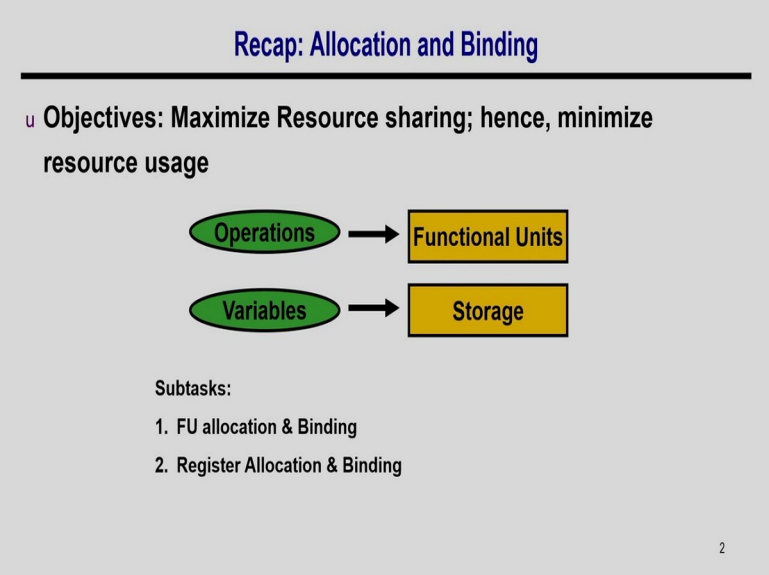
C-Based VLSI Design
Dr. Chandan Karfa
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Module - 05
C-Based VLSI Design: Allocation, Binding, Data-path and Controller Generation
Lecture - 19
Allocation and Binding: Hierarchical graph

Welcome everyone to my class or to C-based VLSI Design. In today's lecture, we are going to discuss the resource Allocation and Binding problem for Hierarchical sequential graphs.

(Refer Slide Time: 01:05)



So, we have understood that this allocation and the binding problem is something where we map these operations of the behavior into functional units and the variables of the behavior to the storage specifically register RAM ROM and memories ok.

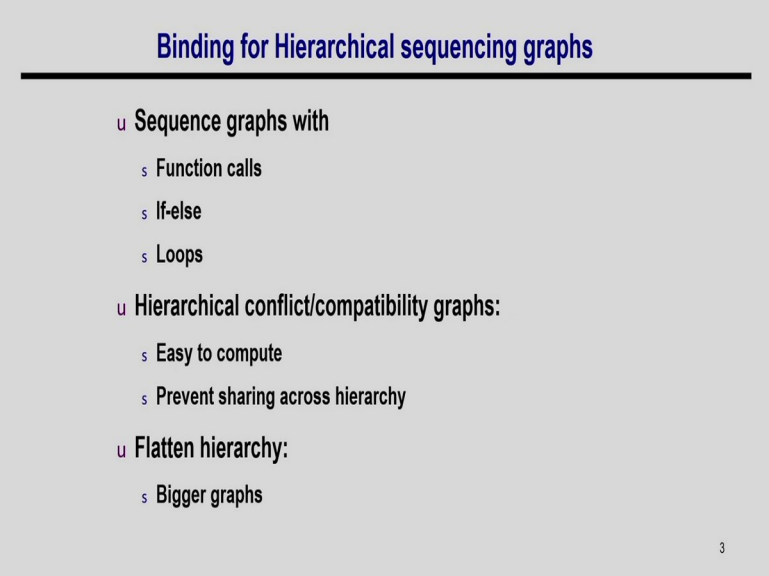
So, and we have seen that this particular problem we have to take these two problems separately, this operation to functional binding first and then variable to storage unit then right. So, we are discussing the first problem that FU allocation and binding, and then what we have seen in previous classes that, we can map this particular problem to either graph coloring problem or clique partitioning problem right.

So, that we have already discussed. And then, also we have understood that this particular problem is actually an NP-complete problem; which means, it's a computationally hard problem; that means, we do not have any polynomial-time algorithm to solve this problem efficiently right.

So, then we have seen that this particular problem is a for a particular class of problem where this particular conflict graph is an interval graph, we can solve this problem efficiently what does it mean? We can actually have a polynomial-time algorithm to solve the problem and then we have seen that we have a left edge algorithm that can solve this problem easily also we have discussed that if we try to solve the problem of FU allocation and binding only for a particular basic block right.

Then this particular problem is actually can be solved in polynomial time right in that part we have already discussed.

(Refer Slide Time: 02:41)

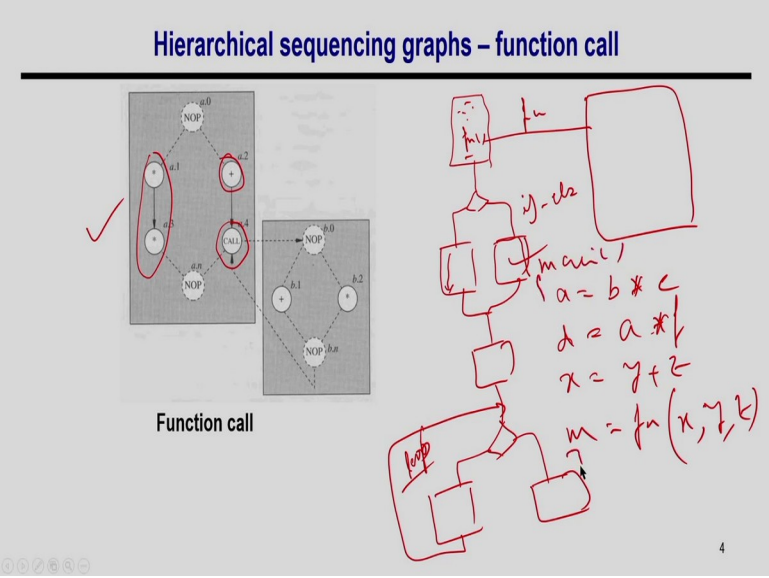


So, in today's class what we are going to discuss is that, in general, when you are given a program, it does not only consist of a single basic block right. It consists of a set of basic blocks and which is basically have a control dependency which basically has if else we have loops and also function calls right function call is very common in C programs right.

So, in this particular class, we try to understand what will happen if we try to solve the problem of this allocation binding complete operation into the whole graph right to the complete hierarchical graph right. So, that is something we try to discuss today.

And we will see under which circumstances this particular problem will be will fall into a polynomial-time algorithm and under which category it will not fall under this polynomial-time algorithm and we have to solve it using the heuristic algorithm because that problem will remain NP-complete ok. So, that was the thing we are going to discuss today.

(Refer Slide Time: 03:39)



So, first, we will understand what is the hierarchical graph. So, the hierarchical graph is something is where you have to say basically in the C program what you have written. So, you write a sequence line of code and then say you have a function call here right you can have a function call here.

