

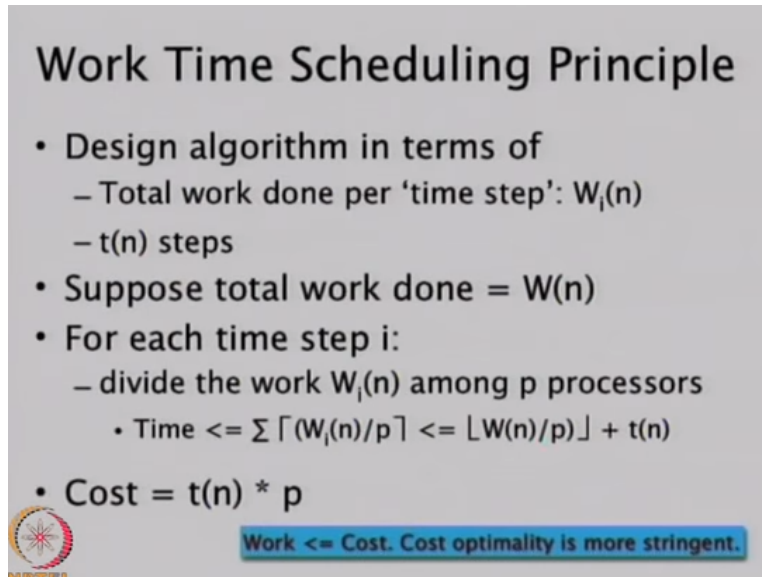
Parallel Computing
Prof. Subodh Kumar
Department of Computer Science & Engineering
Indian Institute of Technology – Delhi

Module No # 02

Lecture No # 10


Models of Parallel Computation, Complexity

(Refer Slide Time: 00:31)



Work Time Scheduling Principle

- Design algorithm in terms of
 - Total work done per 'time step': $W_i(n)$
 - $t(n)$ steps
- Suppose total work done = $W(n)$
- For each time step i :
 - divide the work $W_i(n)$ among p processors
 - Time $\leq \sum \lceil W_i(n)/p \rceil \leq \lfloor W(n)/p \rfloor + t(n)$
- Cost = $t(n) * p$

 **Work \leq Cost. Cost optimality is more stringent.**

Let us get back to this work time scheduling principle and principle essentially says that do not worry about the size of machine that you need. You take whatever is suitable for your algorithm for your problem and figure out how many steps are you going to take. A machine with that many processors how many steps is going to take okay.

Once you have designed the algorithm at that level at arbitrary large number of processors then let an automatic scheduler schedule for each step schedule the work that was done in each step into P sections and allocate them to P processors let them do their shared share or of that step and then go on to the next step. So you do not even worry about how to solve this problem on P processing machine you worry about solving this problem on however many processors think naturally fits the problem.

And then some compiler type (\cdot) (01:49) will give you the algorithm for P processor machine alright. So now the total work done for your original algorithm is going to become a part and this

is what we saw the last time also so now you accounting the step then the each processor each time step okay so W_I is the number of steps taken by each processor I at a given time step.

You take some of the number of step taken by processor they give scaling add it up that is total amount of work done right and we already saw that is going to be the total work divided by the number of actual processor you had plus the number of steps the original algorithm took okay you cannot change the number you cannot go below the step the original algorithm took. Because this automatic scheduler is doing what taking each step and breaking it into as many pieces which need to.

The minimum number of pieces you can break into is 1 is at least the steps okay so work done divided by $P +$ original number of steps is going to important that is how many steps it is going to take on your machine that $(\sum W_I)$ (03:31) P processors. And so T_N is not unimportant okay cost if you remember is simply the number of steps multiplied by the processors number of processor we used.

So cost is the time taken by your parallel algorithm multiplied by number of processor so cost is going to be more than work right. So work is the sum of the $(\sum W_I)$ (04:15) and cost is does not whether ideal on given step or not as am still going to count you for every step that I ultimately ended up taking okay so we can clearly see that work is going to less than equal to cost and so if somebody says it is work optimal it means that it taking work which is equal to sequential algorithm work okay.

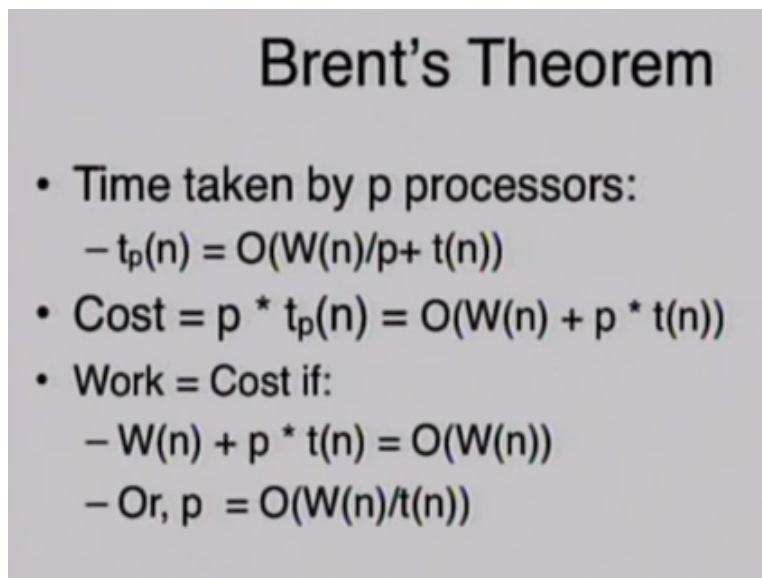
And we are talking about again absolute optimal if somebody says cost optimal we saying that the cost is equal to the number of steps that the sequential algorithm would have taken at if the cost so if you think back of example we are looking at last class summation how many steps did we under taking $\log N$ and the number of processors we used was order N so the cost was $N \log N$.

