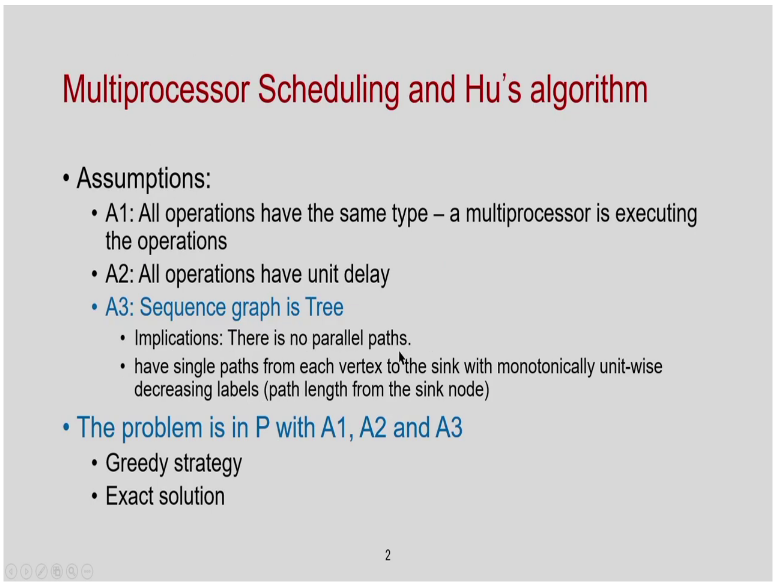
C-Based VLSI Design

Dr. Chandan Karfa
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Module - 03
C-Based VLSI Design: Scheduling
Lecture - 10
Heuristic Scheduling: List Scheduling for MLRC

Welcome students. In today's class we are going to learn about some Heuristic Based Scheduling, specifically the List Scheduling for MLRC problem.

(Refer Slide Time: 00:53)



Multiprocessor Scheduling and Hu's algorithm

- Assumptions:
  - A1: All operations have the same type – a multiprocessor is executing the operations
  - A2: All operations have unit delay
  - A3: Sequence graph is Tree
    - Implications: There is no parallel paths.
    - have single paths from each vertex to the sink with monotonically unit-wise decreasing labels (path length from the sink node)
- The problem is in P with A1, A2 and A3
  - Greedy strategy
  - Exact solution

2

So, if you remember that this scheduling where we are going to assign time step to the operations is in general NP complete problem. And the primary reason of this problem is that once you try to schedule say certain operation at certain time step say you are try to schedule the operation in time step 1 and say the number of resource available is k and there are more than k operations is available to be scheduled in the time step.

And since you have to select a subset of the operations from this available set, and selecting the subset greedily at that point sometime is not possible, because the choice that you have taking is may not be guaranteeing to have a optimal solution, that is why this problem is NP complete in general.

Then in previous class what we have seen that is we take certain assumptions and under thus those assumptions we have seen that particular scheduling problem become polynomial type solvable. And what are those assumptions? So, we assume that all operations are of same type. In general your the sequence graph has operations which is addition, multiplications and so on.

And now in this particular multiprocessor scheduling we have seen that we can have a single type of function unit which is a processor, multiprocessor which can execute any operations. So, you are assuming that all nodes of same type. Then you assume that all operations of unit delay, the delay are 1.

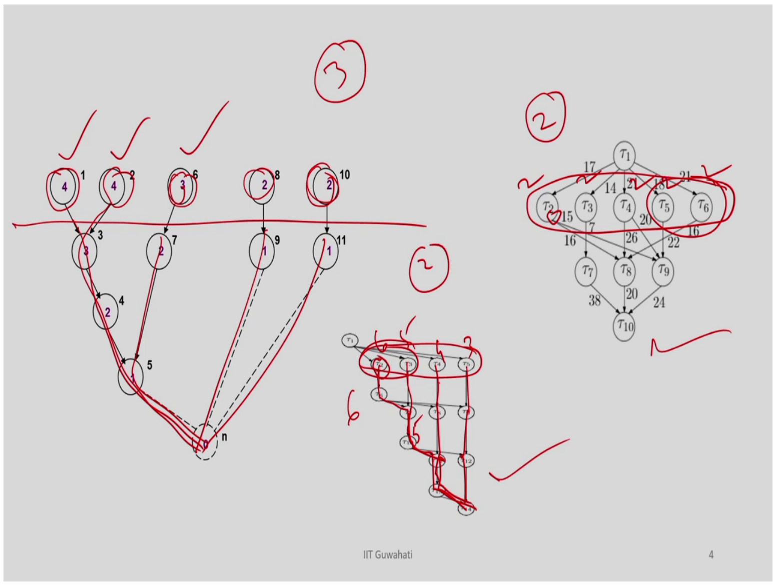
And specifically this sequence graph is a tree. The data dependency graph that we have given is tree; that means from any node you have exactly one path to the sink node.

So, at that point because there is no multiple paths or there is no multiple possibilities to reaching the sink node, so, for this particular cases where this assumption 1, 2 and 3 holds you can have an algorithm which can select that subset that I am just talking about that you have say n number of operations available in the time step I have to select k.

You can select a subset based on those nodes distance from the sink node and that guaranteed that particular solution will result in minimum number of latency to schedule those operations. And this is what we have learned in the previous class and we have seen that the Hu’s algorithm that actually exactly done that. And specifically that important thing is that the distance, we are actually leveling the nodes with the distance from the sink node.

And what is that? So, is basically talked about of distance from the sink node and that particular distance is important.

(Refer Slide Time: 03:34)



Because if you just think about this node. So, in f separate time step 1 there are 5 nodes which is available to be scheduled because they are that depends on the inputs and say suppose I have to select 3, I should select the nodes that has the maximum level. So, I am going to select this one this, one and this one.

Because this node delay is 4; that means, there are 4 nodes still to be scheduled following this node. Similarly, it is 4 there are 4 nodes to be scheduled for this node there are 3 nodes to be scheduled. So, for this 2 nodes to be scheduled, for this node 2 node to be scheduled. So, if we select the node that has higher level; that means, I am actually making sure that there are many other nodes that are yet to be scheduled they will become available earlier.

So, as a result this will ensure the schedule which is minimal. So, that is something we have understood in the previous class. So, now, for generic graph where the path is not unique specifically, the assumption 3 that this is tree does not hold and also your operation set can have a different delay and then you can have different type of operators.