So, you have some code here, you have function call here then you might have a branch, then you do certain things here you do some certain things here, then you come back you do something here and then say suppose you have another condition and then you do say some set of operations in the loop. So, this is a loop, and then once the loop ends you do something right. So, this is basically a function call. So, the function call is nothing, but another body.

And then this is if-else right this is a function call and this is loop right. So, this actually this control flow actually is a realistic one right. So, we have discussed that if we have a straight line of code there is no control flow, there is no if-else there is no function call there is no loops there is this straight line of codes; that means, a series of codes written there if you take that only one basic block, that is then we can actually solve this allocation binding problem efficiently using left edge algorithm.

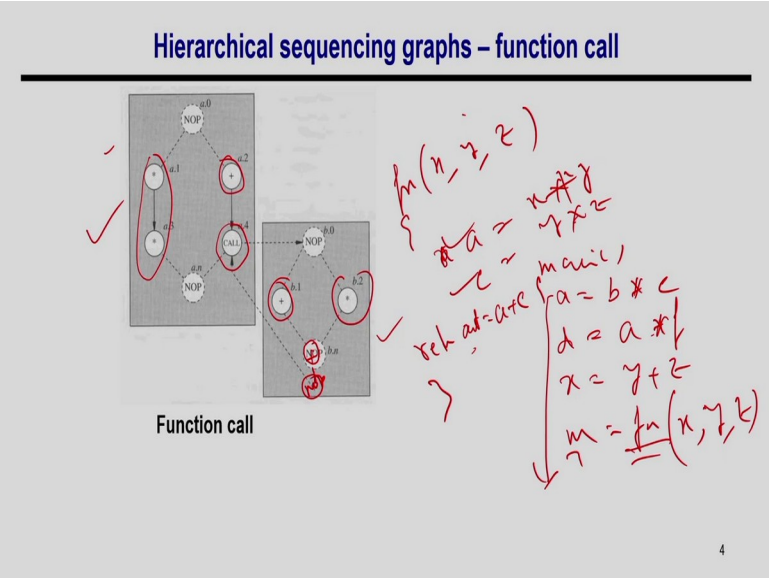
Because the conflict graph will become an interval graph. But in general when you consider the conflict graph like this which is the realistically the or C code will be like that only then how you're this conflict graph looks like and whether that particular remain in the conflict graph will remain an interval graph or not that is something we need to understand ok.

So, let us understand the hierarchical sequence of the graph first right. So, suppose you have a straight line of there is a code like this and there is a function call here right. If I try to map this particular diagram to a realistic code it is like this right. Say suppose you have

a = b * c and then suppose d = a * f right. So, this is that part. So, this is I am mapping to this, this is my a and this is my d and say this is x = y + z, this is that operation and then you have a function call right.

So, suppose we have a function where I am passing x say m=fn(x, y, z) and I am getting the value into say in say in m some variable right and in the function what I am doing this is the this is this body this is my main function. So, this is my say main right. So, this is my main so, I can delete this part right.

(Refer Slide Time: 06:23)



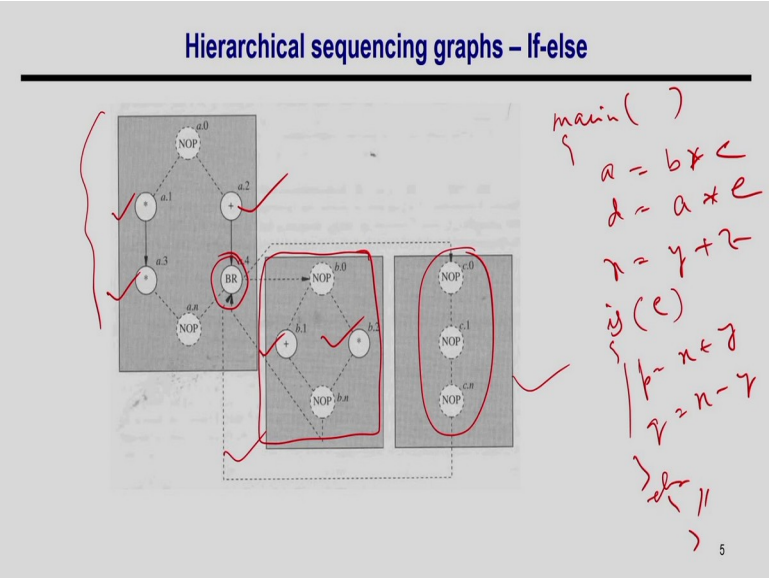
So, this is my main function and then I have to say this function, this function body says this is the function where I am passing x, y, and z.

And what I am doing here say x plus say its suppose I am doing say x say a = x * y and then some c = y * z and then say there is a + operation here and then there is a NOP ok. So, then say I am sending this out = to a + c right so suppose this is what I am doing. So, now, you have this code. So, this is a straight line of code there is NOP only here there is a function call.

And then in the function call, we have this body where I am just doing this two operation this operation is this, this operation is this, this is plus, and then finally, I am sending this out as a return right. So, this is I am returning. So, now, this is what the hierarchical graph looks like right. So, now, I have scheduled this particular main block because this is a basic block, I also schedule this basic block right I have a schedule.

And now I have two schedules together; I have to merge this club I have to club these two schedules together and I will get a combined schedule graph right combined sequence graph and then on top of that I am going to do that resource allocation and binding right this is what is about the function call.

(Refer Slide Time: 07:46)



So, now, you have if-else is also similar that you have a say the main function and then suppose there is an if-else call here right.

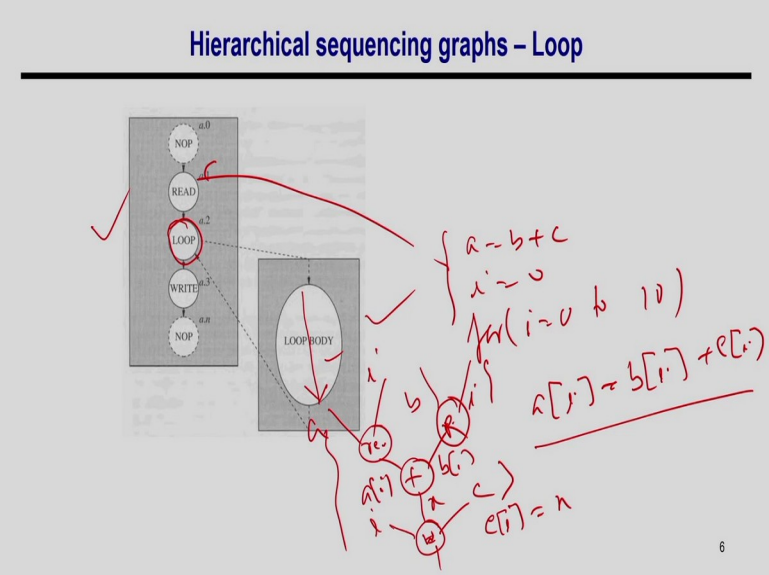
So, if I just try to map this into a realistic code it is like this, this is similar to the previous code that I have to say a = b * c this is this operation then say this is d = a * e right. So, this is this operation and then say x = y + z this is this operation and then suppose I have a function call right say suppose some condition under certain condition if this condition is true, then I am going to do say p = x + y.

