High Performance Computing

Prof. Matthew Jacob

Department of Computer Science & Automation

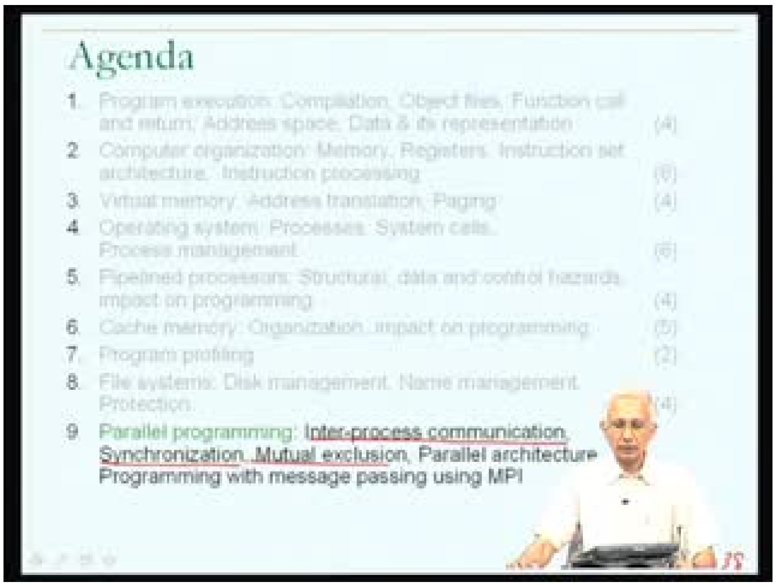
Indian Institute of Science, Bangalore

Module No. # 09

Lecture No. # 38

Welcome to lecture 38 of the course on High Performance Computing. In this lecture we will begin the last topic in our agenda, you remember that we had nine topics in our agenda, then last of which was parallel programming. In this topic, we will look a little bit at aspects of parallel architecture and then look at some of the issues related to parallel programming.

(Refer Slide Time: 00:37)

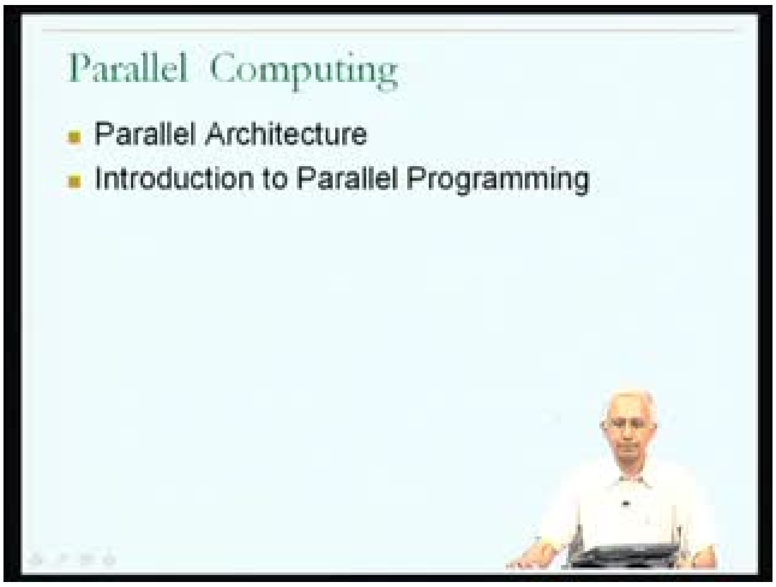


Agenda

1. Program execution: Compilation, Object files, Function call and return, Address space, Data & its representation (4)
2. Computer organization: Memory, Registers, Instruction set architecture, Instruction processing (6)
3. Virtual memory: Address translation, Paging (4)
4. Operating system: Processes, System calls, Process management (6)
5. Pipelined processors: Structural, data and control hazards, impact on programming (4)
6. Cache memory: Organization, impact on programming (5)
7. Program profiling (2)
8. File systems: Disk management, Name management, Protection (4)
9. Parallel programming: Inter-process communication, Synchronization, Mutual exclusion, Parallel architecture, Programming with message passing using MPI

I had covered some of the sub bullets, some of the sub topics, in this subject earlier, in particular synchronization, mutual exclusion, to some extent inter-process communication. You will remember, when we were talking about concurrent programming, there is some relationship between concurrent programming and parallel programming. But let us without further ado get into parallel computing.

(Refer Slide Time: 00:57)

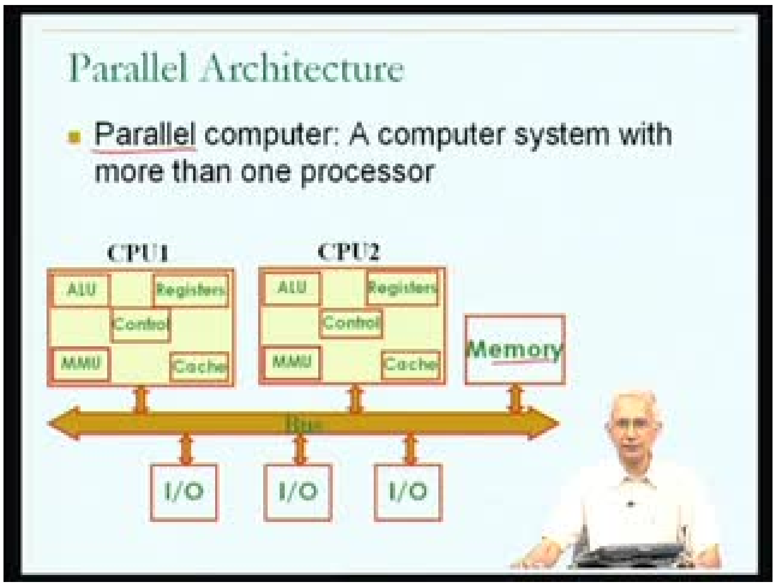


Parallel Computing

- Parallel Architecture
- Introduction to Parallel Programming

Now, we are going to not only look at some of the hardware issues in parallel computing, to understand what the underlying hardware is, but also spend some time, the bulk of the lecture- remaining lectures, in some topics in parallel programming.

(Refer Slide Time: 01:16)



Parallel Architecture

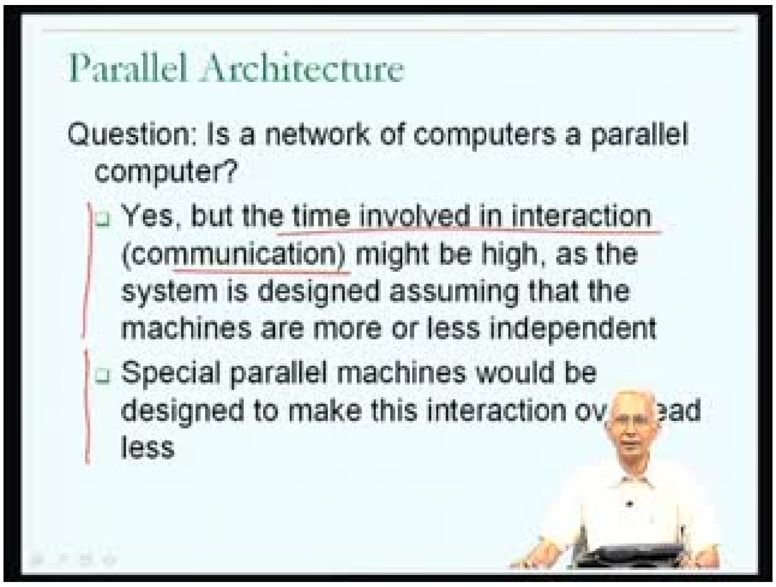
- Parallel computer: A computer system with more than one processor

Now to start off with, I will clearly state that the parallel computer is any computer system which has more than one processor. And this is important that we get this out into the open, because we have talked about ordinary computers where there was only one processor, there was the possibility of a lot of activity happening in parallel.

For example, it was possible for one instruction to be fetched, while another instruction was being decoded etcetera in a pipeline processor, and that was activity happening in parallel. But here we very clearly are talking about a situation where, we have identified parallel computer by the fact that the parallel computer has more than one processor, and therefore, in fact, more than one instruction can be executed at the same time, unlike what we were talking about up to now.

So, the computers that we were talking about up to now, were characterized by this kind of a block diagram. Memory- essential for storing the programs and the data, there could be multiple IO devices, and a single CPU which we now understand, in the case of a parallel computer, at the very least would have to have 2 CPUs. So, this would a diagram of a very simple parallel computer, in which there is more than one processors, specifically two processors.

(Refer Slide Time: 02:28)



Parallel Architecture

Question: Is a network of computers a parallel computer?

- Yes, but the time involved in interaction (communication) might be high, as the system is designed assuming that the machines are more or less independent
- Special parallel machines would be designed to make this interaction ov[illegible]ead less

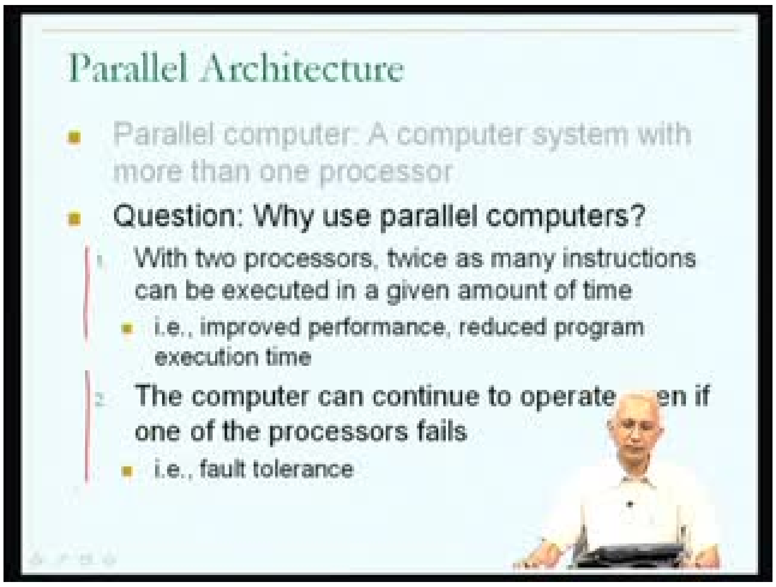
Now, the question which will arise in your mind is; if you think about the ordinary set up that you probably have in your college, where there is a network of computers, there once again it is, or even in your home where you might have a laptop and the desktop, multiple computers which are actually networked together, then the question which may arise is, could one view the network of computers as a parallel computer?

