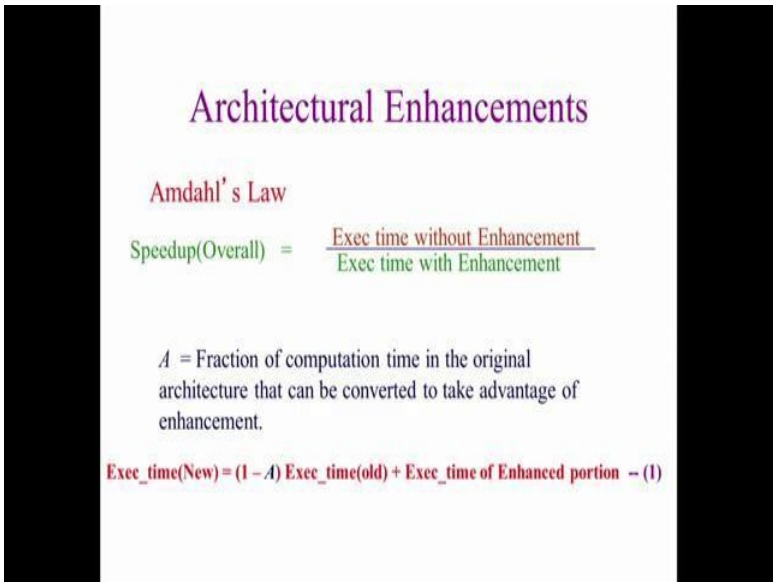
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 37
Part – 01
Shared Memory Architecture

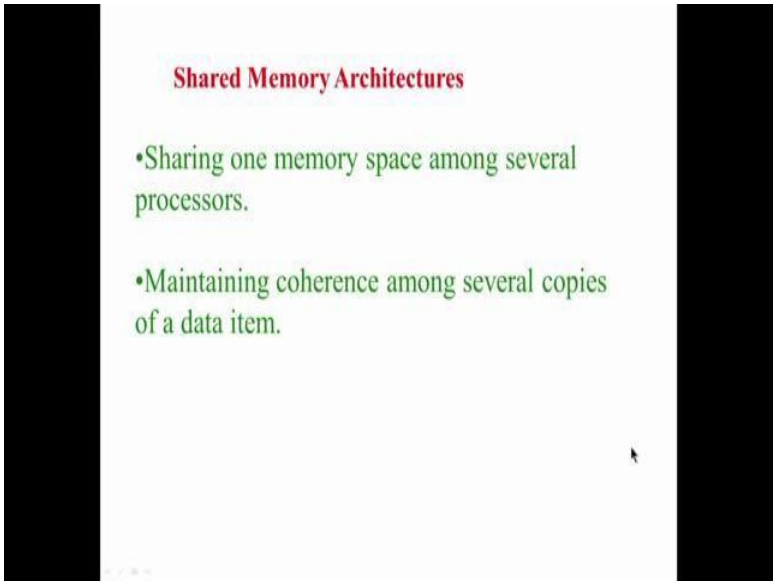
So, let us start with a positive note. So, I hope yesterday you went through that cache lecture by Prof. Manik Choudhary though the volume was low, but at least the slides were there. See that is the type of research that goes into server class architecture right what is the difference between a Xeon server or an Intel server or an IBM server and the mobile phone that you are using or the laptop or the desktop that you are using one of the major difference is there is in the cache. So, it was just a opening for you know, I am not going to ask questions from that of course, is just a opening for you that. So, much of effort goes into making a server and that effort 90 percent of that effort which distinguishes a you know chip from a server class chip versus desktop class chip is basically on the cache hierarchy. So, when you take a course on computer architecture you will be you know educated more on that stuff.

(Refer Slide Time: 01:23)



So, with this, we will now go into the last part of our course we finished Amdahl’s law.