In spite of the work was going down because $N + 1, 1$ by $2, 1$ by 4 so work was only order N the cost was $N \log N$. So was optimal cost was not if it cost where also optimal it would definitely be

also work optimal right if I design a cost optimal algorithm then am done which always going to beat any algorithm and so there is Brent theorem that is based essentially on which we will look at the theorem in different way also.

But it is look at that WN over $P + TN$ is the total time taken right.

(Refer Slide Time: 06:10)



Brent's Theorem

- Time taken by p processors:
 - $t_p(n) = O(W(n)/p + t(n))$
- Cost = $p * t_p(n) = O(W(n) + p * t(n))$
- Work = Cost if:
 - $W(n) + p * t(n) = O(W(n))$
 - Or, $p = O(W(n)/t(n))$

If says that I have P processors not added the subscript P to signify that this now the time with P processor. T is T of N is what we had on last class work done divided by p work done of the original algorithm divided by P plus the time taken by the original algorithm you would want those two to be as balanced of processor right. So the Brent's algorithm says Brent's theorem says that you want WN over P to become equal then were order is going to be the smallest.


At otherwise you order is written by the whoever is the bigger okay TN is bigger than it is order TN WN over P is bigger WN over P . So if you want to make sure that neither of them is bigger we will bring them to same level okay. You say $WN + PN$ P times TN should be ordered WN that is the case then T should be ordered WN over TL that will be the optimal number of processors okay.

So now let us look at the optimal the summation again but this time we are going to make cost optimal.

(Refer Slide Time: 07:36)

Optimal *Summation*

- Using n processors for parallel sum
 - Work = $O(n)$
 - Cost = $O(n \log n)$
- Suppose, we use $n/\log n$ processors.
 - pardo:
 - Computes sum of its share of $\log n$ local elements.
 - Compute sum of $\log n$ partial sums
 - Using previous parallel sum algorithm (with $n/\log n$ processors)
- Time = $\log n$
- Cost = $n/\log n * \log n = n$
- Often useful to have large sequential sections
 - as long as there are enough of them to keep processors busy



Reminder work was order N cost was $\log N$ but now we do the same thing but we do everything in parallel all time. We have to make few things sequential we are going to start with algorithm that says we have got not N processors for N over \log in processors okay. Now you are going to add N things using the N over the \log in processor so what is the algorithm?

You take $N \log$ algorithm each processor takes one it share equal share it adds them up together how much did it take each one as $\log N$ right. N by \log in processors distribution N things each gets $\log N$ then $\log N$ time you are able to do yourself look yourself. Now you are going to do the same thing I did before because I got this N by $\log N$ partial sums and then I need to add it together.

I am going to assume N by $\log N$ processors of this part again that is what I received for first part also how would N by $\log N$ processor do? N by $\log N$ processors, N by $\log N$ things same as before right the total number of stress will be \log of N over $\log N$ which is same as order $\log N$ total number of steps is second half is also $\log N$ total number of steps in first half is also $\log N$ total of steps as remain $\log N$ okay what about the cost N by $\log N$ processors $\log N$ steps cost is the product right.

So cost now becomes order N so slight change to the algorithm will give you a cost optimal summation algorithm you said I am going to take assume a different function of N as the number of processors and this is algorithm it takes goes into two steps there is local computation and there is local computation and at the time of it you got the N okay. It is still few enough processor are ideal few enough steps that they do not $(())$ (10:07) at the end they do not go into the cost.

And so there is a lesson here says that if you can somehow break your task into to doing some independent things first. Recently large parts can be done in a sequential way independent of other processors relatively large parts when you are likely to gain as long as you large part is not so large added takes up all of the computational and every other processor is ideal as long as N you can break it into a chunk size that is big enough to do significant amount of work completely locally sequentially and it small enough so that there is enough of them to keep many processor is busy.

