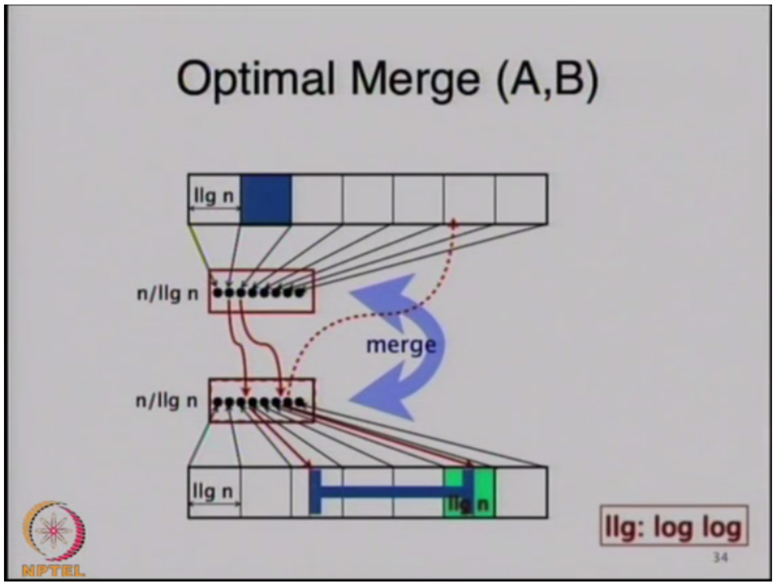
Parallel Computing
Prof. Subodh Kumar
Department of computer science & Engineering
Indian Institute Technnology – Delhi

Module No #04
Lecture No # 20
Algorithmic Techniques (Contd)

Alright so the idea was that we have two sorted lists given to us and you are going to merge that like we do in merge step. For example and we took each list and broke it into log and size blocks log and log and is log log N log log N is size block and end over log log N was such blocks alright. And we take one element from each block ok.

(Refer Slide Time: 01:01)


Optimal Merge (A,B)

From one list and similarly from one element each block from the other list an now you got a smaller set and we are going to merge these two merging these two using the previous algorithm ok. Whatever it was the previous algorithm was fast but did not give you the best work which is linear ok. Now the work for this will be linear because we are only what the work was and log log N right.

Now the work is the size of the set has become n over log log n so it is that log of that so its N log log N times log of N over log log N which is the same thing as that so far just to merge this using the previous algorithm. You can do this is in time log log N you can do it in order n work ok and that is our goal to keep it fast and remain within the budget of order n for work ok. But we only merged these two lists right I found the rank of every dot on the top list in the second dot list and the rank of every dot in the bottom list in the top dot list ok.

As if the real two arrays do not even exist for this purpose so far ok now I going to take the next step where I going to find the rank of every dot in the full list ok. Hardware find the rank of that I know the rank of that dot the one shown in radar in the other dot list ok which means it lies between those two dots in the dot list. And those two dots tell me where it might lie in the main list right which is our size log log N so if even I spend a sequential amount of time one process are doing all the work.

I can log log N extra time l the position of that dot from this arrow begun in the full array ok. How many such dots are there n over log log N right if everybody takes log log N time? Then work is order n and time is log log N right similarly for the other side ok. So now we found its position in the full array somewhere we can find the position of every element in the dot list top dot list using so far only n order work and log log N time. And again the other side every dot in the second list we can find its position in the full first list in log log N time order N work.

Similarly right now what remains to be done I only know the ranks ultimate ranks final ranks of all the dots I only know the dots ranks ok. Now what about all of these elements that are between those two consecutive dots I do not know their ranks. But I know the ranks of the two end points so we know those two positions in the full list full second list. So you know all of these the blue elements are going to be finding the ranks somewhere in that range of the what color is it contain blue ok.

If I can merge these two all done and so every block of blue on top list I am going to find the corresponding block of blue in the bottom list and merge it and this bottom list can be too big.
If it is too big I want to then I have to do something smart to merge this to other way if it is too small or if it is very small nothing is too small or it is very small say log N size. Then I could merge that list with list sequentially in log log n time right unfortunately this is not small enough for me.

But if I look at one of the blocks within it which is log log N it is definitely going to land somewhere between in that blue block. And we exactly there right there is no new block same blocks yeah in log log N size no point what we want to know is purple is it purple block is going to fall there in the other list. If we know where the end points of the purple blocks are then we done right and the end points were selected in this dot list we know their ranks right and definitely their ranks are going to come in that sign space.

Because that sign and this purple the bigger blue it signed here bigger blue and the smaller blue at the top are going to merge with each other. We just figure out which parts of this bigger blue merge with which parts of smaller blue which can be easily figured out by going in the backward direction because we know where the purple will going to go right. It's going to be one part of that bigger blue now we are guaranteed that every pair of merges had at most log log N size right.