The answer is that, in some sense one could, because there is this is a situation where there are more than one computer, more than one processor, to be precise, in which it is possible to run programs under those programs could interact. So, the primary characteristic that we looking for in a parallel architecture is, they should be more than one processor, giving us the capability of writing programs that can run on more than one processor, in achieving the common objective of the program. Now, the problem with the network of computers would be then that, technically we could call it a parallel architecture, but the time involved in interaction or communication is likely to be more than it needs to be. Because in the case of a network of computers, the individual computers were designed to be independent of each other, and they happen to have been networked together using networking hardware.

But in truly parallel computers, which were designed specifically for running programs in a mode, where there is more than one process running each on a separate processor, it is likely that the machine itself would be designed to make the interaction overheads or the inter- process communication overheads as low as possible.

So, while one could think of a network of computers as a parallel computer, and one could, in fact, write parallel programs to run on a network of computers, there are likely to be, and many of you are aware of, special purpose parallel machines on which the parallel programs can run much more effectively. Because the parallel machines have been designed to make the inter process communication or the interaction between the different parts of the parallel program as low overhead as possible.

(Refer Slide Time: 04:30)



Parallel Architecture

- Parallel computer: A computer system with more than one processor
- Question: Why use parallel computers?
  1. With two processors, twice as many instructions can be executed in a given amount of time
     - i.e., improved performance, reduced program execution time
  2. The computer can continue to operate en if one of the processors fails
     - i.e., fault tolerance

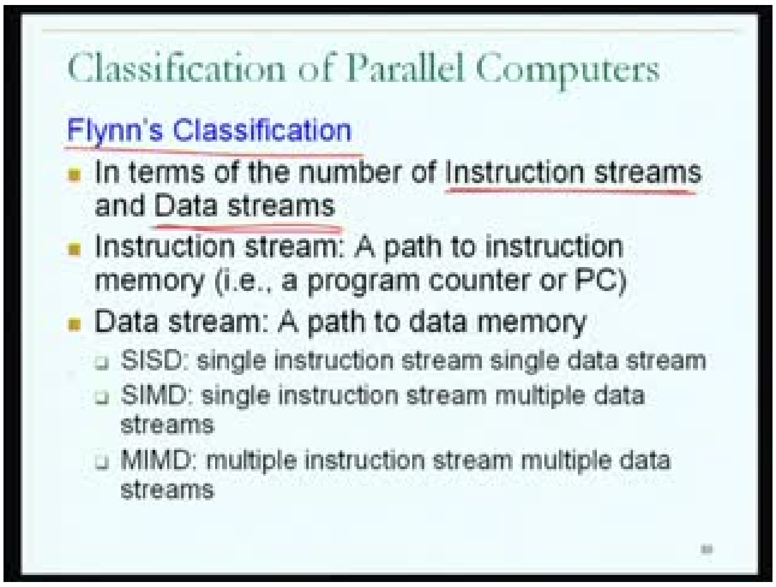
Now, another question which will arise in your mind is; why use parallel computers? Why are we studying about parallel computers? And I am going to just mention two reasons. The first is that; if you have at least two processors, and then if you write a program which runs on the parallel computer by using both of the processors, then your program should be able to run twice as many instructions in a given amount of time than if your program was running on a computer, which had only one processor.

There are two separate processors, each of which has the separate program counter, each of which can fetch and execute instructions. Therefore, in any one clock cycle, it is possible for two instructions to complete. In general, the more the processors, the more the number of instructions can be executed in parallel in a given amount of time. And therefore, the faster our programs will run. Of course, the program may have to be modified from the form in which it could run on a single processor. But, if the modified program is making use of all the processors of the parallel computer, then one should see improved performance. So, that is one reason for using parallel computers.

Another is that, if the parallel computer contains more than one processor, even if one of the processors fails, for some reason, one should be aware that hardware occasionally will fail, by fail I mean ceases to function or ceases to function correctly, then the fact as a matter is, the remaining processors in the parallel computer can continue to work, in fact, can continue to execute your program. And therefore, one could actually have a situation where, the parallel computer will allow on to write programs which can be fault tolerant, which will continue to run despite the fact that parts of the computer system may have failed. And this might be important, for example, in certain kinds of setting such as, you may be well aware that in banks they use computers to keep track of the various transactions and the information, vital information, and very clearly there it is a situation where, one would hope that there is fault tolerance. Even if one of the computers fails, even if one of the processors and one of the computers fails, they should be no loss of transactions or the information, that was being compute, that was being transacted at the time that the failure happened. So in general, there are two very good reasons for wanting to use parallel computers, and as a consequence there are many parallel computers today.

Many of you would have heard about clusters, for example, clusters of computers, which are potentially thousands containing thousands of processors, and capable of doing, of running parallel applications extremely fast, compare to the alternative of trying to run the same application on a single processor. So, one finds parallel computers wide spread in many different kinds of settings.

(Refer Slide Time: 07:08)



Classification of Parallel Computers

Flynn's Classification

- In terms of the number of Instruction streams and Data streams
- Instruction stream: A path to instruction memory (i.e., a program counter or PC)
- Data stream: A path to data memory
  - SISD: single instruction stream single data stream
  - SIMD: single instruction stream multiple data streams
  - MIMD: multiple instruction stream multiple data streams

Now, there are different kinds of parallel computers. Therefore, before we actually start talking about programming them, it could be good to get some idea about how they are classified. And one of the popular classifications of parallel computers, given that there are different kinds of parallel computers, is known as Flynn's classification.

Now, the way that Flynn's classification tries to distinguish between the different kinds of parallel computers, which is what a classification is meant to do is, by characterizing any one parallel computer, in terms of the number of instruction streams and the number of data streams, that are present on that parallel computer, when a program is in execution. And how do we understand an instruction stream and the data stream?

One could think of an instruction stream as being a path to instruction memory. For example, in a single processor computer, we know that the path to the instruction memory was because there was a program counter; because the single processor, kept track of which instruction it was executing at a time; and use the program counter, as the indication of which instruction to be fetch from memory next; and hence the program counter, acted as the mechanism to create a path to instruction memory.

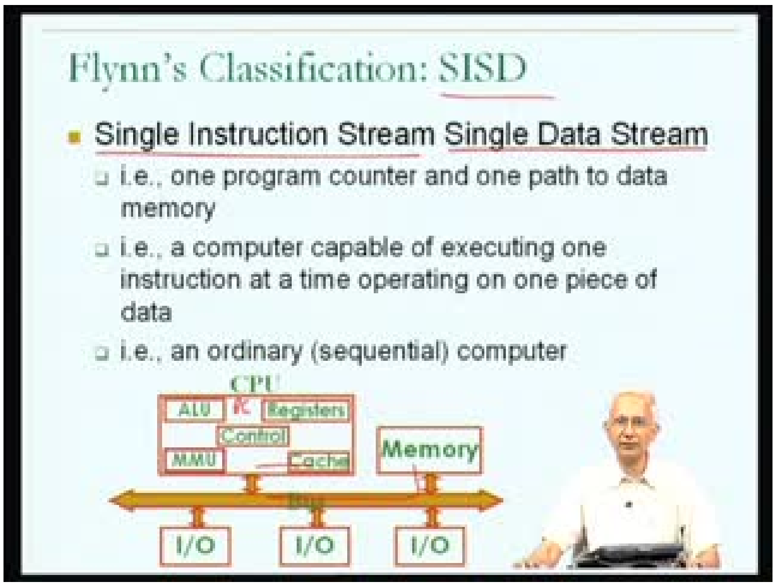
So, Flynn classifies parallel computers, first of all, in terms of the number of instructions streams. So, they could be a parallel computer with one instruction stream or a parallel computer with six instruction streams, and from that we understand that, it actually keeps track of six program counters. On the other hand, what is a data stream? Remember, Flynn's classification uses number of instruction streams and number of data streams.

A data stream can be viewed as a path to data memory, just like an instruction stream was a path to instruction memory, a mechanism through which instructions could be fetched, a data stream is the mechanism through which data can be fetch from memory, and conceivably a parallel computer which has multiple such paths, would have multiple data streams.

So, in short, with these two concepts Flynn proceeds to classify all possible parallel computers, but he does not use a complex notation, he merely talks about situations, where there is a single instruction stream, a single data stream, as oppose to a single instruction stream or multiple data streams, more than one, as oppose to multiple instruction streams and multiple data streams. To with some what concise classification, basically breaking the realm of parallel computers into three, based on the number of instruction streams being either one or many, and the number of data streams being either one or many.

Let us just look at each of these to get a better understanding, so that in effect, the objective of a classification is that, if we come across a new instance, a new parallel computer, by understanding the classification, we will be able to place the new computer into the classification, into our realm of understanding of the space of parallel computers.

(Refer Slide Time: 09:59)



Flynn's Classification: SISD

- Single Instruction Stream Single Data Stream
  - i.e., one program counter and one path to data memory
  - i.e., a computer capable of executing one instruction at a time operating on one piece of data
  - i.e., an ordinary (sequential) computer

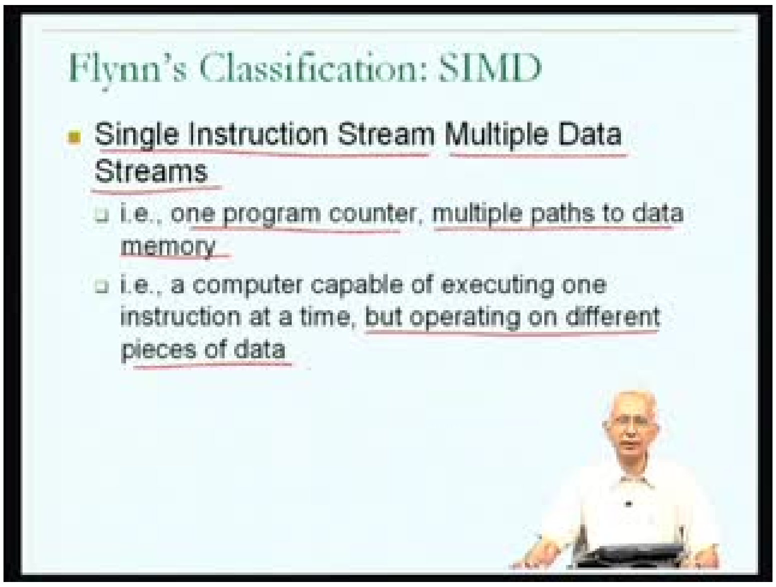
CPU
ALU
PC
Registers
Control
MMU
Cache
Memory
Bus
I/O
I/O
I/O

So, let us start by trying to understand what it means to be a single instruction, stream single data stream. So, such a computer would have; one program counter, that is what it means to have a single instruction stream; and it would have a single path to data memory. And we have already seen this, because that it is really what our the simple computer block diagram that we had up to now corresponded to, one of the special purpose registers was the one program counter in the computer, and that constituted the path to instructions or the instruction stream, and then there was the possibility of fetching through after address translation a piece of data from memory, and that went across a separate data path, which typically went through the data cache.

