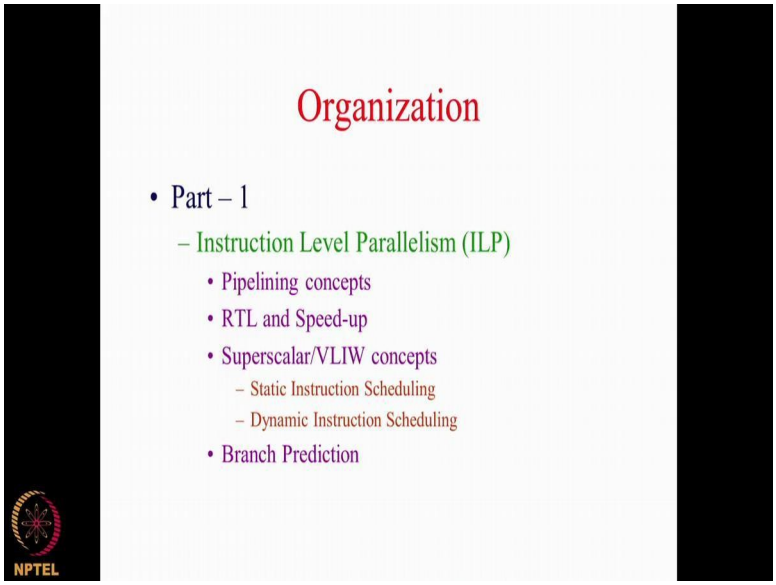
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 17
Part 1
Introduction to Pipelining

We will go very slowly understanding of this is extremely important because the machine set you are going to work the CPU that you are going to handle are basically pipelined and multi core CPU's fine.

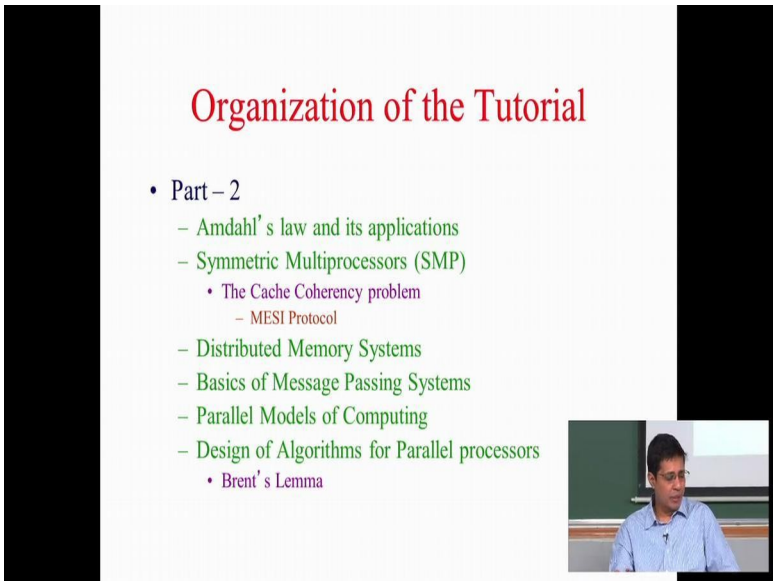
(Refer Slide Time: 00:36)



So, now let us start. So, this whole set of lecture I am organized into 2 parts, the first part is instruction level parallelism where we will be talking about pipelining concepts, RTL the register transfer level and how do you get a speed up there.

We will talk about super scalar we will talk about very large instruction word or very large inter word VLIW concept we look at basically static instruction scheduling and dynamic instruction scheduling.

