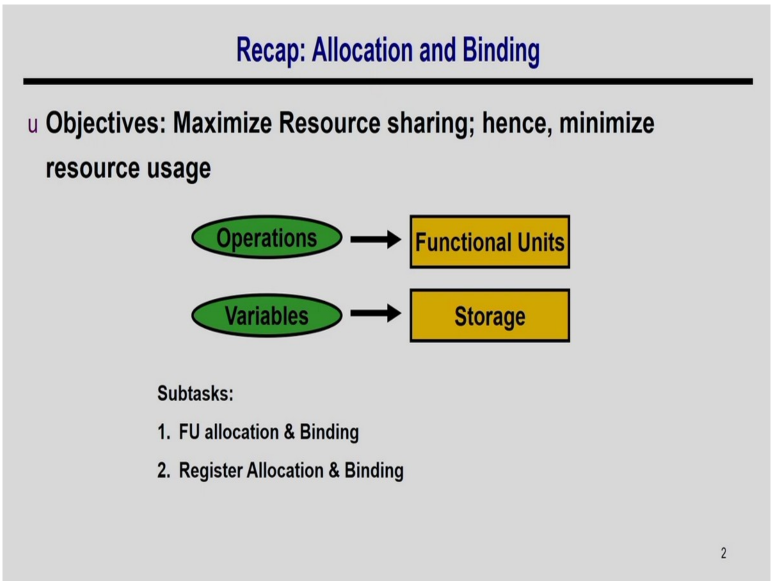
C-Based VLSI Design

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 - 17
Resource Allocation and Binding: Left edge Algorithm

Welcome, everyone. So, we will continue to discuss our topic Resource Allocation and Binding. So, just to recap what we have discussed in the last class.

(Refer Slide Time: 00:55)



Recap: Allocation and Binding

- Objectives: Maximize Resource sharing; hence, minimize resource usage

Operations
Functional Units
Variables
Storage

Subtasks:

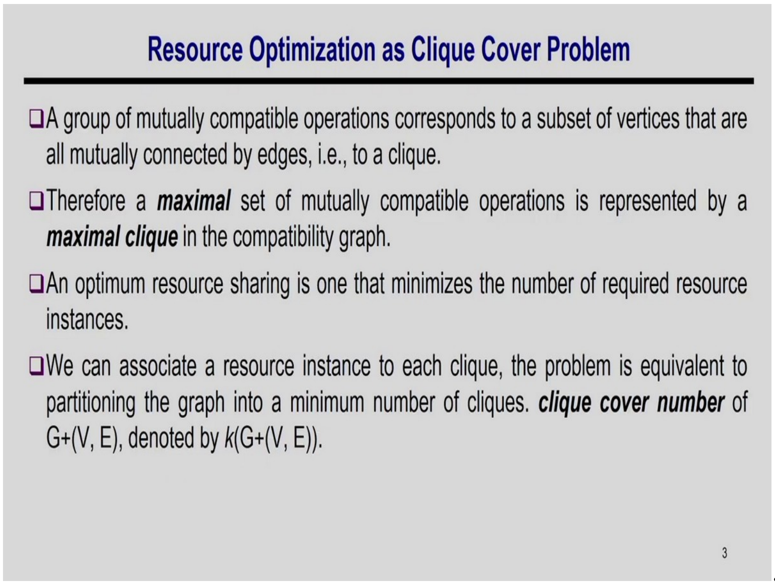
1. FU allocation & Binding
2. Register Allocation & Binding

2

We found that in the high-level synthesis when the scheduling is over; that means, we know which operation to be executed in which timestamp, the next step is to find out the resource or the hardware resource we got right. And specifically, two types of resources you have to determine the first one is how many functional units you needed and the second one is basically the storage which is to store the variables right.

So, the operations of the behavior will be going to be mapped to the function units and the variables are going to be mapped to the storage unit. So, we found that this particular resource allocation problem, in general, can be mapped to a clique cover problem.

(Refer to Slide Time: 01:32)



Resource Optimization as Clique Cover Problem

- A group of mutually compatible operations corresponds to a subset of vertices that are all mutually connected by edges, i.e., to a clique.
- Therefore a ***maximal*** set of mutually compatible operations is represented by a ***maximal clique*** in the compatibility graph.
- An optimum resource sharing is one that minimizes the number of required resource instances.
- We can associate a resource instance to each clique, the problem is equivalent to partitioning the graph into a minimum number of cliques. ***clique cover number*** of G+(V, E), denoted by *k*(G+(V, E)).

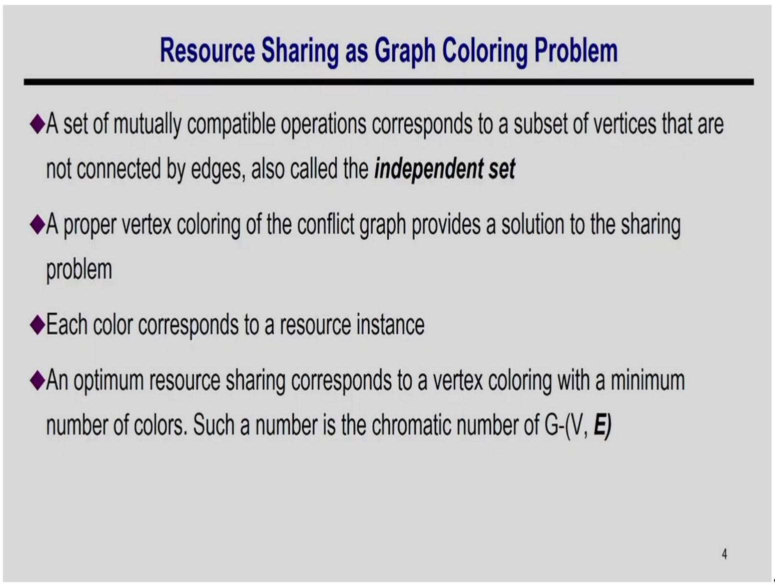
3

.

