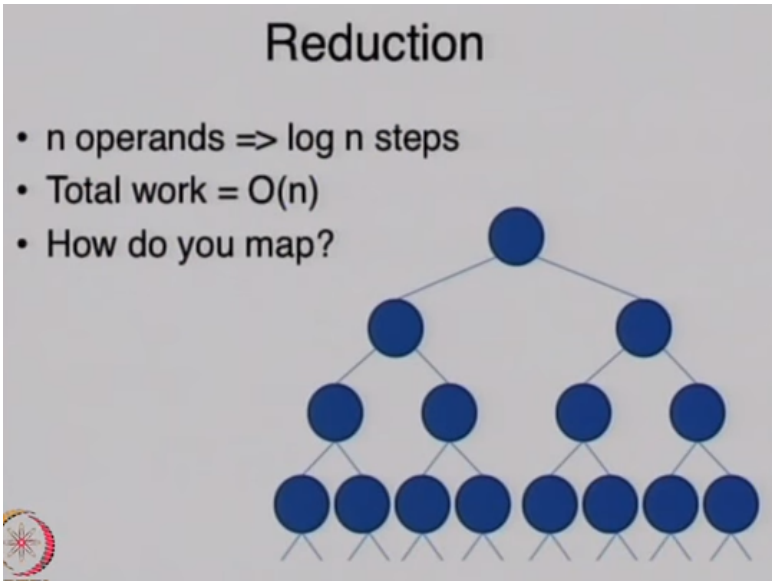
Parallel Computing
Prof. Subodh Kumar
Department of Computer Science & Engineering
Indian Institute of Technology – Delhi

Module No # 04
Lecture No # 18
Algorithmic Techniques

Okay so we are going to talk about algorithm techniques in general although I will be taking up few specific algorithm but will also be categorizing that. So this that style of algorithm development and the other one is the other style of algorithm development. So the first one is something we have seen lots of time already binary tree style reduction. So reduction is a binary complete binary tree sometime we say balanced binary tree sometime we say algorithm technique.

(Refer Slide Time: 01:07)



And it is essentially this is a logical tree of computation and you can imagine having data at the leaves with every node performing some computation for this disruption we are not focusing on whether it is open MP style or MPI style or any other style it is just parallel threads may be on the same machine may be sharing.

Some memory may not be sharing memory may be communicating somehow but this is at higher level of logic and every level you get some information some output of computation from your

children you perform some more computation on it and produce an output for you parent okay. And when this reaching up to the level of the root then the computation end result of the computation of the root is the final result you can reversal it you can say root does something then it provides some input to it children they do for the processing and provide something more input to their children and the output at the leaves together is the output of the system okay.

Anything that maps to this style will typically have log N steps of whatever size so assuming that each node is doing order one computation and you can change order 1, 2 anything there are log N of order 1 computations right because until the bottom of level is done its work next level cannot begin and all of them of that in parallel but the next one cannot be run in parallel with the previous one so after one unit of time the next level of node speaks activate after one further unit of time the next level activates and after log N unit of time the root has activated or vice versa .

Total time is order N order log N and total work will be order N right assuming again order N times whatever the individual computation so if everybody is doing order one taking order one amount of work to do then total amount of work will be N right. And the remaining question remains is of course this is you are using order N noes to process order N things out of produce order N things.

In fact 2 N nodes right to process N things and you will probably have N processors not twice the size of input and so what would be a with to map these P processors okay. So similar to work scheduling that we talked about earlier what would you need to do how what does work scheduling way? What should you be doing? Everything that is done in a unit step is distributed to the processors available so every level is going to distribute to the P processors and at some point the number of nodes is going to become smaller than P and then some processors will be idle okay.

So something like this you have a processors 0, 1, 2, 3, 4, 5, 6, 7 performing the leaf level leaf node so to speak computations and then in this particular 0 to 4 onwards are doing next level could have said 0, 1, 2, 3 half of them are going to do then at the next level half of them are

going to do right and so once the number of processors becomes bigger than the size of the width which you do not okay.

