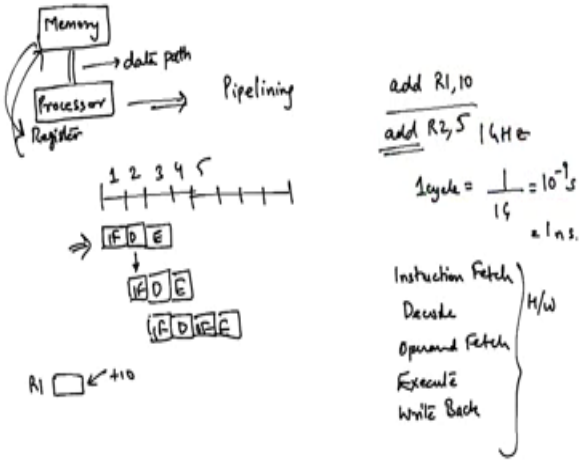
Introduction to Parallel Programming in OpenMp
Dr. Yogish Sabharwal
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture - 02
Parallelism in Single Processor Architectures

So, before we get into parallel processing architectures right we need to understand what is the architecture of a single processor, that will help us determine what is the performance of our code on a single processor architecture, it will help us to identify what are the bottlenecks right is it bound by the processing power, is it bound by the memory, and then we determine that if I put more processors; however, going to overcome those bottlenecks right what algorithm do I need to design to overcome those bottlenecks.

And also understanding the single processor architecture will help us you know figure out that what is the kind of speed ups, I am going to get if I put more processors right. If I put 10 processors then I am going to get a 10 factors speed up or not right. So, those things to understand those things you need to understand what the single processor architecture is.

(Refer Slide Time: 01:24)



So, let us look at what a single processor comprises of. So, in a single processor we have a professor, we have a memory unit and we have a data path. Now any of these could be the bottleneck for the algorithms that you have which you want to run faster. So, let us start with the processor, what are the features that the architecture of modern day processors have right which helps them to improve performance.

So, the very first feature that you should be aware of is pipelining right. So, what is pipelining? So, what happens is that the instructions that you give right if you say add 10 to some register R 1 right how is this actually executed on the processor? So, actually what happens is that each instruction is broken down into stages and the processor now executes these stages, and possibly it will execute one stage in one cycle right? So, what is the cycle? So, let us say that you have a one gigahertz processor, what is that mean? That means that the clock frequency is 1 gigahertz. So, one time period of the clock we call it one cycle is going to be 1 by 1 gig right which is 10 to the power minus 9 seconds right this is one nanosecond ok.

So, one cycle of the processor one clock tick or one cycle as we call it is going to be of length one nanoseconds. So, let us just draw timeline of the cycles as we go by right. So, this is clock cycle 1, clock cycle 2, 3, 4, 5 and so on. So, now, what happens is that your instruction may take some time to execute right it may take 10 nanoseconds or you know 15 nanoseconds or if you have a division is an expensive operation it might take you know if the order of 50 70 nanoseconds.

So, what is done is that typically these instructions are divided into stages right and what are these stages? So, there is stage like instruction fetch decode right. So, instruction fetches you fetch the instruction from the instruction cache, decode as you decode the instruction you determine what that instruction is, and then you may have to fetch some data from the memory right. So, let us call it operand fetch and after that you may have some operation to execute. So, you execute the operation, and let us say you want to write back the results to memory right this is just an example of 5 stage pipeline right.

So, modern day architectures you will find you know have 20 or more stages its quite common. So, in pipelining what happens is that you have this instructions say add 10 to register R 1 right and how its executed is that, its divided into stages and one of these stages is executed in each cycle right. So, we will we will take a simple scenario where

one stage is executed in each cycle that may not be necessary, but let us go with that assumption right. So, in the first cycle what will happen is that you will fetch this instruction, in the second cycle you will decode in the third cycle. So, this does not require any memory operation, this memory is not involved over here 10 is a constant and R 1 is a register right.

So, you do not need an operation fetch. So, you can directly execute the instruction right and there is no write back also its not writing anything back to the memory right. So, what does this do what does add R 1 10 do? It basically just adds 10 to (Refer Time: 05:30) register right the register over here is R 1. So, what are resistors? So, the processor right it has a registers set with it, and all the operations all the mathematical operations are basically done using the registers right. So, if you want to let us say add 10 to some location in the memory, you cannot directly add that number to the memory location the processor has to get that data get it into the register, add 10 to it right that operation can be performed only when the data is in the register, and then write back the result to the memory ok.

So, this is essentially what pipelining does it splits the instruction into stages and in every cycle we have one stage that is getting executed. And now the point is that there is different hardware that handles each of these stages like the instruction fetch the hardware is different, for decoding the instructions there the different set of hardware and so on.

So, what does that allow us to do? So, now, let us say I have another instruction to execute after this. So, after add R 1 10, I want to let us say add R 2 5 right I want to add 52 another register R 2 the point is that in the second cycle now. The first instruction is executing the decode stage it is in the decode stage right. So, the instruction fetch logic is not being utilized. So, at this point of time you can utilize the instruction fetch logic to do the instruction fetch for the second instruction right.

So, instruction fetch for the second instruction add R 2 5 is being done over here, in cycle number 3 you do the decode and in cycle number 4 you do the execute. In the third cycle you would be doing instruction fetch for the next instruction and so on maybe you have an operand fetch execute and so on right.

So, even though one instruction takes something like 10 nanoseconds or 5 nanoseconds or maybe 15 nanoseconds to execute right, but what you are able to do is you are able to pipeline them and the throughput that you are getting right you could literally get a throughput of one instruction being finished being completed every nanosecond every cycle.

This is used in real life also when you are let us say trying to assemble cars right, there are 10 different parts to be assembled there is there is a left door or right door and so on right and typically what happens is that you do not if you want to assemble 10 parts let us say in different parts right it and let us say each part takes about 10 minutes to assemble, then how long is it going to take to assemble the car? It is going to take 100 minutes right.

But that means, that after every 100 minutes you are going to get one car that is completely assembled right, but that is not the way it happens, you have an assembly line and what happens on the assembly line is that while the right door of the first car is being assembled, right the next car is already a step ahead and the left door of the second car is being assembled and so on, right. So, throughput that you get is that every 10 minutes you have one car finishing its assembly, right.

So, one of the things that this requires is hardware, you need separate hardware for each of the stages of the instruction that you have.