So, what we can do we basically we found that specifically for a FU functional unit allocation and binding that if two operations are actually running in parallel; that means, they are scheduled in the same timestamp they are not compatible with each other; that means, we need a different kind of resource to execute these operations right. So, by taking the compatibility of the operations we can actually construct a graph and each node basically represent an operation. And then their edge represents the compatibility.

And then we actually conclude that finding the cliques, the maximum clique of that particular graph or the number of cliques that are needed to cover all the nodes of the graph is nothing, but the FU requirement and each clique represent one FU and all the nodes within the clique represent I mean can be map to that particular FU right. So, that we have understood.

(Refer Slide Time: 02:33)



Resource Sharing as Graph Coloring Problem

- ◆A set of mutually compatible operations corresponds to a subset of vertices that are not connected by edges, also called the ***independent set***
- ◆A proper vertex coloring of the conflict graph provides a solution to the sharing problem
- ◆Each color corresponds to a resource instance
- ◆An optimum resource sharing corresponds to a vertex coloring with a minimum number of colors. Such a number is the chromatic number of G-(V, ***E)***

4

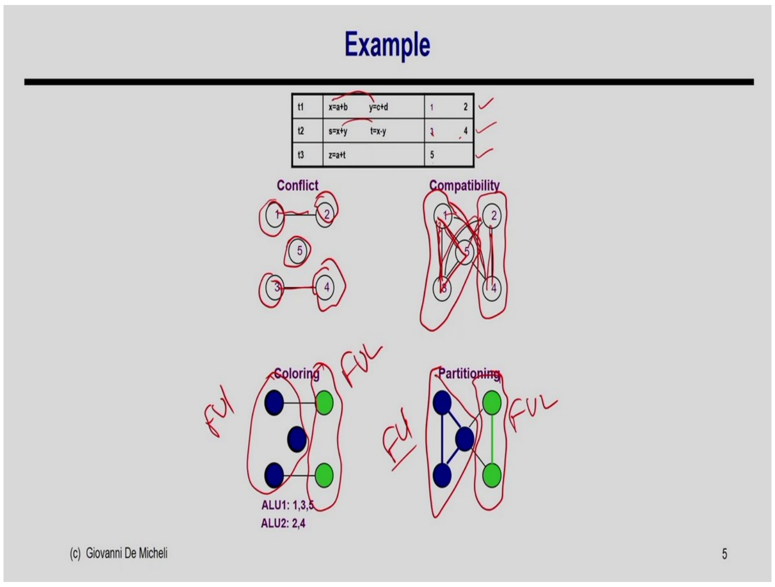
.

Similarly, we have seen that the same problem of this resource sharing can be mapped to graph coloring problem, in the graph coloring problem what we have seen that we can construct the similar kind of graph where each operation will represent a node, and then if two operations are conflicting to each other; that means, they are actually executing the same timestamp I am going to add an edge. So, then it will become a conflict graph and which is basically the complement graph of the compatibility graph.

And then what we found is that it is basically if you just try to color the nodes of the graph and then if the nodes have the same color they cannot be adjacent. If that is the logic we apply then the minimum color needed to color all the nodes of the graph is nothing, but the resource requirement right.

And each color represents an FU and all the nodes that have the same color actually can be mapped to the same FU in hardware right. So, that is something we understand is very interesting thing the resource allocation and binding can be map to either clique cover problem or graph coloring problem ok.

(Refer Slide Time: 03:34)



Example

t1	x=a+b	y=c+d	1	2
t2	s=x+y	t=x-y	3	4
t3	z=a+t		5	

And then just to give an example that suppose we have this schedule like this and then in the schedule. So, there are five operations and then the operation 1 and 2 schedule in timestamp 1, 3 and 4 operations is scheduled in timestamp 2 and 5 is of scheduling in operation I mean time stamp 3.

So, we have to consider the operations of same type at a time because we cannot map a multiplier and adder to the same FU right, no point taking multiplier mapping and adder mapping together. So, we are going to consider the multiplication mapping once I have done the problem solved I mean and then you will go for the mapping of the adder and so on right.

So, here all the operations are ALU type of operation which is addition and subtraction ok. So, what is the conflict graphs? It is basically this is the conflict because operation 1 and 2 is executing in the same timestamp so, there is a conflict here. And then you have this operation 2 and 3 is happening 2 or 3 and 4 is happening here so, there is a conflict there is no other conflict right.

So, in this graph, if you try to color the nodes I can give the color like this 3 nodes the same color because 3 nodes are not adjacent to each other and I can give the same color. So, that is what I have shown here. So, this is basically my FU 1 ok and I can put the color green to this node and this node and this is my FU 2 right and then this FU 1 is going to execute the operation 1, 5 and 3 and FU 2 is going to execute operation 2 and 4 right. So, this is the graph coloring problem reduction of this resource allocation and binding to the graph coloring problem.

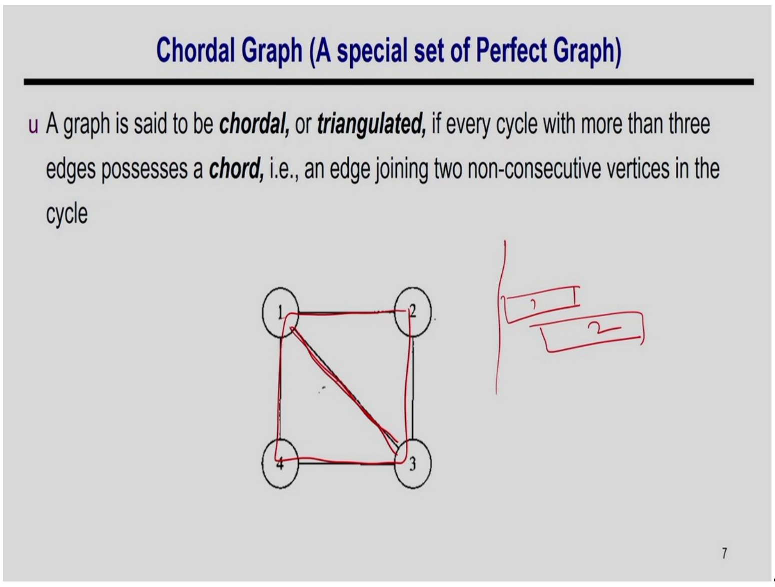
Similarly, we can actually construct the compatibility graph here it is basically the edge represents the operation compatible to each other not, we can see here that operation 1 it is compatible to operations 3, 4 and 5 because they are running in different timestamps. So, from 1 I have an edge to 3, 5 and 4. Similarly, operation 2 is compatible to operations 3 4, and 5. So, this way I actually construct this graph.