(Refer Slide Time: 11:07)

Notions of Optimality

- If sequentially optimal algorithm is $O(t'(n))$
 - Work done by **Work-optimal** parallel algorithm:
 - $O(t'(n))$
 - Work-scheduling on p processors takes time:
 - $t_p(n) = O(t'(n)/p + t(n))$
 - Optimal speed-up $t(n)/t_p(n) = \theta(p)$, if
 - $[p \cdot t'(n)] / [t'(n) + p \cdot t(n)] = \theta(p)$
 - $p = O(t'(n)/t(n))$
- **Work-time optimal** if:
 - $t(n)$ cannot be improved

You are going to get a fast arrival okay and so that brings us notions to the optimality optimal is something that takes amount of work that is equal to the best sequential algorithm okay. Work optimal does not mean it is going to be the fastest for every machine right because on some machines the number of steps becomes importance. If I got a big enough machine if I had said I

assume $N \text{ over } \log N$ processor and give me $N \text{ over } \log N$ processors then why do I need to care for cost or work.

Such that steps is how much work is going to how much time is going to take because I have no need for sub dividing work among P processors that each step and so this is another way looking at the same thing the time so work optimal parallel algorithm takes time order of $T \text{ prime } N$ right which is the same as $T \text{ prime } N$ is taken the sequential algorithm can count so work done by the work optimal parallel algorithm is the same $T \text{ prime } N \text{ big } O T \text{ prime } N$.

If you schedule it on P processors now if you only have P processors then $T \text{ P of } N$ is the same as before right is nothing in it $T \text{ prime } N \text{ O } P + T \text{ over } N$ and how will you get the best speed up you know the time you are taking which $T \text{ of } T \text{ P of } N$ you know the best time of algorithm would update the prime of N your time rather their time divided by your time is the speed up and optimal speed up would be P .

If I have P processor I would like to go P times faster than the sequential algorithm and so am going to relate that to P right. The speed up which is in the square brackets over there which is the ratio of the two different times should be theta of P should be in order of that ratio to be theta of P you can do a little bit of math T is going to be the ratio again Big O of the ratio of the time taken by the best sequential algorithm divided by the time dual expecting the parallel algorithm.

This is same thing is before but it says that P is big O of as long as P is you do not have more processors in that you have got an optimal also cost of optimal solution it is work time optimal that is work optimal as well as time optimal if you cannot now improve $T \text{ of } N$. $T \text{ of } N$ is the time you would take in your parallel algorithm okay. So that brings theorem applies to it is possible to do that to anything and so there exist as a range of values for which the given algorithm is really good.

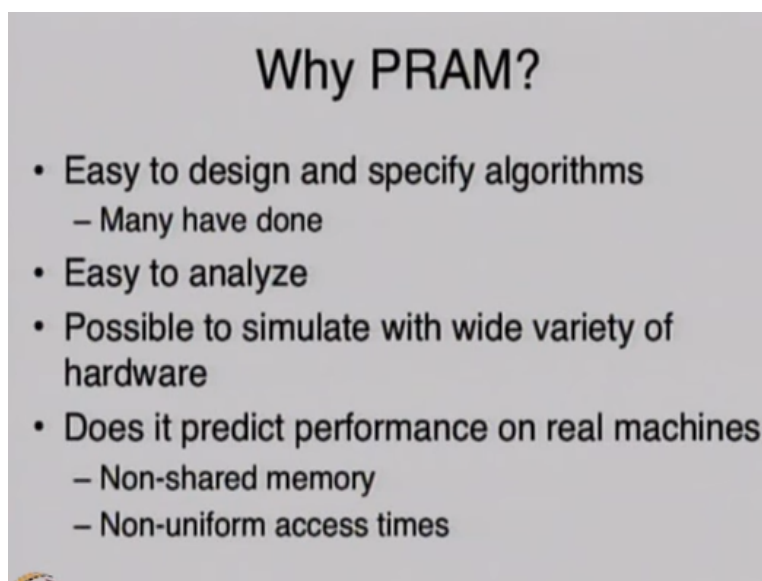
And beyond that we have more processors than that that algorithm is not the best for example again coming back to the summation example if I break it into two parts that is only good as long as P is no more than $N \text{ over } \log N$ it can be less actually just work it out if you had $N \text{ over } \log N$

wired N processors it was still be okay if you have enough processors if you have large number of processors then at different algorithm may be one that keeps those as many processors as might have busy okay.

So again said this before also work optimality is the main thing will be focusing on them if we can come up with something work optimal we can worry about making the time better if we can you will generally not be able to get so if I have summation which can be done in order in time with large enough of processors P I would like it to be $P N$ over P time right if I have N over $\log N$ processors then I know it takes $\log N$.

But if I had N processors I would like it to run in more time unit time and that would not be possible okay and we will look like complexity classes let us say what is what not defined complexity classes in the parallel context T^* so there is a type of error that should have the P prime it is not T^* let me fix it alright.

(Refer Slide Time: 17:04)



Why PRAM?

