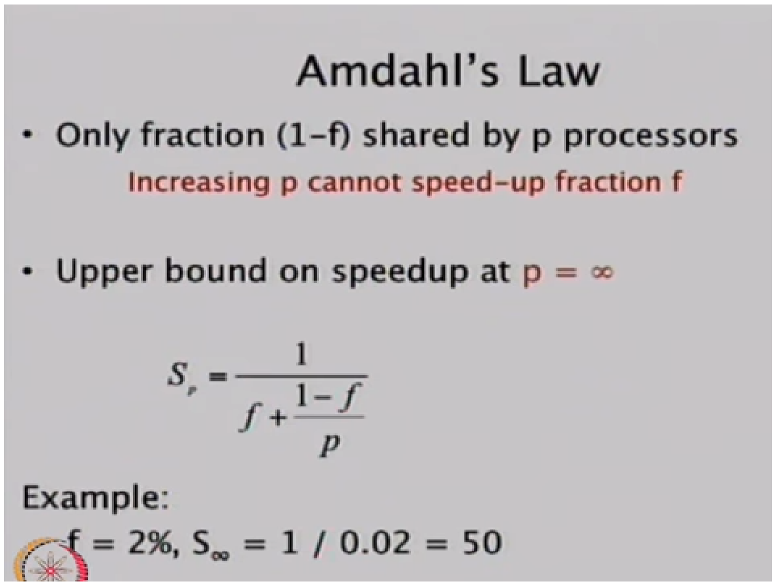
Parallel Computing

Prof. Subodh Kumar
Department of Computer Science & Engineering
Indian Institute of Technology – Delhi

Module No # 01
Lecture No # 03
Parallel Architecture

(Refer Slide Time: 00:34)



Amdahl's Law

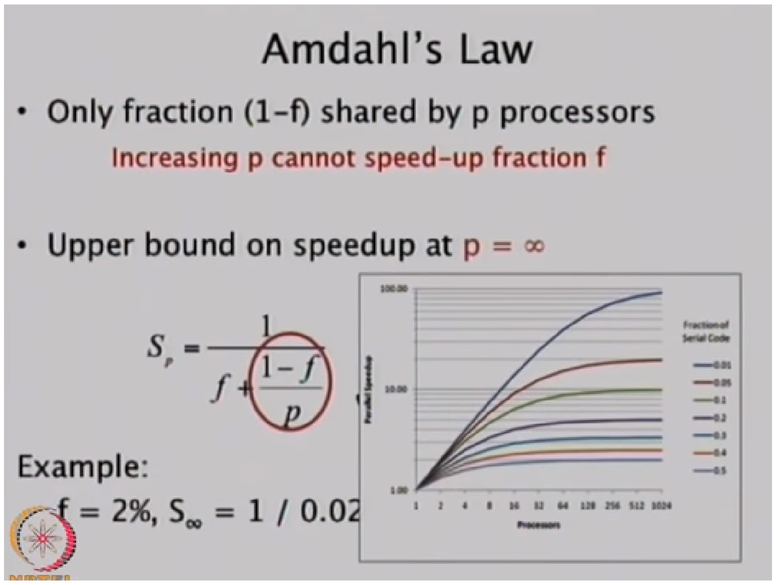
- Only fraction (1-f) shared by p processors
  Increasing p cannot speed-up fraction f
- Upper bound on speedup at $p = \infty$

$$S_p = \frac{1}{f + \frac{1-f}{p}}$$

Example:
$f = 2\%$, $S_\infty = 1 / 0.02 = 50$

So let us being we were talking about Amdahl's Law at the end of last class basically which says something very into it says that is that lots of things you want to do and something is going to slow down at what then over thing are limited in how fast you can be okay.

(Refer Slide Time: 00:59)



Amdahl's Law

- Only fraction (1-f) shared by p processors
  Increasing p cannot speed-up fraction f
- Upper bound on speedup at p = ∞

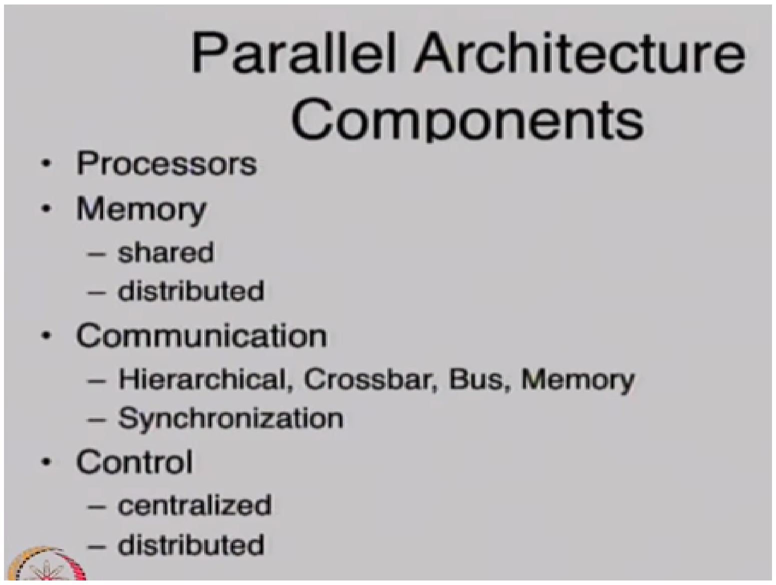
$$S_p = \frac{1}{f + \frac{1-f}{p}}$$

Example:
f = 2%, S_∞ = 1 / 0.02

100.00
10.00
1.00
Parallel Speedup
1 2 4 8 16 32 64 128 256 512 1024
Processors
Fraction of Serial Code
0.01
0.05
0.1
0.2
0.3
0.4
0.5

And we looked at couple of graphs or few graphs now what this was all basically part of introduction to some set and now moving into getting a sense of what parallel architecture is like today as well it has been like for the past few years in terms of high performance large scale parallel computers okay.

