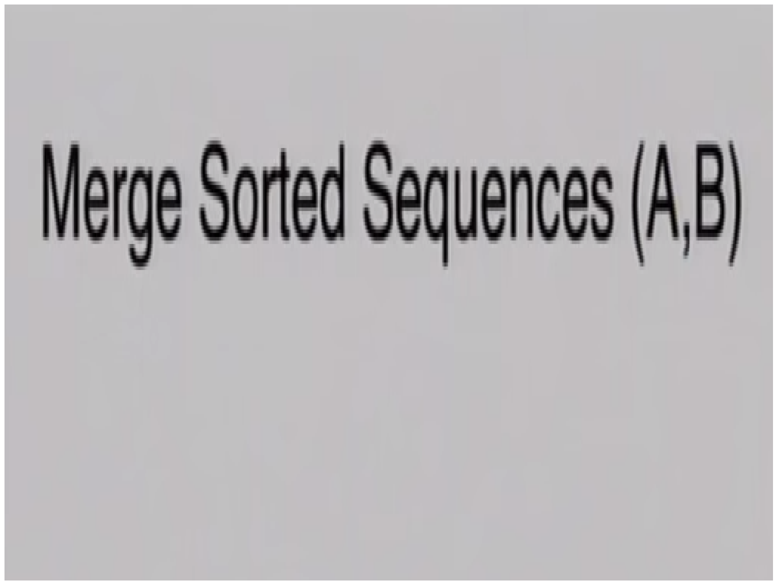
Parallel Computing
Prof. Subodh Kumar
Department of Computer Science & Engineering
Indian Institute of Technology – Delhi

Module No # 04
Lecture No # 19
Algorithmic Techniques (Contd.)

Organizing parallel computation which is tree based and anther that we have been we had started one which is partitioning it to smaller problem and then solving that independently.

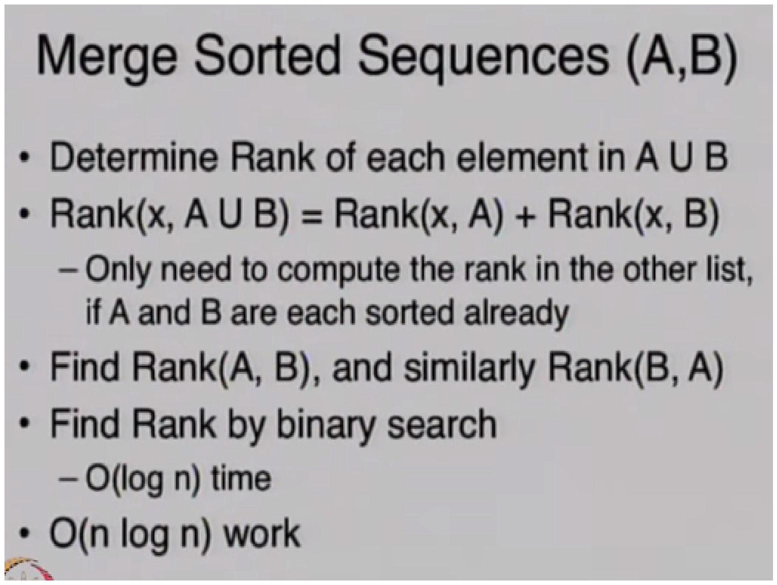
(Refer Slide Time: 00:39)


Merge Sorted Sequences (A,B)

So this is the example that we are going to start with tell me something tree based how there is many way for doing this. So we are going to determine every elements rank in the final position right I know my rank what is the rank the number of elements less than me. It is first element I know the rank is 0 if I am tenth element and the rank is 9 so 9 elements are less than E.

So I know the rank if I am in A my rank in A I just need to find my rank in B if there are thirteen elements that are smaller than me in A and 16 that are smaller than me B then 29 elements small than B okay so I am the thirtieth in the final array. And hardware find my rank in the sort it final research find it okay.

