Parallel Computer Architecture
Prof. Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Week - 11
Lecture - 57

Lec 57: Relaxed Consistency Models (2)

Hello everyone. We are doing module 7 on memory consistency. We are continuing the topic of relaxed consistency models in this lecture number 4. So, safety nets to relax the W followed by R order. So, if we want to preserve the program order, that is we still want to maintain this order in the program, okay. That is, I have written the program that I want this order to be maintained. It was running under an SC machine and now I want to run this program on a machine which supports the TSO or the other models, okay, other relaxed models.

So, how can we make that happen, all right. So, we will see that for all the 3 models we are discussing. So, for the IBM 370 model, we have a special serialization instruction available to do this. So, if the programmer wants to maintain the W followed by R order, then the programmer has to insert this synchronization instruction here between these two.

So, this is the write and this is the read. So, between the write and the read, if I want to maintain the order, I need to insert this sync instruction, okay. So, the semantics of this is, if a sync instruction is there, everything before that is completed before it starts for the next instruction. So, this write to flag 1, then we have the sync, and then we have the read of flag 2, right. This and this will not be reordered because of the sync instruction.

So, this sync instruction acts as a safety net. The implementation of this could be a variety of things available, one option is to use a compare and swap instruction. In the TSO model there are no explicit safety nets available, but we have this RMW instruction. So, read modify write. As the word says we have to read, suppose this is a location a, and I want to atomically modify this location.

So, I am going to use the instruction RMW which says read the location a and then modify it. That is I want to change, suppose it was equal to 0. I want to change it to 1. So, I will write it like this. So, this means read the location, change it to 1 and write it back. So, this whole thing of reading it and modifying it and then writing it back, all this happens atomically. This is guaranteed by the hardware.

So, this happens atomically, that is nothing else can happen when this instruction is executing. So, that is the RMW instruction. So, you can use this for writing. If I want to use this for a read, suppose I want to replace the read with an RMW, what will you do? If I do RMW of a. Well, I do not want to change the location, but I want to read what is there. So, once you do RMW you are going to get the value of a and now you want to do the write because that is part of this instruction. So, what will you write because you are not supposed to change the value.

So, what am I going to do? The value which I have recently fetched that same value I will pass as the second argument. Okay. So, when I am writing, I will send the value I want to write and when I am reading, I will send the same value which I have read out just now from the memory. Okay. So, this is how I can use the RMW. So, for the TSO model, we are going to replace the reads and the writes with the RMW. Okay. So, that was our program where we change the flag to 1.

So, I am going to replace the write instruction with this RMW instruction or you could do this or you can also change this. So, any one of them if you change, it is sufficient because when an RMW is there it will not be reordered. Okay. So, if the programmer uses this, this operation will provide an illusion of the program order between the read and the write order. Okay. So, I can replace either the write or the read with RMW because once an RMW is installed, it will not be reordered. So, that is the safety net for TSO. Okay.

So, in TSO the RMW prevents the writes to any location. So, it could be for the same location flag 1 with flag 1. So, if you have a write to F1 followed by read of F1. So, even there it works and it also works for write F1, read F2. So, across all locations the RMW will work, but in PC the RMW only prevents the writes to the same location.

So, if I am doing RMW on one location, the other locations, that is other variables are permitted to change whereas in TSO it applies to any location. So, when one RMW is executing no other location can change whereas in PC model, when one RMW is executing, only that particular location cannot change, but other locations can change. Okay. So, that is the restriction with PC. However, I can easily replace the reads with the RMW which work fine. Okay. So, therefore, our RMW does not prevent others' writes from being completed and reordered.

So, we need to look at this aspect. Okay. And about write atomicity, write atomicity, the IBM 370 model does not require anything. Why because it does not relax write atomicity. It says you cannot read your write early, you cannot read others' writes early.

So, both restrictions are not there. In the TSO model, we allow to read our own write early.