And then if the there is a clique this is a complete subgraph I can find a complete subgraph of 1, 3 and 5 we can see there is a complete subgraph here so; that means, these all nodes are compatible to each other and they can map to a single FU and that is a maximal clique.

So, I found a clique of 3 nodes and again this is representing my FU 1 and then I found that from the rest of the nodes I found another clique of 2 nodes there is an edge between them. So, this is my second clique. So, 2 cliques cover the whole compatibility graph and one clique represents FU 1, the second clique represents FU 2, and same I can see that the results are the same in both cases right. So, this is how we have discussed the problem can be mapped to either clique cover problem or graph coloring problem.

Then we have found that in general, the problem of this graph coloring or clique cover problem is basically NP-complete; that means, we do not have an efficient algorithm that means, a deterministic polynomial-time algorithm to execute the problem. So, usually, we go for a heuristic algorithm.

(Refer Slide Time: 06:51)



Chordal Graph (A special set of Perfect Graph)

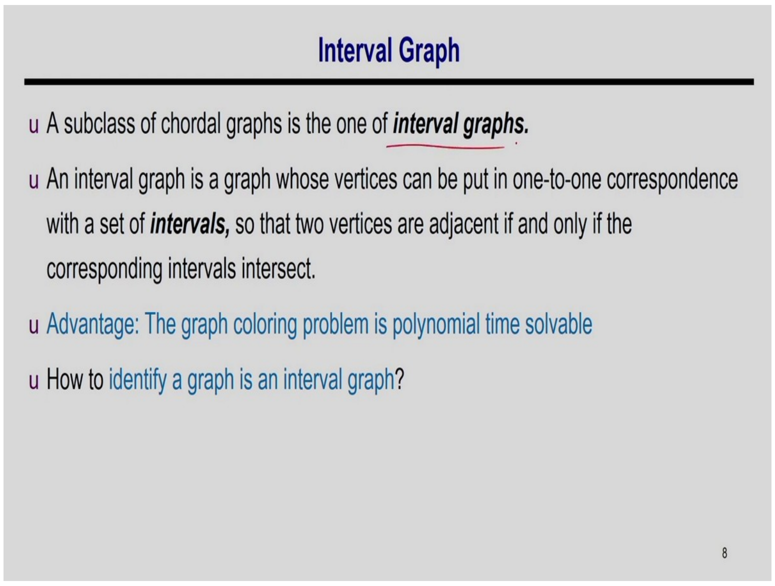
- A graph is said to be ***chordal,*** or ***triangulated,*** if every cycle with more than three edges possesses a ***chord,*** i.e., an edge joining two non-consecutive vertices in the cycle

7

.

Then we discuss that specific set of graphs and specific class of problem this problem is actually polynomial-time solvable. And we found that if the conflict graph is basically chordal; that means, if you have a in the conflict graph if there is an edge cycle of at least 4 nodes then always you have an edge chord right. So, this is a cycle of 4 nodes we have a chord there right.

(Refer Slide Time: 07:17)



Interval Graph

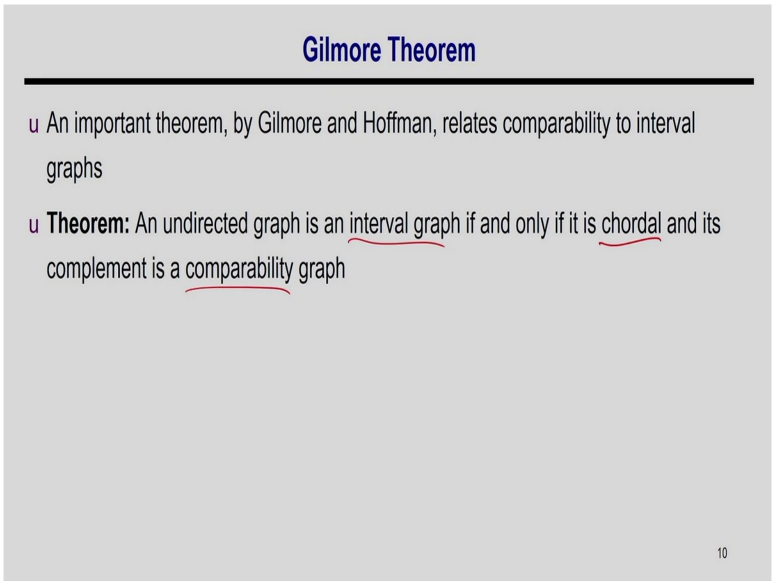
- A subclass of chordal graphs is the one of ***interval graphs.***
- An interval graph is a graph whose vertices can be put in one-to-one correspondence with a set of ***intervals,*** so that two vertices are adjacent if and only if the corresponding intervals intersect.
- Advantage: The graph coloring problem is polynomial time solvable
- How to identify a graph is an interval graph?

8

So, if we have if that particular conflict graph is chordal and then we found that there is another subset of the chordal graph is an interval graph. So, which says that if we can map this node of this conflict graph into some intervals such that if there is an edge between these two nodes. So, then their interval will overlap. So, for example, here 1 and 2, I just suppose I just map them into this interval. And then I show that this is because there is an edge difference between 1 and 2 there is overlap.