(Refer Slide Time: 01:24)



Parallel Architecture Components

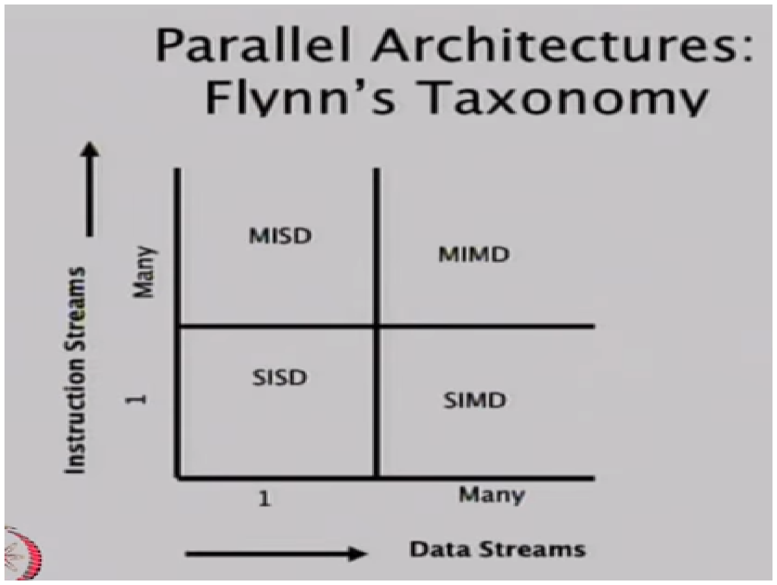
- Processors
- Memory
  - shared
  - distributed
- Communication
  - Hierarchical, Crossbar, Bus, Memory
  - Synchronization
- Control
  - centralized
  - distributed

So the components that we will have to understand and be familiar with comfortable with or of course they will have processor they will have memory either and usually both processor will have their own memory private memory as well as shared memory that every processor for it will some of the processor C and the communication between right. Sometime it each communication will be purely through memory you write data 2 memory you want to send it to some other processor and that processor reads the memory.

At other times you will be through some interconnect direct wire or some switch or some network that will take you from one processor to other. So all of that is the general components of these are also components of computation processor and memory are required as long as there is two things being only processor than memory there is going to be interconnect connecting otherwise this memory is not connected to processor then it cannot handle much okay.

And so there is nothing new where except you are going to be looking at this at slightly expanded level so to speak and then there is control and this is the many ways is different from sequential components what is the overall combined control of this many processor is there one central processor that says everybody execute one instruction now or is it completely distributed or just somewhere in between okay.

(Refer Slide Time: 03:16)



And one common way to classify this control is what is known as Flynn's Taxonomy where 4 classes in which Flynn's distribute divided parallel competent architecture is based on how the instruction operates on days in the context of many processors being used to operate this instructions and operate on sub data. So in this use basically easy to access one is how many different of data how many types of data how many different blocks of data modules of data whatever.

How many different items of data are being processed and on the vertical access you see instructions okay which is how many different instructions processed together concurrently may be in the same clock. So there are 4 categories that will automatically followed and there are listed as SISD which is single instructions single data any clock you execute one instruction or one piece of data meaning that in the instruction there are opponents they make it two opponents right you are may be adding two things we generate the third result okay.

So there are three opponents in some sense but they are not three pieces of data there is kind of together the data that relates to the instructions okay in that sense all the sequential computation is single instruction single data at one time you are executing one instruction or one piece of data okay the other is single instruction multiples of data meaning that your instructing the computer to do this but do this on whole array of things okay.

So again you can say add these to numbers to produce the results but is the same instruction at but there is an array of numbers and the first who have added to produce the first result. Second to have been added to produce the second result and so on okay this is data parallel same instruction lots of data being applied at the same clock are not is that not critical but let us just say same clock which means concurrently together we will study some architecture that uses the SIMD instruction style and the other which would be more generic would be NIMD.

At multiple instructions on multiple data meaning I have got a number of processors everybody has its own piece of data everybody running its own instructions own piece of data and that is the more generic type of architecture okay and then there is the fourth which is in fact the least used of all three. If at all was people generally do that more in research system and all on commercial ones which is multiple instruction single data MISD.

Basically it is one piece of data everybody is reading we are doing some other different things on that piece of data okay typically works in a pipeline sense that you are generating more results that you are going to be used by the earlier things in the pipeline but on the same piece of data okay. Multiple instruction single data but that you could so MSID more of the something that hidden in architectures but do not really see an architecture based on running multiple instruction on one data and then taking a next piece of data and then instructions on that piece of data okay.

So it is possible that once in a while you are going to do multiple things on same data but an architecture may be designed for that in which case you just say one piece of data and you do all those things but typically architectures are not designed for that meaning for take that piece of data and give it to that stage or unit and again the replica of piece of data are being given to another stage of another unit to do something else.

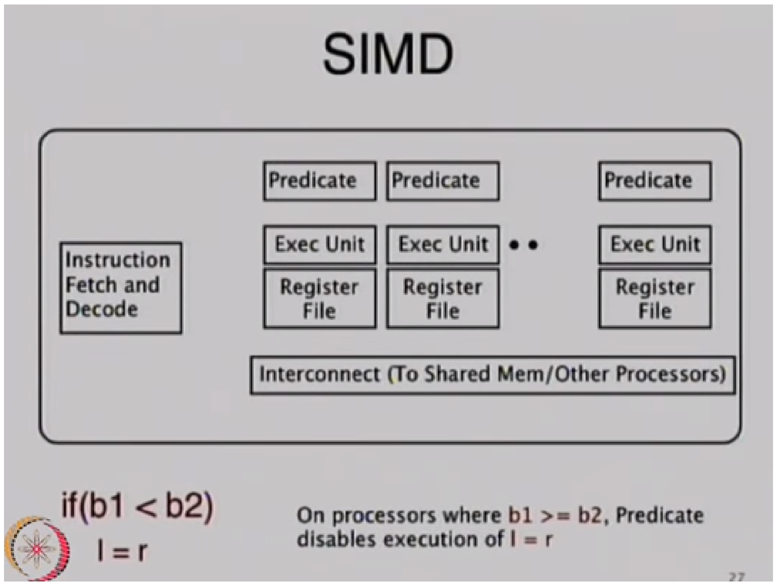
So it is not exactly multiple same piece of data so pipeline is hidden in all of this so once we get into some more detail which we will shortly these days there is nothing that is not pipeline okay so when I say that is why was hesitant able to say doing this in instruction on multiple data at the same clock because at the same clock is not very well defined in unless you set up a context very well defined in the context in pipeline right.

So you might say that I have got a pipeline and you are going to start instructions into the pipeline every clock and then you might mean to say every clock am going to start one instruction in many piece of data and then next clock I will start in next instruction or many pieces of data while the previous instruction is still going through the pipeline we will look in more detail at the SIMD MIMD these days in fact very start to look performance computers of today and the top 500 HPC computers you will that they are some mix of SIMD in it.

There are lots of SIMD units with respect to each other work in MIMD okay and so those to start sections are the once we will talk more detail about which are the once that are more popular SIMD is going to why not just do memory okay MD is most powerful you have the freedom to do multiple instructions to execute different instruction at the same time if you want for some reason your program to be in the same D fashion.

