COL380

Introduction to
Parallel & Distributed Programming

- “A multiprocessor is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.” [Lamport, 1979]

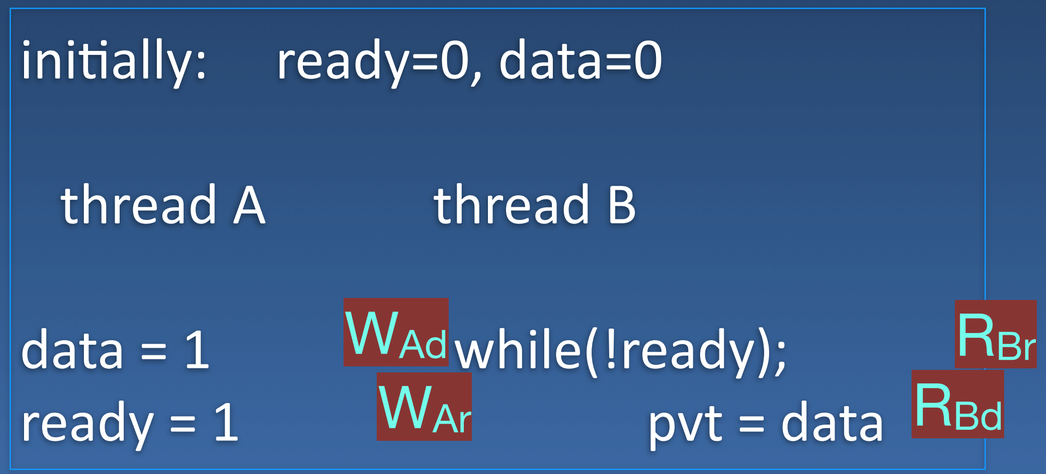
Memory Consistency Semantics

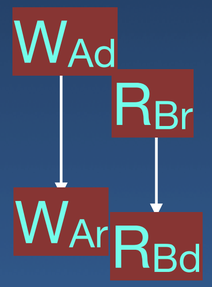
- Threads always see values written by some thread
  - ➡ No garbage
- The value seen is constrained by thread-order
  - ➡ for every thread

```
initially:    ready=0, data=0

  thread A          thread B

data = 1          while(!ready);
ready = 1         pvt = data
```

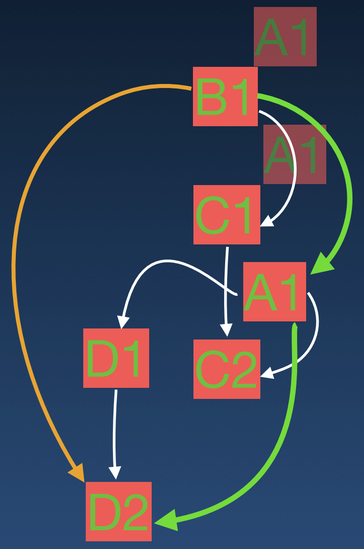




If B reads ready =	Then B may read data =
0	1
0	0
1	1

If B sees the new value of ready (1),
it must also see the new value of data (1)

Not SC



thread A	thread B	thread C	thread D
A1 x = **a**	B1 x = **b**	C1 y1 = x (**b**)	D1 z1=x (**a**)
		C2 y2 = x (**a**)	D2 z2=x (**b**)

NOT SC

Inefficient SC

- Hard to implement
  - ➡ Poor performance
  - ➡ A new memory operation cannot be issued until the previous one is completed
  - ➡ No re-ordering of instructions allowed

thread A	thread B	thread C	thread D
x = **a**	x = **b**	y1 = x (**b**)	z1=x (**a**)
		y2 = x (**a**)	z2=x (**b**)

- Alternatively, out-of-order execution is allowed
  - ➡ Detect and recover from SC violation
- More popular to simply switch to weaker models
  - ➡ with better performance

Inefficient SC

- Hard to implement
  - ➡ Poor performance
  - ➡ A new memory operation cannot be issued until the previous one is completed
  - ➡ No re-ordering of instructions allowed
- Alternatively, out-of-order execution is allowed
  - ➡ Detect and recover from SC violation
- More popular to simply switch to weaker models
  - ➡ with better performance

Relaxing SC

initially: ready=0, data=0

thread1	thread 2
data1 = 1	SYNC
data2 = 1	sav1 = data1
SYNC	sav2 = data2

Causal Consistency

- Write is causally ordered after all earlier reads/writes in its thread
  - ➡ write depends on the current ‘state’
- Read is causally ordered after causative write to the same variable
- Causality is transitive
- ∃ sequential order of causally related operations consistent with every thread’s view
  - ➡ Non-related writes may be seen in different order by different threads

Processor Consistency

- All threads see all memory writes by each thread in the order of that thread
  - ➡ all instances of write(x) are seen everywhere in th
  - ➡ Writes to different variables from different threads do not need to be seen in a consistent order
- Easy to implement
  - ➡ Two or more writes from a single source must remain in order, as in a pipeline
  - ➡ All writes are through to the memory

FIFO consistency relaxes this constraint

FIFO consistency is also known as **PRAM consistency**

Weak Consistency

- Notion of special synchronization accesses
- Synchronization accesses are sequentially consistent
- Other accesses are consistent with synchronization accesses
  - ➡ Before any regular read/write is allowed to be visible to any other thread, all previous synchronization accesses must be completed
  - ➡ Before a synchronization access is allowed to complete, all previous ordinary read/write accesses must be completed
- Non-synchronization accesses may be re-ordered
  - ➡ suitable for optimization

OpenMP Consistency

- Special memory ‘fence’ operations (flush)
- Order only fences with respect to each other
- Two flushes of the same variable are synchronization operations
- Weaker than “Weak”
  - ➡ Non-intersecting flush-sets are not ordered with respect to each other

Consistency Summary

Model	Description
Strict	Global time based atomic ordering of *all* shared accesses
Sequential	*All* threads see all shared accesses in the same order consistent with program order -- no centralized ordering
Causal	All threads see causally-related shared accesses in the same order
Processor	All threads see writes from each other in the order they were made. Writes to a variable must be seen in the same order by all threads
Weak	Special synchronization based reordering -- shared data consistent only after synchronization

Subodh Kumar

What's the strictest memory consistency model being supported? Why?