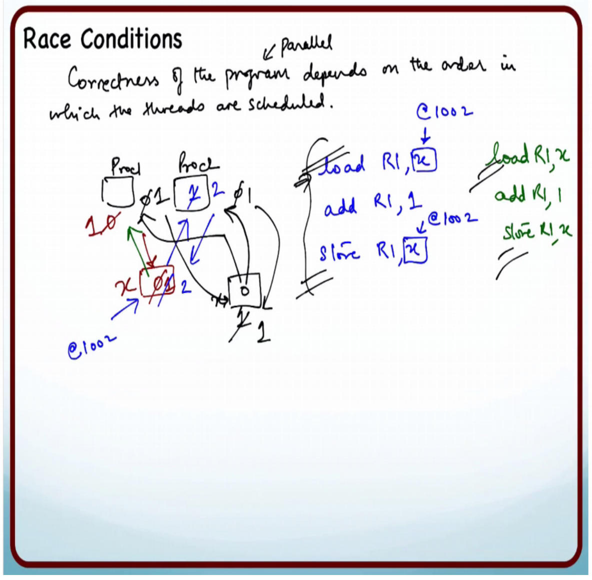
Introduction to Parallel Programming in OpenMP

Dr. Yogish Sabharwal
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 18
Race Conditions

(Refer Slide Time: 00:02)



Race Conditions

Correctness of the program depends on the order in which the threads are scheduled.

So what is a race condition? So, whether your program runs correctly or not obviously, you want your program to do something right and whether at outputs or it does what it what you intended it to do, the correctness of the program, if it depends on how the threads are scheduled in which order they are scheduled right and how the scheduling is interleaved. If it is dependent on that then we say that there is a race condition a condition exists; and when you run your program the race condition may happen and it may cause a problem or it may not cause a problem.

So, let us see some very very simple examples of race conditions. So, let us start with very simple case where you have processor 1 and you have processor 2 and you have a shared variable x and both these processors want to increment this shared variable x right how does a processor typically increment a shared variable. So, you will say load x to R 1 right. So, you are loading x in through register R 1 and then you say add one to R 1 right see we are adding one in the register and finally, you will say store R 1 back to x right. This is the set of three instructions that are required in order to increment a variable which is in the memory this actually points to some location maybe this is location 1002.

So, this will also be location 1002 which is nothing, but the address of x in the memory right. So, this is what thread 1 wants to do and what does thread 2 want to do? Thread 2 also wants to do the same thing. So, it wants to say load into some register may be let say R 1 x add R 1 1 store R 1 back to x which pointing to the same address all right. So, how do these instructions get executed right? So, if you have let us say two processors each one of them has their own register set.

Now, what will happen? So, the first processor let us say that it loads the value of R 1 into its register right. So, what is the value that will come to processor one? So, whatever is stored over here. So, let us say that initially the value stored over here is 0 right. So, processor 1 will fetch 0 into its register and now let us say there it is executing add R 1 to one. So, this is this has nothing do with the memory right its just incremented the value of the local variable to one and then it stores R 1 back to x. So, it overwrites this 0 with one and then thread two comes along it executes the instruction load R 1 x.

So, what will happen here? This value one will be fetched over here in processor 2, one will be added to it executes add R 1 1. So, this will become 2 and then store R 1 x. So, it is going to store two back to the memory location right everything is fine. So, let us say I was trying to count the number of occurrences of let us say a word in a document and I split the document between processor 1 and processor 2. So, each one of them were counting their own words and they were updating right. So, this is let us say the number of times that the word exams appeared, right. So, processor 1 sorry; the word exams one.

So, it try to update the variable x which is the location which is keeping track of the number of times that the word exams appears right one and processor 2 also for exam ones and it also try to update the same variable and eventually you got 2 and there were two occurrences you got 2 in the final variable. So, everything seems fine right if any of the processors now tries to print the value of x right you will get the value 2, which is what your program intended to do, right.

But now what could have actually happened is something different right whereas, another alternate thing that could have happened this is your variable x. So, let us say that first thread 1 loads R 1 into x. So, what will go to thread one the value 0, let us say 0 was stored over here first right and it executes add R 1 1. So, it updates this, this to one. Now before it can execute store R 1 x let us say that processor 2 invoke load R 1 x right.

So, what will happen? It will fetch the data from this location and that data happens to be 0 and then it executes add R 1 1. So, it updates this to one.

And now let us say that processor 1 executes store R 1 x. So, this comes back and it writes one over here and now processor 2 to execute store R 1 x. So, it is going to override this one with one and now the output is incorrect that is not the output that you wanted right. So, the correctness of your program is dependent on how the threads are executing the instructions right in which order the executed in the instructions or how did the memory see those instructions being executed, right. If the memory receives this set of instructions this load and this store first and then it sees this load and this store you will get what you desired right.

But if it sees this load followed by this load and then maybe one of the two stores, then the answer you get is going to be incorrect. So, this is a race condition and race conditions are very very common in when you are working with multiple processors when you are working with parallel programming or even if you are working on a single processor with multiple threads right because the threads can get scheduled because of time slicing or because they were perform some blocking operation and they get scheduled out right we discussed this last time.