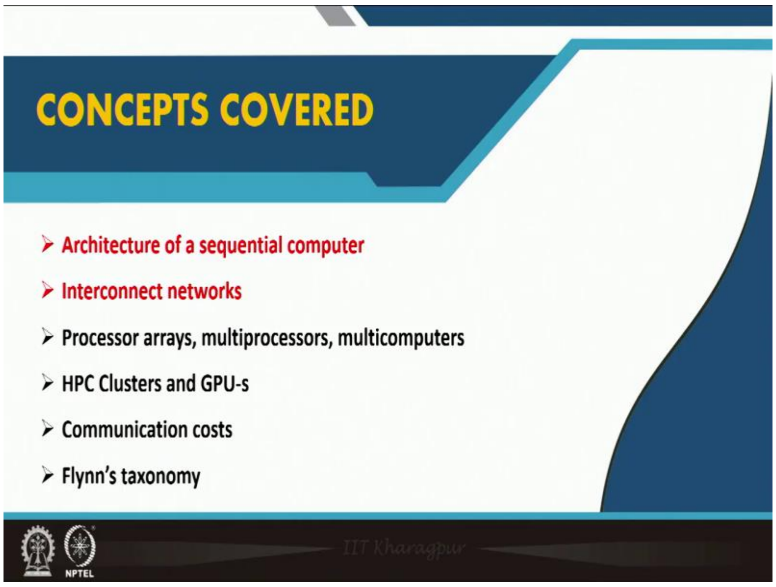
High Performance Computing for Scientists and Engineers

Prof. Somnath Roy
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Module - 01
Fundamentals of Parallel Computing
Lecture - 03
Architecture for Parallel Computing (continued)

Welcome to our MOOC course on High Performance Computing for Scientists and Engineers and we discussing Architecture for Parallel Computing which is part of the first module of the course which is Fundamentals of Parallel Computing, we are continuing the discussion from the last lecture.

(Refer Slide Time: 00:43)



CONCEPTS COVERED

- Architecture of a sequential computer
- Interconnect networks
- Processor arrays, multiprocessors, multicomputers
- HPC Clusters and GPU-s
- Communication costs
- Flynn's taxonomy

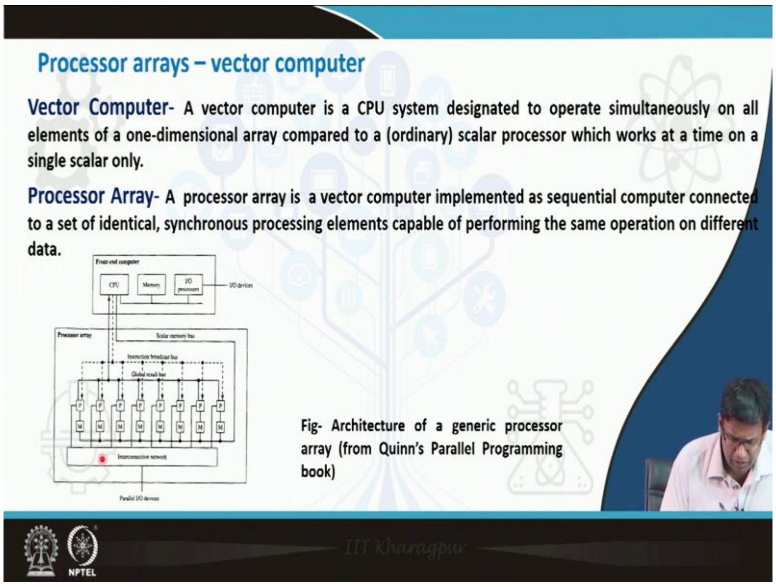
IIT Kharagpur

We discussed about architecture of a sequential computer, we discussed about interconnect networks; that means, how are different computers connected in both shared memory and distributed memorial arrangements. And we have also quickly looked into couple of terms like cache coherent protocols, false sharing and communication costs.

And next we will look into some of the examples of parallel computing architectures which are processor arrays, multiprocessors and multicomputer .Then we will see some of the examples of practically working parallel computing large infrastructures which are

HPC clusters and will also look into GPU-s, will try to put some more detailed discussion on communication costs and then discuss about Flynn’s taxonomy.

(Refer Slide Time: 01:41)



Processor arrays – vector computer

Vector Computer- A vector computer is a CPU system designated to operate simultaneously on all elements of a one-dimensional array compared to a (ordinary) scalar processor which works at a time on a single scalar only.

Processor Array- A processor array is a vector computer implemented as sequential computer connected to a set of identical, synchronous processing elements capable of performing the same operation on different data.

Fig- Architecture of a generic processor array (from Quinn’s Parallel Programming book)

IIT Kharagpur

Processor arrays are the most initial days development in parallel computing and one of the examples are the vector computers.

A vector computer is a CPU system designated to operate simultaneously on all elements of one-dimensional array, compared to an ordinary processor which works on a single scalar only. So, if I write s is equal to b plus c sequential computer will consider that a is a variable, b is another single valued variable, c is another single valued variable and add b and c and get a value.

If I use a vector computer then a is a vector A is an array or a vector, A has said 100 elements, similarly b and c have 100 elements. So, the same operation will be processed over all the 100 elements of b and c, there will be multiple computers which are doing same operation on different elements of a vector. So, instead of processing one by one of different elements on the vector, all elements of the vector will be worked on a go.