So, this is the generic graph. In that case you may not, there is no you can have a polynomial time algorithm which can give you optimal solution. So, what is usually is taken care is a heuristic based approach. It is basically some kind of approximation you are doing so that you which is basically more logical in the sense. So, the for example, say what about just applying this Hu's algorithm for this kind of graph.

So, which is also making sense this may not be give you the optimal solution in some corner cases, but most of the time this is a logical assumption even for a sequence graph which is a graph not a tree. That if so, among the operation that is available to be scheduled I am going to take operation switches higher level.

For example, say in this example suppose these are the nodes are available to be schedule in a time step 1 and if you see here for this node the distance is 1, 2, 3, 4, 5, 6. So, it is 6 for this node 1, 2, 3, 4, 5 it is 5 for this node 1, 2, 3, 4, it is 4. So, 6 5 and this is 3, 1, 2, 3. So, basically; obviously, here you can see that you can actually choose if there are two if you available you should select these two.

So, basically for this you can actually apply the same logic and most likely this will work because in the same logic holds here that the distance this if you select this there are many operation yet to be scheduling in distance. So, it will actually help. But, in some cases like for example, here if you have say 5 operation now ready to be scheduled in say time step 1 and you have say 2 operator.

See here for all nodes the distance is 3. So, you can see here the distance is 1, 2, 1, 2, 1, 2. So, there are many paths. From this node there are many path, one path two paths. One path from this node there are one path from this node, one path from this node, there are two paths and so on. So, this is not definitely not a tree, but what I am try to say that here is because all node that level is 2, so, that particular logic.

So, here is a problem. There are 5 nodes which have the level all have leveling 2 and you try to select only two of them. So, which one to select? So, this something you have to select arbitrarily and specifically if you choose some arbitrary one that may not give you the optimal solution.

So, that is why this particular Hu's algorithm is not directly applicable for generic graph because basically this choice may not be unique and the choice that you are going to make may not guarantee to give you an optimal solution.

For example, here you just try to look into this problem closely. So, for example, say for this node it has only one output for if you try to see that from this node there are many paths, two paths. So, probably among these two I am going to give preference to this node because this if you select this there are two other path nodes are have the dependency is now resolved.

So, those nodes are now will be ready to be schedule in the next subsequent step. So, this kind of logic might come as a secondary decision making, but again that may not always hold for any time. So, this is what is the heuristic that has talked to you.

So, first heuristic that we talked is among the nodes. You select the node which has the highest number of prior level or the priority. So, which holds perfectly for here, but if you as I mentioned for this since the all the node has the distance from the sink node is 2, so obviously, that particular logic will take arbitrary one and that arbitrary it may not give you the best solution.

So, what we I just argued that maybe you can have a heuristic 2 is in this case of case there is a collision. I am going to select the node which has from that there are more has more paths than the other one. So, that is am my heuristic, but it may it may it may work for this graph, but it may not work for some other kind of graphs.

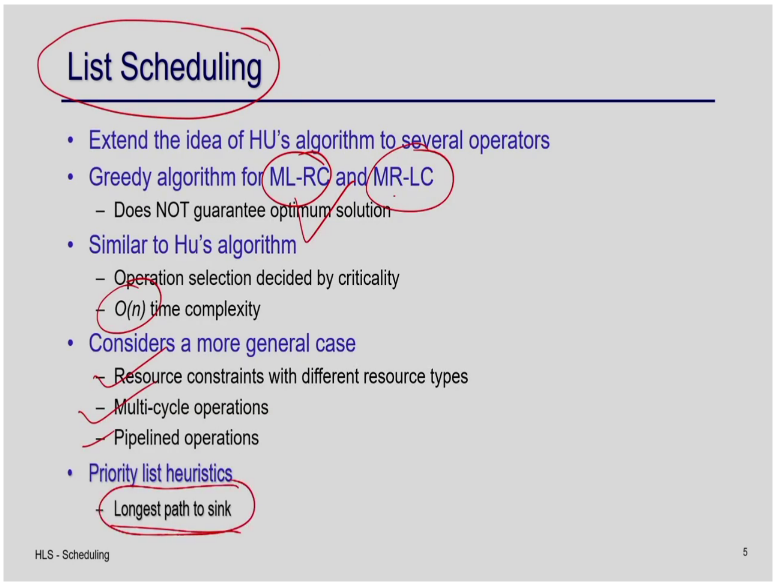
So, heuristic is something like that. So, specifically the heuristic approach whenever you have more choice you have to select a subset from a set of nodes you have to have some priority. You must have some priority. You have to decide on those nodes and based on the priority you select subset.

So, your algorithm will be faster because you are just going by some priority rules and you just select them. So, the algorithm will run in very fast manner and that priority must be sensible enough the way I explained here. So, and that particular priority may give you good solution or near optimal solution in most of the practical cases, maybe some corner cases it may not give you the optimal solutions.

And I am happy enough because I am not going to run for 2 to the power n number of complexity rather I can just run in order of n or say order n log n kind of complexities. So, this is what is the heuristic approach. So, in nutshell that you give a, you assign a priority to in nodes and whenever there is a collision you have to select a subset from this.

You actually apply this priority and based on the priority the nodes has the highest priority you pick them and this priority can differ. So, you can have your own priority and you can prove that priority is something is works for many scenarios. So, that is the overall idea of the heuristic based approach.

(Refer Slide Time: 10:18)



List Scheduling

- Extend the idea of HU's algorithm to several operators
- Greedy algorithm for ML-RC and MR-LC
  - Does NOT guarantee optimum solution
- Similar to Hu's algorithm
  - Operation selection decided by criticality
  - $O(n)$ time complexity
- Considers a more general case
  - Resource constraints with different resource types
  - Multi-cycle operations
  - Pipelined operations
- Priority list heuristics
  - Longest path to sink

HLS - Scheduling 5

And one of this most important heuristic based approaches is called list scheduling. And what is this list scheduling? It is basically the generalization of the idea of Hu's algorithm. So, it is basically take the same idea of heuristic of the Hu's algorithm which has applied for multiprocessor scheduling, but let us say I am going to apply the same logic to a general sequence graph where you do not have the assumption 1, 2 and 3.