So, the situation that we have with the SISD kind of parallel computer in Flynn’s classification is that, this is not really a parallel computer, in the sense that, it has more than one processor, it is really the same old computer that we saw up to now, capable of executing one instruction at a time and the one instruction operates on one piece of data, and one could just think of this as the ordinary sequential computer.

So in effect, Flynn's classification includes SISD, just to include the ordinary computer that we have seen before, and in fact, Flynn's classification includes not only parallel, but also sequential computers by including SISD. So, this is not of interest to us in our current discussion about parallel architecture, that leads just two elements in Flynn's classification to discuss, SIMD and MIMD.

(Refer Slide Time: 11:27)



Flynn's Classification: SIMD

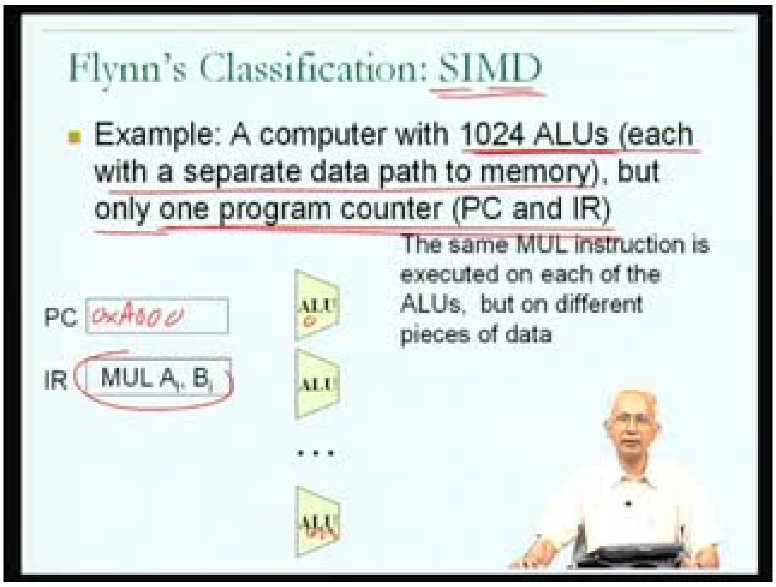
- Single Instruction Stream Multiple Data Streams
  - i.e., one program counter, multiple paths to data memory
  - i.e., a computer capable of executing one instruction at a time, but operating on different pieces of data

So, let us look at SIMD. So, an SIMD computer would be one that has the single instruction stream and more than one or multiple data streams, and what it means to have a single instruction stream is that, it has one program counter. But, multiple data streams means that, there are multiple paths to data memory.

Now, if there is only program counter, that basically means that the computer is capable of executing only one instruction at a time. But only one program counter, you can keep track of only one instruction which is being executed, and therefore, clearly only one instruction can be executed at a time.

But, the way the parallelism enters the picture is because, there are multiple data streams, multiple paths to memory. And what this means is that, the one instruction may be only a single instruction executed at a time, but at the same time, it is being executed on different pieces of data, and that is where the parallel part of the parallel architecture comes in. Let me just give you a simple possibility as far an SIMD computer is concerned.

(Refer Slide Time: 12:23)



Flynn's Classification: SIMD

- Example: A computer with 1024 ALUs (each with a separate data path to memory), but only one program counter (PC and IR)

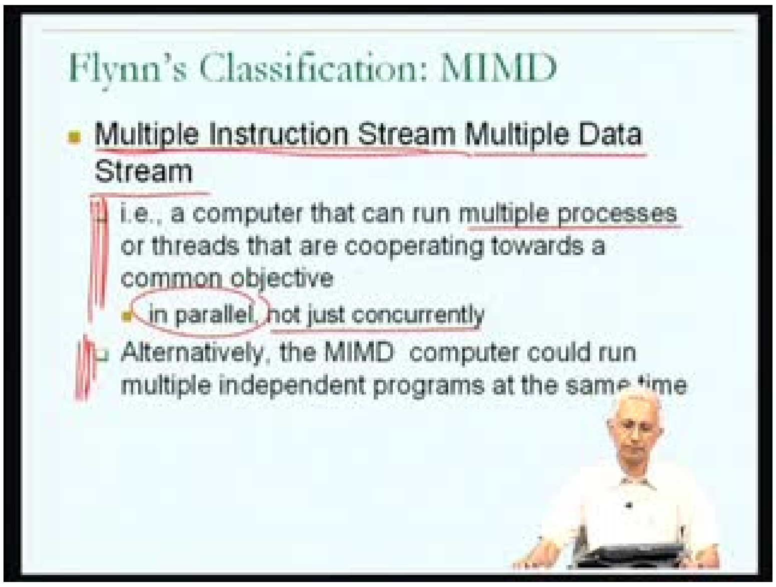
The same MUL instruction is executed on each of the ALUs, but on different pieces of data

So, let us consider a computer, parallel computer, which has only one program counter and therefore, only one instruction register, there is only one instruction which is being executed at a time. But, let us just suppose that, the parallel computer has 1024 ALUs, and that each of the ALUs has a separate independent data path to memory. Now, this is quite different from anything that we have seen before, the computers that we saw had one ALU, which could do arithmetic or at most two ALUs; one to do arithmetic and one to do, lets say, calculation of a branch target address.

Here, we have something a parallel computer, which has been created specifically with the objective of being able to do 1024 different operations at the same time, all driven by a single instruction. Let me just pictorially represent this. So, there is only one program counter, in other words, the address of one instruction can be remembered, the address is going to be some binary value, that it is not important. One instruction register, which means that one particular instruction, which is going to be executed at any given point in time. But, there are one 1024 ALUs; ALU 0 up to ALU 1023, each of which possibly has it is own register file. And therefore, when the instruction multiply A1 comma B1 is executed, each of the ALUs will get it is own values of A1 and B1 and multiply them and proceed.

So in effect, we have the capability of doing 1024 multiplies of pairs of values in a single instruction, and since there is a single instruction doing this, it is called the SI, but it specifying operations on a 1024 sets of data. Hence the MD. Very clearly one could see such a piece of such a computer being quite useful for doing, for example, operations on vectors and, in fact, they have machines called vector super computers or vector computers, which operate very much in this mode, as do many of the graphics processors, that you may have in the laptop or in your desktop. So, SIMD parallel computers are fairly wide spread. Now and that the way to look at this is, the same multiply instruction is executed on each of the ALUs but on different pieces of data, hence allowing an operation on an entire vector to happen, and specified as the single instruction and happening with the individual vector elements being computed in parallel at the same time.

(Refer Slide Time: 14:44)



Flynn's Classification: MIMD

- Multiple Instruction Stream Multiple Data Stream
  - i.e., a computer that can run multiple processes or threads that are cooperating towards a common objective
    - in parallel, not just concurrently
  - Alternatively, the MIMD computer could run multiple independent programs at the same time

Now, that leads us with a Multiple Instruction Stream Multiple Data Stream part of Flynn's classification. Multiple Instruction Stream, implying that, there are multiple program counters, multiple instruction registers. Multiple Data Stream, implying that, each of the instructions could actually have it is own path to memory, operating separately based on what its instructions require. For the first time, this is a truly in the, this may be a parallel computer closure to what we had imagined.

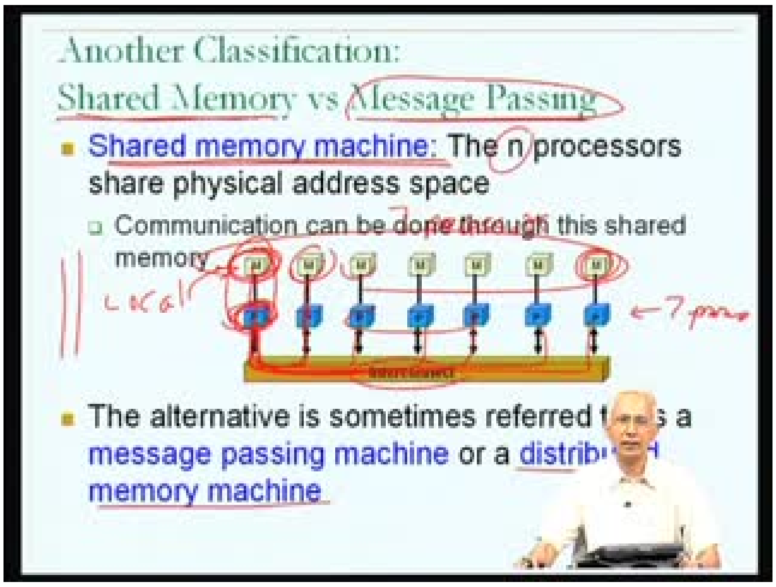
Because this is a parallel computer, that could actually run multiple processes or threads, one on each of its, one represented by each of its instruction streams, therefore, if there are 7 instruction streams and 7 data streams, I can view it as the capability of executing 7 processes in parallel or 7 threads in parallel.

So, this is clearly a different kind of a parallel computer from the SIMD, that we had just seen. The key thing to note here is that, what we understand in the case of the MIMD parallel computer is, there are separate hardware to do each of those processes, and we are not just talking about the processes running concurrently, which is what we were talking about when we had only a single processor, until this point in the course, and the operating system was managing the sharing of the processor among the different processes. Here the situation is quite different, we actually have multiple pieces of hardware, so that, each of the multiple processes can, in fact, run in parallel, in other words, at the same time.

So, that is one possible way to view how the MIMD computer could be used. Multiple processes, which are cooperating towards their common objective and to do create such a program, one would have to write the program specially to run as multiple processes. Now, alternatively if I had an MIMD computer and I know that it contains, lets say, seven processes, each of which can execute a separate process, then I could just as well used that parallel computer, MIMD parallel computer, to run seven independent programs, and therefore, get seven things done at the same time.

So, this is an alternative way that an MIMD computer could be used. They could be used either to run a single parallel program, which had runs a seven processes, in this example, or in fact, to run seven independent programs, and that would be another way to keep the hardware of the parallel computer well utilized.