So, if I soon write early meaning, if I write to the location A and then I read the same location this is permitted and I want to stop this reordering, okay. Write A followed by read A. And I want to maintain this order, that is do not allow reordering. So, for this you can simply use the RMW instruction in TSO because TSO only we have one type of write atomicity relaxed. Whereas in PC model, you have both types of write atomicity relaxed, that is you can read your write early as well as others' writes early. To read your write early, it is very straightforward, similar to this replace it with RMW.

But in the cases when you are permitted to read others writes early, so that you cannot control with RMW. So, what is happening, if you replace a write with an RMW, it only prevents that particular location from being written, but other locations can still be written. So, in this case I have control over the read instruction to be replaced with RMW, because if I am changing a location and some and there are three more readers they can still read before others, right. So, read others' write early is permitted in the PC model. So, if I am reading somebody else's modified value, then I have to replace my read with an RMW, and I also have to guarantee that all the others replace their reads also with RMW.

So, the control is not only with me in my program, but I also need to make sure that other processes also change their reads, writes to particular or appropriate RMW instructions, okay. Right. So, in the PC model, our own write order can be maintained using RMW whereas, others' writes can be not, we cannot control it alone using RMW because others' writes become visible to everybody earlier. So, what do we do? So, in this case all such reads in other processors have to be replaced by an RMW instruction to prevent them from reading it early. So, it is not a local solution, but the solution for write atomicity in the PC model relies on other processes cooperating and replacing their instructions, okay. So, this is how, we can see as an example. We have these three processes: the green pink and the yellow. There are some instructions inside this.

So, here I have taken an example for a write followed by a read order on the same variable. So, to maintain this order, I can replace it with RMW. So, the same thing I do for the pink and the yellow process. So, every process replaces its same location with RMW, if they want to maintain that order. Now the question is for B.

Now this B is not modified by the green process, it is modified by the yellow process. Similarly, the pink process also accesses B which is modified by the yellow. The yellow process accesses D which is modified by the pink process, okay. And here, this green

process is permitted to read the B before the pink process has even seen it, right. So, this can do an early read over the pink process.

So, how do I stop this from happening? So, how do I make sure that everybody sees the new value of B before I can start using it, okay. So, that is the restriction we are looking at. Here, we need to make sure that all such reads in other processors must be replaced by the RMW instruction. So, all these reads to shared variables will now be replaced by RMW instruction, okay. So, if I just look at the green process with respect to variable A, my solution was very local. I just changed my own instruction.

But with respect to variable B, we had to cooperate with other processes, to make sure that everybody changes their reads to RMWs, okay. So, overall we have to see that this RMW instruction is not reordered and the hardware guarantees that and therefore, the SC holds. So, sequential consistency will hold, if you appropriately insert the RMW instructions. Of course, the advantage comes with a small disadvantage. What is the disadvantage? The system's RMW implementation may not be suitable for you or the other problem is, every read gets replaced with a write and you understand from coherence perspective that, whenever you do a write all the other copies will either get updated or invalidated.

So, unnecessary invalidations happen just because we are replacing the read with the RMW instruction, okay. But still we need to use that because the relaxed model to guarantee sequential consistency, the safety net is a mandatory requirement, okay. So, that was for the 3, 2 models. In the SUN Sparc model, we have the memory barrier instruction instead of RMW. It uses the MEMBAR instruction. This instruction says that, whenever MEMBAR is encountered, all the instructions before the MEMBAR should finish before the instructions after the MEMBAR can be issued, okay. So, this is how it works. I have a write followed by a read. I simply insert a MEMBAR in the middle. If I insert this, then the reordering will be canceled or it will not be allowed, all right.

So, we have established the relaxation of W followed by R, but it is not very beneficial because the reads and writes are very finely interleaved in the program and I intend to relax more orders. So, I also want to relax these orders, okay. So, let us relax more orders, that is W followed by R and W followed by W. So, we have seen this. Now I add an addition one. So, this new one if I add, what advantage does it give me, right.