So, if I can map all these nodes to such interval such that whenever there is an edge the corresponding intervals will overlap then I can tell this particular graph is an interval graph, and then we have what is the advantage of this, we found that if that conflict graph is an interval graph and I can solve the problem in polynomial time ok.

(Refer Slide Time: 08:09)



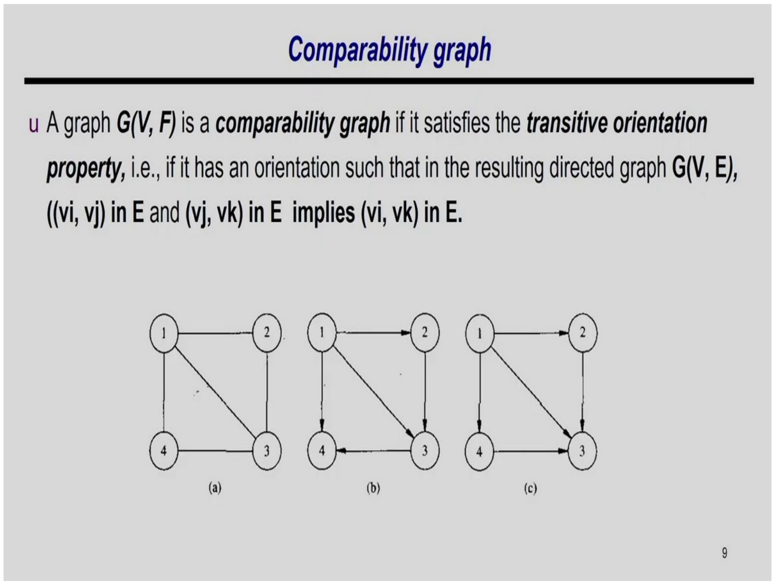
Gilmore Theorem

- An important theorem, by Gilmore and Hoffman, relates comparability to interval graphs
- **Theorem:** An undirected graph is an interval graph if and only if it is chordal and its complement is a comparability graph

10

So, in general, so what we found by this Gilmore theorem is that if my conflict graph is an interval graph if and only if it is a chordal graph and its complement graph is a comparability graph right. So, comparability graph in the sense that so it has a transitive orientation right.

(Refer Slide Time: 08:22)



Comparability graph

- A graph ***G(V, F)*** is a ***comparability graph*** if it satisfies the ***transitive orientation property,*** i.e., if it has an orientation such that in the resulting directed graph **G(V, E*)*,** **((vi, vj) in E** and **(vj, vk) in E implies (vi, vk) in E.**

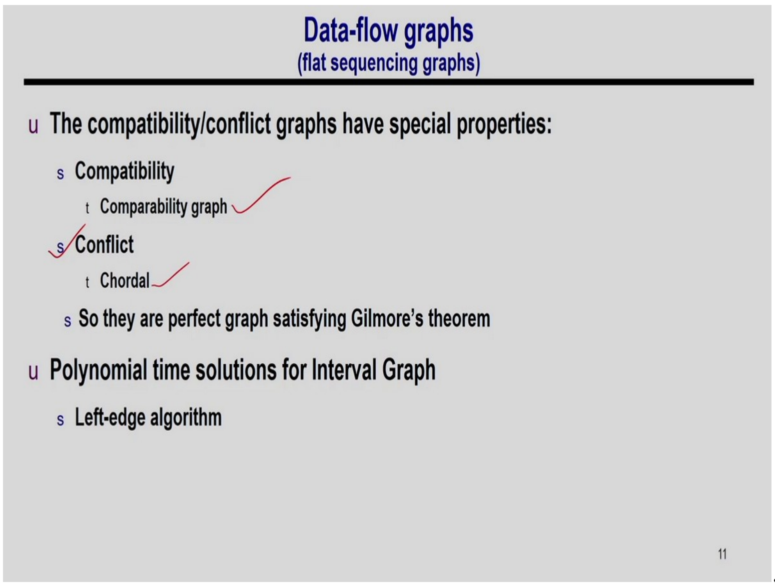
9

So, although this compatibility graph is an undirected graph I can add direction to the edges. So, I can make this particular graph as an undirected graph to a directed graph and then whenever there is an edge from i to j and j to k then I have an edge from i to k. So, the transitive property holds by this transitive orientations right then I will say that particular graph is comparability graph. And we found that if one graph is an interval graph if and only if that graph is basically chordal; that means, it has a chord and its complement graph is a comparability graph; that means, it has transitive orientations right.

So, the conclusion of this discussion is if I found a particular graph an interval graph then I have an efficient algorithm right polynomial-time algorithm and to identify a graph is an interval graph or not I have to apply this theorem I have to check whether that particular graph is a chordal graph or not and its complement graph is comparability graph or not right. If it is that then it is an interval graph and if it is an interval graph I have an efficient algorithm which I can apply to solve this graph coloring problem ok.

So, then we discuss that if we take only one basic block at a time right because during scheduling we discuss most of the time that you take one basic block of the whole chord at a time and then try to schedule that and then you combine all schedule together to get the combined the overall schedule right.

