Parallel Computer Architecture
Prof. Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Week - 10
Lecture - 54

Lec 54: Sequential Consistency

Hello everyone. We are starting module 7 now. The title of the module is On Memory Consistency. This is lecture number 1, where we are going to discuss about sequential consistency. Okay. So, before we begin this topic, let us understand what was coherence. Okay. Coherence and consistency are the two things we need to satisfy in these parallel architectures.

So coherence was guaranteeing that we have a sequential order of accesses to a location. Reads and writes interleaved across different processes and we make sure that when a write happens all the other copies get invalidated, then the write takes place and then the subsequent read to this write will get the newest value which was written. Alright. So, a location always gets the most recently written value to that particular variable. However, it never told us how early are we going to get this value. That is when I wrote a value of x equal to 2, is it at the next millisecond or nanosecond interval which is going to be effective or it will be effective after several hours.

So, we do not know when the value will be available. So, coherence only said that value will be eventually available. It never said how early and when it will be available. So, when this write which we have done becomes visible is not mentioned by the coherence and hence coherence is not sufficient. Because at times I want that my read should return me a particular writes value. For example, a process change the value of A and the other process wants this particular value of A.

It does not rely on arbitrary interleaving. It does not say that you interleave anyhow and I am okay with any value you will give it to me. Because the interleaving is always arbitrary. Every time you run the set of programs you will get a different interleaving. So, this arbitrariness is not desirable in all situations. So, if I want to control the value of the read which I am getting, how do I enforce it, because coherence does not help me to do this. Okay.

So, let us understand this more. So, here, suppose I have this process in running on a program processor P1. So, this blue color is just a random sequence of read write

accesses and I have another process with these yellow and brown accesses. So, when I send the second read, the brown color read, what value of A is it going to get. So, here if I see a program order will be followed and the value which this blue read gets is equal to 2. Because it is following this particular write.

But if I ask you, what is the value this read will get? Will it be 2? Maybe. And if I say initially suppose A was equal to 0, when you began the program, will A be equal to 0? So, well it depends on the type of interleaving which took place. Suppose this brown read happened first and then this write happened and then this and then this, in this order you would say A is equal to 0. But if this write A equal to 2 happened and then this way in that case my value of A will be equal to 2. There could be further interleavings which would say that if I have 1, 2, I come up to here and from write A equal to 4, I do that read and then I come to this in that case my value of A will be equal to 4.

So, we have these different values and they are perfectly alright when we consider write serialization because eventually the variable here is getting the correct value according to the interleaving sequence. In the red, it got the value A equal to 0 because A became 2 after that and so on. But if I want to control what value this yellow read should get, if I want to say that or enforce that I always want this to be 4. Then I need to do something more which coherence does not give me. Similarly, if I have another read after that. So overall the type of interleaving which will take place here, will determine the values of the reads. So, in case I want to enforce that the read A should return me the value of 4, that is I want to establish an order between this.

So, this write should happen before this read, then what should I do? So, I would need to establish an event synchronization across them. So, when do I need event synchronization? When I want a read to return a value of a particular write. And this cannot be done using coherence. Okay. So, let me take an example. Here, I have a process P1 which will do a value of A equal to 1. And I have another process P2 which wants to print the value of A. Now my requirement is that I want this value to be printed.

Similar example, the previous slide you do not know the interleaving and the value of A could be anything given that A is equal to 0 in the beginning, this print A could print either 0 or 1. So, we cannot ensure that. So, to ensure this we need to establish a synchronization event across them. So, how would we do that? What I will do is, I will use flag and I will put a while loop here and I will say while the flag is 0, keep waiting, and in P1 we will make that flag equal to 1. So, as long as the flag is 0, P2 will not come out of the while loop and whenever P2 comes out of that while loop, you will know that this flag equal to 1 would have executed and programmers intuition is this order would be maintained and hence the value this will print will be equal to 1.

So, this is how we can establish an event synchronization if I want a particular value to be returned. Okay. So, from the programmers intuition print A results into 1, and how have we done that using a flag as a synchronization variable. So, alone coherence was not sufficient to do this, we had to do something more. Alright. One more example. Here, you can see process P1 and P2. P1 is doing these two writes A equal to 1 B equal to 2. P2 is doing the two reads. When I say print, print is nothing but read the value of A and print it on the output screen. Okay.