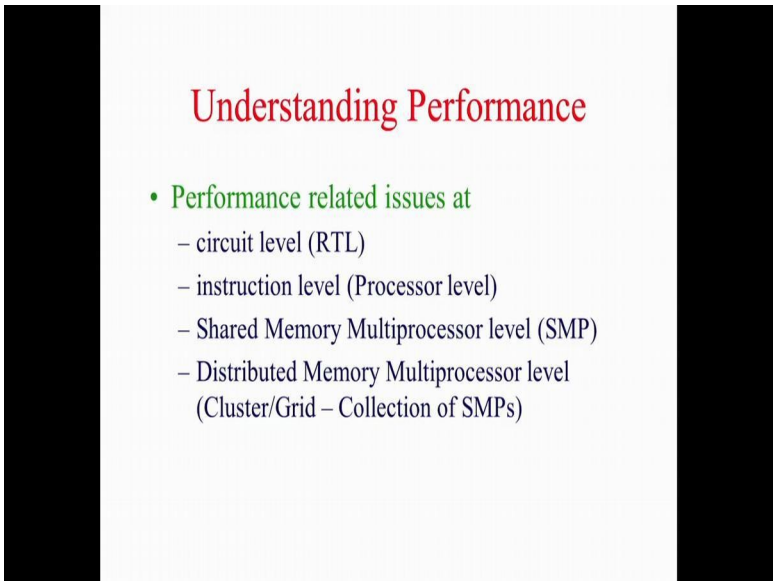
(Refer Slide Time: 01:10)



Then we will do some amount of astrology called branch prediction we will do that; then the next part 2 we will on to the multi core where we will have Amdahl's law applications, symmetric multi processors, what is a challenge in building a multi core the biggest problem that we will see something called cache coherency problem, I did mention in the third semester course what is cache coherency remember no I will repeat; then distributed mem we will look at distributed memory systems, message passing systems parallel model of computation, how do you design algorithms for parallel process some amount of coding. So, it is not enough I said this is a very big dam now I should tell you how to pour good water into it. So, I will also teach you some amount of parallel programming, this be an introduction to parallel programming ok.

So, how. So, you all done some sequential programming already as a part of your DSA lab correct. Now we will tell you how to do this parallel programming and there is some open m p and other software available in our super cluster. So, these 2 parts we will cover very slowly, but in great depth ok.

(Refer Slide Time: 02:17)



So, when we look at performance right I cannot get performance just of a blue, I cannot say I will write an excellent compiler and that will give me the best performance, I cannot say I have a best hardware that will give me a best performance right, I cannot say I will have the best micro architecture that will give me the best performance. If I need performance all of than have to work together right if I need performance here is one timing another is power another is area all these things if I that is what I already I told you right.

If I could make a machine that needs one kilo watt of power for execution nobody will buy it correct if I make a CPU chip which need one kilo watt to you know execute nobody we will buy it, we need to make a system that has good power dissipation that is that can be put in to small form factors. So, power area and performance together makes something saleable today and so, if I want to make hardware I need to make a hardware that is saleable. What is saleable there? My power area and performance need to be there. So, suddenly I cannot say you know performance I have a excellent compiler will invest so much on compiler I will get the performance it is all crap we cannot do it. So, performance is an joint orchestration of several components in the

system which will finally, give you the best results.

So, when I am looking at performance I should start looking at the circuit level, which we call as the register transfer level RTL right. In the next thing is now we will go and see if there is performance at an instruction level these 2 we have already have a glimpse right we designed a you know wallace tree multiplier, we designed a carry look ahead adder and there we did show that between a carry ripple adder and a carry look ahead adder there is a performance difference from n to log n, but we did increase the size of these circuits.