And then and say q = x - y right this is the operation I am mapping these two operations these two are basically independent operation and then else there is nothing right. So, else nothing is there right nothing. So, then what will happen? This is my if-else if branch is also basic block and which is this code and else branch is there is nothing. So, this is NOP is the no operation right.

So, and now if you have this is the main function and this is my if-else and then this is my combined sequence graph right. So, I will schedule this part of the basic block once I will schedule this part of the basic block once, I will schedule this until there is no operation, but it might operation might happen there right.

So, now we have the combined schedule graph right. So, I know the what this is is the overall combined scheduled graph and I want to do the resource allocation and binding problem for this particular graph ok.

(Refer Slide Time: 09:28)



So, this is what is the if-else thing ok and then the same next part is the loop right. So, this is the loop and loop is something you have a loop call here and then this is the loop body. And you know the loop body is something that is a straight line of code right. So, you can suppose.

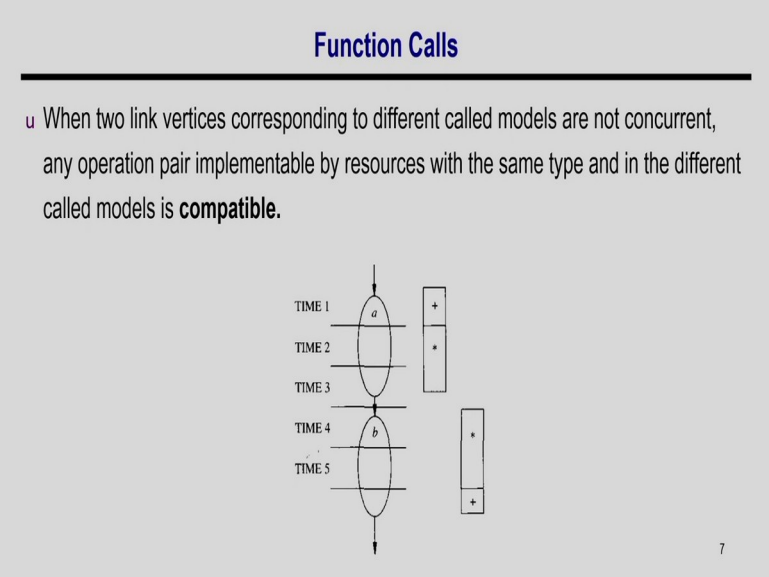
So, basically you have say some codes like this say you are doing something a = b+c then i = 0 and then in the for loop i = 0 to 10 you are doing say a[i] = b[i] + c[i] right. So, then what will happen? So, in this part, this code is basically mapping to this sum of the nodes here.

And then you have a loop body and the loop body is this. So, you have another sequence graph here say it will be like this say you are doing like this some plus operations you are having an operation where you send a send i it will give you a[i] you have another node where you give b you give i you will get b[i] and then you will get the result x. And then you basically have another node where you give i you give c you give x then it is basically here it is happening equal operations where basically c[i] = x right.

So, this is how you can actually this is read operation right this is read and this is written and this is a plus operation ok. So, this is how you can actually draw this sequence graph for this loop body and then you basically schedule this loop body independently you independently schedule this operation, you independently schedule this the main block and then you have a combined sequence graph and you try to do the resource allocation and binding problem for that combined graph right.

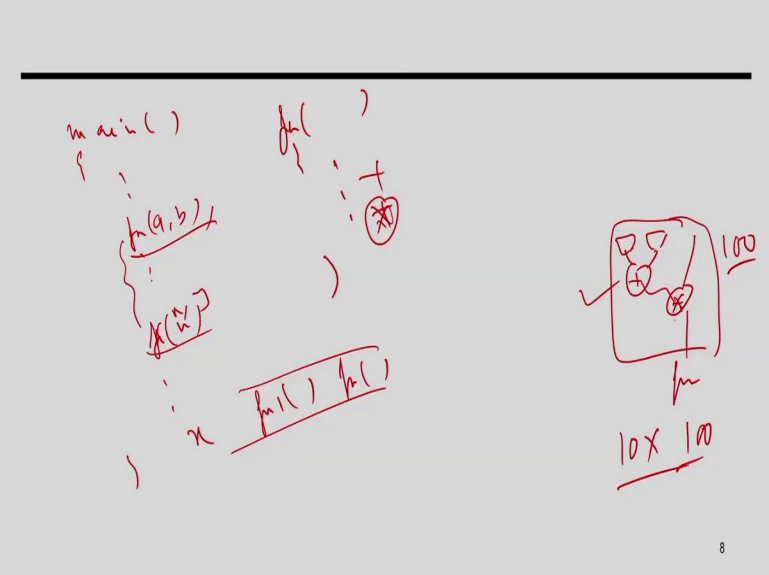
So, we understood that I mean along with basic block the main control flow in the hierarchical graph is a function call which is primary, and then if we have if-else which is branch and we have loop body right. So, that if-else I try to mean that it's a conditional branch, it can be a switch case as well right. So, I am just trying to give the branching thing and the loop is can be while loop, do-while loop, for loop anything, but I am just trying to say the loop ok so, in general terms.

(Refer Slide Time: 11:45)



So, now what we are going to discuss here is if you have function calls how your conflict graph look like whether this particular problem will remain interval graph or not that is something I am going to discuss now ok. So, first of all, you try to understand one thing. So, we have function calls right. When you write function in a program, say suppose we have a main and you write. So, you have some code then you say call the function right.

(Refer Slide Time: 12:08)



And then again after some time you call the function. So, when we write certain things in a function when basically that particular code is going to repeat many times right. Instead of repeating the whole code in the main I just write a main copy of the function right set of lines and then I am going to call this function many times right. So, that my code size remains I mean small right.

So, what does it mean? We have a repetitive kind of operation which has to be executed multiple times and then what do we do? We basically take that particular repetitive part of the code into a function and I am going to call this particular function multiple times right. So, that is the purpose of a function call in a C program ok. Now you think you have one copy of the function now you think about I have a hardware right.

So, in the hardware what usually happens is that this whole operation will map to some hardware units like function units, registers interconnections, and all right what will be our objective of corresponding to a function that is something you should first understand right. So, if say 1 function is called 10 times in the main program in the hardware what do we want? We need 10 copies of the function because the hardware is a fixed unit right.