We are just make all the instruction are same of but the thing is that because you have you know that there is only one instruction all the data as going to be applied or is going to the instruction is going to be applied on the data you know that you are going to fetch on instruction need only one counter need basically you one control state of the machine which says that am going to fetch this instruction decode this instruction and then there are this multiple execution units okay.

(Refer Slide Time: 0:56)


SIMD
Instruction Fetch and Decode
Predicate
Predicate
Predicate
Exec Unit
Exec Unit
Exec Unit
Register File
Register File
Register File
Interconnect (To Shared Mem/Other Processors)
if(b1 < b2)
l = r
On processors where b1 >= b2, Predicate disables execution of l = r

Something which is little bit more clear in this diagram there would a bunch of execution units each being able to add or subtract or multiply some of the instructions says each having it is own registers and typically those registers are all be called R1 to RM so everybody knows it is own R1 and the instruction says add R1 and R2 so result in R1 so everybody has R1 notion of R1 everybody is going to take it is own notion of R2 or everybody is going to take R2 data added to the R1 okay.

So the things that are replicated that are register the files and the execution units but all of the control that goes well. As a result of which it is able to take less area as a result it going to process less it also it is more power efficient because it is less replicated control doing the same thing or even doing different things there is only one piece of data that doing.

However not every application is going to be able be fit with the single instruction running on to data and so there are trade us and that is you do not see this so there have been in this case the entire machine was SIMD. But in the machines of today you will SIMD units so that they there small components that they fit this model but then the entire application might not fit in this model.

So you break it into components that each fit well and then you provide each of these components to each SIMD unit one SIMD engine okay. The other thing that and because we will be doing programming on SIMD engines the important to understand right from the beginning is that every clock every execution unit is applying the same instruction and so if I have a program that has a condition says if this conditions happen then do that thing everybody is going to evaluate the condition on local piece of data and the instruction says evaluate this condition.

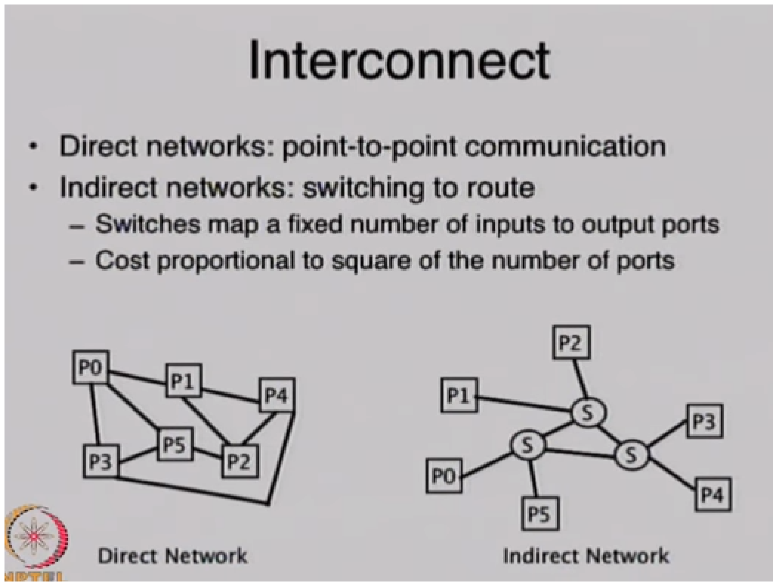
For example if we B1 is less than B2 then everybody as its B1, B2 pair and so some places B1 will be less than B2 some places B1 will not be less than and so only those places where B1 is less than B2 should L be assigned the value of R and in every SIMD engine there will be some kind of a conditional disable of this execution units. So when somebody says everybody doing this instruction in parallel that means every active processor is doing this construction architect.

The other inactive ones just do not take it cannot execute some other instruction it just do not execute that also means if I have not if this is true to do A otherwise else do B what happens so some places it will be true some other processors it will be false and so wherever is false becomes inactive and everybody executes inactive of A and then everybody executed the instruction of V with the first set being inactive and second set becoming active okay.

So although it would be that A had 10 instructions B had N instructions and every processor is running either A or B every processor is running 10 instructions you are going to take multi clocks right because when wherever the condition was true inset of instruction so which is A is happening the other processors are ideal okay so SIMD programming one are the more important things to consider is how to make SIMD be somewhat condition be and often if there is a condition try to organize this SIMD into the routes of work units where the conditions are likely to evaluate to either or true or all clocks.

Is that is nobody is setting it to false you still have to spend that clocks with nobody is doing anything that optimization is three so if nobody is active then you do not have to wait alright.

(Refer Slide Time: 16:03)



Interconnect

- Direct networks: point-to-point communication
- Indirect networks: switching to route
  - Switches map a fixed number of inputs to output ports
  - Cost proportional to square of the number of ports

So that is the control structure let us get a little into the interconnection so what does the instruction look like there is in our code and there is an opponent code okay. So the opponent code says register 1 register, 2, register 3 and the (()) (16:30) okay. So everybody has a local notion of R1, R2, R3 in which different piece of data sits okay the other thing that as to be there in SIMD machine is when you load something from memory then you have to have an index slope so you say load from these address plus some offset and my offset is 0 this guys offset is 1 that guy offset is 15.

So they are all loading different pieces of data into their own respective R locks memory as well as processors okay. So we are going to talk about generic levels so we are all on the same page in terms of the terminology in terms of understanding the basic organization of the parallel computer. That depends right if there is some full cross bar where N people can talk to N things together then it can be done there are technologies where that is expensive and people do not implement large prospers for the reason but we will talk about it.

So there is an offset register then all they will be having offsets like you say here is the base address and here is the offset register then the offset register will have the data which may be completely different from others so it is not necessarily true they are been reading from neighboring or contiguous chunks of memory okay. And then there are several hidden details that depend on the atom size how much have data you read from the memory gap line size there is lots of detail that we are not getting into at the moment.

That is built into the architecture that so there is instruction sets where you are physically set we can provide update index or a bit array of these conditions which says your active okay. But the architecture execution unit will automatically allow you to set somebody inactive. So now the compiler says if this evaluates to true set yourself they are active and then do this and then toggle your active bit and then do this okay.

Compiler only inserts this instructions this is not known at the compile time okay the instruction says that if this evaluates true sets true active okay. And so there are instructions where you say load and address and register okay where some multiple time they register multiple typically the word with is added to the base address to get the final address to where to load. So everybody is the same instructions so an instruction might look like this where it says load into R0 that content of this base address plus the context of R1 times 4 okay.