(Refer Slide Time: 17:00)



Another Classification: Shared Memory vs Message Passing

- Shared memory machine: The n processors share physical address space
  - Communication can be done through this shared memory
- The alternative is sometimes referred t... a message passing machine or a distrib... memory machine

So, with this we get a good idea about how the space of a parallel computers may look, essentially, they are the SIMD type of parallel computers and the MIMD type of parallel computers. But, given that we now expect that the parallel programming is going to involve writing programs, which run as multiple processes or multiple threads, we expect that an important part of parallel programming is going to be the inter-process communication mechanism, through which the individual processes or threads of the parallel program interact towards achieving the common objective. And hence, if one looks back at the architecture, one might try to classify the different kinds of parallel computers, in terms of, how they support inter- process communication. And we have seen many different kinds of support for inter-process communication, earlier when we talked about concurrent programming, but at this point let me just highlight two, which do not involve using the file system. Some of the mechanisms that we talked about use the file system or pipes, but the ones which we talked about which do not use the file system or pipes, one was shared memory, and the other was a technique, which we called message passing, which we did not look at in too much detail at that time.

So, what I am going to assume for the movement is that, these are the two dominant mechanisms used by parallel programmers, for the interaction inter-process communication, required for the parallel programs to have processes working towards the common objective.

So, if these are the two main ways that the parallel programmers typically think about parallelism, then it must be the case that the hardware, underlying hardware, will either support one or the other, and therefore, one could classify parallel architectures, in terms of whether they support shared memory or whether they support message passing. So, this is an alternative classification to Flynn's classification.

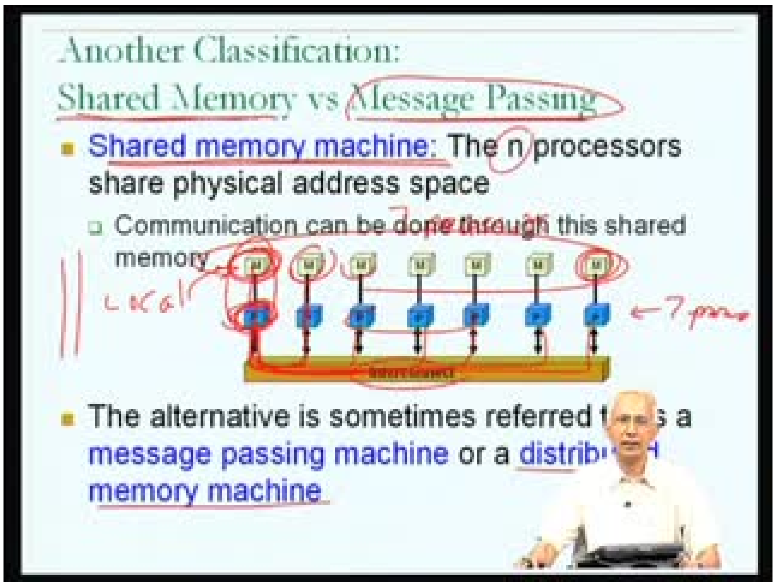
So, I could talk about shared memory parallel computers or shared memory machines and the characteristic of a shared memory machine would be that, there is more than one processors, I will call it n processors, but there, all the n processors share a physical address space, in other words, they share memory, and it is through this shared physical address space that the processes which run on the n processors can, in fact, communicate and work towards the common objective.

So, if I find the parallel computer in which the processors have shared physical address space, I could call it a shared memory machine, whether it be SIMD or MIMD. So, you can see that there is an orthogonality to this classification, when if you look at it from perspective of Flynn's classification.

So, how would I represent, how pictorially how would I think of a shared memory parallel computer. I may think of a shared memory machine or a shared memory parallel computer as being something like this. In this diagram, I am showing you the blue boxes which are the processors. So, in this particular example, there are 1234567 processors.

So, these are 7 CPUs with registers, MMUs the whole works, each of the 7 boxes, but they share a physical address space, and therefore, I show only one main memory constituting the physical memory, which is used by all the 7 processors. And, of course, there must be a mechanism through which they are connected to each other, which I represented by an interconnect box.

(Refer Slide Time: 17:00)


Another Classification:
Shared Memory vs Message Passing

- Shared memory machine: The n processors share physical address space
  - Communication can be done through this shared memory
- The alternative is sometimes referred to as a message passing machine or a distributed memory machine

So, this is one way to view a shared memory machine. The idea being, that there are multiple processors, more than one, and they all have a shared physical address space, which I represent in this diagram by the fact that there is a single main memory.

Now, and I again underlying this is the assumption that, the way that a parallel program would be written is by writing a parallel program in which a shared processes have shared variables, and the communication between, the shared, the processes in order to achieve the common objective, happens through the way the shared variables are used, to pass information from one process to the other.

Now, the alternative to a shared memory machine would be, a machine with supports message passing, and therefore, I could talk about machine which is the message passing machine, and sometimes people referred to message passing machines as, distributed memory machines, to sort of reflect that they are not shared memory machines. And once again if I had to a draw a diagram, a block diagram, to represent a message passing machine or a distributed memory machine, I might draw something that looks like this.

Now, once again there is an interconnect, connecting everything together, there are many processors, in this case, once again 7 processors, that is one of the requirements of being called the parallel machine, but since the 7 processors do not communicate through memory, but rather by another mechanism which is called message passing, they do not have the need for a shared physical address space, and hence I show each of the processors as having its own memory.

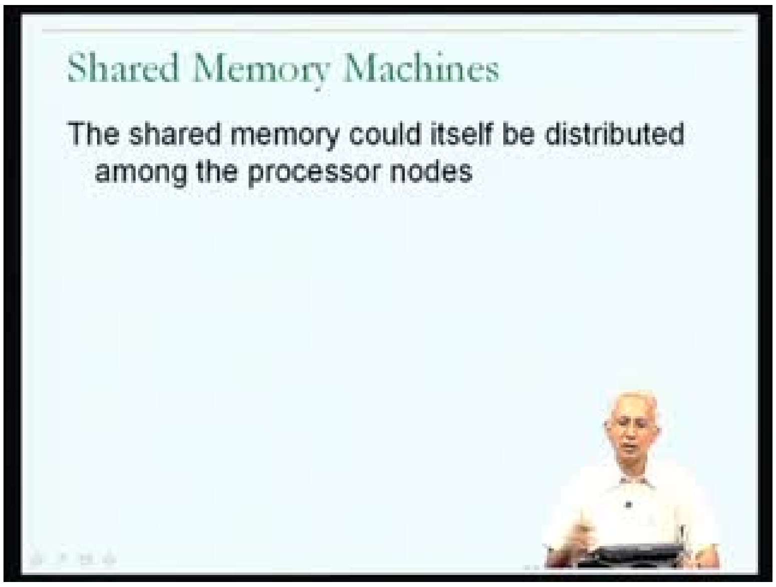
So, this processor gets all its instructions and data out of this memory and so on. So, this is one pictorial representation of what might represent a message passing machine or a distributed memory machine. Now, there is a final point to be noted here, and that is, it is conceivable that the diagram, diagrammatically, the entity which is represented on the screen could, in fact, be a shared memory machine. Because it is conceivable that the machine is constructed such that, a process running on any one of the processors could access anything in its local memory, but it could also access anything in any of the other memories, by going through the inter-connect and the associated node or processor. And if this is the way that the hardware is built, then the diagram on the screen, even though it seems to represent a machine in which each processor has it is own memory, could functionally represent one in which, each processor can access all the memory in the system. There is an interconnect, and therefore, this process, the process running on anyone of the processors the hardware could be built, so that, the process running on anyone of the processors could acts as any part of the memory, despite the fact that the memory is distribute across the 7 nodes of the system.

Therefore, just looking at a block diagram like this does not really tell us whether a machine is a shared memory machine or a message passing machine. And in fact, if one thinks about it further, one might realize that for a shared memory machine, in other words, a parallel machine, in which there is more than one processor and all the processors have a shared physical address space, there might be some benefit from building the shared memory machine along the lines of what is on the screen right now. The benefit would be, therefore, any one of the processors there is going to be some amount of the shared physical address space, which is locally accessible at very high speed, and therefore, the variables, the program of that, which is important for that particular processor, could actually be represented in that region of the shared physical address space. And if there are variables which are not used a lot by that particular process, then they could be represented in another region of the shared physical address space.

And therefore, this allows a little bit more flexibility to the programmer in deciding where the data is placed, in order to get better performance, out of the parallel program.

Therefore, we expect that there could be parallel machines, which are shared memory machines, but in which the shared memory is purposely divided across the processors of the system for a performance reason.

(Refer Slide Time: 24:15)

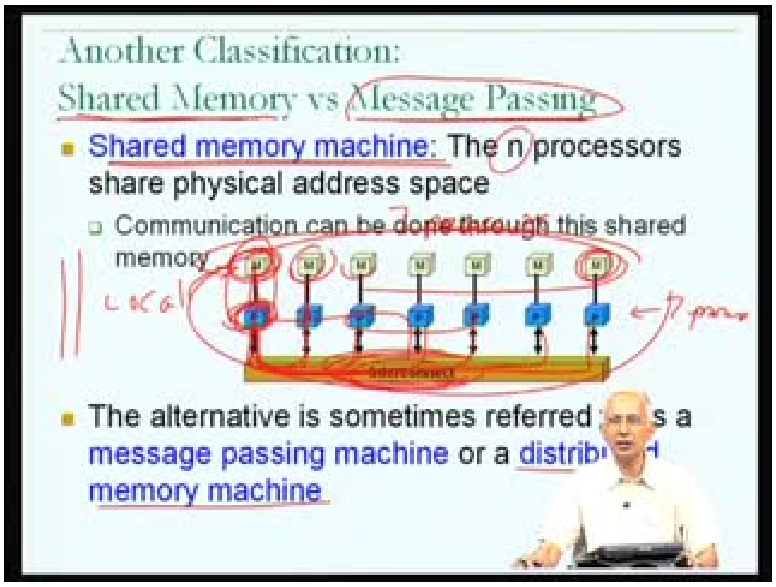


Shared Memory Machines

The shared memory could itself be distributed among the processor nodes

They would still be refer to a shared memory machines, since they do not use message passing as the mechanism for inter- process communication. Let us make a few comments about shared memory machines.