An example of vector computer is a processor array which is a vector computer implemented as sequential computer, which is connected to set of identical synchronous processing elements capable of performing same operation on different data. That means, it is the same operation which will be done like a vector operation adding two vectors.

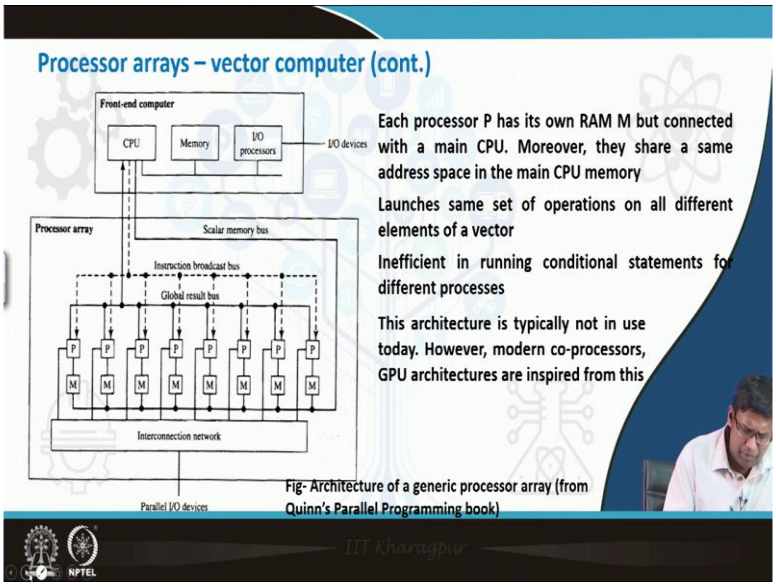
So, same operation addition will be done on all the respective elements of these two vectors and this will be done by different processors, or different processing elements which are synchronized and are designated to do that.

So, a vector computer again while saying. So, I will show you also later, but we can tell that vector computers are not very much of use today because they have some primitivity in their operation, they cannot work on complex algorithms. So, we use something else now.

But let us see what is in a vector computer, there is a CPU which has its own memory and I/O devices. Now this CPU is connected to a scalar bus and then to an interconnect network switch and this interconnect network switch further connects different processors which has their own small memory units.

And they are again connected to a global result bus and this result bus is connected to the CPU that means, scalar memory is one by one goes from the CPU and resides in the memories of different vector processors present in the processor array. And these processor arrays have different CPUs and each CPU is connected with their small memory part. So, this is the scalar memory bus is connected to the memories of different processors and each processor is also connected with the interconnection bus and the instruction goes from the CPU directly to each of this processor. They take the small amount of memory associated with that, work on them and the result goes to a global bus which returns to the CPU. This architecture is taken from Quinn’s parallel programming book.

(Refer Slide Time: 05:31)



Processor arrays – vector computer (cont.)

Each processor P has its own RAM M but connected with a main CPU. Moreover, they share a same address space in the main CPU memory

Launches same set of operations on all different elements of a vector

Inefficient in running conditional statements for different processes

This architecture is typically not in use today. However, modern co-processors, GPU architectures are inspired from this

Fig- Architecture of a generic processor array (from Quinn's Parallel Programming book)

If we try to see in more detail, that there are multiple processors processing elements each has their own small memory. And if we have a large vector, parts of the vector or elements of the vector are residing on this memory and this is the same instruction which comes from the CPU, is replicated to all the processor. This processor takes the memory associated with that processes the information, returns it to the global result to the CPU.

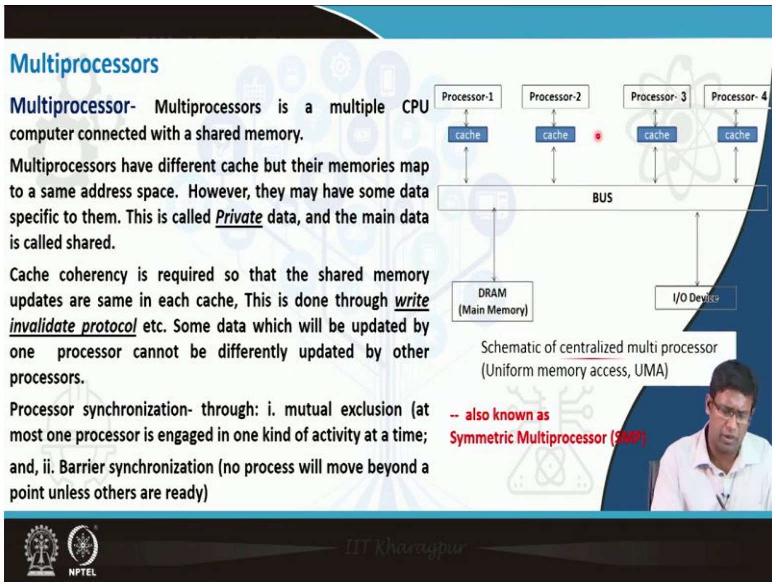
Each processor P has its own RAM M, but connected with a main CPU. Moreover, they shared the same address space which is in the CPU memory. So, all these M small memories are some way mirroring some part of the memory in the main address space.

The vector processors main CPU launches same set of operations through different vector processing parallel units to the processor arrays to different elements of the vector. It is the same set of operation which must go to all the processors.

