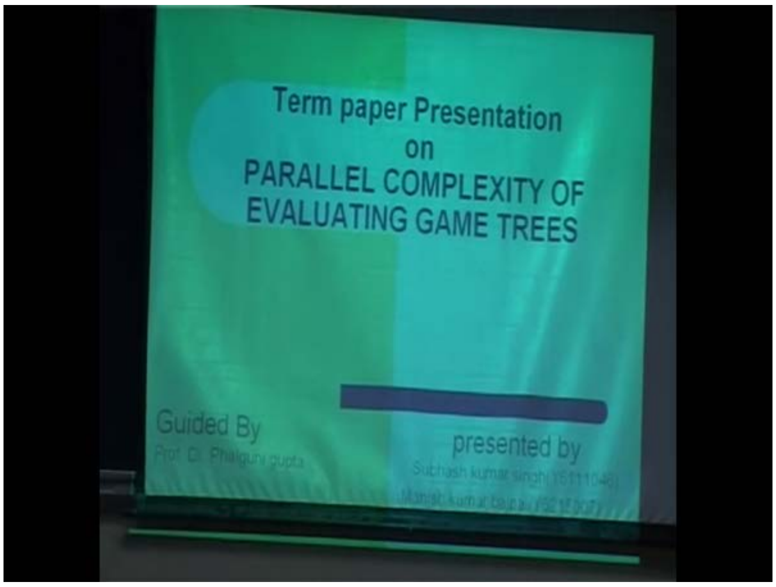
Parallel Algorithms

Prof. Phalguni Gupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 25

(Refer Slide Time: 00:14)



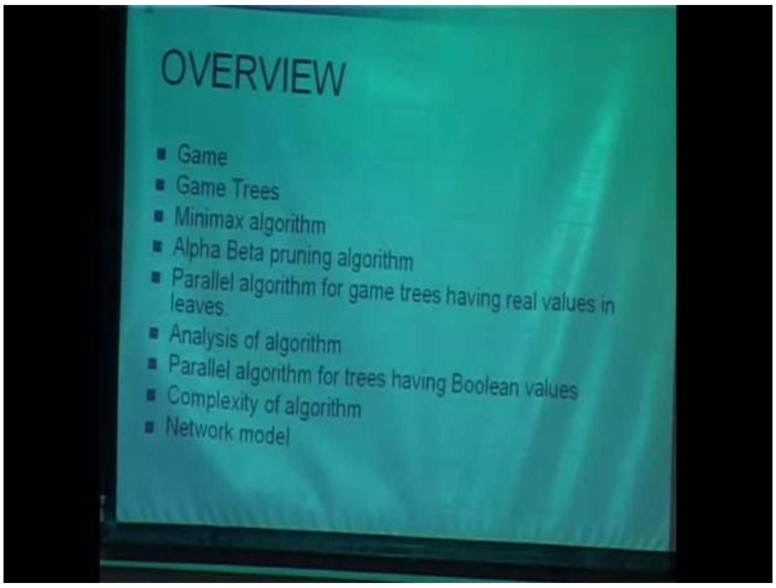
Term paper Presentation
on
PARALLEL COMPLEXITY OF
EVALUATING GAME TREES

Guided By
Prof. Dr. Phalguni gupta

presented by
Subhash kumar singh (16111046)
Manish kumar bajpai ([illegible])

Good morning to respected sir, and hello friends. We Manish Bhajpai and Subhash are presenting the term paper presentation on parallel complexity of the evaluating game trees.

(Refer Slide Time: 00:25)

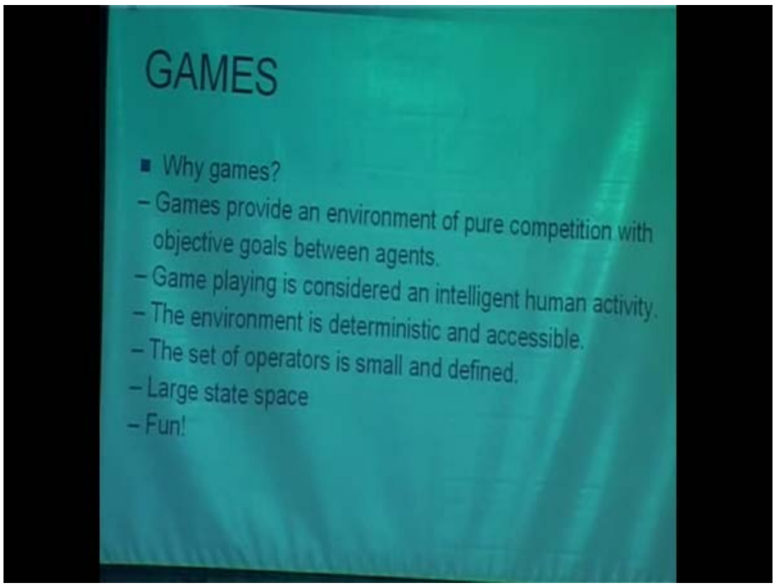


OVERVIEW

- Game
- Game Trees
- Minimax algorithm
- Alpha Beta pruning algorithm
- Parallel algorithm for game trees having real values in leaves.
- Analysis of algorithm
- Parallel algorithm for trees having Boolean values
- Complexity of algorithm
- Network model

Here we will mainly discuss the, what is the game and what is the game tree? How do you these game trees problem, can we solve using the some algorithms in Minimax algorithm and the alpha beta algorithm. Then we proceed to the parallel parallelization of these algorithms. Here we will introduce two types of problems, where the game trees have the real values in their leaves and in second where the Boolean values are restored in the leaves. Then we will discuss the complexity of the problem and the network model which are appropriate for these algorithms.

(Refer Slide Time: 01:02)

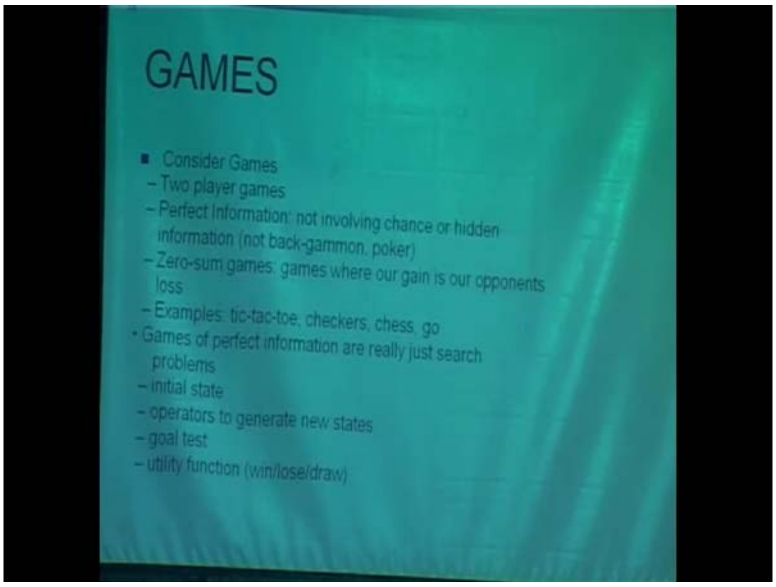


GAMES

- Why games?
  - Games provide an environment of pure competition with objective goals between agents.
  - Game playing is considered an intelligent human activity.
  - The environment is deterministic and accessible.
  - The set of operators is small and defined.
  - Large state space
  - Fun!

First is the what is the games? The game provides the environment for the pure competitions among though two agents or two or more than agents, and game is the purely intelligence human activity and no doubt it is provide the fun to us.

(Refer Slide Time: 01:19)



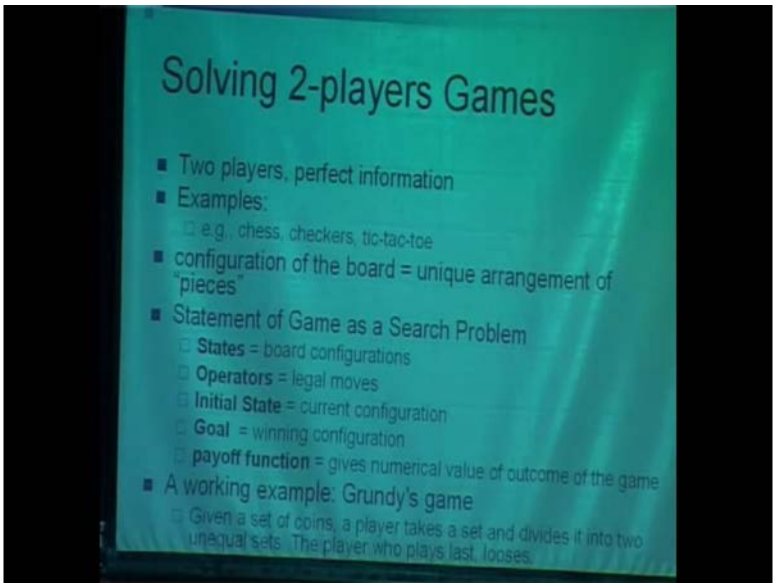
GAMES

- Consider Games
  - Two player games
  - Perfect Information: not involving chance or hidden information (not back-gammon, poker)
  - Zero-sum games: games where our gain is our opponents loss
  - Examples: tic-tac-toe, checkers, chess, go
- Games of perfect information are really just search problems
  - initial state
  - operators to generate new states
  - goal test
  - utility function (win/lose/draw)

Mainly, we are discussing about 2 types of games, where we have the information, perfect information about the game what the options and what will the result. And in another case, we are discussing about the games where we have no problems. In the games here, we have the perfect information there is nothing but it then something searching problems.

In this slide we all after this, we are giving in places of 2 players game. All the game trees are made for 2 player’s game.

(Refer Slide Time: 01:53)



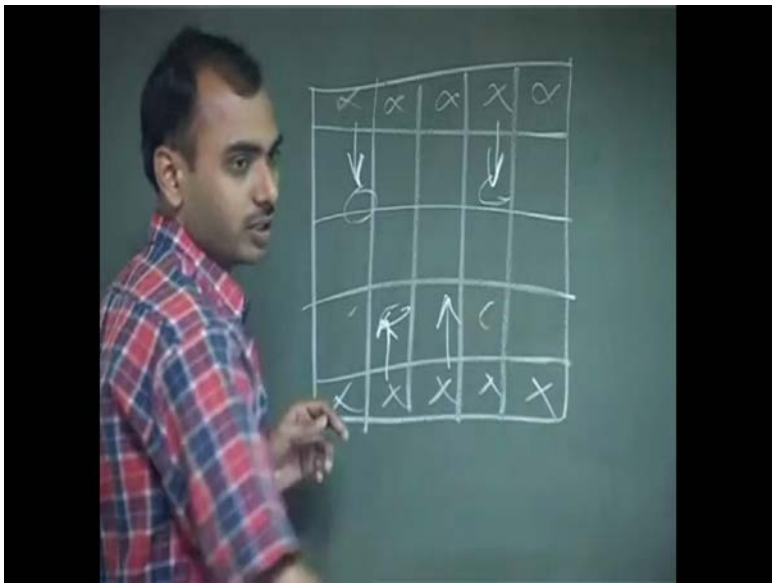
Solving 2-players Games

- Two players, perfect information
- Examples:
  - e.g., chess, checkers, tic-tac-toe
- configuration of the board = unique arrangement of "pieces"
- Statement of Game as a Search Problem
  - **States** = board configurations
  - **Operators** = legal moves
  - **Initial State** = current configuration
  - **Goal** = winning configuration
  - **payoff function** = gives numerical value of outcome of the game
- A working example: Grundy's game
  - Given a set of coins, a player takes a set and divides it into two unequal sets. The player who plays last, looses.

For solving 2 players game, here in game trees we need perfect information means we need the information of node of leaves, that may be for any games tree, In any game there are 5 things, states teaching is a making in board operators, initial moves, goal and payoff function.

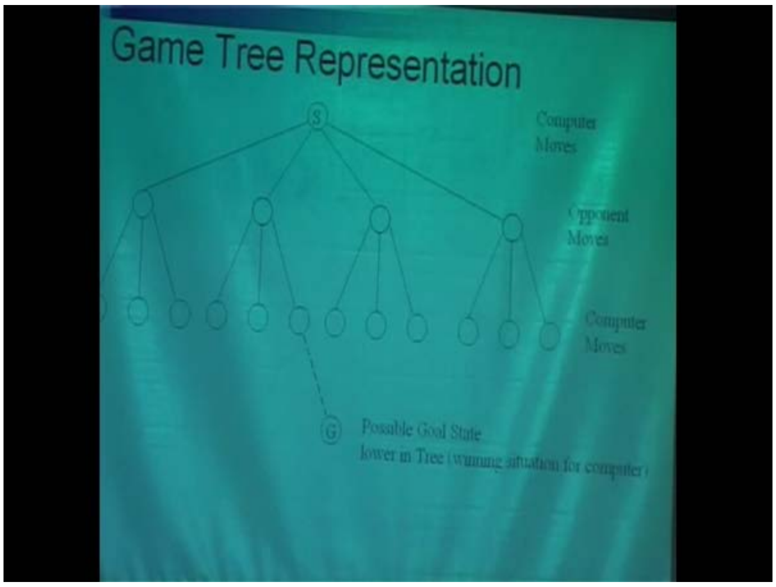
So, this is the initial state which is the board configuration in easily.

(Refer Slide Time: 02:29)



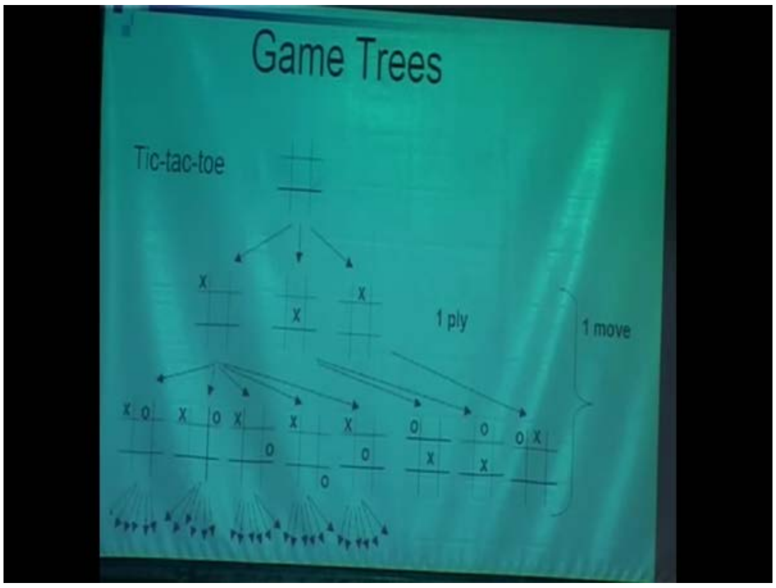
If we take a one move, this is the legal move which is known as the operator and second one is the goal. Goal is the winning configuration. How we take the moves to win this game and what is the remaining in states? This is the board configuration.

(Refer Slide Time: 02:50)



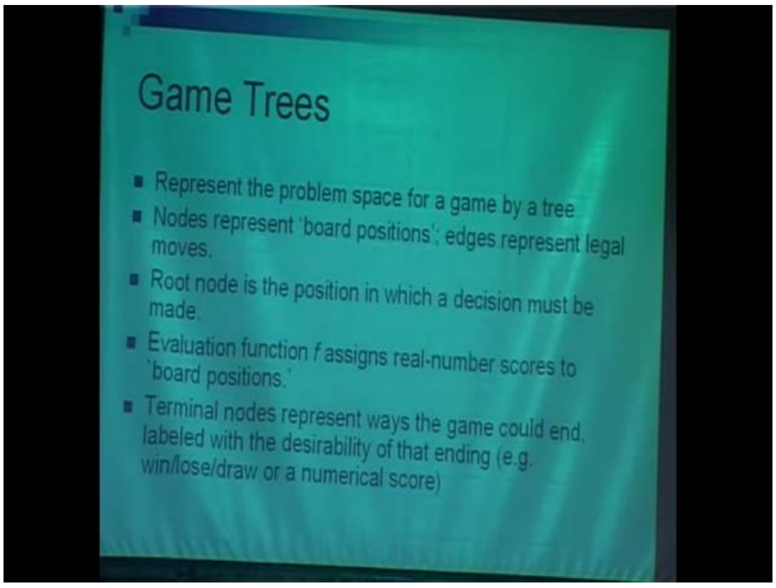
All these streets are the board configurations and goal is for both players, one is for winning, one is losing. Here, it is game how the game trees are made. This is for two player game, one is computer, one is opponent or user, this can be changes in any, where this is first play of move first ply means 1 move is made of two ply’s, in this slide it is clear.

(Refer Slide Time: 03:17)



For example, this is tic-tac-toe game. Here this is first ply, where he made cross and the second ply which is played by second opponent player that is he is giving 0. And this is 1 move, for one move, one move is consist of two players in game trees, also the same thing means one is route and after every consecutive things that is one ply.

(Refer Slide Time: 03:48)

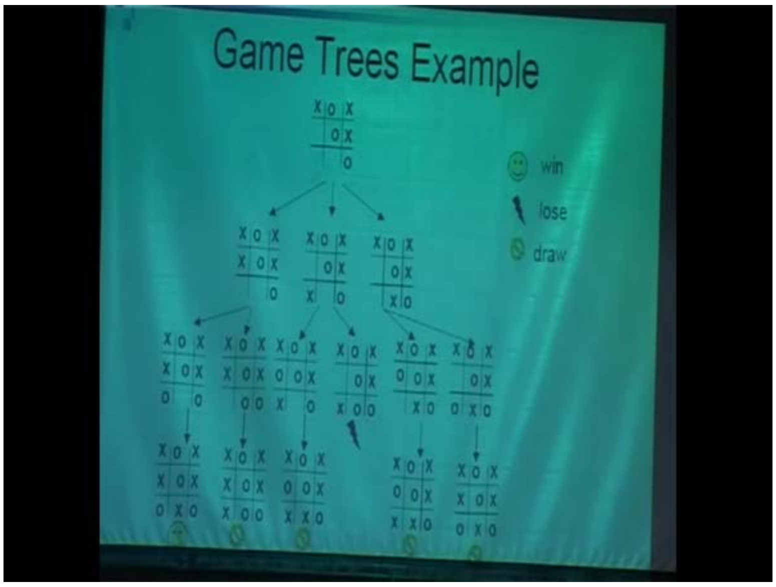


Game Trees

- Represent the problem space for a game by a tree
- Nodes represent 'board positions'; edges represent legal moves.
- Root node is the position in which a decision must be made.
- Evaluation function *f* assigns real-number scores to 'board positions.'
- Terminal nodes represent ways the game could end, labeled with the desirability of that ending (e.g. win/lose/draw or a numerical score)

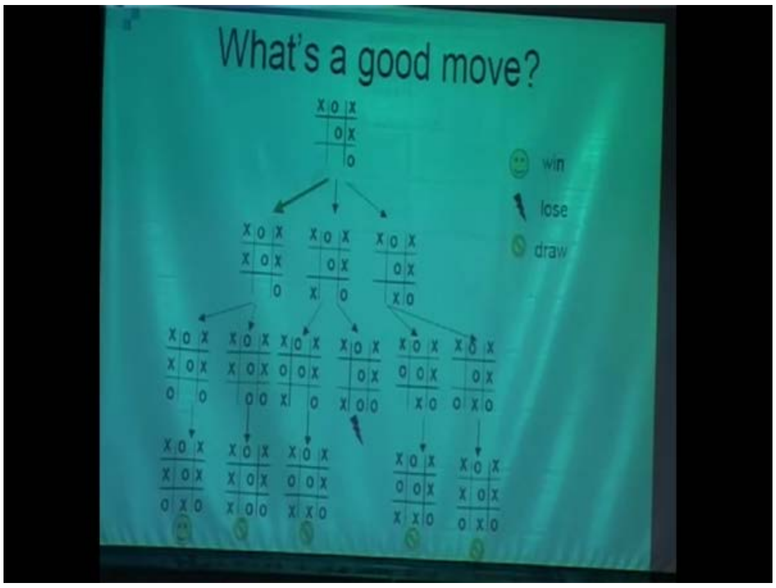
Game trees: Game trees representation of any problem by a tree, in a game problem by a tree is called game tree. A node represents board position. For example, this positions root node is the position in which decision must be made. Root node is the final position which says that which player wins. Evaluating function of every node and by evaluating nodes of child node, we calculate ancestors of that node? And finally, we calculate what the root node which says which player wins.

(Refer Slide Time: 04:35)



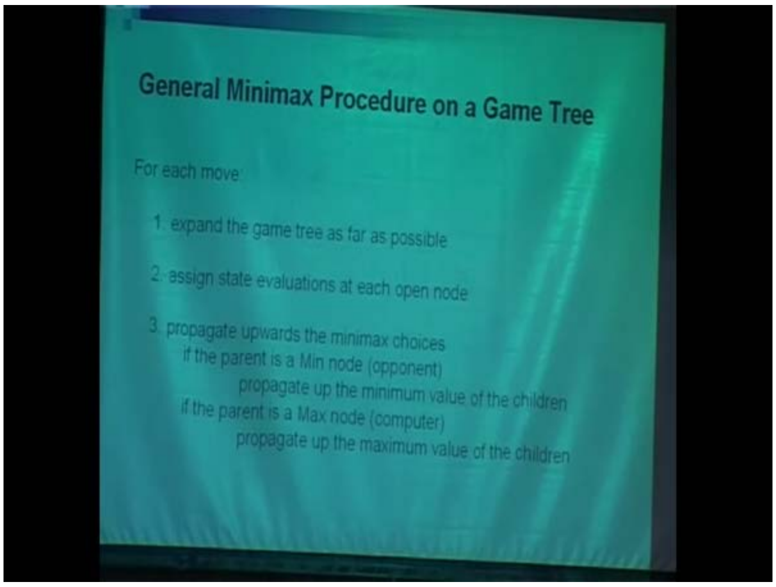
This is the example of tic-tac-toe game, how they here it is win by. This is the winning. Next one, here it is winning and this is simply will be node.

