# High Performance Computing for Scientists and Engineers

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

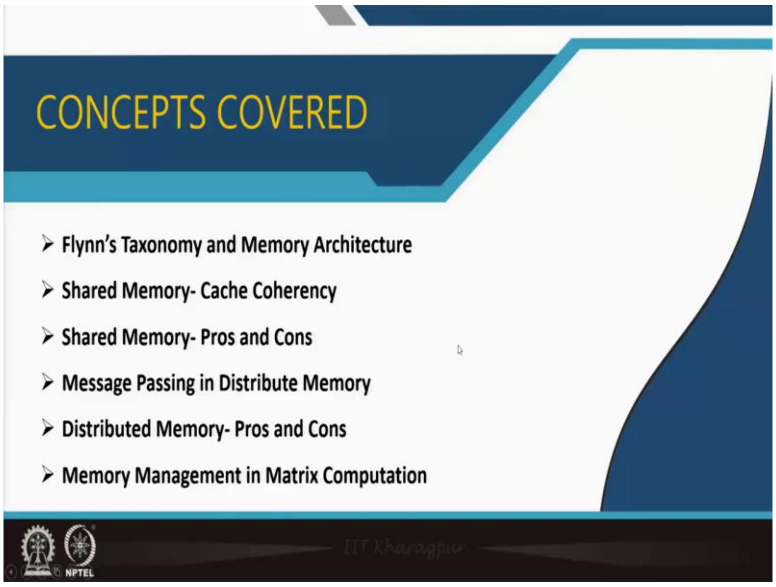
**Module - 01**
**Fundamentals of Parallel Computing**
**Lecture - 05**
**Shared Memory and Distributed Memory in Parallel Computing**

Hello, welcome to the class of High-Performance Computing for Scientists and Engineers. We are continuing in our first module which is Fundamentals of Parallel Computing. And this discussion will be on Shared Memory and Distributed Memory for Parallel Computing.

In last few sessions, we have discussed about parallel computer architecture and during that discussion ,the concepts of all the computers sharing a same address space which is called a shared memory ,and each CPU is connected with a different memory unit and they do not share a same address space, which is called a distributed memory were seen.

We will see some of the details on that. And we have also seen what are the communication costs, overhead in different type of memory architecture, but we will have some more detail discussion on it today.

(Refer Slide Time: 01:31)



## CONCEPTS COVERED

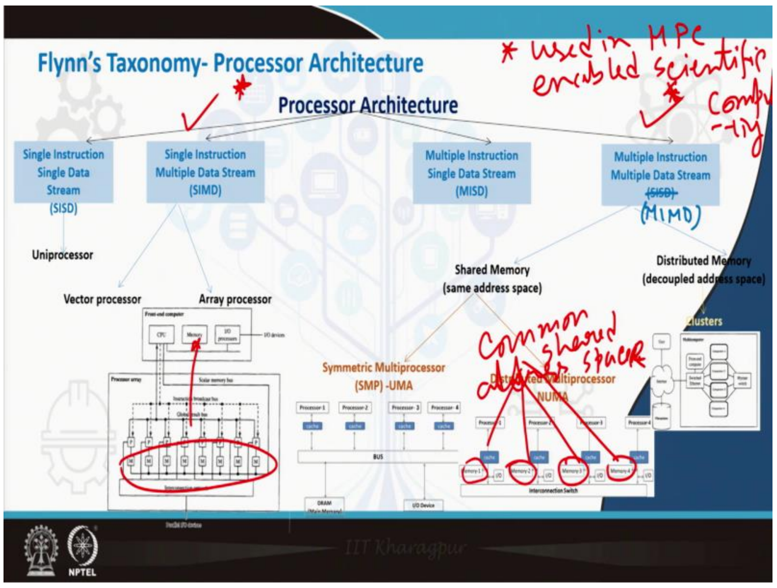
- Flynn's Taxonomy and Memory Architecture
- Shared Memory- Cache Coherency
- Shared Memory- Pros and Cons
- Message Passing in Distribute Memory
- Distributed Memory- Pros and Cons
- Memory Management in Matrix Computation

So, what we will do today? We will see how shared memory and distributed memory computers lie as per Flynn's taxonomy. We will look into shared memory machines and one of the important issues called cache coherence. And what is the issue exactly and how this can be handled and some of the pros and cons of shared memory.