(Refer Slide Time: 02:19)


Merge Sorted Sequences (A,B)

- Determine Rank of each element in A U B
- Rank(x, A U B) = Rank(x, A) + Rank(x, B)
  - Only need to compute the rank in the other list, if A and B are each sorted already
- Find Rank(A, B), and similarly Rank(B, A)
- Find Rank by binary search
  - O(log n) time
- O(n log n) work

So eventually my rank in the union is my rank in my array which I already know plus my rank in another array which I need to complete each element can compute in log in time there are N elements and so N log in work N log in Time. We will be able to compute the entire merge list right so once I know my rank how do I create the merge list just put it my rank is thirty then I put it prove myself thirtieth position as it required concurrent read concurrent write what does it mean.

Concurrent read would be needed because all these different elements are log search at the same time the concurrent write is not made once I know my position my position is different from everybody else is doing this position if things are not unique what you have to do that it came to the same rank then concurrent would be needed the both would all too needed. One of the array would different need is that enough.

How do you compute your rank in other element? Find the multiple things that are equal to you and where do you find your position so if they are equal then you know count down. One of then that is the thing right you cannot do one of them without concurrent right I have got multiple copies of 39 array as well as in array B so a given thirty nine array A must come to rank in array B right.

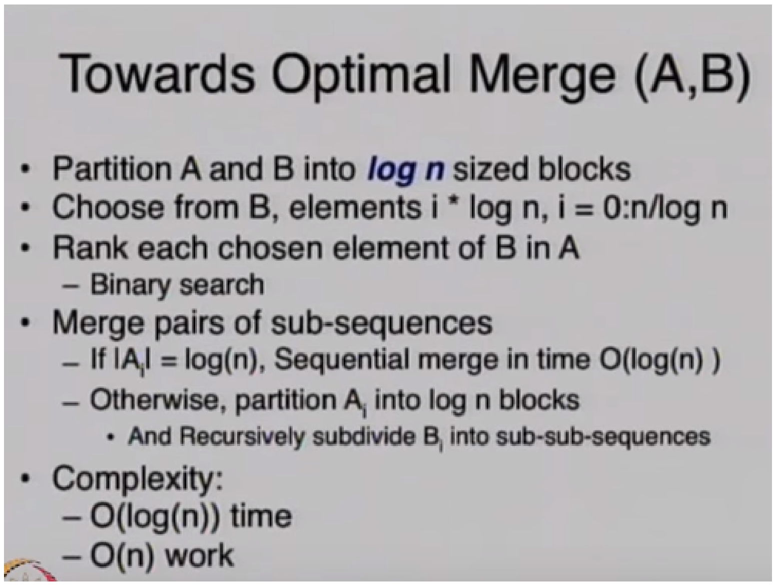
And a given 39 must also come to its rank in array B how do you figure out their respect ranks do they count the 39 scenario B or they do not okay. So that leaves what happens in the 39 and the other array right how do you make sure that they come to ranks greater okay that is the interesting thing to sink about I will leave you to think about and you think about it let us talk about the basic thing right and then there are various way in which you would want it right yes.

They are all in right direction think about it let us now think about that in class let us talk about how to make it so log in un log in is correct you can find these rank it is quite efficiently by mind research in log in time and in log in work okay. Can you do better that was question? If you do then again you are following some kind of binary pattern which we take you log in time right so we want to improve currently our work is bad.

We wanted to be good as order M and sequential we do it in order N time may be not clear and the time we have to log N maybe we can do better okay. So if you are trying to do better what would you do I am not sure how why is it taking that much time? sorry that much work? Lots of useless comparisons right we know a lot about so we taken this N things independently we are doing log in search on it we know their relative positions there with respect to each other right.

So we are not using it there are lots of comparisons that we already knew of and may not if given element is found at third positions in the other array then elements to the right of it cannot be found in the second position right it has to be found in the third position okay let us try this.

