Parallel Computer Architecture

Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati

Week - 02
Lecture - 09

Lec 9: Shared Memory Paradigm

Hello everyone. We are doing module 1, Introduction to Parallel Architectures. This is lecture number 9. Shared Memory Paradigm is what we are going to see in this lecture. In the previous lecture, we have seen the different layers of the computer system and also discussed the way a parallel program can interact with each other. One was a multi programming method where the programs were completely independent of each other. The second was a shared memory paradigm where the programs or the threads joined at the memory. And the third one was message passing where the programs joined at the interconnect.

So these are the three main topics or divisions which we had seen. So in this lecture, we are going to elaborate more on the shared memory paradigm. So the shared memory as the word says, we are sharing the address space. So a processor has a set of addresses, most of them are private to the processor and the others can be made as a shared variables.

So here the communication between the threads or the processes occurs with help of the memory and when you say memory, it means we are accessing the memory using load and store instructions. This is not a new idea. This has been existing since 1960s from the time of the mainframe computers. The processes as I said have shared and private data items. So we will see how can a process share its address with other processes.

So suppose I say that this is one of my process, the process normally has data, instructions, instruction data that is the code. So instruction means the code and among the data, you can have some private data variables and some shared data variables. Okay. And then there is the stack segment. So these are the various types of segments a process can have. And among the data segments, suppose this is my data segment, we can say that part of this data is shared and part of it is private.

So this shared portion of the data has to be accessible to another process. So if I say this is my process P1 and I have another process P2 to which I want this shared data to be accessible. So this P2 should be able to come and read this and vice versa. It should be

also able to read as well as write to this shared location. When P2 has to read write to a shared address space of another process, suppose the address range here is from 100 to 200 in P1's address space. Right.

So what are these addresses with respect to P1? They are the virtual addresses of that process. Right. So these are virtual addresses. The P2 wants to access those virtual addresses. We need a translator which will do this work for us and effectively to be able to access that it should also have a similar shared address space in its process state. Right. So in the address space of P2, I should also have the shared region mapped.

So we need to map this shared region into P2's address space also. And probably the addresses that is the virtual addresses could be different for P2. But overall that is the same memory. So this shaded region which is in the virtual address space of both P1 and P2 is actually mapping to the same physical region in the main memory. So if this is my main memory, then the shared region is nothing but a set of physical addresses.

So this is some physical address in the memory which is equal to the shared address range of P1 and also shared address range of P2. Okay. So this is how an address space can be shared. Okay. So this picture shows the same thing. We have process P1 and P2. The orange one is the private address space which has the other private variables and the green one is the shared.

So both of these green have may or may not have different virtual addresses between P1 and P2 or Pi and Pj, but they map to the same locations in the main memory. See, this is the same physical address to which these two green addresses map, this one and this one. So if I say address range A1 and address range A2 map to the same locations here. And this blue is the data item which is being shared. So here Pj, process j is going to write to that data item and process Pi is going to read.

So there is a load here and a store there. So this green address space is essentially shared by several processes in a shared address space. Okay. So that is how the mapping of addresses is. Okay. So a write to a shared address by one is visible to the reads of others because we are going to write to the same physical location which may be mapped to different virtual addresses across the different processes. Okay. So the communication architecture, what is there? These processes could be on different physical machines.

We need a communication medium through which we can read and write this data items. So using conventional load and stores. And when we access a same location by two different processes, you need to make sure that there is no race condition, there are no clashes when one race the other has to wait and vice versa. So we need all these read and write operations to be atomic. So atomic operations also need to be handled using

synchronization. Okay.

So what we need? We need conventional load store ISA and synchronization methods to be able to implement a shared address space. Moving on. Right. So to generalize the idea, this is our popular generic parallel architecture, where I have depicted three nodes. You can have multiple nodes that are disconnected to a global interconnect. Each node has got a processor or a cache, memory, a network interface through which it connects to the internet, sorry, interconnect. Okay.