We will see what is the main issue in distributed memory, that is message passing in between computers and again look into to the pros and cons. And this is important because as an application engineer of high-performance computing you need to decide that how do you want to put higher performance in your program. You have a sequential program you have a program which runs in single computer. And how do you want to make you parallel which type of architecture you will use, and which type of memory you will use you have to make a decision on that. And that is why it is important to see that both shared memory and distributed memory platforms are available to you.

We assume the fact and then how the which one you will chose based on your applications. So, the pros and cons are important to know. And then we will see some of the matrix computing algorithms, very basic matrix computing operations like vector, vector dot product, matrix vector product, etcetera. And both in shared memory and distributed memory how to do the memory management, if we think a parallel implementation.

(Refer Slide Time: 03:18)



## Flynn's Taxonomy- Processor Architecture

* used in HPC enabled scientific computing
Processor Architecture
Single Instruction Single Data Stream (SISD)
Single Instruction Multiple Data Stream (SIMD)
Multiple Instruction Single Data Stream (MISD)
Multiple Instruction Multiple Data Stream ~~(SISD)~~ (MIMD)
Uniprocessor
Vector processor
Array processor
Shared Memory (same address space)
Distributed Memory (decoupled address space)
Symmetric Multiprocessor (SMP) -UMA
Distributed Multiprocessor NUMA
Clusters
Common shared address space
BUS
Interconnection Switch
IIT Kharagpur
NPTEL

So, as per Flynn’s taxonomy, we have seen processor architecture can be classified in 4 classes; Single Instruction Single Data stream SISD, which is a sequential computer. We will not discuss it much further because we are trying to get higher performance which will of course not get in single instruction single data stream computer.

Single Instruction Multiple Data stream or SIMD, Multiple Instruction Single Data stream or MISD and Multiple Instruction Multiple Data stream. Well, the first one is uniprocessor, it is a single processor connected with the single memory unit. It cannot do anything in parallel. We are not discussing about it any further.

These two, single instruction multiple data stream and multiple instruction multiple data stream, are used in HPC enabled scientific computing. So, we will discuss about these two schemes, that is, single instruction multiple data stream and multiple instruction multiple data stream in detail.

So, in single instruction single data stream, 2 types of computer hardware are used in this architecture. One is vector processor that is all the computers are working on different elements of a vector. So, a vector processor is, like a sequential computer writes a plus b is equal to c, a and b are scalar and c is a single valued scalar. A sequence vector processor will write ai plus bi is equal to ci and this r for i ‘s which are of all the elements of that particular vector. And then array processor; that means, multiple processors are connected via bus to the same memory unit.

The array processor we have seen it in few of the first lectures, that there is a particular CPU which connects to many processors. Each processor is again connected to an interconnect network switch, up their separate own memory unit. However, all these memories; to a same memory space and physically same memory.

So, they can operate on different data, but they will operate it on the single memory unit on the same shared memory location. Then comes Multiple Instruction Multiple Data MIMD. There can be two different variances of MIMD; one is MIMD using shared memory. They are using same address space; however, the memory units are using same address, however, they can do multiple operation on multiple on different data.

One is a symmetric multiprocessor, where many processors are connected by a bus to a same memory unit and each processor can take different locations of the memory unit. It is a same memory, but there will be different locations of the address space and do different operations there. And it is called a Uniform Memory Access or UMA because it is a uniform memory which all the processors are accessing.

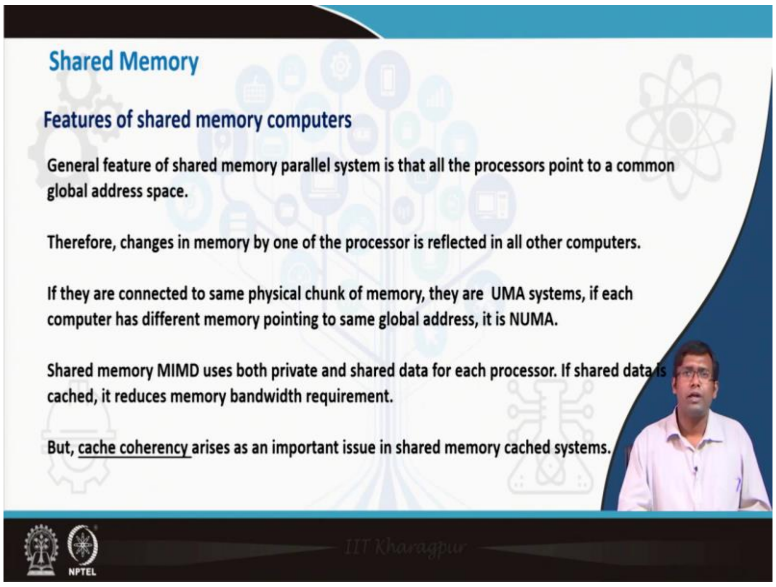
Then can be distributed multiprocessor or non-uniform memory access, that is there are different processors which have their own cache and own memory and these processors are connected through an interconnect switch. And all these memory units, though they are physically different memory units, but all these memory units are some way mapped into a common shared address space.