In some situation it would make sense to use specific sense ones will look at such a situation very shortly and so you would not want to necessarily s this is at because of the kind of wide spread use of reduction in because of some flavor of reduction in pretty much every algorithm we will come across there will be a tree somewhere along the way okay but this is saying that tree is basically your entire computation okay.

So again this is a point I made before when we categorize things it is not just to say that your you have a problem and then you are going to say whether this category of algorithm would suit this problem or that category it is more like these are different ways in which I can process the input see where I can combine what so it is not necessarily that clean of a decision to make okay alright.

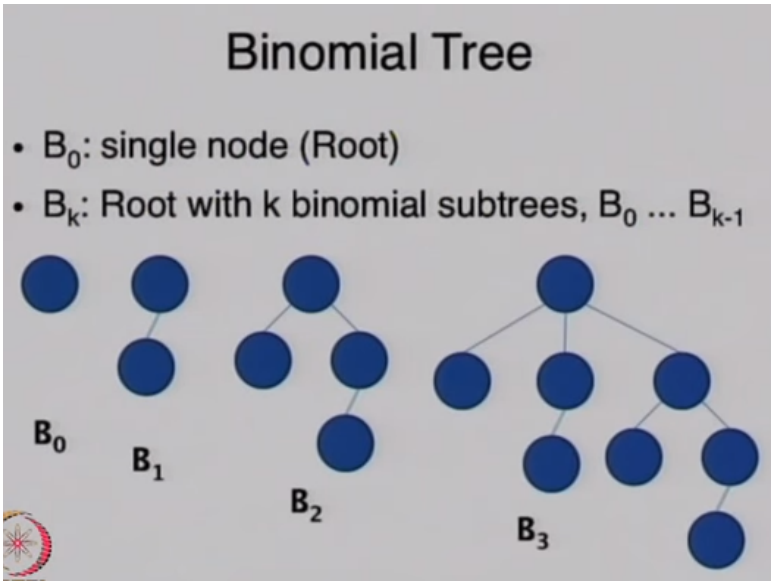
So in this case I am showing an example where at the leaf level you have more processors so 0, 1, 2 and then next level only four nodes to get 1, 2, 3 and then you start skipping that you had a question it can be right. So in this case if there is for example let us say to take that example we are summing things all right and you are basically taking two things adding put producing results and that all each nodes does then all those extra processors at the top level am not going to buy you anything and you can do nothing with it.

But if they are doing nothing are they costing you anything yes it is keep doing some work right but it is one of the nodes that is keep doing nothing probably and so at least somebody else is slowing down because you are running some small load MPI there is no reason for that particular restriction to be universal there is no reason that MPI had to implement that you just easier to do certain things when uniformly and so but I think that is a small overhead that is not worth worrying much about okay alright.

So now in so thinking back in the terms of dependency graph and all that given that this is the computation we are going to have to worry about when a given node is ready right when does it

start its computation. So who are the predecessors in the dependence graph okay and that forms what is known as the binomial tree among all the processors that are available okay. We are not going to go into the details of dependency graph yet but just as a piece a piece of information it is something useful to know.

(Refer Slide Time: 10:54)