- Easy to design and specify algorithms
 - Many have done
- Easy to analyze
- Possible to simulate with wide variety of hardware
- Does it predict performance on real machines
 - Non-shared memory
 - Non-uniform access times

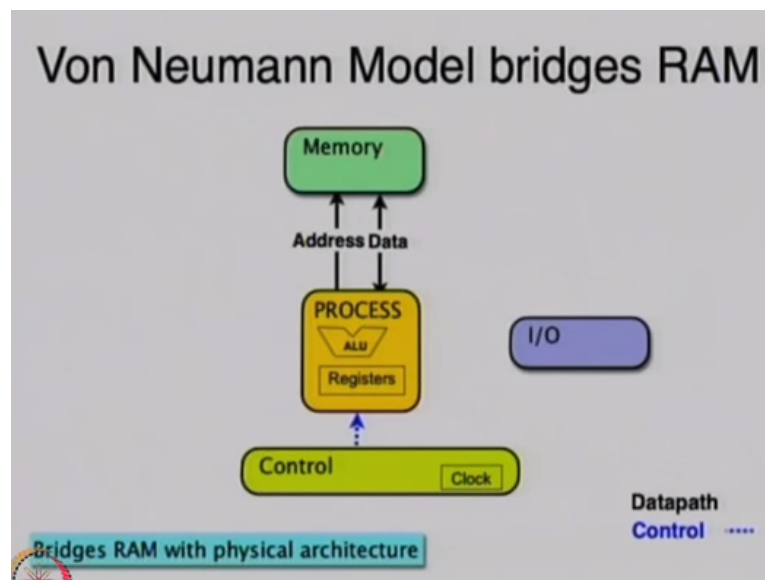
So we have talked enough about PRAM in summary in short it is just a computation model which helps you argue about things reason about things it is something that makes easy to design and specify your algorithm not just like that because it has been processor there are lots algorithm available for it. So you can use them as sub routine into your overall algorithm much more easily.

It is also not exactly what hardware does right does not have the constant time shared memory access and it is words that RAM. RAM also does not have constant time main memory access in the differential RAM and predictive performance and actual performance you have observed in tactics and you actually run things can be sometimes large but here it can be even bigger and because keeping that the keeping the impression that shared memory access is going to the same under time going is a bit hard.

So in any case people have looked at various other types of computation models of computation for parallel computation and one and we are not actually going to talk about most of them because of some level they have some correspondence with PRAM okay their may be a log factor for any given algorithm assume PRAM and we assume something else because they can be simulated with respect to each other in log and time or each step of other models can get in log in time in PRAM.

If we are going to get a cost that is $\log N$ times that other models cost so the differential is going to be that okay.

(Refer Slide Time: 19:21)



But we will discuss one thing which is often called the bridging model of parallel computation okay. Which is originally designed to kind of parallel VON NUEMAN architecture. VON

architecture is a bit more count three than RAM the RAM model of computation it is still not real computer but over time I has come to represent pretty much every sequential real computer which says that there is a processor there is a memory it is I/O stuff there is a control path and what happen to my control path okay.

And programs are stored in the memory you take out the program you execute the instruction of the program which is done again by sending some information into the registration or the CPU and all that. And RAM model is just one level of instructions because of this one RAM model is pretty representation of real computer it is kind of bridging model if you see that PRAM is not actually helping out in predicting performance you can get to VON NEUMANN model. In this case there is also richer version of VON NEWMANN model where memory is arrived.

(Refer Slide Time: 20:46)

Bulk Synchronous Parallel Model

- A set of virtual (processor, memory) pairs
 - Processors do not have a particular order
 - No notion of “locality” in mapping to physical processor
- A point to point interconnect
- Barrier synchronization (all or subset)
- Repeat “super step”:
 - Local computation
 - Communication
 - Barrier synchronization

Designed as a Bridging model for parallel programming

22

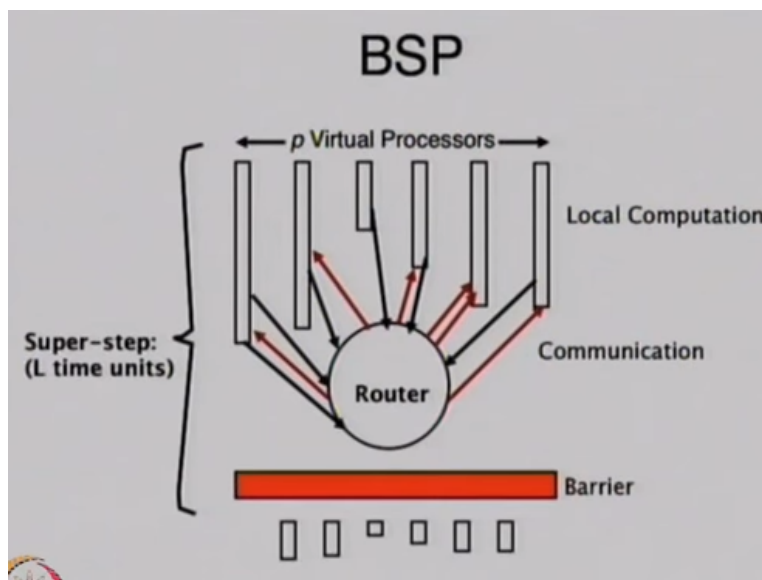
And so the attempt to do that in parallel context is this called bulk synchronous parallel model on computation. VON NEUMANN is essentially it is something like sits between more abstract that RAM model and a more real model. So one is that it helps the computer designed restrict their design space so they keep their design similar to the one line. And the other is that if the RAM model is not predicting your performance you can get down to what one by one would have done.

