

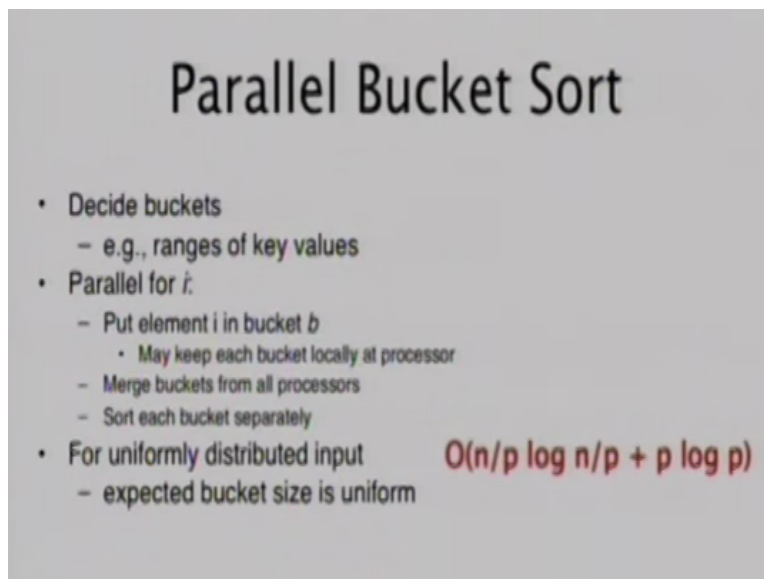
**Parallel Computing**  
**Prof. Subodh Kumar**  
**Department of Mechanical Engineering**  
**Indian Institute of Technology - Delhi**

**Module No # 06**  
**Lecture No # 30**  
**Algorithms, Merging & Sorting**

Okay so let us see, we have work in this. We take all the elements and divide them into buckets. How long does that take, is not included. Actually  $N$  value all processes take just one chunk and keep giving them into the corresponding bucket. How many things go into the bucket? Not listed that is not my question like which is like, when the processor put something into a bucket.

How many different sources are elements coming from into a bucket? Professor student conversation starts will be  $T$  different places, professor student conversation ends.

**(Refer Slide Time: 01:12)**



**Parallel Bucket Sort**

- Decide buckets
  - e.g., ranges of key values
- Parallel for  $i$ :
  - Put element  $i$  in bucket  $b$ 
    - May keep each bucket locally at processor
  - Merge buckets from all processors
  - Sort each bucket separately
- For uniformly distributed input  $O(n/p \log n/p + p \log p)$ 
  - expected bucket size is uniform

You need to put them in a proper place. Where would you put? You need to figure out, how many elements you are going to set. So before you start sending them, you have to do some accounting. So you figure out I have so many elements in bucket one, so many bucket 2, so many bucket 3, so many bucket 4. How many things do I generate?  $B$  numbers because I have  $B$  buckets where I am generating sub number of elements into the number of  $B$  or  $B - 1$ .

Others who are also generating B numbers, everybody is generating B numbers and are the number of elements in the B buckets or a given processor. Each processor is taking N by B chunk and saying you go to bucket 1, you go to bucket two, you go to bucket 10, you go to bucket 14 and so there are B buckets. We are using B the number of buckets is equal to the number of processors.

So we are generating T numbers, every processor says for every bucket. How many elements in there bucket? Now we need to figure out where these elements should go, so we need to figure out the sum of prefix. Some of bucket 1 and one side effect of that is that we know the sum and number of elements in buckets 1. So at the last processor, we had a prefix sum of bucket 2, prefix sum of bucket 3 and prefix sum of bucket.

Each prefix sum will take long T time and pre prefix sums to do. So it is  $P \log P$  to do those prefix sum at the end of this you know where to put. Where to start putting elements? Every bucket for your processing. Professor student conversation starts, for this to be done, we do not need to do the internal prefix sum within the buckets. You just need to know the counts, no prefix sum. You just need to know the count of the whole.

Then prefix sum over the bucket level prefix sum across the buckets. But you still need to know, no you still need to do prefix sum across the bucket. So where do you start writing bucket 1 for that you have to know the sum for the bucket 0 of bucket, 0 the total number of elements that are going to go in the bucket 0 which was the side effect of the prefix sum. So why to do this prefix sum for bucket 1?

Within the bucket or how do we know that in terms? You have to know, so you are generating B numbers are the numbers of elements in B different topics. Why should you write that at the end of the day? You have to generate sorted sequence. So sorted sequence means bucket 1 followed by bucket 2, all the bucket 1 all the bucket processes bucket one followed by all the processes bucket 2 followed by all process bucket 3 and so on.