(Refer Slide Time: 05:08)



And good move good for that 1 because here it is many nodes, this cannot be a good move because the chance of lose is here. Actually this cannot be good move because here is chance of lose here. No, no, 1 is for 0 but, cross is loses the game, 0 is opponent's move. 0 is opponent. So, here it is chance. 0 is opponent then that was also. No, here the chance is this. Why part to that? 1 minute. Act to that stage we has already. Here he did not cross, cross, cross, cross and then here he loses. 1 minute, 1 minute, here he loses and here there is some problem. Actually, there is some mistake in making slides actually.

(Refer Slide Time: 06:25)



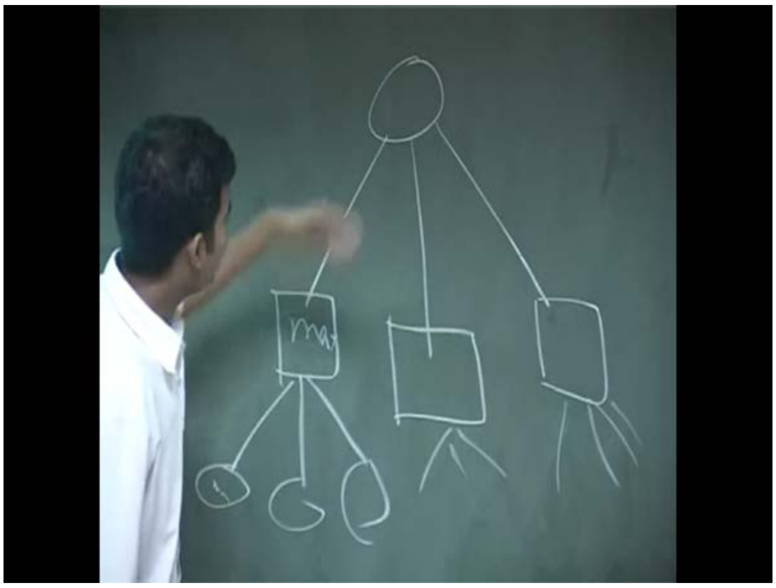
General Minimax Procedure on a Game Tree

For each move:

1. expand the game tree as far as possible
2. assign state evaluations at each open node
3. propagate upwards the minimax choices
   if the parent is a Min node (opponent)
      propagate up the minimum value of the children
   if the parent is a Max node (computer)
      propagate up the maximum value of the children

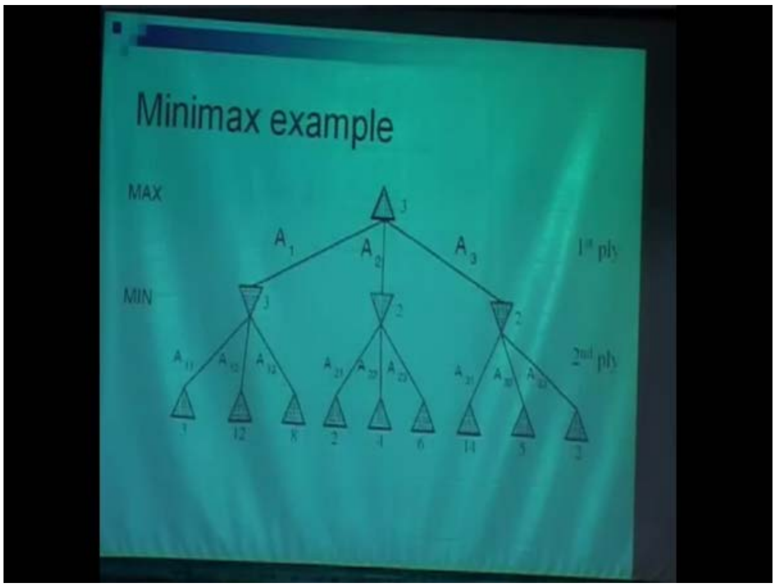
Last one is; this is may be possible that, this is not so important because we are making in games. General mini-max procedure in this game, general mini-max there is also mini-max theorem, which says that mini-maxing and max minimizing are both same. But, we are not going in that they are saying we are making a processor.

(Refer Slide Time: 06:54)



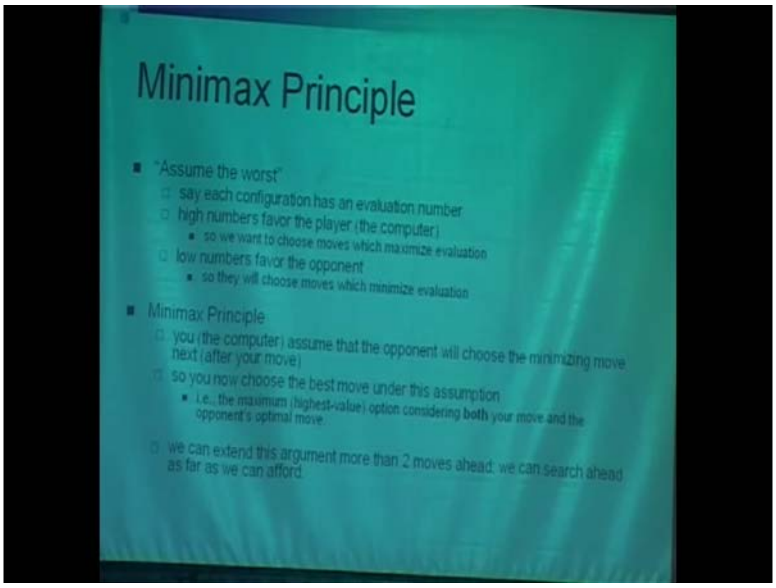
This is mini-max position is maximizing. This is minimizing node or this is maximizing node, then it tries that maximum value of all thing and this is minimizing node. So, we try minimum value of all the value and final output is the value of node. If it is a some stress out value.

(Refer Slide Time: 07:43)



And in many, the most example that is 0, where there is negative and positive.

(Refer Slide Time: 07:49)

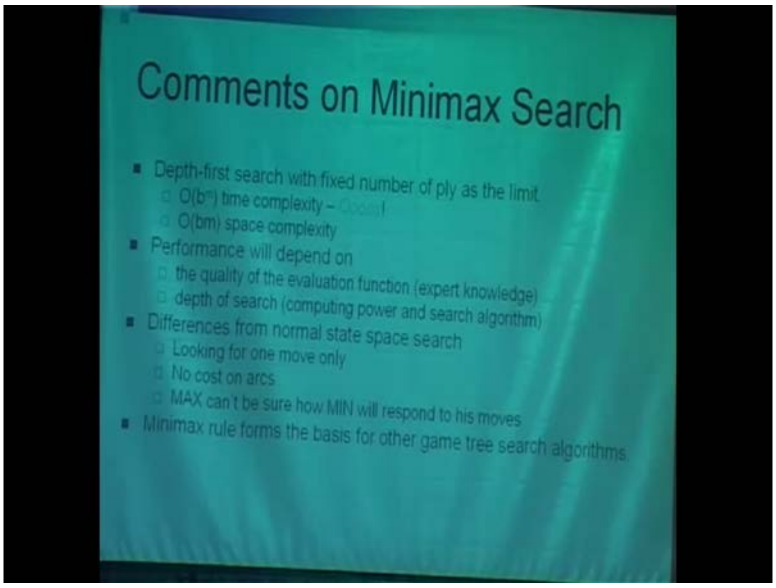


Minimax Principle

- "Assume the worst"
  - say each configuration has an evaluation number
  - high numbers favor the player (the computer)
    - so we want to choose moves which maximize evaluation
  - low numbers favor the opponent
    - so they will choose moves which minimize evaluation
- Minimax Principle
  - you (the computer) assume that the opponent will choose the minimizing move next (after your move)
  - so you now choose the best move under this assumption
    - i.e., the maximum (highest-value) option considering **both** your move and the opponent's optimal move.
  - we can extend this argument more than 2 moves ahead: we can search ahead as far as we can afford

Both value says that same thing which we has, we said that low numbers is for opponent and high numbers is for the first player.

(Refer Slide Time: 08:06)



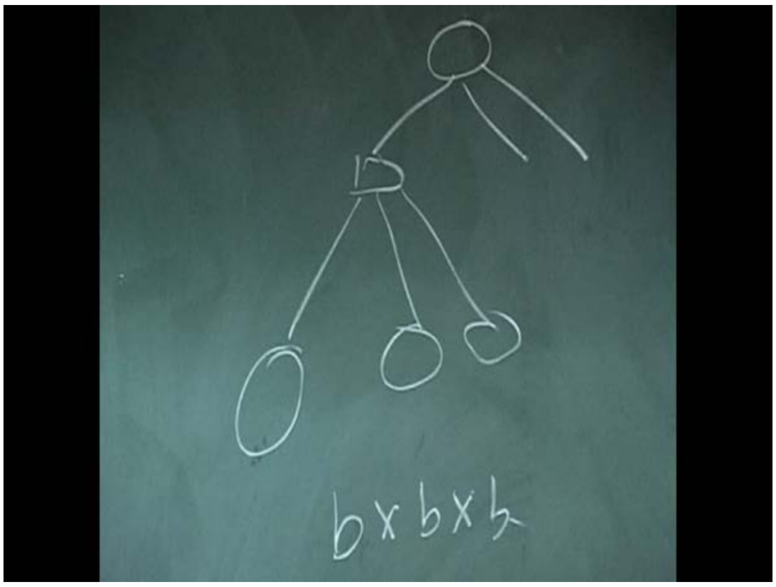
Comments on Minimax Search

- Depth-first search with fixed number of ply as the limit
  - $O(b^m)$ time complexity – Ooops!
  - $O(bm)$ space complexity
- Performance will depend on
  - the quality of the evaluation function (expert knowledge)
  - depth of search (computing power and search algorithm)
- Differences from normal state space search
  - Looking for one move only
  - No cost on arcs
  - MAX can't be sure how MIN will respond to his moves
- Minimax rule forms the basis for other game tree search algorithms.

This is and that here it is said that, How? What is the complexity of minimax? Get that because, the main m it has to reduce the complexity on game evaluate evaluating game tree then general complexity of minimax tree is b to the b to the power m and complexity that is very much because in this example the game tree is very solved but, in general that is very large.

Because in tic-tac-toe example, the game tree size is the number of nodes are 120. So, you can think that how much large for example, in j's that may be in gradients because that two is so simple game. So, b to the power m is b is here depth, b is the degree and d is the depth.

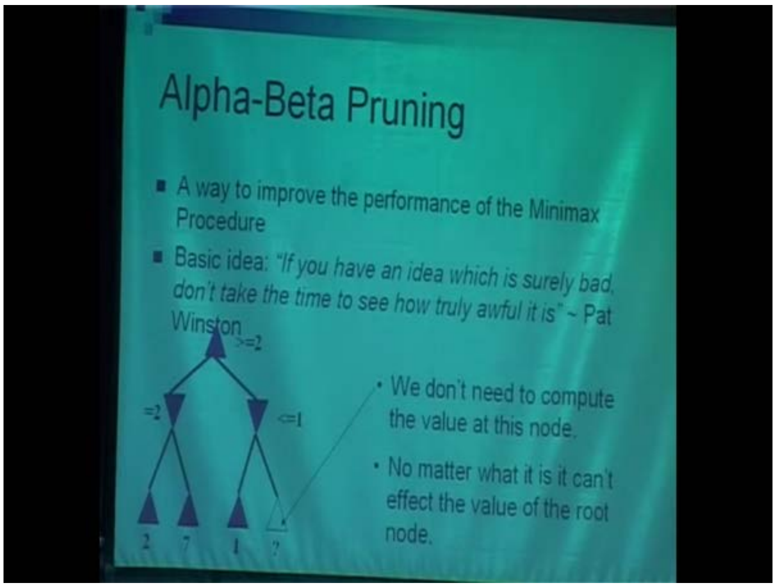
(Refer Slide Time: 09:06)



You are b took this comes because we have to calculate all degree. First calculating group nodes, we have to evaluate all degree and then calculate this and after that again so on, it comes b into b into b up to up to 11. See this is the, you know this is in the worst case scenario is locate. Yes, I think some cases that we will not know. Some cases in tree will not know means. For example, that it is a case of good decision with me arrived right. Node this is, there is a he cannot evaluate the (( )). No this is that a node will be gain which that a Sir we, You cannot go further.

Actually here means early two continuous, early two players are there we and nobody is Say, in the chess board two players are play after sometime the both of they were it is similar to any decision. That comes root node is 0, if you are giving the negative value to the opponent and positive value to the player, then the evaluation of root node is 0. The evaluation can be the performance will depend upon the evaluation function how we evaluate the nodes; this is also important, all though we are not going in this thing that how we evaluate.

(Refer Slide Time: 10:52)



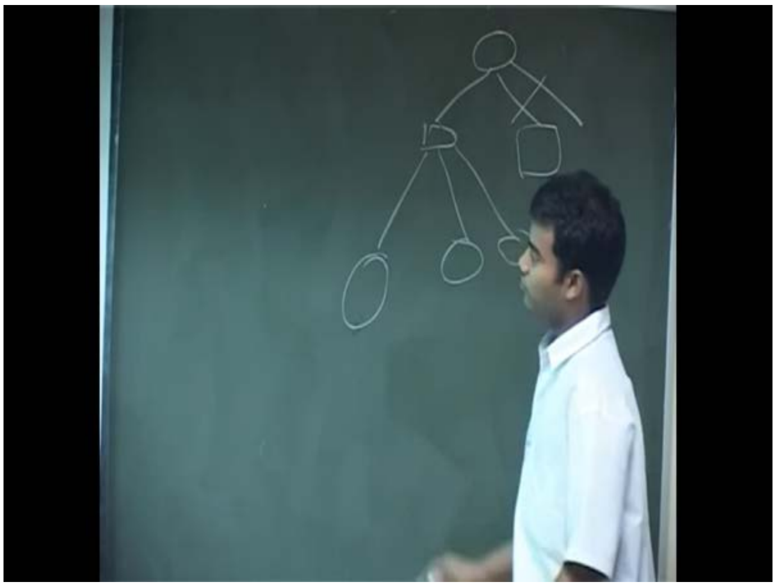
Alpha-Beta Pruning

- A way to improve the performance of the Minimax Procedure
- Basic idea: *"If you have an idea which is surely bad, don't take the time to see how truly awful it is"* ~ Pat Winston

- We don't need to compute the value at this node.
- No matter what it is it can't effect the value of the root node.

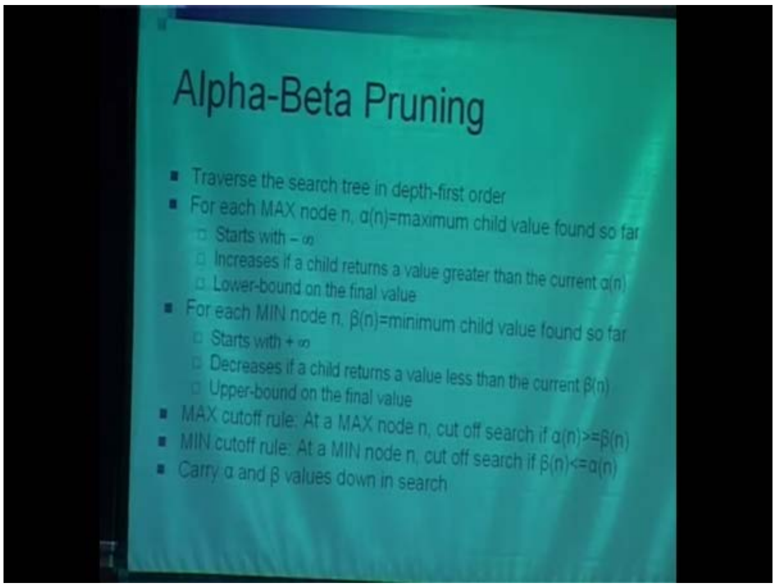
Then, we come that is b to the power d then is very much. So, what we decided that is what we did in Alpha-Beta Pruning. If we know that, this node is maximum value that means did not calculate on algorithm.

(Refer Slide Time: 11:16)



And in this if you know that, this node is minimum of then we need not calculate all the other nodes. So, these are all nodes are provide this distinct called Alpha-Beta Pruning although we extend this idea in our parallel algorithm also.

(Refer Slide Time: 11:40)

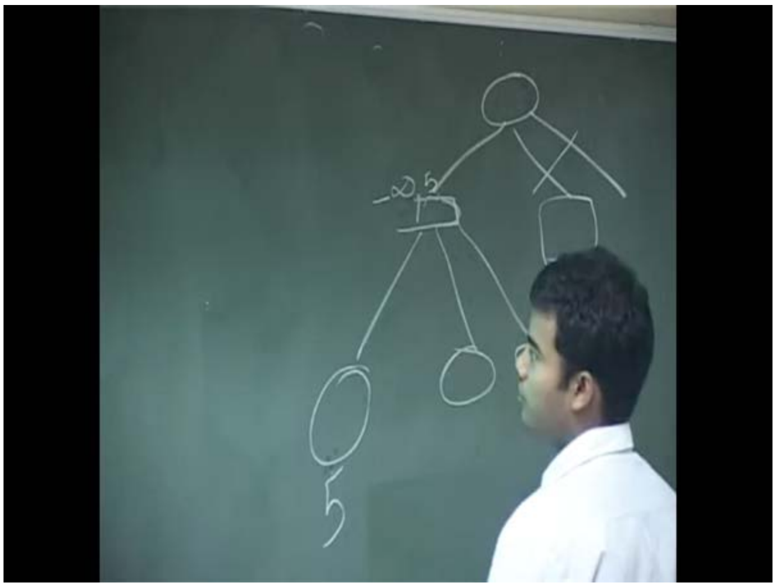


Alpha-Beta Pruning

- Traverse the search tree in depth-first order
- For each MAX node n, α(n)=maximum child value found so far
  - Starts with – ∞
  - Increases if a child returns a value greater than the current α(n)
  - Lower-bound on the final value
- For each MIN node n, β(n)=minimum child value found so far
  - Starts with + ∞
  - Decreases if a child returns a value less than the current β(n)
  - Upper-bound on the final value
- MAX cutoff rule: At a MAX node n, cut off search if α(n)>=β(n)
- MIN cutoff rule: At a MIN node n, cut off search if β(n)<=α(n)
- Carry α and β values down in search

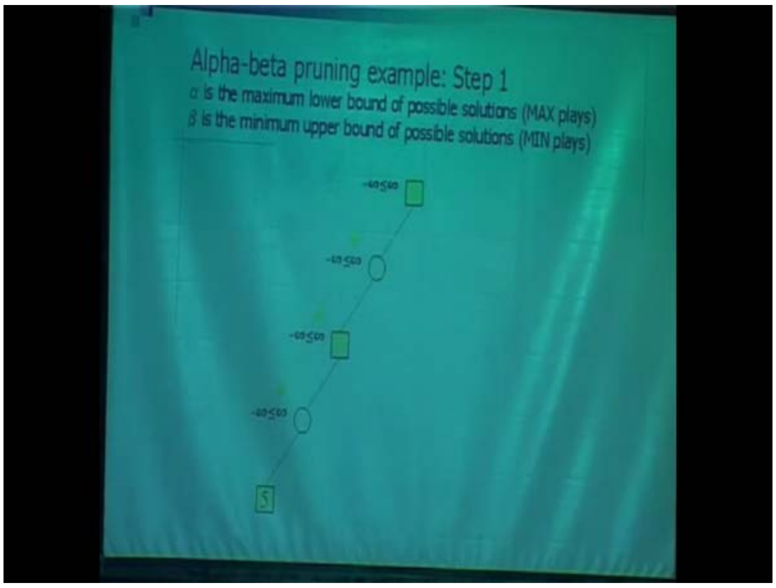
For algorithm quick alpha beta pruning because we need in computer science everything should be elaborately we do it value infinity and minus infinity to the maximizing node and minimizing node and maximizing node both and we compare with the nodes for example, this is minus infinity if it is 5, because 5 is greater than minus infinity.

(Refer Slide Time: 12:13)



So, with those to there and if any other nodes if you know that, these nodes are less than 5, then we need not to conclude this. We will also use this is the algorithm.