(Refer Slide Time: 09:47)



Data-flow graphs
(flat sequencing graphs)

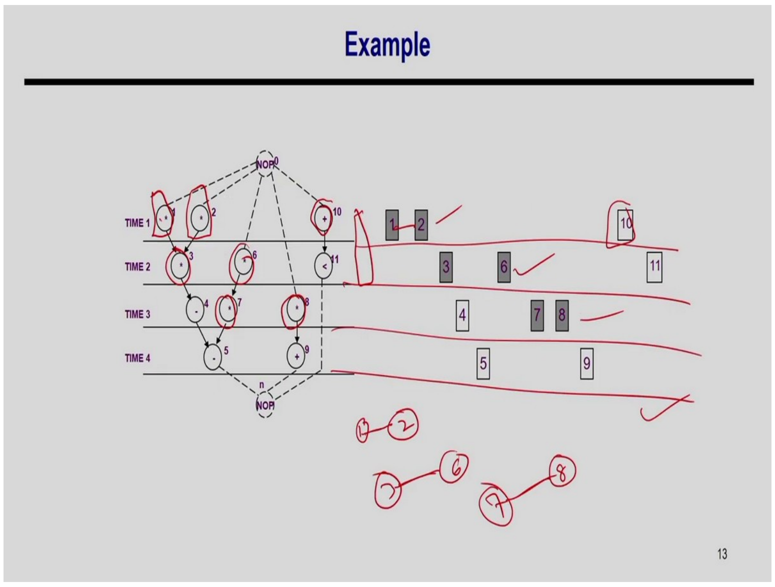
- The compatibility/conflict graphs have special properties:
  - Compatibility
    - Comparability graph
  - Conflict
    - Chordal
  - So they are perfect graph satisfying Gilmore's theorem
- Polynomial time solutions for Interval Graph
  - Left-edge algorithm

11

So, now, if you try to solve this resource sharing along with allocation problem for a particular block or a particular basic block then this particular problem is actually polynomial-time solvable. Specifically, that conflict graph if we construct the conflict graph for a particular basic block then that particular graph will be a chordal graph and then its compatibility graph which is the complement of the conflict graph which is the comparability graph right.

So; that means, if we try to solve the problem of resource allocation and binding for a specific basic block I can solve it easily and we have discussed that we can actually apply this left edge algorithm for that to solve this problem ok. So, we are going to discuss this left edge algorithm in this class in detail so, but before that, I try to show you that what is the interval graph looks like for a basic block right.

(Refer Slide Time: 10:40)


Example

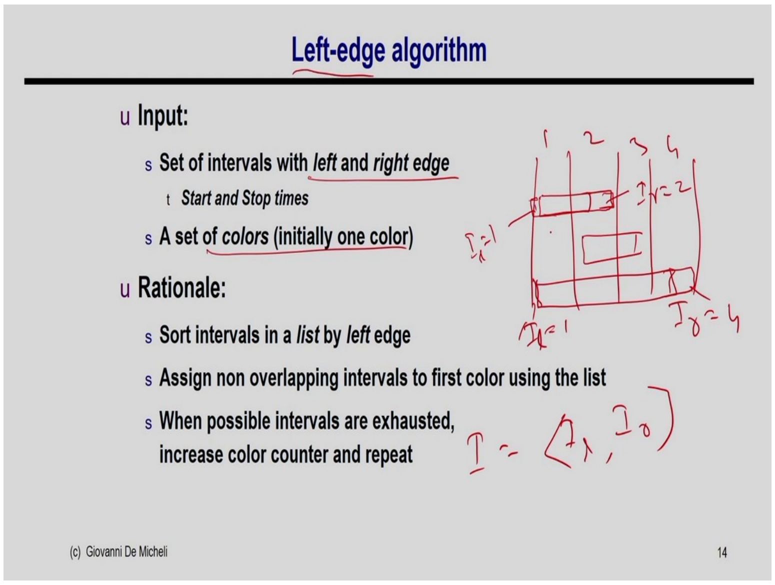
So, if you consider this is the operations in a happening in one basic block, in the basic block means there is no control flow there is no loop it is a sequence of operations right and this is the corresponding schedule that I have obtained from scheduling and then wherever the operation is scheduled that is the interval right. So, this is the interval of this operation. So, this is the interval of operations 1 and 2, this is the interval of operations 3, 10 and so on.

So, now I can consider this is the intervals and you can see here because 1 and 2 is conflicting to each other and their intervals is overlapping right, 1 and 3 is not conflicting to each other their intervals are not overlapping right. So, this way I can show that this is the corresponding intervals corresponding to given this schedule and if I construct the conflict graph here say for the multiplier.

So, this is my 1, this is 2 and say this is 3 and this is 6. Because I have only 6 multipliers, 3 and 6 and 7 and 8 rest of the operations are ALU I am not going to consider them and then say 7 and 8. So, 1 and 2 is conflicting 3 and 6 is conflicting and 7 and 8 is conflicting; obviously, it is a chordal graph because there is no chord or there is no loop and then we can see here is that this 1 and 2 have a conflict so, their intervals is overlapping 3 and 6 has a conflict so, their interval is overlapping 7 and 8 is conflicting. So, their intervals is overlapping ok.

So, this is how I can get the intervals right for a given schedule I can identify the intervals of each operation based on their schedule, it may be noted that if one operation is scheduled for multiple cycles its interval will be over more than one cycle ok. So, this is how I will get the intervals of that node.

(Refer to Slide Time: 12:29)



Left-edge algorithm

- Input:
  - Set of intervals with *left* and *right edge*
    - *Start and Stop times*
  - A set of *colors* (initially one color)
- Rationale:
  - Sort intervals in a *list* by *left* edge
  - Assign non overlapping intervals to first color using the list
  - When possible intervals are exhausted, increase color counter and repeat

(c) Giovanni De Micheli 14

1 2 3 4
$I_{r}=2$
$I_{l}=1$
$I_{l}=1$
$I_{r}=4$
$I = (I_{l}, I_{r})$

And then I will move to the left edge algorithm. In the left edge algorithm as it suggests it actually can solve this problem in polynomial time, what is the problem graph coloring problem? So, what is the input to my algorithm, we have a conflict graph and hence the intervals are right. So, because I know what is the interval corresponding to the schedule. So, I have both this conflict graph as well as the intervals. But this particular algorithm takes the interval as an input ok.

So, I have given the set of intervals and what it will return, it will give you a set of colors right. So, it will give you a set of colors say 3 colors or 4 colors, and also mapping of each node or each interval to that color. So that means if I know that interval 1, 5, and 7 have the color rate; that means, 1, 5, and 7 is going to be executed using the same FU ok. So, it solved the graph coloring problem on the intervals given ok.