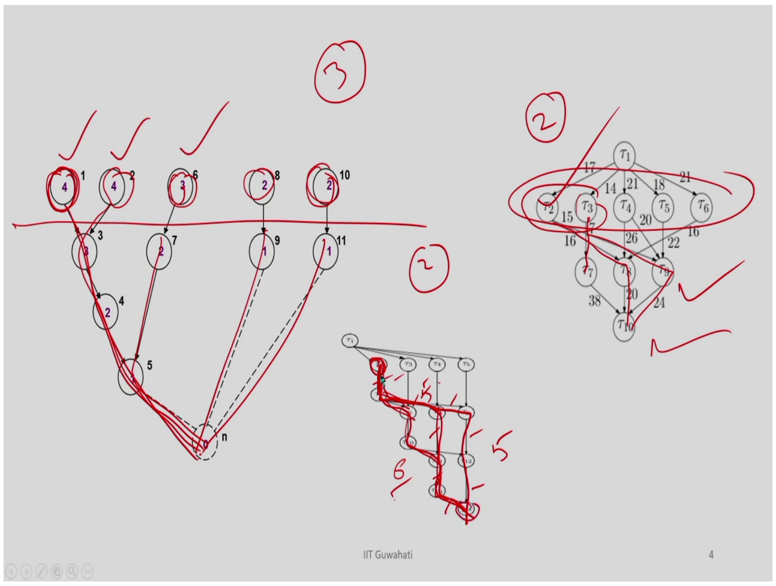
So, that is what is called list scheduling. So, it is a heuristic approach which is initiated the idea from the Hu's algorithm and it is basically generalizing the some of the aspect of the Hu's algorithm so that it is actually getting applied for generic graph. So, obviously, as I mentioned that its complexity is very fast, is just like Hu's algorithm and since its applicable to generic graph its can work for multi cycle pipeline or different type of resource.

So, all kind of flexibility it has and we can have both the versions. We can have the list scheduling for minimization of latency under resource constraint or minimizations of resource under latency constant. So, both is we can have a list scheduling and we are going to learn this too. In today's class we are going to learn this that I have resource constant is given. I have given the resource bound and I am going to identify the minimum latency using the list scheduling.

And as I mentioned that for any such heuristic algorithm you need priority logic and in this particular list scheduling they apply the same logic whatever is given in Hu's algorithm which is the longest path to the sink.

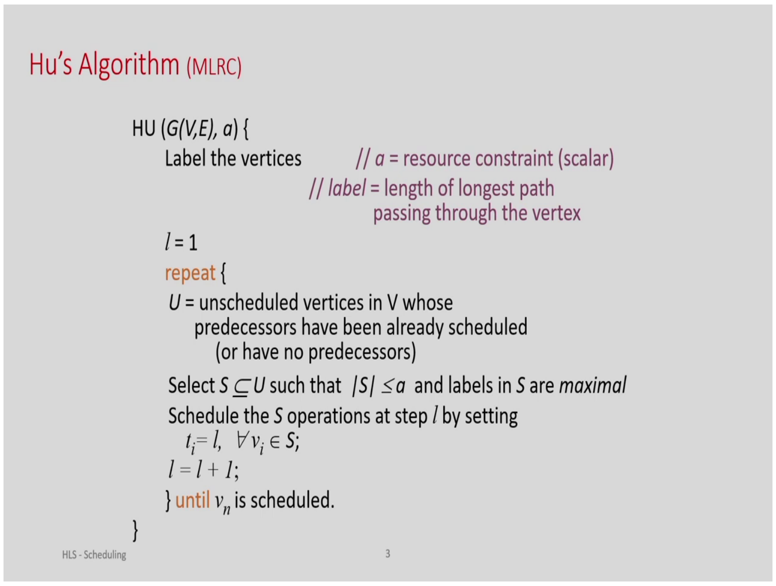
So, here because in that Hu's algorithm because it was a tree you have always a one path, you can always say that the distance from this node from the sink node because it has a one path, but in for in generic case for example, for this graph this is a generic graph.

(Refer Slide Time: 12:17)



So, for this graph we can see here. So, from this path there are how many paths? This is one path. So, this is one path, this is one path. So, we can see here the distance of this node is 1, 2, 3, 4, 5, 6. This node is 1 sorry 1, 2, 3, 4, 5, sorry yeah 1, 2, 3, 4, 5. So, this is 5 this is also 1, 2, 3, 4, 5; so, 5. So, there are 3 paths and one of the path distances is 6. So, I will assume that for this node the distance from this sink node is 6.

(Refer Slide Time: 13:02)



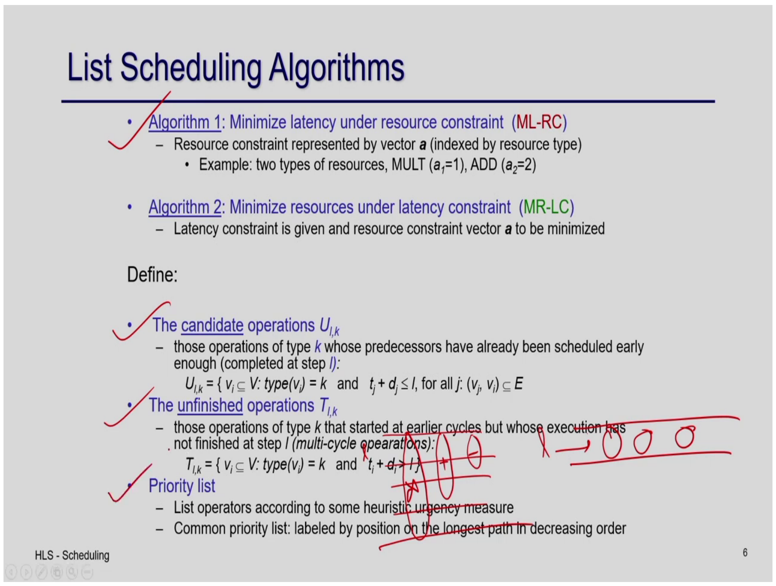
Hu's Algorithm (MLRC)

```
HU (G(V,E), a) {
    Label the vertices      // a = resource constraint (scalar)
                            // label = length of longest path
                                   passing through the vertex
    l = 1
    repeat {
      U = unscheduled vertices in V whose
          predecessors have been already scheduled
             (or have no predecessors)
      Select $S \subseteq U$ such that $|S| \leq a$ and labels in S are maximal
      Schedule the S operations at step l by setting
         $t_i = l, \ \forall v_i \in S$;
      $l = l + 1$;
    } until $v_n$ is scheduled.
}
```

HLS - Scheduling 3

So, instead of so, in Hu's algorithm we have seen that we have taken a level which is basically the distance from the sink node because it has a unique path. Now, you have multiple paths. So, I am going to take the path which has the maximum distance. So, this is what this list scheduling considers as the priority. I hope you understand that.

