

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

Module No # 01
Lecture No # 02
Parallel Programming Paradigms

(Refer Slide Time: 00:26)

Learn Parallel Programming?

- Let compiler extract parallelism
 - In general, not successful so far
 - Too context sensitive
 - Many efficient serial data structures and algorithms are parallel-inefficient
 - Even if compiler extracted parallelism from serial code, it may not be what you want
- Programmer must conceptualize and code parallelism
- Understand parallel algorithms and data structures

Let us begin so the thing would discuss in Monday was essentially motivation for we are learning parallel programming and at a very high level how it is similar in many ways and how it is different from sequential programming right. Basically sequential programming plus in many ways the same issues apply you have to program there are option for bugs just at an increase level.

There are more issues to consider you typically forget the fact that you are one of them rather than the vector so this where we are talking about whether we should learn parallelism or not and one of the questions is it was whether the automatic parallelization should be the vertical why should we do parallel programming compared and it should noble idea.

(Refer Slide Time: 01:30)

Automatic vs Manual Parallelization

- Manually implementing parallel code is hard, slow, bug-prone
- Automatic parallelizing compilers can analyze the source code to identify parallelism
 - They use a cost-benefit framework to decide where parallelism would improve performance
 - Loops are common target for automatic parallelization.
- Programmer directives may be used to guide the compiler
- But:
 - Wrong results may be produced
 - Performance may degrade
 - Typically limited to a subset (aka loops) of code
 - May miss parallelization due to static cost benefit analysis

But it does not quite work if you were to do automatic parallelization it would be you probably may make less programmer errors you would be able to even program faster because you are tune to thinking in a sequential way so you think that that you are only channel you are mine on one stream of or one thread to stream of execution am not going to worry about all the other things happening at the same time.

However the state of the art is that something can be parallelized there are rules that specially things of the nature that are wells structured like a loop can be easily parallelized so you say this loop was $I = 1$ to 100 and if those 100 things can be done at the same time then the complier can figure out then $I = 1$ does not depend on IQ so that can be parallelized so limited parallelization is done by the complier or that tools can be pro process before the compiler so pre-processors.

But it is mostly except for exceptions like open MP which have become not only a D factor standup also useful in the industry many of them are basically research tools used in the research text okay to show one demo or may be 5 demos but they when you run it on a large piece of board you get wrong parallel code okay so the argument that the bug or the number of bugs will be fewer is not quite their yet.

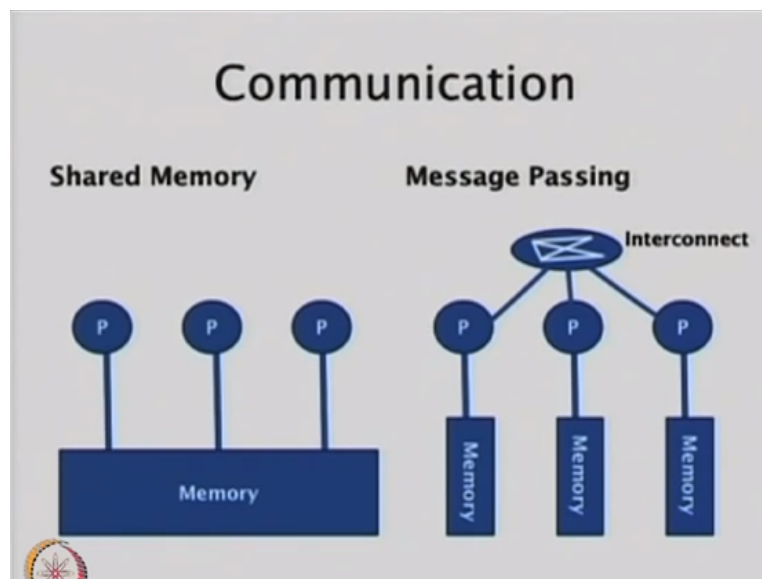
Because even the automatic parallelizing compiler reduces bugs reduces wrong code because it either misses the dependencies either immediately obvious from the analysis that does not quote or there is some programmer bug in the implementation of the compiler itself although that

probably less slightly there are more well tested by the fewer side of people than about that would be because of the programmer writing in the parallel that the other thing that happen even more often than the first time is it will not be paralyze the thing that will not be coming commenced.

It has some kind of optimization framework if I have to parallelize the piece of even if it is independent from other piece of board there is some overhead it has some model of the underline architecture some overhead because of some running things in parallel and that overhead is beginners so it is not going to come. And because of that optimization framework it gets some wrong man times.

So lot of things that could be paralyzed will end up not doing parallelized so these are some of the practical issues with the current state of the art in terms of parallelizing compilers or processors so think of it with the same way as far as we have concerned.