So, even if you see a vector a here there are few elements of the vector a, the next elements of the vector a lie in the other memory unit. Each of the processor, through that interconnect switch can communicate among them and see same parts of the memory which is lying with the other physical RAMs well.

And the third one is a cluster which is a distributed memory system. You have an ethernet by which you can connect many computers and all these computers are also connected through a high-speed switch. And each computer is a physical computer, it has its own memory, own processor, own CPU and own cache etcetera. They are pointing to different memory units, but these memories are not sharing any common address space. They are pointing to different disjoint address spaces.

So, if the first processor has a variable a, the next processor also can have a variable a, and these two variables can be completely different from each other. So, this is a MIMD system which has decoupled addressed space. So, for the shared memory, it has same address space, for distributed memory it has decoupled address space. However, we can have a MIMD using both of these and we can have SIMD using array processor or vector processors. So, it is different architectures both in terms of hardware as well as in terms of program which can be implemented to follow Flynn’s taxonomy.

(Refer Slide Time: 09:51)


## Shared Memory

### Features of shared memory computers

General feature of shared memory parallel system is that all the processors point to a common global address space.

Therefore, changes in memory by one of the processor is reflected in all other computers.

If they are connected to same physical chunk of memory, they are UMA systems, if each computer has different memory pointing to same global address, it is NUMA.

Shared memory MIMD uses both private and shared data for each processor. If shared data is cached, it reduces memory bandwidth requirement.

But, cache coherency arises as an important issue in shared memory cached systems.

So, we come to shared memory machines. The general feature of a shared memory parallel system is that all the processors point to a common global address space. They may be connected to a same memory unit, they may have separate memory unit attached with different processor however, the address space is same for all processors. Even if some of the memories physically located at some of the disjoint memory units, they are treated as a common memory space. Like if you go to your computer and say you have a 16 GB ram, you see that 4 RAM cards of 4 GB connected as 4 X 4, you are getting a 16 GB RAM.

If you a variable that can span up to this 16 GB space. It will be, though they are separate units, it can be considered as virtually as a same contiguous piece of memory. Similarly, even if these shared memory systems they are NUMA multiprocessors, they have different memory units for different processors they are pointing to the same address space. And for uniform memory access if the same piece of memory to which all the processors are connected, so, both physically as well as from the memory management prospective they are pointing to the same address space.

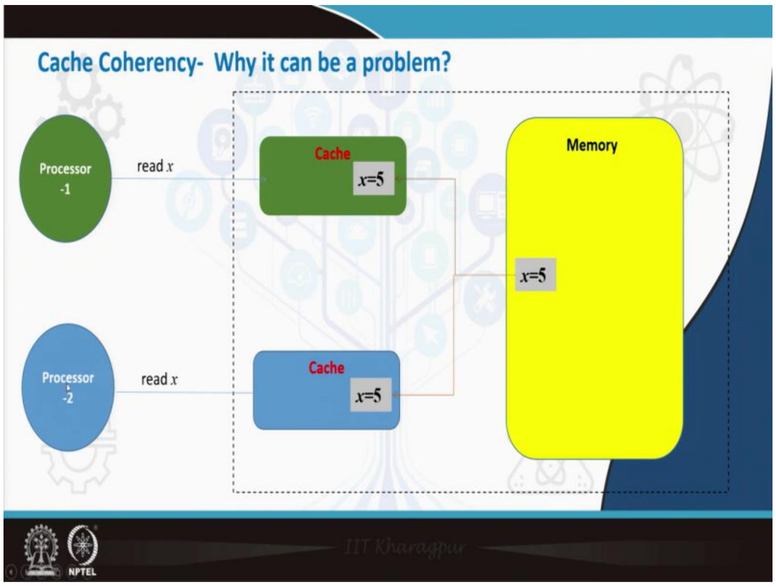
So, shared memory means, it is the same address space which is shared between different processors. Therefore, any change in memory by one processor is reflected in all other processors. If a is a variable array and third element of A is changed by eighteenth processor then all other processor sees that these values has been changed.