If it turns out we perfectly balance this log log N on both sides otherwise this log log N on one side is smaller than log log N on the other side right so each merge can be done in log log N time. How many such merges are there N by log log N right in the worst case (()) (08:52) log log N that many there that many there and work done.

And the log log N independent units of work had been done each being done in log N time so log log N times N by log log N is order n because whenever we are merging either one of the blocks from this array is being used or one of the block from that array is being used. Yes original blocks right so every merge has one of these some merges may have two of these. But atleast every merge has one of these if they are two it is going to be slightly less ok

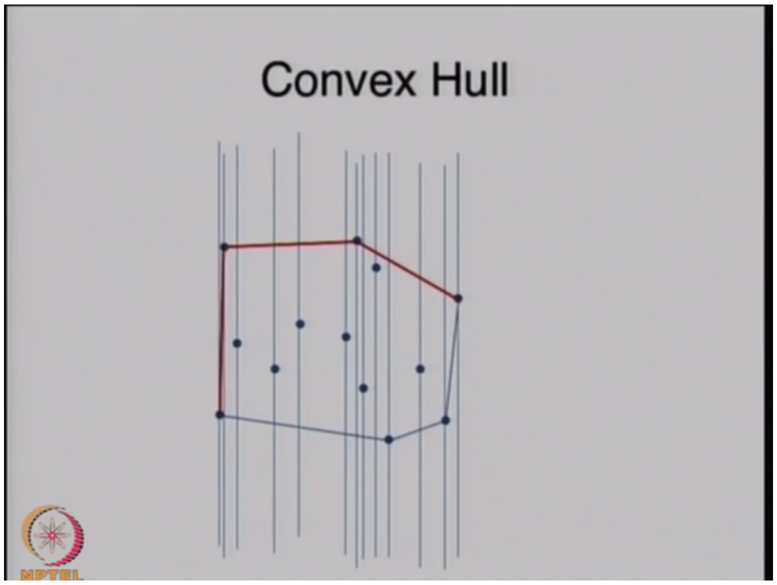
Why so we can figure out where the purpose should go we can take the two end points of the purple to where they land in the blue block at the top. Yes in the beginning you can do both you do not know what are going to need you postponed. It if you want to ok you findout for all yes any questions first we found the rank of the element in the dot list in the other dot list and then because we knew where the dot list came from we can then find the rank with little bit extra work of all the dot list in the full list both ok.

We can find the rank of all the upper dots in the lower full list and all the lower dots in the upper full list final rank is what we are interested in is the rank of every element in the top full list in the top bottom list at eventually we are going to add it to its rank in its list which is triggerly which is you know right. Its position in its list and similarly for all the elements in the bottom list you are going to find their respective ranks in the top list.

So you broken into three steps actually so every of these blocks is bounded by two elements in the dot list right so if you want to know where the block should what the block should merge with you. Figure out where the line in the other big list big list is what is listed in the top and the bottom if you do not think big list is drawn then you have completely not understood. What I was just said those we started with A and B does not matter in the previous example we had B at the top so you can call B at the top here symmetric.

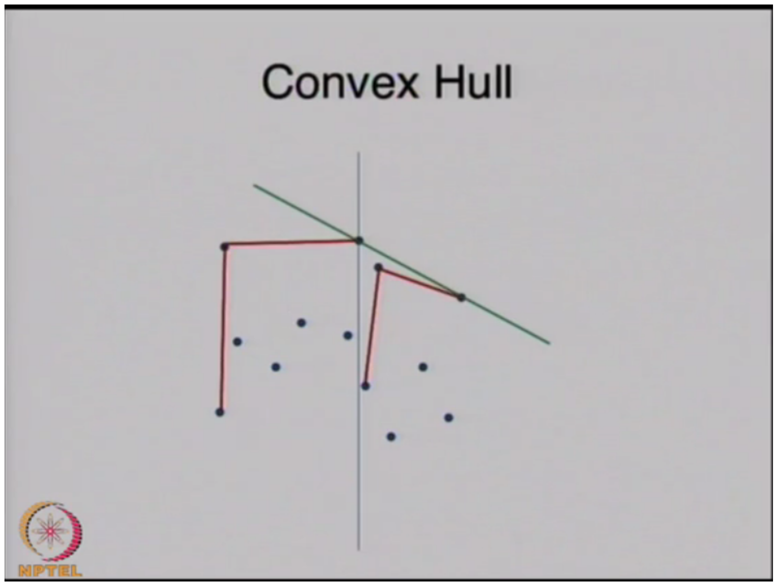
So it does not matter right after you merge now that that step I am not talking about which is once you know your rank in the other list you know you are in your own list. You add those too and you put yourself add that position in the in this c array ok it become c yes that is the bigger size ok.

