*}\]

- Possible \((x,y) = (0,0),(0,1),(1,1)\)
- Impossible \((x,y) = (1,0)\)
How We Will Proceed

• Focus on the instructions issued by a processor, and put ordering constraints among them
  – when a load is seen by others
  – when a store is seen by others

• Define sufficient conditions so that a particular memory consistency model is supported

• Note that accesses issues by a processor to the same variable cannot be reordered.
Performing

- LOAD by Pi is performed wrt Pk when a STORE by Pk cannot affect the value returned by the LOAD
Performing

- STORE by Pi is performed wrt Pk when a LOAD by Pk returns the value defined by that STORE
• Conditions for satisfying Sequential Consistency and other models can be formulated so that….

… Process needs to keep track of requests initiated by itself ONLY
Sequential Consistency

- Before a LOAD is allowed to perform wrt any processor, all previous LOAD/STORE accesses must be performed wrt everyone

- Before a STORE …. (same)

/* Note GLOBALLY performed */
Sequential Consistency

LOAD
LOAD
STORE
STORE
LOAD
STORE

Program Execution
Processor Consistency

• Main idea: LOADs are allowed to bypass STORES

This LOAD bypasses the two STORES

… Honoring, of course, local dependences
Processor Consistency

- Before a LOAD is allowed to perform wrt any processor, all previous LOAD/STORE accesses must be performed wrt everyone.

- Before a STORE ....

  .... LOAD/STORE ...

/* Note GLOBALLY performed */
Weak Consistency

• Suppose we are in a critical section

• Then, we can have several accesses pipelined b/c programmer has made sure that:
  – no other process can rely on that data structure being consistent until the critical section is exited

• Adv: Higher performance (more overlap)
• Dsv: Need to distinguish between ordinary LOAD/STORES and SYNCH
Weak Consistency

LOAD/STORE
....
LOAD/STORE

SYNCH

LOAD/STORE
....
LOAD/STORE

SYNCH

LOAD/STORE
....
LOAD/STORE

Program Execution

Copyright Josep Torrellas 2003
Weak Consistency

1. Before an ordinary LOAD/STORE is allowed to perform wrt any processor, all previous SYNCH accesses must be performed wrt everyone

2. Before a SYNCH access is allowed to perform wrt any processor, all previous ordinary LOAD/STORE accesses must be performed wrt everyone

SYNCH accesses are sequentially consistent wrt one another
Release Consistency

- Distinguish between:
  - SYNCH acquires: e.g. LOCK
  - SYNCH releases: e.g. UNLOCK

- LOAD/STORE following a RELEASE do not have to be delayed for the RELEASE to complete

- An ACQUIRE needs not to be delayed for previous LOAD/STORES to complete

- Accesses in the critical section do not wait or delay LOAD/STORES outside the critical section
Release Consistency
Release Consistency

• Advantages: Higher performance
• Disadvantages: Need to additionally distinguish between ACQUIRE/RELEASE
Release Consistency

• 3. Before an ordinary LOAD/STORE is allowed to perform wrt any processor, all previous SYNCH ACQUIRE accesses must be performed wrt everyone

• 4. Before a SYNCH RELEASE access is allowed to perform wrt any processor, all previous ordinary LOAD/STORE accesses must be performed wrt everyone

• ACQ/REL accesses are processor consistent wrt one another
How to enforce these stalls?

- With Fence instructions
- Different types of fences present in current processors
- Check manuals of processors to see which types of fences are supported
Further Readings


• Culler and Singh course textbook

• Processors have their own memory consistency models: e.g. SUN’s PSO, TSO
Overlap of Operations

ACQ  RD  RD  RD  compute  WR  WR  WR  REL

See figure in Paper
Performance Gains from Relaxed Models

• Gains both in hardware and compiler
• Gains in hardware: Come from latency hiding
  – Overlap several memory operations: RDs and WRs
    • Need a lock up free cache (of course): multiple misses serviced at a time
    • Puts extra pressure on the buffers (read and write buffers):
      – have more transactions pending at a time
      – These transactions need to keep record until fully performed
• It also creates extra traffic

• See Figure 3 and Figure 4
Performance Gains in HW (II)

• Note that paper by Gharachorloo et al (ASPLOS) assumes a very simple processor that stalls on reads. Not representative of current processors
• See further readings for evaluation on Superscalar processors:
  – Allow multiple outstanding reads: Unlock more potential for relaxed models
  – But the computation is also smaller because of ILP
  – As a result: relative performance gains of relaxation under ILP can be bigger or smaller than under simple processor
Performance Gains in SW

• Common compiler optimizations require:
  – Change the order of memory operations
  – Eliminate memory operations
• Examples:
  – Register allocating a flag that is used to synchronize
    While (flag==0);
  – Code motion or register allocation across synchronization
    Lock L
    Read A
    Write B
    Unlock L
    Lock L
    Read A
    Read B
    Unlock L
• Sequential consistency disallows reordering of shared accesses
Performance Gains in SW

- More advanced optimizations such as loop transformation and blocking
- Relaxed models allow compilers to do more re-arrangements
Summary

• Release consistency model
  – Simple abstraction for programmer
  – Performance gains in SW and HW

• Relaxed models are universal in current multiprocessors

• Different manufacturers have different models