

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

Module No # 06
Lecture No # 29
Algorithms, Merging & Sorting (Contd.)

So we looked at by autonomic difference where the number of processes and it is also called sorting network for the reason where that it is really targeted for running n things through the network of n things through several stages. Right, N things in each stage and n things were simple comparisons in exchange. It takes to inputs and bigger one goes up and the smaller one goes below.

Or vice versa depending upon the lower sides or the upper side. What would you do if you did not have that network right, meaning that I give you overtunic sorting to implement I give you n things to sort and you have only P process. What would you change? First of all how long does the algorithm take? The, for N things with n element and n processes. Long Square N right that is the time and n long square is, sorry and the work is n long square done.

What will happen if you have p processes but n elements? What would you do, what is it that would be different? Order n in what, in each stage? Total work is n long square. Something similar will still apply. Right, but sometimes when you look at the exact way in which you. So there is a default algorithm way which we talked about earlier. And that can be done sometimes that takes longer than it would take with this is not the example of that by the way.

It takes if you carefully structure the computation you will see that it still turns out to be more efficient than the group force mapping of K work to P processors each step. Okay, so here in this case I will leave it at then you can in fact do the same thing you can map the work of P things rather n things among p processes but just I cannot resist going one step deeper what would each processor be doing?

So let us take one step and similarly we can do others also. We have one bionic sequence and or one half of the bionic sequence and another half of the ionic sequence and tis is longer than P or two P and you have to compare. You would have each processes compare some N of the

P of them. Which N over, what does it matter? In the original stimulation that we discussed earlier it does not matter.

It simply do P things P step or run by P things over P stuffs. And also steps are the same but on real machines it would not be the same. That it would be, but which N by 2P, we can take every alternate, and we can take blocks. So let us talk about curt. Since we are going there would you want to do it in a global manner or share? What is the step for the given processor?

Read while I give from there read while I give from n by two apart less, compare the two and the bigger one goes here and the higher one goes there. And you just stopping those two things. How would it work if you brought it in shared memory? Being this in shared memory, bring that in share memory so, lock it in share memory and write it back. Do you say anything, not really, you just did an extra transfer through shared memory.

In this case it is an ideal case so you can directly do the single global memory. So in this case bangs are not visible right, shared memory banks are visible but global memory banks are not.

(Refer Slide Time: 07:00)

Split Bitonic Sequence

Subsequence 2

Subsequence 1

- Say $\{a_0 \leq \dots \leq a_{n/2} \geq a_{n/2+1} \geq \dots \geq a_{n-1}\}$
 - Subsequence 1: $\{\min(a_0, a_{n/2+1}), \min(a_1, a_{n/2+2}), \dots\}$
 - Subsequence 2: $\{\max(a_0, a_{n/2+1}), \max(a_1, a_{n/2+2}), \dots\}$
- Recursively sort each bitonic subsequence
 - Subsequence 1 \leq Subsequence 2

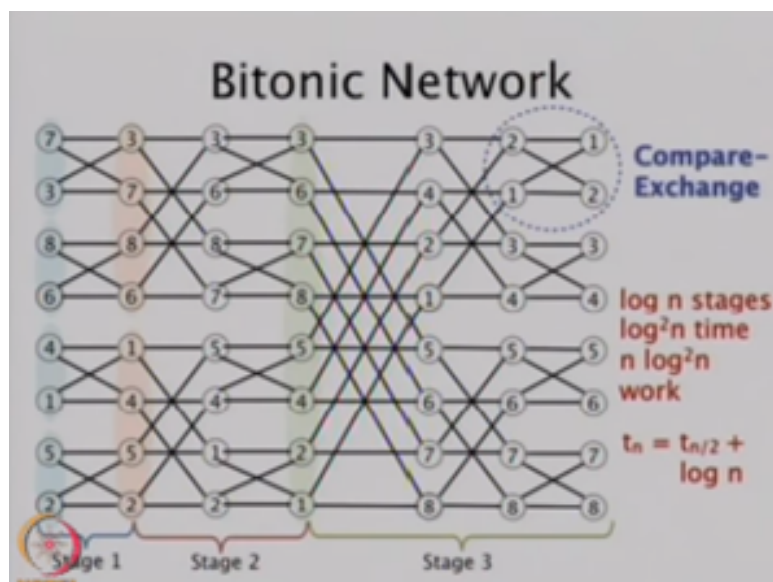
This should tell you right. Or you want the picture slide. So that is picture slide. Second sequence is told at a distance memory from first sequence the firewall sequence are different at start interface locations. So when you read two things you will be reading from different

locations but many threads maybe reading multiple things and you want those things to be read in block right.