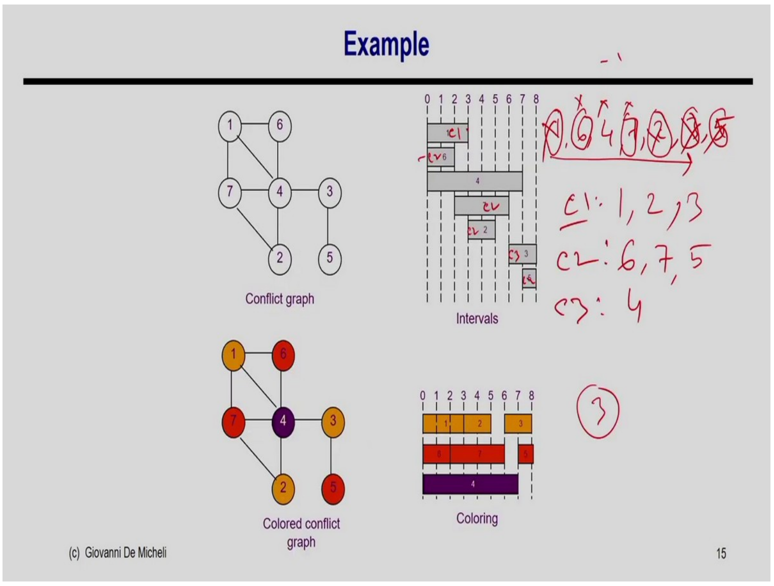
Another question is how I am going to represent the intervals right. So, the intervals are represented for this algorithm as a left and right edge right. So, if you see here so, if I just say this is my time right. So, I now actually in for the previous example it was in the top to bottom direction, now I am going to make it left to right direction you can always rotate it right.

So, now I can say that these is my intervals this is interval 1, 2, 3, 4 and so on and this is how I am going to represent the intervals right and say every interval can be represented by two things right; it is left edge and the right edge. So, this is I_l interval and this is I_r right. So, I can represent an interval I by I_l and I_r right. So, this is how I can represent an interval.

So, given the intervals, I will represent the interval by two the coordinates of their left edge and right edge ok. So, the interval of for example, for this particular edge is equal to 1 and it will be always complete. So, this is basically I_r equal to 4, for this it is basically the interval is it will be complete right. So, this I_l equal to 1 and this is IB equal to 2 right. So, this is the interval.

So, now what is what this algorithm does apparently, it first sort this interval based on their left edge right, as the name suggests so it actually sorts these intervals based on their left edge ok it first does that.

(Refer Slide Time: 15:07)


Example

So, let us take an example first, suppose this is my intervals given and you see here this is the sorted order right. So, given these intervals, this is my conflict graph, and say this is the corresponding interval because I have this is the input.

So, I have sorted the things the sorting order is 1 then 6, then 4, then 7, then 2, then 3, then 5 you can see this is the sorting order of this interval based on their left edge ok. So, now, what this algorithm does? It takes a color 1 right. So, it takes color 1 and it scans this list from left to right and identifies the first node. So, initially, you can assume that none of the intervals is selected.

So, what I am going to do? I am going to select the first interval from this list whose left edge is not overlapping with the previously selected one right. So, if you take the first iteration I can assume that the previously selected one is minus 1. So, I can select 1 right I can select this one because yeah it was previously selected was none. So, I can assume that the right edge is minus 1.

So, I will say that this particular I am going to select this 1 because it is not overlapping with the previously selected interval ok. So, I have selected 1, then my now current selected one is 1, now I will go to the next one I will see whether that particular interval on the left edge it has an overlap with the previously selected one or not right. So, how can I check that? If the left edge is greater than the right edge of the previously selected one right.

Since here the left edge of 6is not greater than of 3 of the previously selected one($6_l < 1_r$), I am not going to select 6 in this iteration, I cannot select 4 for the same reason, I cannot select 7 because the same iteration because the left edge of 7 is not greater than of right edge of 1 right($7_l<1_r$). So, I am not going to select.

But I can select 2; because the left edge of 2 is greater than equal to the right edge of the previously selected one($2_l>=1_r$). So, I am going to select 2 as well. So, I have selected 1, I have selected 2 once I select 2 I can go to the next one it is 3 and I can select 3 because the left edge of 3 is greater than equal to of right edge of 2($3_l>=2_r$). So, I have selected this one, I have selected this one, I have selected this one.

So, 3 if I select I once I select 3 I cannot select 5. So, there is the 3 node which I can put color 1(C_1) and they have non-overlapping interval so; which means, they can be a map to the same FU right. This is how every iteration it scans the list and identifies the intervals that can give the same color ok. Once this three are selected I will remove them from the list ok. So, the same process is going to execute for the remaining iteration.

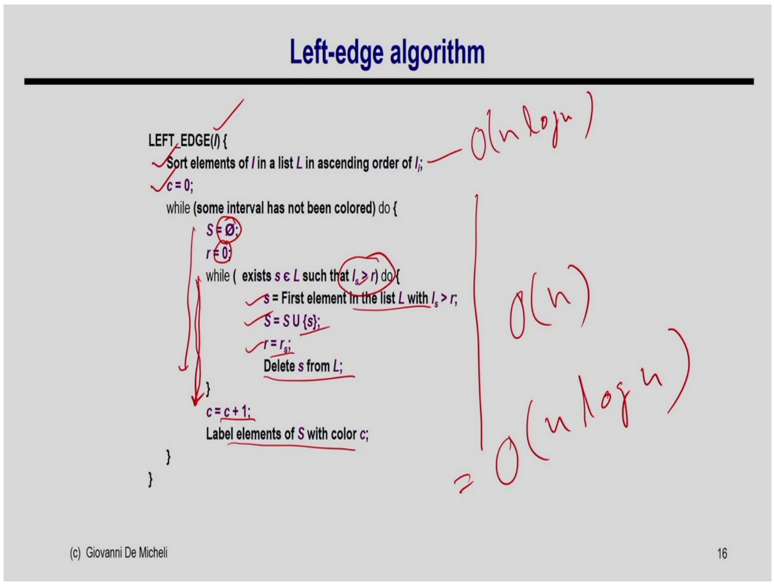
So, now, I will select the color 2 and I am going to select the color 2, I will select 6 because this is the first one I select 6 and then once I select 6 I cannot select 4, I cannot I can select 7 because 7 left edge is greater than equal to the right edge of the 6($7_l >= 6_r$). So, I can select 7 and then once I select 7 so this is C_2, this is C_2, I can select also 5 right you can understand that.