This is inefficient in running conditional statement for different processors. If different processors need to do something different then we use some if else statement to make that. This is inefficient because these processors do not have well develop control unit also, they are supposed to work on the instruction which is given by the main CPU to them. So, if there are some complexity in between them, if there are some logical if else type of logical reasoning statement in between them, it is an inefficient system it cannot work properly there.

This architecture is typically not in use today. However, some of the modern coprocessors like Intel Xeon 5 or GPU architectures are inspired from this, that there are also many processing units which takes job from the instruction from a main CPU and they do essentially similar works on different parts of the data.

(Refer Slide Time: 07:53)


Processor-1
Processor-2
Processor- 3
Processor- 4
cache
cache
cache
cache
BUS
DRAM (Main Memory)
I/O Device
IIT Kharagpur
NPTEL

Multiprocessors

Multiprocessor- Multiprocessors is a multiple CPU computer connected with a shared memory.

Multiprocessors have different cache but their memories map to a same address space. However, they may have some data specific to them. This is called *Private* data, and the main data is called shared.

Cache coherency is required so that the shared memory updates are same in each cache, This is done through *write invalidate protocol* etc. Some data which will be updated by one processor cannot be differently updated by other processors.

Processor synchronization- through: i. mutual exclusion (at most one processor is engaged in one kind of activity at a time; and, ii. Barrier synchronization (no process will move beyond a point unless others are ready)

Schematic of centralized multi processor (Uniform memory access, UMA)

-- also known as Symmetric Multiprocessor (SMP)

So, I said that these are not typically in use today. So, what is in use today for that we need to look into multiprocessors. Multiprocessor is a multiple CPU computer connected with a shared memory. So, there is a bus or an interconnect network switch which is connected to a shared memory, and this memory can physically come in piece pieces, but virtually or in practice this is a same set of contiguous memory which is visible to all the processors they have their own caches. Multiprocessor have different caches, but their memories map to the same address space, even if some memory goes to cache of processor 1 and the same part of the memory goes to cache processor 2, they are must be mapped to the same address space.

However, it is also possible to give some data very specific to each of the processor and this is called private data. We will look into private data when we will see threaded parallelization and especially open MP program, but the main data is shared. So, the main data you can think of a main amount of money in a joint account, visible to everybody but everybody might can have some amount of cash on their own money back.

So, there is also some flexibility to maintain some private data which is not visible to all the processor, processor 1 private data is separate from processor 2. But the main data is in this memory visible uniformly to all the processors and this is again called a centralized multiprocessor or Uniform Memory Access multiprocessor or UMA.

Now, if one part of this data which is residing in its cache is updated, it must be reflected in the main memory, because this data might also be shared by the second processor and if it is not reflected to the second processors cache, then there will be a conflict.

So, say the variable a is called by first processor and because they are looking into the same address space, processor 2 is also looking into and if processor 1 updates variable a from processors 1th cache it should come to the main memory and must also be reflected to processor 2s cache otherwise there will be a cache inconsistency.

So, cache coherency is required so, that shared memory updates are same in each cache, and this is done through write invalidate protocol. That means, if processor 1 is writing changing some value in array a, then processor 2 3 4 are cannot do that. If they try to write same on the same array it will be invalidated. So, some data which is updated by one processor cannot be differently updated by other processors.

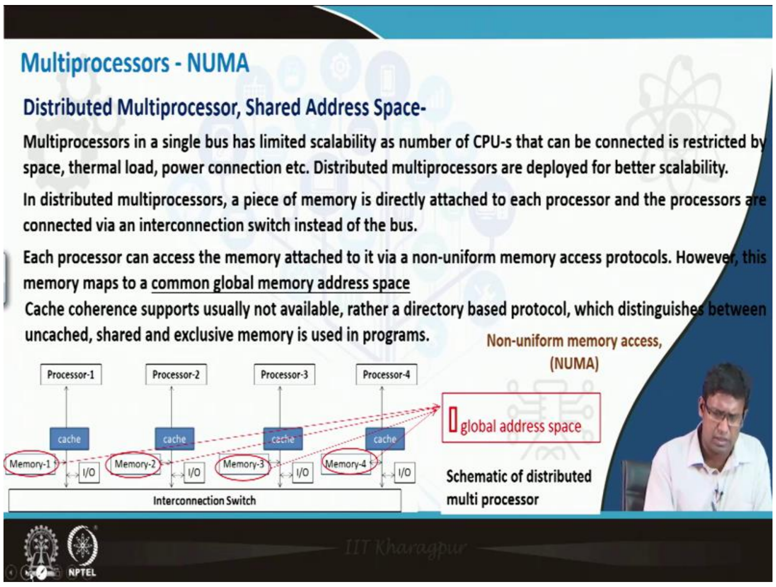
And also, sometime processor synchronization is required, for that, which is at most if one processor is doing some activity and many processors do are designed to do the similar activity, but it is mutually excluded, at most one processor is engaged on that activity. So, if there is one variable all the processors are trying to update, then the others will not be allowed to do that and at most one processor will be allowed to. The three will be some sequentially across the processors and this is called a synchronization.

And barrier synchronization is that no process will move beyond a certain point unless all the processors have done that. So, even if one processor has finished one work, but other processors are also asked to do that work and there is a barrier synchronization put there, this processor which has finished the work cannot take the next instruction set till other processors finished up to that particular set, these synchronizations are often given in multiprocessors.

This is known as a Uniform Memory Access or UMA centralized multiprocessor is also known as UMA or Uniform Multiprocessor. There are also some many very commonly referred to as Symmetric Multiprocessors or SMPs.