So to start writing things on to bucket three you have to know, where the bucket 3 begins? Bucket 3 begins after bucket 1 and bucket 2 are exhausting. So I have to figure out when that begins? So that is where I need the prefix sum of bucket 3. So I as processor number 3, number 10, we will know where to put my buckets three elements. So the Log square B in comparison to sir in adding one more to get 3.

But inside the bucket 3, they are writing parallel, that some training making absolutely. So why they need to do a prefix sum of bucket 3? There is 2, there are B prefix each buckets, N - 1 buckets inside a bucket also. So when we do this B prefix sum you are going to generate sum of bucket one and sum of bucket 2 and sum of bucket 3. Then you do the prefix sum to figure out, where bucket number.

So there local prefix sum which says with my offset processor number 10 for bucket number 5 goes to this position processor number eleven goes to that position. But where does bucket three begin that is sum of all this buckets.


**(Refer Slide Time: 06:28)**

### Sorting if $n > p$

- Simulate  $n/p$  virtual processors per processor
  - like PRAM simulation earlier
- Or, divide element into  $p$  blocks
  - and locally sort  $n/p$  elements (of each block)
- Then perform bitonic sort of blocks
  - compare exchange becomes compare-split

**$O(n/p \log n/p + n/p \log^2 p)$  time**

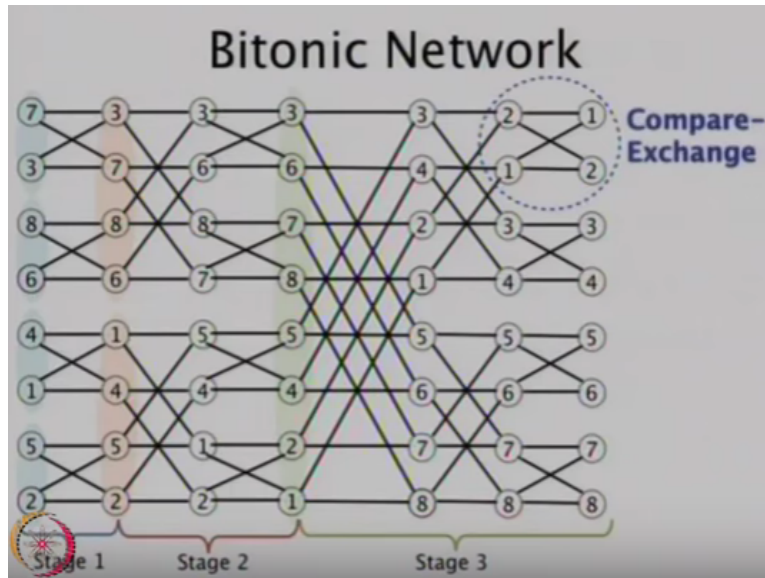
*local sorting       $\log^2 p$  comp-splits*



How does bitonic sorting works? So you take the Log N time to merge two bitonic sequence to one. Similarly it takes Log N times to split it into top half and right half on the one side into left half and right half to sort it. Not to simply split it into top half and bottom half. But totally sort it

Log square Professor Student conversation ends. So if you recall, in fact why do not we go there?

(Refer Slide Time: 07:09)



This one the last stage to  $\log N$  time, stage before that took  $\log N$  minus time stage. Before that  $\log N - 2$  time and so the total time, total number of phases where you are doing compare study, compare exchanges like each column is a comparing stage and braces at the bottom give you this different stages.

So all the compare exchange added up across the stage meaning all the compare exchange column added up cross the stages will be  $\log N$  time by  $\log N + 1$  divided by 2  $\log$  square. The  $\log$  square compare exchange sets that are done each set is done in parallel, because everybody in parallel is separating the exchange in something. So now when we compare to this. Professor student conversation starts, so that is what I was asking this is by ton X sort and cyclic also.

Yes it works on cyclic also. What is the sink? You asked something related to cycle I asked that I had shown you an example in class of just your sequence increasing and decreasing and I asked you that of course it will work for all bitonics sorts even if it is a cyclic shift of your increasing and decreasing. I asked to go convince yourself, prove to yourself that the same mechanism, the same algorithm works even if you have shifted bitonic sequence.

You can still sort it okay Professor Student conversation ends. Log square P comes from that, many stages. But now instead of compare exchange which was done in one time, you are going compare in split. So compare and split is compare exchange over some exchange over P things. So there are Log square P stages. Each stage now is taking N over P time, instead of one time which N over P Log square. Professor Student conversation starts, some like real world system for example, in all these sorting we are done. Whatever we are telling, we are getting the data already inside. Because we basically think P ram as our in back of our mind all the time. But that is not always the case.