Which suggests that all the different threads reading suppose your block size is K, then K threads reading K different things alternatively meaning sliding. First, thread reads the first things, second thread the second, third thread the thread and so on would make sense. If instead you said the first thread makes one block right, element number zero to element number twentieth or I do not know what it is.

Then when you read the element number 0, the second thread s reading element number 29. Right, so doing this in blocks might seem like a good idea but it is not always. So if you look carefully as how to partition then you get vector sweepstakes.

(Refer Slide Time: 08:57)



Okay, unless you get back to more sorting bionic is not that efficient right, at least work wise and log squared is not that great, we already know of the better sorting technique. What is that? Optimal mode sort, right, where we were able to merge two list in log in time with n y. if you use that to merge sort, I think we will come that to later on in the slide.

(Refer Slide Time: 09:43)

Batcher's Odd-Even Merge

- Merge the even indices
 - Get $c_0, c_1, c_2, c_3, \dots$
- Merge the odd indices
 - Get $d_0, d_1, d_2, d_3, \dots$
- Perform an odd-even merge
 - $c_0, \min(c_1, d_0), \max(c_1, d_0), \min(c_2, d_1), \max(c_2, d_1), \dots, d_{n-1}$

We would get better than $N \log$ and N right, $\log N$ stages each stage taking $\log n$ time. What would be the total time spent? $\log N \log N$. and the work. It will be better than that. No the time. Time is $\log N \log N$. what is work? How long did it take you to merge work wise? Merge two sequences of size order of each. $N \log N$ right. That is the amount of work you will be doing at each level of \log in merge hierarchy.

And there are how many levels. $\log N$ levels, right, so there is $n \log n \log n$. so it is not optimal but it is better than this $\log N$. what is in $\log N \log$ right. We will get to also something that takes $\log n$ time and $N \log N$ work in parallel okay. It is slightly complicated algorithm but we will talk about it.

(Refer Slide Time: 11:20)

Batcher's Odd-Even Merge

- Merge the even indices
 - Get $c_0, c_1, c_2, c_3, \dots$
- Merge the odd indices
 - Get $d_0, d_1, d_2, d_3, \dots$
- Perform an odd-even merge
 - $c_0, \min(c_1, d_0), \max(c_1, d_0), \min(c_2, d_1), \max(c_2, d_1), \dots, d_{n-1}$

But looking at some of the simpler method, sometimes we run faster. Because they are easier to implement and at least for relatively average number of elements they work by craft. One of them is merges, odd even merges. Which is question mark. Bucketing typically in practice all report say that bucketing is faster than any other sorting technique. Complex y is the question of how long does it take you to compare two things.

And the truth is that whenever you talk about things you talk about integer keys or something of that sort. In this case comparing two things takes fast in time. But in general once you bring and we never say $N \log N$ comparisons right. If you start to really grow big then comparisons do not take order one time and then it starts to $n \log n$ times the comparison time and then you have to compare that against.

So once you have talked about really large numbers. Compare that against bucket sorting. Right, how well bubble sorting work in parallel? What do you do in bubble sorting? Keep checking tears and something at the top might reach the bottom then you start again and you start again and you do that several times and people are at divide designated places. If you do that in a parallel setting does not quite work right.

It is kind of inherently sequential you compare A and B, then you compare b and c then you compare C and D and so on. It is out that there is a parallel version of it. And it does not have to be in sequential order like that you take tears and want them to this compare and exchange that we have just talked about. So you take the first element of the zeros element and the first element. You compare them and sort them with necessary.

Then you take in parallel second and the third, fourth and fifth, 6 and 7. So each pair and this is called either depending on where you are starting the index even phase or odd phase. And then the other phase, the next phase would be the hard phase or the even phase which is instead of taking 0, 2 you take 1, 2. So it leaves there all alone. You figure out between one and two who is shorter who is less who goes first the more thing goes later.