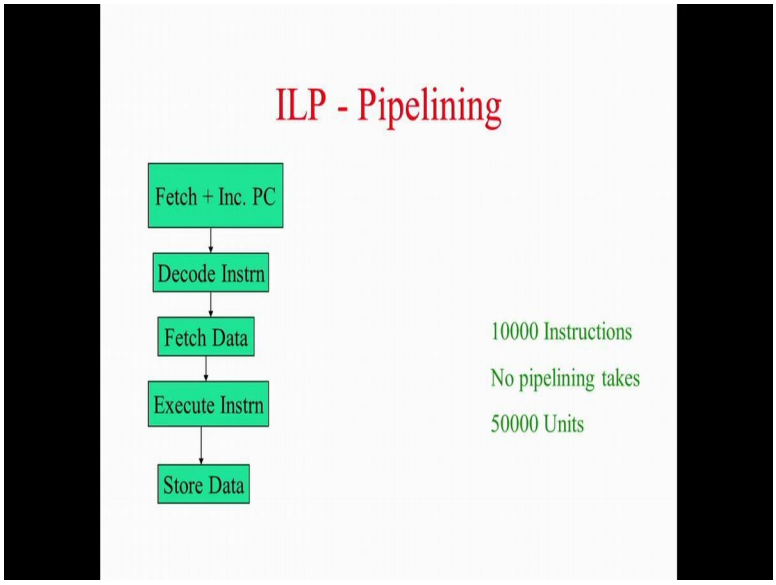
So, with some compromise in the area and the power dissipation, I could get good performance even functionally right. So, we good look at the circuit level at an instruction level we are looked at a s a m d for example, when we looked at the m m z instructions, we did the single instruction multiple data type of instructions we have seen some amount of parallelism there, but this is with respect to one instruction right. Now can I make multiple instructions run in parallel of course, we are going to talk about that in great detail then once I have one processor which has some parallelism inside it right. So, I. So, let me just summarise here I have improved my circuit to give better performance like I moved from a carry ripple adder to a carry look ahead adder; I moved from a sequential multiplier to a wallace tree multiplier now these things are going to me lot of speed. On a processor which has all these features I am bringing in things like s a m d type of instructions etc and there able to parallelism.

So, I have parallelism at the circuit level, I have parallelism at the processor level. Now I put multiple such processors which share a memory which is called shared memory multi processor right then it actually becomes now a multi core system with many many CPU's and which sharing a memory. So, I bring brought in more parallelism there by putting more process. Now one chip could have multiple such you know CPU's and one shared memory. So, this is called a multi core chip. I will put several multi core chips together that actually and each will have it is own memory so that becomes a distributed memory multiple processor which is the next stage of parallelism. So, parallelism can be at the circuit level, it can be at the instruction level, it can be at the chip level it can across chip levels right and so, today when we look at some of the modern super computers that we see that are executing.

The speed there the performance there essentially comes because it is not that they have hundreds of CPU's not just only because of they have hundreds of CPU's each CPU has multiple processing units multi core, and each of that core has several instructions that could do lot of concurrent activity, and those instructions are executed on units which are also highly on circuits realised using circuits that are also easily concurrent. So, when I look at a system when I am looking at performance, if is not enough just I have a super computer with multiple CPU's each CPU could have multiple cores, each core would have multiple instructions which are concurrent and each of these instructions could be realised using hardware that is very fast.

Right. So, we are looking at performance on all these levels, now what we have seen so far is that we have looked at RTL's circuit level, and we have also seen some instructions that can exploit hardware which can give a some amount of parallelism at instruction level. Now we will see more about instruction level then we will move to a SMP then to distributed memory as we proceed on this lecture.

(Refer Slide Time: 08:05)



Now what is pipelining I have done in the class earlier but then I will just for the benefit

of pala gart people I will do it once more here, pipelining is what we term as instruction level parallelism ILP. It is not integer linear programming it is instruction level

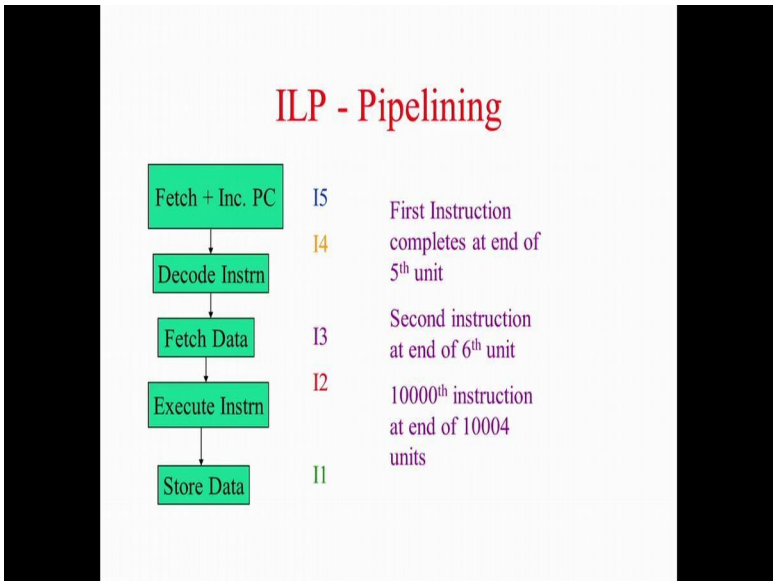
parallelism. Now where is instruction level parallelism let us look at where do we get instruction level parallelism; when I want to execute a instruction I fetch the instruction and increment the p c, the next is I decode the instruction the next is I fetch data, then I execute the instruction then I store back the result these are the 5 general stages in execution of instruction.