(Refer Slide Time: 12:31)



Alpha-beta pruning example: Step 1

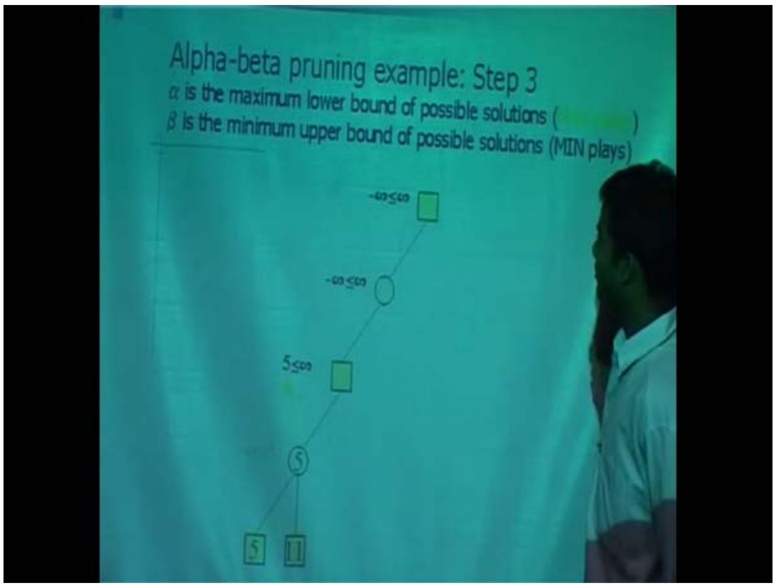
α is the maximum lower bound of possible solutions (MAX plays)

β is the minimum upper bound of possible solutions (MIN plays)

$-\infty \leq \infty$
$-\infty \leq \infty$
$-\infty \leq \infty$
$-\infty \leq \infty$
5

In first step, all nodes here it is minus infinity here. It is 5, because 5 greater than infinity. So, the value of node is 5.

(Refer Slide Time: 12:49)



Alpha-beta pruning example: Step 3

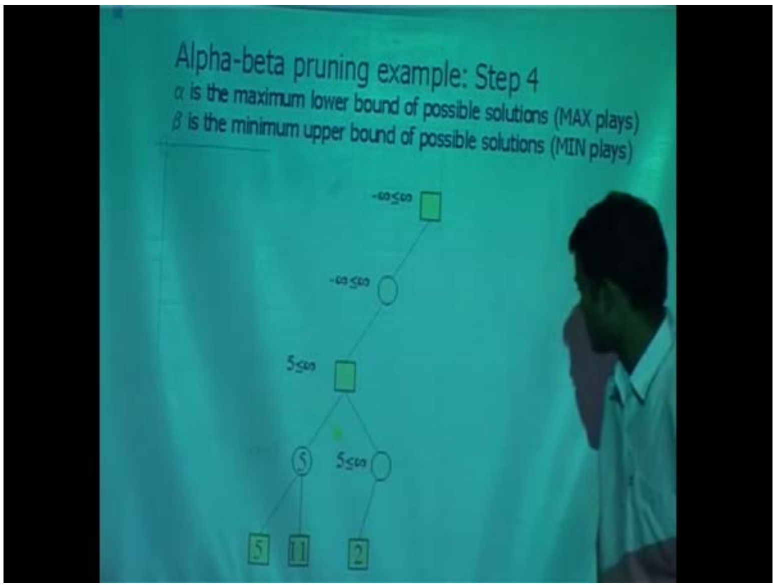
α is the maximum lower bound of possible solutions ( )

β is the minimum upper bound of possible solutions (MIN plays)

$-\infty \leq \infty$
$-\infty \leq \infty$
$5 \leq \infty$
5
5
11

Now, this base 2 means this is now for it pruning number or alpha beta.

(Refer Slide Time: 12:59)



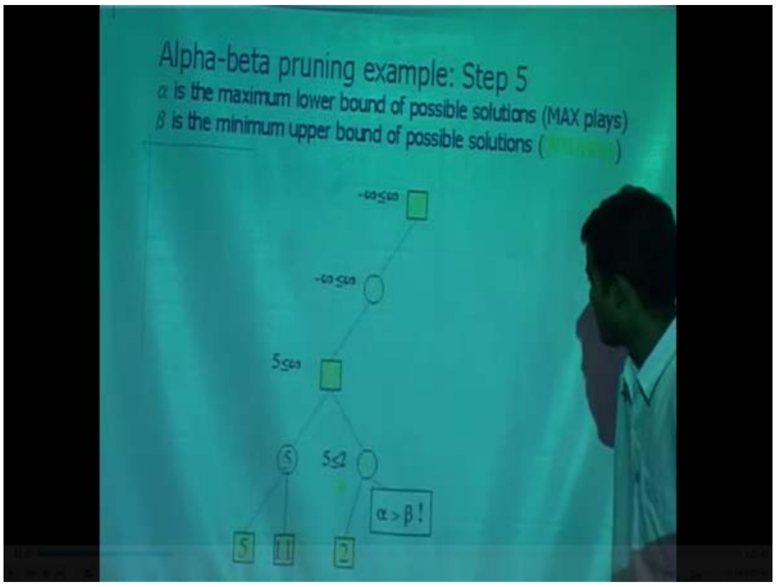
Alpha-beta pruning example: Step 4

α is the maximum lower bound of possible solutions (MAX plays)

β is the minimum upper bound of possible solutions (MIN plays)

Now, next slide here it is 2.

(Refer Slide Time: 13:04)



Alpha-beta pruning example: Step 5

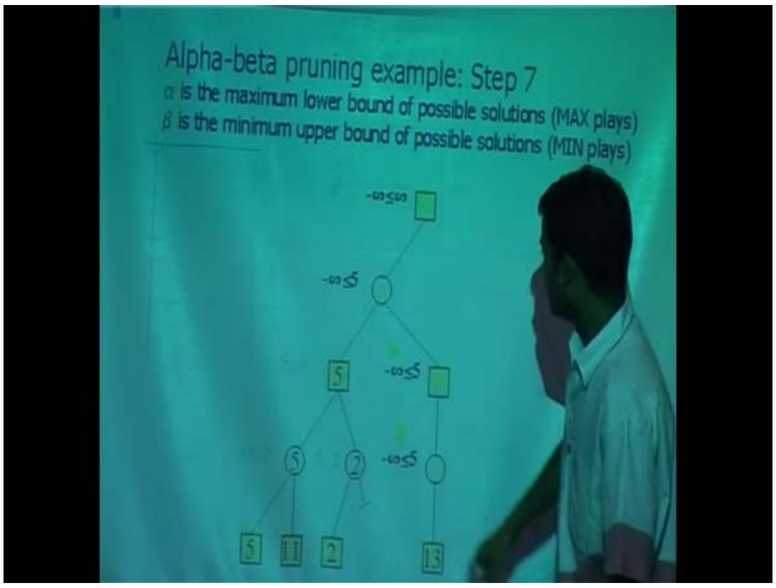
α is the maximum lower bound of possible solutions (MAX plays)

β is the minimum upper bound of possible solutions ( )

-∞≤∞
-∞≤∞
5≤∞
5
5≤2
α>β!
5
11
2

Because it tries to maximizing the things it knows that 5 is 5 comes. So, if it comes to this, no need to he has to maximize this thing and he knows that 5 is there. So, no need so the information of 2 is nothing, because if may 2 comes this is nothing means that information is useless. So, we need not need not use these information you also no file.

(Refer Slide Time: 13:45)



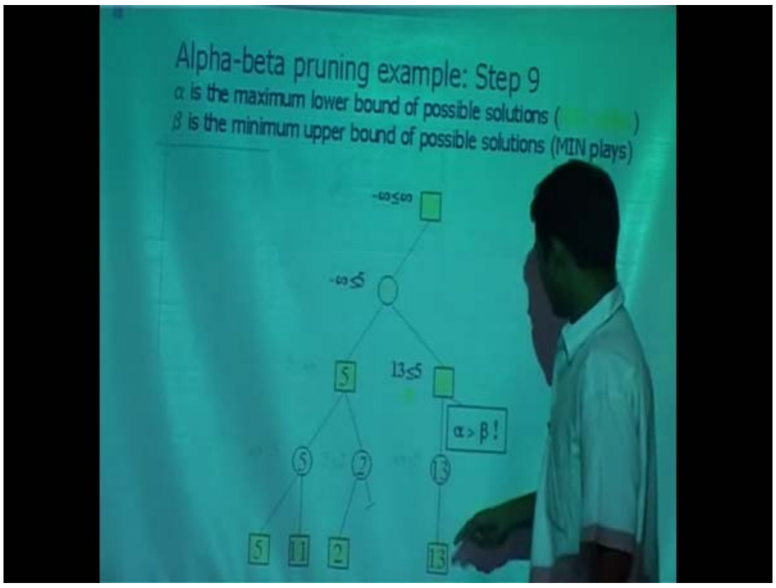
Alpha-beta pruning example: Step 7

α is the maximum lower bound of possible solutions (MAX plays)

β is the minimum upper bound of possible solutions (MIN plays)

So, after that go that these nodes are pruned all nodes all nodes are pruned. Now, here it is 5. So, stored here. Now, we calculate if now here the thing is opposite, because it wants to minimize the node so if any number is greater than 5. Less than 5. Here it is less than. So, here any greater than 5 then it is pruned means less than 5 it is used but, greater than 5 then it is pruned.

(Refer Slide Time: 14:24)



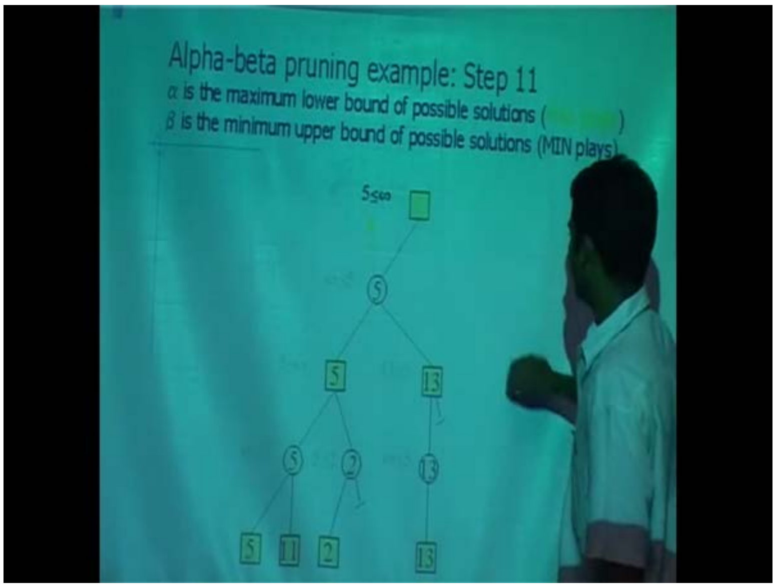
Alpha-beta pruning example: Step 9

α is the maximum lower bound of possible solutions ( )

β is the minimum upper bound of possible solutions (MIN plays)

For example, it is 13 it goes here it comes 13 after 5.

(Refer Slide Time: 14:28)



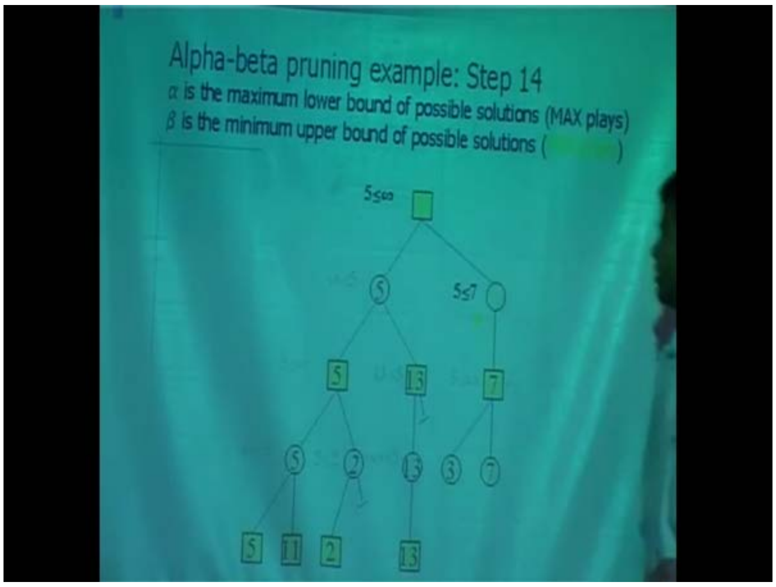
Alpha-beta pruning example: Step 11

α is the maximum lower bound of possible solutions ( )

β is the minimum upper bound of possible solutions (MIN plays)

Then, 5 is prune number for this.

(Refer Slide Time: 14:34)



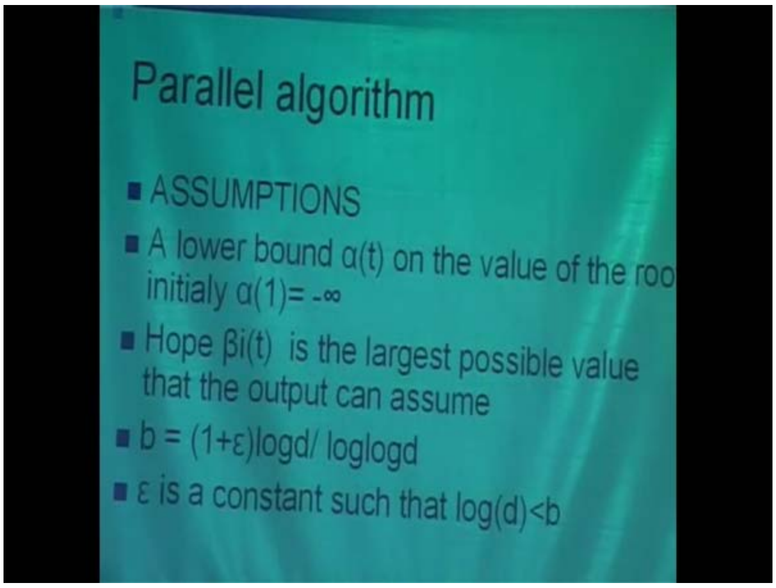
Alpha-beta pruning example: Step 14

α is the maximum lower bound of possible solutions (MAX plays)

β is the minimum upper bound of possible solutions ( )

And again the slide is working. Then I think parallel algorithm is more important.

(Refer Slide Time: 14:38)

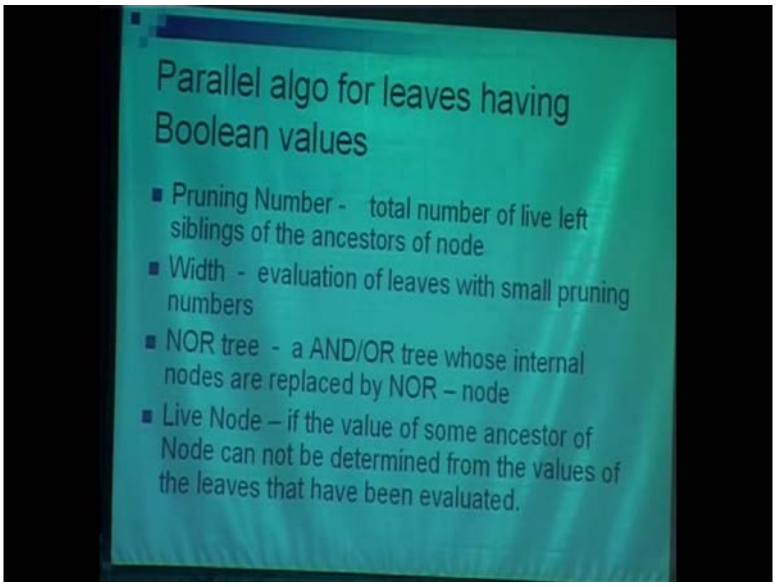


Parallel algorithm

- ASSUMPTIONS
- A lower bound α(t) on the value of the roo initialy α(1)= -∞
- Hope βi(t) is the largest possible value that the output can assume
- b = (1+ε)logd/ loglogd
- ε is a constant such that log(d)<b

So, we are talking about the parallel algorithm here we take few assumptions what is the, we take the lot. One thing I want to tell. What is the problem in means why not any good parallel algorithm comes for temporary.

(Refer Slide Time: 15:23)



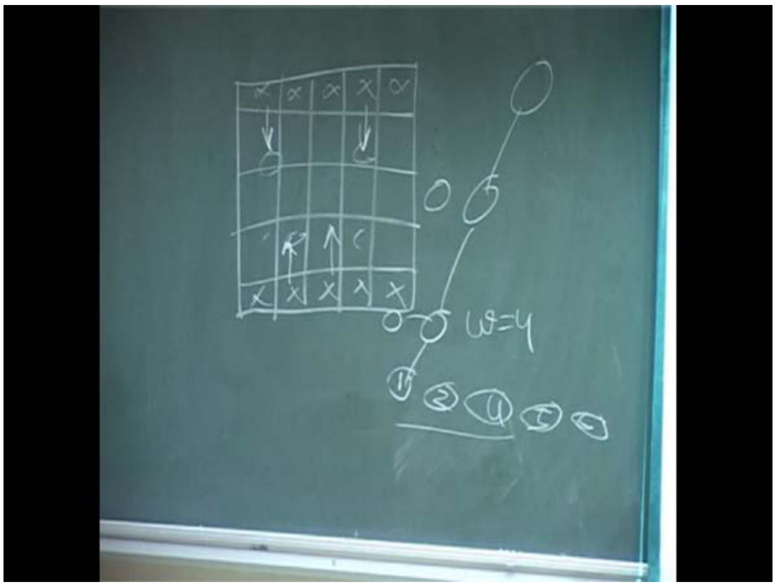
Parallel algo for leaves having Boolean values

- Pruning Number - total number of live left siblings of the ancestors of node
- Width - evaluation of leaves with small pruning numbers
- NOR tree - a AND/OR tree whose internal nodes are replaced by NOR – node
- Live Node – if the value of some ancestor of Node can not be determined from the values of the leaves that have been evaluated.

Because if you have at first there are many algorithm for parallelizing, if you know on the mini-max algorithm, then we divide the game tree in many part in many processor and they calculate but, because we have alpha beta algorithm in sequential therefore, we cannot make parallelize any algorithm, because we need not see, many nodes reproduce all nodes and in parallel it cannot although it is possible but, I am leave for a means although we have also trying to use Alpha-Beta Pruning number.

So, now we are talking about the parallel algorithm for the game trees which has the Boolean values at their leaves nodes. Here we first make some assumption. First one is the pruning number is the number of leaves siblings of any ancestors nodes something like this, if this is a node and here the pruning number of this node is 1.

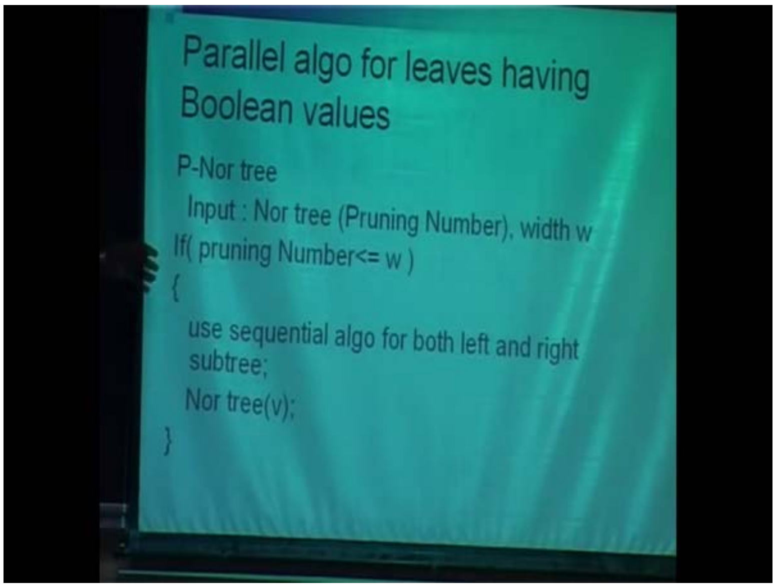
(Refer Slide Time: 16:05)



Then, it means the number of lets sub links of these ancestors any node is also ancestors of itself. So, number of leaves sub nodes of these nodes is the pruning number aim and lead node is that node whose value cannot be determined from the nodes which are evaluated already and the width.

Width is the, if there is some leaves they are and their value is something like this. We define a number W whose value lets 4, and then the node which has the value greater than 4 will be pruned. We only take in our consideration only these nodes which are the width. The algorithm is first for the algorithm, we are taking the inputs at the NOR tree at pruning number.

(Refer Slide Time: 17:00)

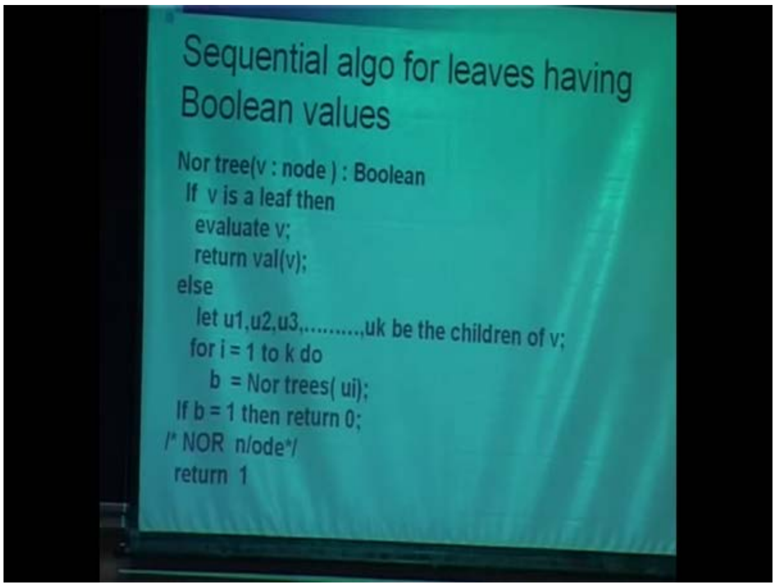


Parallel algo for leaves having Boolean values

```
P-Nor tree
 Input : Nor tree (Pruning Number), width w
If( pruning Number<= w )
{
  use sequential algo for both left and right
  subtree;
  Nor tree(v);
}
```

And with if pruning number is less than W, just put the sequential algorithm to solve any node.