We go to one level deeper because you that actually leading you towards the real hard one more components of the real hardware and so the role of the VON NEUMAAN model was expected to be played by this BSP model okay. But overtime it is also become a model of computation model of parallel computation which takes away that idea that all shared memory access can be serviced in a constant model and the idea is that you got a virtual set of processors again you can assume as many as like and each processor have some local memory and there is a network.

Instead of the shared memory there is a network that lets your communication make something to run the processor okay. Again the distance from processor to processor is not expressed model here so it says that if you are sending many units of data to any other processor it is going to take whatever your network P times N . And so there are other variance is that were distance is taken into account for we will leave at this where it point to point interconnect and there is this notion of super step which is very similar to PRAM.

In the super step you do some local computation okay you do some communication and then there is barrier synchronization at let okay and once everybody reaches the barrier they start the next super step.

(Refer Slide Time: 23:40)

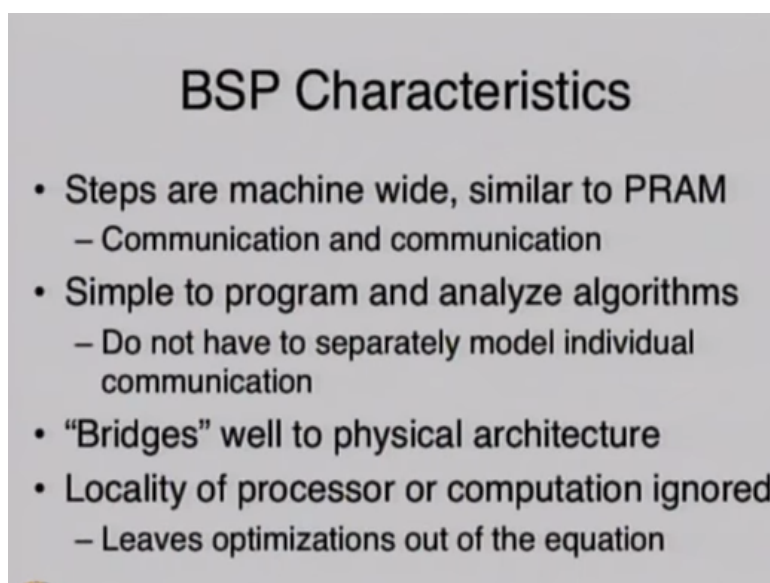


So the kind of looks like this you have got bunch of super processor running at some time during their run do some communication with other processors which means you send something to the

network so that address to your pocket and the network will deliver it at some point you do not know when to a destination processor okay. You may not assume that guy as received something at any given time or a receiver may not assume that I can start use it because that value as been updated in my local memory.

And that is where this barrier comes in after some amount of computation local computation I am done my local computation everybody says that and then they are going to synchronized by the underline computation model (()) (24:41) and at before that received from the barrier all the messaged have to be delivered so in the next super step we are going to have they can assume that data has been delivered or the data has been received so they just collects the data different pieces of data there is not notion of clashes.

(Refer Slide Time: 25:10)



BSP Characteristics

- Steps are machine wide, similar to PRAM
 - Communication and communication
- Simple to program and analyze algorithms
 - Do not have to separately model individual communication
- “Bridges” well to physical architecture
- Locality of processor or computation ignored
 - Leaves optimizations out of the equation

The number of super step size is set to (()) (25:14) and that so there is multiple ways when which you can figure out how expensive given algorithm. You can simply count the number of super steps so you design your algorithm in terms of super steps there is facility that you may decide with you may not with super steps yet so al the processors are doing something and they have determined that L is step size it is fixed number and after else somebody says am not anger.

Then somehow the global state is accessible to everybody and everybody knows that the super steps did not get done and next L time units their on the same super stores so those who have

done got have to remain active so that you simply count the number of super steps (O) (26:15). Number of detailed counting that is subsequently done will talk about that also but essentially the idea is that let us going to be similar to PRAM in that it is simple to argue simple to design for and the cost that you are going to ultimately come up with will be little bit more reflective because you are going to ultimately count for the number of messages you are sending okay.

It still does not talk about the distance does not talk about the details of how many messages you are sending right if I says that you are sending some number of messages of some fixed length okay. So the account of the messages all that accounted for if you say that I have and you can simulate you say suppose I have to send messages of different sizes then it says fine you can send K messages of one unit or one message of K units the same thing to me okay.

Or K over two units of two units each so you simply figure out how many one units you got and that is accounting and that is not always the true because there is latency notion of latency involved right you say am going to send a message that this much you start sending there is some overhead to the beginning of the send and then the messages get delivered quite fast typically okay.

So there is fixed overhead so if you have lots of small messages it will take usually longer large messages those kinds of details are not accountable.

