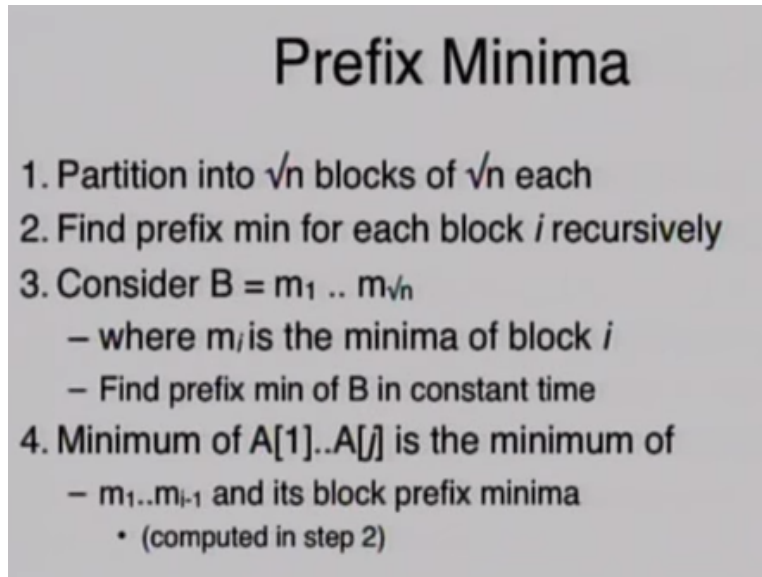


Parallel Computing
Prof. Subodh Kumar
Department of Computer Science & Engineering
Indian Institute of Technology – Delhi

Module No # 06
Lecture No # 28
Algorithms, Merging & Sorting

(Refer Slide Time: 00:27)



Prefix Minima

1. Partition into \sqrt{n} blocks of \sqrt{n} each
2. Find prefix min for each block i recursively
3. Consider $B = m_1 .. m_{\sqrt{n}}$
 - where m_i is the minima of block i
 - Find prefix min of B in constant time
4. Minimum of $A[1]..A[j]$ is the minimum of
 - $m_1..m_{i-1}$ and its block prefix minima
 - (computed in step 2)

We have already done something similar once before with recursively partition into sizes of your size right. So in the beginning we will have N elements in the partitions into root N blocks of root N size each somehow we are going to find recursively the prefix minimum of each block I right and then do what we were just discussing we are going to take the minima of each block and find the prefix minima.

And then prefix minimum as just discussed for every elements can be found right you have to fix minimum is your prefix minimum within the block you already know them the minimum to your left within your block and you know minimum among the blocks right. So the prefix minima of all the minima to the left of you and your minimum your prefix minimum the minimum of those to is global prefix minima. So how long will this take the last argument so I break things into sizes of root M okay.

For each I have found each minima and so that means that is called as local prefix minima you know the smallest element to your left within your block in your root N size. Now because we have divided in root N blocks we can find the minima of each block in fact we do not want to find it if we recursively found the prefix minima then the last element of the each block is the minima of each block.

You take this minima out so the first blocks minima is M1 should started with 0 but it does not matter M1. Second block minima is M2 third block is M3 you take all of these minima and find prefix minima of these okay. Size root N we can recursively call this function again we find the minima of these so for me this is my block I know the minima everybody on my left within my block also need to find the minima of all those blocks because my real global prefix minima could be either in my block or one of these blocks to my left right.

So if I had the prefix minima of all the minima that are to my left then I would know so if I take that guy prefix minima so that guy meaning the second level prefix minima which found the M0, M1, M2 all the way up to MI - 1 right. So the minima of those things meaning prefix minima of MI - 1 in the second level is the minimum of across the blocks before or to my left and I know within my block so the minima these two values will be my global prefix right.

How long will it take to partition root N blocks what about finding the minima of or rather prefix minimum of the root N right. So the step member 3 here we can do order and work one time and similarly first one will also be done within the envelope of one time order N work and then what about the last one every prefix minimum I have to find is a minimum of two things right so everybody is figuring out the old values the minimum of their old values and the corresponding minimize the second level minima array.

So order 1 time but everybody is doing it so whatever involved now the recursive part this is exactly as same as what we have done for min finding right. How many levels of recursion shall we have? Log N levels of recursion until things become constant size or size 1 and these levels will be doing order N work order one time so it will be a total half log log N and log N time.