(Refer Slide Time: 24:36)

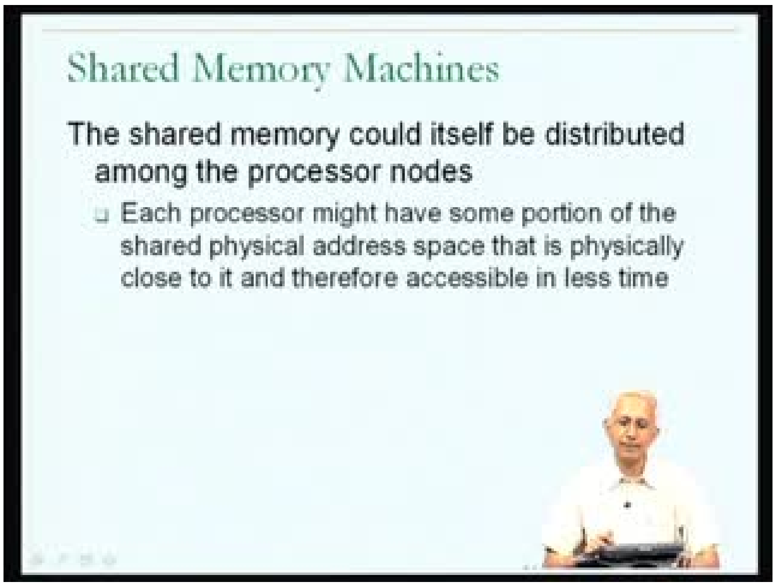


Another Classification: Shared Memory vs Message Passing

- Shared memory machine: The n processors share physical address space
  - Communication can be done through this shared memory
- The alternative is sometimes referred s a message passing machine or a distrib memory machine

Now, as I have just said, the shared memory could itself be distributed among the processor nodes, in terms of, trying to build a shared memory machine, which has more performance capabilities than one in which all the processors have a single shared memory, and you will notice immediately that in a shared memory machine

(Refer Slide Time: 25:03)



Shared Memory Machines

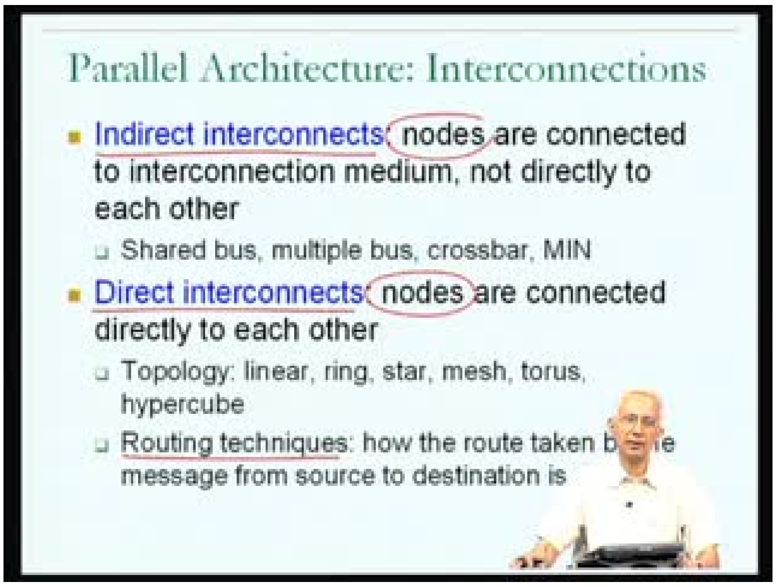
The shared memory could itself be distributed among the processor nodes

- Each processor might have some portion of the shared physical address space that is physically close to it and therefore accessible in less time

If all the processors have a single shared memory, then any instruction fetch or any data fetch would have to take place to the single shared memory, and that single shared memory could become a bottle neck. Therefore, it makes more sense to try to construct to build the shared memory parallel machine with this kind of an organization, with this little bit of the shared memory close to each of the processors, which is what this point is elaborating, the shared memory could itself be distributed among the processors nodes. In other words, each processor might have some portion of the shared physical address space, that is physically close to it, and therefore, accessible in less time.

Now, an important part of the parallel architecture is the inter connection mechanism. We had boxes which were labeled inter connection, and I did want to spend the little bit of time talking about the different kinds of inter connections, that one may encounter in a parallel machine.

(Refer Slide Time: 25:33)


Parallel Architecture: Interconnections

- Indirect interconnects: nodes are connected to interconnection medium, not directly to each other
  - Shared bus, multiple bus, crossbar, MIN
- Direct interconnects: nodes are connected directly to each other
  - Topology: linear, ring, star, mesh, torus, hypercube
  - Routing techniques: how the route taken by the message from source to destination is

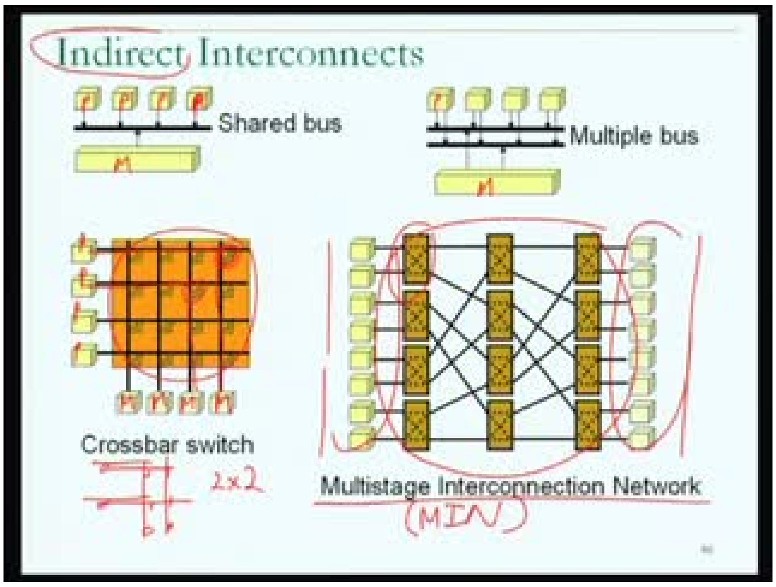
Now, in general there are two classes of inter connections. Remember, the inter connection is the mechanism through which, the processors, the memories, the IO devices of the computer are connected to each other. The first kind of interconnect is refer to as an indirect interconnect, because the various things or nodes, which are connected to each other by the inter connection are not connected directly to each other, rather, all of them are connected to the interconnect, and through the interconnect they are connected to each other.

So, this is what is known as an indirect interconnect. The alternative is what is known as a direct interconnect, in which, the things which are being connected through the interconnect, which are refer to as the nodes, and the nodes could be processors, they could be memories, they could be IO devices. But the nodes are directly connected to each other by the interconnect, and as we look some of the examples we will understand what this means.

But, there are in general two classes of interconnects which are either called indirect or direct, and some of the names of indirect interconnects are: the shared bus, the multiple bus, the cross bar and the min, which we will talk about shortly, and some of the topologies of direct interconnects are; linear, ring, star, mesh, torus and hypercube, which we look at again briefly.

In the case of direct interconnects, and interesting aspect of the interconnect hardware is the routing technique, the decision or how the decision about how to route a message from the processor, which is initiating the communication to the processor which is to meant to be the destination of the communication is an interesting, an important attribute.

(Refer Slide Time: 27:07)


Indirect Interconnects

Let us look at some indirect interconnects. Now, the first indirect interconnect, remember, an indirect interconnect is one in which the different nodes are not connected directly to each other, but rather, each of them is connected to the interconnect. And the bus, which we had talked about until now is an example of an indirect interconnect. Because if you think about it, when I had a processor, in this diagram, I am showing you 4 nodes; one of them could be a processor; the other could be a memory module and the others could be IO devices. And therefore, this diagram corresponds to the block diagram of an ordinary sequential computer system, that we were using up to now, and they are all connected by the shared bus.

Now, you notice that the processor is not connected directly to the memory, all that is done is both of them are connected to the shared bus and through the shared bus they can communicate with each other, which is why this is called an indirect interconnect. Now, one property of the problem with the shared bus as a interconnect is that, the shared bus itself will be a bottle neck, because any communication between one node and another node, whether the node would be a processor or a memory module, and this diagram I might have 3 processors and one memory module, this might be a shared memory machine, but in any given point in time, only one communication can be happening over the shared bus..

The shared bus itself becomes the bottle neck. I could talk about having 3 processes running on this parallel computer, but in effect, only one of them could be using memory in any clock cycle, because the shared bus does not allow more than one activity at a time.

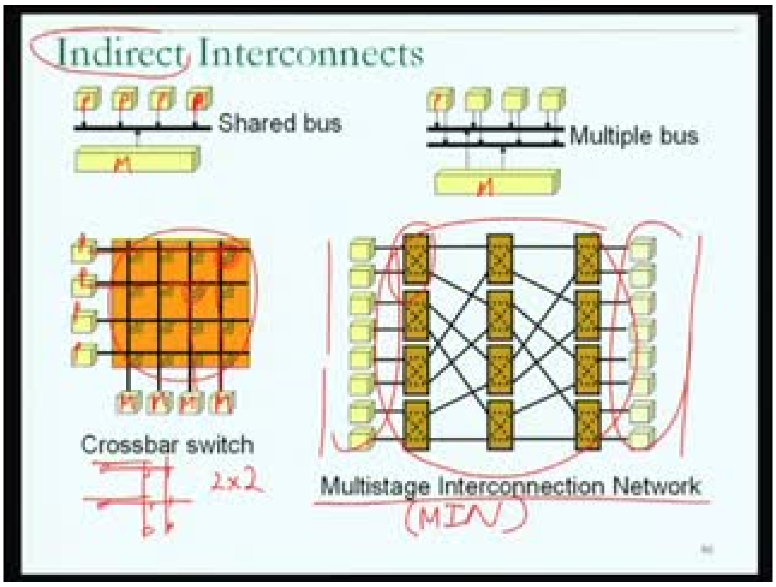
One could improve this property of a shared bus by actually having, well in this diagram what I am doing is, I am actually going to show you the memory which is shared, this is diagram of a shared memory parallel machine..

So, in I have now I am ended it.

So, they are 4 processors which are sharing one memory. So the problem now is that, only one of the 4 processors can actually being accessing memory at a time, thus a shared physical address space, a shared physical memory, but only one of them can be accessing memory at a time, because there is only one shared bus.

I could improve this problem with the shared bus by actually having a modified version of the shared bus, in which there is more than one shared bus in the parallel computer, and this could be called the parallel computer, which uses a multiple bus interconnect. And the idea here is that, I could have, in this diagram, I am showing you the situation where there are two shared buses and each of the nodes whether it be a processor or the memory is connected to each of the shared buses, and hence at any given point in time I could actually have 2 processes communicating with memory. And this will have twice the memory bandwidth of the first, and therefore, it would should we viewed as being an improvement over the first. And you could keep on increasing the number of shared buses, until let us say the number of shared buses was equal to the number of processors, and that would improve the performance, because you could now have all the processors being able to access main memory at the same time.