(Refer Slide Time: 28:12)

Interconnect

- Realize arbitrary *h-relation* in a super-step:
 - VPs send (receive) at most h messages
 - each of length 1 (same as 1 message of length h)
 - Communication cost: $gh = th + s$
 - t : network throughput
 - s : constant overhead, small
 - Ignores message count, distribution

So more detailed account is with this notion of H relation and it says that there are H messages being sent in a super step and the network has certain speed it has throughput it can deliver G messages or in a time okay. And you figure out how many messages how long it will take to deliver H messages okay and you figure out how much time each computation is going to take and you figure out how much time is barrier going to take okay.

And so here you are seeing just the communication we will look at the other aspects of the accounting and the communication part basically says in short there is one scale parameter that relates it to how fast computation is versus how fast the network throughput S and that value is G . So if you are sending H messages in a super step then you cost is H time versus G . G is constant for the machine H is what the algorithm provides.

It argues that I am accounting for the overhead the startup cost because the startup cost is going to be much smaller than GH and hence that subsume into GH and so GH is good enough indicator how long it will take a real machine and again as a I said ignores how many messages what size the messages where who you are sending the messages to all of them that is not part of the accounting.

H is the parameter that the algorithm is generates right so if you are sending H messages then the cost is H times G we have this three end of time there is a little bit of accounting of we are going

to continue that on Thursday before moving on to more practical techniques how you use PRAM to solve some common programs as well as some more higher programming decomposition techniques how to take a program and make it parallel figure out how to turn into a parallel program.

So we will stop here any questions the idea is to figure out how to how it is going to call you we will do that shortly. Understand what the model is and what is the cost model is right so this I will revise this slide which essentially says that there are some number of processor memory players right we call them virtual processors of virtual or whatever and then there is a network that network is how processors communicating each other at the communication is point to point.

So processors says I want to send this data to that other processors alright virtual processors is the means that these are as in the context of PRAM right there were any processors that you are wanting. So same sense here also so many number of processors that you want which can be limited with P processors okay so in the context of the model you do not have even have to think of it in the virtual processor.

It is a processor memory player I mean network right any number of processor may require and the network. And you do your local computation and that is not in long step but after a certain period of time you are going to synchronize with every other processor okay. So there is a barrier which says that all the processors have done if their local computation starts okay. if they are done with the local computation step that also means that they are done with all the communication that they were doing during the local step.

There are variance in the model there is a variance you do local step then you do the communication then you wait for barrier there is no theoretical difference between that versus then the overlap communication with your computation and then wait for the barrier and they implicit assumption is that only after you come out of the barrier can you assume that any of the communication that you did before the barrier as been completed okay.

So the data that you sent has either been received or the basically that the data to be sent as been received or the data that you are waiting to receive is here somebody must have sent okay. For the cost model which is where the intend of this model was to make the cost accounting the little bit more precise breaks things down in terms of how local computation you do how much it is being sent in communication of the data and how much time you spent on making sure that everybody comes to same sequential point.

(Refer Slide Time: 34:38)

BSP cost Accounting

- Barrier cost of l
 - fixed for a given count of virtual processors, p
 - A lower bound on l is the diameter of the network
- Super-step cost:
 - cost of the longest running local computation
 - cost of global communication
 - cost of the barrier synchronization
 - $w + hg + l$ (Non Overlap model)
- Total cost = summation over steps:
 - $\sum w_s + g \sum h_s + Sl$

So there are three components of the cost and those are listed here there is a fixed cost associated with the barrier given the number of processor so more processor need that more of fixed cost but the once you determine how the processors you have cost may consist. So the fixed cost for barrier and then there is local computation step however number of steps that the largest of the computation takes.

Because everybody need not do the same amount of computation takes because everybody need not do the same amount of computation between barriers so that is the super step and communication. And communication is based just purely on the number of messages that been set okay. So there are H messages being sent in one super step and G is the throughput of the network then H times G is the cost of those messages right.

Throughput is essentially a relation do not think of it use of the bites per second the standard measure throughput it says how many units of network delay or network time is related to one unit of processor time. So that you can add in together then you say am taking W units of processors at time I cannot simply say I am taking H units of message time. G relates how much the processor time and message time the ratio processor time versus message time.

So you bring everything into processor for example to bring everything into this context everything into processor block simply to speak. So G is the multiplier between the message clock or the network clock or the processor clock and then there is a fixed amount L for barrier which is again given in terms of processor clock. So you add all of this together and you got the total amount per super step and you add it cost all the super steps and you got the cost.

How long a given algorithm on this model is going to take okay and this is where the non-overlap model simply adds up $W + HD + L$ in an overlap model you would say that some W may overlap with HD let me say \max of $WI + \max$ of HIG and then you add the those together and so there are variance of it but at syntactic level that is not going to make much difference.

So now let come back and compare H2P RAM okay what is the benefit of the model designed or is it re remote powerful than PRAM not necessarily. It just means that we need to be able to send messages from point A to point B right you can do it on internet. As long as their reason network which connects every pair through whatever problem that the apology of the network may be and as long as there is local computation to capacity right and here it does not says that local computation is essentially a RAM amount of local memories unlimited that not exactly between true.