Because I mean it if you have to sort hundred gigabyte of data, you will never have machine. Hundred elements of, never Say Never. But yes I mean, so you will have to try this at some point of time within your like in between your sorting algorithm itself and if you have making. For example, this kind of parallel sort each machine is receiving convertible machine and you have not depending on only one particular side of element point of time.

If you have doing it rightly, then you may end up writing, when you are doing different at the same time, when you are different writing on the list at the same time, it will basically kill your system. So how will you there is a entire area which is based on this data structures. This complete area anyways gets into P model. We are discussing not exactly, so let me what I mean even for a single fore machine sequential algorithm.

There is an area that is (()) (11:29) just data structures help often also called often computation and just as we see in that, we cannot simply take this all the time. Sequential algorithm is expected to perform well parallel same things work. We cannot simply take basic sequential algorithm and expect it to perform well in that (()) (11:54) model and so now we are having the same problem here you simply cannot take (()) (12:03) model and expect it to always perform well in (()) (12:06) model in the p ramp context.

That is the area that is not well studied at all (()) (12:12) model in the parallel context, it is well studied in sequential context sir, I was like find the any system to work at all you have to think about, if you take this up in the core or touch. Actually there used to be a course called file

system and data processing it is not I have seen in past few days that is where things like this was discussed.

You can actually have an entire course on offline of course computation or out of core computation. So my point was because of this an entire that is absolute necessity for like any kind of parallel computation that you are dealing with large things should not be discuss it in this course. That is the thing, we will not have time. There are other big swords that we have not touched upon, I would like to go through them.

Because just setting up the model for the trio and in fact as I also earlier said in the context of parallel algorithms P Ram models that area is not well studied at all. So there is not even enough background material to cover, if we are thinking in context of parallel models, so that would make lot of sense. It can then pull in concepts into parallel domain, but doing that in sequential context itself is a good topic should be covered in some course.

So back to the same question, how do you solve when small data cannot fit in P Ram in the model? You are not covered, you look at out of core sorting. we can perform two way merging, there are merging technique. But you have still to think your cost model changes. Instead of asking how many steps of computation? You are doing, which you still do you because like you are sometimes so much computation, so much communication, and so many blocks of data.

There is notion that there is fixed block that you can read if the block is continuous, you read it together if not continuous it becomes different. So that is the basic idea but and some time algorithms can change significantly to account for you too have low cost on that correspondent alright anything else that you need to odd even.

**(Refer Slide Time: 15:09)**

## Odd-Even Exchange

- Divide into  $p$  blocks
  - Sort each block locally  $O(n/p \log n/p)$  local sort
- Repeat  $p/2$  times  $+ O(n)$  time  $p$  comp-split
  - odd and even phase compare-split
    - each done on  $p$  blocks by  $p$  processors
    - each processor sequentially splits  $2n/p$  sized block

It is basically  $P^2 N$  by  $P$ . This is not completely obvious looking at it, but it is correct it is  $P^2 N$  by  $P$  not exactly, of course  $P N = N$  by  $P$ . So you have to repeat the same  $P$  times and last step is to keep  $N$  by  $P$ . So whole time  $P$  what happens and what about the first one? I think you have got it basically right, but I did not want to give you the answer why do not you think about?

It is in fact on the slide, if you just do the analysis properly  $P$  process constant operation that you have to find out. So at every level, you are doing the same amount of work as the previous slide divided by two which is going to add it is like the binned tree. You have got  $N$  nodes I did not have the batchers odd, even merge.

(Refer Slide Time: 16:19)

## Parallel Splitter Selection

- Divide  $n$  elements equally into  $B$  blocks
- (Quick)Sort each block
- For each sorted block:
  - Choose  $B-1$  evenly spaced splitters
- Use the  $B*(B-1)$  elements as samples
- Sort the samples
  - Choose  $B-1$  Splitters
- Arrange elements by bucket in output array
  - Count the number of elements in each bucket
    - Perform prefix Sum of counts; Reserve space per bucket
  - In-place
  - No bucket contains more than  $2*n/B$  elements

Why did you do the quick sort of each block over? Why did you do parallel data selection? So the idea is that from each block you pick out sufficiently distributed set of numbers. So you would like some numbers at the lower end of blocks, some numbers at the middle end of the blocks, some numbers at the higher end of the blocks and there are probabilistic methods that do that, but one way to do that simply sort that block quick sort.