They are also known as Symmetric Multiprocessor or SMP. Uniform memory access called because this is one piece of memory or one single address space or one piece of memory connected with the bus which is visible by all the processors is uniformly visible by all the processors.

(Refer Slide Time: 12:38)



Multiprocessors - NUMA

Distributed Multiprocessor, Shared Address Space-

Multiprocessors in a single bus has limited scalability as number of CPU-s that can be connected is restricted by space, thermal load, power connection etc. Distributed multiprocessors are deployed for better scalability.

In distributed multiprocessors, a piece of memory is directly attached to each processor and the processors are connected via an interconnection switch instead of the bus.

Each processor can access the memory attached to it via a non-uniform memory access protocols. However, this memory maps to a common global memory address space

Cache coherence supports usually not available, rather a directory based protocol, which distinguishes between uncached, shared and exclusive memory is used in programs.

Non-uniform memory access, (NUMA)

Schematic of distributed multi processor

The other variation of a shared address space system is distributed multiprocessor or any non-uniform memory access. Multiprocessors in a single bus has limited scalability as shown in the previous arrangement, because it is a single bus in which you are putting many processors, so you might not be able to put many processors on that bus due to space issue, due to electricity issue, due to heat generation issue.

So, one idea is that use distributed multiprocessors, where you can increase number of processors to a large extent. In distributed multiprocessor each processor is attached directly to one piece of memory and all the processors are connected via interconnection switch with each other. Like many processors each processor has its own cache and own memory, with own input output also, and all the processors are connected through an interconnection switch. However, since we are talking about shared memory system, therefore, all these memories must map to a common global address space.

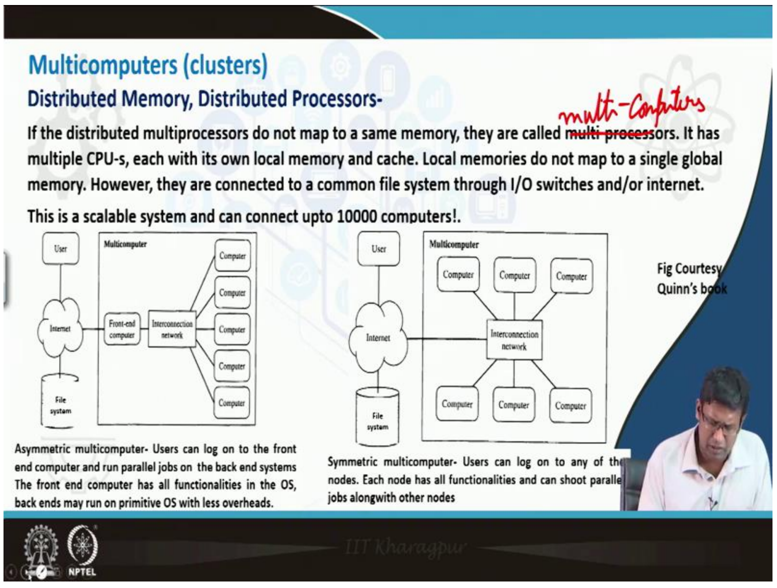
So, even if some part is residing here some part is residing here, some part is residing here, some part is residing here, etcetera and if there is some requirement to communicate, some requirement to access this data by this processor, this should request to the interconnection switch and look into the data. However, though they are physically different memories connected to different processors, but they point to the same address space.

Each processor can access memory attached to it via non uniform memory access protocol; that means, the memory is not uniformly accessible by each processor, and each can directly access its own memory also, but this memory is mapped to the common global address space.

Cache coherence supports are usually not available, because each has its own cache also. Rather there are some of the architecture which gives non uniform memory access cache coherent protocol, but they are very inefficient system as there are large overheads.

So, rather directory based protocol which distinguishes between the uncoached shared and the exclusive memory. So, for some part of the memory is known exclusive to processor 1, some part of the memory is known from processor 1 to shared with other processor, these directory-based protocols are given while storing the data in the memory, for this type of systems. This is NUMA or Non-Uniform Memory Access system.

(Refer Slide Time: 15:43)



Multicomputers (clusters)

Distributed Memory, Distributed Processors-

If the distributed multiprocessors do not map to a same memory, they are called ~~multi-processors~~ multi-Computers. It has multiple CPU-s, each with its own local memory and cache. Local memories do not map to a single global memory. However, they are connected to a common file system through I/O switches and/or internet.

This is a scalable system and can connect upto 10000 computers!.

Fig Courtesy Quinn's book

Asymmetric multicomputer- Users can log on to the front end computer and run parallel jobs on the back end systems The front end computer has all functionalities in the OS, back ends may run on primitive OS with less overheads.

Symmetric multicomputer- Users can log on to any of th nodes. Each node has all functionalities and can shoot paralle jobs alongwith other nodes

The next variant is distributed memory distributed processor. So, if the memory given to NUMA systems they are pointing to a common global address space, if they do not; that means, they do not do that means, they are different memories given, and different address spaces given to different processors we call it a multicomputer or a cluster and this is really a distributed memory system.

If the distributed multiprocessors do not map to a same memory, they are called multi computers; that means, multiple computers connected together through an interconnect network switch.

It has multiple CPUs each one has its own local memory and local cache. Local memories do not map to a single global memory; however, they are connected to a single common file system through I O switch and internet and some data through an interconnect network, they can share some data from one processor to other.