(Refer Slide Time: 05:20)

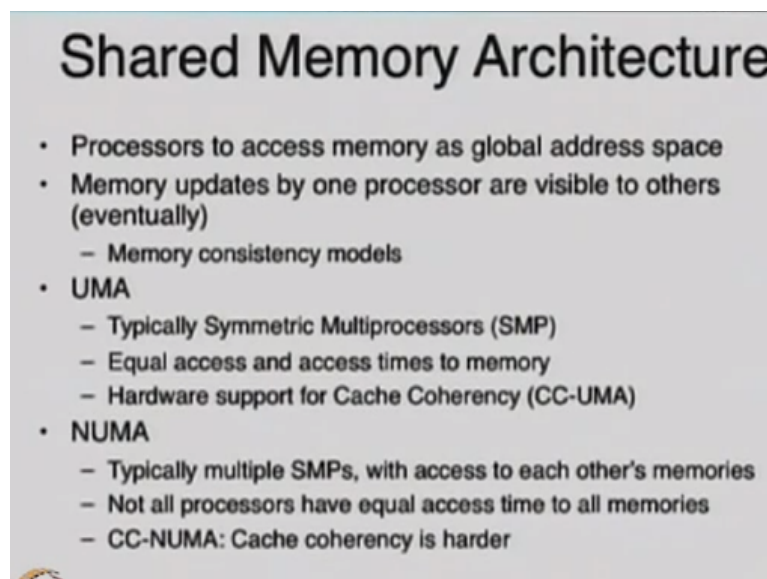


So getting back to this idea of this shared memory versus message passing so now we are basically getting into the framework for parallel so far we are talking about should we be learning to parallel programming at all is it worth it and the short answer is at this point the current state of computer science research it is needed right and enforce with future also it is going to be needed.

And so now we shift to actually learning a programming and we are going to begin with a I have shown you this slide before it basically show two broad classes of parallel architecture ok one is shared memory where you got multiple processors connected to one logically may be even physically shared memory and the other is distributed memory or message passing architecture where there are any number of processors they are individually connected to their own memories and they know about their passing.

And they want to access data there is address space which need to sent to their own local memory and when they need to talk to other computers they send a packet of data to the other computers.

(Refer Slide Time: 07:01)



And so these are the broad architecture styles in the shared memory architecture the main thing is that global address space okay. Meaning that all the processors address 0 to address and the we will talk also about the shared memory programming model which is select different from the architecture itself when you program you assume certain things ah for example that when you declare a variable that variable as the same value on all the processor because it is pointing to over it and talking about or referring to the same memory location okay.

But time being we are not talking about the high level programming model which sits on the top of the architecture we are talking about the architecture itself and so in the architecture we can think of it as at a hardware level there is one block of logic which is memory there is several

blocks of logic which are processors and there is some inter connect between we will get to bit more detail on that which takes you from the processor to the memory.

So when you say I want this address you go to that address and memory whichever processor says that address which goes to the block of the memory and they patch that data which means there is some notion of when two processors are accessing the same address what should they see when one writing that address and one is reading that address what value and how the value that are respectably written unlike should be later.

And so there consistency models in terms of what can be read you never want that something is read which was never written by anybody obviously so that is the basic consistency that says that if something as read somebody must have written and there are other more advanced engine models that are get to later in the course processors course yes memory is the physical one right and there may be a cache and all that so am keeping architecture at a very simple level at the timing although will refer to caching and all that is it arises.

When now a days there is also two kinds of shared memory architecture that have a become a well though one is the uniform memory architecture which is U1 and other is the non-uniform memory access is U1. An uniform memory access is basically the picture I showed you all the memory is connected to all the processors or in other words every sees all the memory and as far as the memory is concerned no processor is more important than a other processors so it is symmetric.

Every processor gets equal opportunity to read every piece of memory equal opportunity to write into every piece of memory. So there is no preference for processors they take they get in the same cube take the same amount of time to read from the memory again at the cache issues and all the often even the cache may be those the same cache which means go through the same cache it in the cache for everybody okay.

The NUMA architecture is little different in that the global address still shared memory but they may be memory that local to one processor then further from another processor than access it faster than other processor okay so the access to that memory so not uniformly distributed or not

uniform among available processors okay. So I will bring you back to the same picture that I had earlier okay.

So this looks like the picture you had earlier but this time on the right hand side is not a disputed memory system is all message passing system but a new message okay and another left hand there picture it was the shared memory picture it was instantly expanded in that layer of memory controller has been added. The memory chip itself as only one program one processor can read it at a time.

So somebody have to arbitrate among the processors so that one wire goes from the memory probe into this controller and then multiple wires then there from the controller to the processor and so on the left hand side you see the uniform memory access meaning that memory controller X as a memory for uniform access amount the processors and it basically as a queue handset and you want to read something and gives you that piece of information from the memory chip and gets you done.

On the other side every processor have some local memory that is a practical implementation of the new architecture for examples these days Intels, Zions all have shifted focus on NUMA side. So you do not get to go through the controller you simply go through if you want to access and address in the same global space in common but if that address happens to be in the range which is local to you.