So, any it is same memory feature which all the computers are connected with so, all processor is seeing any change made one processor must be visible to all other processors. If they are connected to same physical junk of memory, they are called Uniform Memory Access or UMA system.

If the computer has different memory pointing to same global address it is called Non-Uniform Memory Access or NUMA. Shared memory MIMD uses both private and shared data for each processor. So, some data can be private to the processor; that means, it is not a shared memory that is not in the common memory space. It is some data which is only known by the processor and other processor can have similar data like this.

However, this is the same amount of data which is usually used for writing the loops, saving the parameters etcetera. If shared data is cached and we need to use cached systems for better performance. Again, if it is cached then the memory bandwidth requirement is reduced. So, it is good to have cache in in shared memory MIMD systems too. But, in case you have cache yes there is an important issue called cache coherency which is a very important in terms of the overheads and operation of a shared memory cached system.

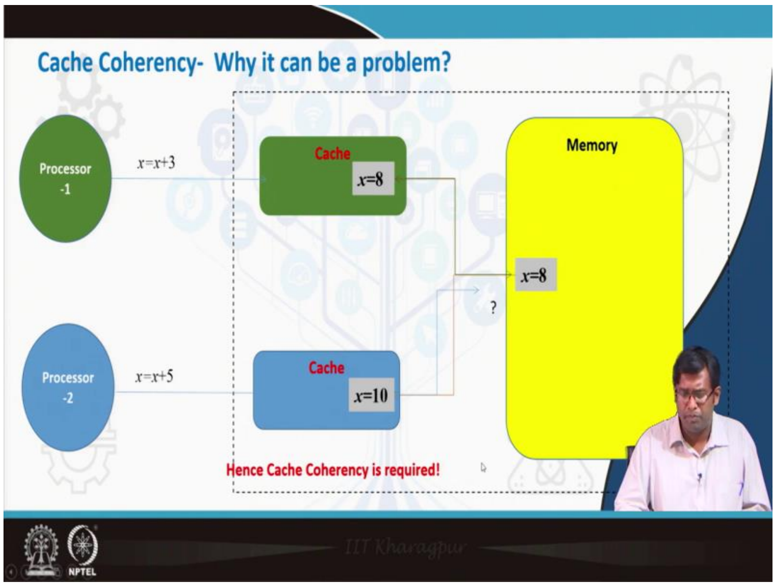
(Refer Slide Time: 13:30)


## Cache Coherency- Why it can be a problem?

We will see what is the cache coherent system. Say there are 2 processor; processor 1 and processor 2. They have their own caches and they are connected to a large memory system M. Now, there is some variable x in the memory system and both the processors will operate using that variable x. So, the first operation while doing let us say you have both the processors are asked to do x is equal to do writes x plus y, read the value of x from the memory, add it with some y and return the answer.

So, both will be there working in parallel connected to the same memory unit. So, both will be asked to read x. So, they will go to cache and ask cache what is x? Cache will prefetch some of the data, say x is equal to 5. So, this x is equal to 5 is prefetched and already lying in the cache. They will read what is x. They will read x is equal to 5. Both the processor's register will now know that x is equal to 5.

(Refer Slide Time: 14:28)



## Cache Coherency- Why it can be a problem?

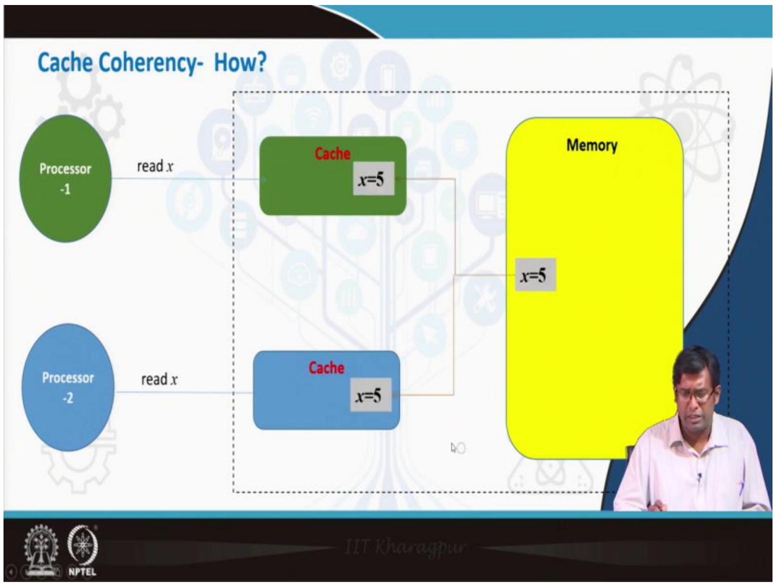
Hence Cache Coherency is required!