So when I need to have a shared address space implemented in this, we can have certain pages mapped to this location. For example, here I am saying the Node 0 is going to store the addresses 0 to $N - 1$ Node 0. Then the next N pages will be kept in the next node. So here I am going to store the first N pages, page 0 to page $N - 1$ or you can say address 0 to address $N - 1$. Essentially, I am evenly distributing the pages or addresses across these different nodes.

And if this process P wants to access the page, 2N, it has a read on page 2N or a write on page 2N, what should it do? It should be able to traverse from here through the network interface and then go over the interconnect and then reach this node. Right. So this is where the page 2N is residing. So this is how we will access. However, it should be noted, that if this page is a shared page, it can be in any node. But if it is a private page for the first node, it is desirable that this page be mapped to its local memory bank or this local M.

So I am calling this local because it is closer. Because this is closer to P, this I am calling a local memory node. And if this access is for a private data item, it is good that it resides on the same node as the compute node. Okay. So in this system, how do we increase throughput? Alright. So to increase the throughput, we need to increase the interconnect. We need to increase the memory capacity, the I/O capacity by increasing more I/O controllers and I/O devices.

Memory capacity is very straightforward, add more memory modules to the interconnect. And what happens with this? You are increasing the capacity, that is more storage is available. But can I access this more storage at the same speed that I was able to access earlier? So that translates to do I provide the same amount of bandwidth when I add more memory modules? Similarly, to increase the processing speed of the complete system, I can add more nodes that is add more processors, which are able to work more faster. So faster and more number of processors will increase the processing speed. So the point to check here is adding memory capacity, is it giving me overall increased bandwidth? Right. So if you increase the memory capacity, is your available bandwidth

increasing? So we need to check this.

I/O definitely all right, but if you increase the processing speed, with the processing speed, is the throughput increasing? So if you increase the processing speed, am I able to increase the throughput? So for achieving this goal, overall, memory accesses is the main thing in any process because that is the most latency consuming task. If I am able to access it in a sequential manner, I access memory in a sequential manner, definitely there is going to be contention. One process at a time goes to the bank, reads and comes out. So sequential access will be slower. To make it faster, we can have a multi banked memory and also use address interleaving to make it little faster.

So multi banked and interleaved addresses will make the memory access fast. Okay. So our target is to achieve this. And in whole story so far, the interconnect is gaining more and more importance. Now why? So we said I want more memory banks, we will interleave the data across these banks so that we can parallelly access the data. Now accessing this data is possible if I am able to visit the multiple banks and that too together.

I want to access bank 1, 2, 3 and 4 because my 4 variables are in these 4 banks. Okay. So am I able to access it faster? Is it sequential? Is it parallel? Do I have so much bandwidth to go to all the 4 banks at the same time or as fast as possible? So this all depends on the system organization and how the interconnect is designed. Okay. So goodness of the interconnect is decided by the two matrix. One is the available bandwidth, and the system throughput. The bandwidth of an interconnect is good provided or rather the memory access bandwidth I would say which is same as the interconnect bandwidth available to a processor.

This is good if I have a direct connection between the processor and the memory because there is no disturbance. Whatever is the amount of physical bandwidth available that complete bandwidth is available to this connection. So adequate bandwidth is the first matrix. The second is better system throughput. Even if I have given individual one to one links between the processor and the memory, how much parallelism is available? Can multiple such requests or such connections be established in my interconnect? So this will give you a better throughput.

That is first thing is available bandwidth. Is it more? And to get better system throughput, is it possible that several nodes can access the memory module, so different memory modules in parallel? So if I am able to do this, then the interconnect performance will be good. And all this is possible only if I have a good interconnect. Okay. So these are three basic types of interconnects. There are more specific versions

which we will see later.

So today we are just going to see these three. The first one is the popular bus. Then we have a crossbar and then a multistage. So we will see them one by one. Okay. So the first one is a crossbar interconnect.

