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

Module No # 07
Lecture No # 34
Lock Free Synchronization, Graph Algorithms

Okay so we were discussing how to steal from a busy processors without too much for the busy processor okay.

(Refer Slide Time: 00:43)

Idle p_i Edits Busy p_b's Queue

- Delete the second half of unprocessed WorkList[p_b]
  - Array implementation: *atomically* write ***end*** [p_b]
- Add work beyond end[p_b] to WorkList[p_i]
- Read new WorkList.Front[p_b]
  - Read ***front*** [p_b]
- Advance WorkList[p_i] to new WorkList.Front[p_b]
  - Start at new ***front*** [p_b]

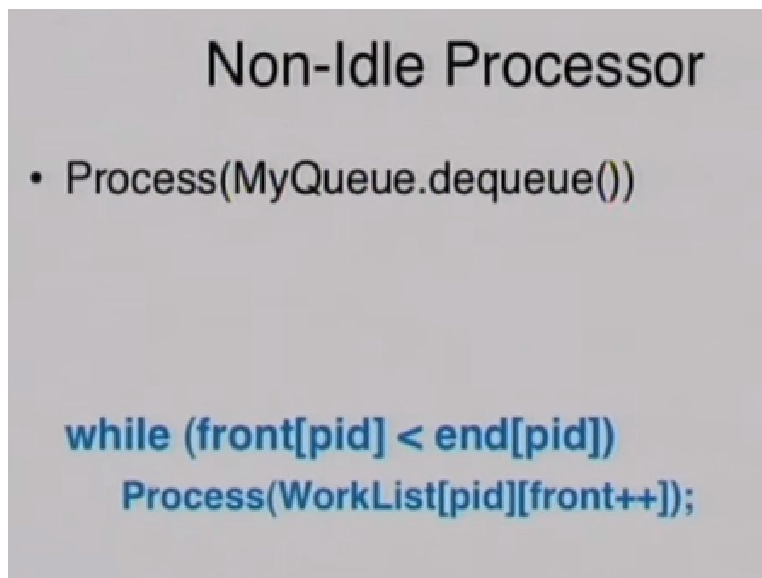
And the algorithm is essentially at the base level you simply figure out a new end for the busy processor and right end okay. And then you in private copy that the meaning elements and so something else okay. So issue is that the end which you wrote the other processor the busy processor may have gone pass that end by the time you wrote it. Because the other processor does not know that you are trying to steal so far.

So let us again not talk about cache issue any time you are sharing things cache is always going to place ((()) (01:41). So you have to turn it into ()) (01:45) thing and that it is given if you are sharing a resource may share it is all time so that is one option you can say I was too slow updating the end how would I know I have read it from the end. If I read before I update that does not have because after that when I update by that time it might have processed.

So I have to update and then read okay so I have updated the end then I read where the front has reached may be beyond. If the front is before end then I know that guy is not going to stop it then you end. If the front has passed end then what are the possible things to do I can continue I can say now are my steal fail and this steal right. Actually that will work if you take the new front you instead of starting from the end + 1 start from front + 1 will it work?

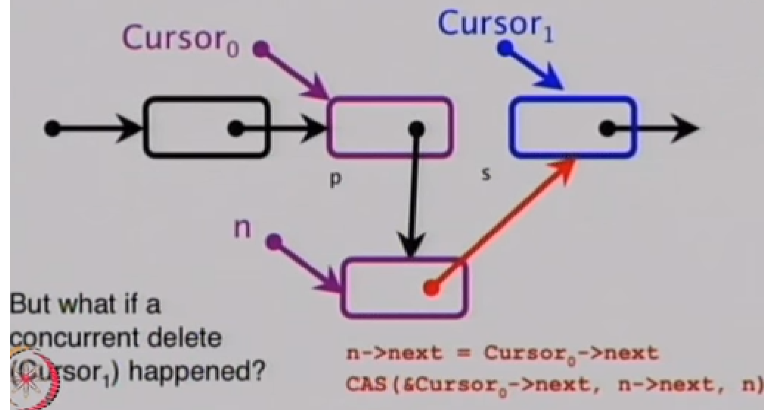
And your end is there old end so that was not changed if their front is beyond the old end then your front is beyond your new old end and so you will not do anything more. It would not go beyond the end but there is a race condition on front if you recall I have not shown you the parallel version here but it is where is it yeah this.