(Refer Slide Time: 27:07)



In an example of a more general kind of an interconnect, which uses that kind of an idea is, what is known as, the crossbar switch. The way that a crossbar switch might be used to construct a shared memory parallel machine, might be that I have is notice if the crossbar switch has the capability of having nodes connected on the left, and in this particular diagram, also nodes connected of the bottom.

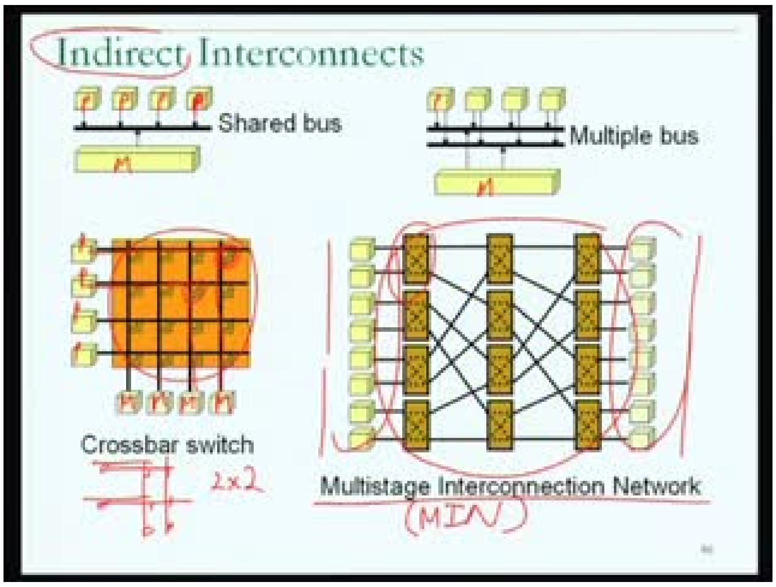
So, I might have my processors aligned on the left and different main memory unit is aligned on the bottom. What the crossbar switch could provides me with is, lets say, a number of buses, there is one bus associated with each processor, which are the horizontal thick lines, and there is one bus associated with each main memory, each part of main memory, those are the vertical thick lines.

Then, they are the interconnections between these two pairs of buses, which are the right angle arrowed boxes in between, each of which is typically refer to as a cross point. So, the crossbar switch itself, as an interconnect, is the collection of cross point switches, there is 16 cross point switches, this is one, any one of the cross point switches which I have circled. So, the collection of those 16 cross point switches provides the capability for the crossbar switch to provide a connection, between let say, through this cross point, this processor can be connected to this memory, to this cross point this processor can be connected to this memory and so on.

Therefore, by a appropriately setting the cross point switches, the hardware is capable of allowing 4 processor to be connected to 4 different memory modules at the same time, giving a great deal of flexibility, increasing the complexity of the interconnection a little bit, but allowing for a much increased utilization of the memory bandwidth. So, this is what is known as a crossbar switch. It has a problem that it is much more complex than the shared bus. In this discussion of interconnects, one can see that there is going to be a tradeoff between the performance, which is the effective memory bandwidth that can be used by the processes running on the processors, and on the other hand the tradeoff of bandwidth against the complexity of the interconnect, and the crossbar switch is clearly quite complicated. When I talk about connecting 4 processors to 4 memory modules, I was referring to a cross bar switch, in which they was 16 cross points, if I was to think about a larger kind of a parallel machine, which say, a 100 processors connected to a 100 pieces of memory, then there would be 10,000 cross points within the cross crossbar switch, which is going to be making the crossbar switch a very complicated and potentially a bottle neck, as far as it is operation.

So, there are intermediate kinds of indirect interconnects which try to have some of the benefit of the shared bus and some of the benefit of the crossbar, in terms of the tradeoff between complexity and performance, and one of these is what is known as the multistage interconnect network, and the idea in the multistage interconnection network is that, if we use a if we use very simple crossbar switches, but use a number of crossbar switches rather than a single very complicated crossbar switch, then we might be able to play the tradeoff between performance and complexity, to the benefit of the program or the programmer.

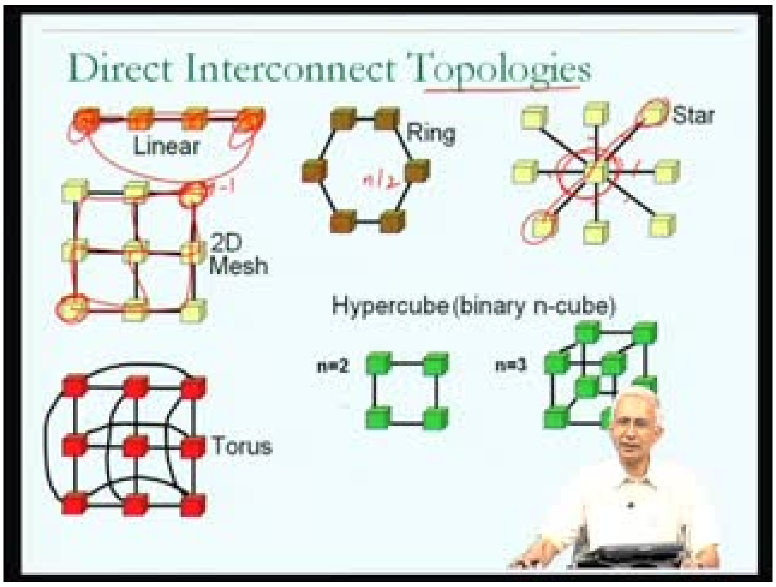
(Refer Slide Time: 27:07)



So, the idea of the cross multistage interconnection network, that I am going to represent is that, I want to connect, let say, 8 processors which are the 8 boxes on the left to 8 memory modules which are the 8 boxes on the right, to create a parallel machine using small crossbar switches. The brown diagram, which I have drawn there is meant to represent, a small 2 by 2 cross bar switch, when what I mean by a 2 by 2 crossbar switch is one which can connect 2 inputs to 2 outputs through the use of 4 cross points. So, this is a 2 by 2 crossbar switch. So, this diagram represents a 2 by 2 crossbar switch. So, this can has 2 inputs are shown over here and 2 outputs. So, if I connect the 2 inputs to 2 of the processors, and I similarly I have a larger number of such 2 by 2 crossbar switches, and at the other end I have crossbar switches which connect to the actual memory modules, then with an intermediate layer of 2 by 2 crossbar switches, which are connected both to the switches on the left and switches on the right, where appropriately setting all of the crossbar switches which constitute the multi stage interconnection network, which is often abbreviated to min. I can actually provide connections between each of the 8 processes to 8 distinct memory modules at the same time, and therefore, very good memory bandwidth at much less complexity, in terms of the number of cross points as compared to the crossbars switch.

So, the multi stage interconnection network is an attractive intermediate between the crossbar switch, which is very good on bandwidth, but somewhat unpleasant from the perspective of the complexity, and therefore, the cost of the interconnect.

(Refer Slide Time: 34:39)



So, this gives us some idea about the different kinds of dominant indirect interconnects. What about the direct interconnects? In the case of the direct interconnects, one talks about the, remember that objective of a direct interconnect is to directly connect one node of the parallel machine to another node, as shown in this diagram, notice that the node label 0 is directly connected to the node label one, which is also directly connected to the node label 2, node label 2 is connected not only to one, but also to 3, and these links constitute the interconnect. It is a direct interconnect because the nodes are directly connected to each other and therefore, we talk about not so much about the interconnect by name, but by the topology represented by the nature of the interconnections.

So, the first diagram shows 4 nodes, 2 of them could be processors, 2 of them could be memory modules or each of them could be a node, which contains both the processors and some portion of memory. And that the first is showing a what could called a linear interconnection, you will notice that the connections are such that, each of the nodes has utmost 2 neighbors, and they form a straight line, hence the name linear. And that the property of the linear interconnect is going to be that, one could talk about the amount of time that it will take for a communication between a processor running on node 0 and the processor running on node 3, and you notice that if the processor or node 0 wants to communicate with that on node 3, then the message would have to pass through the intermediate nodes. And in general if there was a shared memory machine, which had n nodes in it then, in the worse case, a communication between one and another of the processes running on the nodes would have to take n minus one steps to the interconnection. And therefore, that is an unpleasant property of the linear interconnect which is why people will sometimes talk about a ring, which is like the linear, but the two ends are connected to each other leading us with this closed representation, one could have unidirectional rings or by directional rings, in the property would be that, it takes in the worse case, the number of links the number of links that one has to go through in communication between one node and another would be less than that in the case of the linear.

For example, in the case of linear, the worse case was one went from one the node at one extreme to the node at the other extreme; in the case of the ring, that communication would know boil down to two neighbors, and the worst case communication would be to go half way through the ring, which is improving on the number of steps, the number of hops through the interconnect, that was necessary in the case of the linear.

So, some of the other topologies that are fairly wide spread are the star topology, which looks like a star and has the property that there is one central node to which all the other nodes are connected, and therefore, in the worse case, communication between any one node and any other node would required only two steps, two hops through the interconnect, which is much better than the approximately n by 2 or the n minus one property of the previous two interconnects.

But the weakness of the star is that, if the central node fails, or if any of the links into the central node fails, then many of the processors in the share memory machine would be isolated from each other, and in shared memory machine would not be able to operate. In fact, if the central node fails then the shared memory machine becomes completely useless.

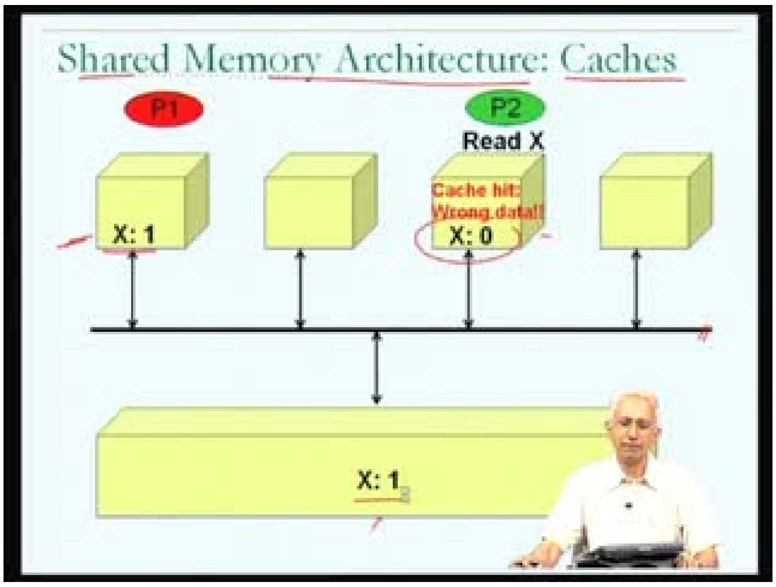
So, there are topologies which are more resilient, such as, the topology shown over here, which shows the nodes connected to each other in the shape of a mesh, in fact, this kind of a direct interconnect is known as a two- dimensional mesh, and is more resilient because there are multiple paths, multiple different ways or routes from any one node to another. For example, if I think about how a message from the node, which I have just circle to the node in the far corner could go, you will notice that they are many possibilities and therefore, the failure of anyone node and anyone link could not cause the entire parallel machine to become useless.