(Refer Slide Time: 01:36)



So, we will go into the last part of this cache this discussion on cache and that is what we call as the shared memory architecture. You will spent quite a bit of time here because I want to teach you some amount of parallel programming I also want to cover some amount of issues in building multi core process. And as I told you, you will not find a single core chip anywhere. If you want to buy an 83866 single core, it will cost you 5000 dollars because last few pieces museum antique value, it is a like a furniture some 200 years old in your house. So, now, we have to now see what it means to bring up a multi core architecture and also we needs to see how to program it is there some way by which you could program fine. None of the many of the university curriculums in my opinion none has matured enough to actually educate you on from a background of a multi core processors, still we are sticking to the old you know single core or algorithms or single again a sequential algorithms that we teach. So, there needs to be a so the technology is much far ahead of what the actual software could be you know may today or how the mind actually a programmers mind work today.

So, one important thing that we need to understand here is there is no more sequential programming. If you are going to stick to your sequential programming, we are going to be in a bigger soup. What is that soup? Today, I actually suggested this paper long back I think even the third semester end of conventional end of road for conventional micro

processors, this is by a group at Texas at (Refer Time: 03:19) University (Refer Time: 03:23). So, they wrote this paper where in it says that beyond some 3.5 giga hertz

frequency your frequency will not scale; you would not go to a 4 giga hertz Intel processor I think we do not we did not see an a m d or an Intel processors with 4 giga hertz speed. The reason was that as your frequency increases there was a huge amount of power search that was happening and that is why we stopped; and then what they did was that they gave multi course which needs a giga hertz.

Now, today I have 4 core each running at 2 giga hertz. So, essentially I could get 8 giga hertz provided I know how to program it. If I run a program which is which was fine tuned for a 3.5 giga hertz processor, today the same program I run in this machine it will be much slower. Are you able to follow that? It will be much slower than the you know what we got as a performance in 2004 or 2006. So, today when I want to exploit the computing essentially I mean that means that I need to know how to use these 4 into 2 giga hertz - 8 giga hertz how should I get that performance I should know parallel computing and that is precisely why we are trying to look at this.

Now, what is parallel computing after all. I have a job you have already seen this suppose I have the hundred numbers to add and I have four processors split it into 25 junks of 25, make each processor add that 25 and then interact with each other to find out the final answer. So, there is some computation that is going to go on in your process individual we will call it as a core now individual cores. And then these cores have to talk to each other, share that information to come out with a final answer. And the sharing of information suppose we four are four cores the way by which I exchange information between you and between us is through the memory and that is why this is called as shared memory architecture. So, I share the memory where the memory is shared with among four of us and whenever I want to communicate something from A to B or when p 1 wants to communicate to p 2, it uses the memory. So, p 1 writes to location thousand p 2 reach from thousand. So, this is basically the shared memory architecture.

So, when I want to start doing parallel programming, this is one fundamental thing that we need to agree upon that we need to share information between the process some points of time and the way we share it to is through memory so that gave rise to a computation model. So, we have already seen an algebraic model of computation based use your order n squared, order n squared means what order n square n squared operations, operations who dictates that operation there is a model of computation that

dictates the operation. I have covered that in many times I have mentioned about this right even in this and data structures I will throw out of taught you do this things.

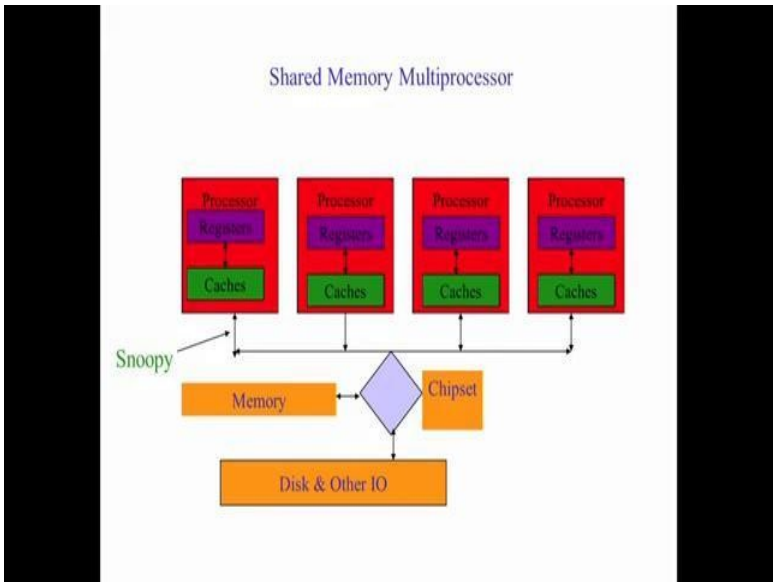
So, now like how I had a sequential model of computation, I will now have a parallel model of computation. So, I think whatever I have told now is just basically summarising this slide. Let me just read it out. Sharing one memory space among several processors is what we mean by a shared memory architecture. And the most important thing is that I need to maintain coherence amongst several copies of a data item. Now, I am going to have a cache right there is one shared memory and each of these processors are going to have a cache correct. So, when I read thousand that thousand will belong to me. So, there is a version of thousand in the main memory and that is also a version of thousand in my local cache. And for performance reason, I will not have a write through cache I will have a write back cache you understand the difference.