Now everybody has n R0 everybody sees the base address there is part of the instructions this instruction is the same everybody has an R1 content of R1 is different for different because in the beginning you set R1 to 0 okay. Everybody loaded base address plus their index so somebody loaded let us say R1 was = 0 everybody has loaded that address +0 +1 +2 +3 all the way up to processor number 15 lets say.

So their loaded from the different pieces of memory so they have different pieces of data in R0 okay and now you again say load R to R0. Now R0 has different pieces of data for different processors and so on to will loaded from desperate location no all of them even so this is the base right you are right this is the base handles R0 times some strip is going to be added right plus in fact I can put it right here plus the processor level what do you been by special control?

SIMD machines has control unit and execution unit right so by the ay since this is also a topic we are going to move on to interconnect we are going to officially take the break now continue this discussion and then move on to interconnect after the break okay it can be typically SIMD instruction said will be more or less X86 may be few things that are hard to do taken away but a few things added just to support the SIMD type because there will not be get my processor ID.

So there will typically be some instruction like since I have already used R0 let me already make it R2 or R3 or R8. Some processor or some instructions set may that R0 is the processor ID okay so it is hard so whenever you read R0 you will get 0, 1, 2 or 3 and then if you want something to depend on your processor ID you will simply read R0 into some other likes I just not a real register here is some hard set of things and then you put that R0 into another register and then use it.

So reading R0 means getting processor ID would not make sense to (()) (24:14) things are that sort could be added into SIMD. Memory lay out has nothing to do the data has been organized in this particular formula that is up to you right. You read from the place while you write not the different way each one is basically saying I want to read from some instruction no this is part of instruction right so this is not different it is the offset that is different in fact I am probably over extending by saying next some processor number is going to get added right.

Typically a SIMD designer will dispense that by saying you add R0 your processor or R8 before you do this R8 otherwise we will already decide. Offset computation is not individual it is an individual that is determining the offset and way it determine the offset is reading the content of its R8 multiplying by the known site and adding the base which is part of the instruction set R1 to 0 here no then it would not added.

If you set move R1, R0 where R0 is understood to be processor ID then they will doing map reduce have big element of SIMD yes but then that is more of a program model underline you may or may not have a SIMD. I am assuming here it is shared memory right if it is a private memory to processors then everybody will have their own copy and it would make sense even when you are reading the same thing they are actually reading different memory items.

Of course it is sometimes make sense to read the same things because you want to broadcast some every processor so everybody is reading from same processor that is reasonable to do. So now we are going to move on to talking about interconnect the types of interconnect that are have been or in common usage in general will divide them into two categories one is the direct connection where you have a port to which a wire is connected and that wire directly leads to another port to which another processor is connected so it is your connectivity to that processor or you are not okay.

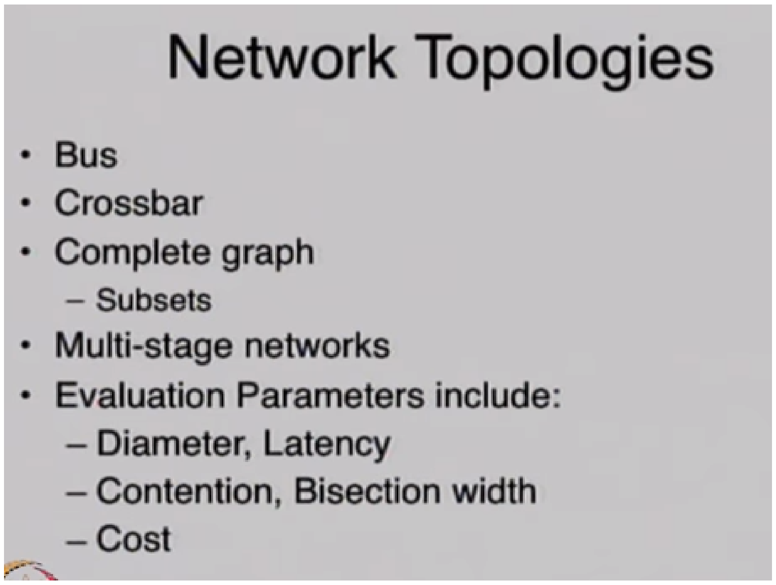
So the picture the left hand side you see P0 connected to P1 and P5 is not connected to P2 that means Po cannot send it P2. Unless there is some protocol where P5 can be get from P0 and decide that it is to be forwarded and so on. As far as instruction set is concern you got three addresses to which you can send a piece of data alright. So that is a direct connection P here is stands for processor you can also sync of it has another memory okay which I had not drawn.

Meaning that is you their instruction this typically has much wider instructions set the next set this type architecture where you will say send so many bytes from this address to processor number 6 or your port number 3 yes right. Memory may also set memory that is addressable as a node may also set somewhere okay and then you have to say protocol must exist for you to just say send this piece of data to that processor but since this piece of data to get that processor to the stored at this location.

So now that is part of architecture right now be some buffering involved which will take that data from the program application and will add will convenient time to send the data clause okay and once we get to programming model we have to look at we will look at the different issues involves with when you send it how do you know it has received and so on hence so forth.

The other on the right hand side is the indirect network they are non-recipients so to speak in the middle which is where you I want to send some piece of information to some address and the switches will determine you are connected to one switch may be even more than one and you sent to one of your switch will determine using some routing protocol how to get the data right so the internet is not direct network and in terms of the different typology and then the typology is will they links themselves may be one of various kinds.

(Refer Slide Time: 30:36)



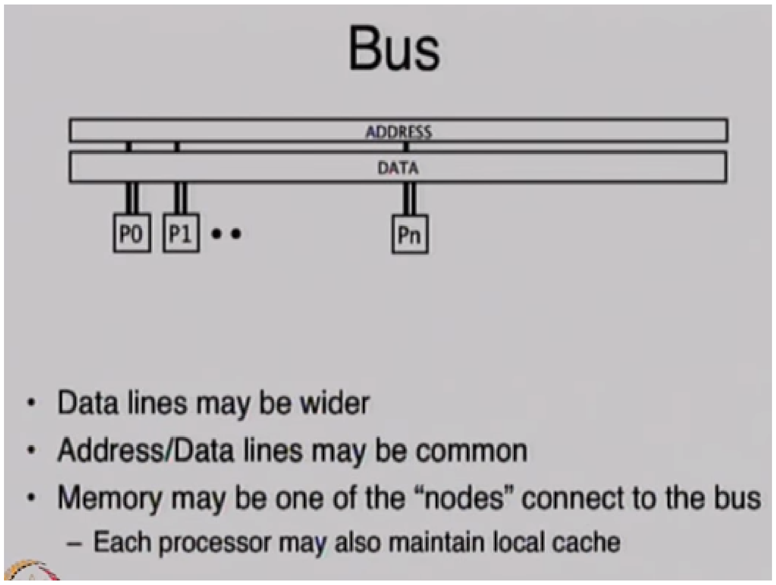
Network Topologies

- Bus
- Crossbar
- Complete graph
  - Subsets
- Multi-stage networks
- Evaluation Parameters include:
  - Diameter, Latency
  - Contention, Bisection width
  - Cost

So typology is in general will be bus which is for example Ethernet was lots of notes have bus is going to have lots of notes and different notes will look in that bus okay normally in the parallel computer context there will be address bus as well as the data bus put address on to the address bus and the data consult from the data bus or you put the data to the data bus and addressing to the data address bus.

