Parallel Computer Architecture

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

Week - 02
Lecture - 08

Lec 8: Programming Models

Hello everyone. We are doing module 1, introduction to parallel architectures. This is lecture number 8, where we are going to see the concept of programming models. Okay. So, Almasi and Gottlieb in 1989 gave the definition of how a parallel computer is. So, they essentially said that parallel computer is nothing but a collection of processing elements which communicate and cooperate to solve large problems fast. So, very neat definition.

What does it say? It is a collection of processing elements which communicate and cooperate to do something fast. So, these are the keywords which we need to keep track of when we design parallel architecture. So, what essentially is it? It is a basic conventional computer architecture with some extensions. So, what are the extensions? Communication and coordination.

We have to address communication and address cooperation among the processing elements. So, when we add these extensions, we need to extend the conventional architecture with better communication and coordination methods to move from conventional to parallel. Okay. So, a parallel architecture can be written as a simple equation, conventional architecture plus communication architecture. Okay. So, what is conventional computer architecture? What we understand as an ISA and a good organized system and what essentially it gives us? A set of operations and data types on which we can operate them. And then an organization structure which helps us to run programs in a high performance manner. So, we need to deliver high performance.

So, computer architecture, the conventional one is operations, data types and then the structure. Moving forward, what is communication architecture? Again, it gives us communication and synchronization operations. So, it has to give us operations using which we can communicate and synchronize and then an organizational structure to realize these operations, that is communication architecture. Okay. So, to let us understand this with a picture. This one is a, look only at the left hand side, this part, you have three multi core systems. Right.

So, we have processor P, this is the processor, memory and IO. So, this one vertical line is a single core or unicore processor and I have three of these. If we run parallel program on all of these three machines, so I am running program P1, P2, P3 or, so these are the three programs I am running on these three machines and suppose they are part of one big application, that is these are pieces of a bigger program. How do they communicate with each other? Because these are physically different systems. So, when the systems are physically different and they want to communicate, how do we establish that communication? Okay. So, the easiest and intuitive answer is through a network. Okay.

So, I can use this IO, I can use this layer and establish a link between these machines to do the communication. So, that is one way. The other way is you could say that yes, my IOs are interconnected because we need a global interconnect, but we do not want to send messages across, but we will coordinate using the memory. So, we will say that the coordination happens at the level of the memory and not at the level of the IO. So, here this is joining at, so I use the term join at.

So, join at means where do they join to talk with each other. They join at IO in this example, here they join at memory, that is they use memory to speak with each other. And what option is left? The last one where they join at the processor. So, what does this mean? Joining at processor they are actually not talking with each other, but they are solving a similar problem on different data items. If you can visualize that processors joining means they do the same task on different data.

What is this called? Same task, different data, single instructions, multiple data, so this is a SIMD type of a design. Alright. So, joining at network, this is one type of communication, joining at memory is the second one and joining at processor is the third one. Alright. So for the first system where we join at network, what type of a program will you write? So for such systems the programs need to communicate using the network and this is popularly called message passing. So, you need to send a message from one system to the other via the network. So that is called message passing type of programming.

If we are joining at the memory, that is all the communication happens through the memory, there is of course the network will be used to transfer the data, but we are not sending messages, but we are only doing read writes to memory and so this type of program is called a shared memory programming model. And then the third one, we all concluded that there is actually no communication, but they do the same task on different data. Hence it is a data parallel or SIMD type of a program will run on the third system. So that is the historical view in the sense that a message passing program will be running on a system which joins at the IO or shared memory will run on a system which joins at the memory. And if you see the joining points are different, so a program which runs on a message passing type architecture will not run on a shared memory type of an architecture. Okay.

So that is the architecture dictates the programming model and vice versa. So if I have a programming model of one type, I have to run it on that type of an architecture. So there is a tight coupling between these two. So architecture decides programming model and programming model decides the architecture. So there is no possibility of cross compatibility.