(Refer Slide Time: 10:00)



Towards Optimal Merge (A,B)

- Partition A and B into ***log n*** sized blocks
- Choose from B, elements i * log n, i = 0:n/log n
- Rank each chosen element of B in A
  - Binary search
- Merge pairs of sub-sequences
  - If $|A_i|$ = log(n), Sequential merge in time O(log(n))
  - Otherwise, partition A_i into log n blocks
    - And Recursively subdivide B_i into sub-sub-sequences
- Complexity:
  - O(log(n)) time
  - O(n) work

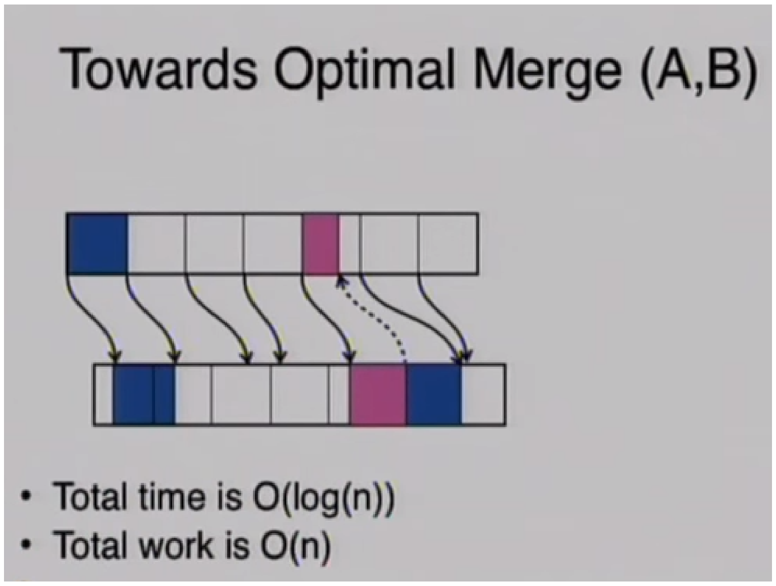
Suppose we partition it into blocks okay and although it says log in size blocks will figure out what is the good size and we are going to choose from one of those arrays let us say B one element in each other of this blocks okay so log N element we can start with the first of the boundary condition does not matter start with first also log Nth element 2 log Nth element, 3 log Nth element all the way up to be other hand okay.

So now I got by N login element of B that i have chosen if only want try to rank these elements in A how long would you take? Just the same algorithm as before instead of N elements to rank I only have N over log in element instead of. So log if this right time, time is log of this is not it time is log of size of A. Time is log N correct and the work is exactly N by log N time log of N by log N right sorry Log N is the other side.

N by log N times log of N and the work has come down to order but we have not done yet right what remains to be done we have found for every sample in B were it should be found in it what about the element between the sample the found between those respective spots very found those let call them pivots or sample right. So element between samples will be found between the sample locations okay.

How many position there maybe there are how many elements between samples log in right so log in things that approximately login that we do not know the rank of so we need to find their rank but how many elements of the other array may there B may be we need to find out rank of each of them.

(Refer Slide Time: 13:00)



Towards Optimal Merge (A,B)

- Total time is O(log(n))
- Total work is O(n)

Just look at this picture okay I have got A and B and I have taken in this case the upper one is B based on description I had lower one is A I have taken every log in F element is B located in somewhere in A right only thing I know is that am not going to cross okay somebody is location at location A location B then everybody to its right will be located to it right but it can be anywhere.

For example if I wanted to figure out where the elements in blue on the top will go somewhere between those two equations okay similarly elements between every color the second third block from the right of the top elements there will be anywhere in this bigger than log N size so if we were to do binary set then we would take log of size of this we would not log N necessary.

It can be as bad as N - log N right all of them are to the beginning and then there is chunk left all of them in the remaining are they working in the fresh piece. So if you going to take approximately N things right because N over log N things are only log N elements of B know the rank of remaining N – log N we do not know that so that many ranks need to be funny okay and some of them may take a long time some of them may not what tis the longest that anybody could take?

Because other this bigger A can be as bigger than okay how many of them can they be are you sure? How many of them from the blocks of B would have big things to merge with. We are trying to figure out the work right the time is Log N assuming that I have enough processors that every element can be done independently then the worst case time to look for in the bigger A or one of the big logs of A is log N.