You simply access your local memory directly connected to your local memory through cache and all that through ignoring that one. If it is not in your range to figure out it is in the range of one of your co processors you tell the co-processor I need it and that co-processor is going to fetch it and give it you or you writing you spell the co-processor that you are going to write to it and that will take care of right.

And so it is in many ways similar to distributed memory or message passing system except you get one common access to the memory and in terms of address space and again one variable maps to one area program level is disjointed from knowing is whether locally here or locally there it is not saying send it to processor on the three. Programmer simply saying send it to

processor number three programmer is simply saying read this very well or write to this very well that makes shared memory programming it means relatively easier to do.

(Refer Slide Time: 14:42)

Pros/Cons of Shared Memory

- + Easier to program with global address space
- + Typically fast memory access (when hardware supported)
- Hard to scale
- Adding CPUs (geometrically) increases traffic
- Programmer initiated synchronization of memory accesses

In fact again those things are optimization details as at a hardware level this is the facility has been provided and yes if you were not very careful about how you allocate memory new architecture would be bought so in person and the OS manufacturer all go to great lengths to have these specific modules that at the allocation time will make sure that the correctly try make sure that memory that is allocated in is in the range where it is going to be needed for used.

It is mostly at the hardware level so Intel as designed the protocol at the hardware level so that it that can happen right. But the application cannot be compared at the OS so to not necessarily application this application has aware of it but it can still take advantage of it at least the over saturated to be aware of it. So when application OS is this set is address so may be it is more likely the that time we will use it and so it is going to give you an area in this address space.

This part of the address space okay and again an whole area research in terms of how to figure out where it is going to be needed and allocated time. You mean in the application does not have this facility or in fact there is if you look at open source Linux or Solaris you will get this special modules it is called PFM or something like that which our job is to figure out where this memory should be allocated.

It is not only whose is calling it but variety of other juristic okay yes that it could is you can call it NUMA basically right at some level either the application is aware of the being local and all or it is not. So it is whether you suppose that to the programming model but the architecture is still no more. So at the architecture is still node so at the architecture if you have common address space with some of the address space be faster for you because you made that closer physically memory wise.

Then that belongs to the new architecture if they are all the same it belongs yes that is not shared memory right. So when we get could have realized typical high performance architecture will be a grand mixture of many things and it is going to be one prime example of it so it is not going to be that this is the entire machine in fact we will look at a few machines if we to that before we end up they are some of the either historically top of the line from parallel computers or today's top parallel computers.

It is not that entire machine is passing space so entire machine is NUMA entire machine is you must shared is that clocks that have one architecture and then clocks are top of it and hierarchy of architecture that are typical in today's architecture okay. No we are talking about architecture right now which means uniform model means that at the hardware level we have one it need not be one chip necessary but one logical space.

And symmetrically all the processors are connected to it through some interconnecting memory like memory control. So it does not have to do with what the virtualization level is patterns or access method is okay. Typically yes but they do not necessarily communicate to themselves yeah so memory controller this belongs to this chip and so all this chips are hanging of this bus or sometime of interconnect and it knows in fact it would be extreme simply hard to have a single chip memory in today's computers right.

Because we are talking about the terabytes main memory I am getting a single chip memory for all of that is simply out of the question okay. You can very go up to one Giga byte or of that range but not terabytes memory on single chip so typically there will be many chips but you see that one memory. So the main advantages of shared memory are that of course they are very easy

comparatively program to deal with this data belongs to that computer and let us send it to that computer of anything at that cell.

It is some way similar to the regular sequential program regular machine where you allocate variables you use variables right and if that variable is UMA or NON UMA but variable is going to be fed to you sometimes it may take longer if it's NUMA but you do not care about it at the programming level. At the design level you may still want to make sure that that memory getting used is local but at the program we do not require it.

And for the same reason basically that all these processors are to be symmetrically surrounded by the memory controller whatever in between memory and the processors it is extremely hard to scale in fact that is one of the reasons why as we discussed the chip may be slightly going but the number of the chips is definitely going fast which means typically number of course is growing fast.

And if you are going to have one block of memory being serving multiple processors the complexity of the piece of hardware which is going to give equal access to the processors those are geometric and because the number of processors you make up one more core and that core going to interact with all these other cores because that is what this guys we are going to do the memory controller.

Whereas if you had a distributed memory machine somebody's complexity will still go on it is some interconnect which is going to have one more processor machine processor memory node attach to it okay. So the complexity is interconnect which may be as simple plus the Ethernet it just one long wire with everybody hook to it or typically something more complex in it but on the architecture side it is very simple you just log in one node into this interconnect.