(Refer Slide Time: 13:23)



Convex hull this is another example of divide and concur and I am going to go very quickly through the trigger parts. Essentially we are computing the upper hull only lower hull will be computed similarly. And just connected to the other one so focusing on the upper hull and that is what is being done by dive and concur. So we will divide it into two halves based on the erect spotters and then compute the upper hull of left my left does not matter what my you think upper hull of the other side and then find the common tangent right.

(Refer Slide Time: 14:08)



Convex Hull

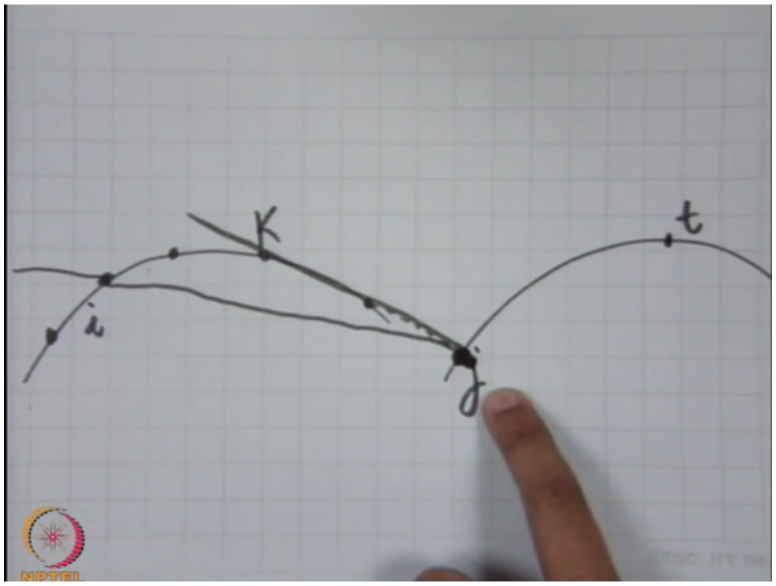
So something like that the red is upper hull of what one half and other red is the upper hull of the other half and then the green is the upper tangent that that's the mean part of this algorithm right. Which is what I am going to talk about but once you have found the green what is the combined upper hull everything on the other side other sides from the tangent side right? Meaning the divider line side of both the halves if you did not understand what I said you did not understand what I said earlier also write into that those question then is not it obvious.

What you do with that upper tangent to find the two hulls if you understand what the hulls are do you understand what the hulls are yes. The upper hull of everything so once I have drawn the tangent what is the upper hull of everything suppose we say both the hulls are always computed in clockwise direction ok. So we have given the clockwise hull of left side clockwise hull of right side and I want to generate the clockwise hull of the result what needs to be done.

If I tell you the two points on the two hulls where the tangents meet so let me be move to size we got a 0 A1 A2 A3 all the way upto A end list and B0 B12 B2 B3 and all the way upto end ok. And

I say I is the tangent point on one side on the left side. J is the tangent point on the right side what is the resulting hull exactly a0 upto AI. Then the reverse of BJ upto bI ok because right not reverse that's also in the same direction just BJ to be end ok. Now only thing we going to do is where do I got this I and j from and can somebody remind their sms right.

(Refer Slide Time: 17:41)



So I have got two hulls and I do not know the real IJ I am taking two kinetics any candidate on the left side. And any candidate on the right side I would typically take the midpoint right because I want to do something for bind the research suppose I fix J ok when can I tell if I is the tangent of the other hull from J of this hull really you going to do cross product. This is two d so if you if you look at the two points on the two sides if they both lies below the line.

Then you got the tangent below the line right ok now suppose they are not the tangent as in this case no suppose I that I chose was not the tangent because two points were on the opposite sides which where should I move is it a tangent on that side or on this side. Why what is the condition that you say this on the right side what if I were drawing a line into this other line at the point sorry one is left one is right.

We do not need the angle we just need to know which is below and which is above and we know that its going from left to right clockwise we just need to see so one is upward one is downright. As we go from left to right I will be going from down to up or up to down if you are going from down to up then we need to move further up right. So the left all the left points from I can be discarded tangent cannot bigger similarly if I am on the other side can all the right points can be discarded and you only concentrate on the left side.