Then 3, 4, 5, 6, 7, 8 and so on. And so these odd and even phases make up one stage. You do this end times your list will be sorted. I am not going to finally prove it here but you can intimately probably see it. How long will this take, how many time per stage? Positive times.

I said do it ten times, order n stages so total time is order n and work finished stage was order N.

And so total work is n squared. can it be better it is not that bad work wise right, its end stages it stage did not arrive at work and you end up it is not optimal interference of log N. but considering that we are starring at bubble spot that is what we would expect right n square. It can you think of the time being better than this, can it be better than order N? Okay, I will leave that as an exercise you can do it and we will talk about that in next class.

(Refer Slide Time: 16:44)

Parallel Odd-Even Transposition

- Consider a block of n/p elements per processor
- The first step is a local sort.
- In each subsequent step, the compare exchange operation is replaced by the compare split operation.
- $$T_p = \Theta\left(\frac{n}{p} \log \frac{n}{p}\right) + \overbrace{\Theta(n)}^{\text{comparisons}} + \overbrace{\Theta(n)}^{\text{communication}}$$
- The parallel run time of the formulation is cost-optimal for $p = O(\log n)$.

Which means I should not be working about these slides. Again you will get the same question as before. If you had n processors rather P processors and elements what would you do and how would you do to put things together. And one thing I forgot to mention when you were talking about the same thing in context of bionic sort, it things that typically things that you would do locally any two processors are not sharing data.

Whenever you do one compare exchange you are really talking about two processes data being compared and exchanged. When you have so it backing up packing away from this just for a minute if you had to sort end things, and I knew that locally I can sort things fast. Comparatively in the context of having not having to send data across to another processor and interacting with another processor and all those things.

How would you structure your algorithm? If you did quick sort then what would you have to do? There are I am running ahead of myself because there are some slides later noticed.

(Refer Slide Time: 18:39)

Parallel Odd-Even Transposition

- Consider a block of n/p elements per processor
- The first step is a local sort.
- In each subsequent step, the compare exchange operation is replaced by the compare split operation.
- $$T_p = \Theta\left(\frac{n}{p} \log \frac{n}{p}\right) + \overbrace{\Theta(n)}^{\text{comparisons}} + \overbrace{\Theta(n)}^{\text{communication}}$$
- The parallel run time of the formulation is cost-optimal for $p = O(\log n)$.

Since we are talking about it, I will discuss here, so in quick sort what will you have to do? Right, and that is an excellent algorithm. Only after you have separated. Problem is creating them takes 2 hour time typically again there are variance where you do not separate insist on separating into 3 equal parts. If you want to do it in three equal parts then you will going to spend more than what you will make up for later.

But if you are willing to say maybe if I am between P and $2P$ or P and $5P$ something like that, then you can separate them out reasoning well. And after that it is great everything is perfect. That is one approach. No we assume that the data, so again we have to talk about in terms of whether it is a PRAM or communication order. You are thinking in terms of communication order.

No sometimes not very precise but it means talking to other processes, sharing data right. Even in P there model there is a notion of local memory that everybody has and shared memory where you have to write if you want it to be visible to others. So when I am talking about sending data I am talking about writing it into shared memory or physically sending it through some communication network.

So in both cases doing it locally is much faster, right. Think of MIPS you would rather do it on given machine than figure out which part of data is being handles by the other person, which part is being handles Y me and then compare some things from my data and some

things from their data. Does not matter for that I have to seek the other persons data that in a communication network that is a typical approach, right.

Well I have some data the other person has some data, I take the other persons data I give my data and then I somehow for example do the comparisons and keep the lower once that is also the comparisons keep the higher ones. Okay, it is repetition of work here. Whoever making the same comparisons alternatively you could make the comparisons and send the lower ones or the higher ones to the other person.

That has, why is one better than the other, or what is the comparison that is leading? Relative right, you would rather in the beginning exchange the data and have not wait for other person to compute the result and then give you the answer. You would rather in the men time compute it. Again that is not always the one way stream it again depends upon the context.