So, when I have a write back cache; that means, I am going to go ahead go ahead do a lazy write back later. So, the value of thousand that I will have in the processor core in the cache could be different from the value of thousand value that will be stored in the actual memory location thousand and that gives us the you know cache coherency problem. So, when we look at shared memory architecture, there are two things which need you to know how are we constructing this shared memory architecture. We know how to construct a super scalar out of order processor know, how can each of these super scale processors called a core. How of these cores, so four or five of them can be put into the same chip and made to you know made to execute a parallel programming right so; that means, you have to facilitate some inter core communication and that communication will be through the shared memory.

Once you establish that model, now we need to have a parallel programming model and we need to see that it works properly. So, this is you understand this. So, now we will take two things here one we will look at shared memory how does how this coherency is going to be you know handled that is one part that is the challenge. Now, I put four cores each has this one cache over common memory I have a bus through which it can read and write. Now, what is the problem to solve coherency I will. So, what is that you need to understand at the second year of your curriculum with respect to understanding multi core, how cache coherency is solved if that is understood that is a big thing. And then the

second thing that you need to understand is how to program these machines both we will cover in some depth.

(Refer Slide Time: 09:25)



So, this is the whole stuff here summarising what I have done so far. So, what I talked so far now there are four processors processor 1, 2, 3 and 4 in red each as its own internal resistors and each has its own L 1 cache and all of them you know connect through a common bus and then there is a memory that is connecting it. It is connected to the memory right. So, processor one will talk to processor two by writing into this memory and this fellow will read from this memory. There is no direct communication between the cache, this fellow cannot read him to this fellows cache; he can only read his cache and his memory, he can only read his cache and memory. If I could start interchanging between cache as then it becomes a nightmare. So, we cannot build an architecture practically infeasible for us to do such a thing right.

So, with this let we have one extra word in this slide, snoopy, everybody likes I like to snoop. Now, why it is called snoopy can you just guess I will give two couple of minutes. What is snoopy to do in this slide? See the point is when I am trying to read thousand write, now the some other processor could have the latest value of thousand. So, let me say we both are working. So, he has select thousand, he has written in thousand that

value of thousand the latest value will be in his cache not in the memory. Now, I want to read thousand I do not find thousand in my cache. So, I have to go up and

find it right to the memory. At that point he has to somehow come and tell me, hey do not go to the memory I am having the latest value of thousand, wait I will update it you go and read it from correct.

Let me again explain we both are working together. He has read the location thousand he has brought it into his cache and he has changed the value. Now, after that I am trying to read thousand. At the point of time, I do not know whether he has changed his value it is his internal affair right it is between his processor CPU and the cache I do not know what has happened to that thousand. Now, when I am trying to write or read from thousand somehow this fellow has to stop me saying hey, hey, wait, wait, wait I have the latest value of thousand. So, do not read stop then he has to update that value of thousand and then I have to go and read from it. Are you able to follow what I am trying say?