So, if you try to analyze this program, so when we do this lecture I would encourage that you pause the video several times to understand what the processes are doing and then try to do it yourself before moving on. Okay. So, here the programmers intuition, is it clear to you? I would say, we do not know, because the two processes are in parallel, one is just writing the variables the other is printing and we do not know what actually the programmer intends to do this. So, overall the programmers intuition is not clear here and whatever could be the answer, coherence has nothing to tell us about this. So, what do we actually need? So, what we need is in the previous example, we were not sure what the output would be and the output could be variety, there could be one two or three different outputs which will come and when you write programs, you do not want multiple outputs, you want one deterministic final output and as a programmer you should be able to reason about it and you should be able to confidently say that, when I run this program this is my output. Whereas in the previous example you do not know what that output will come, right. Because coherence alone was not able to tell us anything about it due to the arbitrary interleaving. So, how do I come to a conclusion or how do I give a clear semantics of what is happening? So, for that I need a formal model, an ordering model which is there in the system, which will be used as a reasoning basis by the programmer to judge what is actually happening. Okay.

So, this model which we will discuss, the ordering model is called the memory consistency model for the shared address space. Now, this consistency model is going to specify the constraints on the order in which the memory operations must appear to be performed. Okay. They may or may not be performed actually in that manner, but they appear to be performed in a particular manner. And this ordering will include operations for the same location or for different locations, by the same process or across different processes. Remember that coherence was only talking about relationship between different processes operating on a single variable. The same variable, shared variable, whereas consistency talks about different processes amongst each other and across different variables also. Okay. So, overall this is kind of a super set.

So, I would say that memory consistency subsumes coherence. Okay. So, what we need

is an ordering model for my clear semantics for my program and it should be across all locations. So, that the programmers can reason about the result because in the previous example I did not know what the result would come. So, that is my memory consistency model and it is defined for the same location or different location and by the same process or different processes. So, that was memory consistency model.

Now why is it important? It is important because it helps me to define a correct behavior of my program.Okay. And memory consistency model will be implemented by the hardware and hence, it is a contract between the system and the programmer. So, the system is saying I am going to give you such and such ordering model and the programmer can then rely on that model to reason about their programs. Well, coherence was not visible to the software and it is not at the ISA level, but for consistency the system provides that as a contract. So, the consistency model is known to all the layers which we saw in a software development. So, the compiler has to know, the operating system has to know.

So, every layer in the development cycle has to know about the consistency. Right. So, I can have a same consistency model and because coherence is not seen by the software, we can always have newer and fancier coherence protocols implemented. So, I can have newer coherence protocols using the same consistency model. So, what does this consistency model do at the end? Well, it only restricts the ordering of loads and stores. It tells us what will be the order of load and store messages and it does not care about the ordering given by the coherence. And these load stores are across same location as well as different locations. Okay.

So, memory consistency comes in various flavors. The fundamental one is called the sequential consistency. So, sequential consistency is the main foundation of understanding memory consistency models. Once we understand sequential consistency, we will elaborate on its limitations or the restrictions it puts and these restrictions will motivate us to become more relaxed and give more easier or relaxed memory consistency models and so on. So, when we relax something, we are going to say, pay some price.

So, for that we will have to have more programming level or instruction level constructs to be developed or introduced to, at the end of the day satisfy sequential consistency. So, sequential consistency if I try to describe informally, it is saying that, it is a desirable ordering for the shared address space, that allows me to reason about multi-threaded programs and when a multi-threaded program runs on a uniprocessor or unicore system, how is it running? Multiple threads, using the same processor, the same memory. So, they get arbitrarily interleaved. They run one after the other in some manner and that is

why they work correctly. But if I move the same multi-threaded programs to a multiprocessor environment they are going to run on different processors.

They will run more parallelly than in a uniprocessor system, but even in that setup, the outcome should be same as it was in a uniprocessor system. Okay. So, that is our objective and this was formally defined by Leslie Lamport in 1979 using this definition. So, let us look at the definition. Okay. So, Leslie Lamport said that multiprocessor systems are 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. Okay. So, we are saying system is sequentially consistent if the result is same as if the operations were executed in some sequential order. Okay.

So, there was an arbitrary interleaving. Right. You have a total order with operations coming across different processes. Right. So, there is an interleaving happening and in this interleaving we get the total order. So, there is a sequence and all the operations happen in this sequence and when I try to understand or infer what operations belonging to a particular process happened, for that process all the operations took place in the program order. So, individual processors' actions appear in this sequence specified by its program. Okay.

So, this was our model. So, we have these multiple processors each running sequential different threads and when they all come and access the same memory, we have the switch which randomly selects either to execute this instruction or this one or this one. So, we are going to have an arbitrary interleaving between different programs. I can have two instructions of P1, one of P2, another of P3, again of P1. So, I can have an arbitrary order defined at the memory level or overall the system level. But what we need to guarantee that, across the sequence, if I extract the P1 subset it is going to be in the program order of P1. Okay.

So, within the process we have the program order and across the processes, we have some interleaving and I am not interested in that interleaving. Let it be anything because it is not guaranteed that every time you run that you are going to get the same interleaving. Okay. So, to understand this further with that definition, I would say that each process issues memory operations and completes them, one at a time, atomically and in program order. So, if we are able to do these three things we will be able to establish sequential consistency. So, when I say one at a time, it means memory operations are done by the process and when one operation is done it is guaranteed or it makes sure that until this operation finishes it does not issue the next operation.