I cannot run one program of one type on the other architecture. Okay. So that was the historical view or the way parallel programs were developed earlier. okay. So elaborating that further, architecture decides programming model. So if we join at network, the program has to be of message passing type. If we join at memory, the program has to be of shared memory type.

And if we join at processor, the program has to be of SIMD or data parallel type. The reverse programming model decides the architecture. If you have written a message passing program, you have to run it on a message passing architecture. If you have written a shared memory program on a shared memory architecture, an SIMD program on a SIMD architecture. But the question is, isn't the hardware basically the same? Do you really have a different hardware if you are saying I am having a join at the network or join at the memory? What is essentially the basic hardware? You have the processor, the memory and the IO.

So can we arrive at a convergence? Is a convergence possible? Because why not have a generic architecture on which any programming model will run? That is a generic parallel machine, which is able to run any program. Right. So because a problem can be designed using any programming model and can I have a hardware which is able to run that. Okay. So we have seen this generic parallel architecture in a few lectures back, which essentially had a node. So this was called the node. And a node had a processor core sorry a processor core, a cache, a memory and each node was communicating with the interconnect via a communication assist or a network interface.

And we had several such nodes connected to the interconnect including the disk. Okay. So what we want is with this generic parallel architecture, we want to separate the programming model from the architecture. Let the architecture be general and let me see how I can run any programming model on this generic design. So for doing this, what essentially we want to do is establish a communication mechanism. Okay. So all we need to do is how do we communicate? Because communication is very important.

If we can establish better resilient communication models which are generic, we could run any programming model on this particular generic parallel architecture. Alright. So what is communication? To understand communication, let us see the layers of abstraction of a system. Okay. So here, this picture shows you the different layers of abstraction. At the top you have the applications. Below the application is the programming models and we have seen the programming models, shared memory, message passing, data parallel. Right?

And then multi programming is a model where independent programs are running. So it does not matter, they do not need to communicate with each other. Okay. So the topmost layer of the abstraction is the programming model. This model wants to establish communication. How will it establish communication? With the help of primitives.

And who provides the primitives? The hardware in the OS. Okay. So all these programming models would need help from the operating system and the hardware to do this. So we need help from the operating system and we need help from the hardware. So that suppose this is one machine and I have another machine. So a message passing program would want help from the operating system because it has to send a message to the another computer.

So it goes through system calls and other library routines, then goes to the communication hardware through the physical medium, it will go out to another machine. getting it? So the programming models would need help from the hardware and OS to establish a communication. Definitely compiler and library support is equally important. What happens for shared memory? Shared memory does not need to send messages.

It simply has to write and read to a common memory location. So it does not need the OS as such directly. You compile the program and the program can directly talk to the memory or communication hardware to do read write to different locations. So shared memory requires load store to memory and that is more straightforward. Whereas message passing would require library and system call to construct the messages because we need to construct a message and write it into a buffer.

So slightly more work to be done with message passing. Hence message passing part, if you see this, you can see that it has to go via the OS to complete its task. And then eventually at the bottom layer, we have the communication hardware that sends the messages directly on physical wires. Right. So we can use direct physical wire based communication and these wires help to connect different machines. So that is the layers of abstraction, meaning how a communication can be established across different machines and to realize a different programming model. Okay.

So how do I define a programming model? So it is nothing but a communication abstraction. I hope you have all now understood that we have conventional architecture and communication. The way we communicate will decide the programming model. So it is like a contract between the hardware and the software. The software wants a particular way of communication and hardware provides it.

So that is all what a programming model does and you need to realize that programming model is not equal to a programming language. Alright. So what is it? It is a conceptualization of a machine that is the understanding of how a machine works which a programmer understands. So programmer has to know the conceptualization of the communication mediums and communication facilities available in the machine, so that the programmer can code the applications accordingly. Okay. And coding means you are going to write the same task but mainly now objective will be to cooperate and coordinate. So how do we cooperate and coordinate the activities at the program level? So for this the programmer needs to know the programming model.