But generally if you would not be doing anything else you might as well use the time to redo that piece of computation and save the communication. As we talked about it before also. Okay, so that is one what are the other ways in which you can make things local, Merge you can also work like that right, I take b sequences sort each of them locally on the p processes.

And then I have got p sorted things and have to merge it. Okay, so in this case, this analysis is based on the fact that I have taken this data locally, I have p sorted things and I want to do compare change among this sorted list.

(Refer Slide Time: 23:29)

Example of a Fast Sort

- Rank sort
- Given Array A,
 - For each i , find rank $A[i]$
 - $A[\text{rank}[i]] = A[i]$
- How fast can you find $\text{Rank}(A:A)$?
 - If you had n^2 processors

Coming to this business of fast sorted one ultimately the business of fast sort is to find your rank, right. I want to know given this list of unsorted things I pick out item number six and say only things are less than me and I know my position mark, I just put my position over there. How fast can we find the ranks? For example clearance we have red processes, done things of this sort before, do not look up your notes.

Whatever it was, I do not want suppose I give you N Q processes. How fast can you sort, not sort find grains for everything? So remember when we were merging we were actually finding ranks is not it. Sorry, but they were all finding the ranks when sorting. Sorry okay, that is correct I was again maybe getting ahead of myself when we were merging things we were finding ranks in sorted things but okay.

This we have not done before I was thinking that we have already discussed this before. If it is unsorted, I have got an unsorted list and I want to find the rank of the given element. This probably needs a little bit of sort. Let us assume unique, that sometimes makes thinking about thing easier. And then we can extend to non-unique things also. Hang on, okay so for every so we are finding the element the rank of one element right now okay.

Maybe others maybe done similarly one elements rank you compare it with everybody else. And, you figure out how many elements are compared to 0. So less is 0 positive means one, it is actually tuff leaving around but does not matter how do you count the number of zeros? Log N is one way but that takes how many how much work, does it need N square? With N process you can do this in log N time.

With N work you can do this log N work. Can you do better? Update the count, atomically, that does not really even belong in P rank atomic right kind of thing so everything is synchronize. Sorry, it is some, prefix some, we probably need to stop. We will continue this but there is something that I needed. What did you just say? You, atomic right if you think about it, an atomic update is a sequential operation right.

If N people do atomic updates you are doing octagon operations. So, that does not belong here, sometimes we do that because we need some consistency and all that but do not think atonic as your first choice of things to do. Okay, let us top suggesting and think about what

prefix or for that matter word processes are. I will give you n^2 processes, can you do something with it? With N processes and $\log N$ time, right.

Then you have not used the N^2 processes right. So you mean to find the rank of everything. no I mean find the rank of, this question is asking how fast you find the rank of a array the entire all the list of elements. So yes, with order end things it can done in N time. And that would be $n^2 \log n$ time and $n \log n$ sorry $n^2 \log n$ work and $\log n$ time, right.

So time is good but $n^2 \log n$ is not acceptable as work. But back to the question of if I had n^2 processors could you do better? For one element and I want to know the rank of that element among all the others. But \sqrt{n} using \sqrt{n} things right. Because you then you are using n total number of processors. How exactly? You have taken \sqrt{n} and time, number of zeros using a sequential processes right.

If I give you an array with zeros and ones and I give you one CPU, and I want you to find how many zeros there are, can you do it in $\log n$ time? Can be done right, and things you have to look at n things. Without looking at n things you could not tell how many zeros there are. Okay, I will come back to this. Order 1 only, you are on the right track but how, maximum? Can n processes determine if my rank is 3?

Similarly some other n processor determining my rank is 4. Okay, I do not want to get struck on this problem for too long.

(Refer Slide Time: 33:15)

Parallel Quick-Sort

- Group of p processors sort a sub-sequence
- Initially all processors sort the entire sequence
- The sequence is divided into n/p blocks
- Together processors choose a pivot
- Processors rearrange elements into two 'halves'
 - Prefix sum
- The groups is subdivided into two and assigned to each 'half'
 - The size of each subgroup may be proportional to the size of the 'half'
 - If subgroup size = 1, stop subdividing and sort serially

Alright so parallel quick sort we have talked about this, the main issue is in how do you divide an element? Or list into some E sub sequences, in the traditional quick sort we find a medium right. When we do not spend the time finding the medium what do we do, we guess randomly think can you do something of that sort? If you randomly pick P , things what is the probability that the resulting partitions are relatively uniform that is not very high.