(Refer Slide Time: 04:50)



So you are saying front + + the other the stealer does its see the front + + or after the + + so it is the matter of one job. So if you are okay by I have having one job overlap then may be it is okay if you leave at this point you can guarantee that at most one job will be done by both the busy processor and the ideal processor okay. Alternatively you can simply make this increment atomic okay.

(Refer Slide Time: 05:32)



So the next thing I want to talk about how to do this link list so I have lots of different processors shared linked list and I want to edit and for the time being will say all the edits are happening to this enterprise that will provide you can find elements in the link list or you can add a new note and believe that is the only thing we can. So let us talk about insertion I give you a pointer cursor into the link list and say insert after this that is the typical way which you will insert.

So one way is you create an new note okay this node is as it own reference some end you fill new node which value you want to add into the link list. You make it next pointer to be cursive 0 is where we are inserting cursor level you make its next pointer to cursor 0 dot next right. So far original list is as it was now I am going to atomically switch what am I going to switch cursor 0 dot next = N.

Assuming that is it is not always the case depending on what you are underlying architecture is but let us say a variable assignment whatever size or let us say pointer assignment is atomic okay. It may not be in the reality but the time being which simply say cursor 0 dot next = N right so one will be lost so both of them both of them started off both of them said cursor 0 created a new note.

Both of them made a new point to cursor one right and now they are atomically trying to set cursor 0 dot next to their new node one of them will succeed one of them will fail and that node

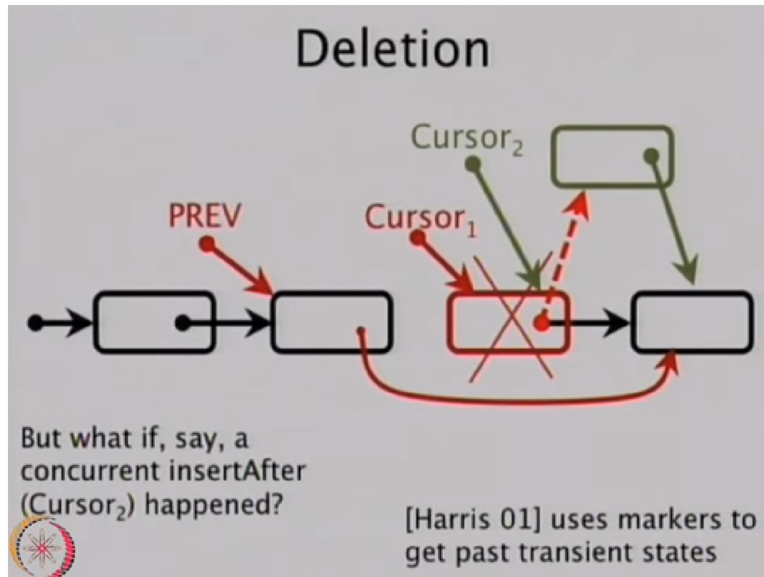
will that insertion will have got lost okay. If you are not dong cache if you are doing cache then what you do how do you what should the cache do? So it knows what is the old cursor 0 dot next that is the thing it is trying to modify and it only modify cursor 0 dot next was still cursor one okay.

So we can now do this in a cache sense what is that face because the node structure has changed two people want to insert after the node it is not clear that I should go first or that guy should go first somebody has to co-ordinate right so when happen to succeed is as if that operation succeeded somebody has to figure out where this new things should go after that the other new thing or before that other new thing.