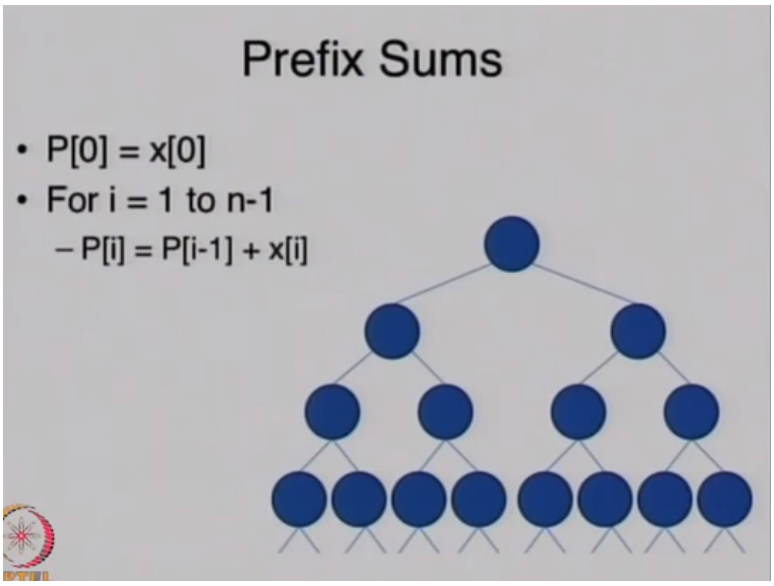


A binomial tree is a tree where number of children keeps growing as we go up the tree. So recursive definition of a binomial tree that if you have a single node it is binomial tree call it B0 typically single node binomial tree just the roots and BK is a tree which as tree which K binomial trees as its children okay. So B1 is root with one binomial tree as a child which is B0 is single node and similarly.

So take an example that B0 that is B1 right B1 as root with B0 at this child and this is B2 which is root with B0 and B1 as it children and that is the root with B0, B1 and B2 as its children right. So how it is related to the dependency? If you go back and look at exactly the top 0 depends on 2 its children as well its children its side okay. So if you expand this out you are going to get this one exactly this morning by binomial tree okay.

So now let us talk about somewhat non trivial or non-obvious let us say algorithm that uses this stream reduction is one right we have looked at that prefix some was the other thing that I have mentioned in we are going to talk next as the algorithm.

(Refer Slide Time: 13:03)



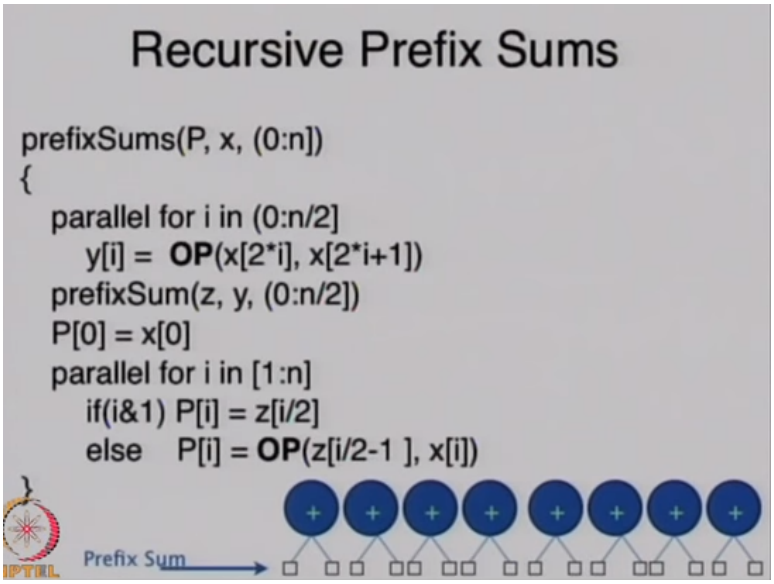
So this is the definition of the prefix sum is just to repeat I will leave that up there in the corner we have got a set of number and this written in a PRAM style you can quote it to openMP or MPI whatever your wish X is where it is an array where the input is given to you and P is an array that you are producing which contains the prefix sum all right. So for the first element of the resulting array the first element of the simple array at it P0 is X0 and P1 onwards is sum of all the elements in X before that okay.

So P is 7 is the sum of X0 to XN and P10 is sum of X0 to X 10 okay how would you compute it efficiently so if you look at the dependence tree it is very clear that each dependence on the previous computation as this loop has been written off which means you are going up to repeat something replicate something so that this can be done is parallel something else an if we can map it to the binary tree style computation algorithm then we can expect that we are going to be run in log N time with order N work.

If you do it incrementally like it I written here then it takes N step it is not then how implemented when you do broadcast we have to eventually think of cost also if it is shared memory model where you can write somewhere everybody can read then okay. But in other model it can be expensive but again complete your algorithm. So assuming broadcast what do you do but why did the parent get the sampler.

So parent is summing one element at a time should both of them will do X0 + X1 okay when what will they children do both have computed the same thing X0 + X1. So there are two nodes both have X0 + X1 and you are saying one of the node will ideal itself now and so now basically we have one node with X0 + X1 and it is two children will do what okay what will the other two children do or of the node of the X0 + X1 that is what I want to get to.