(Refer Slide Time: 13:18)



List Scheduling Algorithms

- Algorithm 1: Minimize latency under resource constraint (ML-RC)
  - Resource constraint represented by vector **a** (indexed by resource type)
    - Example: two types of resources, MULT (a_1=1), ADD (a_2=2)
- Algorithm 2: Minimize resources under latency constraint (MR-LC)
  - Latency constraint is given and resource constraint vector **a** to be minimized

Define:

- The candidate operations $U_{l,k}$
  - those operations of type *k* whose predecessors have already been scheduled early enough (completed at step *l*):

    $U_{l,k} = \{ v_i \subseteq V: type(v_i) = k \text{ and } t_j + d_j \leq l, \text{ for all } j: (v_j, v_i) \subseteq E$
- The unfinished operations $T_{l,k}$
  - those operations of type *k* that started at earlier cycles but whose execution has not finished at step *l* (*multi-cycle opearations*):

    $T_{l,k} = \{ v_i \subseteq V: type(v_i) = k \text{ and } t_i + d_i > l \}$
- Priority list
  - List operators according to some heuristic urgency measure
  - Common priority list: labeled by position on the longest path in decreasing order

HLS - Scheduling 6

And now so, as I mentioned there are two type of algorithm either you do this for minimization or latency under resource constraint or you do this minimization of resource under latency constraints. So, there are two type of resource. So, today going to learn about this algorithm 1.

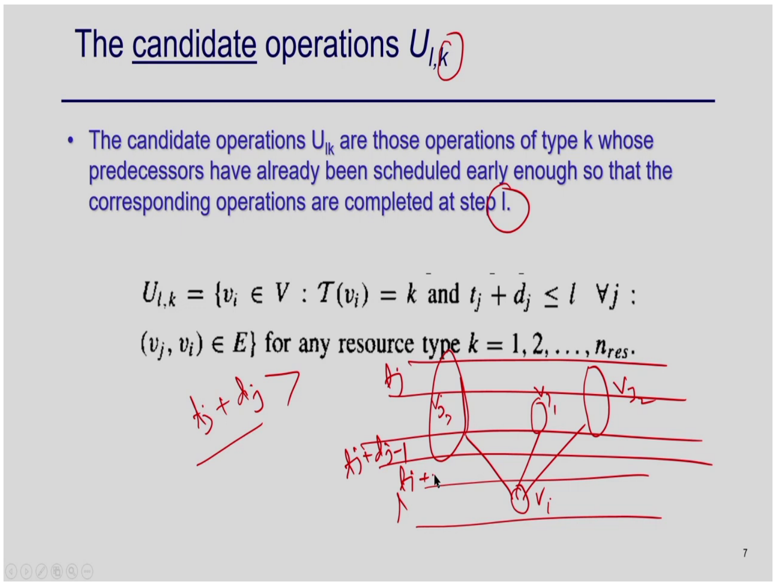
And so, if you remember that in Hu's algorithm, so, we have a level, and basically since all the nodes are of a unit delay, so, if it is schedule in set time step l it will complete the execution in time step l only, but in general the nodes can have multiple delays. So, if the nodes schedule in time step l it may continue for 2 cycles, 3 cycles or a many cycles based on the type of a node.

So, this may be multiplied this may be adder this may be divider and so on. So, that the delay may be more. So, in Hu's algorithm we just if I schedule them here we are done, but in list scheduling we have to check if the operation is start its execution in time step l till what time it is running.

So, only if when it is complete its execution its subsequent node will be ready to be scheduled. So, that is something we have to understand. So, there are two type of things we will understand; the candidate operator and unfinished operator. I am going to explain them in detail here.

And also we have a priority list that I have already explained that it is from the for any node it is the longest path distance of the longest path from the sink node.

(Refer Slide Time: 14:56)



The candidate operations $U_{l,k}$

- The candidate operations U_{lk} are those operations of type k whose predecessors have already been scheduled early enough so that the corresponding operations are completed at step l.

$$U_{l,k} = \{v_i \in V : \mathcal{T}(v_i) = k \text{ and } t_j + d_j \leq l \quad \forall j : (v_j, v_i) \in E\} \text{ for any resource type } k = 1, 2, \ldots, n_{res}.$$

$t_j + d_j$
t_j
v_{j_1}
v_{j_2}
t_j+d_j-1
t_j+1
v_i
7

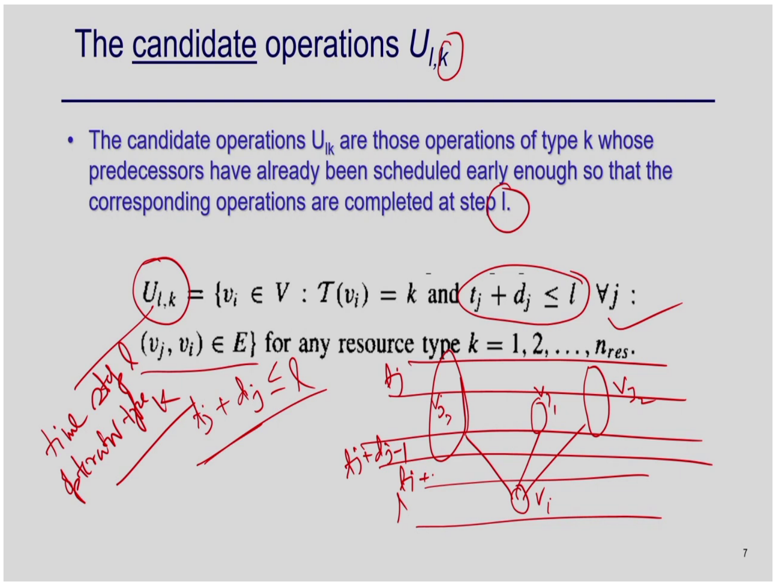
So, let us try to understand the candidate operator. So, as I mentioned here you have not a single type of operator, you have many type of operators. So, let us say I am going to take a operator of type k. So, this k can be multiplier or adder. So, now, for this type k, what I am going to identify the? This is the candidate operator; that means, this operation can be scheduled in time step l.

So, I am actually talking about a time step l. So, when this particular an operation is ready to be scheduled in time step l? When so, this node all the predecessors say there are this may be 3 predecessors. So, it may say one predecessor here one predecessor here and one predecessor is here.

So, when this operations will be scheduled? So, when all the predecessors are completed their executions. So that means, if there is an edge, suppose this node is v j sorry v i and its predecessors are v j. So, this is v j say this is also v j 1, this is v j 2.