So previously doing what if concurrent cursor one need time was deleted if you did that this is the caste would have done but it still not work if the one has gone out so not you fix that let see what deletion should do. We just marked what is it in the list? In java you do not even free so you have to mark if you mark it right deletion do not actually happen their only mark deletions then this does not have right the thing you are pointing to has gone out it is still there.

But anybody proceeding through the list has to skip pass this marked deletion I will just scan out to the purpose right. You never delete a note now all you deleted nodes are marked and at some point you will have lots of this junk line between any two valid nodes you would like to delete it but marking is a good suggestion right there is a this algorithm actually uses that.

(Refer Slide Time: 10:36)



So let us say let us see how deletion might work we are deleting the cursor 1 has to delete we have to know the pointer reference to the previous node and that notes that pointer is going to go pass it right so this is going to go away and that is going to have. But if there is concurrent insert and delete then we know that it is happen right so simply doing this pointer assignment using a cache is not going to be sufficient.

What we are looking for too cache right we have note is note on this side pointing to it this note on this side it is pointing two not to make sure that has not changed and that has been changed or the same time. So there are richer variant of cache that some system support most do not so you cannot simply say do this as long as this variable as this then this variable this that. So we have to do something else and that something else will be mark applying like a mark just like we just discussed okay.

So there is a insertion happening and it will go back so how does the mark will work so this is work in 2001. You do mark but have to (()) (12:27) we have to get rid of the thing that you have marked. Mark is only temporary to go across the fact that nobody is doing two things at the same time okay. So how will this work ultimately you have to get rid of mark thing if you have to get rid of the mark things then you are done.

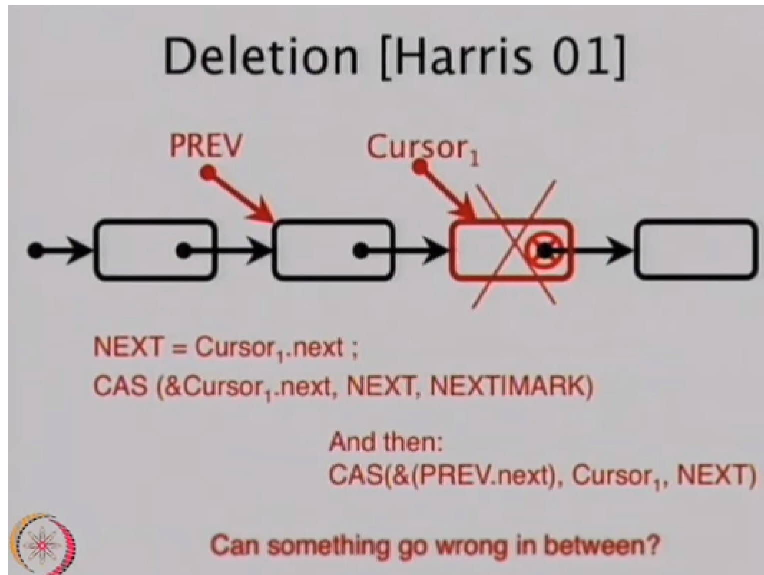
That means you are saying at some point you locked the entire list nobody else works on it and do garbage clean. That is how actually garbage clean typically works they have this macro locks that lock out everything at least for small period of time that is not a great solution that will of course be true right whenever there is mark you will never be inserting after the mark right because typically inserting after research.

You search for a location and then you say insert over there we have to put some kind of lock right because if we simple say do not do if it is unmark then I say after I check whether it is unmarked whether you have unmarked it so there is raise unmark also and it may seen that they postpone the decision when you are actually going to delete it something may go around actually basically it was same way as we just said.

You were kit opportunistically removing this debt notes using cache still so search insert or delete whichever finds at that note removes it okay. Inserting the fourth note being the last note after the second note you delete it and insert it again. But in the mean time that this green note has come along and so the same process remains okay. You do not want the black note no two concurrent may have so multiple such green notes may have but right now we are talking about why you are deleting the red note agree note was insert.