So, these functions are doing some adder operation multiplication operation right. So, suppose there is an adder here and there is a mult here and their interconnections right it is coming like this. So, there is a mux here and so, on. So, something like this right. So,

some function call this is my function body say in the hardware, the hardware realization of the function body.

Now, the question is, if I have 10 calls to this function what will be our objective right? So, our objective will be to create one copy of this hardware right in the hardware. If we create 10 copies of the hardware; that means, my area requirement will be 10 x or my resource requirement will be 10 x because if the x is the say 10 times right. So, if the total requirement is taken as is 100 here. So, 10 * 100 so, this 100 is some units ok.

So, what I am trying to say is here once I try to map these functions into hardware my primary objective will be to create one copy of the hardware right for (Refer Time: 14:49) to a function, but it might be that the way schedule has been generated that sometime it may not be possible also right. It is because you schedule such a there are two calls and there is no dependency of these two functions and probably t schedule will do that they actually run these two in parallel right.

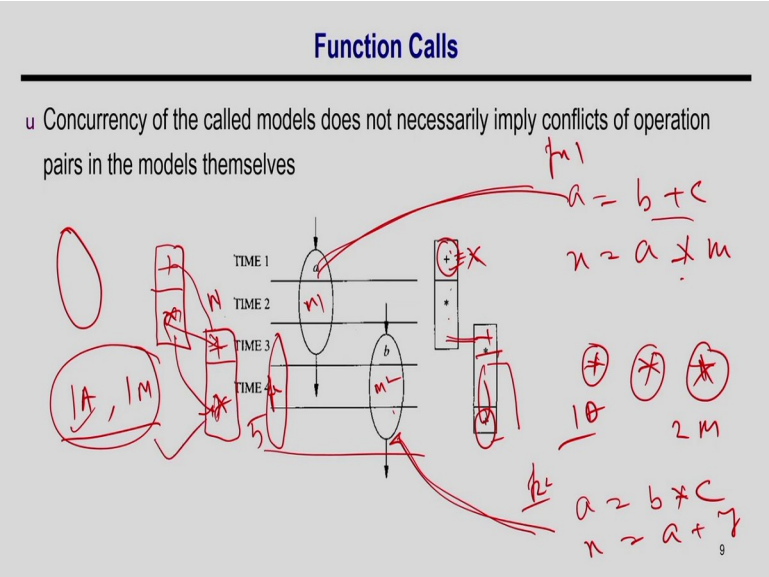
It can happen because suppose it is taking the argument a and b and this is taking into argument x and y there is no relation between these two. So, these two functions become independent to each other in that case the scheduler might schedule these two functions in the same time step right. So, you call these two functions 1 say, two instances in the same time steps this is a time step x in that case I cannot do that right.

But if the scheduler generates the schedule in such a way that it is possible to have the nonoverlapping of their execution time right. So, if that if the time where these functions multiple calls are actually getting executed, they actually have nonoverlapping then I can create one copy of the function and I can share that or I can actually use that same thing to execute multiple calls right. So, that will be my primary objective.

The secondary objective will be is to even if say I have a function and I understood that it is said that is only a function which is called only one time for simplicity let us assume that. So, once I have one. So, if I one function one call, then it definitely it will be a separate module right. So, there is no question about that. My secondary objective will be there is some multiplier there is adder within the function call and after this function call is done this particular hardware will remain ideal right.

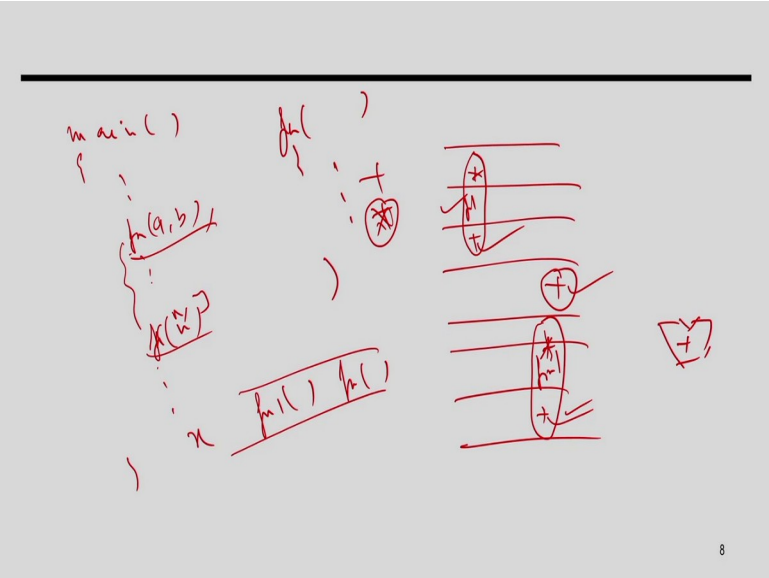
So, it has no use further because it's only one call or even if say there a multiple calls after the call is completed, this particular multiplier or adder for this function body will remain ideal for the rest of the time right. So, my secondary objective will be, is it possible to share this multiplier or adder within the function body with the other operations right. So, that will be my secondary objective.

(Refer Slide Time: 17:03)



So, with this objective in mind let us move on right. So, let us try to understand if there are function calls and what they look like right. So, suppose the first thing is if there are two calls of a function that is nonoverlapping right. So, suppose you have a function say. So, let me try to explain this, say suppose you have a function here and there is a function here and then they. So, this function is scheduled say let me to try to put it in properly.

(Refer Slide Time: 17:33)



So, this function is to schedule this call right. So, this is the function 1 call and then say this is scheduled somewhere here. So, it's a 3 cycle you need 3 cycles to execute this. So, the same function it called two times right. Since they are scheduling into different time stamps if their execution time is nonoverlapping its just like a normal thing right. So, it might have say some multiplication suppose I am (Refer Time: 18:00) add here add operation here.

So, this multiplier is taking two-cycle and this adder is taking a cycle. So, I can definitely have one copy of the module right. So, if they are nonoverlapping the problem is easy I can just take one copy of that and then I can actually share the same thing with some other multiplier say suppose one addition operation is happening here. So, I can utilize one adder right. So, one adder to execute this addition this addition because they are all over nonoverlapping right.

So, the summary of the this discussion is if we have multiple calls and their schedule time is nonoverlapping, it is just like a flattened graph flattened sequence graph which is similar to a sequence graph corresponding to a basic block and then this particular problem will remain chordal graph this conflict graph will remain this interval graph because it is there is no problem, it is basically just it is same like having a sequence graph corresponding to a basic block ok.

So, this is the case one where the function calls the same function call multiple times and they are nonoverlapping execution multiple functions calls multiple times and they are all execution things are nonoverlapping. So, in summary, if all the function calls have their execution time in a non-overlapping thing, I can flatten that sequence graph and I can get a single sequence graph where all these operations have this nonoverlapping time.