But if you did that how much work would you be doing you will be doing log N work there are approximately N things whose if there are enough number if there are half of them are going to have big chunk to be in a big chunk then it is N log N. There is be only constant number of big chunks so big is order N then B constant only big size. I said suppose all the pivot happen to be go to the beginning right then only one way suppose half the pivot go to the beginning and the remaining are distributed at the beginning.

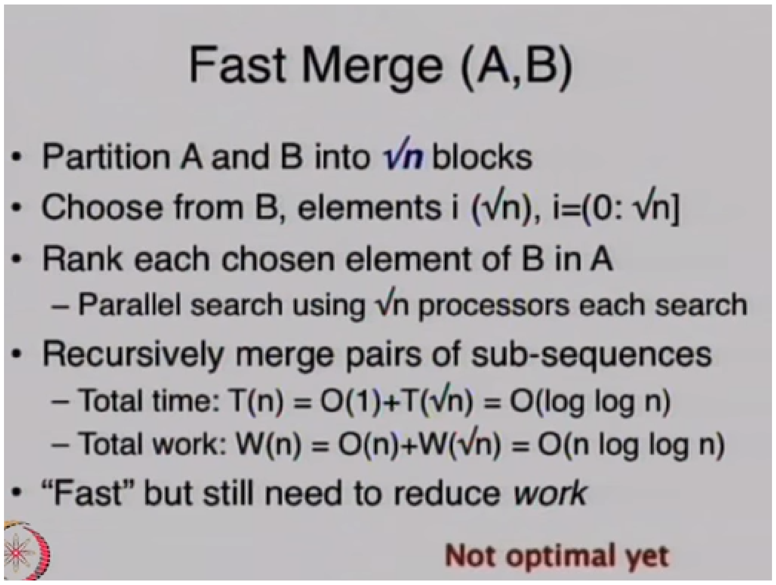
But you can say for sure is that you to the big block that you found it can be also be broken into size of log N if we know where these elements are in the other block B then what are we guarantee. All the pivot between this big B block are going to the same block right at most in the log N size right. So now there are when you try to find the elements of the block here and find the ranking here and you discover that this is too big then you say let us instead find them in this block okay.

Another result of which now those two blocks will be found in two sections of the same block and so total time will be Log N total work will be order N you can do sequential merge at this time because there is only log N things to merge okay and I log in different things and each merge sizes order log N.

Now can you where can you do better only time right because you could not cost of do better than order than work otherwise you have a sequential sub linear merging algorithm. So where are we spending time to do this search that is the where the log N time is being spent. Can we search through a smaller section may be do we actually search through a whole thing can we search among the sample along the other end.

Some of that okay so you do not really because it is taking too long to search to this entire second array will search through some of them and hopefully then remaining elements final ranks still needs to be found and that can be done fast enough so we can expect better.

(Refer Slide Time: 21:03)



Fast Merge (A,B)

- Partition A and B into $\sqrt{n}$ blocks
- Choose from B, elements i ($\sqrt{n}$), i=(0: $\sqrt{n}$]
- Rank each chosen element of B in A
  - Parallel search using $\sqrt{n}$ processors each search
- Recursively merge pairs of sub-sequences
  - Total time: $T(n) = O(1)+T(\sqrt{n}) = O(\log \log n)$
  - Total work: $W(n) = O(n)+W(\sqrt{n}) = O(n \log \log n)$
- "Fast" but still need to reduce *work*

Not optimal yet

So let us do this you are going to break instead of log in blocks we are going to break into root N blocks if you just did the previous algorithm with instead of log N root N we have that log N just take root N. Root N blocks and root N sample pivot takes root N elements every root nth element now right and we are going to rank it in B sorry in A sorry we are taking it from A how can be speed up that search?

And reason for root N will become clear when you think about how can we speed up that is root N elements root N searches we have to made right so we have less searches to make now in the first case we were making N by log N searches now we are reduce number of searches that we are making which kind of means that we can assign more process also if we have N processors we could assign root N processor as per search.

So if I had root N processor and I wanted to find one thing in an array of size N what is the binary search of the variant of parallel binary search where are using P processors tell us how fast can you do it log N to the base root N right which is constant I so we are spending enough processors to speed up the time because we have done the work brought it to order N we are trying to get that the search time down.