This is a scalable system because they are different computers and they are just connected through a network switch; you can use even a simple gigabit network switch which we use for your internet system. You can make an intranet and connect many computers and say that these are multi computer system and actually can perform distributed computing HPC computations using that system.

So, and you can increase the number of computers in that using higher end switches and you can you have 10,000 or more computers connected together. This is a very scalable system, and you can increase number of computers as much as possible. And one of the ideas is called asymmetric multi computer; that means, there is a user who can talk to the internet and through the internet he can access the front-end computer and the front-end computer is connected with interconnection network, which is connected with many other computers.

So, if any job is given it is given to the front-end computer, it launches it to the slave computers attached to it. This is called an asymmetric multi computer. User can only log on to the front-end computer and instruct it to run a parallel job at the back-end systems.

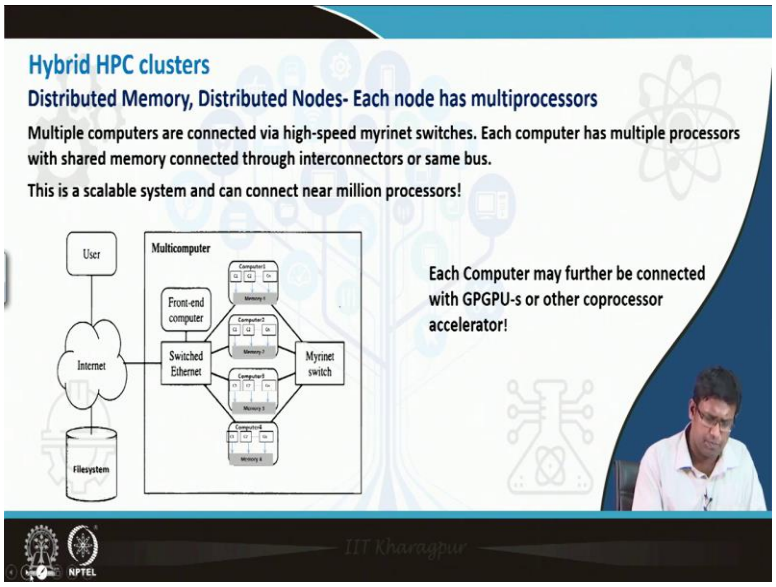
The front-end computer only needs to talk to the internet and user; therefore, it has an updated operating system with all the functionalities. The back ends are only designed to run some of the parallel applications which is launched by the front-end computer to them. So, they do they have a primitive operating system they do not need to have even graphic support, they are only supposed to do some of the arithmetic logical operations as instructed by the front-end computer.

So, and each of this computer have their own memory and they are connected through the internet connect network and front-end computer to a file system where each of the computers data can be written down. But while operating, they are operating only on the local memory given to them and they do not point to a single common address space and this is an asymmetric multi computer.

The other one is a symmetric multi computer where the internet is connected to the interconnection network and it can access any of the computers and these computers can communicate with other computers and make a group among them and launch a job. So, each node has functionalities and can shoot parallel job alongside this with them.

So, this is typically you see a very well developed cluster with a head node and slave nodes, and if you have your own lab computer and connected through an internet switch/ interconnection network which is basically an internet switch and then use another computer to talk to the internet and you can access any of these computers log on to any of these computers and launch a job. So, both these arrangements are quite well used. This again taken from Quinn's book.

(Refer Slide Time: 20:02)


Hybrid HPC clusters

Distributed Memory, Distributed Nodes- Each node has multiprocessors

Multiple computers are connected via high-speed myrinet switches. Each computer has multiple processors with shared memory connected through interconnectors or same bus.

This is a scalable system and can connect near million processors!

Each Computer may further be connected with GPGPU-s or other coprocessor accelerator!

The next one is a more modern and more developed cutting-edge application is a hybrid HPC cluster. So, this is a distributed memory, distributed node system, but now inside each node but inside each node there are multiple multiprocessors. So, each node is itself a shared memory multiprocessor or UMA type of thing.

So, multiple computers are connected via high speed myrinet switches. Myrinet switches are high speed network communication switches. Each computer has multiple processors with shared memory connected through interconnectors or same bus.

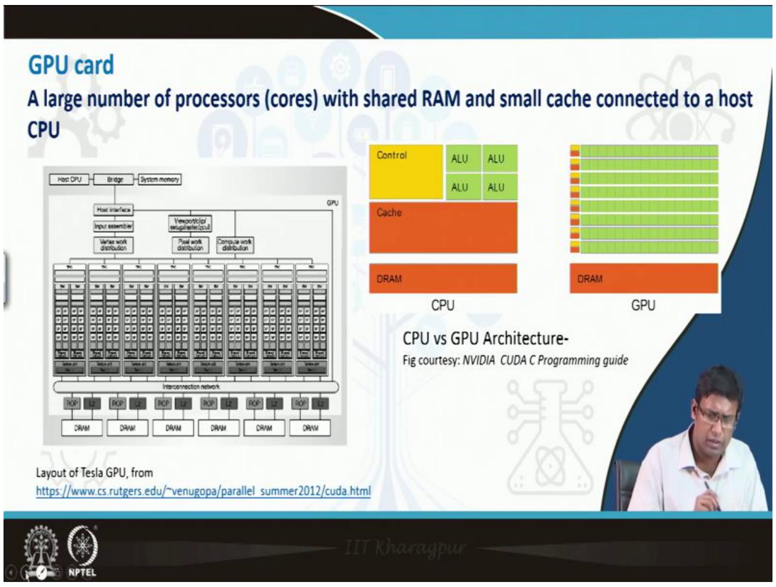
So, if you see this is a scalable system it can connect near million processors. If you see that there is user internet which you can connect to the switch Ethernet to a front-end computer and then through the network switch, front end computer can launch job into multiple computers in between them, and each computer is like today’s multi core computer. If you buy a computer today it is at least dual core two processor; that means, four core computers.