Otherwise does not matter it is local sort, that is why it is quick sort it is supposed to work well that pick every items. What you are picking from every one of the blocks and now you have got  $B^2$  items? These are sorted and you need to sort them. So there are  $N$  things you want to merge  $n$  things into one. Because of  $N$  things already to be things that you are getting is already sorted.

You do it better in  $B \log B$  time, but for the time being think. Professor student conversation ends.  $B^2$  things, so you can do it better than  $B^2 \log B$  time. But because that does not add up in the final analysis just leave it at root cause and then once you have figured out this splitters which in the high probability is going to distribute the elements into  $B$  relatively uniform buckets you simply figure out where your element goes which is prefix sum.

After prefix sum that is what  $\log B$  comes from. Professor student conversation starts, here we have determines to guarantee that no bucket will contain that is why we did that is why this initial sort is necessary, even if we did not there are probabilistic methods, where you say you do not initially sort.

But you say that no element is more than some  $B$  by too away from its original position. Then you can come up with another bound on how far and that any two buckets maybe from, Professor Student conversation ends. And this is what we were looking at

**(Refer Slide Time: 18:54)**



# Radix Sort

- Bucket-sort by each key(part)
  - Sort stably
- For each key, find its rank
  - Count number of keys ' $<$ ' in the sequence
  - Count number of keys ' $=$ ' before it in the sequence
    - Use parallel prefix sum for
    - For 1 bit key:
      - $\text{notbit} = \text{lbit}$
      - $\text{psum} = \text{escan}(\text{bit})$
      - $\text{nZeros} = \text{psum}[n] + \text{notbit}[n]$
      - $\text{nBefore} = (\text{idx} - \text{psum}) + \text{nZeros}$
      - $\text{Rank} = \text{bit} ? \text{nBefore} : \text{psum}$

This is clear how this works, so you did not look at it. This is for single bit only, I have got 1 bit at 0 and ones and I want to know for my position or my number which is 0 or 1 where should I put it in the output it can be in fact be done in place once everybody knows is the number where it needs to go to the parallel invite there. So the first thing negate the bit to do a prefix sum and basically is doing prefix sum in zero.

Because if you negate the bit then zeros become 1 or 0 becomes once. Then you work doing a prefix sum undergoes is Escan for exclusive scan, something which we saw in open headed context meaning that you are prefix summing everything until you note counting right before. So P sum is how many zeros are before you and total number of 0 is P sum of the last guy.

But it was number of zeros before that and if the last guy is also zero, we need to add one to it. You had last guys bit, last guy is not bit negative of the bit or negation of the bit. We also need to know, how many 1 there are? At the end of the day you need to know, how many 0 elements there are? If you are a 0 element, then you need to know how many zero element are before you.

If you are a one element, then you need to know how many one elements are before you and the total number of 0 elements. So if you want to figure out how many ones are before you, take your index and subtract how many zeros or before? There are so many elements, before you if

your index is 5, your 5 element. Before you of which you know, three are zero are supposed to be one.

So index - P sum is a number of ones and to this you are adding number of zeros, total number of zeros. So now you are getting your prefix sum for ones I did with number of zeros and so last step that remains is if your bit was positive. Then you should take the second value which is number of zeros plus the one prefix sum. Otherwise you should take the first value which is just prefix zeros, prefix sum of 0.

Professor Student conversation starts, so how can I found the indexes in the auto case available? How do I write it was once everybody nose what its position is in the output everybody has computed it ? You do a barrier and figure out everybody has done it. Now you know where your bit goes, you simply write to that position. Everybody bit goes to a different place. What is the record for this place?

Why to the same place to into the same array in a different slot? What if you are assuming that, we do not want generally considerate it everything is paired with, when you are actually trying to do this? You could do this sum in this way you do assuming it is within a block. You do think after you have finished all of your operations. Then write , read the element, no you have already read your element.

So that is what I am saying if you have not read, there is a big record. The rest of the records, so you read the record, then you sync, then you write it everybody has read it loudly. But then you are assuming in processes, no I am just assuming every locations and local memory locations. But everybody is doing that so only meeting temporary error. Yes for example if you say temporary date that work not be alright.

So my notion of implace was in place in global memory that is why I was referring you read it in the register. So assuming that there is a limit size record you read it in temporary space and you write it you are using alternative extra space. That is tough one, I would say, in sequential

context it will certainly would not be called in place in parallel context every time. You ready things in parallel, write things in parallel.

You have to have space to read in parallel reading and things and so you are not really making copy of anything simply in a P Ram state, what would you say? Read an element, do local computation, write it. So when you read an element, write an element in the same place is it in place or not when you are using N registers. Because you have N processes is it in place or not technically it is not.