So that this data gets to that place to the address but it is not necessary in Ethernet there is no separate address bus and separate data so separate bus on which you say here is the address here is one data and here is the block and people figure out who it is for what the data is.

(Refer Slide Time: 31:38)



Bus

- Data lines may be wider
- Address/Data lines may be common
- Memory may be one of the “nodes” connect to the bus
  - Each processor may also maintain local cache

So that is a bus interconnect typically data lines would be much rider that than the address lines and the address and data lines can be common as we were just discussing for example in Ethernet and on the same bus may sit on memory loss. So you can send the address and the address may point in some memory and data will come out from that memory so some memory controller sitting on top of the memory deciding how to suck memory pins values and how to read values out of memory.

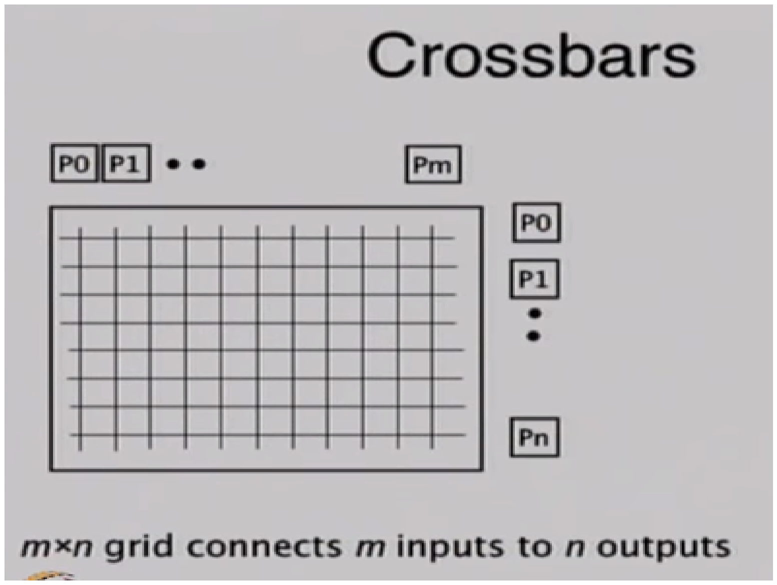
It is not important to what model is okay both models will typically have some interconnect even in SIMD model in fact if we go back to SIMD model picture see something that says interconnect to shared memory as well as other processors may be a bus may be direct connection may be a architecture that use one or the other in some architecture there may be one architecture to other processors but they will be an interconnect to a memory which everybody is going to share.

So interconnect is going to independent of the organization of the control alright the buses would of course easy to implement right imagine having an long bus with lots of different ports and you add a processor simply hooks as long as port available you can simply plug in that the problem is because of the protocol that the buses need to run they do not scale very well especially in terms of performance beyond something like does not course does not processing units right.

And Intel architecture uses buses to connect processors to memory and so they have not quite scale beyond it or processors in a single chip. Be other extreme the bus is an extreme in the sense everybody is connected to a common stratum okay and whenever you want to send the data to anybody else you say here you take I have one port you send it to this port and other addressee will read it okay and there are conflict issues contention issues.

So one of the reasons it is not very scalable is contention lots of people trying to use the same bus and trying to put the data at the same time some people will have to back up of an then this repeat back of repeat back off costing you lot of.

(Refer Slide Time: 35:00)



Crossbars

m×*n* grid connects *m* inputs to *n* outputs

The other extreme is the crossbars and basically cross bar as P input port rather N input ports and N output ports and anyone of these N can be connected to any one of these N so basically in the internal is actually some kind of a cross wires someone vertical one set of wires in one way other set of wires is another way with some switch at the junction which either connect those wires or keeps them apart alright.

And so if you want to send data from certain row to certain column simply turn back switch that means that will connect this wire to that wire. So whatever this processors put on that wire is available on that wire okay. I have shown in this picture P0 to Pm going at the top horizontally and P0 to Pn going at the bottom going on side typically is the same P0 to Pn okay same set of processors which are also on the top side or also on the horizontal side which typically mean that input and output are typically all together and the horizontal wire will have connections.

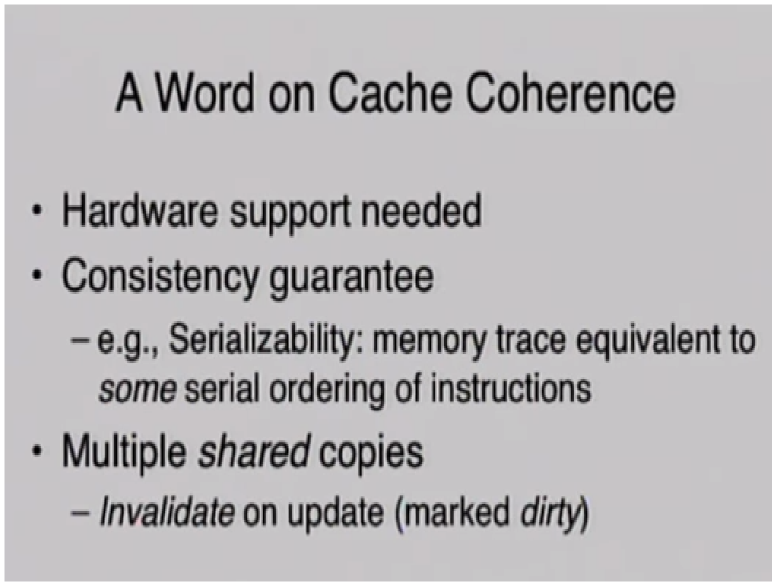
So that the ports can be made pins can be made on one side what do you think about scalability at the cross bar lots of power right anybody talk to anybody quickly no interference from other people unless of course two people want to talk to the same guy and that guy listen to only on at a time but those are minor issues compared to the same stratum being used for everybody. So cost is the performance is going to be reasonable but the cost is going to grow quickly okay.