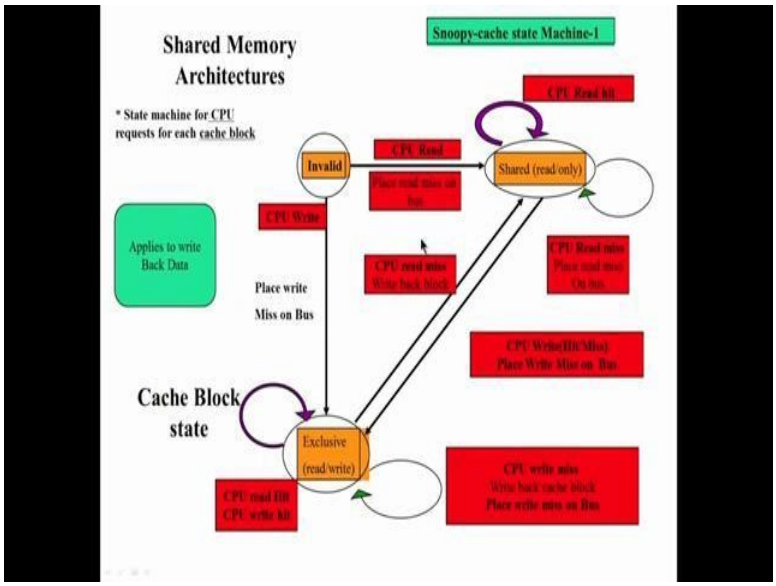
So that means, his hardware after modifying thousand has to keep snooping into the bus is there somebody else trying to read thousand. So, I have the latest value of thousand, after I get the latest value of thousand this his cache controller or whatever some part of the hardware should be snooping. Snooping means reading what is happening in the bus, what are other fellows are doing that what we call snooping right or the definition literally meaning of snooping. What he is doing right, I am not aware of what I was this is (Refer Time: 12:48) base right relative getting space, it is not just important how well I score how others scores, so that is what.

So, we keep snooping on this and that some point of time you say if you that fellow is going to read thousand and I have the latest value. So, I go and stop him. So, all protocols that enable you know cache coherency that solve the problem of cache coherency in some way or other does snoop. And snoop is the best is the way that we can basically handle you know cache coherency issues. It is a very practical way by which we can handle cache coherency. We could have some global ways of handling this, but this is much more localised and distributor and this is a very neat way.

So, snoopy based cache protocol is an area of research people have come out with several snoopy based cache protocol and that will be actually if you take a course on computer architecture you will know much more about this. But we will do one simple snoopy based cache protocol as a part of this course. Are you able to get what I am trying

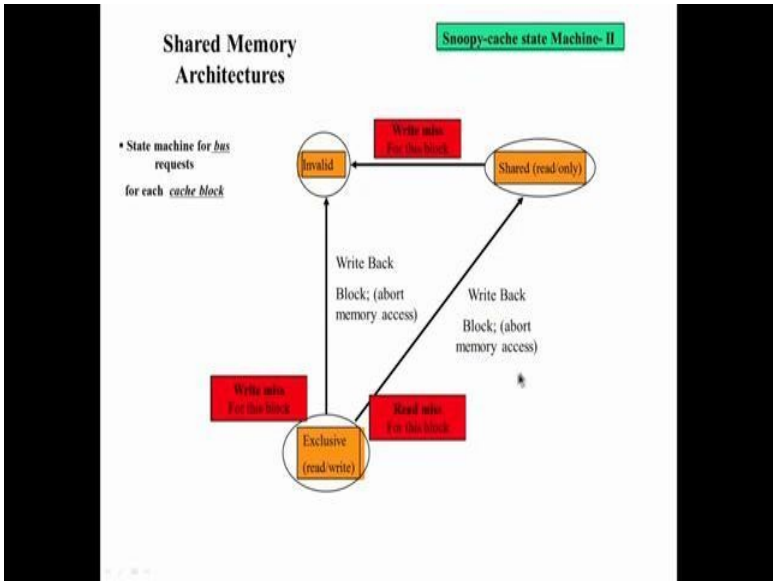
to say? If you do not follow this, please stop me here because it is going to become a little more intense as we go from the line.