(Refer Slide Time: 05:38)

General Algorithmic Techniques

- Pipelining
- Balanced binary tree
- Divide and conquer
- Partitioning
- Accelerated Cascading
- Pointer doubling
- Symmetry breaking

And so let us back to the general algorithm technic we have discussed so far and the I want to there is a couple that I had not got around to finishing that I will do now before moving on to talking about specific generic algorithm right to sort and how do you balance previous things of that sort.

So we have talked about pipe lining balance, binary tree, divide and conquer, partitioning, accelerated cascading, pointed doubling is the last thing we were doing I am going to start there I am going to give you very quick review of pointed doubling and then the symmetry breaking is the thing we are going to talk about next.

(Refer Slide Time: 06:23)

Find Roots in a Forest



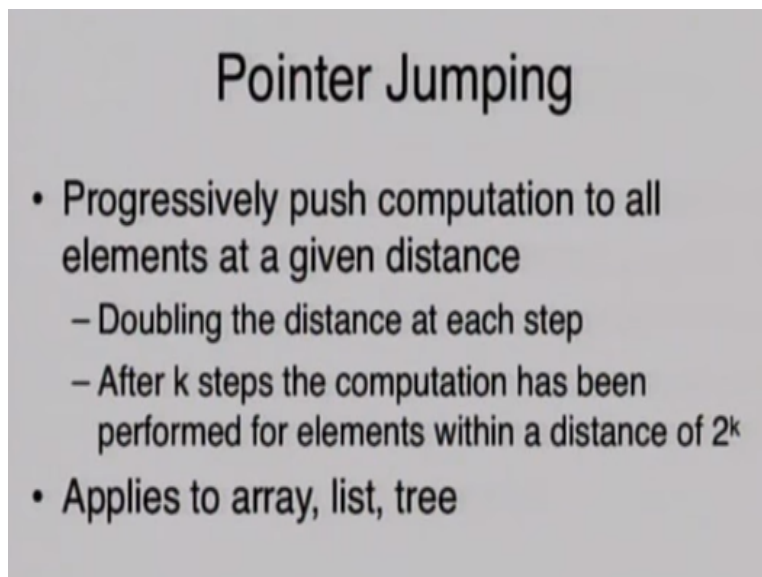
- $p(i)$: parent of node i
- Do in parallel
 - $p(i) = p(p(i))$
- Stop if $p(i) = i$
- Time: $\log(\text{height})$

So how did pointer doubling work? I am not going to go through the detail example here because we have already discussed the main idea in this case the example is you find the root of the forest it is tree but disconnected but forest and every node is need to find node. So average its parent at the same time so in $\log N$ step you will have the wherever your root was okay.

And so $\log N$ is yeah is should say that the height is the distance of the tree $\log N$ of that I did not mean the \log of the number of nodes. So everybody is doing parent of I is parent of parent of I and again just a reminder if you are doing this you will be doing this scratch space rather than your main pointer because you probably want to use it again sorry \log of height.

If you its height steps then you will take as you each steps will be going up one step here you are going \log of height you have to travel H by jumping over the pointer because everybody else is jumping along with you can reach their in \log of height time. The jump will become twice than four times and eight times and so on okay and the work will be N time \log of height because everybody is doing that.

(Refer Slide Time: 08:26)



Pointer Jumping

- Progressively push computation to all elements at a given distance
 - Doubling the distance at each step
 - After k steps the computation has been performed for elements within a distance of 2^k
- Applies to array, list, tree


And so the main idea is you push your computation to the certain distance and everybody is doing the same thing computation bringing your results and after \log to distance that you have sent this results and after that you have your going to be able to push it all the way through and so it is another did we do list ranking I think we did right.

(Refer Slide Time: 08:52)