So if after have not deleted the green note you say now talk my note and insert it after this other black note when did you get this other black note if you got it before then you are pointing to the other wrong note then the green is the one we have to insert it okay.

(Refer Slide Time: 16:00)



So this is going to work by simply marking it first right just this way you make a mark and then the best way to do this is simply use some bits of the pointer at least in a theoretic sense not so make it into a strut where there is a mark and a pointer so that goes to (()) (16:19). Here we are using one of the bits of the pointer so when I talk about difference of this entity then I am talking about taking the pointer and I am talking about mark and the mark out okay.

This in the same place so I say cursor 1 dot next as long as it remains next am not going to just blindly mark it is still have to follow the thing I was expecting to repoint into and then I make it mark okay and what you see here is look like next I mark by seeing next or with mark both next and mark has been written it over there next combine with mark or it is not necessarily.

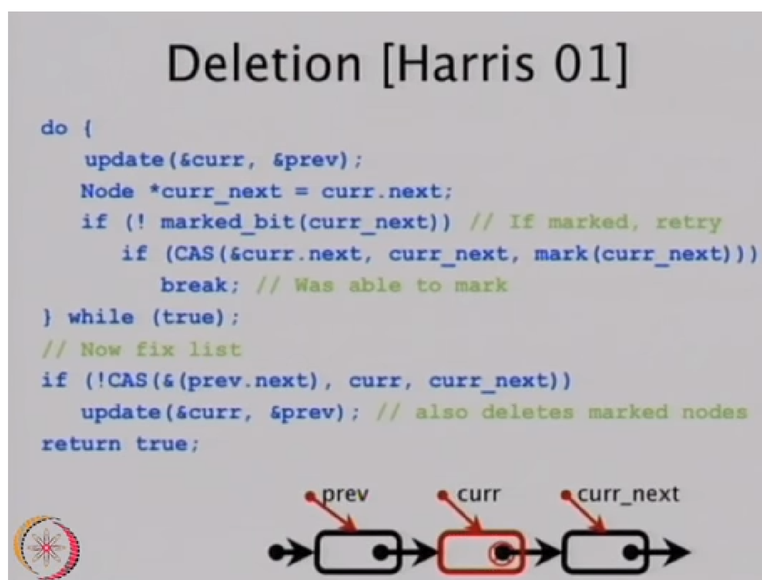
And after you do this somebody come along and do whatever they want to perfectly find they just have to realize that mark modes are to be skipped over and so they should not be inserting after the mark note and now you yourself try to remove the mark so as long as the marked note still in that position right previous is pointing to that mark note nobody has done an insertion just before it you can get go ahead and delete the mark you can go back and write just be the pointer part of it and mar points what thing go wrong.

See the first cache if it does not succeed then you know deletion does not happen right somebody else was missing up with that area at that time. So you have to figure out how to where to delete

next not deletion has happen but the other thing succeed we could not remove the mark I will show you basically I am just seeing says that you first do the mark and then try to remove the mark okay.

But it is possible that when we try to remove the mark you fail because somebody else was in the query. But I want to get rid of as remove the mark note so if you see it is previous dot next is going to point to next okay as long as it is still point into a mar note nobody has come along to do anything better okay.

(Refer Slide Time: 19:18)



And so now I will show what you mean by N then it goes something like this you somehow you have a method of figuring out where the current and figure should be right. For example when I am inserting I fail I update my current previous information I delete in deletion i fail as i update my previous incurrent. So that I can make sure that I know where why should be deleting all where I should be inserting to.

So think of update has as context sensitive method that knows how to tell me where the insertion or deletion should be high. What if somebody else was trying to do something right somebody might trying to delete it somebody else may be trying to insert some just before it so your after that insertion you say are pointed over here and I have this element lies between here and the

next this dot next dot next needs to go and somebody else as modified this pointer somebody else and it has inserted.

Now if you mark it here you may be marking the wrong note now I am assuming here that when you have just inserted something before it this may not be the note you want do delete right. Let me finish this session and continue this discussion. So let us get back to where we where we are talking about talking a note that has been deleted and we just talking about why we just need to be cache okay.