And hence I can solve the problem for normal sequence graph nonhierarchical sequence graph and this particular problem will become, it becomes this the conflict graph that you going to get it from here it will remain interval graph and I can solve the problem using left edge algorithm ok. So, this is the first part the second part is that if say there are two calls and they are overlapping right.

So, this is the call say of function M, I call this is the function first-time call this is the function second-time call and they have some overlapping. So, the overlapping does not always indicate that there is no sharing possible right. So, that is the first example I try to mean here suppose this function is doing one addition followed by multiplication right. So, suppose you just doing a =b +c and then say x = a * m.

So, this is the function body right. So, there is an addition followed by multiplication and the multiplication is a two-cycle operation ok and say this function is executed in times 1, 2, and 3 and this function is going to execute in times 3, 4, and 5 right so. So, now, you see here this is the corresponding intervals right. You can see here that these multiplication intervals are overlapping right, but the addition is not overlapping with this adder.

So, I can so, although there are two calls I cannot blindly say that since there is some overlapping of this function, I cannot do any sharing that is not correct, I can actually share the same adder to execute both this addition operation, but I need two separate multiplier right. So, I need two copies of the multiplier and one adder. So, 2 multipliers and 1 adder execute these two calls of the function right.

So, this is what I try to mean even if there are two function calls I mean calls of a function that has some overlapping, then it is not always the case that I cannot share anything. So, here I mean I try to mean two functions this is my function 1 where

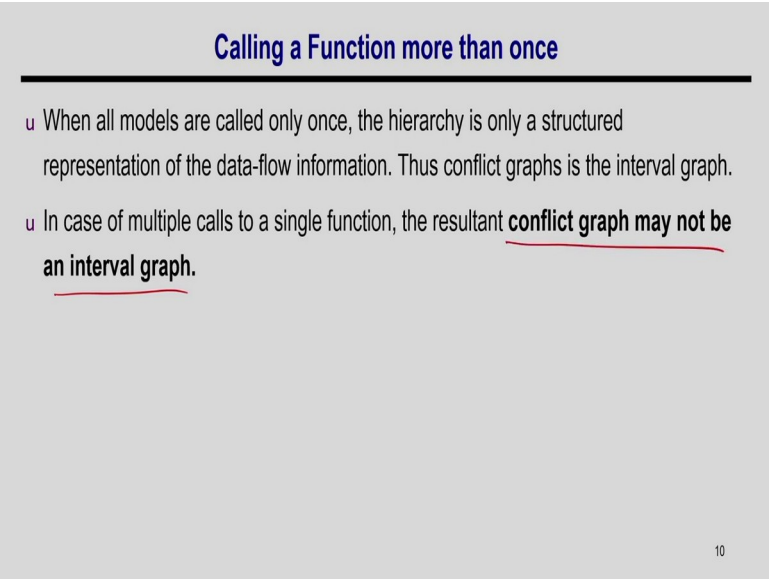
basically addition follows by multiplication, here multiplication follows by additions right.

So, this is function 1 right this is function 1 and this is function 2 will be like this that we can understand this right a = b * c and then x =a + y right this is this function, but if it is the same function then you can understand this would be plus followed by a multiplier. So, then I mean both can be shared right. So, let me. So, if I try to give the examples say suppose this is the function a function 1 and then again I am calling the function 1 only right. So, this is function 2.

So, in this case, I need one adder one mult. So, I can actually have one copy of the function here right one copy of the body and I can actually do a certain kind of sharing the I am multiplexing the input so, that I can actually execute both the addition operation for two calls using one adder and both the calls of the multiplier using one multiplier and here the example of two different function unit this is function unit 1, this is function unit 2.

And then it is addition followed by multiplication, here multiplication followed by addition and here I can say that even if there are two different functions I can share the adder between these two functions, but I need to do separate multiplier right this is what it talks about here ok.

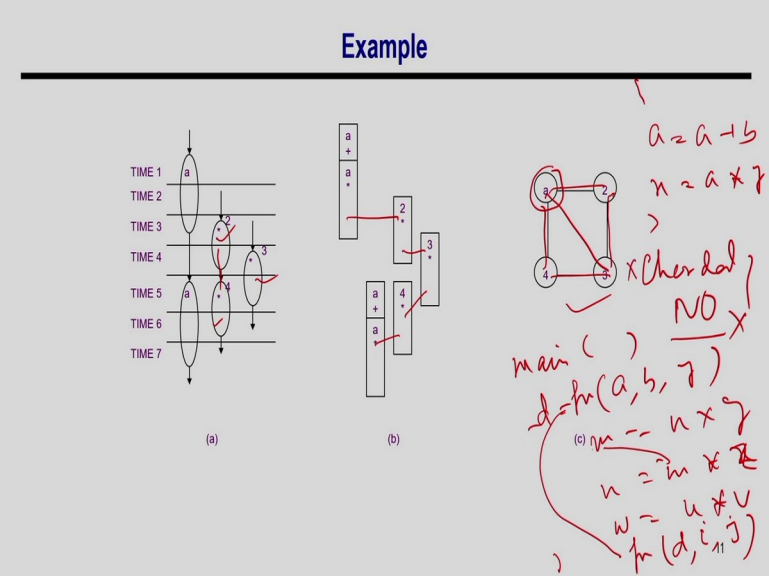
(Refer to Slide Time: 23:28)



So, let me move on, say suppose now I have multiple calls to the same function right. So, where they are overlapping ok so, the I discuss this scenario where there are multiple calls to the same function which is overlapping, but the operations I mean it can be shared right.

Now suppose I have the multiple calls to a function which is basically having multiple calls and in that case, what I try to say is that, although there is some kind of resource sharing is possible, creating an only your separate copy sometime it is difficult and it in that case particularly this particular graph will not remain an interval graph ok.

(Refer Slide Time: 24:09)



Let me explain what is that. So, let me take another example say suppose I have a function ok which is addition followed by multiplication. So, you have to say a =a + b, then say x = a * y right this is the function body and it is scheduled 2 times I mean it is called 2 times. In the first call happening here and the second call happening here and say in addition to that in the program there are some other multiplication operations are there which is possible right.

So, say suppose. So, the call is like this. So, suppose your main code looks like this. So, the main is like this that you have a function which you are calling with a, b and y, and then suppose you are doing some independent operation say m= n + q this is this operation and then say this is n = m * z right, and this third operation is said w = u * v ok.

So, this is and then you have another call here function which is it is returning the function say d and then I am calling the function with d say and then i and j right some function fn(d,i,j). So, basically, there is a dependency between these two functions because it is returning d and I am using the d here and there is a dependency between these two operations which is reflecting this and this is an independent operation right. I try to form this corresponding example here.