So, I am assigning color 2(C_2) to 6, 7, and 5 and then there is the next iteration only one is left 4. So, color 3(C_3) will be assigned to 4. So, this is how every iteration you try to find out the maximum number of possible intervals that can be given the same color right this is how I can solve this problem and here it is the way I just discuss that 1, 2 and 3 is scheduling I mean given the color 1(C_1) and then of 6, 7 and 5 is given the color 2 (C_2) and 4 is color 3(C_3).

So, 3 colors are needed and operation so if you just look into this. So, now, the I just take the conflict graph so; that means, 1, 2 and 3 have the color yellow and say this 6, 7, and 5 have the color say orange and 4 has the color purple as you can see here that no two adjacent nodes have the same color that is the rule of graph coloring right.

So, I am going to give the color to the node such that no two adjacent nodes have the same color and that actually gets satisfied here, and hence I need 3 colors to solve this problem ok. And mapping of each of the nodes or the operation to the FU also is determined by the left edge algorithm ok.

(Refer to Slide Time: 19:57)



Left-edge algorithm

```
LEFT_EDGE(I) {
  Sort elements of I in a list L in ascending order of l_i;        — O(n log n)
  c = 0;
  while (some interval has not been colored) do {
      S = Ø;
      r = 0;
      while ( exists s ∈ L such that l_s > r) do {
            s = First element in the list L with l_s > r;
            S = S ∪ {s};
            r = r_s;
            Delete s from L;
      }
      c = c + 1;
      Label elements of S with color c;
  }
}
```

(c) Giovanni De Micheli

16

So, I will just formally explain the algorithm. So, it takes the intervals I which are given by the left edge and right edge. And then I am going to sort the intervals based on the left edge, not by the right edge it is by the left edge. So, you can see here it is already sorted by the lift edge ok.

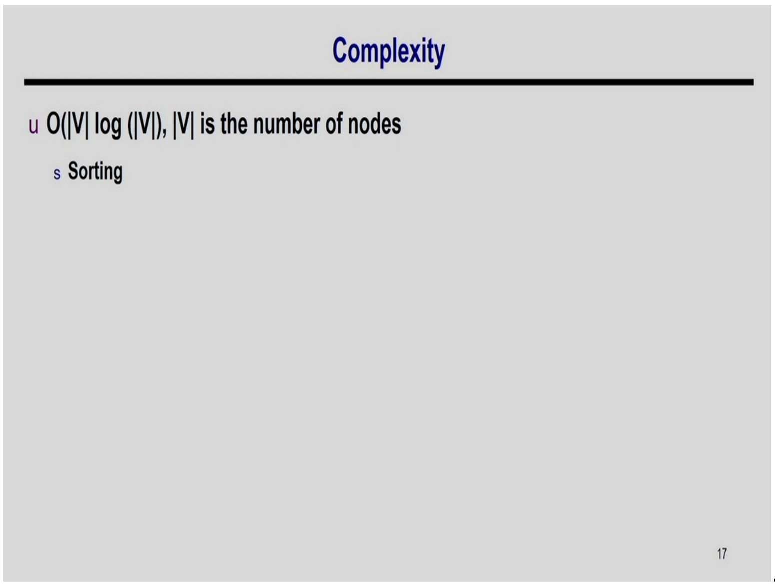
And then what I am going to do, I am going to take one color at a time initially say my color is say C_0. So, then this outer loop will execute until all the intervals get over, and then in every iteration of this insight, I am going to find out the nodes that can be given that particular color right.

So, now this particular while loop identifies all the intervals that can be given the same color and that is given by this logic right. So, this is greater than equal to that I am going to select the next one whenever my with the Is that left edge of that is greater than equal to the r, r is the previously selected one right edge ok. So, once I found something I will find that particular node, and then I will put it into the lists.

So, all the nodes initially put into S can be given the same color initially $S = \Phi$ right. So, there is no element there. So, gradually and initially my right edge is 0 because I am starting from 1 ok. So, then I select the element I put into the list and then make my $r = r_s$, the right edge of the currently selected one that the way I discuss in the previous one right, and then I delete s from L because it is already selected.

So, this way this one iteration of this loop will identify all the intervals that can be given the color c current color c right, and then I increment the color and the basics I giving the color 1 2 3 and all. So, I am giving the color and then I go for the next iteration and that will go for color 2 and so on right. This is how the whole algorithm works I hope you understand.

(Refer Slide Time: 21:49)



Complexity

- O(|V| log (|V|), |V| is the number of nodes
  - Sorting

17

.

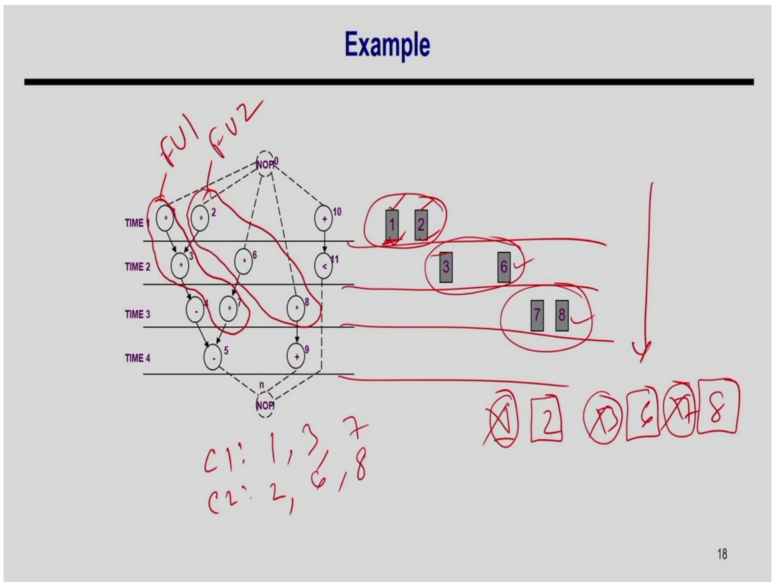
So, what is the complexity let us understand that? So, sorting numbers so, let us say the number of nodes is n. So, which is O(n*logn) right that is the most efficient comparison of the sorting algorithm we can apply. And then there are two loops right. So, looks like kind of n^2, but let us understand the amortized cost of this.