(Refer Slide Time: 17:08)

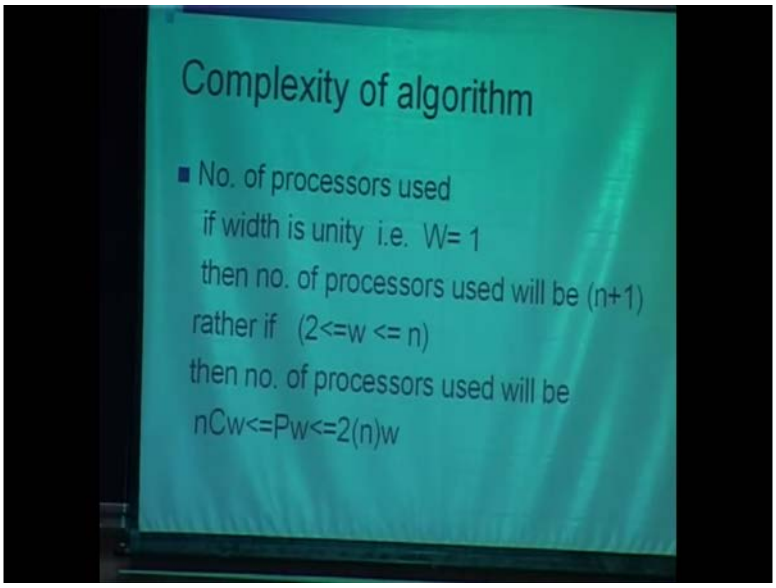


Sequential algo for leaves having Boolean values

```
Nor tree(v : node ) : Boolean
 If  v is a leaf then
   evaluate v;
   return val(v);
else
   let u1,u2,u3,.........,uk be the children of v;
  for i = 1 to k do
     b  = Nor trees( ui);
 If b = 1 then return 0;
/* NOR  n/ode*/
 return  1
```

The sequential algorithm is if v is a leaf, then evaluate v no and if we are not a leaf, then there will be a leaf their children’s u 1, u 2, u 3 take the nor 3 examples, recursively put and the value of the b we can be calculate. Then if we it is 1 return 0 rather it is work.

(Refer Slide Time: 17:32)



Complexity of algorithm

- No. of processors used

  if width is unity i.e. W= 1

  then no. of processors used will be (n+1)

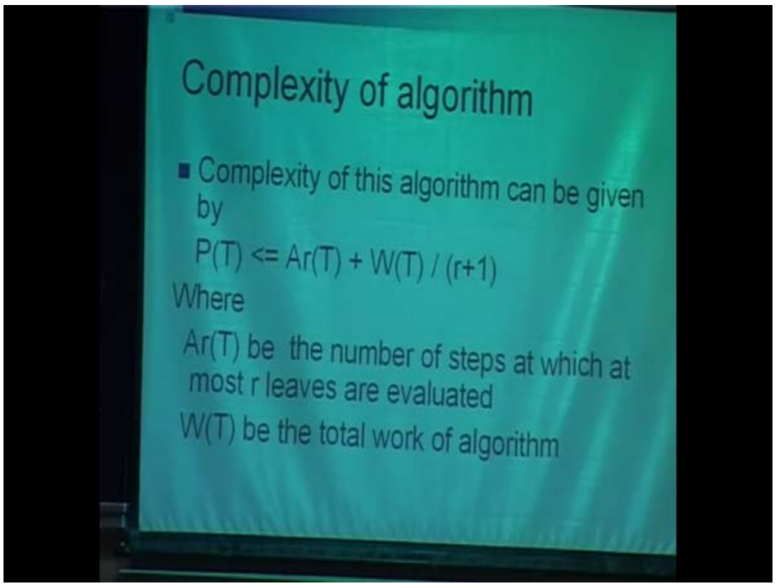
  rather if (2<=w <= n)

  then no. of processors used will be

  nCw<=Pw<=2(n)w

So, what is the number of processor used in this algorithm? If it is 1, then the number of processor is n plus 1. This is very less number of processors, if width is greater than 2 then the number of processors lie between n C w 2 times n to the power w this is not two n w this is 2 times n to the power w.

(Refer Slide Time: 18:00)



Complexity of algorithm

- Complexity of this algorithm can be given by

  P(T) <= Ar(T) + W(T) / (r+1)

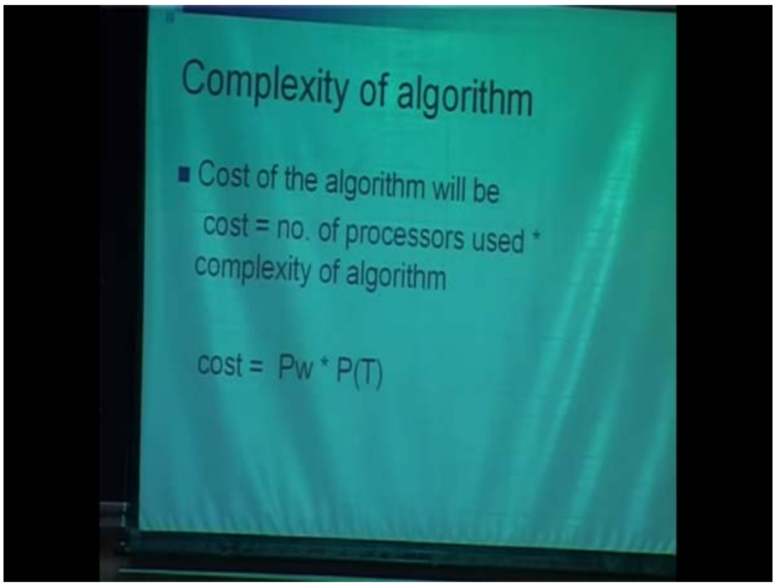
Where

Ar(T) be the number of steps at which at most r leaves are evaluated

W(T) be the total work of algorithm

The complexity we will dominated by the number of parallel steps in this algorithm. And number of parallelism the A r, A r is the number of parallel step in which you at most r leaves are evaluated in each step. And W T is the total work done. So, this term will provide us the number of parallel step in which a minimum or leaves are evaluated. So, this is the sum of parallel e steps that is why we are saying that to these complexities dominated by the parallel steps. So, what is the cost?

(Refer Slide Time: 18:39)



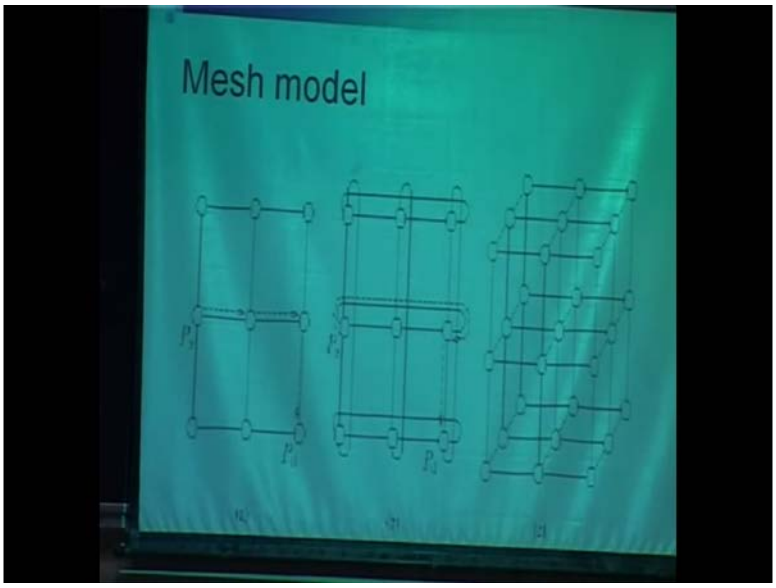
Complexity of algorithm

- Cost of the algorithm will be
  cost = no. of processors used * complexity of algorithm

  cost = Pw * P(T)

Cost will be the just complexity into the processor.

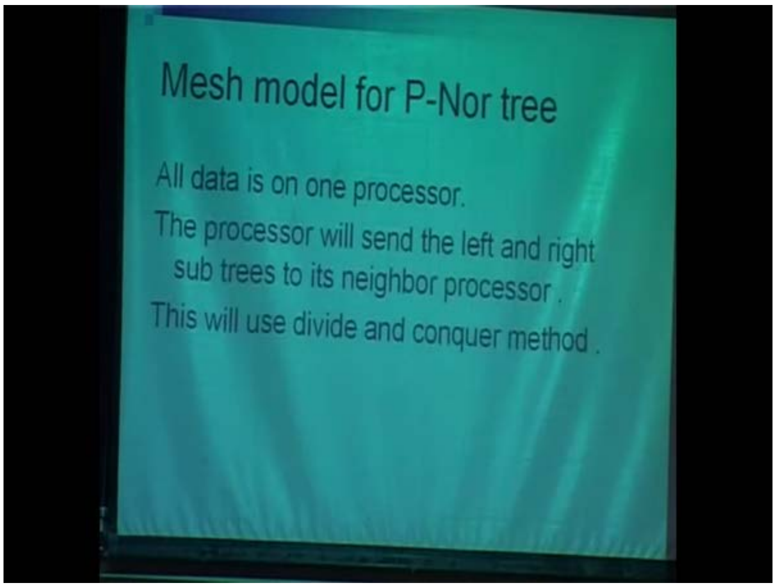
(Refer Slide Time: 18:43)



Mesh model

What are the network models is more suitable for this algorithm? We are in thi algorithm we are dividing our whole problem into 2 sub problem, left sub tree and right sub tree. So, we can communicate this part of information to neighbor processor.

(Refer Slide Time: 19:03)



Mesh model for P-Nor tree

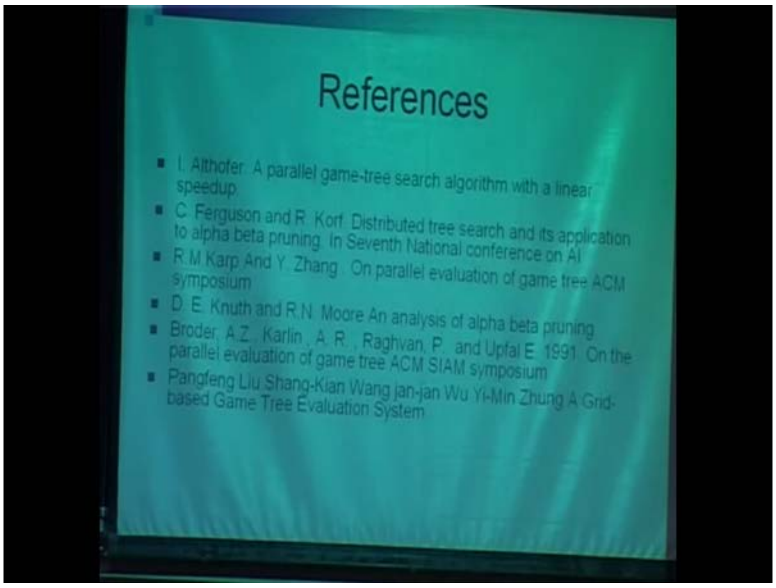
All data is on one processor.

The processor will send the left and right sub trees to its neighbor processor .

This will use divide and conquer method .

So, put all the data in one processor, it will divide it into two sub parts, left sub tree and right sub tree, and their neighbor processor will decide to do further decomposition of the problem. After all the processor has the leaves, they were evaluating the function and just pass it the result to their neighbors.

(Refer Slide Time: 19:26)



References

- I. Althofer. A parallel game-tree search algorithm with a linear speedup.
- C. Ferguson and R. Korf. Distributed tree search and its application to alpha beta pruning. In Seventh National conference on AI
- R M Karp And Y. Zhang . On parallel evaluation of game tree ACM symposium
- D. E. Knuth and R.N. Moore An analysis of alpha beta pruning
- Broder, A.Z. , Karlin , A. R. , Raghvan, P. and Upfal E. 1991. On the parallel evaluation of game tree ACM SIAM symposium
- Pangfeng Liu Shang-Kian Wang jan-jan Wu Yi-Min Zhung A Grid-based Game Tree Evaluation System

These are the references. Over the one reference Sknuths paper you seen Knuths paper.

Student: Knuths paper (( )) knuth.

One alpha beta parallel alpha beta.

Actually we are, we sustain alpha beta that algorithm one each s s a and one is and but one thing there is means not any paper provide. What is the network module for solving this thing not and we try to do in which network, because we are solve it and we get, we calculate and complexity is not improve? But, because we made general situation means we.

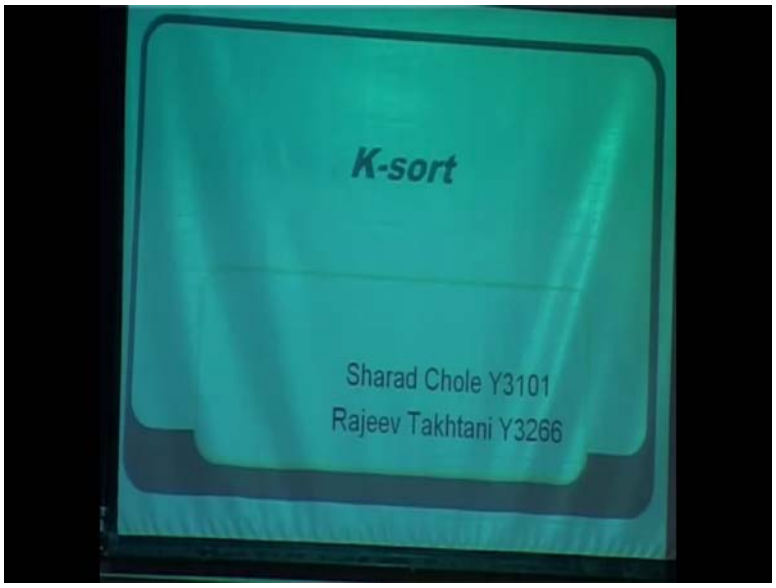
But the complexity will be improved, if we are using the master and slate process, then complexity will be definitely improved but, it is not may it is under research because in one paper is.

There are where many means for example, where is young whether extended this. We try to we not try means we included this thing as a case in our algorithm. So, in our algorithm we are getting some cases and include all good algorithms.

Any question?

Tree model for the network? No in tree model actually we thought but, there is very less connection, if we need this. We need more connections as we act to keep the things, all the nodes with us also one thing we also need to means evaluation of this node. We have to tell that means significance of ancestors we have to tell. So, tree model is means we cannot do in tree model this can be becomes as sequential and there is problem we try with tree model also we tried in happen to tree model and.

(Refer Slide Time: 22:20)



K-sort

Sharad Chole Y3101

Rajeev Takhtani Y3266

You are going to tell us?

Student: Yes sir,

What is the difference between your model and their model?

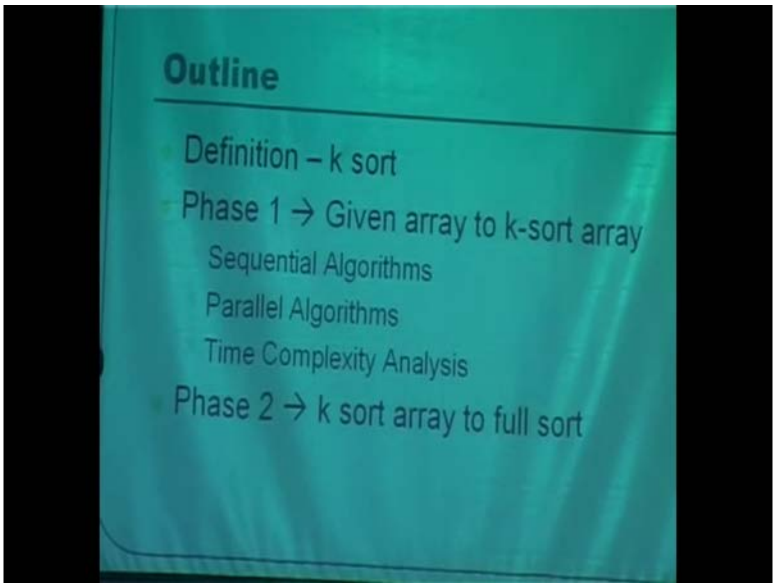
Student: We will compare the complexity and I will do that.

And your here opponent,

Student: But sir, we have compared that we will mention their complexity also.

So, good morning everybody, like the problem we turn data and term paper was k-sort. So, the outlay of our representation will be keeps us define the problem.

(Refer Slide Time: 22:50)



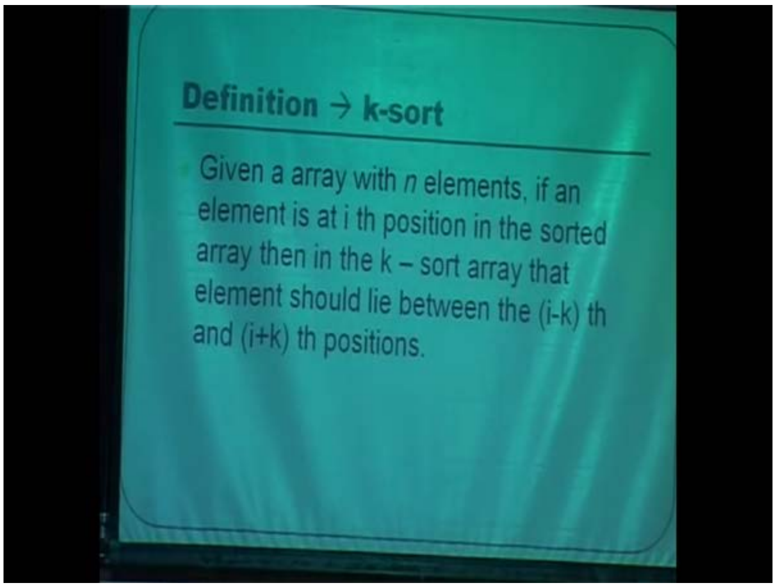
Outline

- Definition – k sort
- Phase 1 → Given array to k-sort array
  - Sequential Algorithms
  - Parallel Algorithms
  - Time Complexity Analysis
- Phase 2 → k sort array to full sort

Then, in phase one we will do like given any array. We try to get a k-sort array will suggest some sequential algorithm. So, that parallel algorithms and will do the time complexity analysis along with the problems and in a phase two will also tell you one algorithm to do k-sort to full sort.

Student: Ok sir, I will do that but, I will need that.

(Refer Slide Time: 23:08)



Definition → k-sort

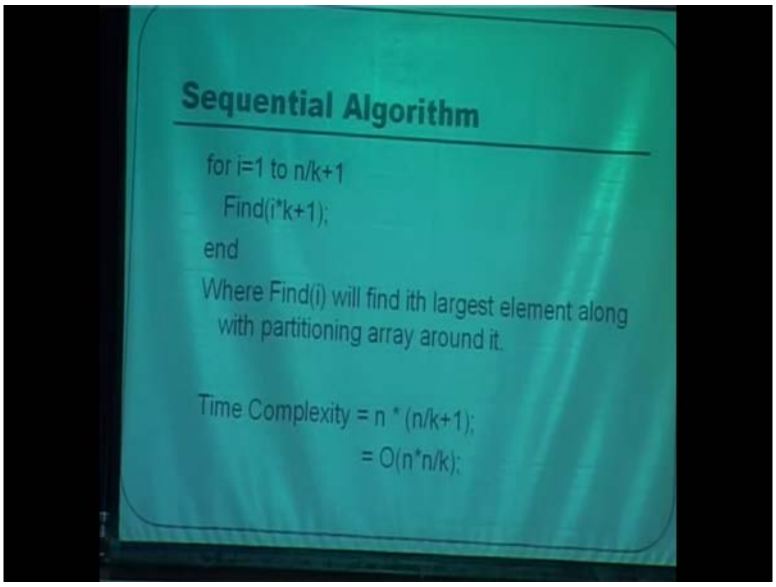
- Given a array with *n* elements, if an element is at i th position in the sorted array then in the k – sort array that element should lie between the (i-k) th and (i+k) th positions.

So, the definition of k-sort is given array with n elements.

Because, this is your platform where you know now we say previous speaker you observe the problem that page there is no synchronize both the person was to right. So, this is the happy view to.

Ok sir, with definition K-sort is given array with n elements. If an element is at the i minus position in the sorted array. In the K-sort array, it should lie between the i minus k minus and i plus k.