But in the parallel context you would probably call it in place because you are not really making in the shade mainly and it additional second count which needs to be accessed by multiple. Professor Student conversation ends. So there was a quick example which I am going to rush to original number of bits.

Then there negations, then prefix some under Escan and the zeros and the second one is your index and the last column last row is effectively Escan on the number of ones. Let me stop questions on the basic one bit sorting. How does this work? Now when you have many such one bit, you can simply repeat it K times, if you K bits you may also

**(Refer Slide Time: 26:04)**

## Blocked Radix-sort

- Consider  $b$  bits at a time as "key"
  - Prefix-sum still required per-bit?
- Cuda [Satish, Harris & Garland 2008]:
  - Divide sequence into blocks, Divide key into nibbles
  - Load each block into shared memory
  - Sort by 4 iterations of single-bit stable sorts
  - For each block, write its 16-entry digit histogram and the sorted block to global memory.
  - Perform a prefix sum over the 16B histogram tables
  - Copy elements of each block to their correct output position

Block it instead of saying just 1 bit at a time. I can read K bits at a time or K one bits at a time. How will this phase change if I have K bits instead of 1 bits. Now I have got K bucket 2 to the K buckets. So you do prefix sum for each individual either you can do it in one row if you can copy multiple numbers or you just do it K times technically doing it in one go would be much faster.

Professor student conversation starts how will you do that? You still have the same binary tree that you construct for prefix sum. But each day time instead of saying take the sum of this, you are taking a sum of K things pair wise sum to two arrays rather than two elements. If I have taken T as number of digits zero to ninth. We can just compare the digits then we can divide would be the same as we compare the bits.

You would compare for example, this algorithm by Darlen and company which actually got published in two thousand nine to this they had an earlier version of two thousand eight also, which divide it into 4 bit, once at a time and then it really is taking the entire input in, I am asking out this four bits checking those four bits and then doing something like that. So is it better to for example, I am giving with the number I am doing radix sort of number.

So it is better to take just compare the time leg its meaning that your entire key, the entire legit as a key. How many buckets do you have 1 digit at a time? First four bits and then form, what do you think? Basically is that your question is it mod 16 or mod ten base, that is your question. So what I am saying that for example in the radix. So we started LSB and one thing is that was the digits.

So you can form 10 bucket and you say anything with 0 will go to for 1 to the second bucket. What I am saying is it better compare 0 and 1 if you come there, first four bits and then you merge 4 bits will tell you whether it is 0 or 9, is that what you are saying not much fully understand your question. But would you like to be base 16 is easier. We can get the 4 bits out next time you have to do some work get that digit out.

Nobody is store till it gets done except we, professor student conversation ends. So it does it block a time it brings in a block into a shared memory and in the shared memory it's simply

compares your number if 0 goes to 0 bucket, one and locally. You can do kind of local at least you in the context of local memory. You can do it one bit and at a time you can simply look at all the 4 bits together and make nine buckets.

Which would you rather do, if you do it in a multiple of sixteen is fine but one phase of this is 4 bits worth of comparison. One way to do the fourth bits of comparison is to do four stages of 1 bit sort like you did earlier or one stage of 4 bit sort where you say I will generate 16 different prefix sum anyway and that lots of parameter that going to, so I do not expect you to able to come up with that answer quickly.

But at least from the shared memory that doing it together is faster. So iteration is two thousand eight paper, where they first presented the iteration and then updated anyway. So now they generate this histograms which can be prefix sum and you figure out where digit numbers are involved and then you go back to global memory. Put it in the right place, you do it one block at a time.

But about the cross blocks, professor student conversation starts through this individual each block, we did this individual. If we wanted to do it for the entire block, what would have to change in this algorithm? You will have to know where this bucket begins? So there are sixteen buckets, for you to handle.