The programming model also tells how to communicate and synchronize operations. So you need to cooperate, coordinate, communicate, synchronize. All these things are to be known by the programmer before we can use a particular programming model. We have seen four in the previous slide. So we will just list them one by one here.

The first one was multi-programming. So multi-programming as the word says we are doing multiple programs. So they do not depend on each other. Each program runs on a different core. Okay. So no communication is required here. We can exploit the multi-cores independently.

The second one was shared address space. So shared address space was join at the memory. So this is like a bulletin board. You want to communicate amongst your peers. So how will you communicate? One is that you will put a notice on a notice board and you go and write on the notice board and your friend goes and reads it. So this is how you have communicated with each other by writing and reading to a common location.

So this is the shared address space. The third one was message passing. So it is sending a message that is you write a letter, post it and that letter reaches to your friend and vice versa. So you need a communication channel. You need to write the letter, put it in an envelope and then send it across. So packaging the message and so you need more support from the hardware to do this message passing.

So it is like sending letters or even making phone calls to do this. There is a small difference that you need to know the point of contact because when you send a letter you have to write a correct address on it so that it reaches the correct person. So the point of contact is very important in a message passing. In a shared address space that is you have written in a bulletin board, anybody can read it. Of course your desired person would also go and read that but it is also available for others.

Hence in a shared address space you need to take extra care if you want to prohibit others from reading that. So we need to also take care of that in this model. Okay. And the fourth one was data parallel. So here it is more regimented actions on the data. So every processor works on a different data items and if they want, they do not want to communicate but the data could be distributed across different nodes because even if the instructions are located with a processing element, it would need data across a distributed memory.

So it will still need a shared address space and some method of communication to bring the data it wants to operate on. But there is actually no communication or sharing information while the program is running. So we have a regimented data actions. So that is data parallel. Okay. So when the course can work independently, so writing parallel programs is the same as writing serial programs, because it is like multi programming model.

But it gets complex when the course have to coordinate with each other. So what does coordination involve? We have discussed communication. Now what is coordination? Every program wants to do something or dependent on some another program. So they need to coordinate that is communicate with each other, some partial sums. In my example of sum of array elements, you had to send that yellow value across the communication network to get the final pink value and so on.

So sending partial sums is communication. Then load balancing. This is again important work because if you have a multi core system and your task distribution is uneven, some processors are heavily loaded, some are not loaded. So you won't be utilizing the system to its fullest extent. So you are going to waste computational power if the work is unevenly distributed. Hence load balancing is also important when we look at such systems.

The third aspect is synchronization. Synchronization is when do we go and access values that is read values or write values. Okay. In an example, if the array, the blue array in my example, if you are reading that, there will be a master processor who reads all the values. Then, these values are distributed across small programs. So the master is going to read values from the main memory from somewhere. And once these values are read, only then the other small programs can start the execution.

Until then they cannot start the execution. So first, all the blue dots have to be read only then the computation can begin. So there has to be a way of telling that the data is available to be operated upon that synchronization. So only when the values are available, the threads can start computation. When to start computation? Otherwise they need to keep waiting because the data is not there.

And for this we would need synchronization. The other end of synchronization is collection of the answers, because initially we distribute the data, and later we collect the answers to generate the final sum. So when are the two yellow dots ready to create the blue dot or the sorry, when are the two yellow dots ready to generate the pink dots? So you had y1, y2, these two values were there and then we had to add it to get the pink value. Alright. So to do this, are these values ready or should I wait? That has to be done. So to compute partial sums we need to do this. So synchronization will be required at multiple locations during your program. Okay, right.

So to summarize, we looked at the definition of a parallel computer which is essentially a conventional computer with communication, so that they can cooperate with each other. We also discussed that we can have a generic parallel architecture such that any programming model can run on the system. We looked at three programming models, shared memory, message passing, data parallel. To realize any programming model on a generic system we need strong and generic communication, a good coordination mechanism and synchronization mechanisms. Okay. So thank you.