Now, what will happen is if I have 10,000 instructions to be executed and each of this unit take one unit of time then to execute one instruction it will take 5 units of time. So, hence to execute 10,000 instructions it will take 50,000 units of time. Now what we see is that when an instruction is fetched these 4 units are not doing anything, then an instruction is decoded the remaining 4 units are not doing anything.

So, when an instruction is in some stage I let us say stage 1 stage 2 stage 3 stage 4 stage 5 if it is some stage I all the remaining stages not j not equal to I remain silent they are not doing anything correct so; that means, in the in a single cycle in a in one cycle the instruction finishes of all the 5 steps in that only at any point of time only one fifth of that hardware is working the remaining 4 fifth is sleeping right.

So, can I do something about it? So, there comes the notion of a pipelining what is pipelining you have already seen some auto mobile or any manufacturing sector there is something called a pipeline for example, of I want to make a scooter, somebody will assemble your you know seat and then your body, and then somebody will be fitting the tier while they are fitting tier for this first scooter the second scooter body will be assembled. I am and they are putting the break for the first scooter second scooters tier will be there third fellows body will be assembled and so on So, this is called a pipeline right.

(Refer Slide Time: 10:35)



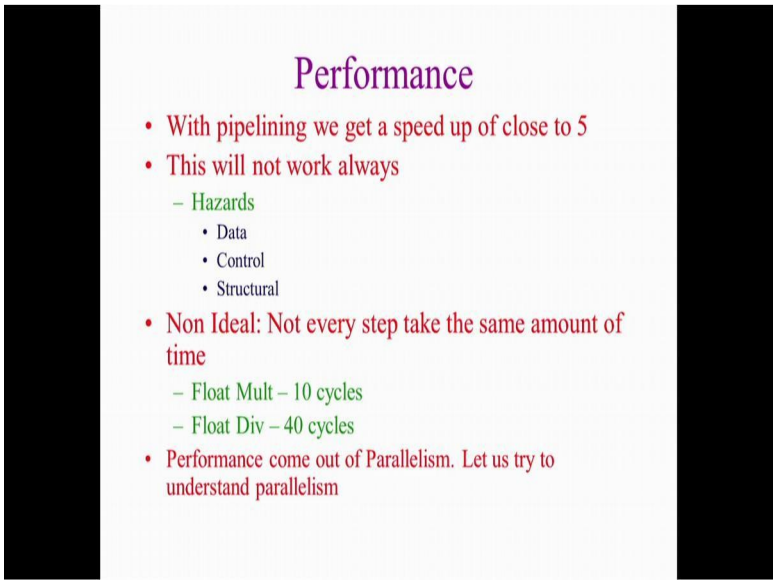
So, similarly the same concept is seen here with pipelining say I 1 is fetched, when I 1 is decoded I 2 is fetched when I 1's data is fetched I 2 is decoded I 3 is fetched when I 1 is executed I 2's data is fetched I 3's instruction I 3 is decoded and I 4 is fetched when I 1's results completes execution and results are showed I 2 is getting executed I three's data is being fetched I 4 is decoded and I 5 is fetched. So, the first instruction completes at the end of 50 unit of time right, but the second instruction when it will complete at the end of sixth unit third instruction at the end of seventh unit.