(Refer Slide Time: 23:48)



Sequential Algorithm

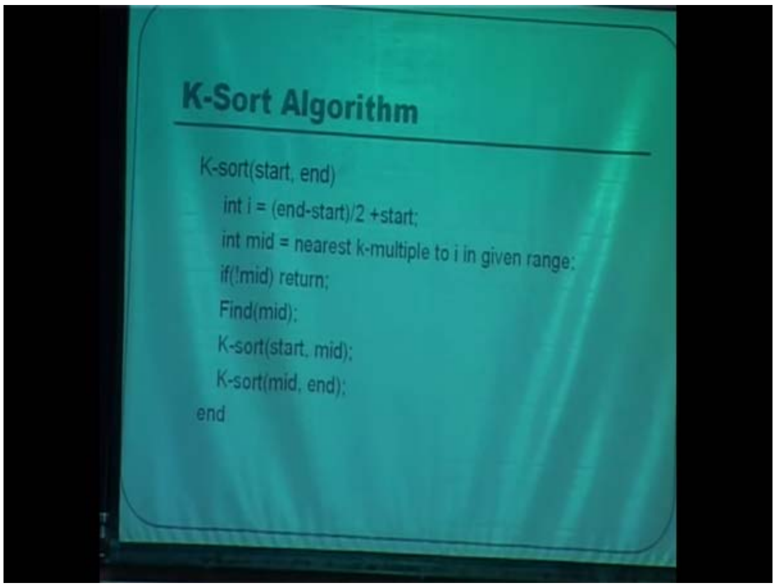
```
for i=1 to n/k+1
  Find(i*k+1);
end
Where Find(i) will find ith largest element along
  with partitioning array around it.

Time Complexity = n * (n/k+1);
                = O(n*n/k);
```

So, like the first sequential algorithm that will describe is like a not a very tough one but, like it says for I ranging between this. We find like what we do is like given any array we find the elements. k minus 2 k minus 3 k minus k minus multiples of the and then partition along with that. Like sequentially down through the array, and like the elements less than k are sorted kept over here and along with that.

So, the find layer what it does, it will find it is i minus largest element along with partitioning array around it. So, the time complexity for this algorithm will be n into n upon k-th plus 1 n into k-th plus 1 is the loop it goes along that and n time to through the array.

(Refer Slide Time: 24:28)



K-Sort Algorithm

```
K-sort(start, end)
    int i = (end-start)/2 +start;
    int mid = nearest k-multiple to i in given range;
    if(!mid) return;
    Find(mid);
    K-sort(start, mid);
    K-sort(mid, end);
end
```

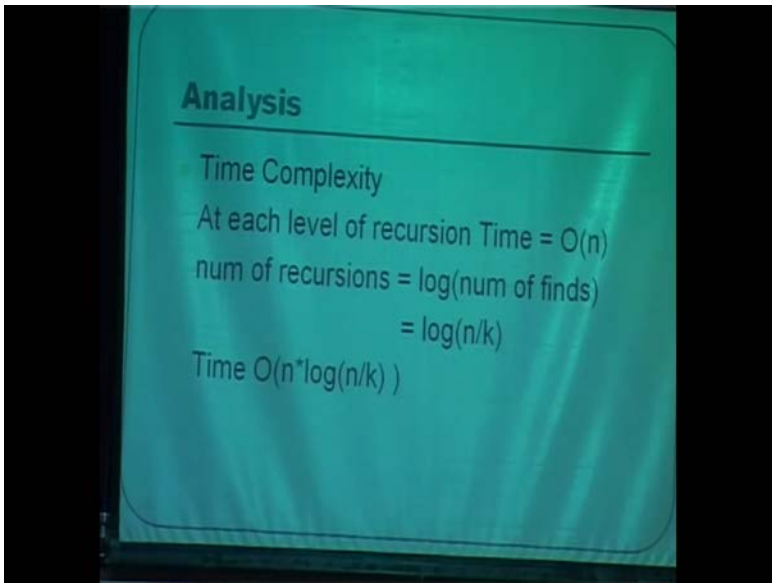
So, other algorithm will be the k-sort start end in this. What we do is? Will then we find like from the middle are k minus largest multiple will near to k-th largest multiple. And we partition along that and give the algorithm recursively, from start to mid and mid to end.

So, what you are doing? You are finding algorithm value.End may higher. And then you are calling a.

Student: No from near the middle element we find the k-th multiple.

Will do that mid is equal to the nearest k-th multiple of all near to the middle element. And if, there is normal net that means like it is less than K, we delete that particular array as it is otherwise we call this particular.

(Refer Slide Time: 25:20)



Analysis

Time Complexity

At each level of recursion Time = O(n)

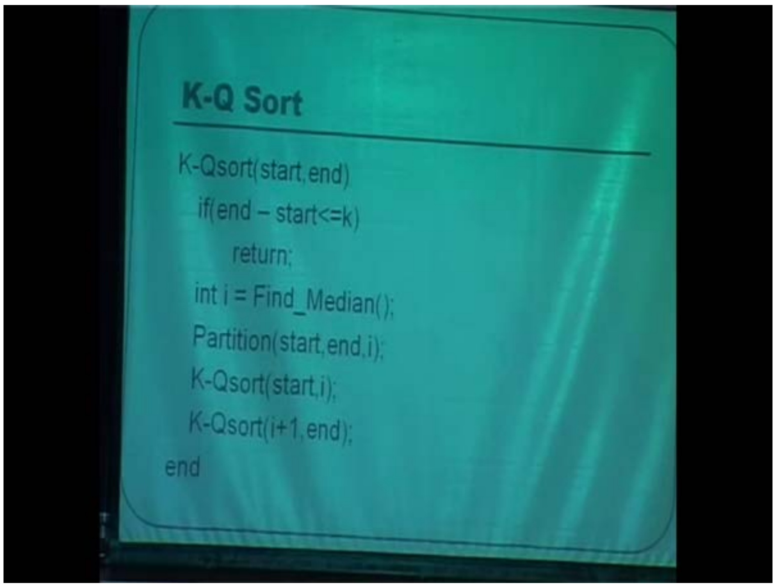
num of recursions = log(num of finds)

= log(n/k)

Time O(n*log(n/k))

So, the time complexity for this will may be reduce, and it will now become like n into log n by k. Because a number of recursions here it becomes log n by 2 because we call it.

(Refer Slide Time: 25:32)



K-Q Sort

```
K-Qsort(start,end)
  if(end – start<=k)
      return;
  int i = Find_Median();
  Partition(start,end,i);
  K-Qsort(start,i);
  K-Qsort(i+1,end);
end
```

And then, there is like K-Q sort. What we do is like this is a variant of quick-sort, in which we do is like whenever we have like a partition that has a length less than k believe that as it is. So, it goes like if end minus start is less than K that is like partition has a length less than K with the leave it. Otherwise we find the median you use median sort to find out this and hence it is recursively calculated the K-Q sort it is liquidity come to this, so in this case.

The result is?

Student: Yes sir,

There are some positions, where he going to his own position, right there is some elements because you are partitioning.

Student: Yes,

So, partitioning element will get the own position. So, suppose you have done instead of x number after you have done say 5 steps. So, 5 elements at least 5 element get his own position.

Student: Yes sir,

Agreed that means that i of this 5 element, they are not satisfying your K sorting finally.

Student: No sir, they will be lying in a position then explain the.

Position say is own position yes or no.

Student: Yes,

So, the 5 little so you have given more competition.

Student: Yes but,

You have to be sure. Sir there is no have to be sure to exactly to the array K-sorted either partition length is done.

Student: I am trying to justify myself.

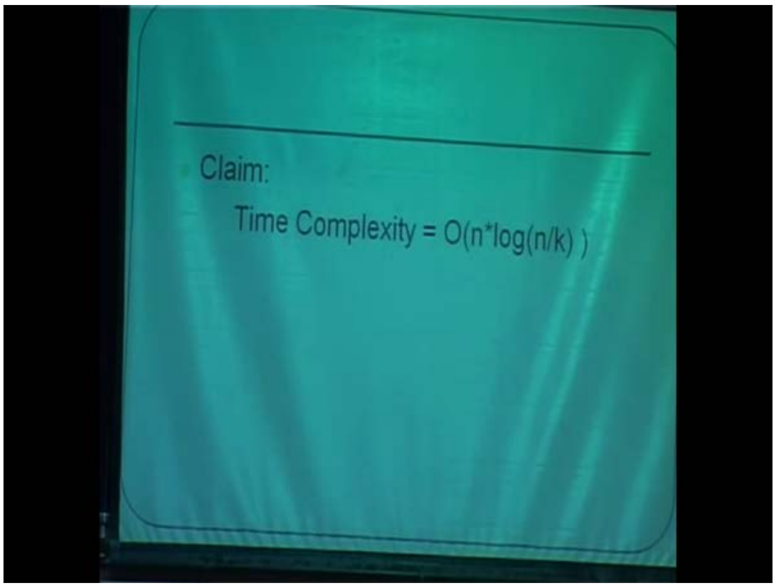
But, in this stage also like that complexity is an order.

Suppose, instead of order n time or n minus 1 time

For partition suppose, I do n minus 5, then this data is equal to be i-th plus or minus 1, then you called be sure about next partition.

Sir, we have to be sure from the first wide partition element last partition element, then we will be sure that partition can exactly in the same position.

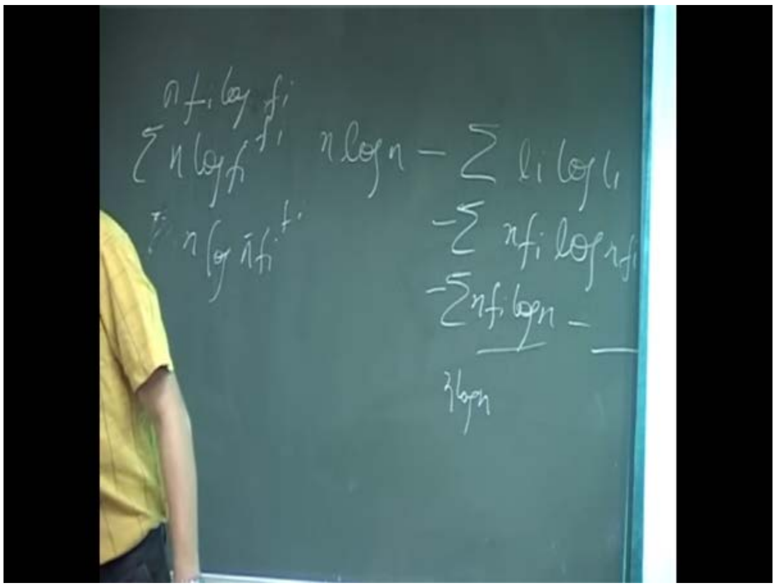
(Refer Slide Time: 27:52)



- Claim:
  - Time Complexity = O(n*log(n/k))

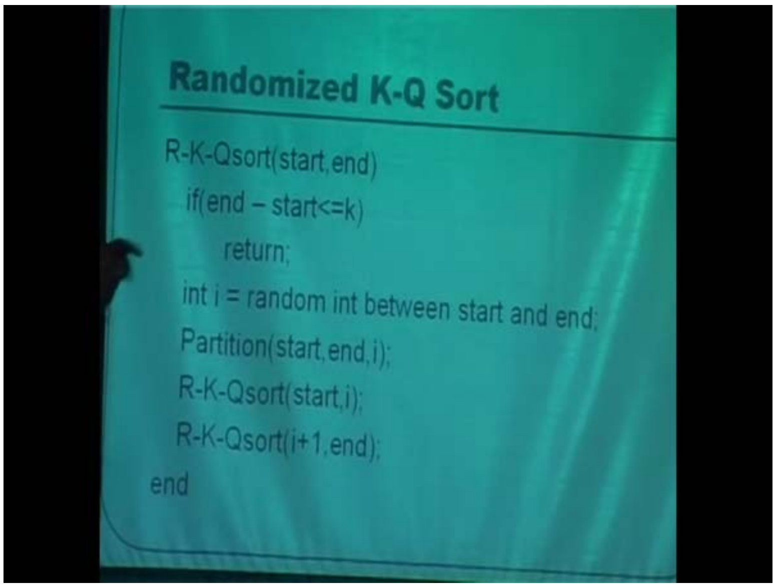
So, if this particular case also you claim the time complexity comes like this only. Because like apart from the quick sort, we do a less amount Sir, need I explain this term? This is ok? That will I want to say. Like it will be like, n log n minus the sum of that will not do.

(Refer Slide Time: 28:14)



That will be like summation l i log i where the l represents the partition length that we are dropped out. So, it will come out like minus if we say like l i to be represented as n into f I, where f i is the fraction then we will write to like this mid like will keep it like n f i log of n n minus this part. So, this will come out to like n log n and this will come as I write it here f i n n i n f i log n f i. log f i. log f i, thanks. So, it will come as log n f i n into log f i to the par f i summation of this. So, this will come as n into log phi of f i f i. I will sure that in the output…

(Refer Slide Time: 29:13)

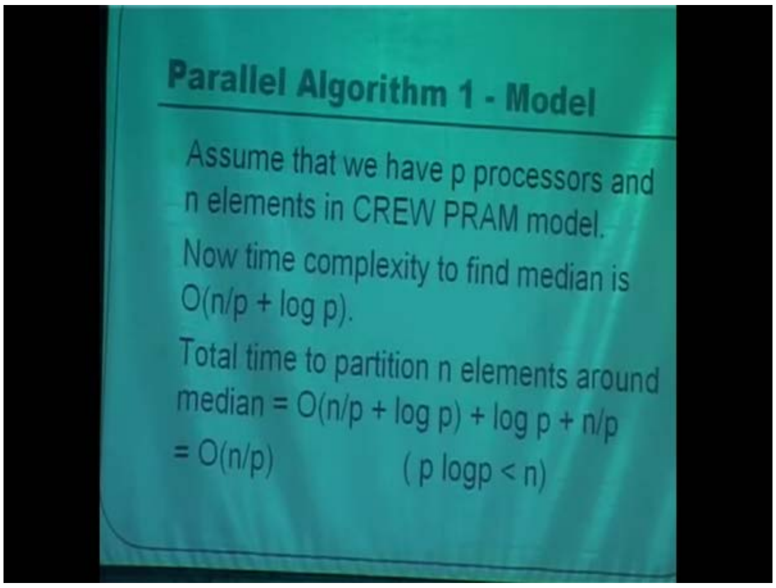


Randomized K-Q Sort

```
R-K-Qsort(start,end)
  if(end – start<=k)
      return;
  int i = random int between start and end;
  Partition(start,end,i);
  R-K-Qsort(start,i);
  R-K-Qsort(i+1,end);
end
```

And then, we can have like randomized version of this particular thing in which we start with the integer randomly chosen between the start and end it is like. So, using these sequential algorithms, we try to suggest the parallel Algorithms.

(Refer Slide Time: 29:21)



Parallel Algorithm 1 - Model

Assume that we have p processors and n elements in CREW PRAM model.

Now time complexity to find median is $O(n/p + \log p)$.

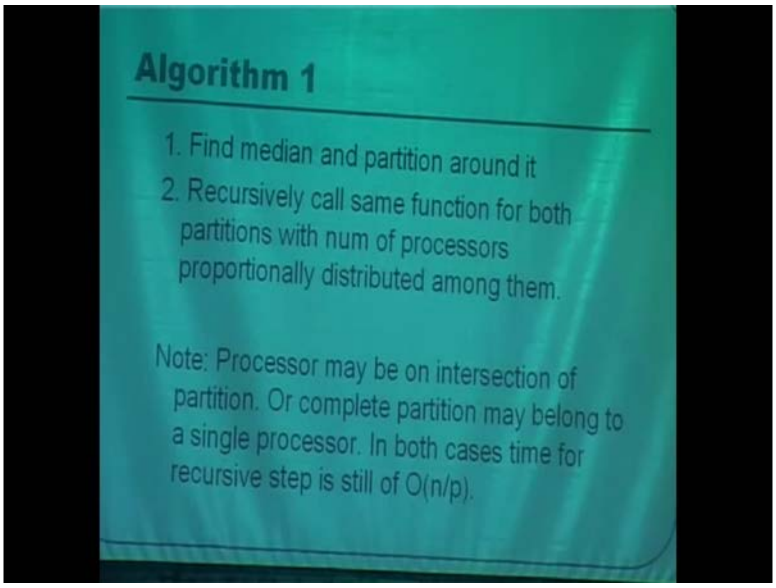
Total time to partition n elements around median $= O(n/p + \log p) + \log p + n/p$
$= O(n/p)$ $(p \log p < n)$

First algorithm 1, we are going to propose two algorithms for parallel k-sort. In this algorithm, we assume that we have 3 number of processes and n number of elements in c r c w c r e w pram model. Now, time complexity to define them median in n elements using p processor is ordered n by p log p, this can be explained as each processor will have n by p elements and it can find its own median means own median in n by order n by p time and using p processer and p median that we found.

We can find them median of that in log p time this is the selection problem. And after finding them medians of medians, we partition around this after finding medians of medians, we call it as the median of the n elements. And the partition will be at least 25 percent to 75 percent, what place.

Because you are, Yes, of course. Now, total time to partition n elements around median is equal to first finding the median, and then partitioning will be done is comparing n by p processer comparing each processer will compare its n by p elements to the median, and then there will be summation of all these, each processor we compare the n by p elements to the median and it will know how many less element are, how many greater element it will have. And then the linear means linear doing the summation of how many lesser element it will have. Each processor can find its index in the partition direct one, doing the summation will log p time. So, it will require order end total time and given that p log p is less than n, that is the condition.

(Refer Slide Time: 31:30)



Algorithm 1

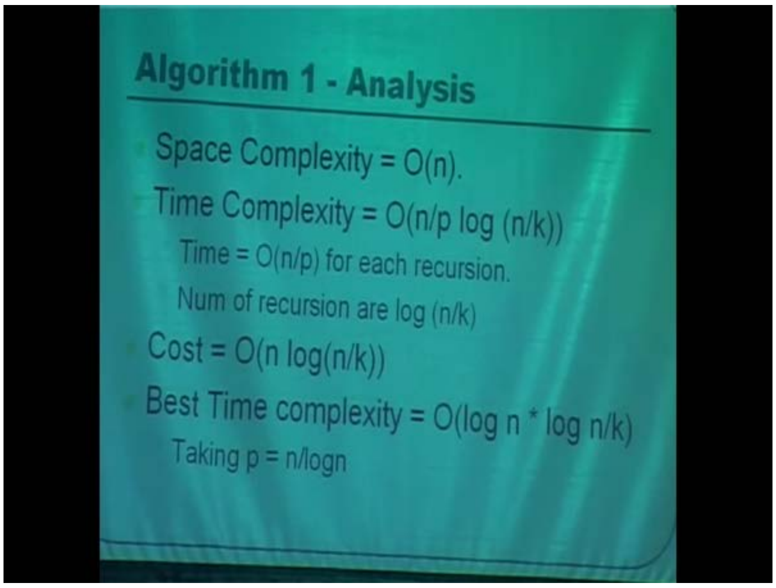
1. Find median and partition around it
2. Recursively call same function for both partitions with num of processors proportionally distributed among them.

Note: Processor may be on intersection of partition. Or complete partition may belong to a single processor. In both cases time for recursive step is still of O(n/p).

Now, the algorithm is quite simple, find the median and partition around it and recursively call the same function for both partition with number of process, number of processors distributed among that means if the processer is before the partition, be the processer will be belong refer to partition processer on the intersection it will have to do the work of both this intersection of that partition and the other partition.

And if the case is that the processor contains the n by p elements, that processer have contains the whole partition, then also the time required for finding the median and sorting around, it will be both the order n by p because it is linear.

(Refer Slide Time: 32:15)



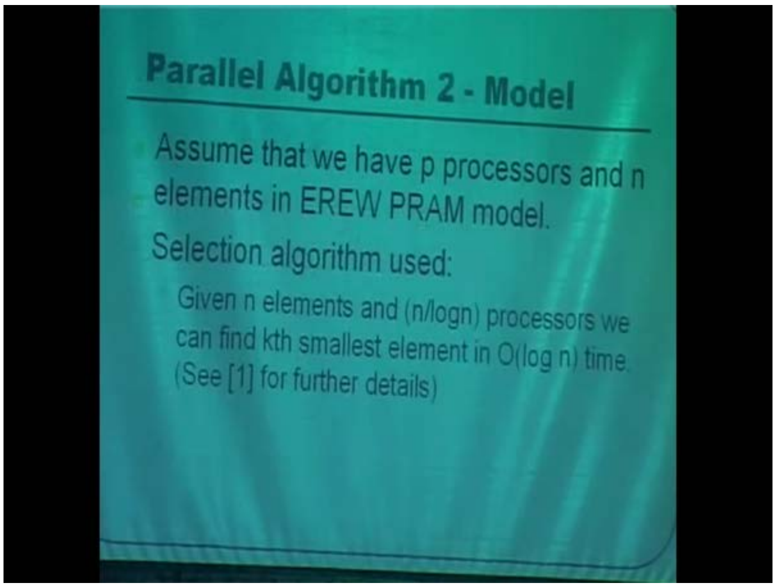
Algorithm 1 - Analysis

- Space Complexity = O(n).
- Time Complexity = O(n/p log (n/k))
  - Time = O(n/p) for each recursion.
  - Num of recursion are log (n/k)
- Cost = O(n log(n/k))
- Best Time complexity = O(log n * log n/k)
  - Taking p = n/logn

From space complexity order n that is or time complexity will be order n by p log n by k, because each recursive step will require order n by p step order n by p time. As we discussed earlier and number of recursion will be order log n by k because each time we will have at least 20 percent 75 percent partition and we will do that till the partition side is less than k. So, time will be order n by k log. So, that, the number of recursion will be order of log n by k log cost will be.

So, the time complexity is same as the sorry the time complexity is minimum n p is value of p is largest and the largest p value can be taken as n by log n. And in that case the best time complexity, it comes out to be order log n into log n by k which is same as algo, also proposed by Satish. In this case and cost is a multiple time complexity by p. This is come as n into log n by k order and it is a better factor log n from the algo proposal solution and we are using the same c r e w model.