We talk about a multiple instruction system. Processor 1 is asked to do x is equal to x plus 3. Processor 2 is to asked to do x is equal to x plus 5. So, processor 1 as it does x is equal to x plus 2 it is cache will do it x is equal to 8. This x and second processor will do x is equal to x plus 5, its cache will write x is equal to 10 and any data in cache will be reflected.

So, x is equal to 8 goes to cache and x is equal to 10, the same variable x which is mean change by processor one as x is equal to 8. Now, x is equal to 10 is lying in the cache, but it cannot go to the memory because the memory has been written by x is equal to 8. And therefore, we can see that there is a conflict because it is the same variable x, but their caches have different values.

So, either it will overwrite it; that means, this operation will go in junk or it has to do something to establish coherence. So, we need cache coherency. So, when one variable is updated, it is updated uniformly in all the caches.

(Refer Slide Time: 15:48)

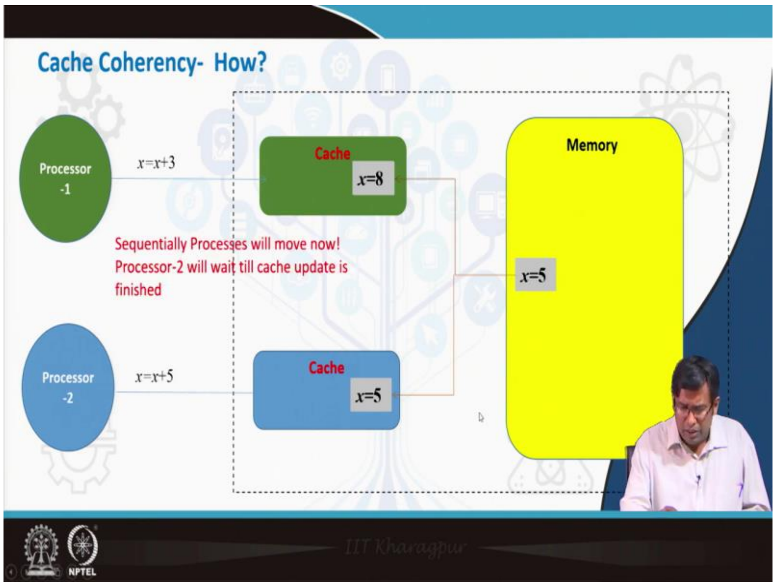


## Cache Coherency- How?

Processor -1
read $x$
Cache
$x$=5
Memory
$x$=5
Processor -2
read $x$
Cache
$x$=5
IIT Kharagpur
NPTEL

So, what can be cache coherent way? That it reads x is equal to 5.

(Refer Slide Time: 15:54)



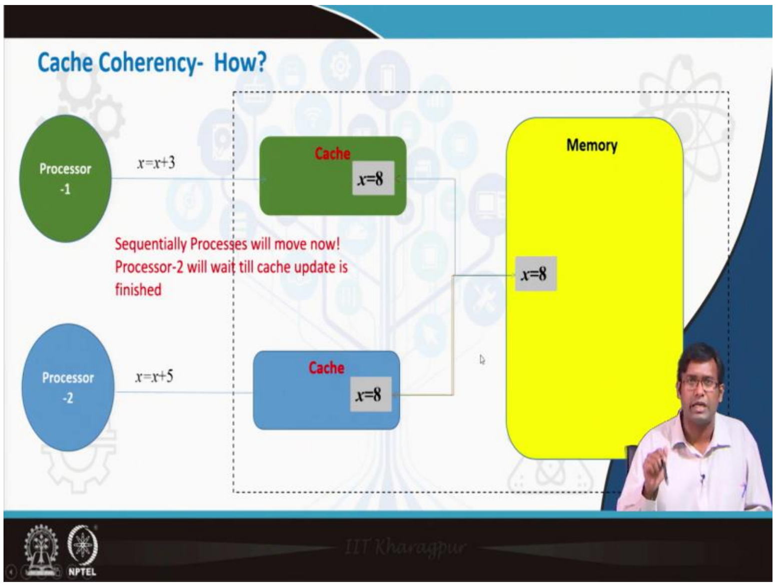
## Cache Coherency- How?