And it is checking that the note that you are deleting is still pointing to the note that we are expecting after it right. That by itself seems in a course seems like it is not needed but remember when you write a mark you are writing a pointer that you have read before modified with the mark right in this scheme of things pointer and mark go to (()) (22:35) so you have read a pointer to the next and now you are saying write the marked thing marked same pointer but marked.

If in the meantime somebody has inserted something then the pointer you are writing is stay right so you could be writing the same pointer so it is not simply saying turn to this on if you choose to implement it that way that complication will show of somewhere when you have it one item protected by cache you will have two items protected by the cache in the second right. So it is going to show up somewhere okay.

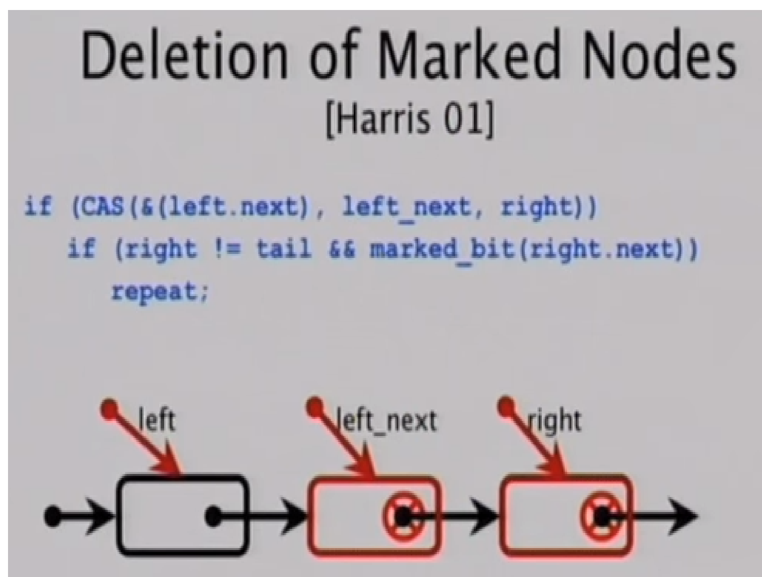
So you have a method that figures out where the pointer should be you have a methods which checks if reference has that bit and on for any reference you can say it is marked reference and unmarked reference and so it checks if it is not a marked entry then you are going to do the cache thing cache is a marked version so when you say when you see mark the current pointer. So the function mark is adding that bit the next bar pointer from the previous pointer otherwise will cache I have seen the Boolean variant of cache.

So you go back update your pointer and then delete again right so you go to the end of the loop and go back to the true part if you succeeded then you break out of it that means you where able to figure out where you have to delete from you are able to hide a marked reference to the next

thing over there and if come out now you simply try to cache the marked out and if you succeed good if you did not again something else some out it is possible to continuously do this loop until you do succeed.

It is suggested by this authors of this paper is to build it into the underline semantics of the list where you say that if you are seeing the marked reference then you skip it in the fine and it insert but you delete it there instead of repeating right now you opportunistic. Any time you see the marked thing you likely to repeat it and then skip over it if you do not then it is like correct list okay.

(Refer Slide Time: 25:51)



But this is what it would do in general okay to keep going forward until you keep seeing a marked reference because now you can have multiple marked references one after the other okay. That is as much as we were going to talk about the examples of lock free synchronization are there any questions on that?

Here also can we maintain be each pointer out of value like we did in the previous example we did for each processor we are trying to say whether it has been acquired by some other processor or not so if we just did maintain up value to the each point and whoever is going to modify a note it will lock two pointer then nobody else. But then there is lots of locks right. Each note there

will be then lock exactly so there is variable it is not really for a note it is for the connection of two notes right.

And so every connection you have got a variable that you have to cache and you have to say I have to edit the variable that link or not with that actually lots of extra variables that you have to deal with okay.