And now if you have this so, the point here I try to make is that if multiple calls this problem will not remain as an easily solvable problem right its polynomial-time not a polynomial-time solvable problem. So, suppose you have this scenario and say this is the corresponding intervals right. So, this is the first call it is scheduled in say in this time step and then this is when the multiplier is two cycles and addition is one cycle ok.

So, this is the interval corresponding to this operation which you can understand and now if you try to construct the conflict graph. So, there are two calls to this function there are two intervals, but in the conflict graph I will remain I will have a single node corresponding to this function why? Because I want to create a single copy of the function right so, if I try to keep multiple copies there is no point of having function call right.

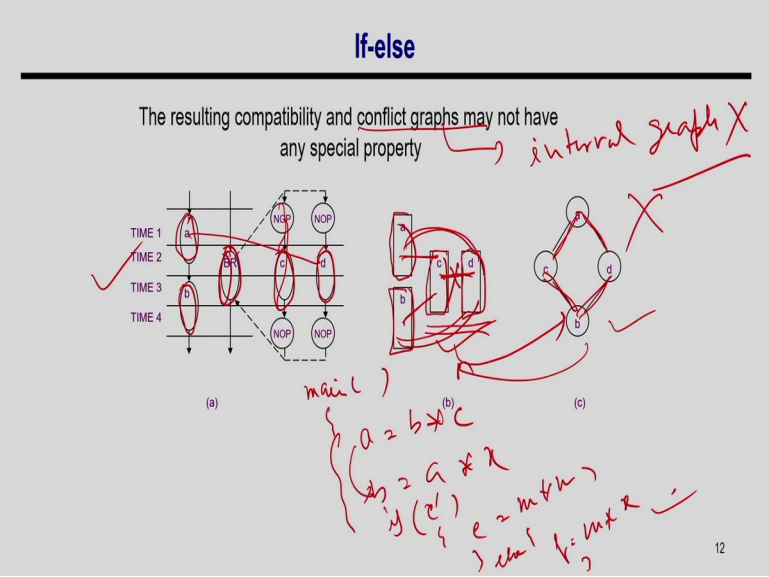
So, my objective is to make this function as a single copy. So, that is why I keep one copy in the node in the conflict graph and now there is overlapping from a to 2. So, there is an edge here 2 to 3 there is an edge here 3 to 4, there is an edge here 4 to a right. So, then this is my conflict graph, but is this a chordal graph? The answer is no right because there is no chord I have a cycle of 4 nodes, but there is no chord.

So, this is not a chordal graph and hence this is not an interval graph right. So, then if I try to solve this problem I cannot apply left edge algorithm ok. So, the summary here is that even if two function calls are have overlapping, I can share some of the resources. If there are two copy which is completely nonoverlapping and there is nothing else I can actually have a single copy definitely.

And it is the point that if there are two functions which have some overlapping still I have some sharing possible for some certain kind of operations that I have already discussed, but in general this problem when you have multiple call to a function and that there are some other operations, in that case, the whole problem will not remain as

polynomial-time solvable because the generated graph may not be an interval graph because it will not have satisfied the property of a chordal graph ok.

(Refer Slide Time: 28:02)



So, this is the summary of the function calls. Let me move on to the next part that if-else. So, I have already given an example that if-else is a single node and it has a branch like this, then again I am going to say that this conflict graph will not remain an interval graph. So, this will not hold even for if-else right. So, let me take an example that suppose I have call like this say suppose there are two multiplication operation, say, let me write this main again try to make this problem realistic.

Say suppose I have an operation a = b * c and then b = a * x right this is the things and then you have an if-else right. So, if some condition says. say, let us say some

if(c') { c = m * n}, else{ f = m * x} some arbitrary operation that to make ok. So, if this is the code and that schedule might be like this because I have a dependency from a to b.

So, I will schedule them in different time steps, but this if-else does not depend on any of these two operations right because they are all these operations are independent of these two operations. So, I can schedule it in parallel right. I can schedule in parallel to this operation suppose this is a schedule it is generated by a scheduler and where if-else it is the if brunch and this is the else branch and the if branch this c operation is schedule here and d operation is scheduled here because this is the else part right.

So, this is my else part operation so, this is my schedule graph and now let us try to put them in the interval. So, an interval is from 1 to 2, b interval is 2 to 3 is a 3 to 4, and c and ds in 2 to 3 right because these two operation scheduling 2 to 3. So, this is my intervals because this is how I get the intervals for these operations now let us try to conflict and draw the conflict graph ok.

So, now you see a and c have a conflict. So, I have an edge, I have c to d I have a conflict I will come to that. So, now, then c to b there is a conflict and then a to d also there is a conflict because they are actually having nonoverlapping at a to d and similarly b to d right now see here this c and d. So, let us consider this interval and now try to construct the conflict graph ok.

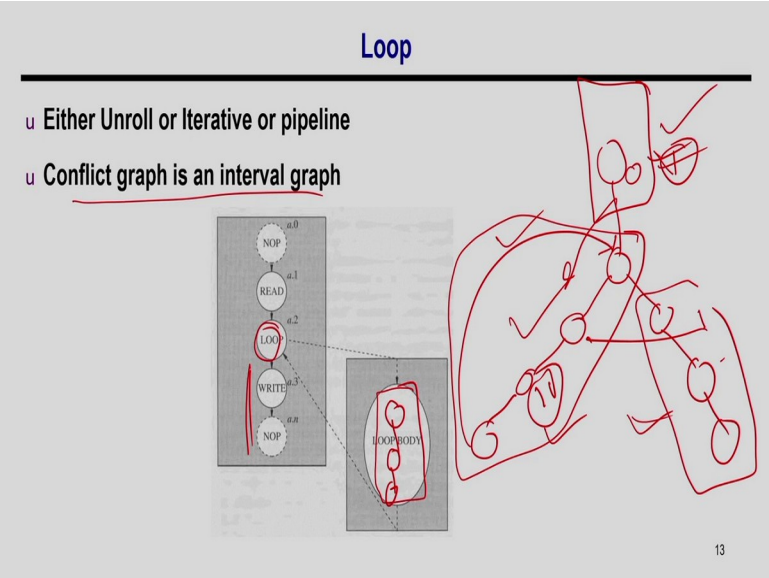
So, now you see here there is one node for each interval and a and c there is an overlap. So, there is an edge here, b and c there is an overlap there is an edge here, a and d there is an overlap. So, there is an edge here b and d there is an overlap. So, there is an edge here. So, what about this edge? So, there is an overlap here. So, we should have an edge here to make this graph.

But this is not a conflict because these two operations are mutually exclusive right when you have if and else, I either will execute this operation c or I will execute the operation d I am not going to execute both of them together right that is that means, they are mutually exclusive. So; that means, that although they are intervals having overlaps, but the corresponding edge will not be there in the conflict graph right so; that means, this interval does not reflect this conflict graph right.