Sequentially Processes will move now! Processor-2 will wait till cache update is finished

Processor -1
$x$=$x$+3
Cache
$x$=8
Memory
$x$=5
Processor -2
$x$=$x$+5
Cache
$x$=5
IIT Kharagpur
NPTEL

The first processor operates x is equal to 8, which when one cache is modified the other cache has to wait till it goes to memory and comes back here. It has to wait for the cache update by first processor to the memory and again cache being updated by from memory to the second processor. Processor 2 will wait till the cache update is fished. So, though it is the parallel program processor 2 cannot, in order to have coherence in cache.

(Refer Slide Time: 16:32)

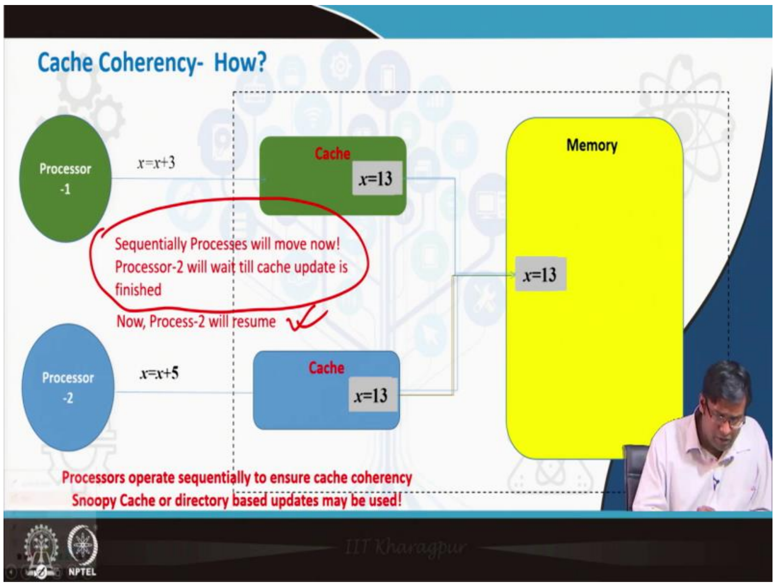


## Cache Coherency- How?

Sequentially Processes will move now!
Processor-2 will wait till cache update is finished

So, x is equal to 8 is in processor 1’s cache. It will go to the memory, it will come back to processor 2’s cache, and processor 2 will now operate x is equal to x plus 5, it will be x is equal to 13.

(Refer Slide Time: 16:44)



## Cache Coherency- How?

Sequentially Processes will move now!
Processor-2 will wait till cache update is finished

Now, Process-2 will resume

Processors operate sequentially to ensure cache coherency
Snoopy Cache or directory based updates may be used!

These will go to the main memory; this will again get back to the first processors cache. Now there is coherence everywhere. So, what happened? The second processor goes to an idle loop otherwise there was the stalemate. First processor has changed the variable, second processor has changed the variable differently, same variable has different values in different cache and it cannot go any further. There is a problem it is a conflict you cannot do anything, but here things can be updated right and x is modified by both the processors.

So, what was the initial x to which both the processors addition are done, but while the first processor has operated second processor has to wait for a while. Once this cache update is done then only it can work. So, there is a wait time by the second processor, which is introduced, and there is some sequentiality.

The processors now actually cannot work in parallel. What they have done in parallel? Parallelly, the caches have read the same variable, but  well some of the efforts are saved otherwise if the single processor is doing it every time going it back and changing the cache adding, it probably would have taken more time and not every time they are operating on the same variable. Certain point of time they are operating with same variable when one processor has to wait, but you cannot avoid it to establish cache coherency.

So, processors operate sequentially to ensure cache coherency and there are some protocols to establish cache coherency. These are called Snoopy Cache or directory-based updates. One protocol is Snoopy Cache; that means, if there is any change in the cache by any of the processors that particular memory sectors will be identified and the system will try to see where does this particular memory unit lie. In which computers have the same memory pointers are in their cache and that part of the cache will be updated in all the computers.