List Ranking

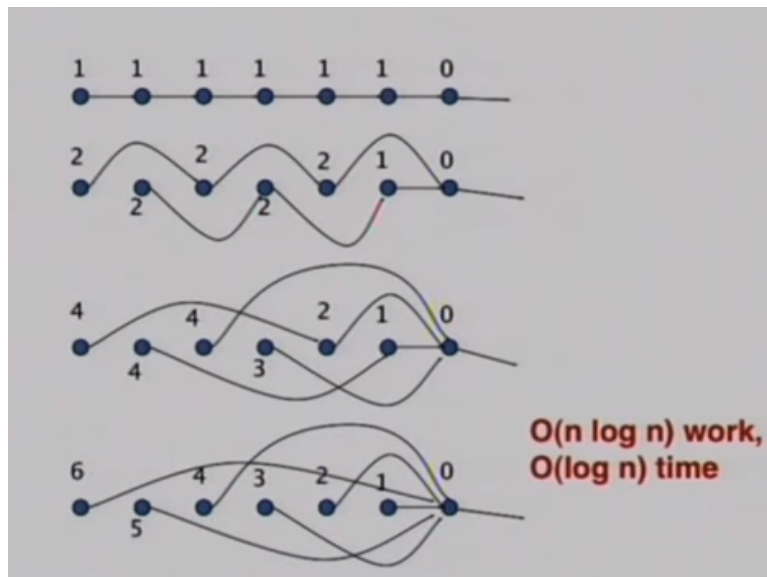
- Compute the number of nodes before node i in a list
Parallel for all i do:
 if $\text{next}[i] = \text{null}$ then $d[i] \leftarrow 0$
 else $d[i] \leftarrow 1$
Parallel for all i do:
 if $\text{next}[i] \neq \text{null}$
 $d[i] \leftarrow d[i] + d[\text{next}[i]]$
 $\text{next}[i] \leftarrow \text{next}[\text{next}[i]]$
until $\text{next}[i] == \text{null}$ for all i

$d(i)$ = distance of i from the end of the list



So another example is list ranking I think we will try so whenever example is list ranking where you define the distance from the last element and the same idea will apply as long as in yeah I know I remember as long as you so here is what you do as long as you are allowed to corrupt your next pointer everybody keeps doing next of next and you reach there in log of the distance time like that.

(Refer Slide Time: 09:23)



Now this takes $N \log N$ work $N \log N$ time can you do better how would you improve it specially the work is slightly maybe on the high side break it into chunks it is a list yes it is a linear list yes. They are already breaking it we are corrupting the next pointer in this process and

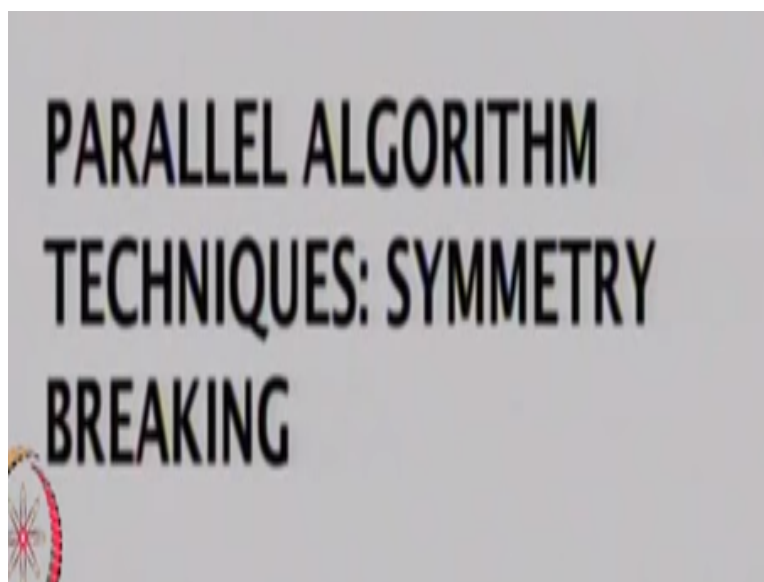
so we are basically saying we in a temporary next point that we are doing all this and so we can break it as long we have a vehicle restore it to the original dimension.

So suppose you are going to take off enough number of element so that you had left only an N by $\log N$ elements we will remove enough element so that we are left with only N by $\log N$ elements and then let us not think about implementation because it is complicated to keep track on which one you removed and bring it back but we will have to and suppose it is not even a list suppose it is a graph just like a tree it would not be order one it would not be order one.

As long as it is within then your $\log N$ envelope you should not theoretically you should not be there okay. So we will let you think about this especially in the context of the graph you are going to have to remove independent sets you would be shrinking the graph in some way you would but you would not want to remove chunk from the same location you want to remove every alternate ones like was mentioned so we will probably we will come back to it later by graph I actually meant a graph but you can think of a forest as well.

