Project 3

Memory Ordering Demonstration in Multicore Shared Environments and Implementation of Sequential consistency and Total Store Order

Anoushka Kumar **Abhik S Basu** **Mohit Jain** **Ritwick Pal**

Group 14

Litmus Tests

Commands to run the programs inside litmus_tests folder:

```
Compilation:
make
Run:
./loadload || ./loadstore || ./storeload || ./storestore
```

Following are the litmus tests and their analysis.

1. **Store - Load Litmus test**

Assume initially X = Y = 0

Thread 1	Thread 2
Store X = 1	Store Y = 1
Load R1 = Y	Load R2 = X

Showing all possible execution states:

1) R1 = R2 = 0

Cycle	Core C1	Core C2	Comments
1	Load R1 = Y		The load and the store gets reordered for thread 1
2		Load R2 = X	
3	Store X = 1		Store reordered here
4		Store Y = 1	

In this case R1 = 0 and R2 = 0, in this case a store-load reorder gets detected. This scenario is not possible without reordering, since if no reordering has occurred then atleast one of the stores will set the variable before load.

2) R1 = R2 = 1

Cycle	Core C1	Core C2	Comments
1	Store X=1		
2		Store Y = 1	
3	Load R1=Y		
4		Load R2=X	

In this case R1 = 1 and R2 = 1, no reorder needed to reach this state.

3) R1 = 0, R2 = 1

Cycle	Core C1	Core C2	Comments
1	Store X=1		
2	Load R1=Y		
3		Store Y = 1	
4		Load R2=X	

In this case R1 = 0 and R2 = 1, no reorder needed to reach this state.

4) R1 = 1, R2 = 0

Cycle	Core C1	Core C2	Comments
1		Store Y = 1	
2		Load R2=X	

3	Store X=1		
4	Load R1=Y		

In this case R1 = 1 and R2 = 0, no reorder needed to reach this state.

2. Load - Store Litmus test

Assume initially X = Y = 0

Thread 1	Thread 2
Load R1=Y	Load R2 = X
Store X=1	Store Y = 1

Showing all possible execution states:

1) R1 = R2 = 0

Cycle	Core C1	Core C2	Comments
1	Load R1 = Y		
2		Load R2 = X	
3	Store X = 1		
4		Store Y = 1	

In this case R1 = 0 and R2 = 0, no reorder needed to reach this state.

2) R1 = R2 = 1

Cycle	Core C1	Core C2	Comments
1	Store X=1		The Thread 1 Load and Store get reordered

2		Load R2=X	
3		Store Y = 1	
4	Load R1=Y		Load reordered here

In this case R1 = 1 and R2 = 1, in this case a load-store reorder gets detected. This state cannot be reached without reordering, since if there is no reorder then atleast one of the loads for a variable will occur before its store.

3) R1 = 0, R2 = 1

Cycle	Core C1	Core C2	Comments
1	Load R1 = Y		
2	Store X = 1		
3		Load R2 = X	
4		Store Y = 1	

In this case R1 = 0 and R2 = 1, no reorder needed to reach this state.

4) R1 = 1, R2 = 0

Cycle	Core C1	Core C2	Comments
1		Load R2=X	
2		Store Y = 1	
3	Load R1 = Y		
4	Store X = 1		

In this case R1 = 1 and R2 = 0, no reorder needed to reach this state.

3. Load - Load Litmus test

Assume initially X = 0

Thread 1	Thread 2
Store X=1	Load R1 = X
	Load R2 = X

Showing all possible execution states:

1) R1 = R2 = 0

Cycle	Core C1	Core C2	Comments
1		Load R1 = X	
2		Load R2 = X	
3	Store X = 1		

In this case R1 = 0 and R2 = 0, no reorder needed to reach this state.

2) R1 = R2 = 1

Cycle	Core C1	Core C2	Comments
1	Store X=1		
2		Load R1 = X	
3		Load R2 = X	

In this case R1 = 1 and R2 = 1, no reorder needed to reach this state.

3) R1 = 0, R2 = 1

Cycle	Core C1	Core C2	Comments
1		Load R1 = X	

2	Store X = 1		
3		Load R2 = X	

In this case R1 = 0 and R2 = 1, no reorder needed to reach this state.

4) R1 = 1, R2 = 0

Cycle	Core C1	Core C2	Comments
1		Load R2=X	The Load R2 instruction has come before Load R1 instruction
2	Store X = 1		
3		Load R1 = X	As both Load instructions have inverted positions so its load - load reordering

In this case R1 = 1 and R2 = 0, in this case a load-load reorder gets detected.

4. Store-Store Litmus test

Assume initially X = Y = 0

Thread 1	Thread 2
Store X = 1	Load R1 = Y
	FENCE
Store Y = 1	Load R2 = X

Fence is added between the two Load instructions to prevent Load-Load reordering during execution.

Showing all possible execution states:

1) R1 = R2 = 0

Cycle	Core C1	Core C2	Comments
1		Load R1 = Y	
2		Load R2 = X	
3	Store X = 1		
4	Store Y = 1		

In this case R1 = 0 and R2 = 0, no reorder needed to reach this state.

2) R1 = R2 = 1

Cycle	Core C1	Core C2	Comments
1	Store X=1		
2	Store Y = 1		
3		Load R1=Y	
4		Load R2=X	

In this case R1 = 1 and R2 = 1, no reorder needed to reach this state.

3) R1 = 0, R2 = 1

Cycle	Core C1	Core C2	Comments
1	Store X = 1		
2		Load R1 = Y	
3		Load R2 = X	
4	Store Y = 1		

In this case R1 = 0 and R2 = 1, no reorder needed to reach this state.

4) R1 = 1, R2 = 0

Cycle	Core C1	Core C2	Comments
1	Store Y = 1		
2		Load R1 = Y	
3		Load R2=X	
4	Store X = 1		

In this case R1 = 1 and R2 = 0, in this case a store-store reorder gets detected. This state cannot occur without reordering in store store, as if no reordering occurs and R1 is assigned 1, that means that X has been set to 1 as well (since X is set before Y) and hence R2 should also have a value of 1.