(Refer Slide Time: 14:21)



So, this is the snoopy based protocol. So, this is what is actually given in many text books and with one, two or three paragraphs of explanation and obviously, you will not understand what it means and there are two state machines.

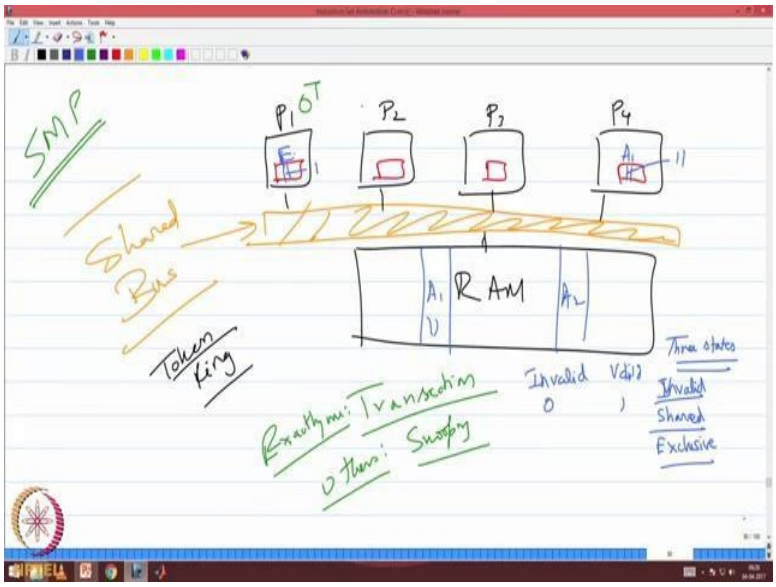
(Refer Slide Time: 14:39)



But you will understand that this is a whatever you see here and in the next slide are deterministic finite automation, it is a DFA. There are three states and many edges that

are going between these states ok now I will now go back. So, just remember this slide, this slide will be something like after you understand what I teach this is very nice for revision. So, these two slides are for good for revision, but not for understanding. I will teach these two things one today and probably yeah if possible today or today or tomorrow, I will teach. I will split it a pros two places.

(Refer Slide Time: 15:34)



Now, we will go back to the our same as this. Now, I have four. So, please do (Refer Time: 15:40). So, there are four processors and there is each has their own cache and there is a RAM. And they all share a common bus correct, they all share a common bus. Now, at any point of time, one of the processor will have control over the bus I will the bus will be allotted to exactly one of the processor so that will call that processor as a bus master. So, we will have some protocol like a what we will learn about this in distributed computing if you take a course, but for quick understanding we will have something called a token ring protocol.

It is nothing but a token a token that is circulating. So, the token first will be with p one when it has the token it becomes the master; after that it will give turn p two. For some time it will be there it will give to p 2, then it will give to p 3 then it will give to p 4, one by one it will this will go like a (Refer Time: 17:12), but it will be going around round and

round. So, this is called a token ring simple token ring protocol. So, at any point of time, one fellow exactly one fellow will be the bus master, the other fellow will be just

watching, watching or snooping. So, at any point of time one fellow will be doing the transaction, and other fellows will be doing exactly one will do the transactions the others will be doing what snooping, what is he trying to transact you got this, you see right. So, every point of time there will be exactly one fellow will be doing the transaction, others will be seeing what he is tries going to transact.

So, for every memory location in the RAM, there is one line cache line in which I can go and recite for every memory location there is one cache line in which I can go and recite in every cache and these are all symmetric multi process that is why we call it as SMPs. It is not shared memory multi processors or symmetric multi process. We also call it symmetric multi process. Symmetric means everyone is equally capable of everything right if I can do something he can also do that if I cannot do something he should he cannot also do. So, there is no asymmetry between each. So, this is a symmetric multi process everybody will have exactly the same size of cache.