I am not going through the map for two it is yes if you pick among n things an element in a probability there that you have given distribution even you will say, fixed ratio is quite high. Okay, that cannot be said if you have to pick p things in all the p right, it is going to get multiplied. Probability this as well as this, as well as that lies within some range. So there are you do.

If you are going to use that process where you are going to say that process where you say I am going to choose some number of elements and say that they are relatively low balance and eventually let the processors process them. Then you have to be careful that how to sampler okay. The alternative is you proceed in the standard binary fashion right. I divided into two parts and let the two parts be handled by half the processes.

Okay, so now if I divided into two parts two things are happened. One, I have with good authority that the two sides are relatively balanced. And whatever in balance there is going to be written across many processes. In fact you can even say that if so suppose if we say that we allow one side one half to be utmost twice the other half then you sub dive that into two sub problems and you have figured out that it is the worst case one half is only by three.

And the other half is two by three then you take three by three processes and sign it to half and sign it to P, by three processes and sign it to the other half. You can do some of that also in the case of when you are breaking it down into local so you take some samples and you say now this is sorry, I should say you cannot do that again into P elements, P sub lists because you have said okay.

If my sample allows them to be within N over P let us say N over P times .75 and N over P times 1.25. Or 1 over .75 whatever that is then you want to make sure that longer list make multiple processes. But you have just said that each list is going to get one processor which is not going to work totally locally. Okay, so there is a balance to be had.

Some point when you starting to get smaller and number of processors might make sense to go ahead and sub divide it into three processors and let each handle it and in a kind of purely basis but when you are large number of processors for example hypothetically you want to sub divide it into smaller chunks and then sub divide your set. Into working on the left half or the right half okay.

Yes you do, you can right, and again one of your friends will tell you medium finding is still hard but you can do it faster than you will be able to do it on a sequential process or a single processor. Okay, so back to the business of trying to separate this, I have somehow chosen medium, guess the median now I need to make sure that this list is going to get divided into two halves then I say processor 0 to N by 2 or whatever the number.

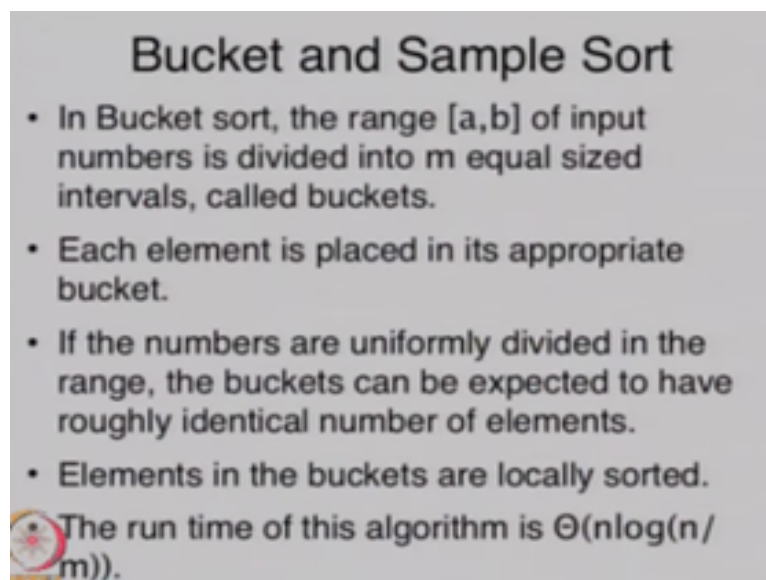
I have decided is go work on the left half and N by 2 to work on the right half. What is the algorithm? That is the I just want everybody to be on the same point, how do you divide it. You can take every element and ask the question are you to the left or to the right of the median, right. Now do this is in parallel. Some will be on the left, some will be on the right, where should they be stored.