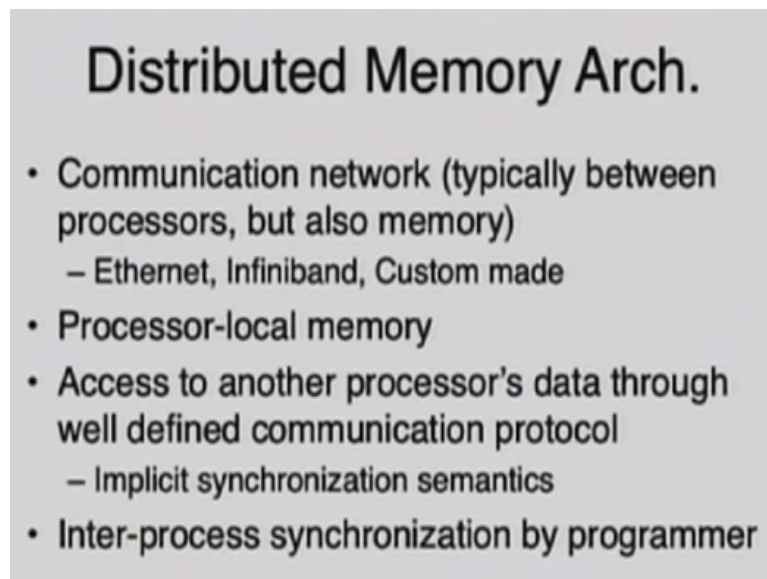
And that node will have its one local memory it will talk to other nodes but there is a no understanding that somehow all of the memory needs to service all of the nodes in a symmetric uniform okay. Also synchronization which is going to initially regardless whether you do shared memory or not is going to have to be programmer control I have mostly set many of the things that distributed memory architecture in takes.

There is going to be communication network but between the processors it is going to be typically visible to the user right there is a communication network in the shared memory side also but this is directly visible to the application and the application here is going to know this is local and this is somebody's else okay there is not notion of global address space. They have been not particularly successful but they have been efforts to distributed memory machines and provide a software layer on top of it that retains that is common address space okay.

So the application level basically allocate space and this layer is going to set of hardware being responsible for getting data from remote locations this layer will send to that node to get from that node and all that however because the software is technically performance lacks significantly but there are systems in fact simply in install on the distributed memory machine so that application programs can think of it as a shared memory machine.

And that the reason the done is the programming shared memory machine is easy assume this set of variables you declared variables we use them.

(Refer Slide Time: 26:10)



And again the synchronization is going to be the responsibility of the programmer sometimes the synchronization is implicit the distributed memory any way sending somebody as to receive and so there is hardware wise as well as the programming paradigm wise the programming

methodology that synchronization is implicit if you are sending in the hardware if nobody is to receive that send is going either it block or will have to not succeed yet.

Similarly at the programming methodology level you have to physically call the sync function level the receive function somewhere only then process can happen of transfer of data okay.

(Refer Slide Time: 27:02)

Pros/Cons of Distributed Memory

- + Memory is scalable with number of processors
- + Local access is fast (no cache coherency overhead)
- + Cost effective, with off-the-shelf processor/network
- Programs often more complex (no RAM model)
- Data communication is complex to manage

And this basically similar pros and cons it is going to become more scalable when you have distributed memory systems but then again you have to be responsible o understanding what is local what is not local typically and there has to manage that but from the hardware point of view some software somewhere is responsible for understanding the local and non-local memory BI right so this memory and that memory is there from others in the distributed memory system there is no need for connection there is no need for protection right.

Is only one wire going to the local memory which is from you so there is nobody can access its no nature protection okay.

(Refer Slide Time: 28:03)

Parallel Programming Models

- **Shared Memory**
 - Tasks share a common address space they access asynchronously
 - Locks / semaphores used to control access to the shared memory
 - Data may be cached on the processor that works on it
 - Compiler translates user variables into "global" memory addresses
- **Message Passing**
 - A set of tasks that use their own local memory during computation.
 - Data transfer usually requires cooperation: send matched by a receive
- **Threads**
 - Multiple threads, each has local data, but they also share common data
 - Threads may communicate through global memory, User synchronizes
 - Commonly associated with shared memory architectures and OS features
- **Data Parallel**
 - Focus on parallel operations on a set (array) of data items.
 - Tasks performs the same OP on a different parts of some data structure



So programming models let us talk quickly already talked about two hardware architecture and then it is not necessary that shared memory architecture is only going to support a shared memory programming model or distributed memory hardware architecture will not supported shared memory programming model right and so in some sense instead of calling them programming models I should be calling them programming tools.

So that it is clear that these are the ways we can write you program and sometimes in the same program one may be better than the other later in the same program later on tool may be better so think of them more as ways in which you may manage your parallel program you rather than saying there is one way of parallel program okay. It is very common right shared architecture says that a piece of data that.

But I have to manage relative access right I need to lock it or whatever if instead I decide am just going to tell that processor you do something to that memory because somehow we have decided that we have come to agreement that you are responsible for that junk and I am responsible for this junk. So that gives big raise to the big different way in which you program rather than locking that piece of memory and writing and unlocking.

And you will realize though locking and unlocking times are unavoidable these things have a significant cost right we will later in the course talk about lock free synchronization technics so that in limited situations even though there is resource that I shared in content by ordering the

access in certain fashion you can get consistent access to it without physically having to meet on a for an locking a piece of memory learning so and hence so forth okay.