(Refer Slide Time: 18:52)



You have the right approach is not fully found so let us look at this algorithm for trial. It is recursive and I have drawn the leaf level here and do other things but I am going to physically enroll the recursion here at the leaf notes you take those two elements add them together okay I have got N by 2 sums each is a sum of their. So first is 0 + 1 second is 2 + 3 and so on. Now I recursively say compute prefix sum on this okay so I have got the prefix sum of that N by 2 set.

But I really needed the prefix sum of it and set can I get from that sum to these prefix sum so these prefix sums of the circles you are seeing here a leaf note are exactly the prefix sum of the even ones. So if that guy wants to know its prefix sum it is just what the parents says okay. So if you number from 0 to 0, 1, 2, 3 then this is odd the right child has the same prefix sum as the parent hat about the left child.

Parent has also added its right child so parent – right sibling or parent sibling plus itself okay so given that plus is the operation we have said a function that reduce operation so it although I am using plus as an example it can be any left associative operation okay which user can even to find. So it may not always be invertible but if it is veritable you could do both of them f it is not in veritable at least do one of them.

This is the final algorithm this is written in again PRAM style so everybody shares memory shares X and P. So for all the elements you take the pair wise here it is written as some operation right. We are talking about some in the previous you perform that operation on X twice I and twice I + 1. So 0 will do 0 and 1 to 1 will do 2 and 3 and so on so that gives you the first level and then you call recursive prefix some on new set Y and get Z in which the prefix summit stored.

And then reinitialize the output now we are generating the final output P 0 element is always X0 so initialize the left most value of the prefix some and then will do what we just talk about right. If I is odd then whatever you parents says whatever square leaf nodes parent says is your prefix sum. So P of I is Z which was the return from recursive all of your parent otherwise it is your parents left sibling operated with you added to you and if it is invertible then you can apply the inverse of the that operation with your right sibling also.

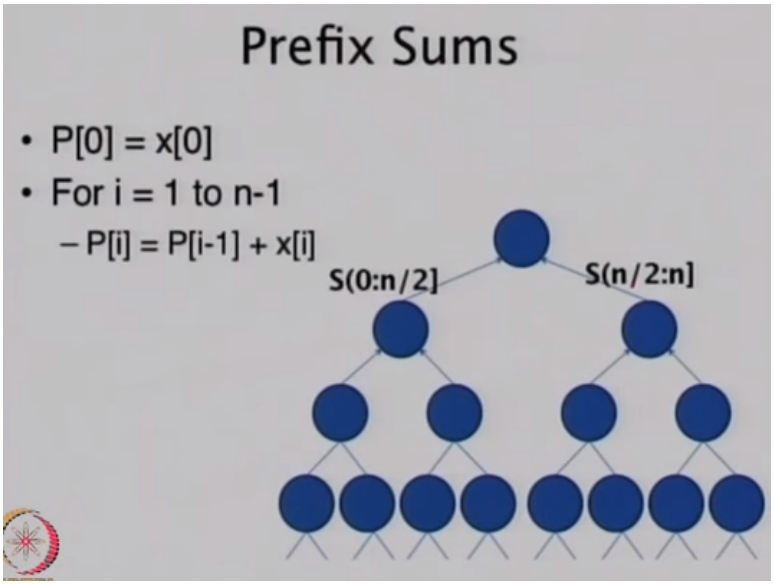
How long does this take? A log N time and what N log N problem is recursion is that it is hard to do the scheduling you cannot clearly see it but it can be done. And so unroll the recursion and then try to shut you let ourselves. Before we do that talking in PRAM terminology is it what ERAW, CRCW what kind of memory model are we using here are there any concurrent rights? Are there concurrent reads? Somebody is reading the parents data right how many two can you get rid of concurrent read?

Actually it already has the getting read of the concurrent if you are the left sibling then you will read your parent. So in that if left everybody is unique and then in else everybody is unique right actually it is exclusive read and exclusive write in the PRAM model. And then you can very

easily extend to MPI model and whatever because there is no reason for two people to read the same thing okay.