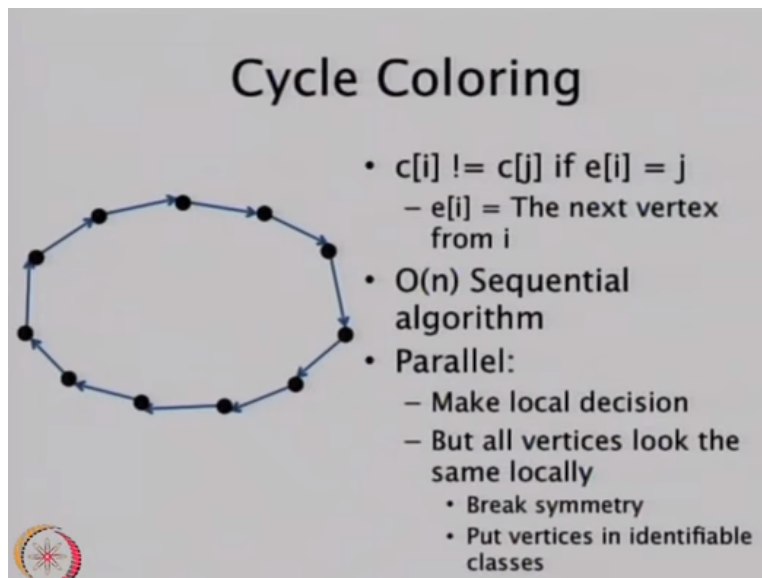
Basically we are not trying to find the shortest part we are just trying to reach the given destination but you can think to simplify or to be more concrete in this case you can think of it as forest in fact you can come back to the list in they do it for list also okay.

(Refer Slide Time: 13:08)



So let us move on to this other technique called symmetry breaking and it essentially is something that very peculiar to parallel algorithms in that.

(Refer Slide Time: 13:32)



Cycle Coloring

- $c[i] \neq c[j]$ if $e[i] = j$
 - $e[i]$ = The next vertex from i
- $O(n)$ Sequential algorithm
- Parallel:
 - Make local decision
 - But all vertices look the same locally
 - Break symmetry
 - Put vertices in identifiable classes

If I have let me give you an example we have a cycle and then I want to color this cycle meaning that alternate nodes should have different color and I would like in the what would sequential algorithm do I think we does not have it but what would it do ? Start some arbitrary point right and say you are 0 next is 1 next is 2 next is 1 and when you come back you cannot do anything at the end we have to you made it to say 2 okay.

It is not always possible to do it in two color so you will give the last one as a fair color if it is a hard number of vertices right. So but let us start at arbitrary point is something that is missing in the parallel world right because you want them to all of them to do start doing something you can randomize things right but so far we are going to the random direction yet. So how do you break symmetry? And in this case let us assume that there is an array which gives you the cycle and the every position of the array it says who the next element in the array.

You can start with TID but if you see that neighboring in the stable then so TID is not away to break symmetry right because the problem as the symmetry but I want to start everywhere I do not want to take order N steps to reach there I want to start in parallel everywhere and start

assigning colors and at the end they should match up that would be what ideal parallel algorithm would do.

It is not just about starting somewhere it is about every doing something but not knowing how they are different from us everybody locally sees it one is coming in one is going out it is the same as the other topologically. How does it help because it is not that TIP number 1 is getting node number 1 otherwise you will have to find where node number 1 is. There is an array right and then basically there is ID and it says your next is that fellow your next is that fellow.


So you cannot say the TID 1 gets node number 0 and the TID 2 gets what did I say? The TID 0 gets node number 0 TID gets node number 1 TID get node number 2 can do that . Random access is there if speed N okay. So let us say we do that where is how do you do the covering there but I am TID 5 okay so I got note number 5 but node number 5 is next pointer says 36. So whatever but it is not that node number point 5 is to 6 right so I cannot say I am odd my node number is odd so let me color myself by odd color.

And so we have to do something based on the index of the something similar there is flavor of point adjusting here also but let me jump ahead.

(Refer Slide Time: 18:03)

3-Coloring

- Use vertex index to break symmetry
 - But too many indices
- Algorithm:
 - Initialize $c[i] = i$
 - Iterate, reducing #colors each time
 - parallel for all i do
 - Let $k =$ Least significant bit in which $c[i]$ and $c[e[i]]$ differ
 - Set $c[i] = 2^k + \text{bit } k \text{ of } c[i]$



So this is kind of one step of algorithm where you give me a coloring to another color and the goal is to ultimately reach 3 colors because we know it another coloring going to keep reducing you give me some in the beginning you can say ID I is your color ID and everybody gets and try to converge to 3 colors right. So this is one step of it so you have got some coloring given to you and based on your color ID in this case you could also use color ID but we are going to use color ID's which is in the beginning is the same as index ID if you will take the color ID and assign to the new color.