Is typically M times N cross bar is also one extreme and people might implement cross bars but would normally do it in small groups sub sets. So this sub sets is everybody connected to everybody this subset is everybody connected to everybody and then there are some ways to communicate from this subset to that subset okay. So that is the hierarchy the its all and mainly talking about the manufacturing cost right but power is a big input into the manufacturer cost.

So cost is high one of the reason cost is high is that its take lots of that is not the only reason but there is co-relation and so typically there would be some intermediate things like multi stage. So you may have several stages of you may have to go through this subset can have direct connections from everyone to everybody that is the complete graph but then you cannot talk to anybody in sub set there will be some representative of that sub set and data to that and then that can replace the destination to anybody of its neighbor and from there you can go to the next neighborhood.

So there is a multistage networks are sometimes used will talk about some other also more specialized sub sets which lies between the extremes when we talk about the examples of architecture the other side when you do not have communication networks directly and when you do have communication networks is the shared memory side. So that memory it is either connected through some interconnect to the various processors well it is actually connected to through some interconnect to the various processors.

(Refer Slide Time: 39:30)



A Word on Cache Coherence

- Hardware support needed
- Consistency guarantee
  - e.g., Serializability: memory trace equivalent to *some* serial ordering of instructions
- Multiple *shared* copies
  - *Invalidate* on update (marked *dirty*)

It may not be at various processor compare to each other and the same interconnect but memory is going have some either a bus which should make sense because all the processors are only copy two memory not to each other so contention is reduced. So we have to again understand the there is this need for caches on the local memory because you do not want to read data from the memory across this interconnect every time you need it going to read it chunks read it in cache and reuse that data from the cache okay.

Everybody was done architecture would have seen that the grade details whenever you have the multiple copies of piece of data you need to make them consistent. So the cache coherence is at the program level that assume will exist right but the hardware level somebody have to spend the effort to provide cache coherence. Cache coherence means that even though there is multiple copies of same data their consistent with respect to each other and this is where the various models of memory consistency coming together.

And one of the models is serial this is the most basic model says that when you look at the profile that any memory item is going through any location in memory is going through it must match the profile that the sequential memory would go through where you take any sequential equation. When you take this different set of instruction that the different processor are executing and execute them in some order okay.

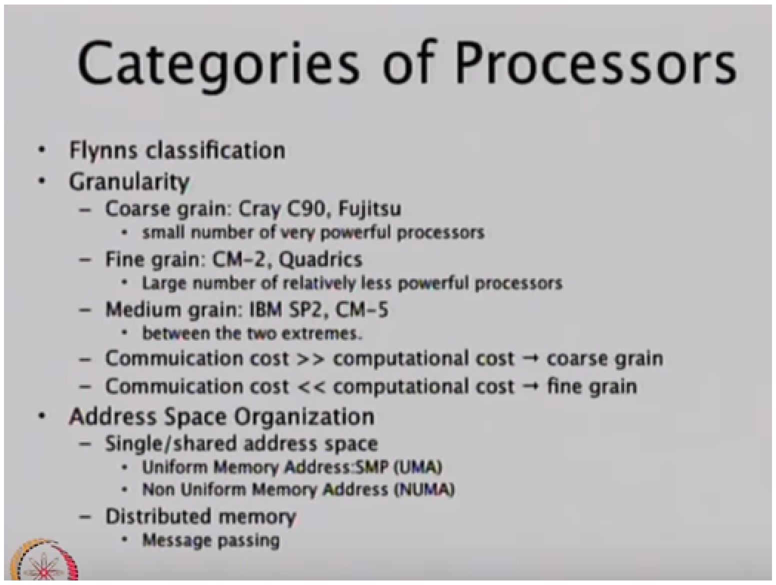
So you take the 10 instructions for processor 1, 20 instruction for processor 2, 30 instruction for processor 3, 2 instruction for processor 4 and execute them internally in anywhere you cannot re order the instruction of any given processor but between any two processor these 20 be anywhere among these 10. As long as one serializations produces the same defect on memory you are memory effect is these.

And one of their minimum is required this not sufficient is required for all programs will be able to guarantee serialization is that there will be cache coherence bent you have the same piece of data that two locations that one of them as to updated these piece of data cannot be given to that processor okay. Because if these piece of data is old and some new data has come in here and these instruction that you are going to execute later on which depends on something that these instruction has done which means it depends on instruction come after that in any order.

In any serialization if it comes after that it cannot have seen the old value okay it has to see the related value that this processor make and so there are various protocols to guarantee we are not going to talk about those in any details various protocols that guarantee cache coherence but at high level you either have to whenever you update your local copy you have to make sure that every copies are up to date and if to do in a fashion that they cannot read the old value right.

Or when you update your copy you in validate your copy and again with the guarantee they cannot read the guarantee okay once they are invalidated whenever they have to read they read it from you rather than through the memory okay and there is lots of difference detail because you are only ordering instructions alright so with those at the high level concepts that is going to architecture.

(Refer Slide Time: 44:24)



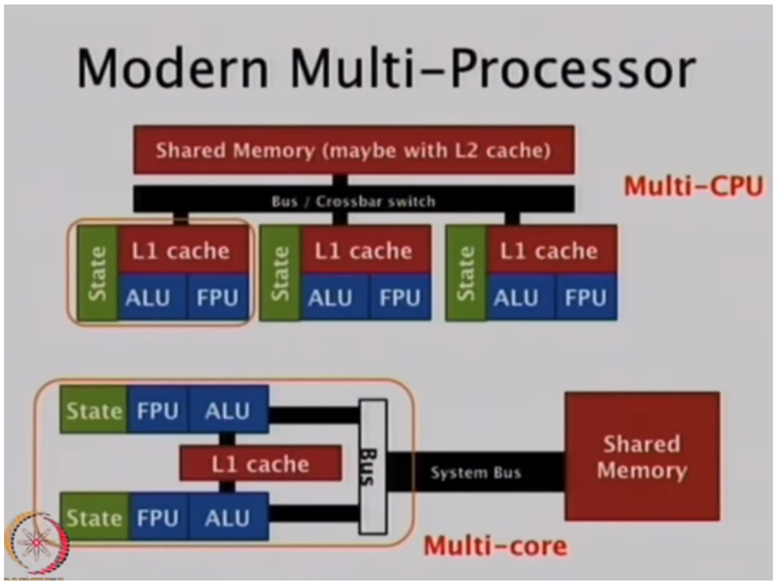
Categories of Processors

- Flynns classification
- Granularity
  - Coarse grain: Cray C90, Fujitsu
    - small number of very powerful processors
  - Fine grain: CM-2, Quadrics
    - Large number of relatively less powerful processors
  - Medium grain: IBM SP2, CM-5
    - between the two extremes.
  - Commuication cost >> computational cost → coarse grain
  - Commuication cost << computational cost → fine grain
- Address Space Organization
  - Single/shared address space
    - Uniform Memory Address:SMP (UMA)
    - Non Uniform Memory Address (NUMA)
  - Distributed memory
    - Message passing

Let us move on to the history of parallel computers and will look at several of these axis Flynn's classification SIMD or MIMD or a combinations what is the size of the processor what is the granularity of the processing elements there may be lots of weak processors or a very few strong processors very powerful processors their address space organization and whether there are shared memory and receive it.