So now let us unroll it we have to just find the address somehow we have done the mapping you have to know where the parents sibling is in fact it would not necessary be the node number because you will have many fewer processors is but so you have done the mapping so you should know where it is okay.

(Refer Slide Time: 25:48)



Let us look at the tree version right unroll the recursion and so in the first pass of the tree going from bottom to up suppose we create all the sums okay. We are storing the sum in each of the nodes and now what I would like to do and by the way there will be that the terminology I am using that is S bracket close bracket 0 to N over 2 means that it is storing from sum = 0 all the way up to N by 2 whatever is just before N by 2.

N by 2 is open and 0 is closed and similarly on the right child of the root giving it S N by 2 to N and similarly of the left child of the right child of the root is giving it S from N by 2 to 3 and X4 and so on okay. So now everybody as these sums so I have the sum of everything and the sum of the left half sum of the right half sum of the left right and so on. How does this helps us to get the prefix suppose now you start to go from top to bottom we have stored typically in your regular someone what would you do?

You will say S = S of some left child S of right child and one of you are holding one of the left child one of the children right if you remember the mapping every level is mapped independently with each other and so say the level below you is 0, 1, 2, 3 then 0 will get 0 and 1, 2 will get 2 and 3. So 2 is holding 2 when I said 2 will get 2 and 3 it is going to add to its S the result of freezes right.

So it is previous S is gone reusing the same variable but we need to store it you need to remember so for example route was also involved in the leaf level computation the roof lead need to store the full some it also needs to store what happen when you added 0 and 1 and it also needs to store what happened when you added 0, 1, 2 and 3 right at the next level and so on. So now what do we at the route it is prefix sum of everybody however the roots to children compute the prefix sum of the left half and right half.

What is the prefix sum of the right half whatever that fellow has computed earlier right which is still at same route same mapping that we have used earlier so it is there. What about the right child? Root – left sibling would you want to do that I do not even know whether minus exists I do not want to go in the direction of minus can you do with just plus in fact that what is coming down the arrow is being shown.

You want prefix of left half and right half so from there you generate if your parents had the prefix sums can you generate the prefix ups will start at the same slide we were trying to work out the prefix sums O to N. So the previous picture you had seen which is now beyond this is what being sent on the two paths for the root to compute the sum of and so he route has the sum from 0 to N and it also has the sum from 0 to N by 2 because of the previous case.

So it can send the data that it need to send so you can think of it as the following to your left child you send your sum the sum that the parent is given you right whatever the sum that the parent computed for you but to the right child you are going to send the left child data. So what is that doing so if you are basically doing the same thing that the previous in the previous case

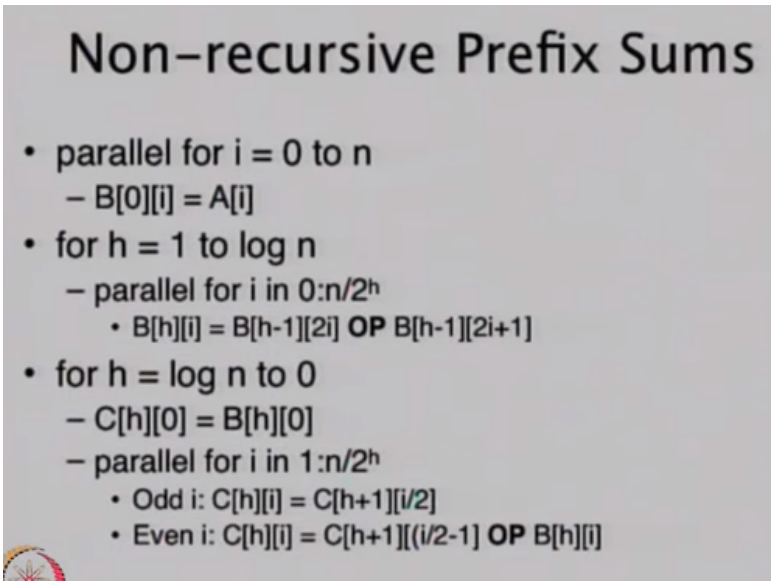
right if you are left child what you have to do to take the parents left sibling right and get that added to you.