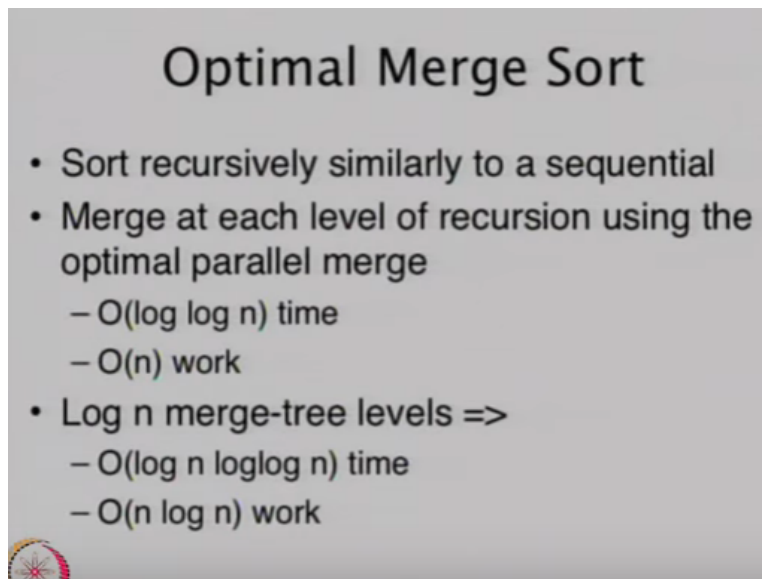
But there are other blocks also handling 16 buckets, the same 16 buckets you have to do a prefix sum across all of that which means they have to come up with their histograms before you can do that prefix sum or alternatively. You can simply generate your results at locally and you can say now I have got these 10 buckets. My 10 buckets are stored at this place, you are 10 buckets are stored at some other place, then later on you simply move this buckets around professor student conversation ends.

Enough about radix sort which by the way is conventional wisdom that for integer type sorting, which is where lot of sorting dollars are devoted radix sort works the best specially in the parallel

domain there has been lot of work which all points to the same. Let us get back to sorting optimal, remember merge sort.

We have already discussed it when we were talking about merges, when we started talking about sorting and we are going to extend it to do a fully optimal sorting which is dealing fast meaning it is  $\log N$  and be done in  $\log N$  time, the basic merge was

**(Refer Slide Time: 34:05)**



**Optimal Merge Sort**

- Sort recursively similarly to a sequential
- Merge at each level of recursion using the optimal parallel merge
  - $O(\log \log n)$  time
  - $O(n)$  work
- $\log n$  merge-tree levels =>
  - $O(\log n \log \log n)$  time
  - $O(n \log n)$  work

Not the basic merge the fast optimal merge was fast and Professor Student conversation starts  $\log N$  fast and  $N$  work Professor Student conversation ends. So if you just apply this algorithm you get optimal sorting  $N \log N$  work, because in  $\log N$  stages of merging and how much time  $\log N$  square time which is much better than  $\log$  squared  $N$  which is being and if common in several of the sorting.

The question is can you do better, the second question and this is where you will also see some examples or some cases where we instead of thinking just in terms of here is time. and here is the work complexity. We will actually try to come up with an algorithm where if we know that there is  $P$  processor. So we will do that scheduling rather, than having the basic  $P$  ram scheduling. I talked about earlier, so let us think about that in the merge case.

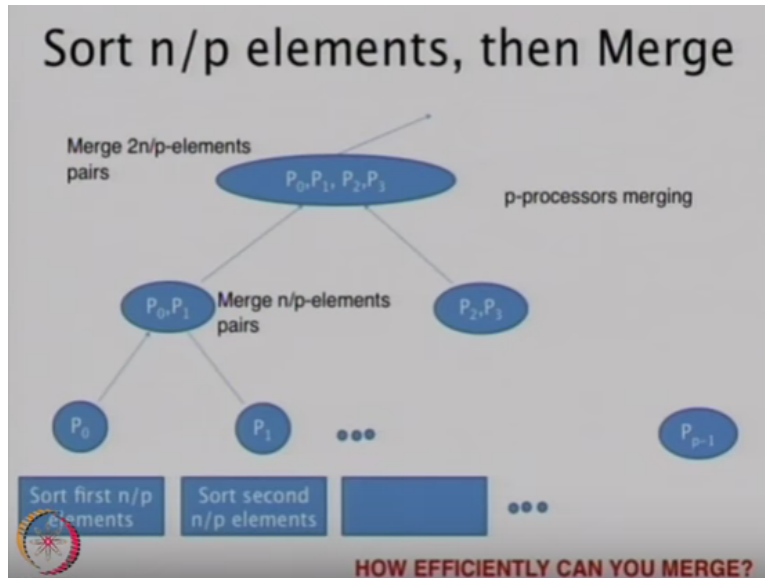
We have  $P$  element,  $P$  processor only and elements to merge and with the knowledge of optimal algorithm and all that, what would be a good way to proceed that? We might say we will take  $\frac{1}{P}$  th of total number of elements. We will do local sorting, hopefully local sorting is significant and faster which is we know how to do it than shared sorting. So we will do local sorting  $\frac{N}{P}$  over  $P$  elements.