So locking as an overhead so you sometime avoid locking by using message passing interface on a shared memory architecture okay. Just repeating shared memory in a shared memory programming model you have a common view of the memory and again this can happen on distributed memory architecture because some layer is going to figure that you are going that access area which means it needs to go to that other node and it is going to do the communication.

So you can think of it as a shared memory and underneath somebody is taking care of sending the data around okay and locks and sound of ours are practically inseparable from shared memory. If you are going to give shared memory specially in the beginning when you are sure about what you are doing you are going to start to lock regions of memory access small piece of it and then unlock entire memory okay.

And then gradually you are going to make it more efficient that is one style of programming now in message passing again the idea is that there is some address give to that node instead of the way the global address space was given to the memory and at a programmer level you simply say I have this much of data to many bytes of data and sometime these are high level constructs can say these many integers instead of bites of data and you send it to node number 39.

There is some addressing if you have done internet based or IP programming you have seen one flavor of this where a you have got a some address in that case it is IP address and you say connect to that address that port. So here it is typically not necessary unless it is Ethernet based interconnect it is typically an in fact even if it is an Ethernet based interconnect typically you got some interface of communication that says all the nodes and number typically from one to end and let me send this data to node number 39 okay.

Have experience with threads may be done thread programming how many of not done thread programming everybody has done or you are to scared to say that you have not. Programming in terms of thread it is again another way of programming another paradigms to speak where you

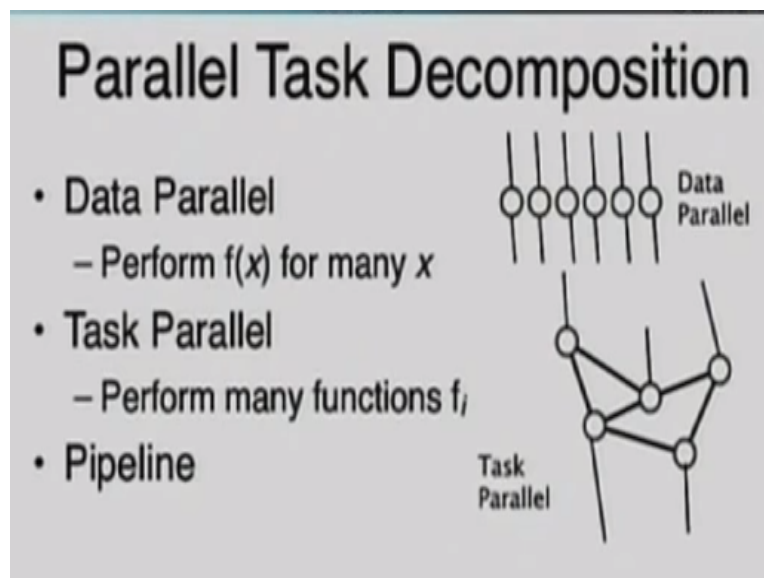
have independent execution stream and you for execution stream you figure out what is that stream is doing.

So you have got independent program so to speak but it really main streams which may be part of may or may not be the part of same program. Meaning same physical lines of code and then these thread either communicate using the communication scheme that message passing provides or they can communicate using shared memory okay. Or they may be some other system dependent inter thread synchronization techniques that may be available that can be used okay.

So you not always but very often break your program in terms of threads and these threads will start to use shared memory okay. And the other styles of parallel programming or technic of parallel programing is data parallel meaning that instead of these different thread that you write which do different things you write one program okay and that program is simply run on many depended items.

Because lots of different data things that you have and you want to take the first pair of them and second pair of add them so everybody addition so just adding different things so that is the data parallel style of programming.

(Refer Slide Time: 35:37)



So in many ways we can also break up your programming methodology into say either am going to be doing data parallel or task parallel programming okay in data parallel programming you

have some function that runs on different values of X at a logical level F is the program and X is what its input with us. So you have got the same F but a different set of X on the different processes different execution X okay.

And it has parallel which is in many ways in generic than data parallel but that also means more complex to manage you basically have for every processor you write one program you do that so there are five different for five processors five different function that you wrote and they do their own things they communicate they either communicate through shared memory we have to get all those they depend on each other if I am generating the data which I am going to send to somebody who is going to receive then that guy as to wait to receive.

Until the generate has been generated has been set okay so there are these dependencies similar dependencies can be in terms of shared memory if I write to a shared memory and I some how know that this write has to occur before that guy is reading it because that producing some piece of information that there guy going to consume then you stop that guy until production as happened okay.

So there are dependencies among these different tasks and the entire business of parallel program is to figure out what they tasks are or to design it is not figuring out mean that there is only one ways to break the task from to design what the task should be so that that there is less interdependence and then whatever the interdependence that remains needs to be managed by the program right.

So that whoever is depending on someone else comes after the someone else as to wait for the someone else to give you the thing that you are depending on you may proceed okay. There is also third styles which is more often associated with hardware than with programming methodologies but sometimes when the hardware is highly pipeline you use this also as a programming methodology which is pipeline okay where you do something even the task parallel thing where I have lot of different processors each running a task of its own.