So, at 10000th instruction we will finish at the end of 10004 units. So, by doing this by re utilizing the hardware what we have achieved we have got 75 percent close to 75 percent performance improvement. What was taking something like you know 50,000 we have got close to 80 percent improvement. What was taking 50,000 units of time currently it finishes off in 10004 units of time. The first instruction take 5 units of time the reason that is called latency pipelining latency please note this word. So, this is called pipelining latency latency of the pipeline. So, 5 first instruction does not come in the first unit it is coming in the fifth unit, but their subsequent instruction comes one after another right. So, I get this 80 percent speed up just like this by re utilising the hardware, but this is an very ideal case there can be

lot of scenarios where this cannot happen as nice as this.

If everything could happen as nice as this then we do not have anything to teach, but lot of lot of people lot of problems come up when we try to do this instruction level parallelism, but. So, with pipelining we get actually a speed of close to 5, but then this will not work for several reasons the reasons are called hazards.

(Refer Slide Time: 12:54)

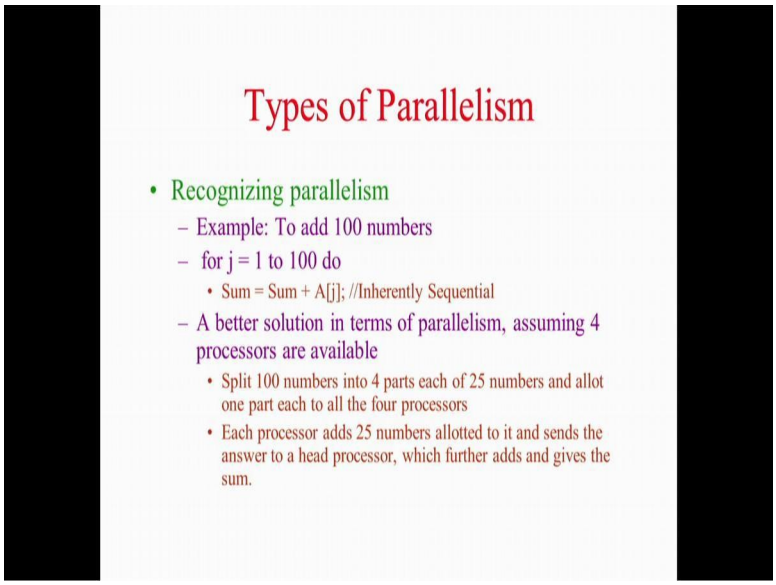


Hazards is one that will stop the pipeline simple and there are many types of hazards one is called data hazard, another is called control hazard, another is called structural hazard there are three types of hazards data control and structural. This is one very important thing. So, you will not have that free flow of instruction one after another, there may be reasons why I cannot proceed pumping that instruction from one stage to another as have seen in the previous slide. It may not be possible for I 1 I 2 to come in this same speed somewhere I have to stop them and the reason for stopping is called hazard. The second important thing is I did assume that every unit takes unit amount of time every unit here takes unit amount of time this is not the case especially for the execution for example, floating point multiplication will take 10 cycles which is very close to an adder, but floating point division we will take 40 cycles. So, me assuming that everything is unit coast right one unit of time

everything can be finished in one unit of time is a wrong assumption.

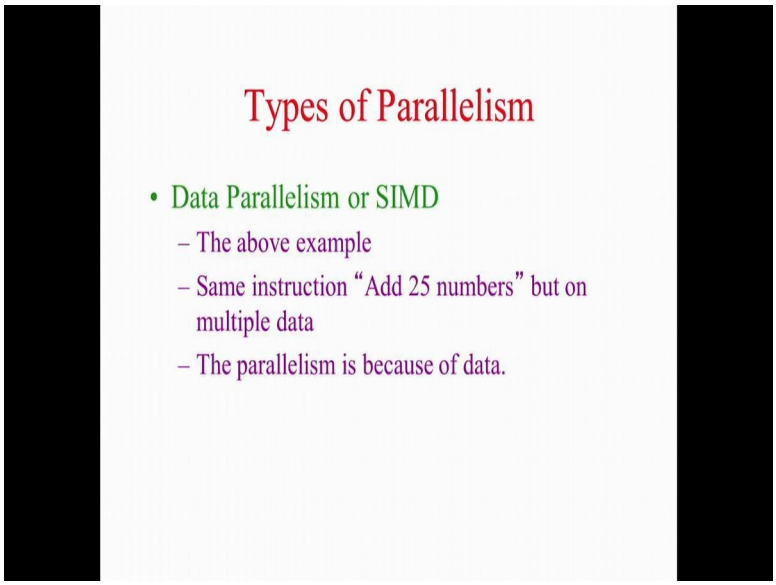
They execute for different instruction for I 1 can be a floating point multiplication you can finish off in 10 units of time, I 2 is a floating point division which can take 40 units of time. So, I assume that every every fellow finishes in one unit of time what is that unit it can be different for different instructions.