Similarly, if you go to each of this computer, they have multiple cores. So, each of the computer is a multiprocessor also which is connected to a local memory, to the computer, but this is a shared memory space for all the processors inside the computer. And apart from the Ethernet, they are connected through myrinet which is a much faster network connection and by this you can connect 10,000 computers like this and each computer can have few hundred processors inside it, as a multi-processor system. So, by this you can have a near million processor system.

And any of the; any of the HPC cluster is basically a hybrid HPC cluster with multi computers where each computer is a multiprocessor. Each computer may be connected with GPUs or coprocessor accelerators to give barrier speed.

And now, if you look to a leading supercomputer you will see that this multiprocessor computer arrangement is there and each computer is connected with something like a GPU or coprocessor. So, these are the most developed HPC architectures.

(Refer Slide Time: 22:53)



GPU card

A large number of processors (cores) with shared RAM and small cache connected to a host CPU

Layout of Tesla GPU, from https://www.cs.rutgers.edu/~venugopa/parallel_summer2012/cuda.html

CPU vs GPU Architecture-

Fig courtesy: *NVIDIA CUDA C Programming guide*

And now we also quickly look into GPUs, A GPU card or a Graphics Processing Unit card, these cards were initially developed by NVIDIA for gaming purpose. In gaming there are different operations done on different parts of the computer monitor and small computing codes are attached with different locations of the monitor and they are taking on the development or the progression of the game at different locations.

So, for that they developed something which has many cores (this is taken from NVIDIAs CUDA guide) connected with this is the GPU card which is at large dynamic RAM which is almost comparable with a CPU.

And many computing cores or computing units or small processors each is connected with a very small control unit and very small cache ,compared to a CPU where there is a large control unit and large cache and 4 or 8 arithmetic logical units are there, but here in GPU there are in reality 1000s of computing cores, but with very small control unit and very small cache.

So, it is something like a processor an inspired architecture .When a job is launched, it is similar job which is executed by all the cores together, but this is a very simple job which they can execute because the control unit as well as the register is small, they cannot execute a complex and large job .It’s a simple small instruction which can go to all the course and that can be executed together.

So, again that this GPU cannot work as standalone, it has to be connected via bridge to a host CPU and we can see that if we can use GPUs efficiently we can get performance much superior to a CPU, we can get even 100 times better performance than a very high end CPU using a GPU because many computing cores are provided here.

The disadvantage is that as the cache and the control unit is small, we have to be very careful about writing the program, and looking into the memory access during execution of any instruction in the GPU.

(Refer Slide Time: 25:23)

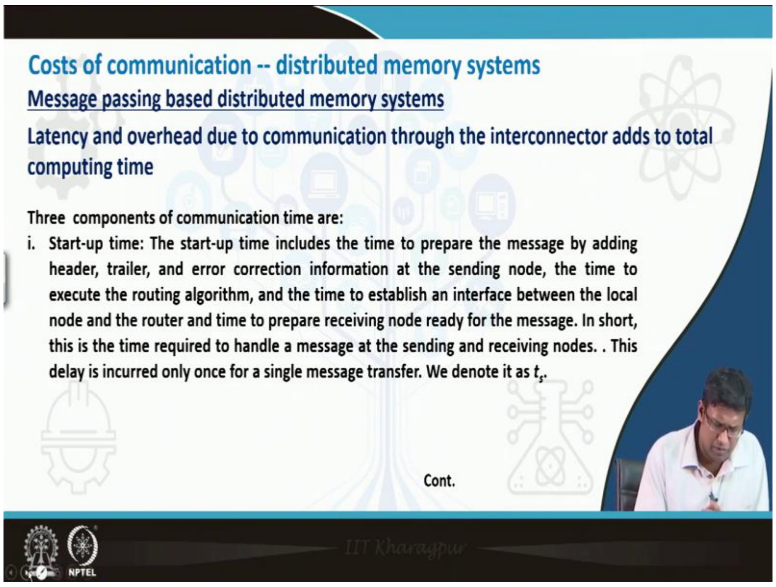
So, these are some of the important computer architectures that we use for parallel computing and now we come to something called cost of communication. We understand that if we talk about a distributed memory system each computer has its own local memory ,they have to ensure some continuity across the memory, one local memory given to one processor may have to be accessed by another computer, but it cannot access, it's in a different computers memory and how can you influence the computer and access the memory.

So, in case one computer needs to know something about a memory which is residing in others computer, the other computer has to send the data and this computer has to receive it. So, in a distributed memory system there is a requirement of frequent message passing across the computers.

Some message which is in form of information or data has to be passed from one computer to another and the other has to receive it and these distributed memory systems are called message passing based distributed memory systems.

So, we see into the latency and overhead due to communication through the interconnector switches, if message that goes it has its own latency because it will take certain amount of time, it has some overheads also this will add to the total computing time. So, the computer like we have seen for the cache for the sequential computer architecture, memory RAM CPU and DRAM system that RAM's latency is adding to the performance of the degrading the performance of the CPU actually.