So you know your value you can add that later but you have to get the parents left how do you get that? First level meaning from the top yeah for the time being we are not ahh the input problem are intermediate problem we can figure out but we will figure out rom this that at the leaves we want some prefix sum at the end. So because I want on the prefix sum of each level so for the recursive operation what have we done it said if you have to compute prefix some of this level you reduce the set compute the prefix amount of that level and then either take your parent or take this sibling.

Left side does not need the right value only the right side needs the left value right so now with that what do you expect? What data should you expect at any level at first level? For example should at the end of the it is business have the prefix sum of the 2 values that it is going to that came to that at the first pass meaning the left child needs to have 0 to N by 2 and the right child has N by 2 to N and left gets the prefix amount that meaning only 0 to N by 2 right child gets the prefix sum of these two things means the sum of everything 0 to N right.

So right child gets what the parent gives it because parent had 0 to N right and left child already had it but just to keep it separate you are going to get it. So for example suppose we are generating in an array then we copy it from somewhere else in that array okay.

(Refer Slide Time: 34:16)



So just to write that out (refer time: 34:16) so now that we are going back to the shared memory PRAM model we are not going say that this process is passing somebody to that process right whoever needs it will figure out where to read it from right so if you either read it from your parent or the sibling. So that is what happening here so you begin and the data is in array called A.

So for some reason X has morphed into A because if said A, B and C A is the input and for all the input data given to put in B 0th level of B. So B is computation of the some in the upward direction C is in going to contain the sums partial sums C is going to contain the prefix herbs alright. So we begin by the 0^{th} level B is a 2D array why is it 2D array we need to store you cannot say S = S + the other child because you need to store all the partial sums that everybody has seen.

And so B will store the partial sums of different level in B0, B1, B2, B3 and the second dimension is how many elements there are on that level okay. So B0 will have N data N elements which will come from the input set A B1 will have half of them which will come from pair Y sums of B0, B2 will have one fourth of the input some which will come pair wise sums of B1 and so on right.

So after we have initialized copying everything to be which so after the first loop which is parallel for the setting of a values into be D0 are then you go for log N levels right from the bottom of level to the topmost level and so for H which is the height you starting from 1 to log N you do in stuff for each level that is parallel for every H value and I goes from 0 to N by 2 to the BH because it is keep reducing I keeps reducing.

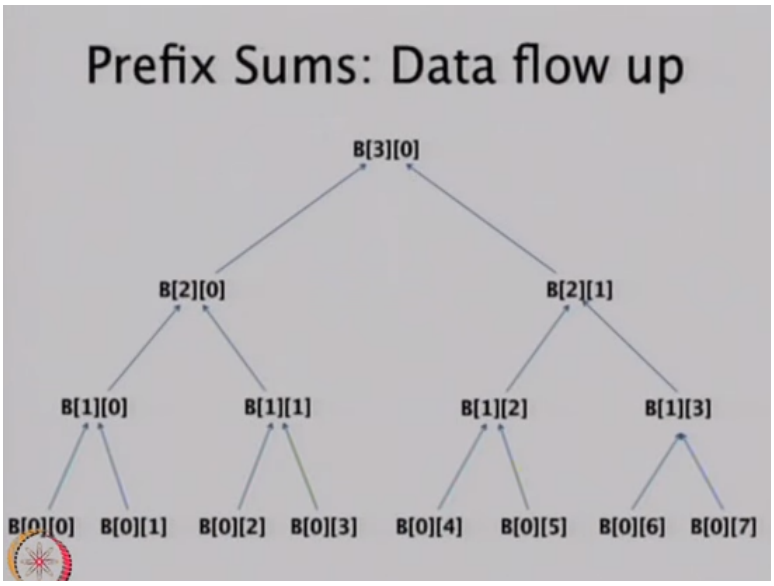
And as I said compute BHI + BH – 1 okay just take pairwise sums and at the end of the second bullet item you have got the sums after you got the all log in levels and now down from log in height all the way top to 0. You initialize the prefix some is now C and so you copy the prefix sum from B to C for the top element right which was at top is only 0 so you Just copy into C.