The directory-based protocol is that if some of the variables are in different computers cache it will be identified first in and this will be handled separately. So, using this protocols cache coherency is established. So that, what I saw a sequentiality in between the processors they can be avoided in certain cases. If there is a no cache conflict in between the processor's operation at one particular go, then they can run very much in parallel. So, this also has to be ensured via these protocols.

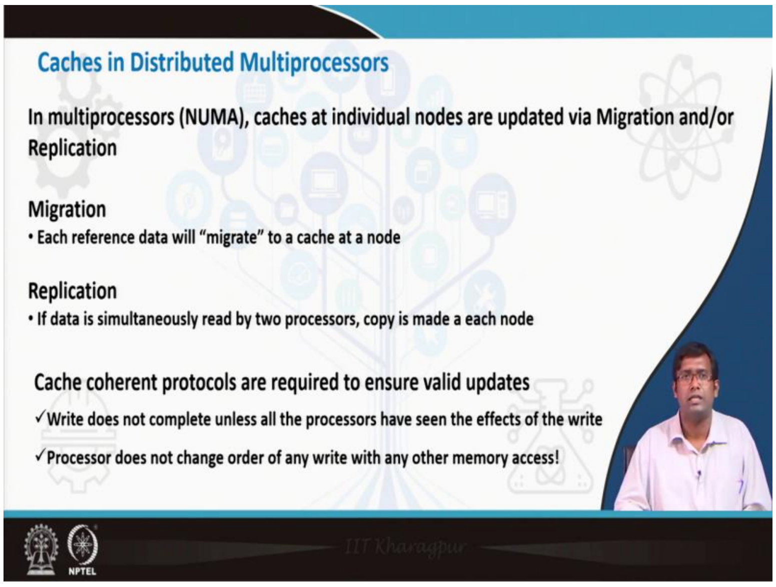
There can be another aspect. We will probably discuss it some point of time later that consider a case that cache is not a single variable cache, it prefetches data in an anticipation that these are the next data which will be used. So, it fetches a line or a line of array. So, instead of x let us operate on array a. An array a. So, a has say 30 elements and first 5 is being operated by processor 1 the next 5 is being operated by processor 2. Processor 1 changes something in the fourth variable of a.

However, when it is operating on fourth variable the cache has taken probably few more variables also of an also. So, it has taken a line of a, variable on which processor 1 is working that is not shared with processor 2, but cache has taken a line that is some of the elements of the lines are also in the cache of processor 2.

So, they have to follow some Snoopy cache or some of the protocol because they are sharing the same cache line. One modifies something on the cache line, the others have to wait till that operation is finished and reflected. It is called the false sharing. Though the processors are not sharing actually any value, but they are sharing a cache line and one is modifying the cache line other has to wait for cache coherency. It also adds to overhead.

Overhead means the parallel cost of communication increases and as a whole the cost of computation increases and the parallel efficiency reduces. We need to discuss about parallel efficiency later.

(Refer Slide Time: 21:34)



## Caches in Distributed Multiprocessors

In multiprocessors (NUMA), caches at individual nodes are updated via Migration and/or Replication

Migration
- Each reference data will "migrate" to a cache at a node

Replication
- If data is simultaneously read by two processors, copy is made a each node

Cache coherent protocols are required to ensure valid updates
- ✓Write does not complete unless all the processors have seen the effects of the write
- ✓Processor does not change order of any write with any other memory access!

So, for uniform memory access system symmetric multiprocessor SMPS it can be found out that it is a same piece of memory, therefore, cache update is coming from same the same piece of memory which is fine, but for non-uniform memory access multiprocessors caches, individual nodes are updated.