Okay, so you simply write one or zero in your respective slot and have to find how many zeros there are to the left of P right. Meaning how many evaluated to the left side before you, which means they should this is kind of stable also you are going to keep. The things to the left of few and want to go in the left half to remain in the left of Q half after the partitioning.

And so you do the prefix sum, you count the how many people evaluated to less than the median on your left by looking at your prefix number and that is your position. Okay, now what was your question? Yes, that is why I said almost equal is not that critical when you are talking about parallel processes. But whenever you talk in terms of real processors, real machines be it code on machine it is not as true as.

No wait it is also reasonably true of Intel machines it was defiantly true of the old parallel machines n cubes mash powers and connection machine it is the kinds we have talked about earlier. Processors are typically some power of two okay, so if you keep the division in some power of two then integral division is somewhat easier, that does not necessarily mean that equal division is required but if they have this suppose it was half or one four three four then you probably get a much better balance than one third two third something of that sort.

(Refer Slide Time: 42:38)



Bucket and Sample Sort

- In Bucket sort, the range $[a, b]$ of input numbers is divided into m equal sized intervals, called buckets.
- Each element is placed in its appropriate bucket.
- If the numbers are uniformly divided in the range, the buckets can be expected to have roughly identical number of elements.
- Elements in the buckets are locally sorted.

The run time of this algorithm is $\Theta(n \log(n/m))$.

Okay, bucket sorting, again you can think of and you know you are going for racket sorting with this but you can think of bucket sort by itself as a way to break things into groups right. Instead of saying I am going to take a sample, take a median so to speak, and or K proxies K flavor and grade the K sections based on those favors. I can say I do not care about favors I am going to choose k random things which gives me k buckets p buckets.

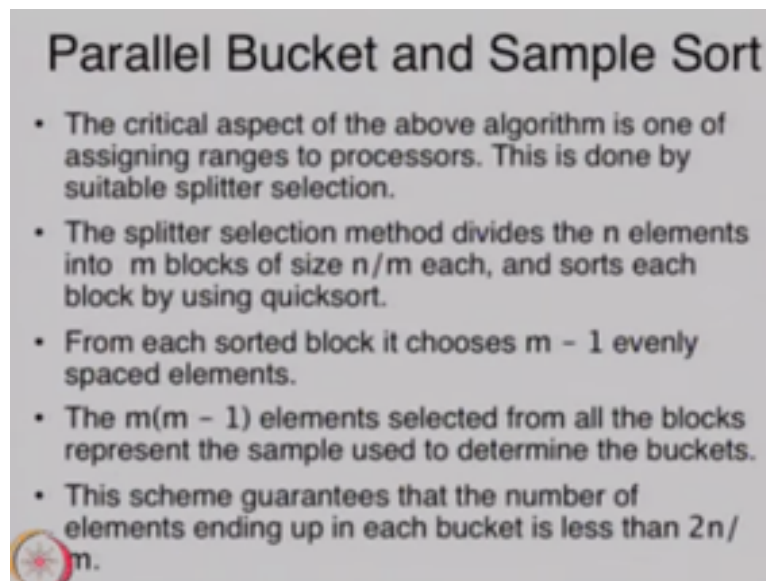
P is the number of process and then I am going to do local sort. Within the p buckets. If you ask me in the context of ready sort then one way to meet this buckets maybe by range right, I assume that my list is uniformly populated by small integer, big integers, medium integers all

type of integers and then I say if you lie between 0 to 1000 then you go to bucket number 0 1000 to something else you go to number one and so on and so to.

And you can see that it is leading to fix number of bits now per pocket right because you can take all the common you are basically making every number within a bucket have the same prefix. Of course in a bank representation you would not say 0 to thousand you would say 0 to 256 or 5, 7 or 5 sort and if it turns out that the numbers are really well distributed then you can stop here and do local sorting.

Of that local sorting is rated sorting even better and if we are assuming that these is an integer and we are dividing it based on the integer or some hash of the key which is an integer which we were using to separate them then in the sort, rated sorting in this side the local groups would make a lot of sorts.