And if as you go down level for every level what you do if you are the left sibling or right sibling you do different things. If you are the left siblings which means I is even then you are going to take the siblings your parent siblings prefix and computed so far C add to it your value you have partial sum which is B and if you are the right child then you simply take your parents C where you prefix sum what your parent computer okay.

So you basically doing the same thing that was earlier shown for the recursive procedure for just the bottom level now you are doing this at every level you are basically unrolled the recursion and it is very clear what the time and work complexities how much time? log N you go in parallel log N steps and then log N step coming down how much time for each one of this steps everything is being done in parallel okay.

Similarly warp will be N because you will be doing order N at each node of you go one's up one's down okay.

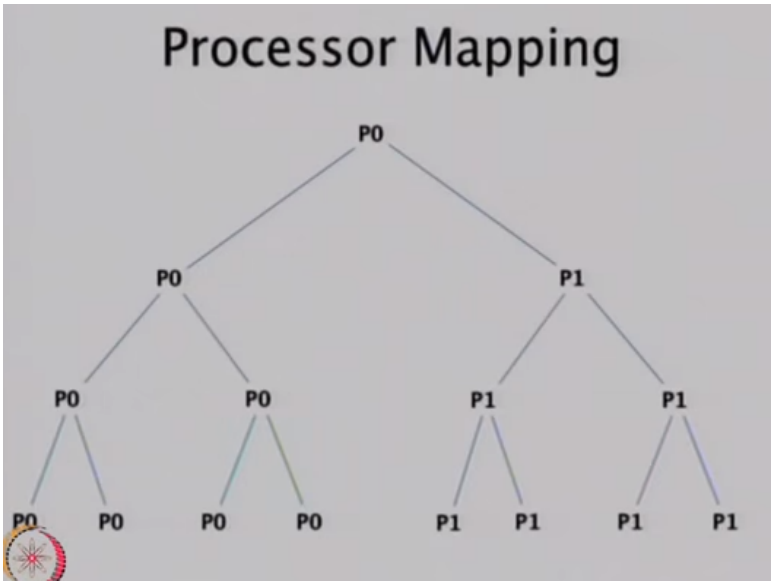
(Refer Slide Time: 39:21)



And so this diagram shows the data flow of how B values B got initialize for 0 from A and then you got the partial sums as you go up to B1, B2 and B3 and then C at the root got the B value initialize and then you keep coming and the way you keep coming down is a being shown by arrows in this diagram okay. So if you are confused then you can go back and look at this diagram and get yourself re-oriented.

Order and work order log N time processor make does it make any difference how we map each level suppose I say 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5 well let us take an example.

(Refer Slide Time: 40:32)

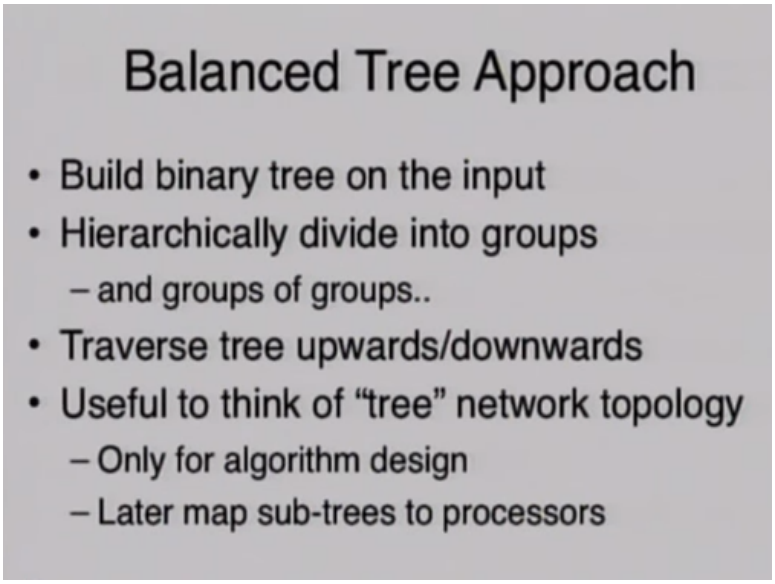


Let us say we have two processors and instead of this mapping P0 is taking the left half and P1 is taking the right hand suppose I said P0 P1 if that were the man would it be any different in certain situation from this mapping yes if we said P0 P1 then very left child would be P0 and every right child will be P1. You read exactly the same amount right from the memory assume shared memory.