So, write followed by write again. I am writing to two different variables and hence I can use many optimizations, like I can use a write buffer merge. So, here write buffers are there and in this I am going to collect variables before they are sent to the main

memory, okay. So, they temporarily reside in the write buffer and then they are sent. Suppose I have changed some variable here, here and suddenly I have a nearby address which can be merged in this buffer. So, this variable change, this variable change and then another variable change which was able to merge with this entry and you can see that this is the third write and not the second one.

So, I have actually reordered the writes because I could merge this and send the data in one go to the main memory. So, this write buffer merging optimization is possible, if I permit a write after write reordering and in some cases if you have cache hits or memory hits, the writes can bypass each other. The write misses can be overlapped because if you are first ,if you have a W1 followed by W2 and this is a cache miss. If this is a cache miss, then you do not need to wait for this and you can of course, finish the W2 which turns out into a cache hit and that is very much okay from performance perspective. So, under this we have a single model the Sun Sparc PSO is the only model in this category.

What about write atomicity? It guarantees write atomicity by other processors. So, if you see example number 3. This example. Here we had a write and a write and I am reordering them. So, I permit them to change the order. If the order can change, my sequential consistent behavior will be violated. So, what will happen? P2 will see the new value of flag and the old value of A. So, that is possible if P1 reorders the writes to A and the flag.

So, how do I solve this problem? Okay. So I want to do this and the Sun Sparc model provides me with a STBAR that is a store bar instruction which is similar to the memory barrier. So, MEMBAR is a memory barrier instruction, STBAR is a memory barrier with a variant of store instruction. So, if I insert a STBAR instruction between the two writes, the first write and the second write. Somewhere in the middle, if I insert STBAR, then they will not be reordered, okay. So, this is what I am going to do. I am going to replace or rather I am going to insert an STBAR instruction in P1 and you can see nothing has to be done for P2 because both are read instructions. What about WR? You have to use the simple TSO model, you can use an RMW instruction.

If you have W followed by W, you can use this STBAR. If you have W followed by R, you can use RMW. So, that is the safety net for that model. And how would you implement a STBAR in a FIFO, that is we have a queue of writes. There are several writes which are queued up and then eventually they will go to the memory. So, I am modifying A, I am modifying B and suddenly this STBAR comes. So, I am going to make sure that I logically implement this STBAR as part of this FIFO, okay.

So, when an STBAR is encountered, it is inserted in this first in first out queue, making sure that these writes before to this. So, these are the earlier writes and C and D are later writes. So, A and B finish writing before C and D are written, okay. So, it makes sure that writes before it are completed and retired before the new writes can be started. Our implementations are variety, but one possible implementation is you can simply use a counter. Whenever a write is sent to the memory you increment the counter, when you get an acknowledgement back from the memory, you can decrement the counter.

So, that you can maintain how many writes were pending and how many have finished. So, all the old writes will finish, when your count becomes 0, okay. So, this was relaxing the WR followed by and the W followed by W order with the necessary safety nets, okay. So, this was good enough, we have relaxed almost all the program orders and the write atomicity requirement in certain cases. But we are not satisfied with this because these are not sufficient for the compiler. We want more reorderings.

So, can I relax all orders, okay. So, that is the topic of the next lecture. Let us see a quick summary of the orderings we have relaxed in this lecture. The first 4 rows. So, we have seen these 4, IBM 370 model which relaxes write followed by read, PSO model, which relaxes the write followed by read and the write followed by write. So, it relaxes 2 orders. Related to the write atomicity, IBM does not relax anything whereas, TSO allows to read your own write early and PC allows to read yours as well as others writes early. PSO allows own write early, okay.

And the last column now, okay, you can understand that these are the safety nets available for all these models. Now the models in these rows will be discussed in the next lecture. These are models which relax all possible orders, okay. And some more real world implementations of these relaxed orders are given here, okay. So, with this we finish this lecture. Thank you so much.