It is possible for it to be organized in a pipeline meaning that I have done a task and I produce the results and give it to some known processor who is going to take the results of my computation and perform some other function okay do some other computation on it produce some results

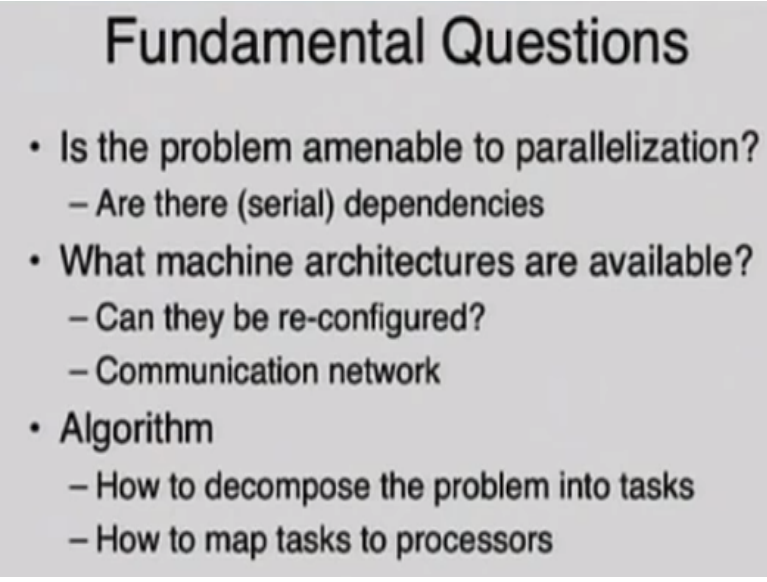
give it to a third processors right. At the same time when I was doing the first computation or a computation the processor down the pipeline was working on my previous results okay.

So there is parallel it is not that I do one thing and that guy is waiting for me to do it and I give it to him and then that guy starts and then I am done okay then there is no parallelism. But the parallelism in is in that the tasks or broken into sequence and there is different pieces of data that the different parts of the sequence are working on one related to each to the other by production and consumption relationship.

So I do something and pass on and then the new piece of data comes to me and I will pass that on so in some sense I as a one of the units in pipeline can be thought as a data parallel processors okay because I am doing the same thing on a stream of data I get a packet of data I do that I get next packet of data I do the same thing the third packet of data I do the same thing and it passing it on okay.

But among them it is task parallel because when I am working one piece of data doing one thing that guy is working on some other piece of data doing some other thing and the third guy is doing some other provides both it is a combination against parallel and data parallel.

(Refer Slide Time: 40:34)



Fundamental Questions

- Is the problem amenable to parallelization?
 - Are there (serial) dependencies
- What machine architectures are available?
 - Can they be re-configured?
 - Communication network
- Algorithm
 - How to decompose the problem into tasks
 - How to map tasks to processors

And so in terms of programming the main questions is that going to ask is how do I break the problem down into tasks given that I have so many processors and how do I allocate these tasks

to the processors I have okay. So decomposition of the task and then allocation of the task to processes and then if the remaining is managing the dependencies okay.

Although when you do this the same thing done on the dependency so that dependencies are reduced okay. And one of the things that you going to be very careful about taking a problem and saying let us make it parallel is to figure out where are the opportunities to do things in parallel if its completely a sequence of population that depends on the previous sequence previous operation in the sequence become a parallelism right.

And so at a high level although that is the main task figuring out how to break it down into task and making it to processors but all level have to also think in terms of work to break into tasks is it breakable or not so all of it ultimately depends on the dependencies. So if to consumptions if to find type operation because once you break into task you realize that dependencies and then you realize to few parallel things or parallel things that are dependent on each other and must do it too long and then you go back and refine it and continue that process okay.

So it is both so when I say that there are there are tools high level cost structure that helps you to do this at that level you do not as a programmer you do not manage it. Somebody else is still there is some programmer that is managing still it is being done at software level so if you are that programmer managing at this programmer then you are not okay. Lower situations when even this programmer this directly managing it not necessarily by saying it run this on that processors.

Sometime you do sometimes you do even that for example when you are architecture where you go a CPU and coder than you say run this piece of code on the coder ran than on the CPU okay. But at other times you simply say I have got these hundred things but I have got only five processors so I am going to say from these 10 things together on a processor these 13 things on some other processor and these 50 things on the other processor so I am doing this distributions okay.

(Refer Slide Time: 43:44)

Measuring Performance

- How fast does a job complete
 - Elapsed time (Latency)
 - compute + communicate + synchronize
- How many jobs complete in a given time
 - Throughput
 - Are they independent jobs?
- How well does the system scale?
 - Increasing processors, memory, interconnect

So these process of literacy which we are going to focus on in this course we will have to depend on few things one is dependencies which is very domain dependent you know from the application requirements not on depends on what so that is one thing. But ultimately aiming to do during this design and re-design and all have is to ultimately get these things running much faster than if you have to run as a sequential programming okay.