You are talking about if it is right each level you will have 2 pieces of data from the two different processors coming and joining up to certain level also if you have got a SIMD like environment right this is everybody is even going to one processors and everybody is odd going to the other processor and both the processor must do the same thing in a given clock cycle then you cannot because they want to do different things at every instructions this guy wants to say do X and this guy wants to say do Y.

Which means every instructions going to get broken into two instruction once when this guy gets to do its X and then the next clock when this guy wants to do it Y whereas if we do ahh lock synchronize as is done in this case then each one is going to enough left enough right to keep them busy. So you will say now I am going to do N number of left that guy also has approximately N number of left and I am going to do N number of rights and that guy also probably a number of right.

(Refer Slide Time: 43:00)



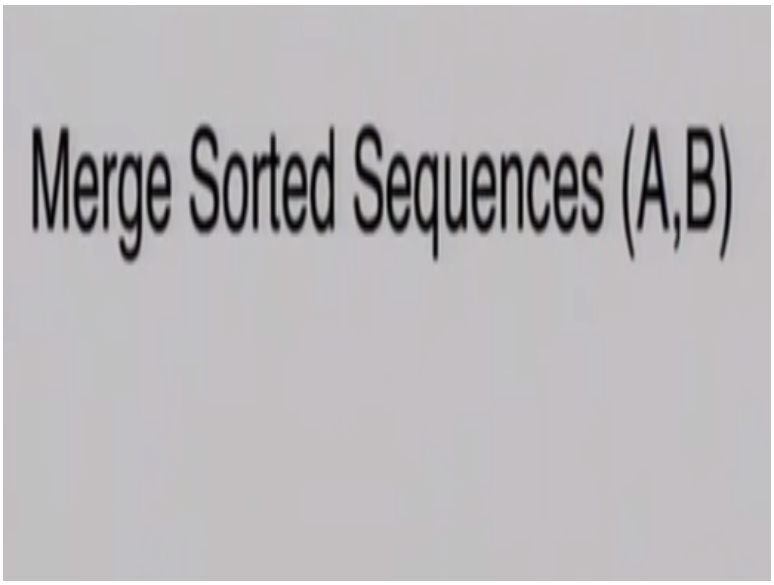
And so every clock cycle we are going to be useful okay alright so in short balanced tree approach is going to restructure or structure rather the computation into tree like fashion and then you are going to traverse the tree upper down it make sense to think of the communication as the network topology as all those diagrams where showing but ultimately it can map to whatever program paradigm you may be interested in and whatever the number of processors may be interesting okay.

You probably should not start on a new topic but I will anyway just to get your thinking about this problem and then we will come back and talk about it on Monday. And this technic is called portioning you will see that again going to use some tree like structure the idea is that you take a problem a sub divided into two problem and solve them okay kind of similar to divide and come.

Sometimes people differentiate between in two saying that divide and conquer you focus on the conquer part more and in portioning you focus on divide part more right you divide it in the way that then independently the problem will be solved and there would be not much combination would do after the individual sub problems are done okay is that you remind it something quick sort right alright.

We are going to do merge soft with partitioning where we are not going to work let me rephrase we are going to do merging of two sorting sequences using partitioning where we partition do the work in the partition data and we have done so that is out next algorithm we will be taking about.

(Refer Slide Time: 45:00)



We have got two sorted sequences A and B and with some P number of processor we want to merge them and remember order N is the work complexities will be interested in and what do you guess time complexity we should be interested Log N is a good guess actually you can do much better think about how you do it log N and then think about how you do better than and we will talk about that on Monday okay any questions okay.