And hence I am not able to map this node into intervals such that whenever there are two intervals. So, overlaps then there is an edge here. So, that particular property does not hold right. So, as a result, this conflict graph that I obtain is not an interval graph right and if it is not an interval graph, I cannot solve the problem in polynomial time. So, I have to go for the heuristic algorithm ok.

So, the summary of this discussion is for if-else I cannot solve this problem in polynomial time or using left edge algorithm I have to go for heuristic algorithm ok. So, now, the last part is the loop.

(Refer Slide Time: 32:05)



So, as we have discussed that loop we have a loop body and which is something have another sequence graph because of the loop the structure is a little bit important to understand. say, this loop is executed in 3 cycles ok. So, this loop body has say 3 cycle operations ok.

And this is getting repeated ok. So, now, you see here in the combined graph whenever I am going to execute the whole thing in hardware. So, whatever the operation followed loops right. So, these are the operation is this, this will have to wait till this loop is done right. So, this loop has to execute separately, and then everything has to wait right, and similarly whatever the previous part is done then only the loop will start right.

And then so, what I am trying to mean is that this loop body has no overlap with the operations previous and the operation that is scheduled later which was not the case for if-else. Because in the if-else it because they may be independent and they are actually having overlapping. So, I can put them here right. So, the loop usually what do we do? We will put this as a separate entity and hence there is no overlap between this operation with this part and this part.

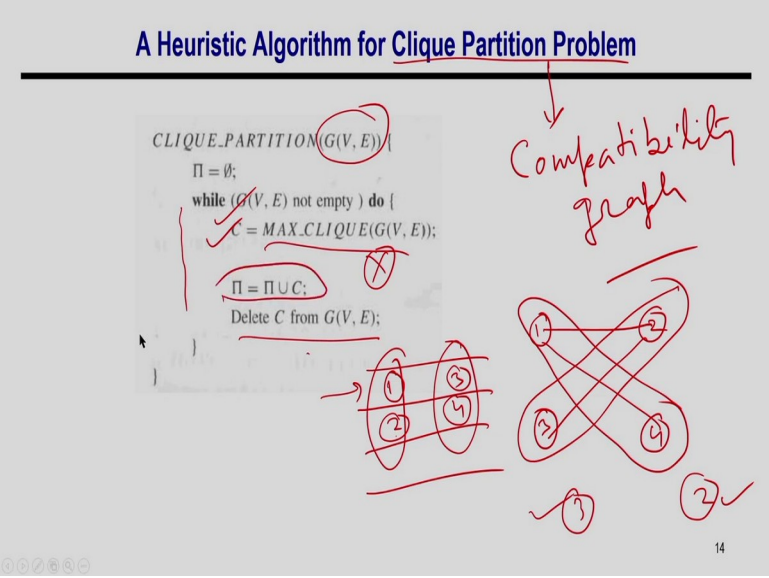
So, I can take this whole thing together and I can actually solve the problem and then in that case conflict graph will remain an interval graph and so, we can solve the problem with a loop. But remember this is only for the loop and if I try to perform the loop independently on the other part ok since this is a repetitive part I should not repeat

anything here or here because I want to execute this part only once, this part says 10 times. So, I cannot just repeat any operation here with this or any operation here with this.

So, then that particular operation will execute 10 times because this part will be repeated 10 times or 20 times right. So, that is why I can actually identify the resource sharing and then I am going to repeat this part 10 times. So, I can actually control this from the controller FSM ok.

So, this is what about the loop. So, we have already discussed the three construct the conditional construct like if-else loops and function call and we also discuss what is going to happen in the sequence graph in the result and conflict graph if we consider all of them together ok. So, this part is done.

(Refer Slide Time: 34:25)



So, I am going to conclude this particular class with a heuristic algorithm for the clique partition problem ok. So, we have discussed that this problem still remains it is not an interval graph then this graph coloring problem becomes an NP-complete problem.

And in the compatibility graph finding the clique cover is remain NP-complete problem right. So, in that case, I could not have an efficient algorithm to solve the problem, but we should have some heuristic algorithm right. So, I am going to discuss one such

heuristic algorithm for the clique partitioning problem which is applicable to the compatibility graph right.

This actually applies to the compatibility graph and complement graph of this where we use the graph coloring problem, but since the problem is dual I can solve one of the problems it solved both right. So, what is this clique partitioning problem? The problem is that you have given a compatibility graph right. So, compatibility graph is what? It has a set of nodes corresponding to each operation and whenever there is an operation that is actually is compatible; that means, there is executed in different time step there is an edge.

So, if you have like this; that means, say operation 1 2 3 and 4 then 3 and 2 and 1 and 2 is executed in different time step 2 and 3 is executed in different time step and 1 and 4 is also executed in different time step right. So, you can actually map this operation to some schedule let me try to put it here. So, 1 and 3 are happening here say 1 and 3, 2 is happening here and 4 is happening here right.

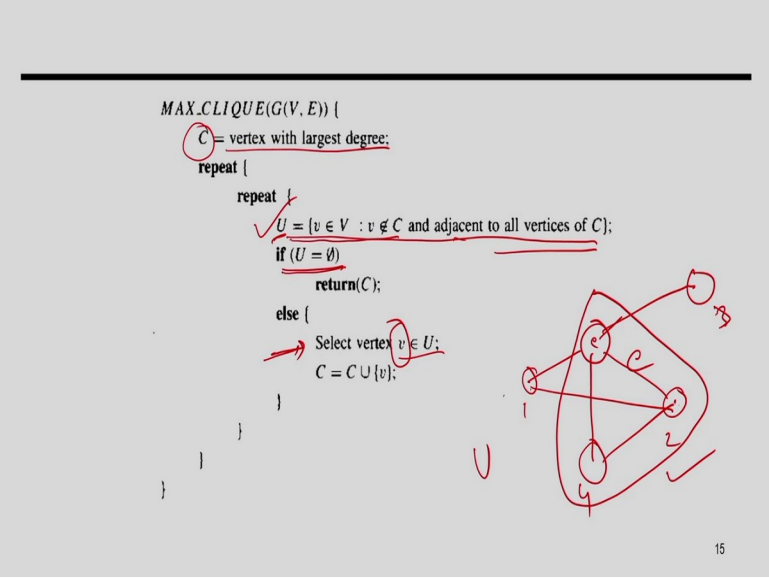
Then I can have an edge from 1 to 2 and 1 to 4, 2 to 3 because 2 and 3 and 1 to 2 right. So, I understand this. So, this is what is the schedule ok. So, if this is there I try to find out the clique. So, I can actually find a clique of these two nodes because there are two nodes that are interconnected to each other and I can actually find this one and this one right.

So, I need basically if I go this way I need basically 3 functional units. So, these are all adders right. So, I need 3 adders, but if I try to partition this way say suppose I club these two together and this is the two cliques then I need two adders. So, you can understand that for the same problem different clique coverage give you different FU requirement. So, identifying the mean also for this you need two right.