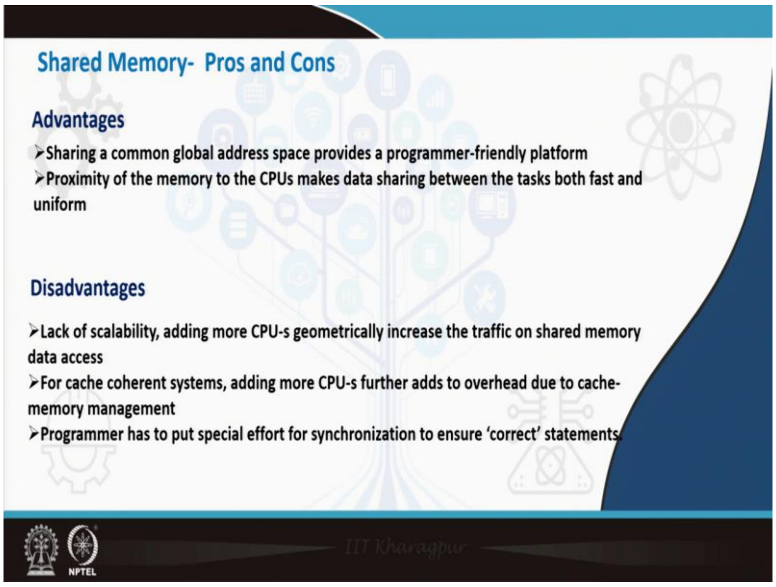
And this update requires that there is some change in the memory unit associated to some other processor, but this processors memory has to be also be changed to ensure cache coherence ,which is done to migration, that each reference data when some data is changed by one processor it will migrate to the cache at a node. From each of the reference data from one processor if the data will go to the cache also, and if data is being simultaneously read by 2 processors, then a copy is will be made at each node which is called replication.

These are used for ensuring proper cache access in distributed multiprocessors on NUMA and cache coherent protocols are again required to ensure that the updates are valid. So, nobody is overwriting something, which should not be over written by a one because it is changed by other processor and resides differently in different caches right. And the cache coherent protocol is that that one write is not complete unless all the processors have seen the effects of write.

So, if one processor is asked to do x is equal to x plus 5, the new x should go to the shared memory space and should be communicated to the cache of all the other processors which are using same cache line. Before that we will not say that x is equal to x plus 5 is complete that we will say that this is complete and the processor can now go for the next steps.

Processors does not change the order of any write or any other memory access. So, if there is some order of writing some of the variables is given first it will be done then the other memory access reading etcetera can take place then this also ensures cache coherence. So, this sees that it is actually having overheads. Instead of the processor being free to do its own work in its own way it has to wait till the variable is migrated and the changes are reflected in the cache of the neighboring nodes (Refer time: 24:17) which adds to the overall completing time or communication time also.

(Refer Slide Time: 24:20)



## Shared Memory- Pros and Cons

**Advantages**

➢Sharing a common global address space provides a programmer-friendly platform

➢Proximity of the memory to the CPUs makes data sharing between the tasks both fast and uniform

**Disadvantages**

➢Lack of scalability, adding more CPU-s geometrically increase the traffic on shared memory data access

➢For cache coherent systems, adding more CPU-s further adds to overhead due to cache-memory management

➢Programmer has to put special effort for synchronization to ensure 'correct' statements

So, now we will see what advantage and disadvantage of the shared memory system. The advantage is that they are sharing a common global address space. So, if you write a variable a it is known to all the processors. You really do not need to think that same variable can have different meaning and different processors except for some of the private data, for the main part of the data. So, it is a more programmer-friendly platform. A programmer knows that if you have discussed a variable, it is everywhere, it is known by all the processors.

Proximity of the memory to CPU is much closer to the RAM and this makes data sharing both tasks both fast and uniform. So, there can be cases that CPU, the memory is connected via bus for uniform and symmetric centralized multiprocessors SMPS and they can be connected and by interconnectors also through different units. However, the memory units are very close to each other these are very closely packed system. And specially for uniform memory access when they are reading from the same memory the reading is very fast and it is uniform also.

The disadvantage is that. So, one of the important is the programming wise it is easier to handle shared memory and also the communication overheads are less because the data sharing between the processors are fast and uniform. Disadvantage is lack of scalability. If we keep on adding more CPUs in the system ,it is a same memory which all the computers are accessing, therefore the traffic on reading the memory or writing the memory increases because there is the same piece of memory, it is almost contiguous locations with different processors trying to access and there can be a lack of band width. For cache coherent systems, adding more CPUs, adds to more overhead because this cache memory management has to be established by different CPUs and it further adds to the overhead.

Programmer has to put special effort for synchronization to ensure correct statements. Because again caches ensure that some of the statements may not be correct, so, programmer has to put special effort for synchronization, so that the cache coherencies established, while not executing any incorrect or invalid statement in the program. Well, so I have discussed about shared memory and cache coherency and the pros and cons of shared memory. In the subsequent class, we will discuss about the distributed memory systems.

Thank you.