(Refer Slide Time: 33:33)



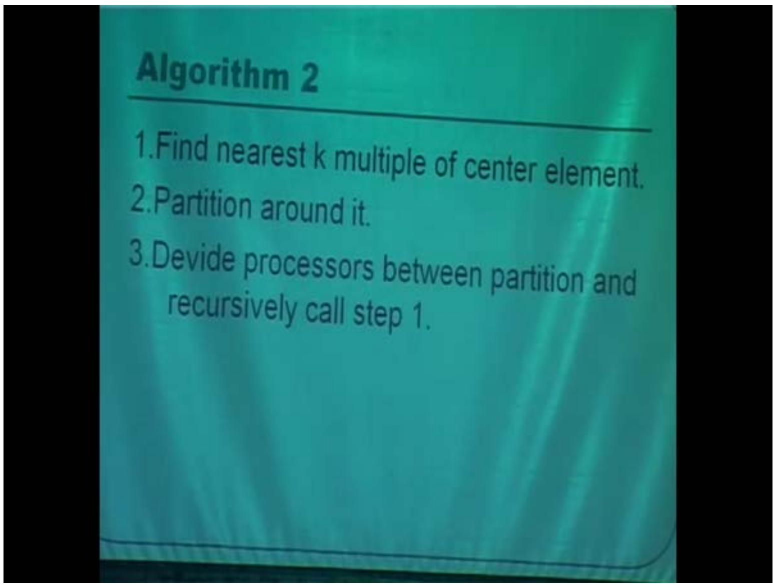
Parallel Algorithm 2 - Model

- Assume that we have p processors and n elements in EREW PRAM model.
- Selection algorithm used:
  - Given n elements and (n/logn) processors we can find kth smallest element in O(log n) time. (See [1] for further details)

Now, let us consider the case when we have e r e w p r a m model. Go back go back. What you are telling in the best time complexity is becoming log n to log n value. This time complexity is the same sir, of same order. But, cost is the less by factor n by log n number of processors and there are union so that, log n factor comes.

Now, in case instead of C R E W model, we have E R E W model and for this case, we will use selection algorithm, this given in paper reference number 1 that algorithm suggest that given n elements are n by log n number of processors. We can find k-th smallest element in order log n time, these consent numbers of processer is there.

(Refer Slide Time: 34:28)

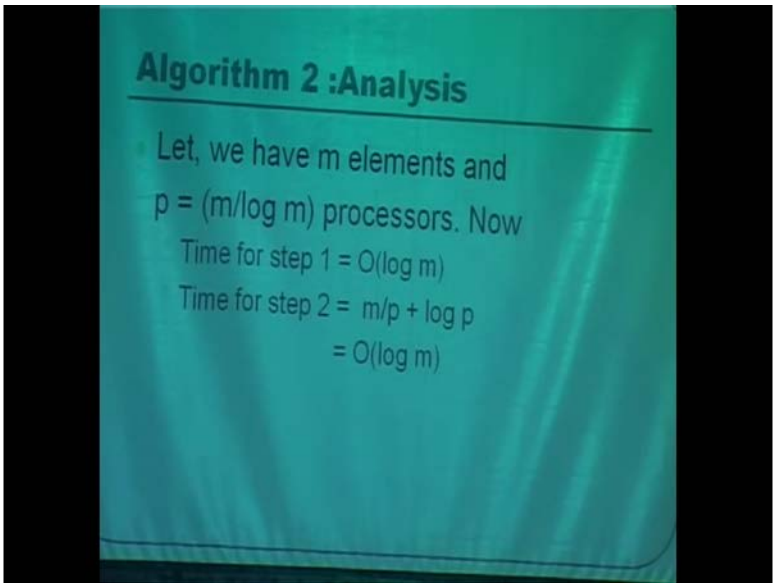


Algorithm 2

1. Find nearest k multiple of center element.
2. Partition around it.
3. Devide processors between partition and recursively call step 1.

Now, algorithm is quite simple. Find the nearest k multiple of the center element, partition around it, divide processor between partition and recursion for the step 1, it is approximately simpler instead of median, we are using k-th multiple of nearest surround nearest case of multiple center element.

(Refer Slide Time: 34:43)

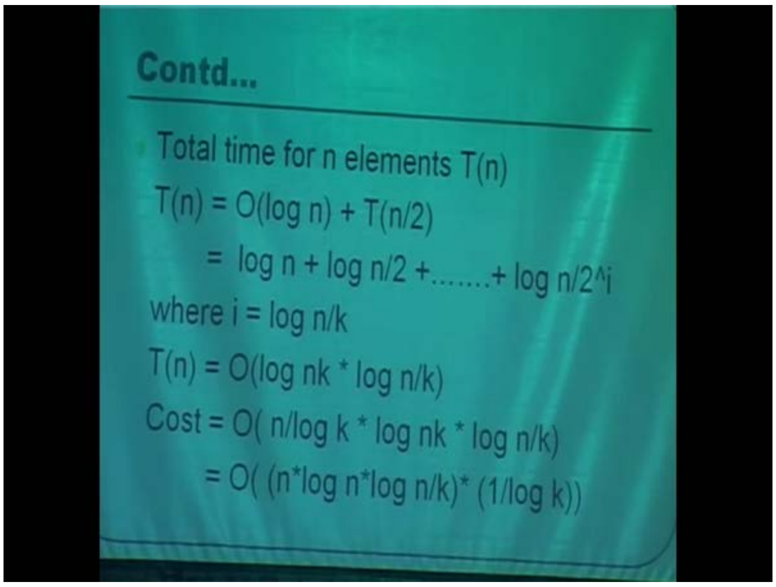


Algorithm 2 :Analysis

- Let, we have m elements and $p = (m/\log m)$ processors. Now
  Time for step 1 = $O(\log m)$
  Time for step 2 = $m/p + \log p$
  $= O(\log m)$

Now, analysis in case of we have m elements and p is equal to m by log m processors. Now, time for step 1 will require that fine, that nearest k-th multiple of center element will be require order log n time. Using n by log n processor, using n by log n is given p is equal to m by log m and time for step 2 is equal to sorry similarly, other explain earlier order m by p here partitioning round the it will can n by p plus log m log p, which is a still order log n so the hold this is for each recursion step the time will be of order number of elements.

(Refer Slide Time: 35:31)



Contd...

- Total time for n elements T(n)

$T(n) = O(\log n) + T(n/2)$

$= \log n + \log n/2 + \ldots\ldots. + \log n/2^i$

where $i = \log n/k$

$T(n) = O(\log nk * \log n/k)$

$\text{Cost} = O(\, n/\log k * \log nk * \log n/k)$

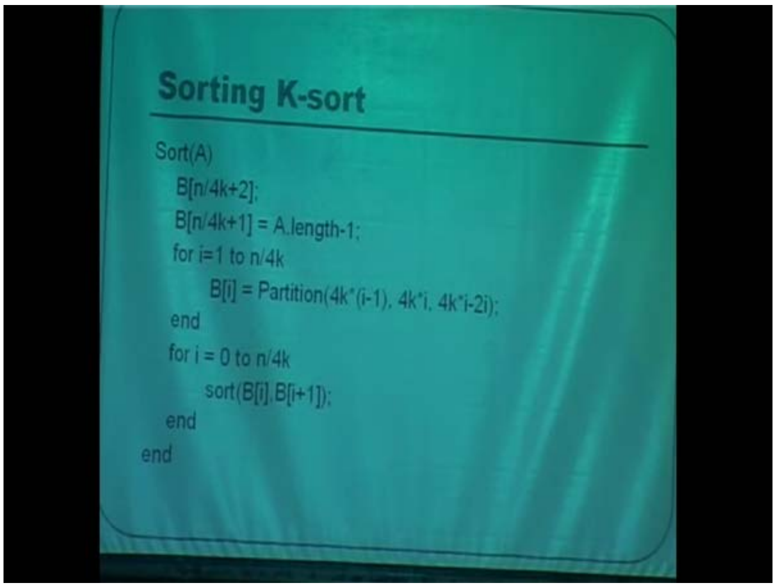
$= O(\, (n*\log n*\log n/k)* (1/\log k))$

So, total time for n elements t of n is equal to t order log n plus t n by 2. So, this is log n plus log n by 2 till log n by 2, n by 2 raise to i where i is log n by k that means, we leave the partition when there partition length is less than k. If we do this summation, it will times comes all comes out to be order of log n k into log n by k, which is still log n by 2 was k value is gone when this time complexity this of same order is they propose and the cost comes out to be n by log.

So, the last thing is maximum number of processer which we needed, for using that selection algorithm, we said the maximum number of processor needed, for when the partition size is smallest that is k and number of partitions will be n by k. So, n by n by k partitions and into for each partitions k by log k of number of processor will be needed for using selection algorithm. So, total number of maximum number of processer will come out be n by y log k.

So, the cost will be n by k log k into time complexity. And it is still factor of 1 by log k means it is fill better than factor of log k what they propose. And even we are using this e r e w model, they were using same that is. So, these 2 are algorithm that we are going to end up next, sorting k-sorted array.

(Refer Slide Time: 37:11)

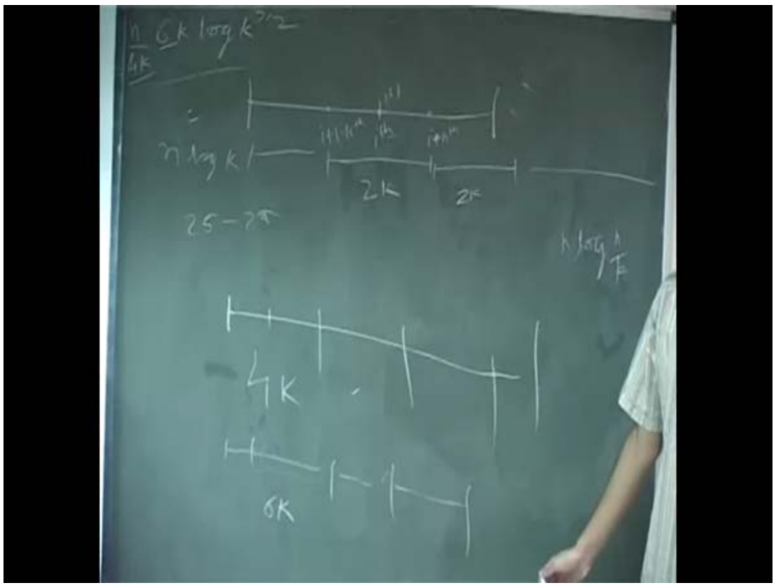


Sorting K-sort

```
Sort(A)
    B[n/4k+2];
    B[n/4k+1] = A.length-1;
    for i=1 to n/4k
        B[i] = Partition(4k*(i-1), 4k*i, 4k*i-2i);
    end
    for i = 0 to n/4k
        sort(B[i],B[i+1]);
    end
end
```

Now, k-sorted array has a property that, suppose, this sorted array invent is that i-th position in k-sorted array.

(Refer Slide Time: 37:26)



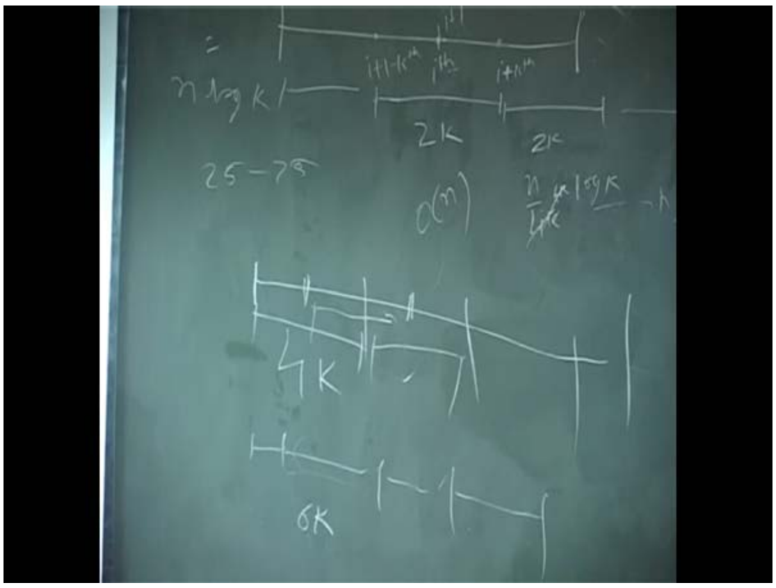
It can be at max at i plus k-th position. Now, element greater than i that is i plus 1 that can be less than, can be at max at i plus 1 minus k-th position. So, after distance 2 k from any elements, all the elements remaining here are less than these elements. Similarly, after discuss 2 k from any elements; all the elements remaining here will be greater than that element. So, what we need off to do only is the partition, choose any element. What we need to do is, given an array partition between as the size of 4 k, not partition just take it as divided size of 4 k elements. Take any elements and partition around it, mostly choose as center element for if you choose a center element, then the partition will be at max 25 to 75 and after partition, it you will have the array like this, and each array will behave the sorted because this after parting, this is the largest investment, this is the smaller investment and this basically, we get this. And this partition will be less than all the other partition. Then sort among the partition so, maximum size of sorting partition will come sort to this 6 k.

So, the total time order is a so time require for that is 6 k log, 6 log k and a total number of partition can be n by 4 k .So, order is a m n log k, and in sequential algorithm the for k-sorting order comes to be end log n by k and summing this it is approximate means it is order n log n for sorting a total array. But, the certain factors will come in this factor. This particular factor, that is three-second that will come here and we cannot cancel k. We do not know exactly how many factor come here doing this parallel is quite use of give this process four k elements and.

Would you write this one?

Student: Yes sir, got there is second algorithm.

(Refer Slide Time: 40:15)



Instead of doing partition and then sorting just sort for each 4 k elements and for first last two k elements of this partition to first two elements of next partition merge them.

Student: Last 2 k elements [FL] algorithm.

Now, this is an time complexity n by 4k n by four k n/4k into log 4 k log 4 k four k log k and this is n log k so sorting can be. Just the extra steps required are this merging steps that we are order that, What is the time complexity coming now? This is a total parallel complexity given any array to get to the final which is algorithm. How?

Student: This is sir, for k-sort array two sorted.

Given stage, This last algorithm is for given any sorted and case of parallel we assume that.There time of this and we called earlier algorithm, it was.

Student: What sir?

Give it 2 k sorted arrays, I want to merge it to get 1 case of sorted. That, they are did it. They absolutely did it using parallel numerator. Merging, Yes,

But we are doing,

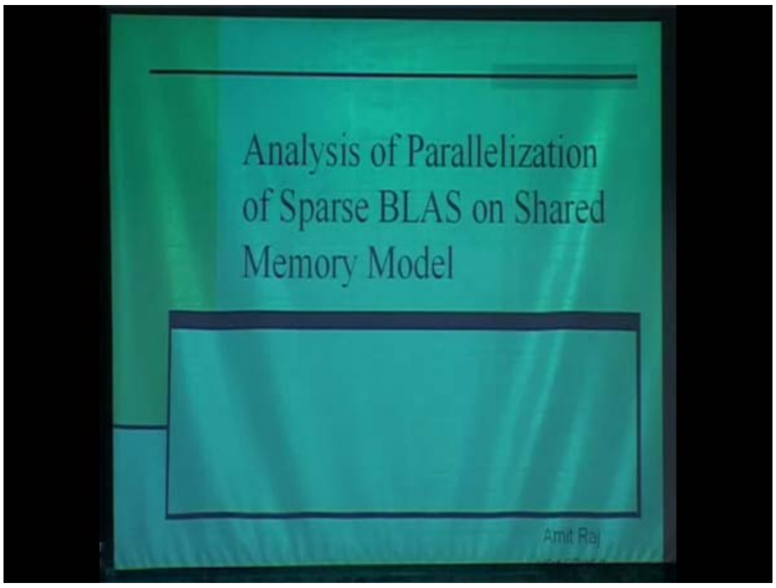
Merging is there sorting is there and get a case.

That that mention in,

Any questions any questions?

Just sitting ideal you ask some question.

(Refer Slide Time: 42:05)



Analysis of Parallelization of Sparse BLAS on Shared Memory Model

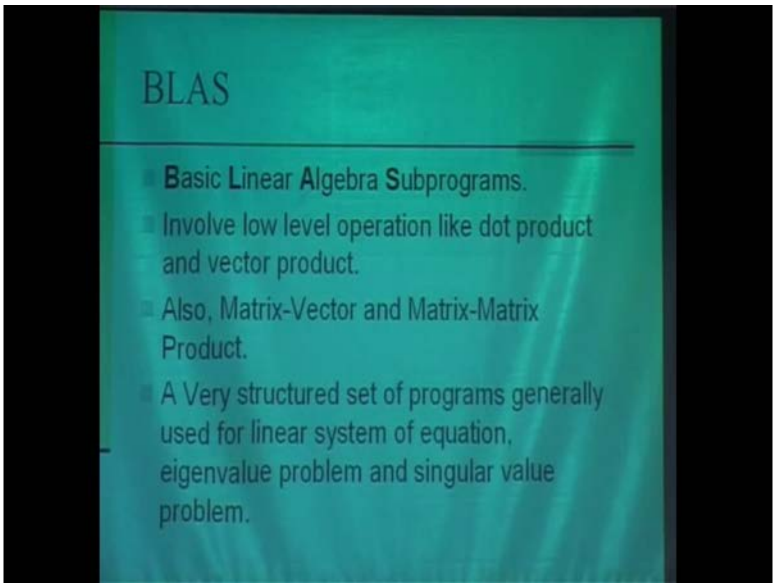
Amit Raj

My name is Amit Raj, My roll number is y2157050 and slow.

Roll number again.

y2157050. And my topic is I mean parallelization of this sparse Blas on shared memory model. And I am not only I mean parallelizing but, analyzing I mean different techniques how to I mean of how to get an optimal parallelization of sparse Blas.

(Refer Slide Time: 42:35)



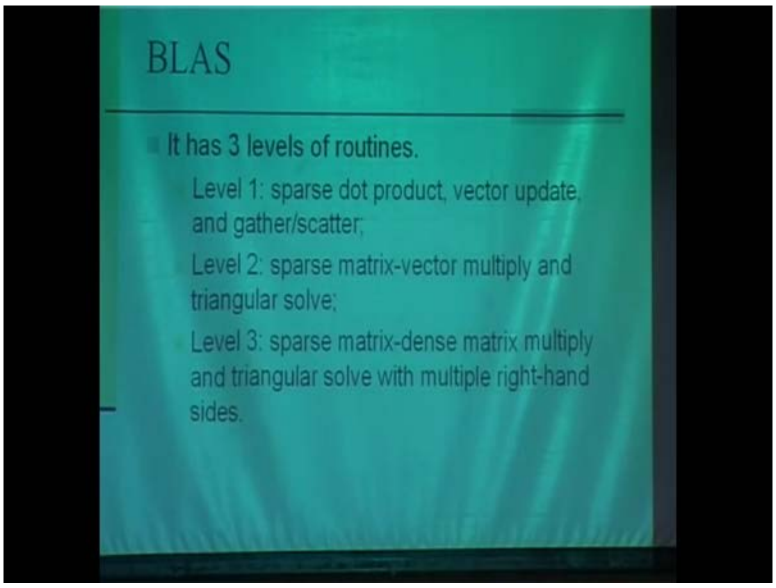
BLAS

- **B**asic **L**inear **A**lgebra **S**ubprograms.
- Involve low level operation like dot product and vector product.
- Also, Matrix-Vector and Matrix-Matrix Product.
- A Very structured set of programs generally used for linear system of equation, eigenvalue problem and singular value problem.

First of all I mean, we starting and we should understand what Blas is? And how will parallelize it on what model we have to parallelize? So, Blas is basically liner algebra subroutines. Blas are some Blas is basically a library which provides some functions to do some calculations in like linear system of equation Eigen value problem or singular value problem.

So, there are some a library routines which in the Blas. These library routines are basically are some like they provide dot products they provide vector products they provide matrix vector product matrix product. So, these types of routines are already available in the Blas. And benefit with the Blas is that this is very structured, this is way very well documentations provide and it is very structured.

(Refer Slide Time: 43:44)

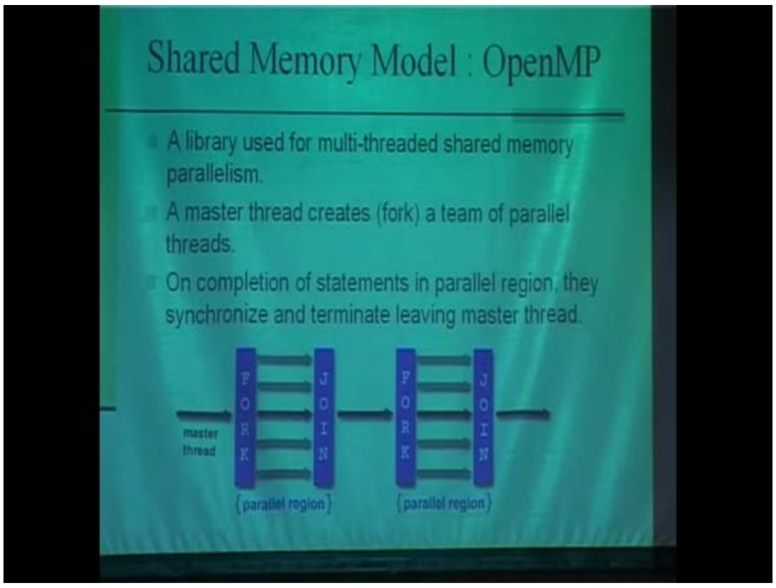


BLAS

- It has 3 levels of routines.
  - Level 1: sparse dot product, vector update, and gather/scatter;
  - Level 2: sparse matrix-vector multiply and triangular solve;
  - Level 3: sparse matrix-dense matrix multiply and triangular solve with multiple right-hand sides.