(Refer Slide Time: 14:35)



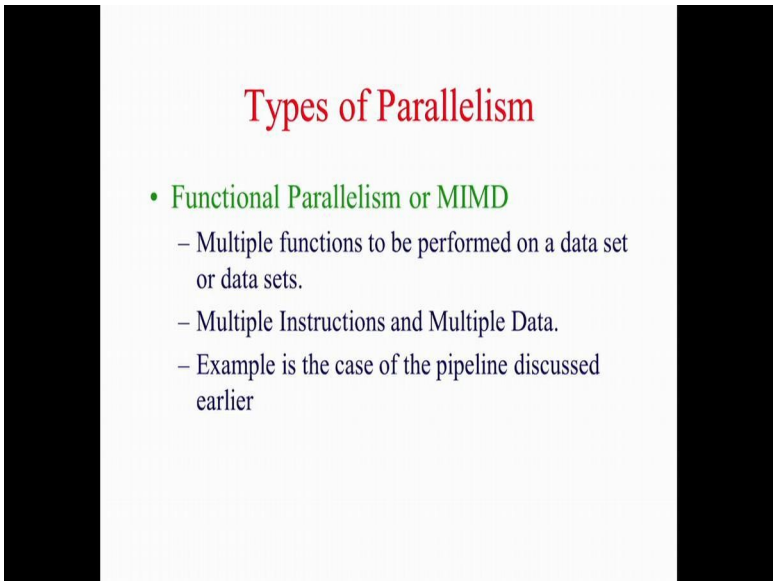
So, what is the reason here the performance actually comes because of some am some amount of inherent parallelism that is existing among the instructions and we will try and understand what are those parallelism. We have already seen this type of parallelism correct this is called s a m d suppose I want to add 100 numbers and I have 4 processors, I go and say split hundred one thing is j equal to 1 to 100 do sum equal to sum plus a j which is a very very inherently sequential process, but if I have 4 processes the best solution is split 100 numbers in to 4 parts each of 25 numbers each allot one part to one of processor and ask all the processor concurrently to add the parts assigned to them. So, in 25 units of time I will get the answer of 25 numbers being added, and then I will have one processor which will add those 4 and give me a answer right.

(Refer Slide Time: 15:18)



So, at the first stage what we are doing is that we are doing this is called data parallelism because the parallelism here is because I have 100 units of data, the parallelism in the example here is because I have multiple data right. So, I had 4 different sets of 25 numbers multiple sets of data please understand; and to each of these 4 fellows I said add those 25 numbers, I gave one single instruction and that is why this parallelism is called single instruction multiple data parallelism and that s a m d is possible only if I have multiple data. If I want to operate on the same thing on multiple sets of data then there is a scope for doing the single instruction multiple data concept.

(Refer Slide Time: 16:09)



On the other hand there is something called we talk about data parallelism there is something called functional parallelism or multiple instruction multiple data. What do you mean by this multiple functions to be performed on a data set or data. So, my data passes through different functional units and each one of them will do something different on the data for example, the pipeline itself if I assume that the instructions are data, and the pipeline units are processing that that itself is an example of a multiple instruction multiple data.

As you see this stored data is processing I 1 execute instructions processing I 2 there is no connection between them. So, it is a multiple instruction multiple data type of parallelism. To sum up there are 2 types of parallelism data parallelism and functional parallelism you have already seen data parallelism with some amount of thing like we have saw recursive doubling etcetera s a m d we saw the m m x instruction set etcetera now we will have a an very good instruction to functional parallelism which is multiple instruction multiple data.