(Refer Slide Time: 27:34)

External Merge-Sort

- Assume n data items, with m items of core (memory)
  1. Read m units of data and sort (in-core)
     - Write the sorted block to disk
  2. Repeat generating S = ⌈M/m⌉ blocks, each of size m
  3. Read the first B = M / (S + 1) items from each (sorted) block
     - one extra for temporary output
  4. Merge S-way and store the result in the output buffer
     - When the output buffer is full, evict it
     - If any of the C input buffers gets empty, fetch the next B items

I am going to quickly go through some of the things that I was going to talk about this is the one that I was not going to talk about since it was brought up there is one slide for out of course order okay.

You have N elements to sort okay and this is essentially a variant of the algorithm we have not the notes out of programming values. So you have M items store in the memory store in one time N items to sort okay and this is merge based so you take N things at a time M things in a time because only that much will fit in your memory and sort it chuck. So now at the end of it you have M blocks of M things each sorted right but N over M of that right.

And you have to merge them together one method is you figure out how many such things are M over M celling S Such things are there and you make a small buffer depending on how did this value S is the buffer is M over S + 1 right meaning that you divide your memory available into a

S + 1 chunks each chunk will get B items so now you have enough space to read B items to get each of this blocks you have created.

So now you are going to do merge across all of them so you have so many blocks some items on B blocks you only have to look at the first M the smallest of the first items is going to go at the result block so the result block is YS + 1 is there right. S things for input one more thing for output when the output blocks get saturated filled it you put it into a file you output it and empty that block out and continue whenever input blocks gets (()) (30:03) because all of these elements as gone into the saturated of into the output block you refill it.

You go to that particular chuck and read what are the problems there are problems that you may see first of all before we see what problem is how many times each data items are there from the disk yes. So here the may thing you are worried about is how many blocks you are reading from the disk rather than the number of CPU operations that may also clear typically this blocks will be big enough and you want to do efficient internals solving.

But for the time being we are not worrying about that twice right in the beginning when you guided you sorted it and wrote it read once read once again the next time you read the next appropriate block appropriate stream if you are and write once read once write once okay. Twice read and twice read okay what are the in double this case nothing hard to what happen to sort you are only using implements of M memory that can also be optimized but yes that is one problem.

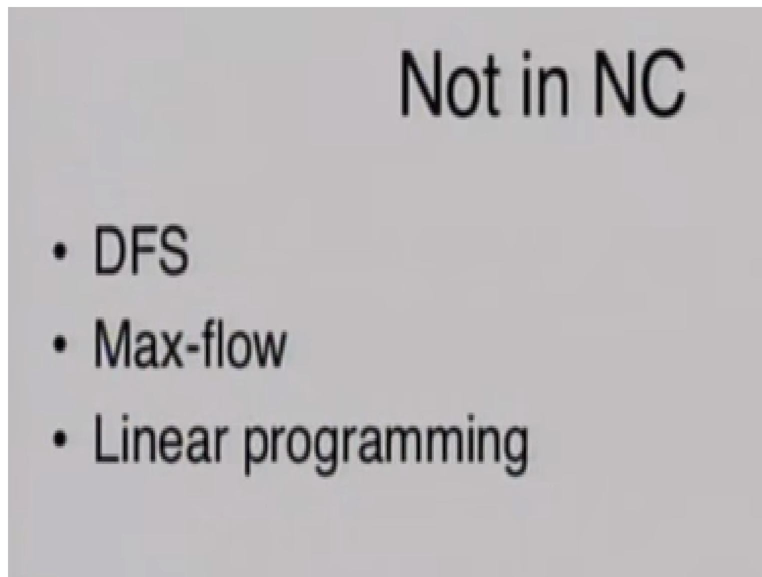
That you disk space doubles typically disk space what you not worried about this case so there is no need for the output block to be the same size as seems to the blocks yes that is a good point. So in the beginning you are reading all of them right so you are waiting for the all reads to happen before you can start the merge because of the first element have to be read before you can start the merge.