At least at the level of computation model we never say that we are out of memory okay this is basically taking the communication cost right the RAM model we said it is every step so the in at high level there is lot of difference NUENN model every step was synchronized here the big set of steps which then causes size of the barrier to be encounter after which there is big set of steps.

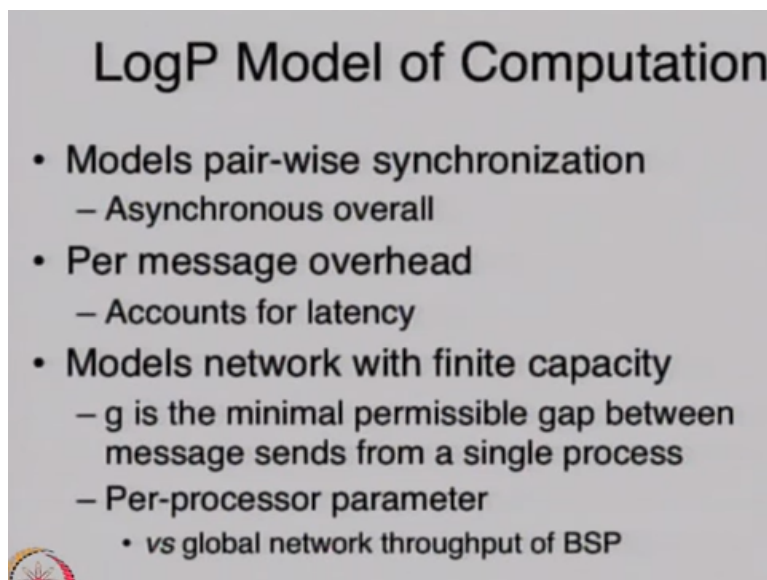
If you think about it PRAM model can also be sort of at the same way but big set of steps is one way because everybody is waiting for the barrier it is like the have to spend time for that step it is on the barrier becomes that sync point.

Instead of each local computation being what in the context of PRAM it used to be read something from the shared memory do some local computation right something to the shared memory it has become do some local computation decide what you are to read and write to shared memory do that and then start the next step okay.

So at that level it still remain synchronized right distribution basically long step the step has been expanded into here is communication here is local computation here is some barrier. And so hopefully little bit more realistic but capacity wise not really adding this much and in other words whatever you could you might be able to do on this model you can simulate on PRAM okay.

There is another model which is again going in the same direction further just like in PRAM you said there are synchronization version of PRAM.

(Refer Slide Time: 41:00)



LogP Model of Computation

- Models pair-wise synchronization
 - Asynchronous overall
- Per message overhead
 - Accounts for latency
- Models network with finite capacity
 - g is the minimal permissible gap between message sends from a single process
 - Per-processor parameter
 - vs global network throughput of BSP

Here also there are a synchronization of BSP and the one that is the most commonly used mostly in theoretical context right so this not something that lots of people implement as a programming

model in the. PRAM again is not something necessarily that is implemented as a programming interface directly. So these are models where you can specify an algorithm reason about it but then you have to see how it relates to an actual programming interface and like an open MP interface okay. Right but the cost will may be different the number of messages you sent there you can say that one step took one unit of cost here you say one step takes some constant or is it some constant plus H so if that H is dependent on the number of the size of the input then the cost will be different.

There you said in one constant unit of time I was able to get this data clause here is I take log in time to send this data clause. So that log in there in the cost so the cost will be different so what you could have been able to do you still be able to do. It need not be more right if it is if everybody is sending constant amount of messages if every processor sent one messages every super and it is the same is the same cost.

Performance of two algorithm right and so that is one aspect of it and other is being able to describe the algorithm. And so this is the log P model which is synchronize version of with the BST model where you do not necessarily say everybody into a barrier there is pair wise synchronization I and that processor are going to somehow come to conclusion that we have shared our data okay.

And this is not very dissimilar from MPI that you will get shortly get to where we basically say I send you something you receive something and there is some handshake that happens that I know you are receiving okay. Until you receive it am not doing anything else I am waiting there it is like a barrier I get pass the send call when you pass the receiver okay. And here also for the cost accounting there is a latency added which is a minor change from the earlier case you simply said that there is so many units of time spent in sending a message of unit length.