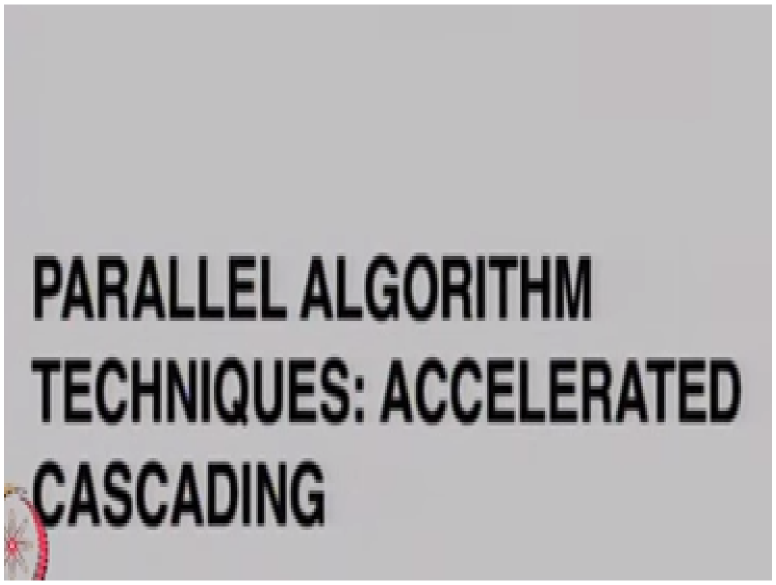
So each step you can discard half the points and keep narrowing it down until you found the tangent ok so I found the tangent from J this is not be actual tangent the actual then the midpoint of the other side we started so in the beginning we chose the midpoint of both sides we connected them we put one as an anchor arbitrarily right.

And we said let us find the tangent on the other side and we found it after long steps ok now we are interested in finding the full tangent do you think n the point that we found on the other side is a point on the final tangent it may be or may not be right. So we could find the tangent from here which is what ultimately you will end up doing but that need not be the tangent. However from this other point we can find the tangent to the other side and keep doing this and the analysis is get little complicated.

So I skip to the analysis part it can still become log n time so each time you are reducing your set to enough but you can see this imputably in the following way let us give this a name the point is K the real tangent of the other hull is T. If I draw a line from the found K to the chosen point J can we tell which say the real tangent is exactly were the same route ok and hence not tangent from only that point K but the real tangent also has the same principle right at any stage.

That is for analysis part the intrusion for the analysis at any stage you can tell that the real tangent is not in this half yes right ok. So without going onto the intricacy of the analysis the algorithm is only this you choose the point on one side search for the tangent from that point. Keep that point and search for the tangent from this point over here keep this point search for the tangent over here at some point you will converge to the real tangent ok after small enough number of steps no yeah tell me and both they are neighbors on both side are below it.

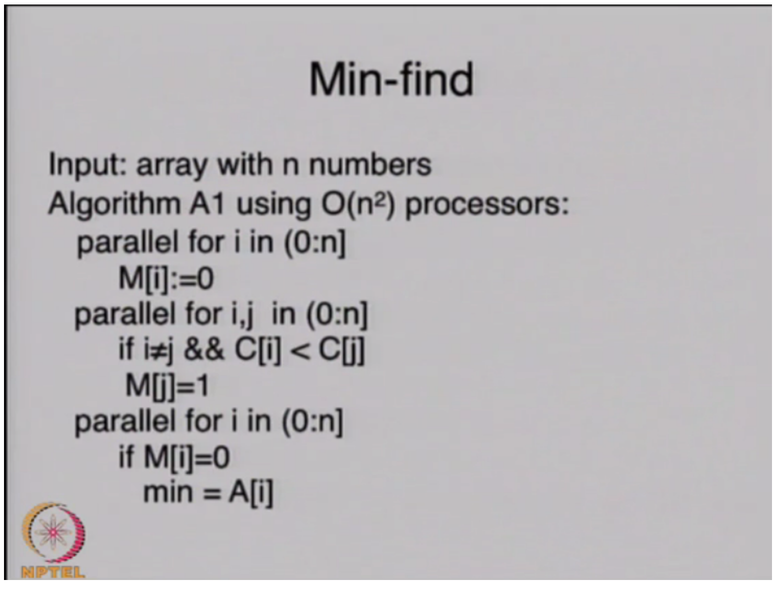
(Refer Slide Time: 24:02)



PARALLEL ALGORITHM TECHNIQUES: ACCELERATED CASCADING

Now let us move on to this to the other algorithm that we were discussing the last time which was to find the minimum of N numbers ok and that belongs to the class of accelerated cascading and I have told you that we have seen one example of it. J just now where we took one fast algorithm and not so optimum and another optimal algorithm not so fast and combined by setting the right size of the input right sampling the input in somewhere ok or turning into blocks we are going to do something similar.

(Refer Slide Time: 24:44)



Min-find

Input: array with n numbers

Algorithm A1 using $O(n^2)$ processors:

```
parallel for i in (0:n]
    M[i]:=0
parallel for i,j in (0:n]
    if i≠j && C[i] < C[j]
     M[j]=1
parallel for i in (0:n]
    if M[i]=0
      min = A[i]
```

But we begin with we are going to find it is almost we have to find an order one in finding algorithm we can find an order log N when finding algorithm using the balance (()) (25:00) technique which will take optimal time optimal work but not a satisfactory time ok just for reminding how does the order one algorithm work. If I had N squared processors so n processors are comparing the element I to on the element another elements right and - 1 processor not going to worry about the boundary condition and what do they do if any of the element is bigger than this element then setup lies.