As the word says crossbar, this will establish a cross. So if I have something like this, I will just draw a rough sketch to understand how the design is. So this is, this center part is the crossbar and if A, I will just note something. I will just write something here, A, B, C, D. So if B wants to communicate with C, then I am going to establish this link here. Okay.

So this is how I will establish a B to C link. If A wants to talk to C, then we need to break this link and then establish the green link. But remember the red link cannot exist when the green link exists and vice versa. Okay. So a crossbar essentially helps to connect between different nodes. And if you are adding another processor, so let me generalize it to a processor design.

Here we have, suppose I have two memory modules and four processor nodes. Okay. So all of them need to connect to the memory. So this is the crossbar which I have drawn there, that figure. So each of this is that circle and if processor P1 wants to connect to M1, you know what type of a link you need to establish. You need to establish this link for a P1 to M1 connection and then P2 to M2 and so on. Okay.

So suppose here a third memory module gets added or a fifth processor gets added, you need to extend this network or this is called the switch. So you need to extend these switches to increase the size of the connections or size of the network. So adding a processor, you need to extend the switches or add more switches to the crossbar. Remaining structure remains same. But as you will realize that this is not so scalable beyond some number of cores. Alright.

So this only works for small number of cores. And so what to do? So what to do if this is not scalable? So what is the solution for this? So the solution for this is to go for a multistage network. We will take a quick look at the crossbar in this figure, the same thing which I had drawn on the previous slide. You have two memory modules and these are the crossbar switches connecting each of them. Okay. So next is the multistage network. Multistage is not like a grid structure of a crossbar, but you have multiple levels of connection.

In a crossbar, there is a single level connection, processor directly to the memory

because I fused this switch. Whereas here, if this processor 1 wants to connect to memory module 2, there is no direct connection, it has to go through these intermediate points. So first thing it comes here, then this switch has to identify whether to go in this direction or to go in this direction. So this decision has to be made, then it decides to move here. Again, here it has to decide where to go and then go here. Right.

So there are multiple stages through which P1 will connect through the first switch, then another switch and then it will reach M2. Okay. So there are multiple stages to reach. But the good point in a multistage network is we can establish connections across all memory modules. Here, this is, I will keep comparing with crossbar. So once P2 connects to M1, P2 cannot connect to M2 unless M2 is free.

Suppose this connection is in use. Okay. So P3 is connected to M2 and P2 is connected to M1. So when these two connections are being used, so P2 cannot use this switch because the above switch is in use. So there is a limitation that how many memory modules a processor can use. Whereas in a multistage network, P1 and P2, both of them can access multiple memories at the same time because there are multiple paths connecting each memory module. Okay. Because we have multiple paths available, there is more generalization of the connection.

We can use most of the memory modules in parallel so that there is more data sharing possible and so on. So with this all good points, what is the disadvantage do you think with the multistage? Okay. So you can think for yourself, pause the video and check whether you can understand or derive a disadvantage of this network. Okay. So moving on, the disadvantage if you have thought it correctly is that the latency increases because your number of hops are going to increase. You are going to go through intermediate switches to reach to the memory. And apart from that, because we are able to access multiple memory modules or multiple processors can now access multiple memory modules, the interconnect is being shared across these connections and hence the bandwidth available to every connection reduces.

So we have decreased bandwidth because of the parallel access is possible. We have of course seen the good points of the multistage that adding, so adding more nodes and processors is easy because it is not as limited as a crossbar. So addition or extending the network or scalability is good in this. Then the second one was a processor can access the memory directly because there are multiple connections available. And because of this property, we can have sharing of data structures and also the I/O events can be very easily handled because if you see here. Okay. every processor is able to access different locations of the memory and different memories in parallel.