(Refer Slide Time: 45:02)

Modern Multi-Processor



And let us begin with the PC of today it is we will look at some numbers later on but it is the super computer from not long ago it is more powerful then super computer run off long ago. And it is a parallel computer okay these days you cannot but a single core machine anymore and so PC's are parallel computers they are all of these notion they will be either SIMD or MIMD or some combination there will have some interconnect among the processor among the processing the memory they may have shared memory and so on hence so forth.

In this case they actually do have the shared memory architecture and hence two pictures are multi CPU on the top separate CPU course separate CPU units chips and the bottom size is multiple course in the same CPU and then really at the low level there is only difference is a little bit of organization right everybody is still connected to the shared memory the controller to the shared memory talks to one unit in the multi core multiple CPU there is some single controller that is providing data all the course.

On the other side there is this cross bars through which all this different CPU's addressing and talking to the memory and so they are have their own independent units you will be able to access a piece of data you able to put this on the address bar and reading the data that is one important the control of the memory one case there is essentially one to one relationship with the CPU and the memory.

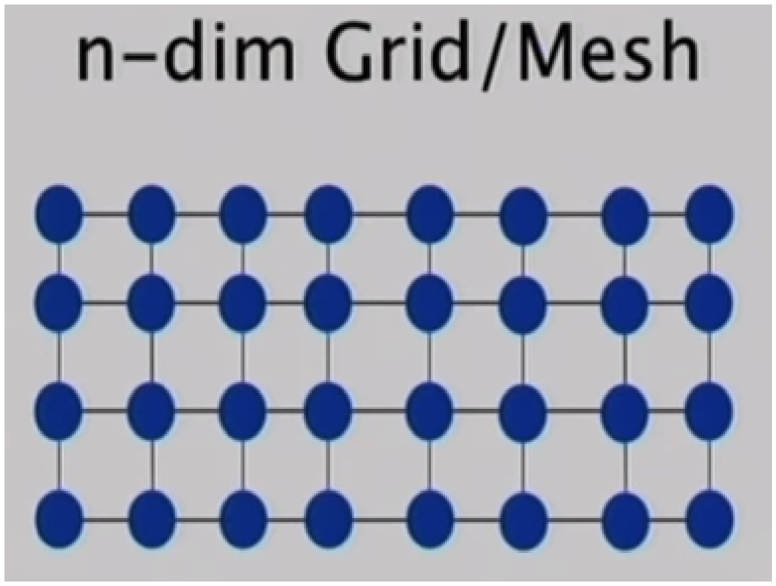
CPU directly fetches it from the port near the case there are multiple CPU's connected to the memory we are doing arbitration. So in case it is one consumer which is the and one producer which is the memory right and there is direct connection and it is on the memory on the consumer side that there is some arbitration that whether this core wanted data or this core wanted this data okay.

But as for the memory is concerned it is one stream of data and one memory slash memory controller memory always sees one address in one data. But the memory controller sees one connected to it but on the multi CPU units the memory controller sees multiple consumers and it says you want this piece of data you want this piece of data you want that piece of data am going to determine who gets it first okay.

And the other is the shared of cache in the multi core there is one cache that so memory is feeding into the cache and then that cache is spreading out the results to whoever wants it and then these two are not two different separation it is really same separation because there is one fetcher from the CPU point of view for the memory in one case and that fetcher for us to feed it to one cache and there are fetches each fetching independent streets of data somehow being arbitrated by memory controller and all this fetches get their data and put it in the effective caches okay.

The bus is the architecture as I earlier mentioned of use because typically there are small number of processors okay.

(Refer Slide Time: 49:23)

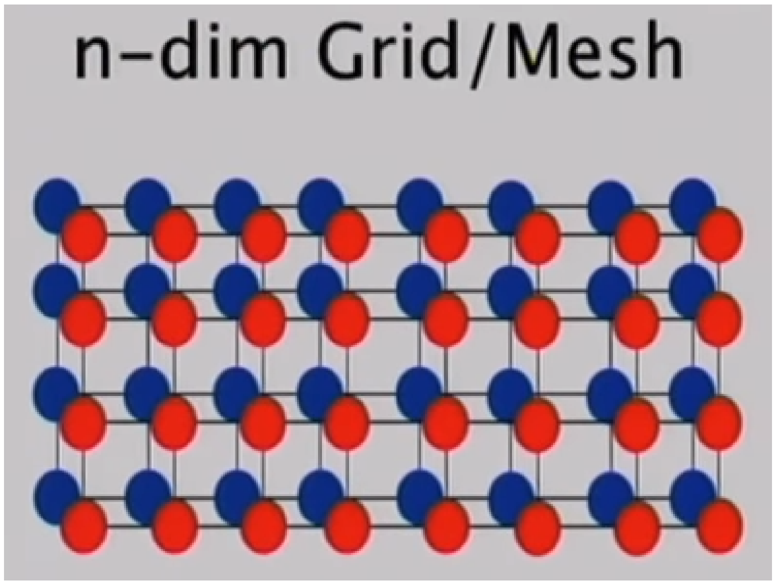


n-dim Grid/Mesh

One of the earliest parallel computers had this direct connection direct network okay and this example of what is often called mesh network or the grid network where you are connected to two processor on two side and the two dim mesh and two processor on the vertical side. Three dim mesh are also been found the third access also been used and processors are also layed out in the fashion.

So that these connection are small the wires are small okay mash power is one example of such a machine we will talk about shortly.