Similarly, for when they are connected through an interconnection network switch and communicating across each other, the latency the communication time is adding up to the computation time and the computing time is increasing



Costs of communication -- distributed memory systems

Message passing based distributed memory systems

Latency and overhead due to communication through the interconnector adds to total computing time

Three components of communication time are:

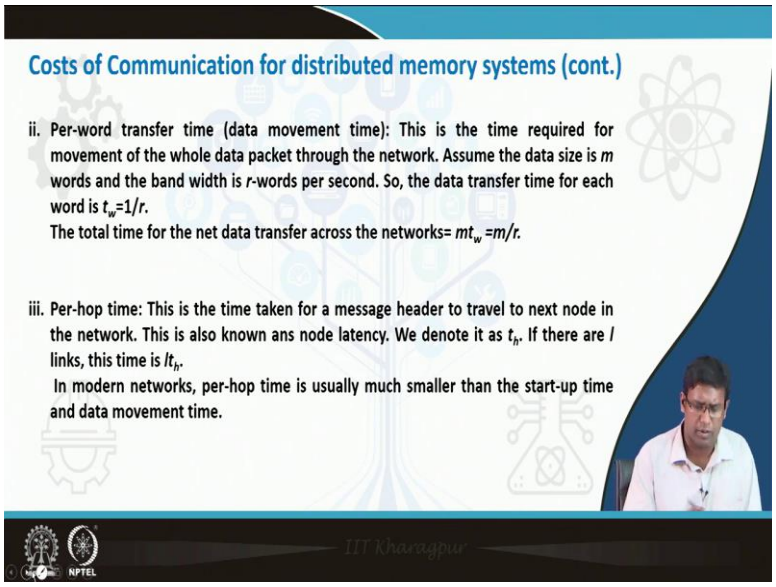
i. Start-up time: The start-up time includes the time to prepare the message by adding header, trailer, and error correction information at the sending node, the time to execute the routing algorithm, and the time to establish an interface between the local node and the router and time to prepare receiving node ready for the message. In short, this is the time required to handle a message at the sending and receiving nodes. . This delay is incurred only once for a single message transfer. We denote it as t_s.

Cont.

The three components of the communication time the first one is called startup time. The startup time includes the time to prepare the message by adding header trailer, and error correction information at the sending node and the time to execute the routing algorithm and to establish an interface between the local node and the router and the time to prepare the receiving node the receiving node must be ready for the message.

If all these times are added up this is the time to handle a message at the sending and receiving nodes, that this message is ready now it will now the receiving nodes know that this message will come through this from this computer to here and the in between router is also ready for that. For any data transfer this startup time or t s is associated with that, then any data for any data transfer first the message should be ready and the startup time should be done to prepare the message.

(Refer Slide Time: 28:36)



Costs of Communication for distributed memory systems (cont.)

ii. Per-word transfer time (data movement time): This is the time required for movement of the whole data packet through the network. Assume the data size is *m* words and the band width is *r*-words per second. So, the data transfer time for each word is t_w=1/*r*.
The total time for the net data transfer across the networks= $mt_w = m/r$.

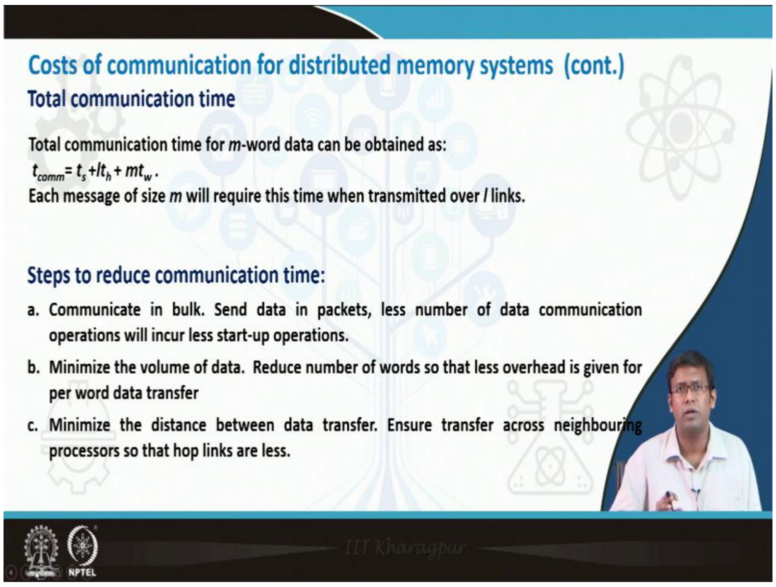
iii. Per-hop time: This is the time taken for a message header to travel to next node in the network. This is also known ans node latency. We denote it as t_h. If there are *l* links, this time is lt_h.
In modern networks, per-hop time is usually much smaller than the start-up time and data movement time.

The next one is (Refer Time: 28:35) per word transfer time. This is the time required for movement of the whole data packet from the network. So, if there is some data that is some information, some electric pulses which will go through certain net path and this electric pulse will propagate through certain path and reach another location.

So, depending on the amount of data or the bandwidth how much data we are sending, there is a time spent on sending the data and if bandwidth is r word per second, then time to send one data packet is 1 by r. If r words are going in a second one word will go in 1 by r second that is per word data transfer. And the total time for the net data transfer if we have a data size m that will be m / r.