(Refer Slide Time: 45:16)



Parallel Bucket and Sample Sort

- The critical aspect of the above algorithm is one of assigning ranges to processors. This is done by suitable splitter selection.
- The splitter selection method divides the n elements into m blocks of size n/m each, and sorts each block by using quicksort.
- From each sorted block it chooses $m - 1$ evenly spaced elements.
- The $m(m - 1)$ elements selected from all the blocks represent the sample used to determine the buckets.
- This scheme guarantees that the number of elements ending up in each bucket is less than $2n/m$.

Okay, okay, so we talked about parallel bucket sorting so I would not repeat it here and talk about resolve heading, if you want to choose samples either using the bit straight representation of your key or median style less than equal to comparative styles. How do you choose these glitters is going to be one important question? So far what we have done is we have divided it into buckets and locals and done local sorting on each other whatever the method on local sorting that you like maybe.

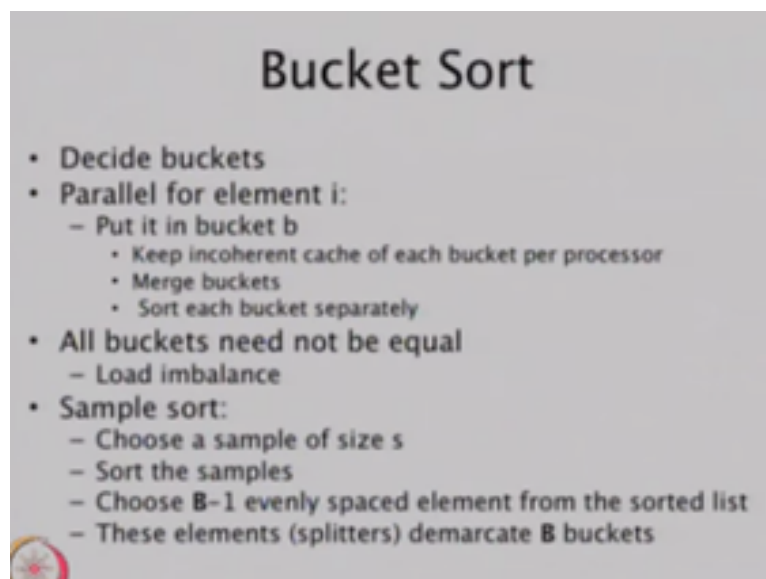
No this does not this is not anything to do with directly with rated sort, you diving into bucket sort into buckets you are sorting it into buckets and then you may do rated sorting inside

those buckets. So the first step was not really the one step of rated sorting. It was just dividing into things that are known to be in order with respect to each other. Inside each, you could think of it that way but do not.

It is not really about rated sorting it is about using the key to, or some splitter either it is some dreads of the key or value of the key to decide which bucket it goes on and the buckets are ordered right, one bucket has only elements less than the next bucket and so, that is all. They would be right, as soon as you start to say that I will have something like P buckets you will not yet to be approximately clicker.

Okay, when we start to have fewer than P buckets then it is okay because more than one processor can be assigned to a bucket.

(Refer Slide Time: 48:14)

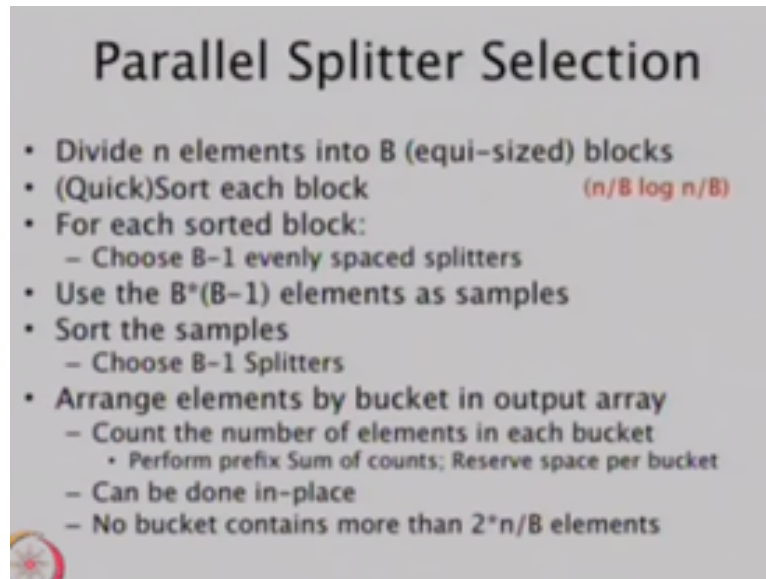


One way to do is the thing called sample score at the bottom. It is just so it is a new name for bucket sorting to speak it is also called sample sort, where we take some random number of sample okay, we have to determine what a good size will be you sort the samples and then we take every so samples, take out one as a pivot okay, not going to again go through the analysis of this is a reasonably well used algorithm.