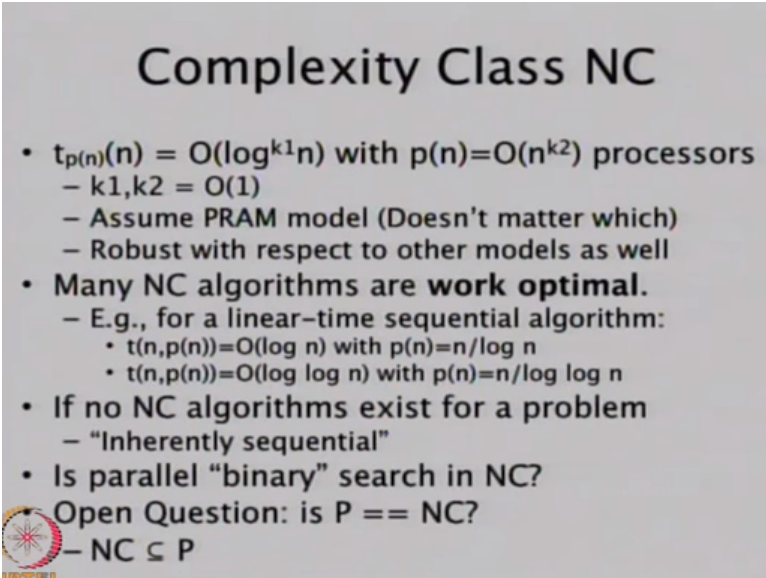
Here you say that in sending message the time spent is fixed latency because latency is the time that he will receive the receiver will receive the message at after I have sent so some throughput level is says I can send G messages per processor clock time does not help you. So size of the messages becomes a model there is no necessarily notion of mass training so those are

implementation issues and they can be variant of MPI you do not need master mode just a way it is in symmetric.

And in the air there is no notion of master note their all equal the only thing that has really changed is that we do not come to the barrier together I negotiate my synchronization with every other processor in the (Π) (45:19) okay. It is called $\log P$ because of the things it introduces to uses in the computation so L is for latency it has nothing to do with $\log O$ is sum symbol that is used for I think the local computation time G was the network and P was something else.

So it is just (Π) (45:48) for what the cost model okay so there are many more and then we will do course on that which I do not intend to do.

(Refer Slide Time: 46:07)

A slide titled "Complexity Class NC" with a list of bullet points. The slide has a light gray background and a small circular logo in the bottom left corner. The text is black and uses mathematical notation for complexity classes and time functions.

Complexity Class NC

- $t_{p(n)}(n) = O(\log^{k_1} n)$ with $p(n) = O(n^{k_2})$ processors
 - $k_1, k_2 = O(1)$
 - Assume PRAM model (Doesn't matter which)
 - Robust with respect to other models as well
- Many NC algorithms are **work optimal**.
 - E.g., for a linear-time sequential algorithm:
 - $t(n, p(n)) = O(\log n)$ with $p(n) = n / \log n$
 - $t(n, p(n)) = O(\log \log n)$ with $p(n) = n / \log \log n$
- If no NC algorithms exist for a problem
 - "Inherently sequential"
- Is parallel "binary" search in NC?
- Open Question: is $P = NC$?
 - $NC \subseteq P$

What I will do though is relates when you read RAM also read about complexes theory you understand what a (Π) (46:18) is behavior means whether some problem is inherently hard can we make some fast solution for something you will all get some flavor of the for the parallel context before moving on to some more practical things so there is various class called NC okay.

It is a (Π) (46:53) again NICK's clause So Nick was the name of the author of the paper to propose this first and so it is started to become no one next clause and it says that nick clause is the class of problems where using a polynomial number of processors you can solve the problem

in parallel in poly log time okay. SO log raise to some constant N so it is says $\log^{K_1} N$ time is what you will take if you have N range to the K_2 processors where K_1 and K_2 are except it is constant but a good parallelizing program will have those small constants.

So the class NC is the class of algorithm that have the poly log parallel solution if you give a polynomial number of processors okay. How do you think this would relate to P? P is the polynomial in PRAM model and it will take polynomial time it is still polynomial right and anything there is NC is definitely NP there is if you took that problem can solve this sequentially you can do it in polynomial time total polynomial time okay.

Is everything in P and NC it is a local just like we still do not know whether $P = NP$ we do not whether $P = NCI$ okay there is a talk on just the topic next week which discusses some recent developments alright so the idea is that if you do it in parallel it should be do it in faster than polynomial time okay. R is worth going to be an issue for NC algorithms before I get to that so I think I already said this but it is worth repeating NCI algorithm basically says that it is parallizable algorithm right.

It is not an inherently sequential algorithm or problem if a problem is inheritance sequential that means you can use as many processor you like you are not going to get great speed up on it. We are not found such a problem the problem may be there we may not have found the solution that now as of now that state of the art is that you cannot find a efficient parallel solution for it.; We do not know that does not exist one okay.

So if it does not belong cannot prove that it is NC that means it is for the time being inherently sequential and this brings us to the question that some of you are probably thinking of right now is binary search. First of all is binary search NC even one processor can do it in log in time okay so it is definitely NC is it inherently sequential meaning can you get reasonable speed up from the parallelization and the answer is in this case no.

And so there is basically shows the limitation of the definition answer is pretty much no we are talking about binary set algorithm not search in general. The definition let us get to that

discussion we will shortly the question I asked was or rather the answer I was given was NC model is not necessarily an over-arching definition that says if it is NC it is good in terms of being parallel is not in NC when it is not good.

Not an NC probably means not good but just because of NC does not well parallizable because the research already happens to be a log problem and s this definition that says I can find polylog solution it only work if you cannot find sub linear solutions in a sequential word okay let me stop here.