Per hop time. If there are multiple nodes in the network, from one node to another node there is a latency. So, there is a hop time say th and if there are l links that a per hop time will be l*th. So, the total data transfer time in modern network per hop time is usually much smaller than startup and data movement time.

(Refer Slide Time: 29:54)



Costs of communication for distributed memory systems (cont.)

Total communication time

Total communication time for *m*-word data can be obtained as:

$t_{comm} = t_s + lt_h + mt_w$.

Each message of size *m* will require this time when transmitted over *l* links.

Steps to reduce communication time:

a. Communicate in bulk. Send data in packets, less number of data communication operations will incur less start-up operations.

b. Minimize the volume of data. Reduce number of words so that less overhead is given for per word data transfer

c. Minimize the distance between data transfer. Ensure transfer across neighbouring processors so that hop links are less.

The total data transfer time is given as for a data of size m over l links, start up time plus m into the per word data transfer time m*tw p+ l*th. Each message of size m will take this time to be transmitted over l links. So, startup time plus the per hop time into the number of links as many links will be there more will be per hop time plus the message size into the bandwidth into the per word data transfer time.

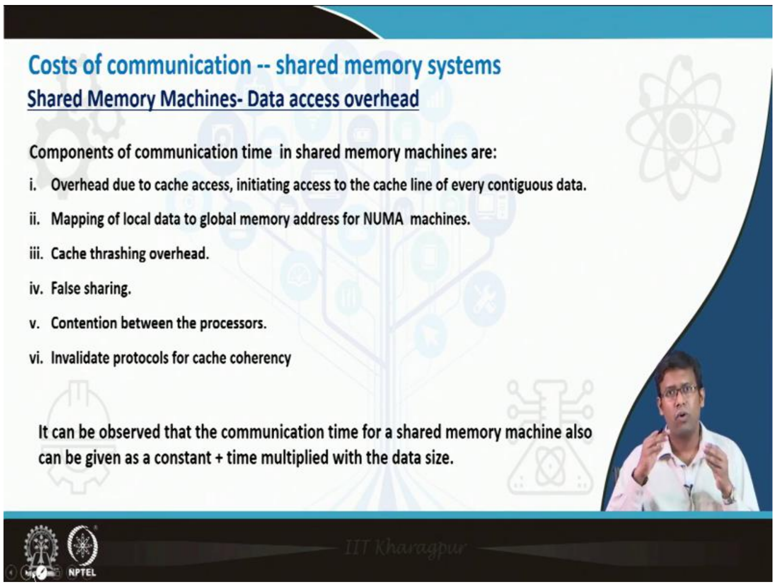
Now, if we do frequent data transfer, if we send if we may take a data and break it into multiple components and send it different goes every time there will be a startup time. So, the overall time will increase. So, if we pack entire data together or multiple data together as a single message and try to send it the startup time will be reduced.

Similarly, there can be some other steps to reduce communication time one is communicate in bulk send data in packets a smaller number of data transfer operations will incur less startup operations.

Minimize the volume of the data. If you can avoid sending some of the data or if you the total data size is same if you can reduce this to a smaller number, the per word data transfer time will reduce. Minimize the distance between data transfer try to ensure the data goes from one the neighboring processors so, that less hop links is there. So, that the per hop data transfer is reduced.

So, this three are three very essential steps we should think while writing the parallel algorithm in terms of in distributed memory that this way we can reduce the data transfer overhead.

(Refer Slide Time: 31:52)



Costs of communication -- shared memory systems

Shared Memory Machines- Data access overhead

Components of communication time in shared memory machines are:

i. Overhead due to cache access, initiating access to the cache line of every contiguous data.
ii. Mapping of local data to global memory address for NUMA machines.
iii. Cache thrashing overhead.
iv. False sharing.
v. Contention between the processors.
vi. Invalidate protocols for cache coherency

It can be observed that the communication time for a shared memory machine also can be given as a constant + time multiplied with the data size.

For shared memory machines there is a data access overhead not data transferred overhead because data is actually not transferred except for some of the non uniform memory access applications rather data is there is a common data space, where multiple processors are trying to access the data and there are issues like cache coherency false sharing etcetera. The components of communication time is basically the communication time is basically the data access overhead time that one processor is waiting till other can update its cache.

So, this communication time can be broken down as overhead due to cache access initiating access to cache line of every contiguous data. Mapping of local data to global address for NUMA machines that it is local data for each processor, but they map to a global address this mapping can take some amount of time.

Will discuss about false sharing in detail when we discuss about shared memory systems. Contention between the processor, same data many processors are trying to write, so, there will be some sequentially in between them everybody cannot write at the same place same location this contention will be there. Invalidate protocol for cache coherency is if one is writing and the others will not be able to write the others right will be invalidated. So, they have to wait to get their turn.

It can be observed that the communication time for shared memory machine which is basically data access overhead, is also given as a constant plus a time multiplied by the data size. So, in distributed memory, startup time is constant plus m* tw (data size into the time based on the bandwidth). Similarly, in shared memory also the communication time is a constant plus time multiplied with the data size.

So, this is extremely important to understand that there is a substantial amount of time required for communication of data in parallel computing system. Especially for distributed memory system this time can be very significantly large and can sometime reduce the performance of the parallel program. So, we should be quite well aware of this time and should follow the right steps to reduce the data communication overheads.

So, we will finish this part session now and we will continue with the discussion on different classifications of parallel computing architecture which is Flynn’s taxonomy in the next lecture.