So, this is basically for all type of all predecessors. So, when this particular node will be available? When all of them are done. So, now, if we assume that this t j is the start time of this node. So, t j and the delay is d j. So, this must be greater than; that means this is already done. So, t j plus d j, so, it is started from this time step t j and it will complete the execution here in t j plus d j minus 1. So, from t j plus d j I am ready.

(Refer Slide Time: 16:43)



The candidate operations $U_{l,k}$

- The candidate operations U_{lk} are those operations of type k whose predecessors have already been scheduled early enough so that the corresponding operations are completed at step l.

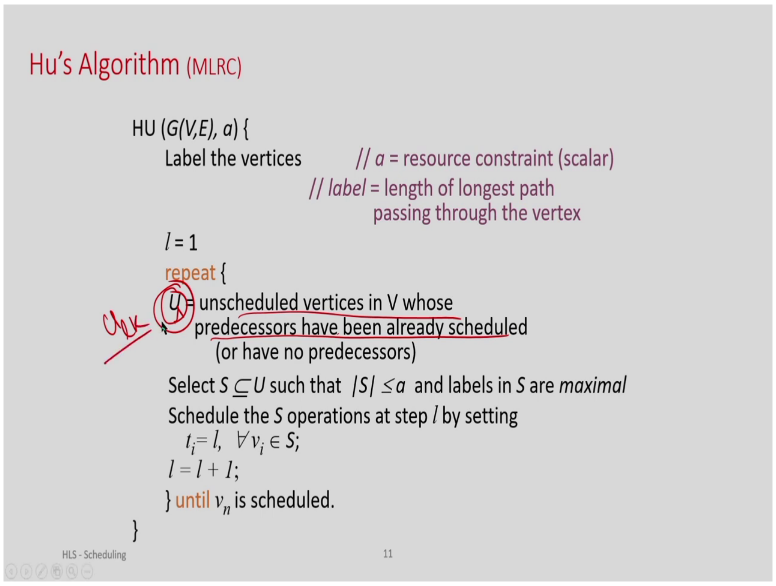
$$U_{l,k} = \{v_i \in V : \mathcal{T}(v_i) = k \text{ and } t_j + d_j \leq l \ \forall j : (v_j, v_i) \in E\} \text{ for any resource type } k = 1, 2, \ldots, n_{res}.$$

time step l
operator type k
$t_j + d_j \leq l$
t_j
v_3
v_1
v_2
$t_j + d_j - 1$
t_i
v_i
7

So, this t j plus must be less than equal to l. This is what the rule and this is true for all j, which is the predecessors of the current node. In high level I have to see what are the nodes are the predecessors of this node or this node v i depends on what are the nodes previously and I have to make sure that all the previous nodes already completed that execution. So that means, now I can schedule v i.

So, I am going to take those such node and I am going to put in U l, k. It is in for time step l, time step l and operator type k. So, I have to do it for all such time step, all such time step and for all such operator. So, this I am going to do. So that means, it is basically similar to that U in the Hu’s algorithm.

(Refer Slide Time: 17:46)

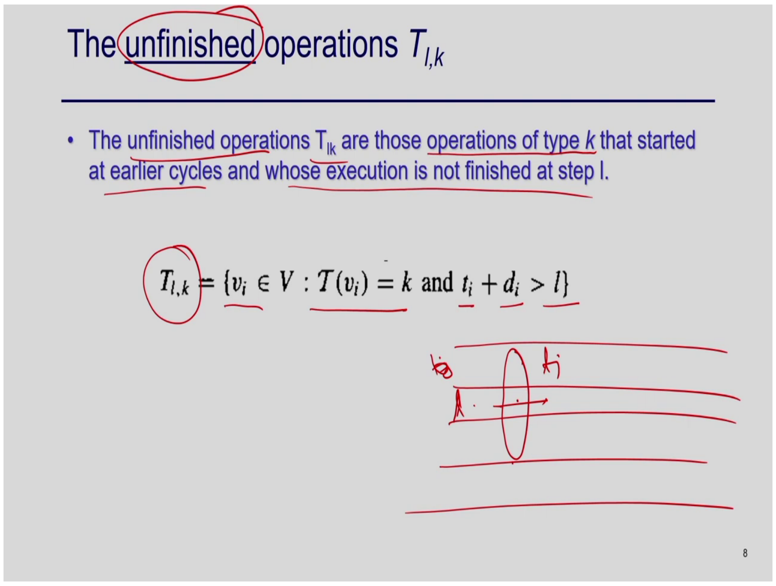


Hu's Algorithm (MLRC)

```
HU (G(V,E), a) {
    Label the vertices          // a = resource constraint (scalar)
                                // label = length of longest path
                                     passing through the vertex
    l = 1
    repeat {
        U = unscheduled vertices in V whose
            predecessors have been already scheduled
            (or have no predecessors)
        Select S ⊆ U such that |S| ≤ a and labels in S are maximal
        Schedule the S operations at step l by setting
            t_i = l, ∀ v_i ∈ S;
        l = l + 1;
    } until v_n is scheduled.
}
```

So, in the U we have that unscheduled operations whose vertices have already, predecessor vertices are already scheduled. So that means, since earlier cases it is the predecessor has a single cycle. If it is scheduled means it is complete in that time step itself. So, this is basically you do not have to do it for the type of operator or you can just think about this is l for each time step.

(Refer Slide Time: 18:10)



The unfinished operations $T_{l,k}$

- The unfinished operations T_{lk} are those operations of type k that started at earlier cycles and whose execution is not finished at step l.