So, what is happening here? So, it has a list of the node. So, this internal node will scan the whole list at a time right, and then the list has n number of nodes. So, it will identify all the intervals that are got selected right. So, I am going to select here and then next iteration those get deleted right. So, in the next iteration, I am going to select only the rest of the intervals available right.

So, this is how the whole thing works and you can understand that if we do an amortized analysis of this it is just scanning the list for few times right. So, it is the number of colors that you identify that is the number of times you are going to scan the whole list, and with every iteration, the list gets reduced right. So, the overall complexity of this will be O(n). So, the overall complexity is O(n*logn) right.

So, where I am assuming the n is the number of nodes in the conflict graph or the number of intervals available to me ok. So, that is why the complexity is O(n*logn) and it is specifically because of the sorting of the intervals ok.

(Refer Slide Time: 23:16)


Example

So, now, let us take the example that I talked about previously that this is the schedule, and this for this schedule I have this is the interval I am going to consider only the multiplier, and then you can consider the ALU operation in the next iteration right. So, here the intervals are actually given in this order. So, you can assume that this is your left edge and this is the right edge ok, this is the left edge and this is the right edge. So, what will be the sorting of this it is 1, 2, 3, 6, 7, and 8 right.

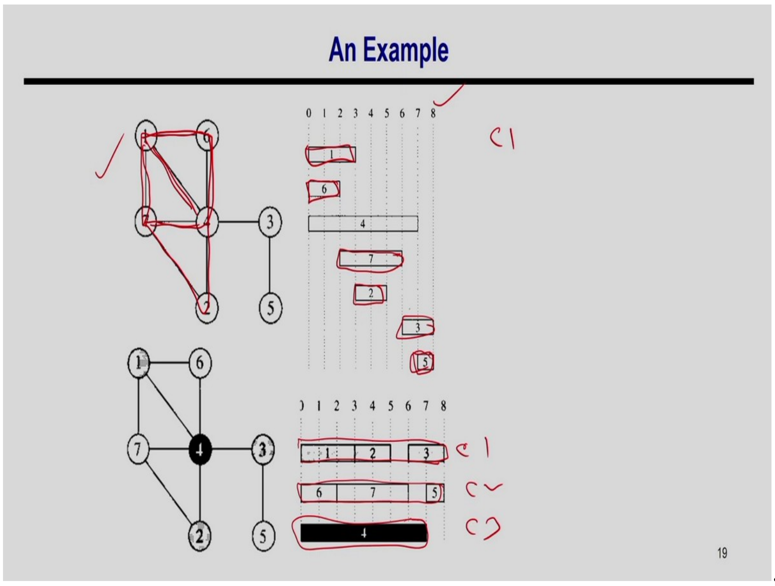
So, now, you take the color 1. So, I can select 1, if I select 1 I cannot select 2 because they are overlapping. And then, but I can select 3 right because 3 the left edge >= right edge of the previously selected one so I can select 3. So, I am selected 1, I selected 3, once I select 3 I cannot select 6, but I can select 7 right. So, I am going to select 7 because it is non-overlapping with 3 so; that means, in the first iteration I select 1 3, and 7 and which is given the color 1 right.

And then I will make it color 2 then I will again scan the list which I only have three elements now. So, I will select 2 right. So, I am going to select 2 and when once I select 2, I can select 3 also I can select 6 and once I select 6 I can also select 8 because they are overlapping right. So, this is the second color. So, in this way, I colored these nodes and then I found that I need only two colors to execute.

And it is quite natural that because you have a maximum of two operations executing here, two operations executing in this clock, two operations are executing what so; that means, maximum clock color requirement is two that is quite obvious from this schedule, but what is the mapping that also it gives that color 1(C_1) that 1, 3 and 7 you execute in time stamp I mean using the FU 1 and 2, 6 and 8 you execute using the FU 2.

So; that means, 1, 3, and 7 these three operations are going to map to FU 1. 2, 6, and 8 these three operations are going to map to FU 2 right ok. So, this is how it works and this is how I can identify the FU requirement and their binding, allocation of the FUs, and the binding of the FUs ok both can be determined. One more example is given here I quickly cover.

(Refer Slide Time: 25:55)



An Example

So, suppose this is a conflict graph I found somewhere, you can see here there is a cycle of 4 nodes and there is a chord and there is a cycle of 5 nodes and there is a chord right. So that means, this is a chordal graph because this is a requirement that your conflict graph must be a chordal graph and says suppose this is one of the possible mappings of these nodes to the intervals ok.

And what is the first step you sort them and it is already sorted you can see here based on the lift edge and then what I am going to do I am going to add color? So, for color 1 I select 1 once I select 1 I cannot select 6 and 4, I can select 2 and then select 3 and I think this is the same algorithm that I covered in the previous example.

So, you can understand that 1, 3, 1, 2, and 3 sorry, 3 can be a map to color 1, and then 6, 7, and 5 which is non-overlapping can be a map to color 2 and 4 can be a map to color 3 right and this is what is the color given right. So, this is how I can solve the problem and with this, I conclude this discussion on the left edge algorithm.

Thank you.