And has scored probabilistic bounce as well. And you can get reasonably bounded probability I do not recall the exact formula that it derives its weird function of logs. But if you used $\log n$ sized sample set and from that use b buckets, select b buckets from that right, you guaranteed to get probability which is practically constant of getting good balance. Okay, the

other way to sampling is if you want b buckets choose b squared samples okay, you again I am not going to go through this analysis all of this.

(Refer Slide Time: 50:22)



Parallel Splitter Selection

- Divide n elements into B (equi-sized) blocks
- (Quick)Sort each block $(n/B \log n/B)$
- 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
 - Can be done in-place
 - No bucket contains more than $2*n/B$ elements

Because there is much grounds to cover in the limited time we have we take three squared samples sort them of them three square samples we have picked every B element which is going to be the pivot okay, intuitively speaking why should every b element found reasonably good? But again somewhat quantitatively if you assume any uniform distribution then everything works.

This is all in the presence of none of this analysis starts in the assumption of uniform distribution. Okay, because it is just a probability of a good pivot. Something of this sort because we have said that we have taken B elements that something of enough elements and then sorted and picked every so of them which means there are which we defiantly know that there are enough elements that we are not picking the two pivots.

Right, yeah so we in this case so that is what we are doing in the $\log n$ size sample set also. In this case we are exactly b , and the probability that those d are uniformly distributed again without any assumption on the input is going to be rather high, maybe what I am going to do is leave this analysis for you, this will be on the website and you are welcome to ask me questions on this might be this is the wrong place.

(Refer Slide Time: 52:59)

Radix Sort

- Bucket-sort by each key
 - 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
 - $\text{Notbit} = \text{!bit}$
 - $\text{psum} = \text{escan}(\text{bit})$
 - $\text{nZeros} = \text{psum}[n] + \text{notbit}[n]$
 - $\text{nBefore} = \text{idx} - \text{psum} + \text{nZeros}$
 - $\text{Rank} = \text{bit} ? \text{nBefore} : \text{psum}$

Let us get to the radix sort variant the standard radix sorting works this way right, we basically separate it out to parts and then find out where bucket sort basically in sequence in part of the P. so in order to bucket sort what you have to do? Again it is basically to find the rank of the P, and this is something that you can go back and kind of plugin in also the quick sort version where we are sensually separating into 2 chunks of 3 we are equal number of equal number is allowed so how would you do radix sorting.

Forget about slides which has you might guess I am going to leave for you to read also but just at high level what would radix sorting algorithm do in parallel? I cannot let someone go on to the next key right, it has to be done in the sequence so everybody should be focusing on this part of the key this key sub key if you will. Maybe this is not repetitive enough that I should go through it do not look at the slide, think about it.

How do you get the local ramp of b, only among p because everybody has to do prefix sum right again stepping back one step out, what are we trying to do there are some different key values in the sub part so for let us take an example we are sorting integers okay, so we have 1 digit as one part okay, so this digit can have ranges zero from nine and I would have all the zeros to go up then ones then twos then threes and nines okay.

Now how should it happen? Something like prefix sum will do it right but how? Ten bit array, so it is a ten bit array of ten bit numbers is that what you are, so at a given column of digits I want to figure out how many zeros there are and at the same time how many ones there are in

the worst case I can do it nine times, ten times. I can first say how many zeros are there then I say how many ones are there.

Then I will say how many twos are there okay, this can be done standard prefix sum right, in fact I not only know $m =$ how many zeros are there I know which 0 am I. can I do it all together that is the question, okay. Well I do not take ten prefix sums or if the key takes alphabet size is K then K prefix sums. I think we have to stop here.