Suppose I have read A and then write B, I am going to make sure that read A finishes and only then the write B takes place. Once I do write B, I make sure that it finishes, only then I do the next instruction. So, I do them one by one. Okay. I am not, so indirectly I am not allowed to overlap operations. I have to do them one by one.

The second condition is atomically. Atomically now I hope you understand that it means globally. So globally we have to make sure that when write happens ,that write has happened atomically, that is globally across all the processes. This write's value is seen by everybody in the system before we can do further operations and then everything definitely happens in the program order. Okay. So, we have to do them one at a time. So, we do not have, we cannot issue a new instruction until the previous one has finished.

We have to do it atomically, that is everything should be globally visible, that is every process in the system should see this action completed before the next one can begin. And it is not important. Of course, we say program order should be followed, but we can be little relaxed here and say tha,t although the program order may not be followed, but the end result should look like it is in the program order. Okay. So, they should appear to complete in manner which says that the program order was followed. And when I say program order it is as seen by the programmer and not as optimized by the compiler. Because program order as a programmer, if I am reasoning, I will only see the program I wrote and not what was optimized by the compiler. Okay.

So, we will take an example. So, in this, we have two processes P1 and P2 and P1 is doing instruction 1A, 1B, P2 is doing 2A and 2B. Now, what matters here is the order in which the operations appear to execute, not the chronological order of events. So, how do I understand what happens in this program? Well, we were discussing this few slides ago and we said that we do not know what the programmers intuition is and what is desired, but we could definitely derive what are the possible outputs. So, the possible outcomes are 0 0 when would that come? So, I would, let me list.

So, A B is the pair A,B, right. So, A is 0, B is 0. When will you get this? You will get this if the prints happen before the assignments. So, I will say the 2A happens, the 2B happens and only then the 1A and the 1B take place, right. If I have this sequence, this is my arbitrary order then my output will be 0 0. I can have an output of 1 0, when would this take place? It happens when I do a 1A and then the B equal to 2 does not take place, then I have the 2A the 2B and then the 1B, okay. So, you can pause the video and try to solve this for yourself and I would encourage that you try it for 1,2. Okay.

Now what about 0 2? Now will this happen? A is printed as 0 and B is printed as 2. So, let us see if this is possible. So, A is 0 means print A gave me answer 0, that is 2B

when I say A equal to 0 it implies that 2B took place first, and only then A became 1, right. So, that is how we will print A equal to 0, because we first print A, and then we assign A. So, when I do this, what does this imply? If I follow the program orders what is the program order for P1? It is 1A then 1B and for P2 it is 2A then 2B.

So, if I elaborate the sequence, I can say that I have 2A 2B, I am just putting 2A in front of 2B and then the 1A and then the 1B. So, I elaborated that sequence and this told me that we have 2A and then I have 1B. So, I have just derived from the sequence saying that 2A happened before 1B. Now, when 2A happens before 1B, what would you say? 2A is print B and this will print the value of B as 0. Okay. Now let me do the second part of this derivation, where we will start with B equal to 2.

If B is equal to 2 what does this imply? 1B has taken place and after 1B, the print B happens. Okay. So, this is how B will become 2. Now if I complete the program order, 1B has taken place and before 1B, 1A should take place. So, I can say 1A 1B and then 2A 2B. With this what will we get? So, if I derive a subset of this using the B, here, I will, I can say that 1B happens first and then 2A happens, right.

So, see this is the small part of the sequence. Okay. So, what you got? 1B 2A and here you got 2A 1B. Okay. So, with this we can say that this is a contradiction. So, they both contradict each other and therefore, we can say that 0 to outcome is not possible, if I am following a program order. Now let us do the 1 2. Now for achieving 1 and 2, I can get that using having 1 and 2.

So, we have 1A 1B then 2A 2B. This will print values as A equal to 1 and 2. If I execute the sequence as 1B, then 1A, okay. I have just swapped the first 2 pairs and I have swapped the other 2 pairs. What about this? The one in the angle brackets. Even for the one in angle brackets your answer will be 1 2 which is correct.

However you will argue that well, the both the processes did not follow the program order, okay. So, that is what we want to understand that it does not matter in which order the operations take place, but the end result should be sequentially consistent, okay. So, the same thing is written here, okay. So, it does not matter that they actually execute and complete in out of order, because in that example we said that you could do 1B then 1A and 2B then 2A instead of, instead of, 1A then 1B. So, I have broken the program order, but still the outcome is as expected and hence we can permit this sequence, okay.