$$T_{l,k} = \{v_i \in V : T(v_i) = k \text{ and } t_i + d_i > l\}$$

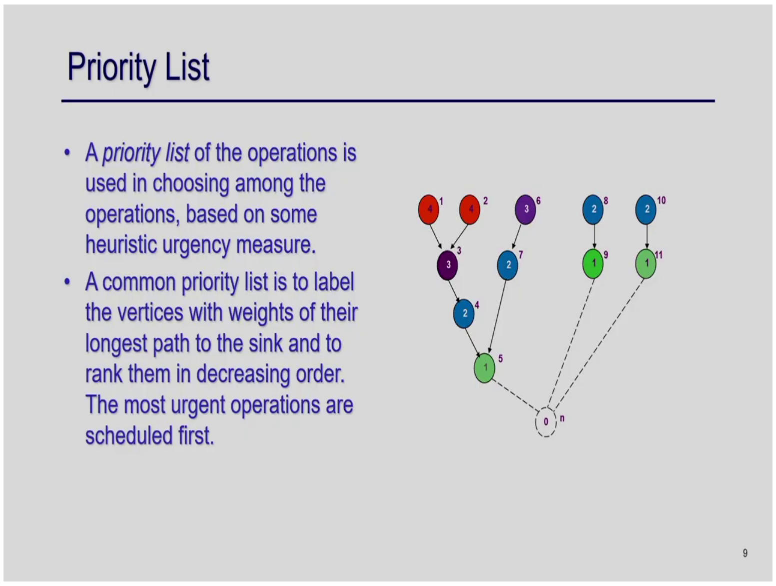
So, this is what is this candidate operator and the next set of operator is the unfinished operator. So, suppose you are as I mentioned earlier that if you have because this operator can be multi cycle suppose you have started this operation from this time step t j and its running. So, you are now in time step l. So, it must be, if it is not completed then say suppose this is your l is this; that means, this operation is running in this time step. So, I need a resource for that.

So, this is called unfinished operation t l, k again for that operator type k and in time step l those operations of type k that have started earlier cycles and whose execution is still not finished at time step l. So, if this operation is started here and it is still going on, so, in l this is running. So, this operation will come into t lk. So, that means, here all the nodes which type is k and it is t i plus d i it is running time is sti   ll greater than l, so; that means, it is still running.

It started here t i and still going on it will complete here, but I am here. So that means, these are the sets which has to be we need to allot resource for these nodes definitely in time step l because these are all running, I cannot stop them or halt them in between. So, this particular things was not there in the Hu's algorithm because there it is basically 1, all operations were single cycle.

So, these two important information the candidate operation is the that means, the operation which is now ready to be scheduled in time step l. And the unfinished operation; that means, these operations started its execution earlier and it is actually running in time step l.

(Refer Slide Time: 19:54)



Priority List

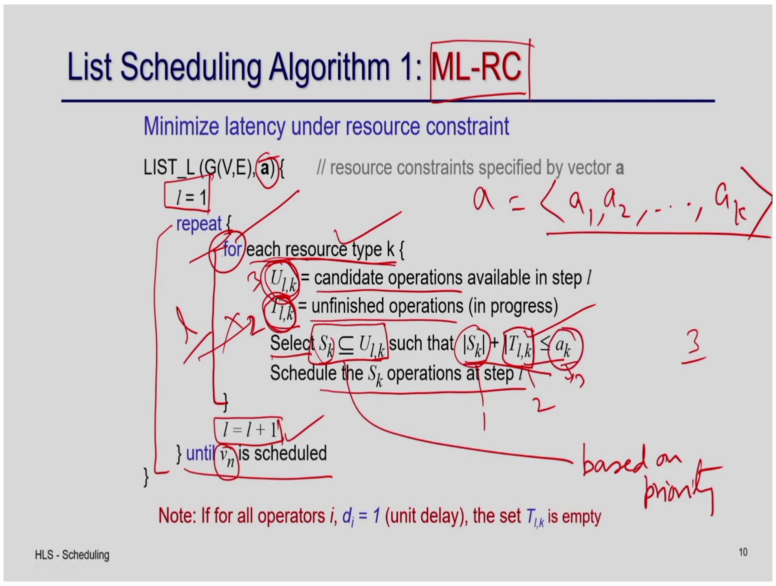
- A *priority list* of the operations is used in choosing among the operations, based on some heuristic urgency measure.
- A common priority list is to label the vertices with weights of their longest path to the sink and to rank them in decreasing order. The most urgent operations are scheduled first.

9

And the priority list as I mentioned that you can actually for each node you identify the distance of the node from the sink node by identify the longest path. You might have many paths, but you identify the path which has the longest, which is the longest among all the paths and then you actually assign the value to those nodes.

And then you can actually sort the nodes based on their level. So, it is basically again I am leveling with the distance of that node from the sink node and I am going to go if there are multiple paths it is the longest path I am going to consider. So, these are the three information's let's say I have available now.

(Refer Slide Time: 20:34)



List Scheduling Algorithm 1: ML-RC

Minimize latency under resource constraint

```
LIST_L (G(V,E), a) {      // resource constraints specified by vector a
  l = 1
  repeat {
    for each resource type k {
      $U_{l,k}$ = candidate operations available in step l
      $T_{l,k}$ = unfinished operations (in progress)
      Select $S_k \subseteq U_{l,k}$ such that $|S_k| + |T_{l,k}| \le a_k$
      Schedule the $S_k$ operations at step l
    }
    l = l + 1
  } until $v_n$ is scheduled
}
```

$a = \langle a_1, a_2, \ldots, a_k \rangle$

3

1

2

based on priority

Note: If for all operators i, $d_i = 1$ (unit delay), the set $T_{l,k}$ is empty

HLS - Scheduling 10

Then this list scheduling is very simple. It is very similar to your Hu's algorithm and let me try to explain and it is for MLRC; that means, I have resource constant is bound is given. So, I have given a resource bound which is a vector. So, a is basically a1, a2; a1, a2 to a k there are k type of resource. So, this is the resource bound for each type of resource. So, what I am going to do?

I am going to do the things for all nodes in my sequence graph. So, I am going to run an outer loop until my all nodes are scheduled. So, v n is the sink node. So, until my sink node is scheduled I am going to do this whole process time and again. And then what I am going to do? For each type of resource I am going to do the things.

So, it is for each type of resource in each time step. So, you can see here I start my time step with l equal to 1 and I am going to increase this time step to this. So that means, this particular loop actually doing the scheduling in time step l. Once the lth time step is over I am going to move to the l plus 1. So, it is basically I will start my scheduling of time step 1, and then I am going to do it for time step 2, then time step 3 and so on. I am going to do it and until my sink node is scheduled.

So, what I am going to do it here? So, I am going to identify for each type of resource k, I will identify the candidate operations which is ready to be scheduled in this time step and that

I have already explained. That means, is for this operations predecessor, all predecessor on which this operation depend have finished their execution before l. It is not running in time step l.

So, that is my U l, k and this T l, k is something is all the type, all the operations that have started early, but still continuing still running in time step l and those operations of type k.

So, this is already I have identified the unfinished operations. And then I am going to I am to make sure that I need resource for this, I cannot avoid this. So, I must have to allot resource for T l, k and if still some resource available then you select a subset from this. You select a subset from this and assign to this time step.