Then we will merge them, one pair at a time how will this work? You have  $\frac{N}{P}$  elements after lead you merge to and you merge to  $p$  by two processes needed for that just that the leaf level. So maybe you can say I have with  $2 \frac{N}{P}$  elements,  $2 \frac{N}{P}$  elements. So we will generate  $2 \frac{N}{P}$  leaves and so will  $P$  elements no  $2 \frac{N}{P}$  leaves regenerate  $2 \frac{N}{P}$  leaves. So that with  $P$  processes, we can merge then in parallel order  $\frac{N}{P}$  each merging 2 at a time optimal  $\log_2 \frac{N}{P}$  over  $P$  and order  $\frac{N}{P}$  work.

But that only gives me the previous stage, one line what about the next line? Now the merge side is the element number of elements in each group has become twice the number of groups has gone down by so I would at least now like that two processes share the merging of one. So two processes merge the first two pair processes, merge second pair, two processes merge last pair the  $P$  by two pair.

Next time four processes merge and at the end  $P$  processor for the entire merge and so you added up together. How much time will it take you can on the leaf level you can say that is sequential merge. But beyond that is parallel merge. Professor student conversation start, but be limited parallel merge. So we are merging certain size blocks with certain number of processes will always two at a time.

**(Refer Slide Time: 38:44)**



How long will it take these steps time, total time number of steps? We started with one processor every month. Finally all processor every month on the entire thing. But how long does it take for all the processes working on the entire thing and then similarly, half the process is working on. I will let you work this out it is variant of this is already in one of the earlier slides.

So it should not be too much trouble to refer to it and this is also very related to the one that I said workout earlier, today the bottom line is that is not going to be order  $N \log N$ . But you think all the merges in order one. So you will multiply to figure out it is just not multiplication it is adding up summing up number of steps per log. Professor student conversation ends. The now from this two way merge, let us move on to  $N$  way merge.

**(Refer Slide Time: 40:06)**



## Optimal Multi-way Merge $N=P^2$

1. Divide L1 and L2, respectively, into P sublists each
  - ith sublist contains elements at positions  $i, P+i, 2P+i \dots N-P+i$
2.  $P_i$  merges the ith sublist from L1 and the ith sublist from L2
  - Result go back to the positions originally occupied by the two sublists
- ◆ Each element is at most  $P+1$  off from its final position
3. Divide each list in blocks of P
4.  $P_i$  merges pairs of blocks
  - block  $2i$  and  $2i+1$  (Results are put in-place.)
5.  $P_i$  merges blocks  $2i+1$  and  $2i+2$ 
  - Results in-place

Let us just simply take those end things at the bottom and merge them using P processor. This T thing at the bottom merges them using P processes and the idea is I hope I did not have all the slides, further ordinal merge sort. But I hope you have looked at the slides, I did have it when I presented it in class. So it was posted, Professor Student conversation starts it is in the slides does not matter, if you have not seen it you will not suffer.

Because this uses a construction, that is very similar to batchers merge. But it can even, if you have not seen it think it later as just a fresh algorithm. But if you have seen the connection then you should be able, if you seen that algorithm then you should be able to make that connection. We are going to begin with an assumption, we will say L is equal to P square. So if we study batchers, we had a optimal batcher was or  $\text{Log } N$ .

All these study called Log, because batchers extension is going to get better then optimal which is what this one is and then after this we will also go back to work time complexity mechanism and in fact even that mechanism. You can come up with an  $N \text{ Log } N$  work and  $\text{Log } N$  time algorithm. This one is relatively simple, next one is a bit involved and sometimes non intricate. Professor student conversation ends.

So we will start with assumption, that we have the number of elements which is square of the number of processes. We have and the algorithm is listed one at a time, it divide we are merging

2 sorted list. This is merging algorithm, so far we have  $L_1$  and  $L_2$  and we are going to merge  $L_1$  and  $L_2$  using  $P$  processor.  $L_1$  is sorted  $L_2$  is sorted. So what it does is it takes every  $P$  element in  $L_1$  and every  $P$  element in  $L_2$  and generate a list.

How many such list will it generate zeroth  $P - 1$  and so on will be one list from  $L_1$ . Similarly list matching list can be generated for  $L_2$  in the next list, will be  $1, 2, P$  and the matching list from  $L_2$  the third list will be  $2, P + 1, 2, P + 3, P + 1$  and so on. So we are generating how many different list  $i$ th lists processor  $P$  simply takes one of these pairs  $i$ th pair and merges it sequentially local.