And then potentially there is a lot of such stream there you generated so the other end of the stream you may be two you just read two the smaller of the two you right of or it may not be the

grant small. So you have to merge square and now you are reading and writing it more than twice and typically there will be a trade out so you will say do not read only two effect time and merger them but may be read a large number of them merge them and then the few more passes away so now instead of reading it twice you are reading it few times and merge it.

And again parallel I/O is an other issue that are not thinking about but all of these processors are reading at the same time and then to worry about where this difference blocks are stored them.

(Refer Slide Time: 33:58)



So that is one thing the other thing I wanted to remind you if nothing else that there are very simple problems run of the mill daily problems which are not parallel which means that any time you see a sequential variant you have to do difference search you have to think about new algorithm non depth based out of it and so on hence so forth so how would depth before search B slow it is very sequential right we go from one node to the next we can come and go further.

What would be a kind of parallel variant you will be exploring from varies places essentially then you have to worry about then two start have to meet when the algorithm changes in flavor significantly.

(Refer Slide Time: 35:11)

Prim's MST Algorithm

- Start at an arbitrary vertex
  - Must be in MST
- Choose the min cost edge out of the vertex
  - Must be in MST
- Grow MST by growing across the minimum cost edge leading to a non-MST vertex

Let me show you one where it is not quite depth or search but has that favor but more importantly just a pure parallelization am not going to go into details of the more expensive ah more optimized less expensive algorithm which too exist but a basic straight forward parallelization this is the twins planning change does it works you grow the minimum.

You start by any arbitrary vertex and you say that minimum cost is coming of this vertex has to be in the minimum planetary and you bring it into the honesty and you say let us look at that all is going to out of MST the smallest has to be MST that bring it in so very sequential process right and so it is not great algorithm for parallelization but if you have to parallelize it how you would do.

You can do nothing about this fact that you have to grow one at a time so you will say let us at least parallelize the growing how much time it takes to grow I can be speeded up.

(Refer Slide Time: 36:22)

Parallel Prim's Algorithm

- *n* iterations of:
  - Partition *n* vertices among *p* processors
    - distance vector d also partitioned
  - Each processor selects its closest node to MST
    - reduction to select globally closest node
  - Insert minimum distance node into MST
  - Each processor updates its part of d vector

And so a parallel version of Prim's algorithm would simply repeat this N times and each what will be done N times it will say so again I am assuming here that P processors and nodes and each node is going to responsible for N over P of an each processor is going to responsible for N over P of the nodes okay. And it is going to ask is any of node connected to MST.

If it is connected to MST what is it cost is going to find out the minimum of what is the it is nodes going to ages of it is nodes going to a MST. Going to find them in can we do a reduction find grand minimum and global minimum and that node becomes the next node to be added when you done this (()) (37:20) okay if it is shared then you write as whoever of course reduction that everybody knows what the value is and update the MST.

They had that node to their local give of the MST and repeat it okay how long would that is so order M is outer vibration what about the inside. Basically it is what you share of processor notes is right for every note is going to ask the question how many things it is connected through in the MST the mysterious enough number of nodes. No even in the beginning what you were saying is I have got all these nodes and I do not know how many are them are connected to them in MST.

So I am going to figure out right then we are going to have rear away then see which are one of the and you can do it in a more optimized way also this is kind of force maximum planetary minimum planetary also no right you still have to once you first have to figure out what is in the

graph was your partition in the graph it is not just the processor but their ages also be partition which ages am I considering my MST as ages go across.

For example suppose we said we are starting at two places right and we simply follow this everybody says take your minima does not work you are going to get something else we are not going to get any that there are problem that are going to happen when you have speed up when you are going to come to no way knowing which should take there is no straight forward parallel version of it mean finding is not sequential but there is N by P has to be done there is N by P that you have to find the minimum across and so that has to be included.

So it is going to be N N by P times N + log of P time okay alright I want to stop here there is nothing I can begin now that is going to end in time.