And this is the way we are doing and this is the way we do it everybody is doing it right so we have a old color C of I my index is I. The node number I am working is I can read the array C of I it says what is this color current color is I can read E of I let us says who is its next node right. So every element is looking at the node assigned to you to it and the next in the cycle may be at which ever and we will say let us look at their big pattern of this two color.

And scanning from the right find the first one that is different suppose that is key starting at 0 if the first one is different it is 0 third one is different and so on you are going assign to you the color of two times K + your bit where you are different okay.

So that is $2K$ bit K of your current color before we talk about Y is the process late we take the bit pattern that you have we also take the bit pattern of your next your successor in the cycle we figure out the right most mismatch which is at position K starting counting at from the right and give you the true color of two times K + your current bit value of that position which will be 0 or 1. So the final color you get is $2K + 0$ or $2K + 1$ so it is either order it so let us look at this here I get two patterns 1011 and 1111K = 2 because it is a third position where they are different.

And so this fellow is going to get and that bit is 0 so it is going to get twice of K + 0 it is next color is 4 meaning 107 it is 3. This is one step this is we would not done it is not coming here at all the idea is that you will be keep doing it you will reach 3 no we will reduce by a lot we will come to that but we will this will converge much faster than it seems. And so another example where the bit different is that where big position and the value is one over there it should say 2 times 0 + 1 is copy and paste problem.

So why does it produce a coloring in the first place right not even talking about how many colors we get is it a color that started with a coloring we will to that also but first is it coloring in the first place. Because I am doing this in a symmetric way in a subset right I am going to look at your bit pattern and locally decide what to do with your color. How do you make sure that the next guy is not getting the same color that you are getting or similarly the previous color.

It need not be but it comparing with the next guy so there will be some position K if it is not the coloring what are you saying that my color after that change and your color after the change is the same. Just that addition of one bit is not going to be sufficient for it right to be the same RK must be the same because K is B that is why can be multiplied in fact even for three it will fit it will be only it is greater than strictly greater than.

So if the K is the same for our and 2 colors to end up being the same so this is proof by contradiction so you said that let us say that our color are the same after the change which means that K must be the same if K is the same the bit must also be the same but if we say that the bit differ at by position right. So there is no way that we get the same color at two consecutive vertex positions.

Alright so how many bits? How many different bit values could you reject? Given that the previous color can be fit K bits am I using K for something else yeah T bits previous color fit T bits meaning the color went from 0 to 2 to the $T - 1$ which is going to fit in how many bits $\log T$ bits right. So $\log T + 1$ that is $2 T$ so if the previous T fit in T bits next will be \log of T bits. So if the previous you started with 2 to 32 different colors which fit in 32 bits the next time you are get 32 different colors 0 to 31 which fits in 5 bits.

And the next time it is going to fit in \log of 5 so it goes down really fast how fast we will have to wait and see. So we are just discussed figuring out the coloring and the number of iteration that you are going to take is log star effect because every time you are reducing the log of log of that previous count because the number of bits is going down the lot number of bits as already log of

different numbers of elements different number of colors the total number you are taking is \log star of N and everybody is busy every step.

So the work is $N \log$ star okay but the number of colors will not go below 6 because you are going to go to $\log T + 1$ right if you come down to 3 bits we are not going to go down so what happens if you reach 6 and this process starts after 6 it does not go down. So you reach 6 bits and you know you can reach 3 bits 6 color sorry every 6 colors you can you can reach three you say I am going to remove color number say starting at 0 then 3, 4 and 5 we will do this in 2 steps first remove 3 then 4 and 5 or whatever 5, 4 then 3.