Because suppose you have say a k. So, if you say as this is a multiplier and there is a 3 multiplier and this T l, k is 2 and this is a 3. So that means, there are 3 multipliers is ready to be scheduled in this time step and 2 is actually have started early and it is still continuing.

So that means, now I must and there are say 3 multipliers available. I must allot 2 multipliers to this and since I have 1 multiplier idle for this state I can select one of this available operations and I can schedule them in this time step.

So, now this selection is the based on the priority. So, now, in this particular class I am just assuming that this is the priority based on the longest path distance from the sink node, but you can have different priority. You can have say some very complex priority. So, that is why this algorithm is very generic. It just say [FL] you select a subset from this available operations such that you do not overshoot the resource requirement.

So, that is what this sentence means, but selection is based on the priority and that is where you can apply different heuristic, different priority and you can select those operations which has the highest priority. And once you select this subset of nodes which is can be scheduled in time step l, you just schedule those operation into time step l. So that means, I am going to do it for each type of resource type resource a1 then a2, a3 to ak.

Once I have done for all type of resource in time step l that means, my l scheduling for l is done I will move to the l plus 1 time step. So, that is what I am doing here. I will just increase the time step and I will go to the next time step and again I am going to calculate the U l, k and T l, k.

You remember that based on your choice here this U l, k and T l, k you have to calculate, because you based on the operation that you are scheduling in previous time steps your U l, k and T l, k it is not fixed for its operation. Because l is it is increase here. So, this U l k; that means, if it is for the time step it is U 1 k in time step 2, it is 2 k.

So, you have to recalculate this based on the operation that was running now whether they have finished the things or not, if finished those operation will move from T l, k to U l, k, you understand. So, this is what is this whole process and this can be done very efficiently and it is very fast algorithm and the important point is this actually works gives good results for most of the kind of sequence graph.

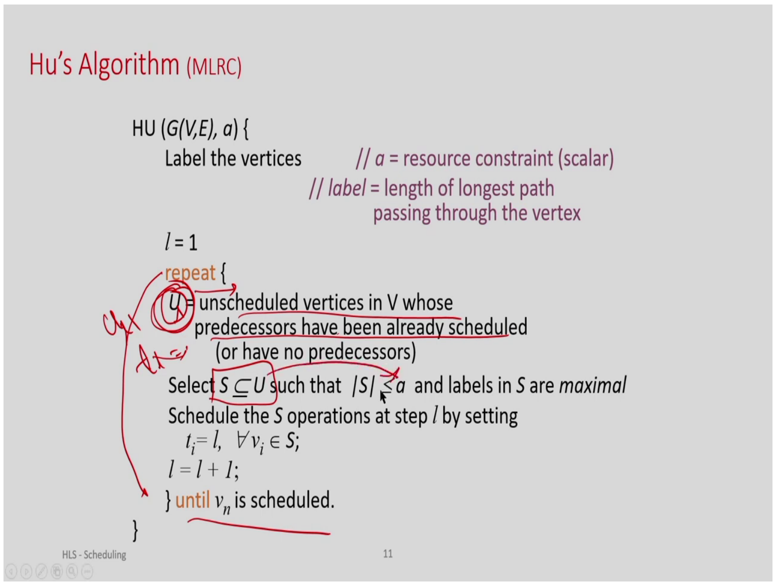
And importantly this particular algorithm is widely applied for any commercial high level synthesis tool. So, that is why it is a simple algorithm, but this is what is getting applied in practical high level synthesis tool because this is fast. And most of the time if your priority is basically the longest path most of the time it gives you good solution.

And now actually you can change this priority, suppose you want to do some reliability things bringing into picture you want to bring say some power consumptions bring into picture, you can actually change your heuristic. So, or say you want to do that so you have some aware of the interconnections cost and so on. So, you can actually think on that, you can actually add those kind of logic into your this priority list.

So, the beauty of this algorithm it is a very generic algorithm and based on your applications, based on your optimization objective you can set your priority and based on that your algorithm will work. The basic logic will remain the same, but you can apply different heuristic just to get your different objective fulfilled.

So, what I am going to do it now? I will just try to compare with the Hu’s algorithm. So, you see here are so, this is your U l, k basically. You are selecting u l, k because there is no k here it is U l because it is the one type of operator.

(Refer Slide Time: 27:04)



Hu's Algorithm (MLRC)

```
HU (G(V,E), a) {
    Label the vertices        // a = resource constraint (scalar)
                              // label = length of longest path
                                 passing through the vertex
    l = 1
    repeat {
        U = unscheduled vertices in V whose
            predecessors have been already scheduled
            (or have no predecessors)
        Select S ⊆ U such that |S| ≤ a and labels in S are maximal
        Schedule the S operations at step l by setting
            $t_i = l,\ \forall v_i \in S$;
        l = l + 1;
    } until $v_n$ is scheduled.
}
```

HLS - Scheduling 11

So, u you can assume that u l 1 because it is k is always 1. So, that is why this inner loop is not there. You can see here. Here I have an inner loop because it is for each type of resource. Since I have only one type of resource this for loop will run only one. So, that is not there.

So, you can see here. So, there is no inner loop, but the outer loop is there. So, it is basically doing for time step 1, time step 2 and time step 3 until I finish my last operation. So, you can understand that this is basically just the generalization of the Hu's algorithm for generic graph, where I just added for loop here which actually run for different type of resource which is not there in the Hu's algorithm because it is a only one type of resource and I am doing the exactly the same thing.

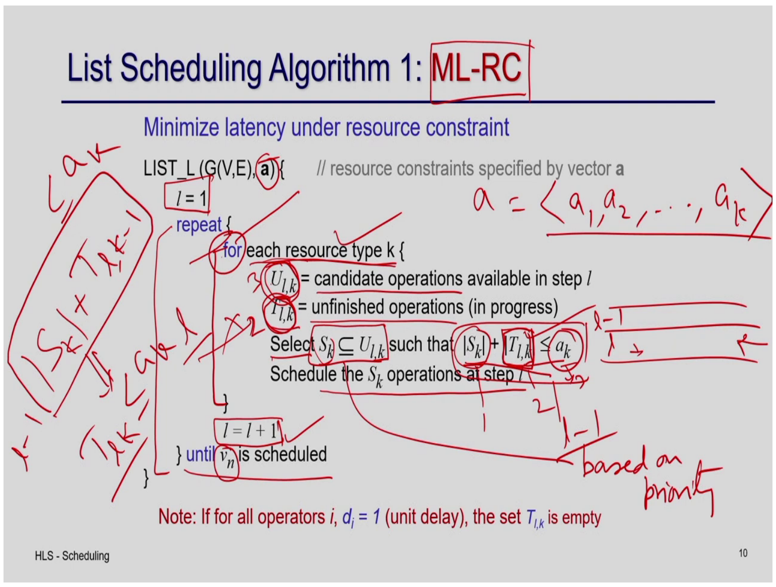
Because I am just there also I am just identifying the operation that is ready to be scheduled, here also I am identifying the operation that is ready to be scheduled. In addition because the operation is multi cycle here I am identifying the operation that running here which is not needed here, it will be always 0 here. So, that t l will be always 0 here because there is no operation if it is scheduling time step l minus 1 it will finish the execution in l minus 1. It will not go to l that in time step l.