(Refer Slide Time: 50:09)

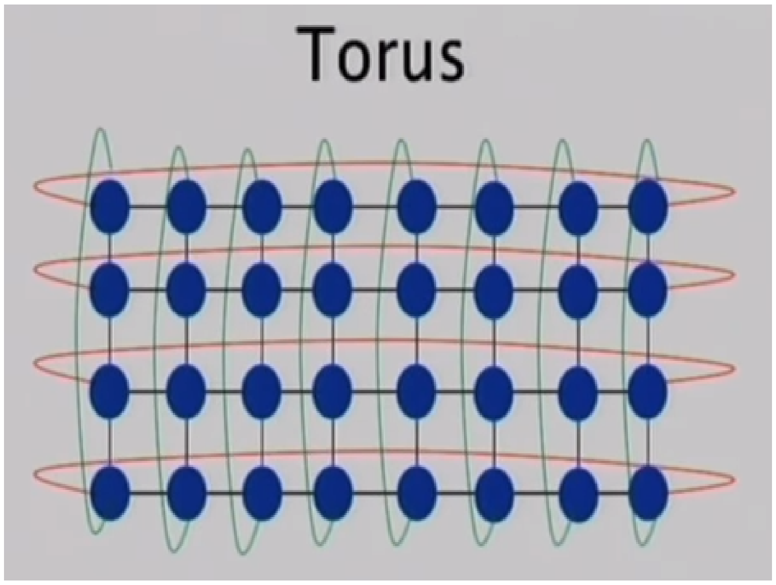


n-dim Grid/Mesh

So this is 3D mesh in some implementation of mesh then you also have some additional wires so instead of just connecting it along the axis you also connecting to the diagonal there is another example of that where you connected diagonally. And so now and although there were other example likes CM which had a mesh connection mash power which is SIMD machine right.

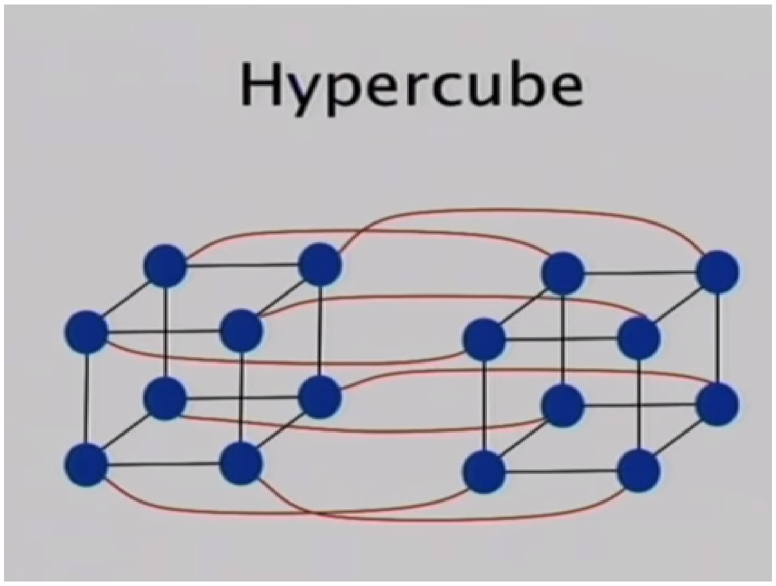
So everybody was executing the same thing at a given time and they could only talk to neighbors. So either they are all adding things adding subtracting whatever or they were saying send to this data to the right hand level okay and it is little asymmetric think about it right if everybody says send the data to my right hand labor what happens the last guy either burns out or as some way to inactive self right.

(Refer Slide Time: 51:33)



So it is not uncommon to have the torus connection where the last have a connection to the first torus so the last guy is right hand labor is the guy on that side okay. And similarly on this side the extension to this the notion of hyper cube connection.

(Refer Slide Time: 52:02)



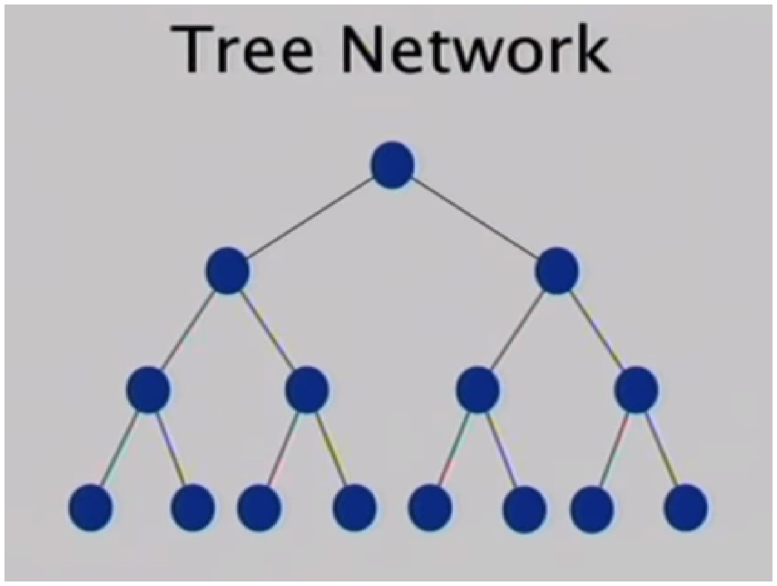
Which is again somewhere between taking is closer to full cross bar connection than bus but at the same time not incurring the entire cross and hypercube connection is also in some ways generalization of the mesh connection. So hyper cube for two nodes is simply a wire connecting right hypercube F4 nodes is wires connecting them in sub set of a grid a mesh.

But three nodes rather so event from 2 to 4 should go to 8 and these nodes size is typically to go up in orders of power of two you get it in the third dimension it is so another 4 and you connect them like this if you want more processors for example 16 you make another cube you connect them one at a time. You want another 32 you make another copy of 16 and then make 1 to 1 connections so how many connector will each node have log of N nodes each connector will log of them okay.

And some of the things we look for when you making connections are how far are the nodes any given node so the distance between two further nodes is called the parameter of the net meaning that it is going to take that many up's to get data from one of the nodes to the other possible. If lots of people are wanting to talk to lots of other people is that of course not right in fact you have already here you have some hoax of distance once some hoax of distance two that distance is typically symmetric or at least in terms of performance evaluation we consider to the symmetric.

The length of wires will not be the same it depends on the very complex layout pattern right but it is not going to effect the speed of computation by lot. Because signal travels very close to speed so making that 1 centimeter extra is not going to make different it's the buffers and delays that are cause because of the control structure that makes the difference so and for all practical purposes these are one cost hoax or uniform single cost of hoax okay.

(Refer Slide Time: 55:39)



The next one would be the tree network by the way I was telling you about the parameters for interconnect one is of course the diameters what is the maximum time might take from node A communicating to node B. The other is the separation it is also called the bisection cost how many wires would you have to cut to separate it into two units meaning that now these set of processors have no power to that set of processors.

Am not going to get into models of failure but that failure is the reason of cost is often used okay but failures of failure models are not necessarily atomic okay. Before we talk about the tree let us take another break.