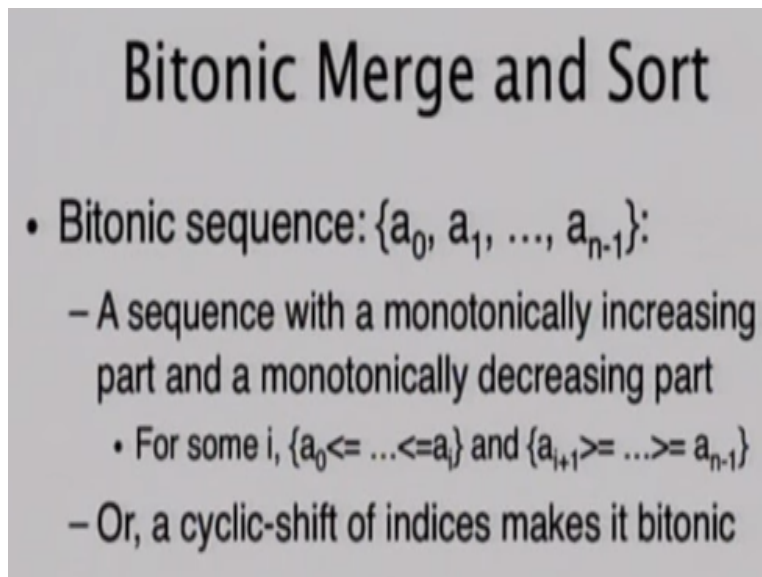
So to remove the color 5 what do we have to do everybody would does not have five still everybody will 5 to do something right. So neighboring colors will can be only 2 of 0, 1 and 2 so there is only two of them so you always have the third color if it is 3 or 4 then you can arbitrary use but later on you are going to get fixed. So that is it coloring and can you prove that although it is reasonable and \log star N is really small but the ideal optimal would be order N and it is possible to that okay.

However the best known algorithm increases if you want to make it work optimally time may go on okay. Alright not let us any question on this let us go to sorting the general idea is in fact this itself can be used as sub routine lots of times where you take the bit pattern and you make some multiple of the bit size okay where the difference is that is the crucial part. And even at higher level the idea is that you do something is based on the bit pattern and you ensure that what you do is not going to same as what the next guy does.

That is what the symmetry construct so similar approaches can also be used when let us say everybody has to open it but you can if you have a arbitrary graph you do not always need to break symmetry some places need to break symmetry on those you are going to have to do something of this. If you do not what do you can you get it in fact for the last step you do need so how do you get the predecessor if you do not.

Basically in order one time order N work you can create that array so parallel sorting we have already looked at some of them there is some quintessential for a solid reasons parallel sorting algorithm we will talk about one of those first.

(Refer Slide Time: 32:04)



Bitonic Merge and Sort

- Bitonic sequence: $\{a_0, a_1, \dots, a_{n-1}\}$:
 - A sequence with a monotonically increasing part and a monotonically decreasing part
 - For some i , $\{a_0 \leq \dots \leq a_i\}$ and $\{a_{i+1} \geq \dots \geq a_{n-1}\}$
 - Or, a cyclic-shift of indices makes it bitonic

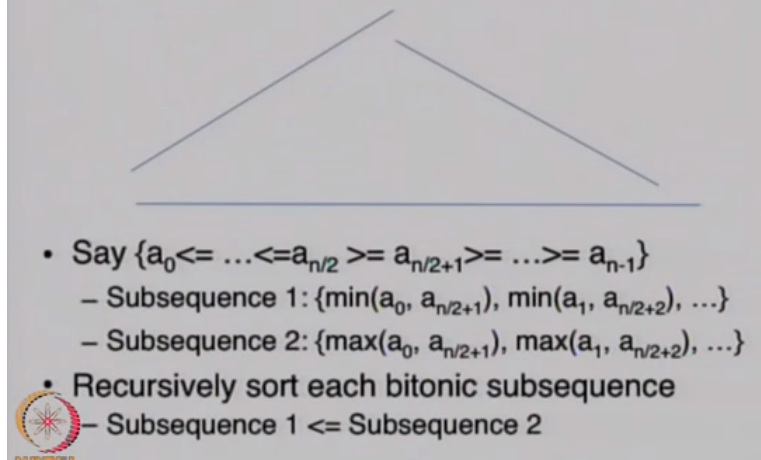
It is called bitonic sorting and then I had shown you a CUDA code example of bitonic sorting the promise that will come to it this is where we come back to it. So bitonic sequence is the sequence the name comes from fact that is 2 tones that is increasing and then decreasing.

Strictly a bitonic sequence is allowed to be a cycle so increasing decreasing module of the size so wherever the smallest is you start scanning to its right and it keep increasing and when you run off to the end you come to the other side either it will keep increasing or at some point start decreasing and then keep decreasing until it reaches the same point. So this bitonic in a modules for the time being let us just consider the pure 2 tones where it increases from 0 all the way to the somewhere and in fact in our case we will even take a more limited kind of bitonic sequence that it increases at the end by two and then decreases.