So, this is an example of a two- dimensional mesh, since it is a mesh which can be laid out in two- dimensional, can be drawn in two- dimensions. There could be three-dimensional meshes as well. There could also be meshes, in which, in addition to the links that are shown, there could be additional links, for example, in the diagram which I have shown at the bottom, I have shown a mesh then I have connected the nodes at the extremes, and you will notice that by doing this I am reducing the worse case number of steps, that have to be gone through to go from anyone node to another. And in fact, this kind of an interconnect would be known as a torus or a 2D torus, in this particular case.

Now, there are even more complicated kinds of direct interconnects, one of which is the hypercube or n dimensional cube, it is also known as the binary n-cube. And the simplest example of the binary n-cube would have 4 nodes connected as a mesh, if one wanted to go through the next larger hypercube, one would double the number of nodes and connect one from each half of the hypercube to the corresponding node in the other half of the hypercube, and one could proceed to apply the same principle to have hyper cubes of dimension 456 etcetera, and this topology has very good properties in terms of, not only the maximum distance, the maximum number of links, one would have to go through in order to communicate between any two of the nodes, but as well as various properties.

Now, the parallel machines that we are going to see would use one of these different kinds of topologies, unless they will based on networking hardware, in which case, they may have these kinds of topologies embedded into the nature of the networking switches.

(Refer Slide Time: 39:53)



Now, proceeding into slightly more abstract mode, I will from now on, some of the examples which are immediately going to follow, we want to concentrate on some of the architectural issues involved in having, lets say, a shared memory parallel machine, and in a shared memory parallel machine, given that we expect this going to be in this diagram, I am showing you 4 processes, but in general, multiple processors all of which have shared memory, let me assume for the moment that they are connected by a single shared bus. Then you will note that, we had the problem that, for any one of the processors to access anything out of the memory, it had to go through the shared bus and therefore, we suspect that for things like fetching instructions or for accessing data, it would be beneficial for the performance of the shared memory machine, if each of the processors had it is own cache. Then once an instruction had been fetched into the cache of that processor, temporal locality could be exploited and performance of the parallel program would benefit. So, it makes sense to assume that, the individual processors in a shared memory machine, in fact, have caches, but lets see how things will operate in a shared memory machine if the individual nodes have processors, if the individual processors have caches.

So, in this diagram, I am going to assume that the small boxes at the top represent the individual nodes, and I am going to concentrate on the caches, therefore, you could actually view the box at the top as being a processor and the dominant feature which we are talking about now is it is cache. So, I could have a situation, where there is a parallel program running on such a parallel machine, it is running using shared memory, and it is running as processes which are running on the different processors.

So, let us suppose that I have a parallel program running as two processes; process P1 is running on the processor on the left, process P2 is the green process running on the third processor, and process P1 and process P2 share a variable which I will call X. Remember we are talking about a shared memory architecture therefore, there could be shared variables. So, the current value of the variable X is 0. Let us suppose that at this point, process P1, process P1 and P2 are running processes, are processes of a single program, the program was written to use shared variables, X is such a shared variable.

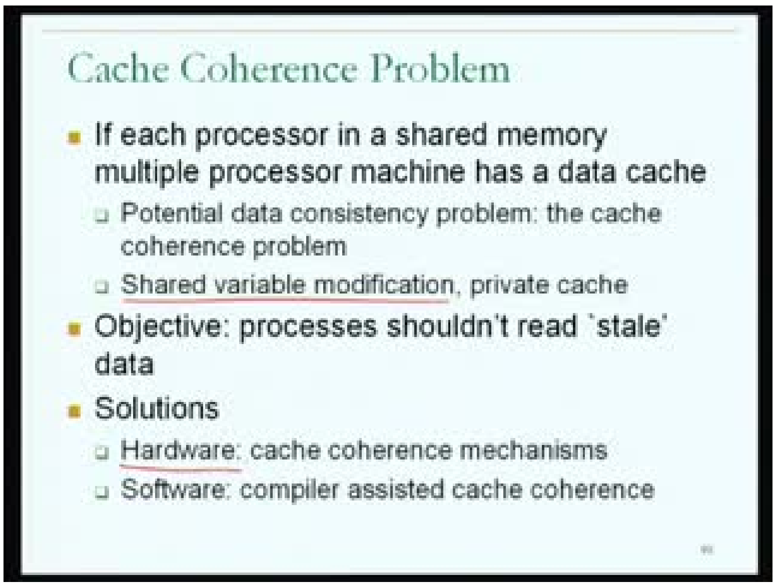
Let us suppose that process P1 reads X. Now, X is a variable, it is represented in memory. When process P1 reads X it will get a cache miss, remember that we are assuming that each of the process the processors has a cache. So, process P1 will suffer a cache miss, the effect of the cache miss is going to be that, the block containing the variable X will be fetched from main memory into the cache of processor, the processor running process P1.

So, the diagram will then look like this, as in effect of the read X. And I will show you a copy of the block containing X, in the cache corresponding to P1. P1 can now read X many times and it will be able to access X at cache speed. Now, what if at this point in time process P2 reads X. Remember, this is a parallel P1 and P2 are processes of a parallel program, you can think of them as threads if it helps in understanding, but they have shared variables, therefore process P2 also can read X, when it reads X it gets a cache miss, and once again the effect of the cache miss is, just as we saw in the case of our simple processors.

The variable will be, the block containing the variable X will be copied from main memory into the cache of the process processor running process P2. So once again, process P2 can access the variable X out of it is cache subsequently at nanosecond speeds. What if now process P1 writes or changes the value of the variable X, using a store instruction, this is perfectly legitimate operation on a shared variable, the net effect is going to be that, the value of X inside it is cache gets change to one. Let me also assume that the value of X inside the main memory gets change to one, possibly by a right through mechanism, but let me just make that assumption, so we now see that after initially the value of X was 0 in one of the caches, in the other cache as well as in main memory, subsequently process P1 modify the value of X to 1 and therefore, that changes the value of X in the cache where process P1 is running, and we are assuming it also changes the value of X in the main memory. What happens now if process P2 reads X again?

If process P2 reads X at this point in time, it gets unfortunately a cache hit. It will get the value X equal to 0, and this is not the correct value of X as far as the parallel program is concerned, because according to the parallel program, X has just been modify to one by it is parallel process P1. Therefore, the problem that we see is, process P2 when it reads X the second time will get the wrong value of X, it will get the old stale value of X equal to zero, because it got this cache hit. It is getting the wrong data, the program will not do what it was expected to do by the programmer. So, this seems to be a problem with having caches in a shared memory parallel machine.

(Refer Slide Time: 44:57)



Cache Coherence Problem

- If each processor in a shared memory multiple processor machine has a data cache
  - Potential data consistency problem: the cache coherence problem
  - Shared variable modification, private cache
- Objective: processes shouldn't read 'stale' data
- Solutions
  - Hardware: cache coherence mechanisms
  - Software: compiler assisted cache coherence

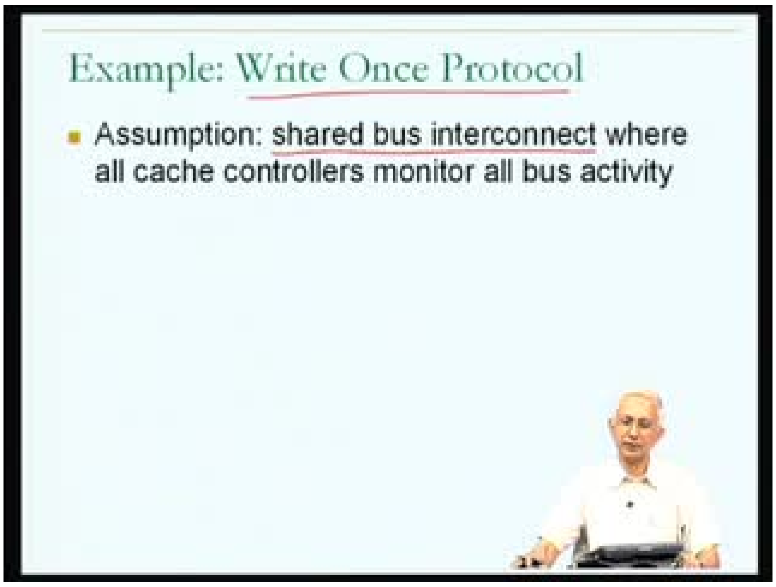
This is a serious problem with caches in shared memory parallel machines, it is known as the cache coherence problem. Essentially, there is a problem in maintaining the coherence of the multiple copies of data that could be present in the main memory as well as in the many caches, which we now understand, would be present in a shared memory machine, one cache for each of the processors at least.

So, the problem, the cache coherence problem arises, if each processor in a shared memory multiple processor machine has a data cache, and the problem arises when it is a potential data consistency problem, it rises when there is a modification to a shared variable in the cache of one of the processors. As we just saw, the cache of the processor where process P1 was running, when process P1 executed the store instruction to the variable X cause this problem to arise, and the problem arose because of an attempt to modify or a modification of a shared variable, subsequent reads of the shared variable from other processors would get the wrong value.

Now, very clearly when one builds a parallel machine, one cannot build a parallel machine purely for the reasons of performance in order to make accesses fast by having caches and then cause programs to execute in a mode, where some of the processes get the wrong data. One of the objectives of building a parallel machine must be that the parallel program runs and produces the result that the programmers expect. And therefore, it should never be a situation that a process reads stale data, such as what happens, and therefore the designer of a parallel machine must design the machine to overcome this problem, this is not problem that can be allowed to exist in the execution of programs on parallel machines.

Now, they are two different kinds of solutions that are used to handle the cache coherence problem. In one, the solution is built into the hardware, in other words, the cache coherence mechanism is built into the hardware of the parallel machine, so that the processes never reach stale data, but always get the correct data. The alternative is that, the compiler could, there could be a software scheme, where the compiler assisted mechanisms are used to ensure that the problem does not arise. I am going to describe one hardware mechanism, just to give us a clearer picture of what might be happening in a shared memory parallel machine.