For what so tools lies this is the same thing whoever is the bigger whose that guys slag you said if I is bigger then set a I is slag which is the same thing assign. If j is bigger set J as slag ok so for every win somebody is setting m 3s slag where three is being compared with all the other things right if three did not win any of the battles. Then nobody is setting 3 slag to one but if 3 is winning in many battles then multiple people are setting 3 slag to one it requires concurrent right ok.

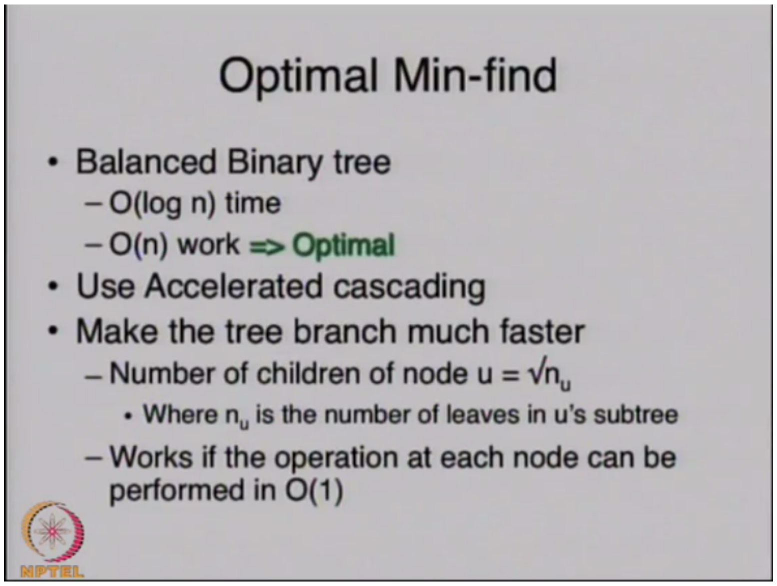
But what kind of concurrent right anything I just need one of those wants to succeed if its common they are always writing the same thing if it is random anybody is one will get written.
If it is priority one of the one will get written with based on the priority right so it does not matter which prior suitably. We have and then who is the winner who is the main could you name it is there is only exactly one which did not mean any of the battles which is the main right so everybody checks the end slag that you have generated.

And one of them will say it is still zero nobody write wrote on there and that is the meaning ok and there is no concurrent thing necessary there if they are not unique. Then you have to do some further if you do not want concurrent over here then you have to do some further breaking tie over there but right now it does not matter it is still be a common right ok.

So we have got a fast but less than optimum right order one algorithm be the bad work order end with square block. If you are trying to implemented with p processors we do not expect to spend N square over P time right versus the earlier the simpler algorithm simpler or not the default algorithm. Which be state about earlier where it was order end work? And over time is all that would be needed right if I had P processors.

So can we now make the domain of a certain size such that we can keep the cost the time may be as good as constant may be its increases slightly. But reduce the work here you will see that this thing kind of technique we applied to merge such two algorithms in the case of merging we will need to be applied again and again. Because we are really far from the optimum cost and square versus N there we were N log log N versus n right so we will have to repeat it a few times ok so let me stop.

(Refer Slide Time: 29:54)



Optimal Min-find

- Balanced Binary tree
  - O(log n) time
  - O(n) work => Optimal
- Use Accelerated cascading
- Make the tree branch much faster
  - Number of children of node u = $\sqrt{n_u}$
    - Where n_u is the number of leaves in u's subtree
  - Works if the operation at each node can be performed in O(1)

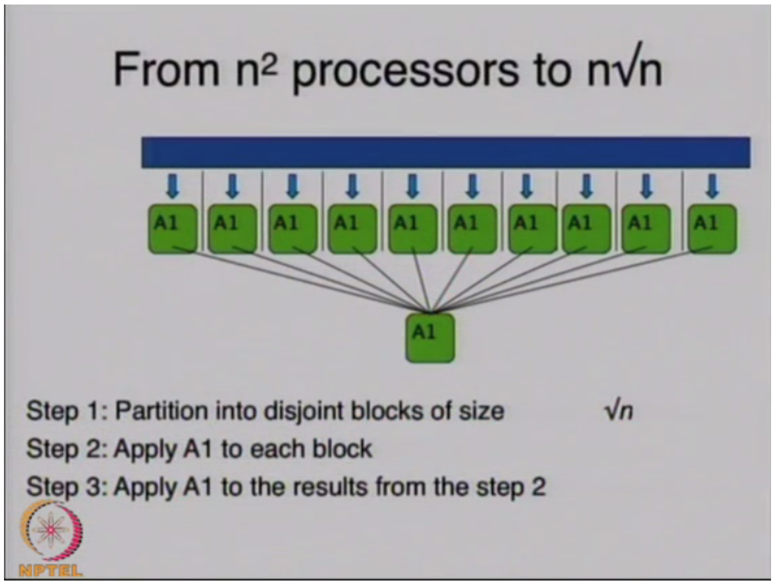
Basically at high level this SI what accelerated cascading is about you? Try to instead of in the basic binary search binary tree type technique you break it into two elements at a time right. It is a take pierce then I will take pierce of pierce meaning that is of all the pierce and so each time I am going to reduce the number of elements to meaning by two and the m winner will winner which is basically the same for all balance point equity.

