Memory Consistency Models

November 2020

1 Introduction

When we write some code the processor may not know the ordering in which the operations should be executed. So, to get the expected result we need to specify and implement some rules of ordering. In a single core processor it is easy to implement program order. But for multi thread codes and multiple cores it is not straight forward. Memory consistency models describe the rules of ordering memory operations on different threads and cores.

To make computers faster multiple cores are used which execute operations parallelly. Now if different cores are working on same data there is very high possibility of giving different outputs. So, we must strictly impose some orderings without compromising too much on efficiency.[2]

2 Memory consistency models

2.1 Sequential consistency model

This is a very rigid ordering model. Rules are as follows:
1) Stores occur before subsequent the loads.
2) Loads occur before subsequent stores.
3) Stores occur before subsequent stores.
4) Loads occur before subsequent loads.
Ordering with respect to program order.
The sequential consistency model is very rigid. Thus, performance is low.[4]

2.2 Relaxed consistency models

2.2.1 Total store ordering

In this memory consistency model, the rule – "stores occur before subsequent loads" is relaxed. Since, the store operation takes more time, the loads are allowed to happen irrespective of stores. Load returns the latest stored value. Here, we may use write buffers specific to the individual cores which stores the write values until it is written into the main cache. Write buffers implement

FIFO order. If a load operation within same core asks for a variable which is available in write buffer, then that value is returned instead of main cache value. Fences can be used to make the operations before the fence mandatory to be completed before proceeding.[4]
EXAMPLE: Initially, A=0, B=0, C=0

Thread 1	Thread 2
A=1	B=1
Read B	Read A
	C=1
	Read C

Most probable read output with respect to TSO A=0, B=0, C=1. Here (ABC) = (001,101,011) are also valid with respect to TSO but the clock cycles required for store in main cache are usually more than the clock cycles for going to next step and load operation.

2.2.2 Partial store ordering

This model can be said to be an extension of TSO. In this memory consistency model in addition to "store then load" relaxation, the processor can also reorder store operations provided they occur at different memory addresses.[3]
EXAMPLE: Initially a=0, b=0

Thread 1	Thread 2
a=1	
a=2	
	b=1
	L1 : r=b
	if (r!=1) goto L1
	Read a

Even though we are ensuring that write 'b=1' is completed we may get Read output as 0 when write to a and b are rearranged.

2.2.3 Relaxed Memory ordering

In this memory consistency model all operations are allowed to be rearranged and if we want to impose an order then we have to explicitly put fences. Fences are of different types like load-store (all loads before the fence must be completed, to store memory after the barrier. Equivalently, we can say loads before the fence can not be reordered with stores after the fence), similarly we have load-load, store-load, store-store fences.[1]
Some of these are usually used in combinations. So, these have got a specially named fence:[1]
Full fence = load-load + load-store + store-store
Acquire fence = load-load + load-store
Release fence = load-store + store-store

EXAMPLE: Initially a=0, b=0, c=0, indicator=0

Thread 1	Thread 2

a=1
b=2
c=3
store-store fence
indicator=1
store-load fence
	
	if (indicator==1):
	load-load + load-store (acquire fence)
	a = a+b+c

a=6, b=2, c=3

2.3 Conclusion

So, we can see the speed of computers can be increased by assigning different operations to different cores. But the programmers and designers have to be careful while distributing the operations. They have to consider the allowed reorderings.

References

[1] Fences are memory barriers.

[2] James Bornholt. Memory consistency models.

[3] Xiao Fangyihua, Martin. A summary of relaxed consistency.

[4] Madhu Mutyam IITM. Memory consistency.