(Refer Slide Time: 47:20)

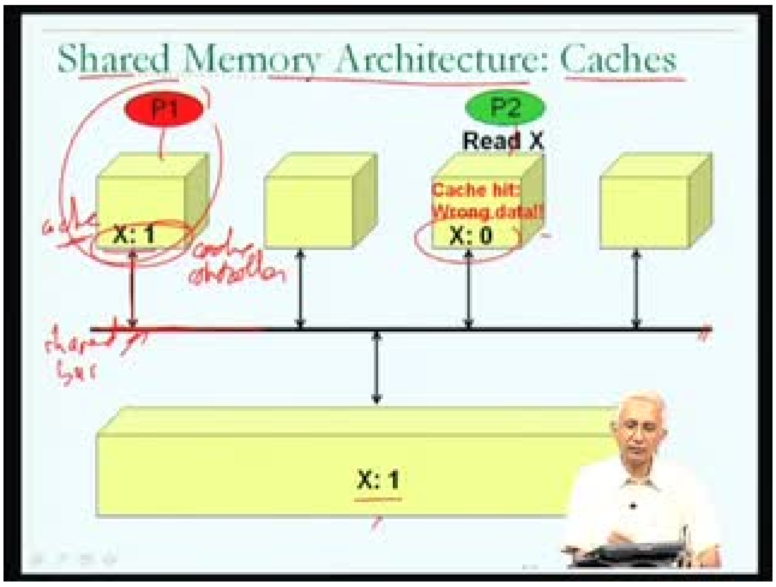


Example: Write Once Protocol

- Assumption: shared bus interconnect where all cache controllers monitor all bus activity

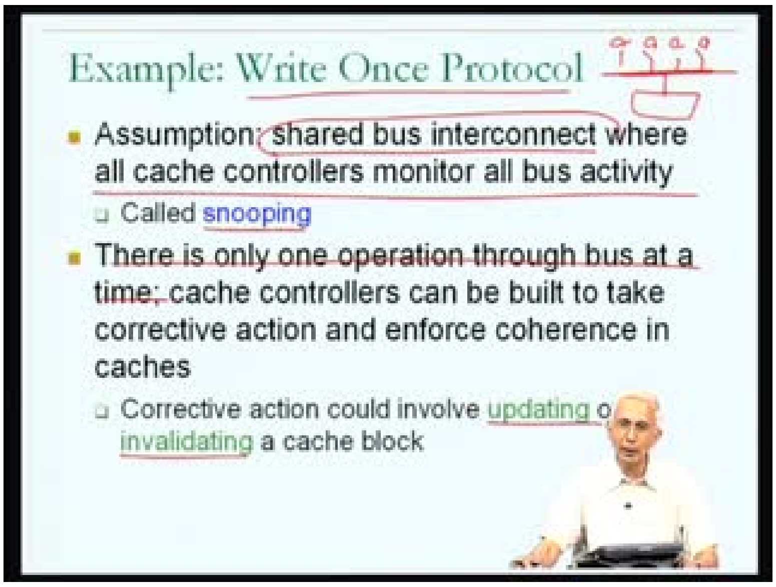
Now, the particular mechanism which I am going to talk about is, a cache hardware cache coherence mechanism, and it is known as the Write Once Protocol, it is the protocol, which is protocol is a communication mechanism, you have heard about networking protocols, you have heard about political protocols, basically they are standardized mechanisms for communication to avoid confusion, and this particular possible confusion that we could have in this context is that, one processor could write a piece of data and another processor might read the wrong piece of data, the objective of the Write Once Protocol is to correct that problem.

(Refer Slide Time: 48:17)



Now, the assumption or the Write Once Protocol can be built into a shared memory machine, in which the interconnection is done through a shared bus. Further, it is expected that all of the cache controllers have the capability of monitoring all the bus activity. And just going back to the diagram which we had representing the shared memory parallel machine, if you think about it, I have shown the shared memory parallel machine as dominated in terms of it is interconnection to the bus. So, this is the shared bus, the thick line over here, and in effect, the diagram which I have shown has a yellow box, which is the cache of the processor and so, each of the processors has a cache, and therefore, the assumption which we are making about the write once protocol is that, it is the cache controller, the cache controller is the part of the cache which manages the operation of the cache, we will remember.

(Refer Slide Time: 47:20)



Example: Write Once Protocol

- Assumption: shared bus interconnect where all cache controllers monitor all bus activity
  - Called snooping
- There is only one operation through bus at a time; cache controllers can be built to take corrective action and enforce coherence in caches
  - Corrective action could involve updating or invalidating a cache block

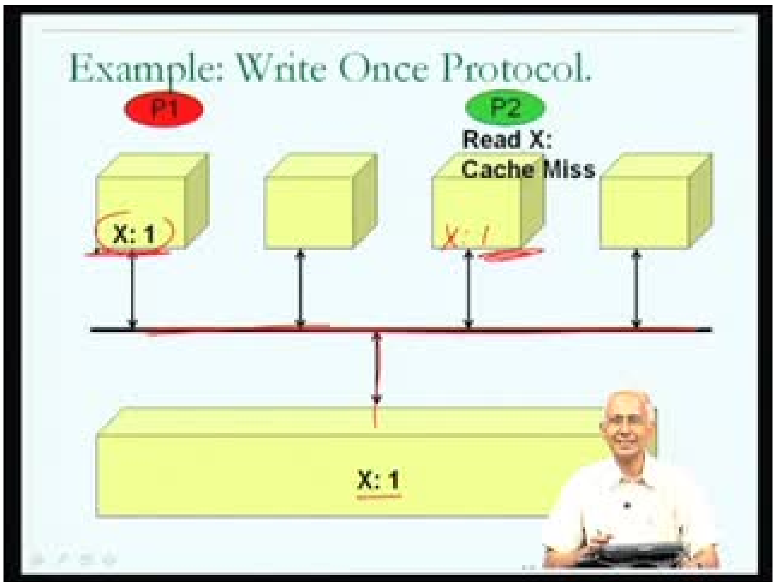
But that the cache controller has the capability of monitoring the activity on the bus, in other words, the cache controller is what is connected, is connected to the interface between the processor and the bus, and has a capability of seeing what is happening on the bus or monitoring the activity on the bus. So that is the base assumption which we are talking about, all cache controllers can monitor all bus activity.

Now, this property of assuming that of having all the cache controllers able to monitor all the bus activity is what is known as snooping, because in effect, each of the cache controllers is snooping or looking at the activity on the bus, which may have been generated by other processors in the system. So, even though the activity on the bus has potentially nothing to do with this particular cache controller, each cache controller is constantly keeping track of what is on the bus, hence it is given the word snooping.

Now, in general the reason for assuming that we are dealing with a shared bus interconnect is, the problem that we had seen with a shared bus interconnect earlier. The problem with a shared bus interconnect was, I could have a shared bus interconnect which connects 4 processors to one memory, but at any given point in time, only one of them could be connected to memory. In other words, only one activity can be happening on the bus at a time.

So, this was a problem from the perspective of the bandwidth of utilization of the memory, but from the perspective of our cache coherence protocol, one can give it as benefit, because on the shared bus only one activity or one operation can happen at a time. Therefore, if all the cache controllers are snooping on the bus, then they can collectively take corrective action to make sure that the cache coherence problem does not result in a processor, a process incorrectly reading a piece of stale data. So, the negative aspect of low bandwidth of the shared bus interconnect is being exploited to cause coherent cache behavior. Now, what kind of corrective action could the cache controllers make? Depending on what they observe on the bus, they could either invalidate their copy of a particular block or they could update the copy of a particular block, based on the activity that they have seen on the bus.

(Refer Slide Time: 51:08)


Example: Write Once Protocol.

Let just run through the example that we are just seen, in order to see how a Write Once Protocol might avoid a cache coherence problem. So once again we have 4 caches, 4 processors connected to a shared bus through which they can access main memory, and I have process P1 running on the red process running on the left processor, process P2 running on that third processor, they have a shared variable called X which is initially zero. When process P1 reads X, it gets a copy of the block containing X in its cache, when process P2 reads X, it gets a copy of the block containing X on its cache.

At this point let us assume that process P1 writes X. So, in writing X, if we as what the Write Once Protocol will do is, it will cause the cache copy of X to get modify to one, it will also cause the main memory copy of X to be updated. Now, when the main memory copy of X gets updated, that can only happen by a bus activity, by an operation across the bus. And if all the other caches in the system remember, so when process P1 modified X, it updates it is own copy and it also updates the cache copy, but the important thing is that, in order to update the main memory copy there has to be a bus transaction. And when there is a bus transaction, if all the other caches in the system, cache controllers in the system or monitoring or snooping on the bus, then it is possible for them to invalidate there copies of the block containing X, which is what is represented by this figure.

So in effect, having X modified by process P1 is that the copy of the block in that cache becomes as X equal to one, the main memory copy as X equal to one, because of the right through in effect, and the bus activity was monitored or snooped at by the cache controller of the third processor, which notice that it is a block which it has apparently an old copy in its cache, it therefore invalidates its copy, so that when process P2 now tries to read X, it gets a cache miss, which makes the behavior of the program better, because there is no danger of it reading the old data of X equal to 0. Getting a cache miss is bad for performance, but it is good for the correctness of the program, because now when there is a cache miss, process P2, the cache containing process running process P2 will have to fetch a copy of the variable from memory, and then proceed as before.

So, now the net effect is that by having this Write Once Protocol built into the hardware, it is possible for the program to run correctly. Process P2 does not incorrectly read the value of X, but gets a cache miss, subsequently the cache miss results in the variable, the block containing the variable X to be fetched into the cache, and process P2 reads X correctly.

So in short, in today's lecture what we have seen is, we have seen that there is a reason for wanting to use a parallel machines. A parallel machine is a machine, a computer which has more than one processor. By having more than one processor, one can have improved performance or reduce the execution times for programs. Alternatively, one could use the different processors to increase the fault tolerance of the application, the application would then be able to continue execution even if one of the processors failed.

We saw that there are different classifications of parallel computers. Flynn’s classification, in which there is predominantly the Multiple Instruction Multiple Data Stream, mode of operation. There is also the idea of the Single Instruction Stream Multiple Data Stream type of a computer, but we are using more another classification which talks about parallel machines, which are good for shared memory types of operations and parallel machines, which were primarily design for message passing kind of communication. We looked at the different kinds of interconnects which could be used in a parallel machine, and then identified that there could be a problem with shared memory parallel machines if each of the nodes of a shared memory parallel machine has a cache. We will proceed to look at some more ramifications of this cache coherence problem in the next lecture, and we will continue to look at different aspects of parallel architecture and programming in the lectures to follow. I will stop here for today. Thank you!