If you want to make it faster you would essentially suitably to what you increase the branching factor and instead of somehow only saying find the mean of two things you want to say. Find the mean of many things and so the tree will get stunted that would be the idea right and that’s the idea of the accelerated cascading and it will decrease time. And hopefully not increase work too much it might increase work right because it into your you are using one processor if you are making blocks of bigger size you probably are using more processors ok.

And the general idea this is not always square root but in order to branch out faster either you will have instead of two children at the node. At the root log N children at the root or root N children at the root things are that is all ok and then each one of them may have root of its size children. For example I will saw See how that works if I break the list into route N size blocks route n of n what can we do with so if we apply the previous algorithm on each of these blocks how long it would take on each of these blocks order.

One time order N work right because route N is the size square of route N is the work and which is N what is the total amount of work N route N right N each of the blocks route N such blocks total amount of work will be N route N right and then we still need we had route N we had one more step to do still route N size.

(Refer Slide Time: 32:53)

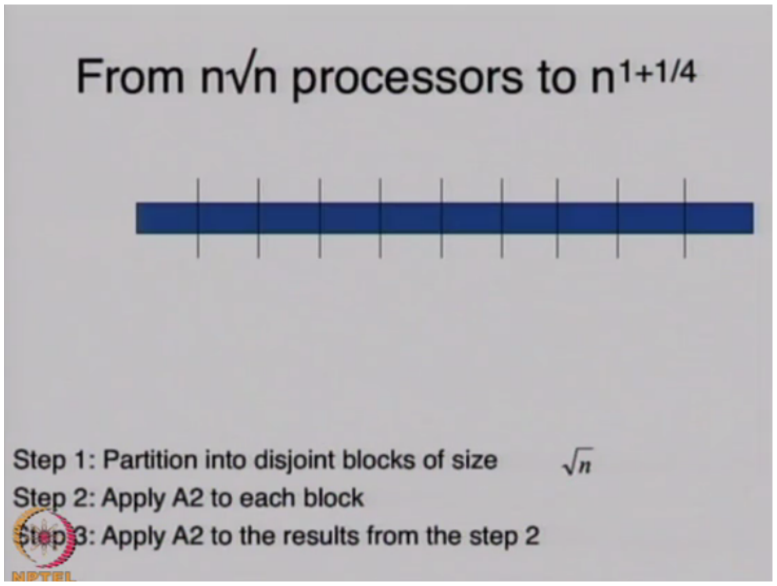


From n^2 processors to $n\sqrt{n}$

Step 1: Partition into disjoint blocks of size $\sqrt{n}$
Step 2: Apply A1 to each block
Step 3: Apply A1 to the results from the step 2

One more step order N more work so total order N work and route N work we had two steps we had figured out before we extend it why route N could we have tried log N or thing that so it would not decrease or too many of them it we make log n then we still have too many blocks left ok so route N is balancing on both sides here because once you find the minima there is small number of N and to find the minima you had small N of number to find the minima of ok.

(Refer Slide Time: 33:54)



From $n\sqrt{n}$ processors to $n^{1+1/4}$

Step 1: Partition into disjoint blocks of size $\sqrt{n}$

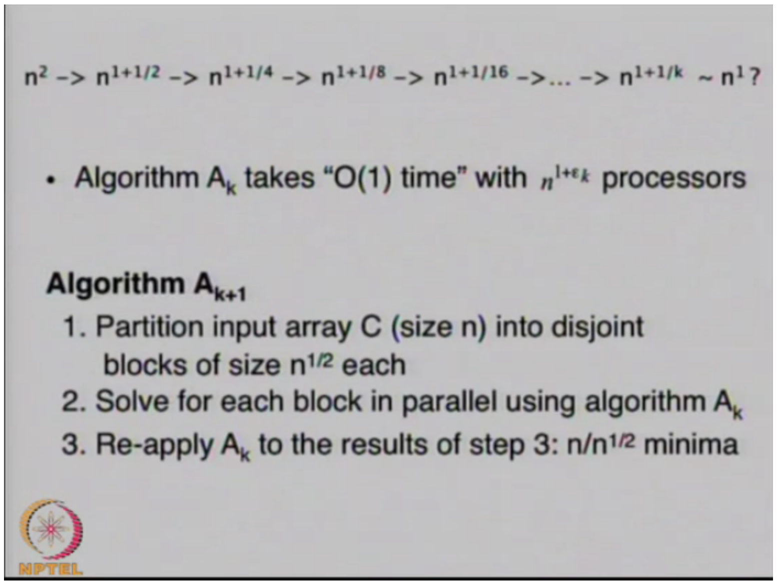
Step 2: Apply A2 to each block

Step 3: Apply A2 to the results from the step 2

Suppose we still break the list into route N things route n sized each we instead of applying the original algorithm you applied the N route N algorithm on it so instead of taking the square of the number of processors it takes number of processors times the route of number of processors. So what is and all happen. We have root N times what is N root N of root N and this leads to what I number 4, 3 the 4 so the entire thing whether root N thing each of size root N and on that root N size thing we are applying the previous algorithm.