So how do you measure performance that would be first task one would be and this is probably the most useful measurement of performance when I say start when you give me the final result how much time is left result is also called the latency typically it is in many applications it is not just that you want to run this application at once. Let us say give me a piece of data share with your result done go home which mean off.

It is typically you are either analyzing stocks so piece of data coming in and you keep losing results or you are trying to figure out whether there is here or not. So you say you may data about this information I do this processing no idea give me some other piece of data okay. So there is turnover of the program that has to happen and so is not just necessarily the elapse time such that I give you one piece of data I will tell me when it got the result back but also the throughput.

How many times am I getting the data or result in a day? Continuously feeding you new input and you are continuously giving me an answer how many times did you do it during a fixed period of time? So that throughput which as a very interesting relationship ad typically not

always but typically latency go out as you put some time on and so on hence so forth is the other important measure that you would be interested in.

And so those together say of fast your program is going to run and then there is a issue of what would happen if these program to run higher number of processors, Because some point who want to run it on bigger amount of data and that has been trend for the last 20, 30 years that the service of input keep going up in fact now a days we collect a data astronomical data for example at a stretch of fast rate that even if all the computers in the world were to simply be looking at it for simply being used to look at it meaning pick that and write it somewhere.

That is called that's looking there would be behind the data collection because the data is being collected faster than all the computers in the world can read and write once. So this scalable is always going to be an important issue meaning that if I have to build a bigger computer would the speed latency through whatever you are doing also improve (()) (47:39) will it become twice as fast okay.

So when you are going to be designing the basic performance numbers in terms of latency or throughput is going to be important but also scalable. If you are going to say that on a machine which is which as happen to have lots of course 36 of that it scales very well that means from the sequential program I am able to speed it up by 36 times which is as best is as could be possibly every do.

But if it goes to 40 it is going to be 20 times faster it is not useful because it could not use of four and use on some other 36. So scalability is going to be important please as we study on how to design alright.

(Refer Slide Time: 48:34)

Simple Performance Metrics

$$\text{Speedup, } S_p = \frac{\text{Exec time using 1 processor system (} T_1 \text{)}}{\text{Exec time using } p \text{ processors (} T_p \text{)}}$$

$$\text{Efficiency} = \frac{S_p}{p}$$

$$\text{Cost, } C_p = p \times T_p$$

Optimal if $C_p = T_1$

Look out for inefficiency:

$$T_1 = n^3$$

$$T_p = n^{2.5}, \text{ for } p = n^2$$

$$C_p = n^{4.5}$$

The metric that we will be interested in example the latency will have to be related to typically to the sequential running time on the same kind of processor. So that you can say how much I am getting because of the parallelization okay and so often we will say if I have a one processor this will take so many units of time. So if I have N processor how many units time it is taken and that is the speed up that has been my algorithm is efficient because it is speed up is equal to the number of processors okay.

No in this we are not talking about memory data right this yes you can always look at the wall clock time. It is almost neither possible to do that but what you but at algorithmic level you can always do that. So when we say this say as a order and algorithm and you say order an on one processor and then because we say that RAM model does not matter we assume all the memory exist.

And they are all going to be access in the same time and we are going to do the same approximation is not the true we want to do same approximation here and ignored okay And speed up going to be important and will call efficiency to speed up divided by number of processors how well speed up match the expectation speed up. Speed up cannot be greater than P and the cost is going to be the number of processors you took to achieve that time okay.

And again this is going to be important because is when we talk at algorithmic level at module of computation complexity level then you say that I have a parallel algorithm that with N

processors on N size input will take \log in time. And somebody says I have a parallel program that with only 5 processors take square root of N okay which is better? How do you compare? The truth is that the locality will cover more than 5 processors and you will probably never have order N processors right.

So we will talk about how we will take algorithm it has I need N processors and used when you have only P processors. But at least in the (()) (52:05) purposes will assume this algorithm as a N processors sometime you say this algorithm requires N square processors or N cube processors and then it takes \log out time. And so to evaluate the algorithm we multiply the number of processors with the time and call at the cost.

So we are going to interest in the smallest cost algorithm if you take N processors and can finish in \log in time you cost is \log in but if you took only processors or only let us say K processors last in number of processors and finished in routine time then your cost is time okay. And again optimal the cost is going to be equal to the time to do complete this is in one processors.

And so this is one processors if in one processors takes N cube time and say you had N squared processors then you end up taking N squared root and time what I am get it. The truth is at the end of the day whatever machine is going to build is going to have cost on number of cost you will know how many processor you are going to run at. And the size of data is especially when you are talking about the high performance band large computation size of going to be extremely.

So one way to account for the efficiency of algorithm is going to figure out how much is caution number of processors. In spite of the fact of give algorithm assumes that then there are N processors N squared processors are N cube processors. And so we will all though process this in fact right code at take it from an N square processor and map it to a P processors machine in order to analyze there is sum up with idea of a efficiency okay.