It has multiple paths and allows more sharing of data and more parallel processing. So the sharing improves in this, but the latency disadvantage is there and you have little poor bandwidth. So that was crossbar and multistage. The third interconnect is the bus type. So when we could pack the processor, the cache, the memory management unit into a small chip or and onto a board, multiple such boards could be connected on a small bus.

So this was a central memory bus to which these modules were connected. So the advantage with this was every processor can access every memory module because there is a single path between the processors and the memory. So if I go to this figure, yeah, here. Okay, so that's a shared one single road is there and everybody can access this road. So this processor can access this memory, can access this memory.

So there is a possibility of being able to access every memory module and that facility is available to every processor. So you can imagine this good point comes also with the disadvantage that if there are more processors, your share or every processor's share of the bus will go down. Also there is a limit on how many processors or memory modules can you add to this connection because the average bandwidth available to every connection will reduce. Okay. So what are the good points of the bus? The bus allows access to any processor from any memory location. Any processor can access any memory location directly without establishing a connection, without making the crossbar connect and so on.

No multi-stages, nothing. A direct connection from the processor to the memory. And the other big advantage is that the access latency between every memory module is same. No matter what processor, it takes the same amount of time for every processor to access any memory module and hence the access is predictable. You know how much time it's going to access and that is helpful during programming. This system is called an SMP that is symmetric multi-processing system.

So bus gives you a symmetric multi-processor based system. The limiting factors as you understood are the aggregate bandwidth decreases because the available share of the bandwidth to every processor and the module changes or reduces if you add more processors and more memory. There is a small solution to this when you reduce the aggregate memory bandwidth. Of course the caches do come to help here because they try to reduce the gap of memory access latency. Suppose you go once to the memory, bring lots of data, so you don't need the bandwidth for a while because you are using the cached data. And problem with this is when you bring cached data, you have to manage the cache consistency and data coherence whatever.

So cache consistency, coherence maintenance becomes a new problem to handle if we try to avoid the aggregate bandwidth issue. Okay. So with this now let us see how can I build a scalable memory machine. Okay. So we have seen interconnects and every interconnect had some pros and cons. We use the bus but bus was not scalable beyond some number of nodes because it gave fixed aggregate bandwidth. That is the bandwidth was fixed and overall it reduced if you add more number of cores.

Then we looked at crossbar, it does not scale because its cost increases with the square of the number of nodes. Right. So the function is square if you add the more nodes or more memory modules. Then how do I build a scalable interconnect which gives me adequate sorry adequate memory bandwidth and at the same time has lesser cost. So I want a cost effective network with good enough bandwidth. When I do this at the same time see that the latency does not increase so much because if you go down on the cost the latency might increase.

So good bandwidth, low cost, improved latency. Memory access is the activity which any program does lot of time. So if you are spending most of the time memory accesses and if each access is slow the process is going to stall. So there is no point in having bigger and faster processors included if they have to wait for the memory access latency. Okay. So my target is best bandwidth, less cost, less access latency to make sure that the processors do not stall because if the processors stall we will have a poor utilization of our parallel architecture. Okay. So how do I build scalable parallel machines? So what are the options available? So we look at two options, one is UMA architecture and one is the NUMA architecture.

The first option is UMA which is uniform memory access. So here we use a scalable memory system and make sure that every processor is able to access every memory module using the same amount of latency. So essentially even if the memory is banked memory we treat it as a centralized memory and every processor takes the same amount of time hence it is called a dancehall type of architecture because the round trip latency for every processor to memory access and back is the same. Okay. Now when we do this the large bandwidth is required for every processor to access the memory. So if you have a good bandwidth then the UMA architecture is good. The other alternative or wait a minute I will just show you the design of the UMA.

So this is the uniform memory access architecture where the processors are on one side and the memory is common. So this is the centralized, so you have centralized memory and each processor can go and access that memory. But the latency of access is same hence uniform memory access. The second one is the opposite of this non-uniform memory access or NUMA architecture. Now this non-uniform word what does it tell that

the memory access for every processor or every location is different.