So this SI root N root N times root N right the root N or them yes I do want to hold so it is what root N root N root N right and the time is what order one is the wrong thing to say it has one of slightly still order one right still order one.

(Refer Slide Time: 35:49)



$n^2 \rightarrow n^{1+1/2} \rightarrow n^{1+1/4} \rightarrow n^{1+1/8} \rightarrow n^{1+1/16} \rightarrow \ldots \rightarrow n^{1+1/k} \sim n^1$?

- Algorithm A_k takes "O(1) time" with $n^{1+\epsilon_k}$ processors

Algorithm A_{k+1}

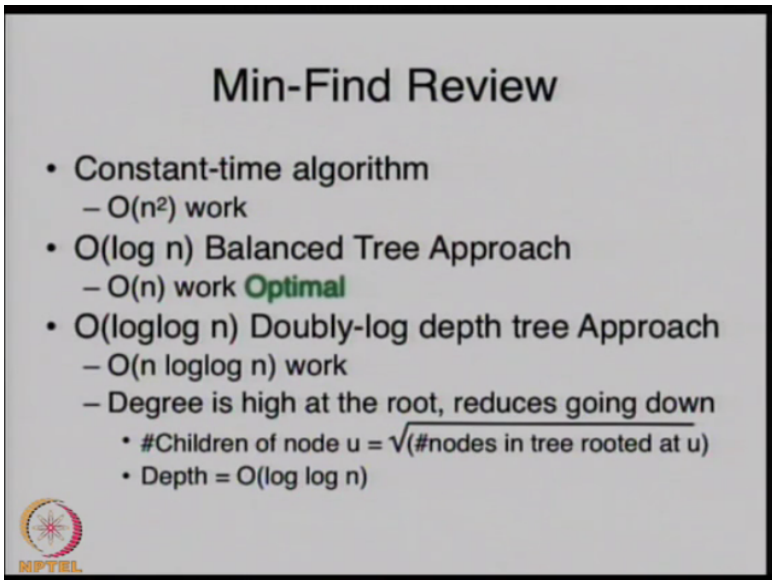
1. Partition input array C (size n) into disjoint blocks of size $n^{1/2}$ each
2. Solve for each block in parallel using algorithm A_k
3. Re-apply A_k to the results of step 3: $n/n^{1/2}$ minima

We can keep going then it would not be order one at some point when you take root N its still is same number right take you get down to two so now divide it into route n 1.4 and 1.4 you can do that right so divided into 1 and 1 and say now take root n to write from the user right so there will be a leaf level some level. So if you just think order one time that would not work right because this is dependent on n and if you look at the series what is it N 1 +1 and raise to the 1 +1 and raise to the E + half.

And raise to the 1 +1 fourth 1 + 1 eighth 1 + 1 sixteenth so the series is N raise to 1+ 1 over total summary after I levels ok and I says how many steps so right if the work is 1 N raise to the power 1 + 1 over 2 to the I then you have taken I steps. And we would like n rise to 1 + 1 over two to the I to be around one right at the leaf level.

So what is I do it on a piece of paper log of N K what is K I cannot reach to any end reach to K right take it to the logical control that leads to become side one also greater than one it is a log log N the total number of levels in the log N and N will be working every single times. So it N log log n work that does not should have come earlier so N raise to the power N over two to the I should reach point right or greater than one greater than one right yeah greater than one some constant number fixed number its N one over two to the I order one agree log log N ok.

(Refer Slide Time: 38:33)