(Refer Slide Time: 54:35)

Amdahl's Law

- f = fraction of the problem that is sequential
 $\Rightarrow (1 - f)$ = fraction that is parallel
- Best parallel time $T_p = T_1(f + \frac{1 - f}{p})$
- Speedup with p processors: $S_p = \frac{1}{f + \frac{1 - f}{p}}$

The other thing that is almost trivial but still something is that quiet often people forget is that speed up that you ever achieve is going to be limited by the pieces of code is cannot be parallelized and you will always have that. Something on depends on something else as soon as A depends on any depends on any B they have to be done one after other.

And whenever that happens so it talked about the dependency graph whenever there is a dependency graph you have sequential pieces of and whenever you are sequential pieces of code the maximum piece of code is down to it. So the AMDAL's law processes that if have got some program and a fraction of the program is sequential and the remaining $1 - F$ is parallel or can be parallel.

And so it is straight forward to come to the conclusion that you could never get faster time than it would take one processor to complete that program. Times $F + 1 - F / P$ because F remains however number of processors in have F is going to take F remaining $1 - F$ can be speeded up as $S \frac{1 - F}{P}$ that is fully parallelized obsoletely overhead $1 - F$ fraction if pre processors can be done in $1 - F$ over P time.

So the total time remains $F + 1 - F / P$ okay and so the speed up is going to be limited to that fraction on the board.

(Refer Slide Time: 56: 52)

Amdahl's Law

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

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

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

And as an example if just 2% of the coded sequential okay and then F infinite if I have as many processes you could ever have theoretically infinite number of processors. Then the speed up never be more than the 50 you can have 50 million processors if you wish speed up is rounded cannot go beyond 50 which just 2% of the code being sequential and that should come as a surprise.

(Refer Slide Time: 57:41)

Amdahl's Law

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

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

Converges to 0

$$S_\infty = \frac{1}{f}$$

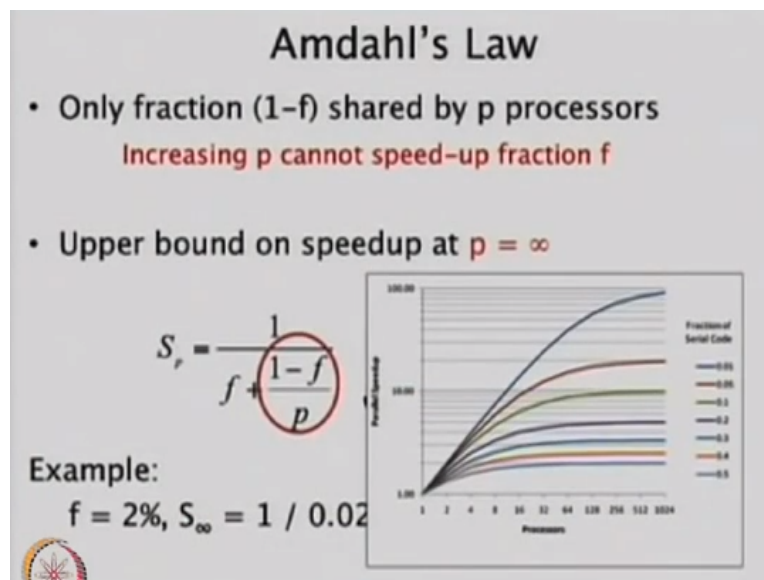
Example:
 $f = 2\%$, $S_\infty = 1 / 0.02 = 50$

And then basically the same function as before right on the when peak converges a P turns to be infinity as infinity is going to be run over F you can do nothing about it. It is a great parallel great code to parallelize the 2% of the code is sequential. So this 2% is not talking about static talking

about dynamic so you are saying that 2% is a very high number okay I will tell you 2 % is extremely high number it is somewhat high may be.

Any typical real world application is enough dependencies of one over the other that there is large block of code even if they are executed only once and that is not always true so that even often true there is enough of such code that 2% is not that high okay. It may be on the higher side not out of the ordinary right.

(Refer Slide Time: 59:33)



Here is that $1 / (f - 1 \text{ over } P)$ chartered out all this different lines you see may not be able to read it because it screen is lightly small. The different lines you see is different values of P okay and on the X axis processors is going it has only gone up to 1024 in this picture and the idea is speed up the best that you can get for a different number of processors is going vertically.

As the number of processors what did I say the speed up that you are going to get is going vertically the number of processors is on X axis right and the different lines you see are the different fractions a sequential part of code. So the line at the top is .01 is the fraction that is the sequential code 0.1% and now wait this 0.1 is the fraction and 1 % it is 100 okay. We are going to stop there now this is basically about the performance issue so far we will also talk little bit about the architecture style.

Another way of classifying architecture styles which also has a role to play in way you program this different architecture before moving on to going basically through a survey of parallel machines short survey of parallel machines so let me stop here.