So we cannot say that it is going to take 5 or 50 milliseconds to access the memory every time. It may be faster, it may be slower. Now when can it be fast or slow? Given that the memory is closer to the processor I can access it fast, if it is far I can access it slow. So we move out from the centralized memory concept and move to a more distributed memory concept in this case. Okay. Look at the picture here along with every processor which was the processor and the local cache similar to this one but we no longer have the central memory but there is a memory bank associated with each node.

And I will call this a local memory however it is shared. Though it is local that is local essentially says it is closer to the processor or is part of that node itself. So we have a processor, a shared cache and a piece of memory attached with the node and we have such several nodes associated on the or attached to the interconnect. If you will rightly observe that accesses to this node is fast and access to a node across the interconnect, so suppose this one goes out here and then here and then comes here. Right. So this is a far of memory and that is slow. So accesses to local nodes are faster, accesses to remote nodes are slow hence it is called a non-uniform memory access.

So local is fast, remote is slow. You would want that most of the private data items that are not shared by other processes, they better be in the local memory node so that you can access them quickly you do not need to go on the interconnect. So most of the system designs make sure that the private data items get mapped to your local memory module. And the shared data items could be anywhere, it could be if you are lucky it may be in your local node otherwise it could be in a remote node. So the shared data items are possibly in local or remote nodes but the private are mostly in the local node.

So what does this give you? The bandwidth requirement of the interconnect goes down. Why? Because most of the accesses are served by your local node and hence you hardly go out. So here, if most of your accesses happen here itself you would not go out on the interconnect to do and hence this interconnect is getting more free time to serve others. Hence the aggregate memory bandwidth or the interconnect bandwidth is better. So this is how we can have scalable networks.

One is a uniform if you want uniform memory access assume a centralized memory. If you are okay or best one then non-uniform memory access is a better choice. So this shared memory paradigm is effective if I have a faster memory access that is the latency is low and good bandwidth of data transfer. Memory hierarchy is definitely helping us here because memory hierarchy philosophy says that we should bring the data we want

to use closer to the compute node. For example in a uniprocessor system it comes from the disk to the main memory to the cache and then stays in the first level caches or any cache on the chip so that the accesses are faster. So essentially I am bringing the data closer to the compute and in a shared memory system the bigger system which we are targeting the objective here also is that the data should move closer to the processor and for bringing this we have to move the data that is migrate the data from remote memory to local memory and at times if it is a shared data you might end up replicating the data because the remote node also is wanting to use that data item.

So replication and migration across a general purpose interconnect is the requirement for an effective shared memory architecture but this comes with challenges and that we will see how to handle them slowly. Okay. So overall for scalable system you need a better hardware interconnect and for cases of this that is I have replicated the data items I need mechanisms for maintaining coherence consistency of the shared data. So these two aspects we need to handle if I want a very effective shared memory system. Okay. So to summarize we have seen that communication and cooperation, communication that is sending and receiving information or sharing data across processes and cooperating with every with each other is the main foundation for doing this and for communication and cooperation in a shared memory architecture we use read and write to share variables.

Each process has its own virtual address space. Some portion of this virtual address space is shared across all the processes probably the addresses could be different but they share and point to the same physical location. Then data is transferred between the processes at the level of bytes, words or even cache blocks. So the amount of data, the granularity of transfer depends on the type of programs in the system. Then your addresses could be either mapped to local nodes or they could be mapped to remote nodes and address translation which is helping us to identify the virtual address in my process with the physical address of the real shared memory is managed by the address translation unit and this unit also makes sure that a process does not access private variables of the other process. Okay. So the security against the data items or the process address space is maintained by the address translation unit. Okay.

So overall a system can have multiple processors. They are may, they may or may not have a shared cache. Eventually they go out and access a memory that memory could be centralized memory in a uniform memory access that is a UMA architecture or it could be a distributed memory making it a NUMA architecture. Thank you.