So I have got one such bitonic sequence and I want to sort it can you do better than just starting off from scratch merging the opposite direction how long will it take $\log N$ time here.

(Refer Slide Time: 33:55)

Split Bitonic Sequence



So it can be done faster but the algorithm am going to show you which is the historic bitonic exhorting method is and in fact it will work even when you have the cyclic shift you take and element from one half of the subsequence and another half of the subsequence and compare those two the smaller one goes one side the larger one goes on the other side and you repeat it okay.

So subsequent one will be formed by minima of pair by pair pairwise comparisons and for each of the same comparisons the maxima will go in the other list this would not sort but every element in one of the subsequences will be less than the other one okay let us see this pictorial we have a so in order for us to do comparison in all the Y axis I have just shifted the second tone to the left the comparison is first here second here third here and so on okay.

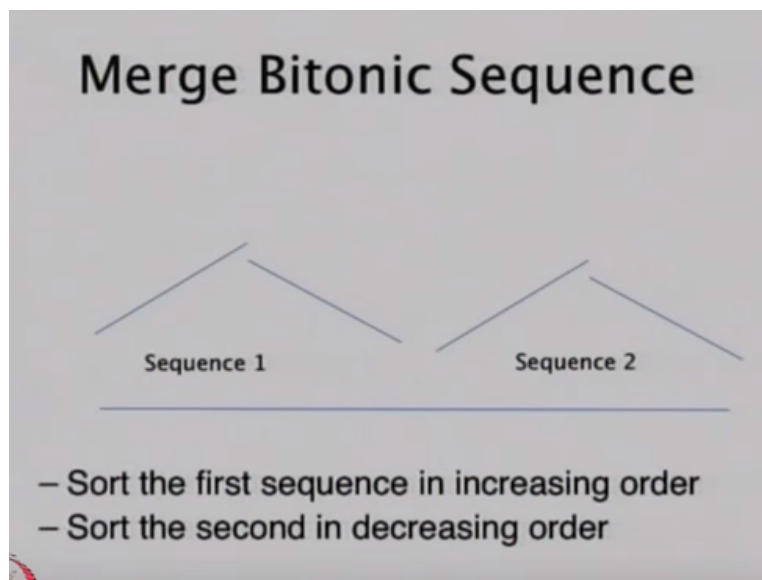
So where is the subsequence one of the lower once where is the subsequence to be in the one sequence one is the left half of your sort out result subsequence 2 is a right half of your sorted results but we have to sort subsequence one sequence 1 is the bitonic sequence of half of the size increasing and decreasing and subsequence two is decreasing so it is the mirror image still a bitonic sequence is half the size.

So how long will it take to sort it order and work order 1 order log in time order one time but we cannot start working on subsequences until the first split has happened after split you can start

making those two subsequences sorted in parallel but how many sub steps will be necessary $\log N + N \log N$ is going to be the amount of work involved but everybody is busy in every step okay. So in $N \log N$ time and $\log N$ and $\log N$ time you can sort a bitonic sequence what if you started with general numbers you can use bitonic sequence sorting to sort any general sequence.

You can if you can generate a bitonic sequence from a random sequence so if I had a bitonic sequence can I merge into one there is two bitonic sequences I have no about their relative values with respect to each other but then what if it is not right we have to merge somehow we have to go back to the original merge algorithm.

(Refer Slide Time: 38:40)



Or we stay with the bitonic idea where we take the first bitonic we sort it sequence just like we know how it you take the second bitonic sequence is nothing more to do one is sorting increasing the other is started decreasing you have got a bitonic sequence okay. So if I can sort a bitonic I can also create bitonic sequence and so this is eventually the final algorithm and lots of people implemented in the hardware although it is not the most efficient sorting algorithm.

It is primitive cells are so simple that you can implement the hardware water software very easily.