So, if I have a location thousand say I have let me assume that there are two locations say thousand. So, let us forget this thousand. So, I have A 1, I may have another address A 2 just for the sake of understanding let A 1 and A 2 map onto the same location in the cache. Let A 1 and A 2 map onto the same location in the cache. Now, so for every cache line here right we had one state in the previous thing what was the state it is a invalid line or a valid line right then we had the t bit etcetera. So, we have a invalid line or a valid line now we will have that was invalid, valid. So, it was a one bit you know one bit thing. Now, we will have three states. So, every cache line cache line inside this will be it had two states earlier, now it will have three states.

What are those states, valid. So, first thing invalid right then we will have something called shared then there is something called exclusive. So, we have three states here one is invalid, another is shared another is exclusive. Now, what is invalid whatever entry is stored in this cache line is junk. What is shared, I am storing some address here, somebody else also is storing the same address in his cache. For example, A 1 maps onto some line if that line is shared that means, A 1 is mapping on some line that line shared means A 1 is stored in that location in the cache. There is a version of A 1 in the cache, there is also a version of A 1 in there can be also the version of A 1 in some other cache of some other processor. And if that is the case then the value stored here, let me say the value is 11, the value here should also be 11 and the value here should also be 11.

So, if a location is in shared state that means, for that address please note it very carefully, for that address I have a value which is the same as the value that is stored in the RAM and somebody else also could have the same value. Somebody could else also could have the same address in that cache and the value will be the same you understand this. So, if a particular cache line in a particular cache is in shared mode, again I repeat it has a value and that value is the same as what is there in the memory and some other processor could also potentially have. It is not necessarily should have could also potentially have the same address in its own cache with the same value. That means, my line is in a shared state are you getting this. So, it is are you are you able to follow what I am trying to say?

Now, what is the next thing there is something called exclusive. Exclusive means I alone have if suppose this is at exclusive state that means, I alone have that address stored in my cache alone and the value that is there need not necessarily be the value in the main memory, I alone have that address stored in my cache. So, if A 1 is mapped A 1 is stored in p 1s cache with exclusive means the copy of A 1 is exclusively in p 1 cache no other cache could have A 1. And the value that I have stored in that cache need not be necessarily be the value that is there in the memory and whatever I have stored is the latest value. So, this is what exclusive means.

To repeat every cache is symmetric, for every address there is a location in the cache where I can map it to. Now, coming to the caches every cache line can have be can have one of the three states, it can be invalid that means, whatever is stored there is junk. It can be in shared state that means the address that it is storing can be part of some other cache of some other processor also. And the value that is stored in that cache is the same as the value that is stored in the memory and which is same as the value that will be stored in the other processor also that goes by the definition.

If the cache line is in exclusive state then the address that in stored in that it is the only cache that is storing that address no other cache of any other processor could store that address and the value it stores need not necessarily be the value that is stored in the main memory. And this is the latest value of it. If somebody wants to read this address it has to take it from this cache. Are you getting this very clear right, should I need to repeat? Please understand that the state of a cache line, the state of a cache line can go between these three it can be invalid, it can become shared, it can become exclusive. And the state

of a cache line can be changed because of three things, what is it, because of a transaction that is initiated by the CPU.

I am a cache belonging to p 1, the CPU tries when it is the master it tries to read and write some memory location and because of that the status of I am a cache line because of that my status can change. And this will happen when p 1 is the master correct. Now, when p 1 is not the master something, so I will be snooping and something that is happening on the bus can come and change my status. So, the status of a cache line can be changed because of two things. Number one, when I am p 1, I am a master correct, I will be doing certain transactions to the memory load and store and because of that my cache line can change, my state of the cache line can change. The other thing is that when I am not the master I will be snooping on the bus; and because of some transaction over the bus, my state can change.

So, the my state change is governed by two state machines. One state machine, which is based on the transaction initiator by the processor to which I am associated to. And another change is because of the state machine which is associated to the transactions on the bus. The first state machine comes into effect when I am the when the processor to which I am associated to has the token. The second state machine comes into effect when I do not have the when the processor to which I am associated to does not have the token, so that is the two state machines that I showed in the slides. So, that is the two state machines that I showed in the slides fair enough. Are you able to follow?

So, what are those state machine for every cache let, for every status bit this state machine is going to run that the status between will be updated based on the state machine.