So the search was taking too long let us divert more processors to search but if I had N by Log N things then it would be probably be too much work. If I only root N things then with N processors I can in constant time figure out where out where this element is okay and their root N are searches to make and I am going to make root N processors to each of the searches so N processors in order one time will have found the pivots okay in constant time.

Now what remains to be done we probably similarly do from the other side also we basically going along the same path did we end of spending too much work here how much work was done in this step? There are N processors you did it in constant amount of time could not have done more than order than work right. So far time is great work is good great but there is work to be done still suppose we recursively merge those pairs.

We have root N block here root N block here we merge two things of root N and we have root N such merge N to do and so we can say the total time is order 1 + time taken to process root N which is long N still not order 1 because this enough work remaining to be done the work will be N log N. Because we did N log work there is enough times you have to do order N work that it accept to get that greater than A.

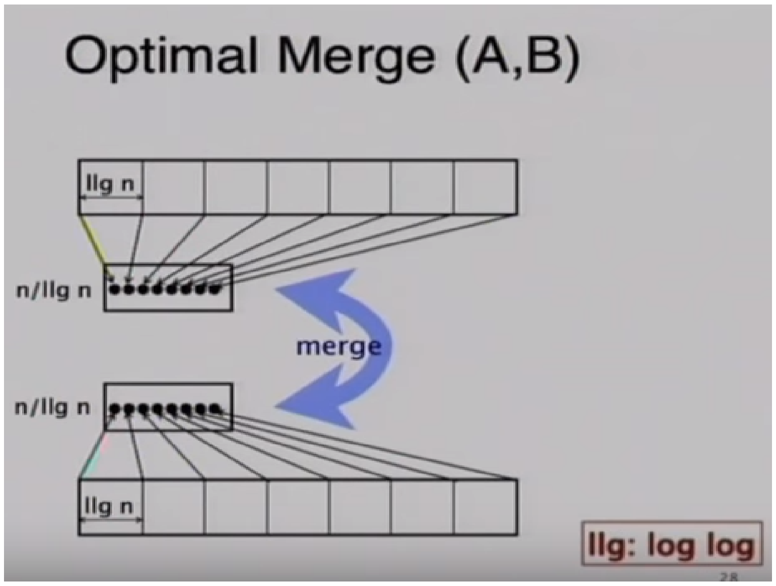
So we have made it faster right log N is practically constant for all practical purposes but the work as gone up slightly still N log N and imagine being almost linear so far we are taking about portioning right and going to tell into another technique were you combine to different algorithms. One which is fast but N are going too much work one that is work efficient not that fast and it combine it to take something that is fast as well as work efficient that is a different technique okay.

Let me stop at this point and think about how you would combine we have two algorithm now one which is root N based which is really fast not perfectly work optimal but not perfectly work optimal. The other one is work optimal but not as fast can we combine these two half to get the best of both works so we will stop here.

Let us being ahh so what we are going to try to do is (refer time: 27:02) do not use the past but sub optimal algorithm on the entire data okay. Use it on small pieces of data ahhh in a way that it does not add up to greater than B optimal work okay. So we are going to make data sizes really small log N so we are going to divide A and B into sizes log N right and sample these how many are we going to get in the A prime and B prime sequential which are the samples and over the log N. So now instead of the problem of size N we have problem of size N over log log N can we take that and solve it using the fast but less than optimal algorithm.

We had N over log N elements to merge how much work would it do order N right and you would not be done yet though. But because out blocks are so small there will be little work left to be done right we made blocks only log log N size. So in this case the approach works because we were not that far away from optima so let us look what would happen?

(Refer Slide Time: 28:42)



Optimal Merge (A,B)

So we have A and B we have sample it every log N now so just because to fit log N into small space calling it LLG. So where you see LLG is log log N and so each size is log N where N over log N it is we are going to take every element in this block and make new sequence A prime and B prime and merge this using for about element what it needs to be done.

We can merge sequences from their end later is that too much left to be done so let us look at one of these elements we have found one of the samples of B among of samples of A we have not look for.