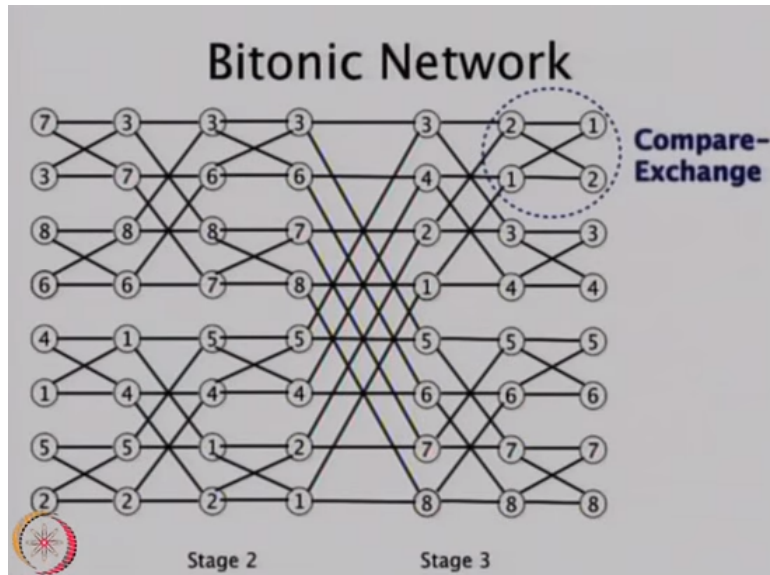
(Refer Slide Time: 39:29)

Bitonic Sort

- Sort each pair
 - alternately in increasing and decreasing orders
- Every sequence of length four is now bitonic
- Sort recursively
 - Again alternate increasing and decreasing orders
 - Forming bitonic sequences of length eight now
 - And so on ..

So the idea is you start with two element sequences which is always bitonic and from two elements you create four elements sequences by merging them from four element 8 element sequences until you have made N element sequence okay after you made an N element bitonic sequence you sort it just like we cleared off okay.

(Refer Slide Time: 39:59)



And so pictorially this is how in fact hardware for this would physically look is it takes a bunch of input someone in and produces the output on the other side. And each of these units which is compare exchange unit takes a pair and puts the minimum of those pairs at the top and maximum at the bottom or the other way around it can be control whether min goes at the top of the maximals at the top.

And so just following that example this case the left most is your input whatever order you wish and so you take the first bitonic sequence which is 37 and sort it so in this second column it you creating bitonic sequences of length 4 by taking length 2 bitonic sequences and merging it. So 37 is sorted in increasing direction and the next pair which is 8 and 6 as we sorted in the decreasing. In this case it came that in 7 3 came opposite way but 8 6 came the same way.

Similarly 4 1 has been sorted in the increasing direction and 5 and 2 sorted in the increasing and as a result of which we are going to get bitonic sequences that are longer okay. So now we have got 3, 7, 8, 6 and 1, 4, 5, 2 and so independently we are going to solve 3, 7, 8, 6 in a increasing order and 1, 4, 5, 2 in decreasing order.

So now we have a bitonic sequence of the full we have to go through the sorting of bitonic sequence of the second and which is again comparison at $\log N$ level it looks very clean and that is also reason why it is implemented in hardware is that it has a very butterfly like communication like you compare with that to I away that to go $I - 1$ of it and $I - 2$ away and then at the end of it you have generated the sorted results.

I do not see pointer jumping there no here actually is 2 raised to the 0, 1, 2, 3, 4 so it distance 8 away then 4 away then 2 away no it is not opposite it is not actually going to 2 to the increasing number of distances away you are not going from 2 to the I 2 to the $2I$. We are going to the 2 to the I to 2 to the $I + 1$ so it is not increasing at fast it is not \log of the height but it is order of height okay.

So that time taken will be $\log^2 N$ $\log N$ times \log and Y whatever one says will be we just figured that out if I had an $N \log$ bitonic sequence ahh then how much would it be to sort it 1 bitonic sequence where we make one by one one to one comparisons and divide it into 2 bitonic sequences $\log N$ time and $\log N$ work. So last stage $X \log N$ time how much does the previous stage take.

We are only taking about time right what we were doing it in the previous stage sorting N by 2 big sequence here and N by 2 big sequence here this in the increasing order this in the decreasing order. So \log of N by 2 right which is $\log N - 1$ and next we take $\log N - 2$ stages $\log N - 3$ stages and so on. So sum it up \log square N stages okay and everybody is busy every time so $N \log$ squared N time okay let us stop here and the if there are any questions on bitonic sorting.

And then in fact let me go through the end of it you can see this as a recursive recurrence relationship your basically saying that you are doing $\log N$ work here and then taking a something this is for the time to do N things is to sort N by 2 things and then we $\log N$ additional stages to sort the final bitonic sequences okay.

So then we can solve this and get not everything will be really CUDA friendly but this idea is in fact the lots of people now essentially spawned the mushroom of research on this area to design algorithm suitable for this architect because this architecture is becoming the wide spread. What changes to the original algorithm you need to make? So that it comes in the SIMD environment okay