You will see that is the focus here trying to do more and more things local and it puts it back into the place. So it is not thinking it is again in place. So every  $P$  by  $i$ th processor  $0$ , so my elements are at  $0, P - 1, 2, P - 1$  and so on. I think of that as an array just my  $Strike$  has become bigger. So I pick up two apparently two array and I teach us individually sorted and my output is going to be the left half of the combined sorted list, merge sorted list, going in the left array and the right half going right array.

So in place put back the sorted list the surprising thing is that elements are not too far away from where they should be, actually it is positions from its actual the rank value. So how do you bring it back to its final position. The rest of it is not that out of the book once you believe and will look at the reasoning for this part once you believe that things are within  $P$  of each other then bringing it into its place should not be too hard.

Professor student conversation starts, you sort what locally that I do not want to sort end over  $T$ , well you would at the end of the day. You have to do that because  $L_1$  will be the sorted left half,  $L_2$  the sorted right half. What I do not want to do is take this entire list and sort, because sorting that is sorting of half the size. I begin with that will kill my efficiency. So it is actually not that hard.

You divide still think of it as blocks of  $T$ . Then you do the odd even merge of batchers. Professor student conversation ends. You take  $\log_0$  and  $1$  merge it and processor does it another processor

is in the meantime merging 2 and 3 the third 1 is merging 4 and 5 and so on. We are not done yet the next step you do you let alone 0 and you take 1 and 2 merge it 3 and 4 merge it 5 and 6 and merge it and at the end you are going to get sorted.

Why is that after first alteration? Why is the left most block which is not participating in the second one in step number 5? Why is this sorted every element that is started in that block was at most P away from this position. So where all might it have gone at most into the second block. So one of them have sorted those, two merge those two list and put it in the right place within that much.

But we know that it only needs to be within that much it is does not need go beyond that and same way element that is within P of its position this Way or that either you sort with the left block or right block it is going to get in that position if you do both then you guarantee then it back to its position. alright why are things only?

**(Refer Slide Time: 48:25)**

### Multi-way Merge: Proof A

- Let  $L_1$ , and  $L_2$ , be the sublists assigned to  $P_i$  in step 1
- For  $X$ ,  $n$ th element in  $L_2$ , there are three cases
  - $X$  lies between the  $m$ th element of  $L_1$ ,  $A$ , and  $(m+1)$ st,  $B$
  - $B$  is the first element in  $L_1$ , and  $X < B$
  - $A$  is the last element in  $L_1$ , and  $A < X$
- Case 1:
  - At least  $[mP+i+1]+[nP+i] = [(m+n+1)P+i]+(i-P+1)$  elements  $< X$ 
    - $mP+i+1$  elements are from  $L_1$  ( $< A$ ) and  $nP+i$  are from  $L_2$  ( $< X$ )
  - At most  $[(m+1)P+i] + [nP+i] = [(m+n+1)P+i] + i$  elements  $< X$ 
    - $(m+1)P+i$  elements are from  $L_1$  ( $< B$ ) and  $nP+i$  are from  $L_2$  ( $< X$ )
- Rank of  $X$  in all elements is between
  - $[(m+n+1)P+i] + i - P$  and  $[(m+n+1)P+i] + i$
- Position of  $X$  after merger is  $(m+n+1)P+i$
- Other cases are similar

$i+nP$

Professor Student conversation starts, blocks would be unsorted. Block 1, block 2 will not be sorted? Good question actually, they are sorted. We have not proved that yet, but at the beginning we have two sorted list. We did have two sorted list, no it is not immediately we are going to get sorted result. Although you will have taken every Pth. So P different things have are involved in sorting this block.

The first block  $P$  different process, first processor  $I$  was only looking at this element, second  $I$  was looking this element, third  $I$  was only looking this element. This was sorted when we begin, but then somebody bringing in other things here, will it still remain sorted processor number six, for was taking every  $P$ th element starting at 6 and it is going to bring something which might bring something from here, something else maybe smaller.

Instead of four then things like that you cannot say that based on four elements. But that may help you figure out, why they are sorted? When you take an example, you understand the structure of it is a little. You can draw connection by element that we bring in it is smaller than it can be it will be smaller than it would have to be smaller than that. It was already there that we may not be necessary it may be too small.

That is why you are saying that  $P$  list having elements at most  $P$  away from it is also done, that it is not it can be. Let us look at the proofs it is not final position, because it is not necessarily as big as it needs not be greater than every number before it right. So in fact I have to define what I mean by these things, are sorted.

These things are sorted ordered in a certain way, there is a diagram values which you cannot read or write. Let us see if we can get this thing set up well take a break and in fact this picture is coming out to be too short and it is not quite readable. A new block which we are getting overall going to be larger than, yes we are stopping now.