So, with that we get some feel of what sequential consistency means. And if you want to implement sequential consistency, we need to satisfy 2 constraints. The first is definitely follow the program order as long as the result, visible result is satisfying the

program order, okay. You can have some interleaving done or some reordering done as we did in that example, but the end result should look like it completed in the program order.

And the second condition is atomicity. Now in the hypothetical total order which we construct which is an interleaving of accesses across all the processes. So, we would say that the operation which is done by 1 process, it is visible to everybody else in the system, okay. So, when I write A equal to 2, I see A equal to 2 and everybody in the system sees A equal to 2, at the same time. So, that is called write atomically finishing a task, okay. So, I need to guarantee program order and also I need to guarantee atomicity.

If I do these 2 things then I am guaranteeing to implement sequential consistency. Once sequential consistency is implemented, the programmer can reason about the outcome of the program. So, program order and atomicity are to be guaranteed. And atomicity is a tricky point because when we say process is doing 2 writes, this write of A done by P1. I want this write to be seen by everybody, it should be seen by all, by all includes myself and others.

So, everybody should be able to see this. Only after this I will be permitted to do the write B, okay. Because as there are multiple copies in the system, when write A happens, we have to make sure that the copies are either invalidated or updated and so on. So, until that is finished, we cannot do the write B. So, effect of write B is not even seen by P1 because if you can say well why not P1 can start writing B, but no it is not permitted according to the write atomicity requirement. Because write A which is the preceding instruction has not yet finished.

So, this write atomicity is extending our write serialization which we were discussing in our coherence topic, okay. So, write serialization was with respect to the same location and write atomicity caters to all locations. So, if it is same location, it is serialization all locations, it is atomicity. So, let us take an example, okay. So, here look at P1, P1 is assigning A equal to 1, P2 is actually waiting for A to become 1 and only then makes B equal to 1.

P3 on the other hand waits for B to become 1 and then prints A. So, if you see intuitively, what would be the programmers expectation, that the value which will get printed here is equal to 1 by transitivity, correct? Although P3 does not directly interact with P1, but via P2 it is going to guarantee that, it should get the value of A equal to 1. So, that is the programmers intuition, okay. And here is where the write atomicity is useful because if you see A became 1 and this value becoming 1, either it will invalidate all the copies, where are the copies? This is one copy of A with P2 and another copy of

A with P3, right.

They both of them could have cached them. So, either invalidation reaches or the update reaches, but now P1 made A equal to 1 and P2 was lucky to see it immediately. So, P2 saw A equal to 1, it came out of this while loop and made B equal to 1. Well, here P2 did not wait for P3 to see A equal to 1, because this A equal to 1 is taking the longer path in the system to reach P3. So, it is going to take a slow path. Hence P2 sees A equal to 1, but P3 does not see A equal to 1. In the meanwhile, P2 changes B to 1 and this is also a fast path.

So, P3 sees B equal to 1 and what value will it print? If I do this, it will print the value of A which is the A equal to 0 which is the old value. Now, why would this happen? Because P2 took the new value of A and it did not bother whether others in the system have seen the value of A. So, here P2 waits for A to become 1 and then makes B equal to 1, P3 waits for B to become 1 and then prints A and if P2 is allowed to continue beyond the seeing A equal to 1, what will happen? It will assign B, P3 will see the B and it will print the old value of A. So, what should happen? P2 should not be allowed to go past the read A until P3 has also seen that value of A and this is what is guaranteed by write atomicity. Okay.

So, I hope now you appreciate the requirement of write atomicity. Program orders should definitely be followed because that is what the programmer wants you to implement or execute correctly and write atomicity is required for such scenarios. So, what are the sufficient conditions for preserving sequential consistency? So, every process is going to issue memory operations in the program order, yes, it will. After every write operation is issued, that is it leaves the processor, the issuing process has to wait for this write to complete across all the processes, including itself, before it can do the next operation, that was for write. After a read, the issuing process which is doing the reading, it has to wait for all the processes in the system to see this value which it is reading.

So, the write which gave it this value that write should be visible globally. In the previous example when P2 read the value of A equal to 1 it did not wait for P3 to see the value and that created the problem. Okay. So, this condition which we are discussing, it says that if a process reads a value then it should wait for all the processes in the systems to see this value, effect of thewrite which produced this value before it proceeds. So, this third condition for write atomicity is rather demanding. Right. The program order, wait for the write to complete across the system and after a read, the issuing process waits for the read to complete across the system. This third condition is, which ensures the write atomicity, definitely it is quite demanding to do.

So, these are sufficient conditions for preserving sequential consistency. However, we can do with slightly lesser serialization constraints, but those we will see little later. So, with that understanding of sequential consistency, we will now see that whether this SC is easy to implement, does it have impact in the system and what are the main drawbacks of sequential consistency. But prior to seeing the drawbacks, we will see its requirement and then the drawbacks. So, that we will see in the next lecture. Thank you so much.