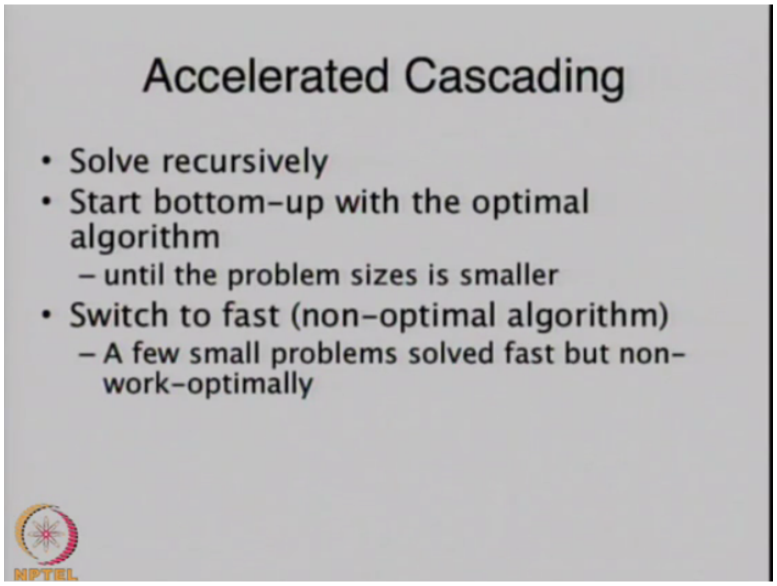

Min-Find Review

- Constant-time algorithm
  - $O(n^2)$ work
- O(log n) Balanced Tree Approach
  - O(n) work Optimal
- O(loglog n) Doubly-log depth tree Approach
  - O(n loglog n) work
  - Degree is high at the root, reduces going down
    - #Children of node u = $\sqrt{\text{(\#nodes in tree rooted at u)}}$
    - Depth = O(log log n)

So its quick review fast constant time algorithm but not nearly optimal combined with an optimal not a very fast algorithm Essentially by extending the branching factor of that the log N style algorithm ok. And it is important to note that the degree keeps going down right it is down on the levels of the tree because the number of things amount of work you are doing in any given node keeps going down.

As you go down the level of the tree so the branch is also dependent on the size of the work that you have going to be done so it now it changing from a constant branching level tree to a branching level that say something about what the size of the work to be done in under levels.

(Refer Slide Time: 39:38)


Accelerated Cascading

- Solve recursively
- Start bottom-up with the optimal algorithm
  - until the problem sizes is smaller
- Switch to fast (non-optimal algorithm)
  - A few small problems solved fast but non-work-optimally

And so at high level it solve recursively which is true for most interesting algorithms although you may not necessarily implement in that way and you do for a small enough size of the problem you do the none optimal version the basic version the compete treat type version and once it becomes big enough chunks then you start repeating the fast algorithm ok. Same so you think about it down not bottom now typically if u going to implement it known recursively you do time to start again it not necessarily ok.

I said that although you design and think recursively you do not necessarily implement recursively ok and typically you will get something like log timing ok. Let us go though one more technique and want to switch to because and lets to what our going to next let me they say is this technique is called recursive doubling sometimes also called pointer jumping why did we get that yes that not this way may be completely new way you do things but if you complain those you not going to be able to get the best of both.

Necessarily without losing anything ok so either you going to end up making the past one slightly slot or you are going to make end up making the work less than optimal sometimes you can get buy one of them another times suffer it both results ok. There is one problem in fact before we look at this problems suppose I get a tree in memory point is some n processors at disposer and I want to know the depth of each nod in the tree spark left to the root I want what no so everybody almost there points in fact got everybody.

I tell will you one nod ask you to find left in that which is more slightly nod general version of the problem you're going to solve here which is I got a list linked list your distance from the end linked list how do you do it as sequential case I want to everybody but then that needs to finish before I start right so that fine so give me the full algorithm you started the end.

Because I give you the list point out the list at behind you go to the end and he say if I just before the end add make me one ok so you figure out the distance of your next and you add one to it right and the your next will similarly do something add one to it and that cant processor as each nod so you give the processor each not a processor.

How do you begin what do what do they start doing broad cast ok similar to the free fixer its very similar to the prefixed but because of the pointers you can do slightly differently you do some prefix but you have to kind to have know may be citizen prefixed but that idea can it work that broadcast type idea suppose I just want to tell you what is my distance from you.

I tell my neighbor my distance is one from you in the mean time they having telling they neighbor they distance still from now I that I established that I am distance one how can the neighbors update who are a distance one became a distance two suppose I am in the going to use the next as a temporary variable imaging coping it. And everybody a is simply figuring out the distance lets two units away this figure out what is your distance something that is two unit away.

And then the next time right because the next time already done the same your doing its four units away now right have to log in time is you directly connected to the root that is so logarithm really simple. Initialized a copy of next I = null instead my rank is equal to otherwise my rank is one and then logging time you keep on ok so all you say is next I is equal to null and my rank is previous ran plus that rank that next logarithm got and then updated the pointed to whatever next year discovered ok after logging times you will have reached root.

And what your rank updated same thing in the case of tree if you want to reach the root everybody trying to reach the root essentially figure out the distance to the root you reached to the next your parent at the same time that guy reaching to our parents that parent that you initially have getting that was initial relation.

As you reach the parents parent that same time that parent has reached his grandparent so next time you reached grandparent login times and being your distance from the end your reached the end ok I am going to break this year. We have many more techniques as well as specific algorithms we are going to take about but I want to switch to CUDA because we are going to be doing next just a brief review of the architecture and then take about the programming model any questions ok let us stop.