Because I need one to execute this part and say this part right basically there are two operations running in parallel. So, I need at least two such function units ok. So, if we apply the heuristic base algorithm, then you may not always get this optimal one you might get this one, but you get a solution, but most of the time solution is good ok. So, let me explain this.

So, I am going to this as my top-level algorithm. So, I have given this complement graph and I find out a maximum clique right. I found a clique from that and then I remove all the nodes part of this clique from the graph and for the rest of the graph I am going to do the same thing until my graph becomes empty ok. So, this is what the overall this high top-level algorithm and let me understand this is the important problem how to find out the max clique right.

(Refer Slide Time: 37:44)



So, the heuristic that I am going to use is very simple what I am going to do it here is, that I will find a vertex that has the highest degree. So, I have a graph now which has the I find out the node which has the highest number of an outgoing edge right I will select that node say then what I am going to do? And so, this is my C now. So, C now I have a single node I will find out all the node which is adjacent to all the vertices in C.

So, initially, I have seen only one node say among all the nodes that have connected to that I going to consist suppose these are the node that has an edge to C right. If there is no such edge I will return right I will return to see this is my maximum clique and then what I am going to do? Among all these nodes I will put all these nodes this was said 1, 2, and 3 into the set U right.

I am going to put all the nodes into this U ok and then I am going to select a vertex from U and then I will make this I will increase the C suppose I have selected this one right. So, I have selected this one then this now becomes my C right because this is a clique of

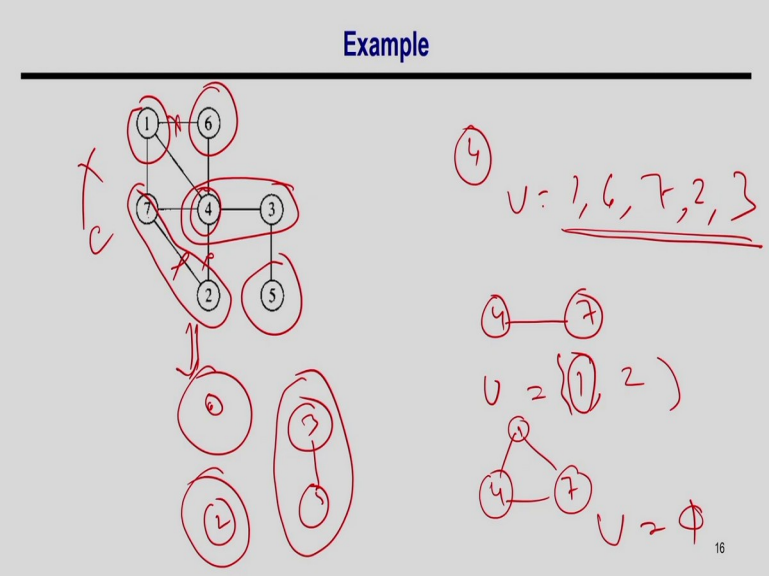
two nodes right. So, this is done then I will go back again then I am again I am going to find out all the nodes which have edges adjacent to all the vertices not only one vertex.

So, now I am going to select this node suppose it also had an edge here and say this node had also and edge says 1 and say 4. So, then I am going to select 1 and 4, not 3 because 3 has an edge to this node, but there is no edge to this node ok. So, next iteration I am going to select all such node which is adjacent to all the vertices of C so; which means, I can add any of them right because this will also form a clique of 3.

Now, say I have selected 4, now my graph this C become this right. So, my C become this. So, it is a complete subgraph of 3 nodes. So, this is why I am going to grow the graph or the subgraph complete subgraph until I found there is no such e edge no such node which is connected to all the nodes right then this is my maximum clique.

So, this is what it will return, and then once I come back to this function. So, I got a clique I will remove all the nodes of that subgraph from the graph, and then I will call again right.

(Refer Slide Time: 40:06)



So, for the remaining graph I am going do these things ok. So, let me take an example and explain. So, suppose this is my say comparative graph right. So, what the algorithms find the largest node. So, what is the node with largest degree is 4 right.

So, I will select 4 and then I am going to identify all the nodes that had the edge to 4 right. So, you can find that the nodes 1, 6 7 everybody is except 5, 2, and 3. So, they are all connected to 3 right. So, now, I have to select say arbitrary one of them since I have selected 7. So, my C become this ok now I have the 4 and 7, this is sorry this is I have 4 and 7. So, this is my this is the C now right so this is my C now and I am going to find out all the nodes which is all connected to both 4 and 7.

So, I have one which is both connected to 4 and 7. So, one will come here and 2 also connected to both 4 and 7 right. This is that only two-node which is connected to both 4 and 7 not any other node ok. So, from there say I have selected arbitrarily one. So, now, my C becomes this right. So, this is my C. So, now, I have these 4, 7 and 1. So, this is the clique I have obtained now and I try to find out U which is connected to all the nodes of this.

And there is no such node that is connected to all 4, 7, and 1 right. So, there is no node. So, this will become 5. So, I will return so; that means, I found a clique with 3 nodes right. So, now I am going to remove this one 4 and 7 from the graph because this is already removed. So, now, I will remain the graphs 6, 2, 1, 3 and 5. So, now, I am going to take a node which is the highest degree and which is maybe because 3 this node 3 also has degree 2.

Two will also degree 2, 6 also has a degree 2 and this node does not exist and the corresponding edges also do not exist right. So, ok so, the only node has that degree 2 is basically 3 because this edge does not exist this edge does not exists these are not degrees. So, after this, my remaining graph will look like this 6, 2, 3, and 5 right this is the remaining graph because this part is already done. So, this is my remaining graph and I can see only the node that has a degree is 3.

So, basically in the next iteration this will find a clique and then this will be another clique and this is another clique. So, I need basically for I have 4 cliques. So, this is one clique and this is one clique and this is one clique right. So, you can actually see that. So, I need basically 4 colors and this is the mapping. Now here is why this algorithm is not optimal which I will explain.

Say suppose here I selected. So, in the first iteration, I found that 4 is the node and there are these all these nodes are basically adjacent to that and I have selected 7 right, but if I

select 3 in that time. So, this will become my clique right because I cannot add more right, and that way I am not getting the maximum clique of size 3 right.

So, it may end up that in some cases you are having more cliques right. So, probably you might end up having this is one. Say, this is one and this is one right. So, so in end I mean you might end up having had like this. So, what I am trying to say is that, since this is a heuristic algorithm and this selection has no I mean there is nothing to decide which vertex to select is there any way you can minimize the size.

So, this algorithm may not always give you the minimum number of cliques to cover the complete algorithm ok. So, this is an efficient algorithm because it runs in b plus c and it can give you the cliques in a quick time ok, but the solution may not be the optimal one ok. So, with this, I conclude today's discussion.

Thank you.