So, in Blas it has basically divides the divide, it is routine into 3 levels. First level is sparse dot product vector update gather and scatter level. 2 is this Blas matrix vector multiply and triangular solve, and level 3 is matrix dense matrix multiply and triangular solve with multiple right hand sides. So, I am basically trying to emphasize on 3 level 3 routines and in level 3 I am emphasizing on matrix product.

(Refer Slide Time: 44:28)



Shared Memory Model : OpenMP

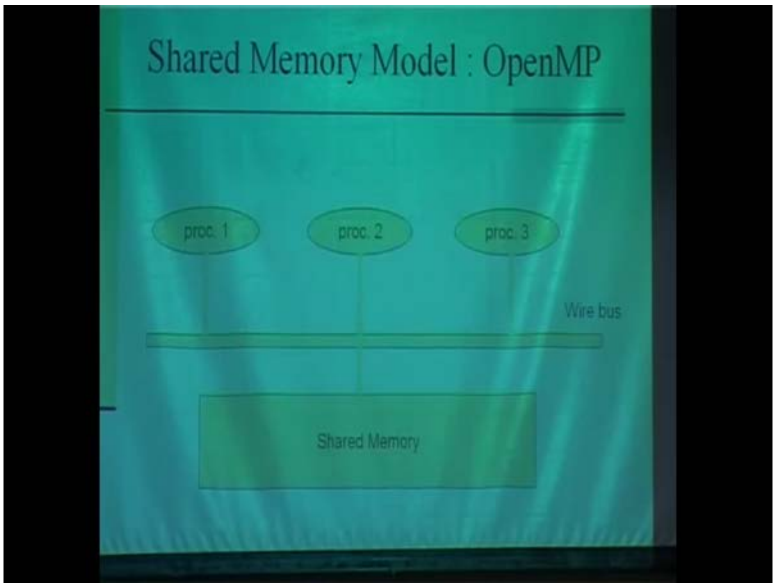
- A library used for multi-threaded shared memory parallelism.
- A master thread creates (fork) a team of parallel threads.
- On completion of statements in parallel region, they synchronize and terminate leaving master thread.

master thread
FORK
JOIN
{parallel region}
FORK
JOIN
{parallel region}

So, this was the Blas and now I have to parallelize the Blas on shared memory model. For shared memory model, I will use open m p open M P. Open M P is basically a library used for multi-threaded shared memory parallelism. It basically works like this a master thread is fork into some child threads and then when the parallel parallelism ends all the parallel region ends, it just joins and the master thread work as sequentially. So, master thread always works sequentially, and it just creates some threads by fork when the parallel region is start and then it joins.

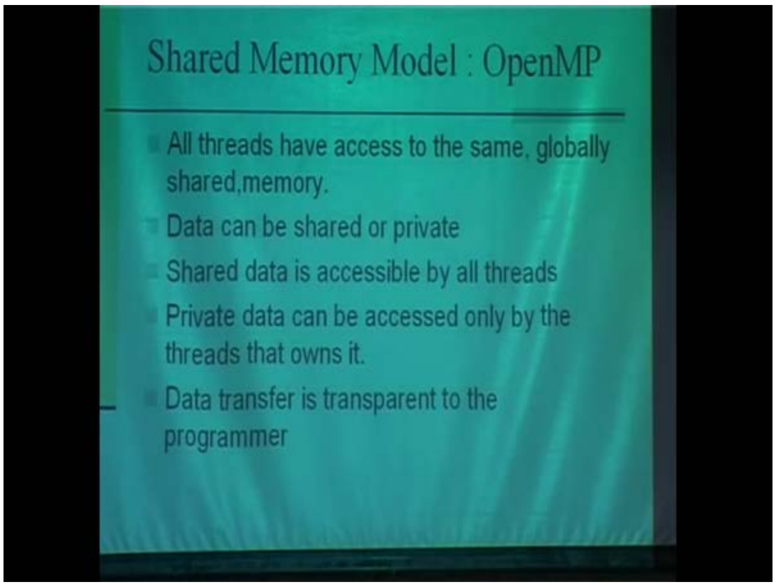
So, the benefit with the open M P is that it provides some pragmas, you do not need to take care of the communication between the threads, and you do not need to take care of a synchronization everything is already there. It has implicit by rehearse it has implicit in communication. So, you do not need to go inside the parallelism, you can just write the pragmas for the parallelism and it is done.

(Refer Slide Time: 45:55)



So, this is the open M P. In, how it works? I mean, this is shared memory, the other processes they can talk to shared memory through a wire bus and they itself has some cash.

(Refer Slide Time: 46:14)

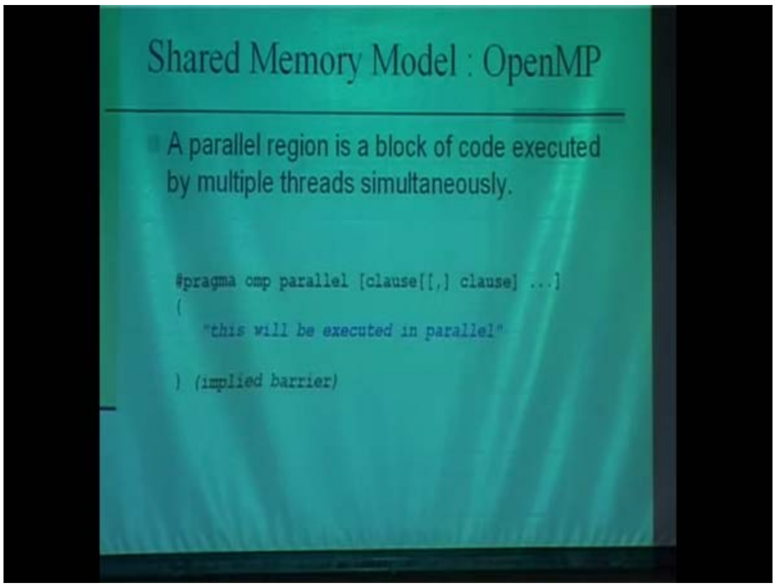


Shared Memory Model : OpenMP

- All threads have access to the same, globally shared,memory.
- Data can be shared or private
- Shared data is accessible by all threads
- Private data can be accessed only by the threads that owns it.
- Data transfer is transparent to the programmer

All threads have access to the same globally shared memory data can be shared or private you can define which data should be private, which data should be shared through Open M P pragmas and some clauses. So, private data can be accessed only by threads data transfer is transparent to the programmer. This is the best thing for while using the Open M P. You do not need to take care of the data transfer it is done automatically.

(Refer Slide Time: 46:50)



Shared Memory Model : OpenMP

- A parallel region is a block of code executed by multiple threads simultaneously.

```
#pragma omp parallel [clause[[,] clause] ...]
{
    "this will be executed in parallel"

} (implied barrier)
```

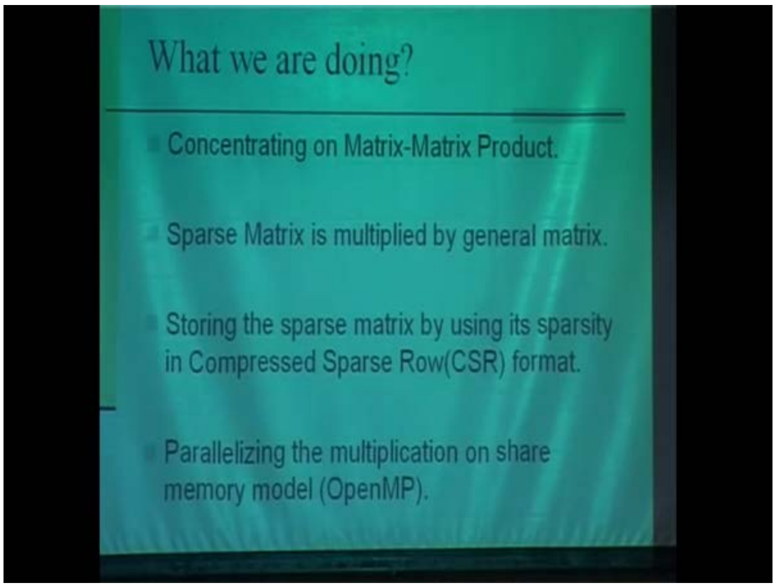
So, a simple parallel structure it looks.

What do you mean by data transfer?

Data transfer,

Data transfer, I mean communication between this processor and this shared memory and even between the threads. So, a simple parallel structures looks like hash pragma O M P parallel and some clauses. So, this is the region for parallel code. And what we are doing is basically.

(Refer Slide Time: 47:28)

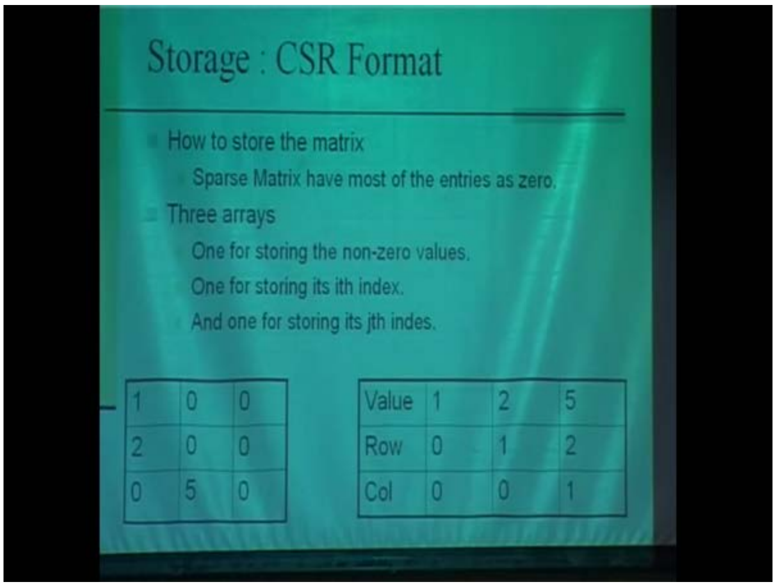


What we are doing?

- Concentrating on Matrix-Matrix Product.
- Sparse Matrix is multiplied by general matrix.
- Storing the sparse matrix by using its sparsity in Compressed Sparse Row(CSR) format.
- Parallelizing the multiplication on share memory model (OpenMP).

I am doing matrix product. And first of all I will I store the matrix into a compressed structure row format and then do a sequential multiplication and then parallelize the multiplication through Open M P.

(Refer Slide Time: 47:50)



Storage : CSR Format

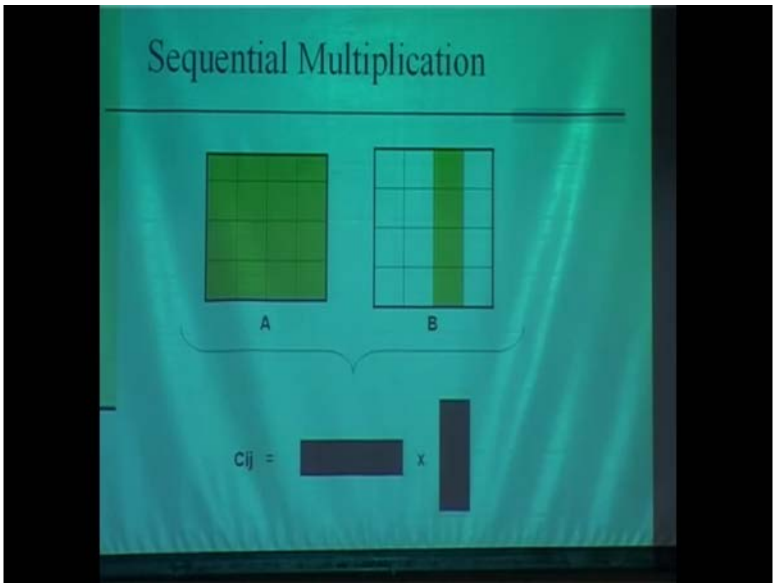
- How to store the matrix
  - Sparse Matrix have most of the entries as zero.
- Three arrays
  - One for storing the non-zero values.
  - One for storing its ith index.
  - And one for storing its jth indes.

1	0	0
2	0	0
0	5	0

Value	1	2	5
Row	0	1	2
Col	0	0	1

So, what is CSR format? Compressed structure formats. So, this sparse matrix has a lot of entries as 0. Around a sparse matrix is has at least less than one third entries non-0. So, that matrix can be assumed as a sparse matrix. So, the easiest representation of sparse matrices CSR formats. So, I am storing a matrix given matrix into 3 arrays. The 3 arrays are: value array row array and column array. So, assume like this is a matrix. So, it has some entries how this entire will be stored in three arrays like values will you store, what are the actual values and the row will row and column will store their indexes? I mean see if value 1, value 1 is that 0 0 coordinate. So, row will be 0, column will be 0. 2 are at row 1 and column 0. So, for 2 rows will be 1 and column will be 0. So, this is how I am storing sparse matrix into 3 3 arrays it basically reduce. I mean size it reduce the memory sparse needed to store sparse matrix.

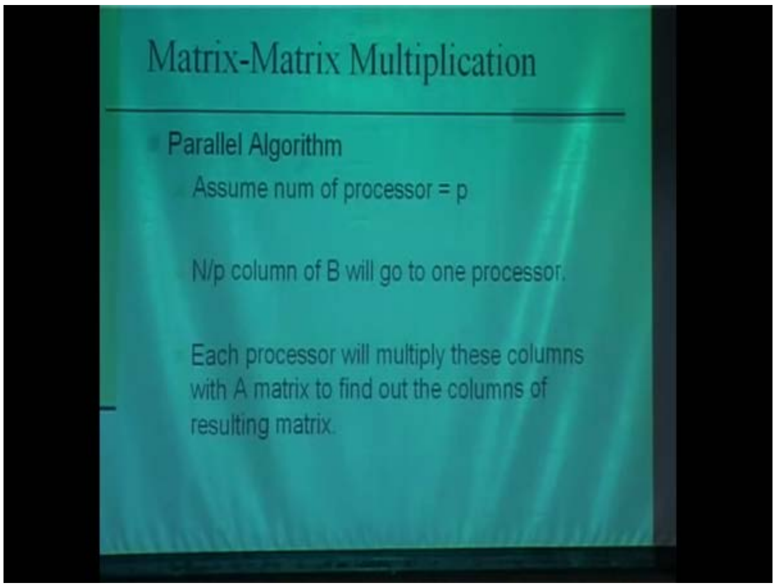
(Refer Slide Time: 49:28)



Sequential Multiplication

So, sequential multiplication given in the blases like, you just take you just take the matrix, and multiplying with the column of b matrix then multiplication of this and this will give you a column. And a sorry this matrix multiply by this matrix which in term calls this row is multiplied with this column, and then it will give a 1 entry. So, when a matrix multiplied with a column it gives a one column of c. So, this is how matrix multiplied with this column gives one column this matrix multiplied with this column gives another column. So, this is how the columns are generated.

(Refer Slide Time: 50:13)

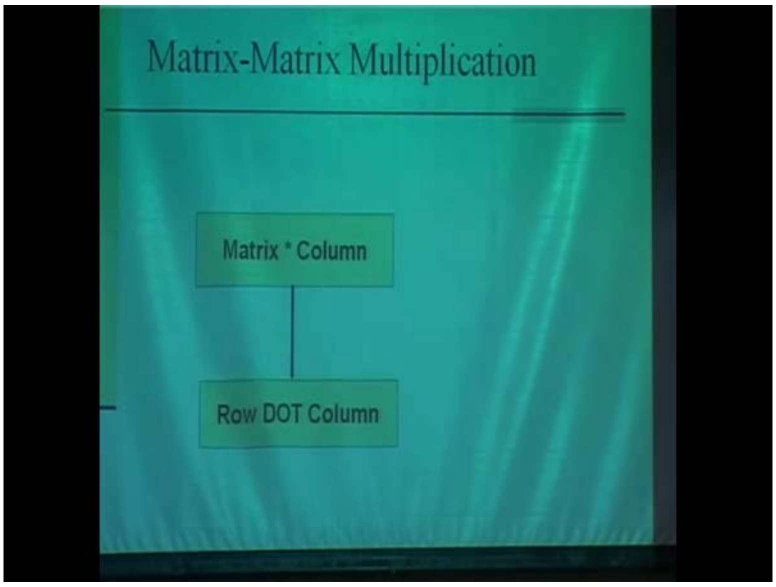


Matrix-Matrix Multiplication

- Parallel Algorithm
  - Assume num of processor = p
  - N/p column of B will go to one processor.
  - Each processor will multiply these columns with A matrix to find out the columns of resulting matrix.

So, parallel algorithms assume p processor.

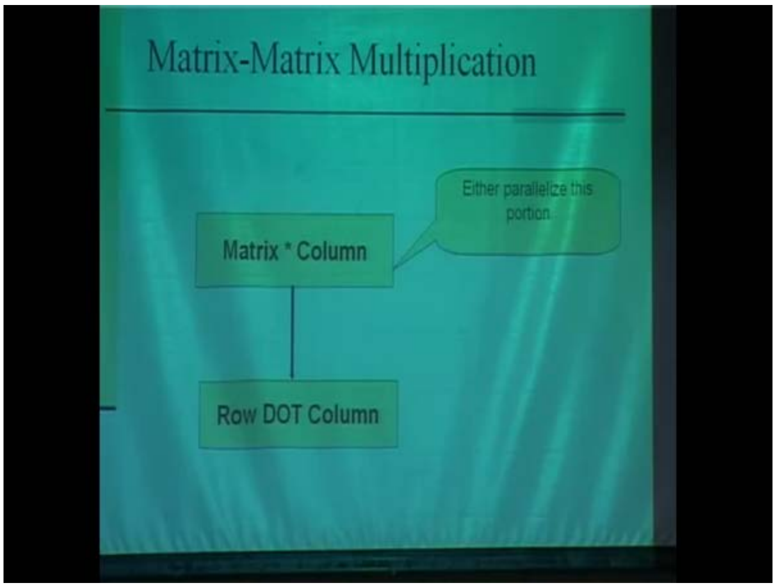
(Refer Slide Time: 50:18)



Matrix-Matrix Multiplication

So, basic 2 things are matrix multiplied with column which in term calls the row and column, dot product of row and column. So, these are the basic 2 steps involved in the matrix multiplication.

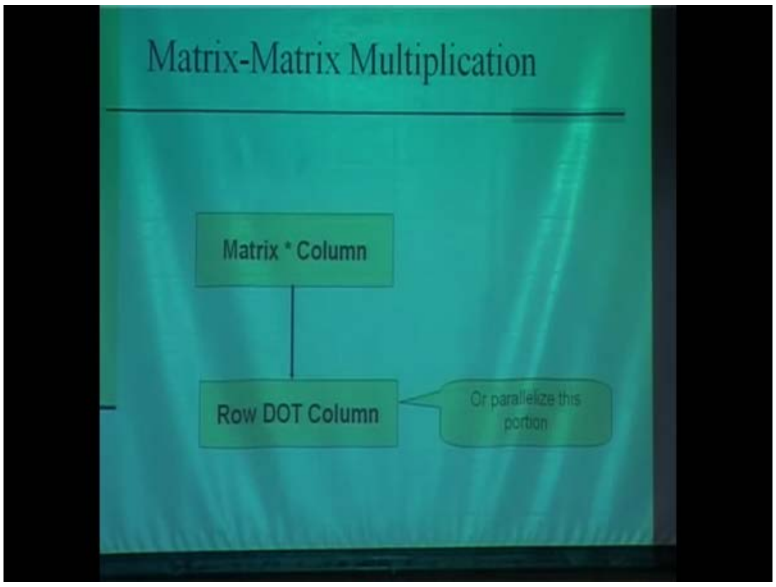
(Refer Slide Time: 50:38)



Matrix-Matrix Multiplication

So, 1 thing is either parallelize this portion matrix into column product.

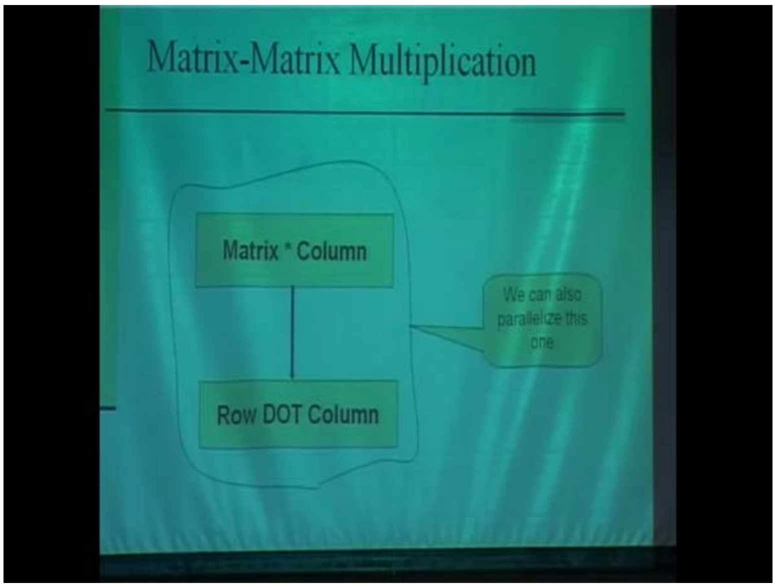
(Refer Slide Time: 50:46)



Matrix-Matrix Multiplication

Or parallelize this row, a row column dot product.

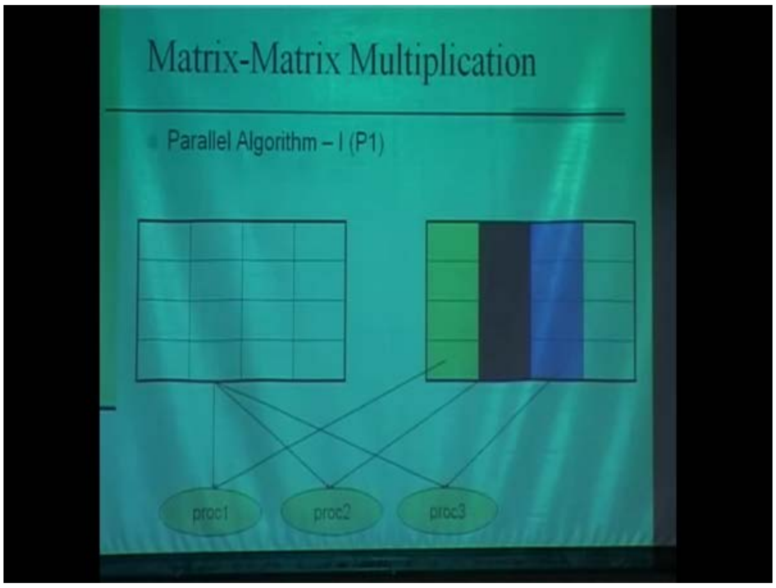
(Refer Slide Time: 50:50)



Matrix-Matrix Multiplication

Or we just parallelize the whole thing. So, they are 3 possibilities of parallelization. So, i tested all the 3 possibilities here.

(Refer Slide Time: 51:00)

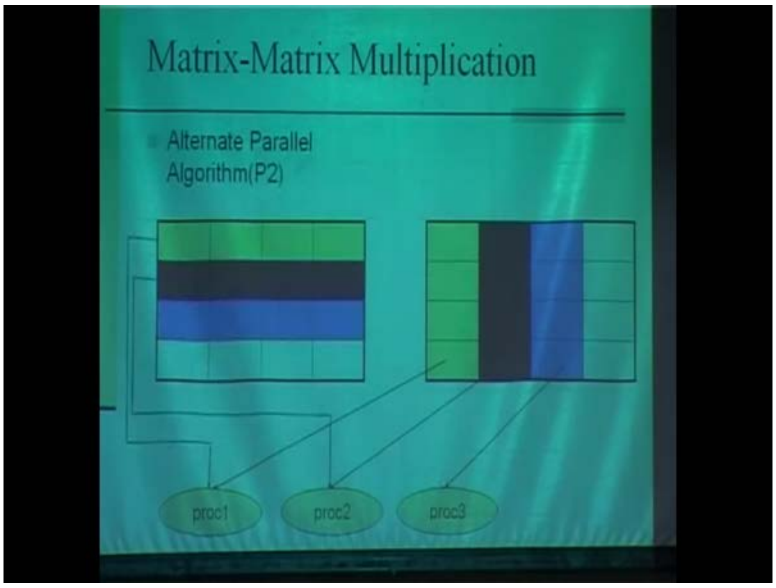


Matrix-Matrix Multiplication

- Parallel Algorithm – I (P1)

So, this is the first thing, in this is I named it as P1 algorithm. P 1 algorithm means, in which a processor will store will take the matrix and a column and multiply it. So, the parallelization is that this is steps matrix into column multiplication. So, P1 is the matrix into column multiplication parallelization.

(Refer Slide Time: 51:28)

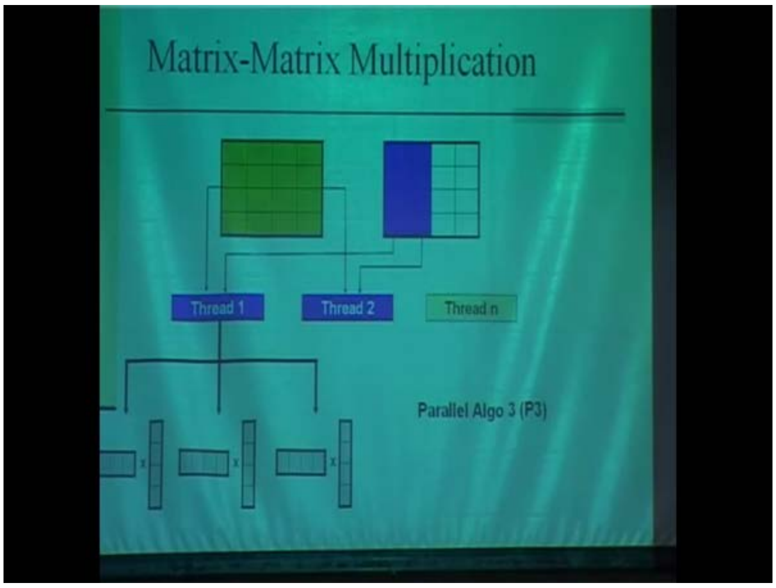


Matrix-Matrix Multiplication

- Alternate Parallel Algorithm(P2)

P 2 is row and column dot product. So, row and column will be multiplied in 1 processor. So, row column will multiply on a single processor, this is I named it as P 2.

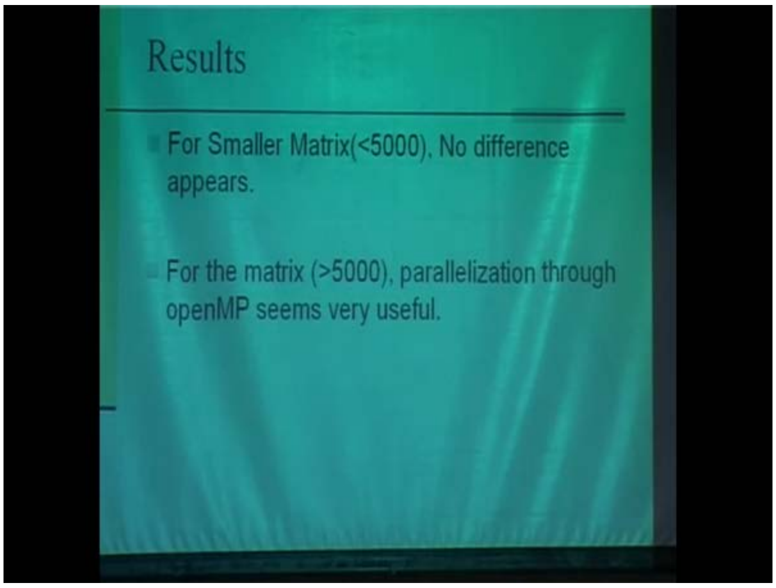
(Refer Slide Time: 51:49)



Matrix-Matrix Multiplication

And another option was that this parallelize matrix into matrix into column multiplication. So, matrix and 1 column will go to 1 thread matrix, and another column will go to another thread matrix, and another column will go to another thread and after that 1 thread will divide into sub threads and then multiply the row and column. So, this was another option with me. So, I named it p3, now I will evaluate all these P 1, P 2, P 3 algorithms.

(Refer Slide Time: 52:28)



Results

- For Smaller Matrix(<5000), No difference appears.
- For the matrix (>5000), parallelization through openMP seems very useful.

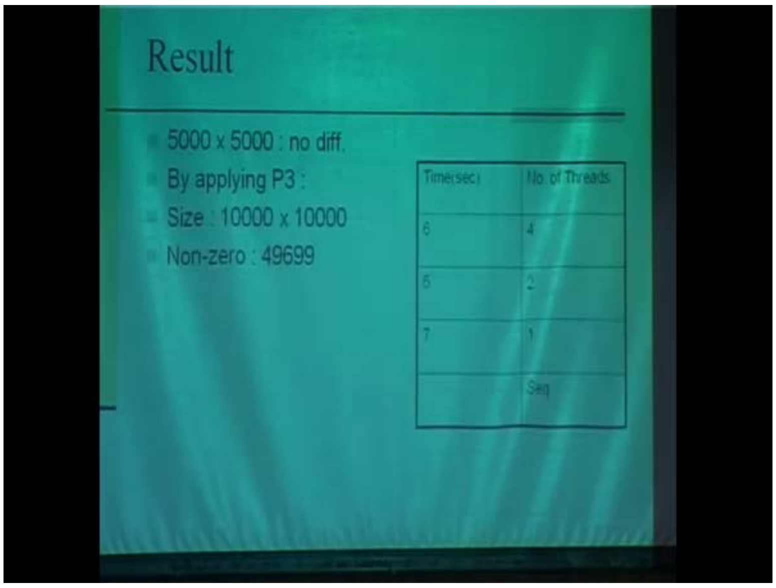
I checked above 5000 because less than. 5 is a random sparse matrix.

How did you get from matrix form?

Matrix market dot com.

They have pass. So, I do not know whether they are random or not. But, the b matrix I take it as a random.

(Refer Slide Time: 52:53)



Result

- 5000 x 5000 : no diff.
- By applying P3 :
- Size : 10000 x 10000
- Non-zero : 49699

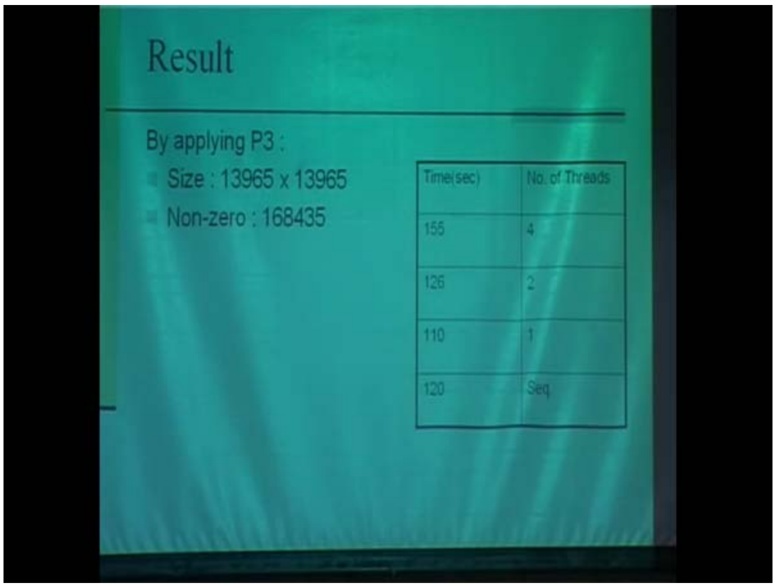
Time(sec)	No. of Threads
6	4
5	2
7	1
	Seq

So, for 5000 cross 5000 no difference, for 10000 cross 10000 this is for P3. P3 was this p 3 was this parallelization at 2 steps ok.

So, for this thing I checked for 10,000 cross 10,000 and number of threads, and time these 6 seconds, 5 seconds, and 7 seconds a 4 number of threads and adulate right for sequential. Because see in open m p it is very important to check out the optimal number of thread optimal number of threads. Because when you create number of threads it takes some. Due to communication and thread creation and thread destroy it takes a lot of time and at the synchronization some threads are waiting for another threads.

So, if you unnecessarily create some extra threads it will take extra time. So, it is not a all ways necessary, that you just create 10000 for 100 cross, 100 matrixes it takes. It will give you good time, it can take more time.

(Refer Slide Time: 54:17)



Result

By applying P3 :

- Size : 13965 x 13965
- Non-zero : 168435

Time(sec)	No. of Threads
155	4
126	2
110	1
120	Seq

So, this is for around 14000 cross, 14000 the time was 1.50.

Sir,

Because, I found it on the sight, I cannot change even this number.

What? So, I mean it is around 14000 plus 14000.

I mean, it is like sequential yes of course.

I mean, it is almost this difference is I mean, when you do some more iteration, it will give you to a 125. It may give you 115.

But, what you do is what is the?

What is the difference between sequential and one number of?

I mean they are always same,

I mean they are same,

10

Even I mean I tested it.

Time should be more.

Yes, I mean,

But, I mean I do not know.

Taken the average of 3 iterations.

May be,

same sparse circle.

Sir, Even I mean sometimes the results were very different I mean in the morning you take some other.

Sir, it was I mean [laugher] [voclised-noise] because mean,

[laugher]

Sir, but I done it at the c s e machine.

You take standard algorithm you want separate off from the net and you make your internal that was in .

No I mean,

No I mean,

You cannot though.

Sir, that was a problem with me I mean, because,, it depends on the number of users connected to this computer all the classes.

This time should be your time right you know who are the other users using a then.

This is the problem sir, now?

Both they are asking there I want four matrix for 5.

Sir, I mean, just say what is open mp and we do not know everything.

No,

He speak very sir [laugher]

So, you need you need!

Yes, of course, I mean,

This time you take you tell.

Yes, that.

How many?

I mean,

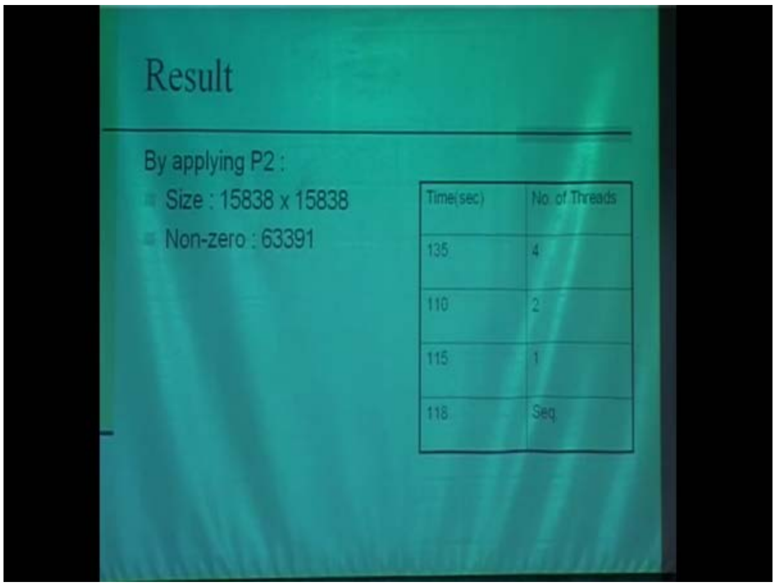
You have left,

Yes, I mean but, I do not have open up.

[laughter]

So, I mean is just look up for the, we can have something confusion. So, I mean.

(Refer Slide Time: 58:54)



Result

By applying P2 :

- Size : 15838 x 15838
- Non-zero : 63391

Time(sec)	No. of Threads
135	4
110	2
115	1
118	Seq

No here, probably more problem you take 1.

Yes,

One side 15, 2 sides 1.10.

I do not know it was dual core or I do not know.

Duel code.

I mean because that is not a dual core I think c s machine this are not dual core.

Black once black once are duel code.

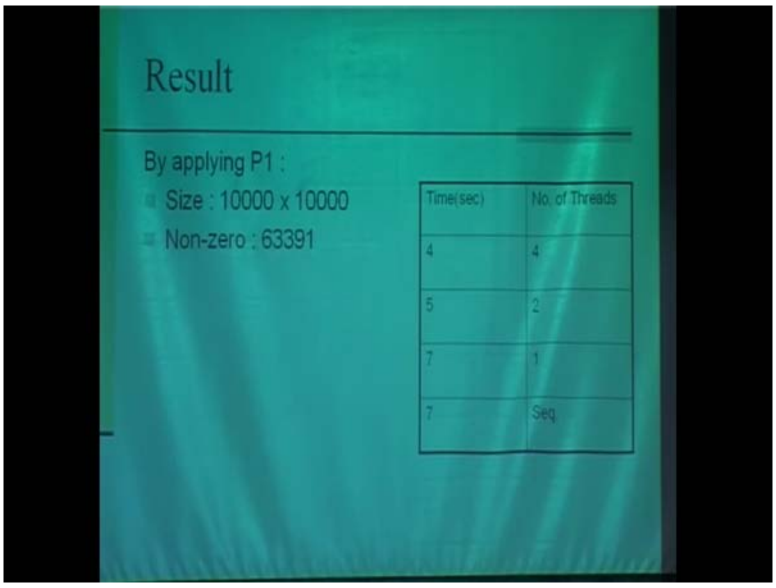
Yes, black one.

Your program is ready right.

Yes,

You need a machine.

(Refer Slide Time: 1:00:07)



Result

By applying P1 :

- Size : 10000 x 10000
- Non-zero : 63391

Time(sec)	No. of Threads
4	4
5	2
7	1
7	Seq

So, these are the some viewings which I got.

Sir,

Sir, there is one distance a c i t s i p with four processers.

He is also.

Automatically blast has automatically.

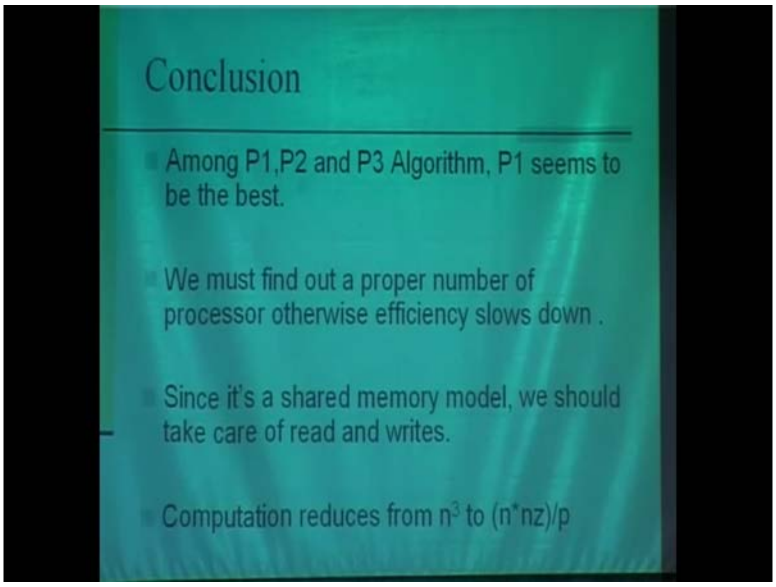
Blass is the routine and that routine automatically find out the number of noises and you decide that is the right and you optimize that thing now what.

So, it should be coming better if I take.

What is better what you are this analyzes?

Yes I mean because it is in effect with some other.

(Refer Slide Time: 1:01:29)



Conclusion

- Among P1,P2 and P3 Algorithm, P1 seems to be the best.
- We must find out a proper number of processor otherwise efficiency slows down .
- Since it's a shared memory model, we should take care of read and writes.
- Computation reduces from n^3 to (n*nz)/p

So, I mean anyway whatever the readings from that reading, why?

These are all uses for you?

Why?

Yours is not the see these your thing is implementation right.

Yes,

So, you should tell now this model or this machine we use fourth right this is the result like the time you have in terms of time.

Yes,

None of this because you cannot block such type of

I mean, we can reduce because, I mean whatever is the number of non non-0s only non-0s numbers will multiplied with the b matrix 0s will not be multiplied. So, instead of this n cube, sorry that should be n square. So, non-zero to non-zero numbers only multiplied with the b matrix. So, that is why the computation.

If computation reduces parallel computation reduces what is computation.

I mean from sequential n cube is the sequential and now when you go to the parallel once then it reduces by p.

See sparse matrix multiplication algorithm cannot take,

Yes I mean yes this should also be a n z into p n z into n sorry.

Say once I.

This side is?

Yes this side is.

This should be n z n z into n.

So, anywhere I mean from whatever the results that we got some those things I just to conclude that p 1 seems with the best.

(Refer Slide Time: 51:00)

I mean P 1 this parallel parallelization seems best because. But, the threads are creating a lot of I mean, one thread is creating then forth threads are creating then dynamic load balancing.

Yes,

That is you dividing the number of major number of non-0 elements.

Yes,

It is based on the number of non-zero.

Then, equal number of non-zero.

Equal number of sir, the number of non zeros,

So, actually this is the root is given in the Blas, I cannot do anything I cannot change.

Why the roots are available know?

Yes, I can change but, I am parallelizing the Blas. So, I will they should be Blas I cannot put I mean this algorithm Blas.

Blas algorithms I cannot change otherwise hold be a Blas that is why and Blas is taking row and multiplying with column. So, if I parallelize with row and column then it may have I mean, one row can have 2 or 3 non-0 elements.

Then why do you representing r c b form.

RCB,

CSR form,

That is only to store. I mean, big matrix to store in the otherwise it will take a lot of sparse and when it will multiply it will just 3 arrays are passed, and a corresponding entries are passed, I can show you the code.

No, that anyway you have to show.

You are filling the?

No I can show you the demo right now.

Demo,

No, the c machines to exes it.

Now? You are prevent it on the machines, and then.

Yes I mean basically.

That will do a load balancing also.

Any other solutions you want to give him?

Is any other solution?

No.