So, that is why this T l, k is not there, but I am just adding it here because it is needed for generic cases and I am doing the same thing here. Then the next step is you select a subset.

Here also we are selecting a subset, but there since there is by T l, k 0 you just select a subset such that your resource bound is met. Here I am doing the similar thing I am selecting a subset, but I am making sure that I have enough resource to run the allocate resource to the running operations. And then only you select this subset.

So, exactly the same thing what is happening in Hu's algorithm, it is happening here also. Only thing I am just making it is a generalized for general purpose sequence graph. So, one question you might have things, how do you ensure that the running operation is always less than a k. This question might come to your mind because I am doing it from l minus 1 to then l minus 1.

(Refer Slide Time: 29:13)



List Scheduling Algorithm 1: ML-RC

Minimize latency under resource constraint

```
LIST_L (G(V,E), a) {        // resource constraints specified by vector a
  l = 1
  repeat {
    for each resource type k {
      $U_{l,k}$ = candidate operations available in step $l$
      $T_{l,k}$ = unfinished operations (in progress)
      Select $S_k \subseteq U_{l,k}$ such that $|S_k| + |T_{l,k}| \le a_k$
      Schedule the $S_k$ operations at step $l$
    }
    $l = l + 1$
  } until $v_n$ is scheduled
}
```

Note: If for all operators i, $d_i = 1$ (unit delay), the set $T_{l,k}$ is empty

HLS - Scheduling

10

$a = \langle a_1, a_2, \ldots, a_k \rangle$
$|S_k| + |T_{l-1,k}| \le a_k$
$l-1$
l
based on priority

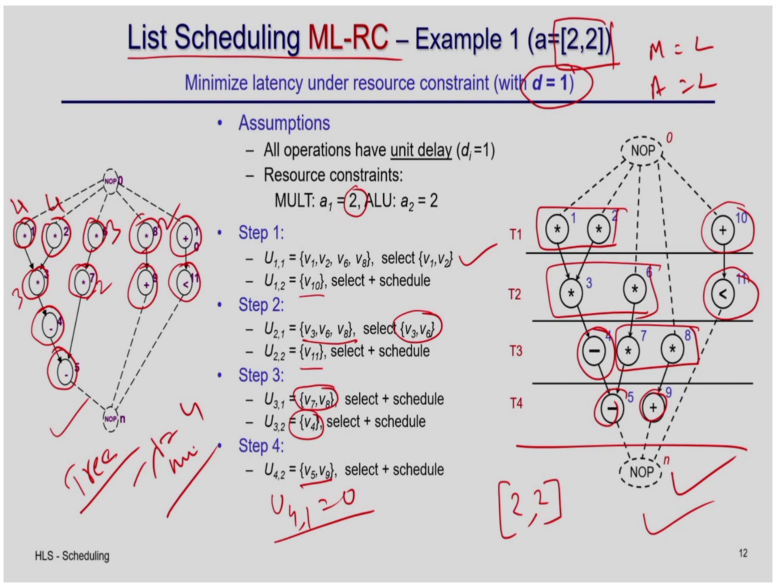
So, you are doing in l minus 1 then you are doing in l. So, how do you ensure that when you are doing in time step l that the number of running operations is must be less than a k. You might have a situation where you cannot select any new operation because the running operation is equal to the k. So, but how do you guarantee that it does not go beyond a k?

Because, you are already this formula satisfied for time step l minus 1 as well. So that means, for the time step l minus 1, the number of running operations plus the operation you are added that is less than a k and those operations at max can run in this time step.

So, whatever your S k plus T l, k minus 1, so, this was the operations you are running in l minus 1 and this will become your T l, k at most. Since this is less than equal to a k, so, this also must be less than a k because in a time step l minus 1 I have make sure that the number of running operations at that time step plus the new operation I have added that must be less than equal to a k and those are the operation at most kind still running in l. Few of them are complete, but at most this all may be running in time step l.

So, this way I am actually ensuring that in every time step whatever the running operations they are always less than equal to the number of resource supplements. So, you do not go into another that scenario anytime.

(Refer Slide Time: 30:46)



List Scheduling ML-RC – Example 1 (a=[2,2])

Minimize latency under resource constraint (with $d = 1$)

M = 2

A = 2

- Assumptions
  - All operations have unit delay ($d_i = 1$)
  - Resource constraints:
    MULT: $a_1 = 2$, ALU: $a_2 = 2$
- Step 1:
  - $U_{1,1} = \{v_1, v_2, v_6, v_8\}$, select $\{v_1, v_2\}$
  - $U_{1,2} = \{v_{10}\}$, select + schedule
- Step 2:
  - $U_{2,1} = \{v_3, v_6, v_8\}$, select $\{v_3, v_6\}$
  - $U_{2,2} = \{v_{11}\}$, select + schedule
- Step 3:
  - $U_{3,1} = \{v_7, v_8\}$, select + schedule
  - $U_{3,2} = \{v_4\}$, select + schedule
- Step 4:
  - $U_{4,2} = \{v_5, v_9\}$, select + schedule

$U_{4,1} = 0$

$[2, 2]$

HLS - Scheduling

12

So, I will just move on. I will what I am going to do it now? I am going to take couple of examples I am going to so that how this this MLRC list based scheduling works. So, first I will take the that same example, so, which is basically tree.

So, for this Hu’s algorithm also applicable only thing is that here I have a different type of operator. I have plus minus multiplication everything.

So, and let us assume that I have two type of resource. One is multiplier; I have multiplier 2 and ALU is 2. So, 2 ALUs, which can execute plus minus and greater than less than operations. So, let me try to understand, I mean explain this and I am assume that all these operations are one cycle. So, this is the simplest possible cases. Say I have all operations of single cycle.

So, what I am going to do? So, in the step 1, so, every time step you have to identify the U, the available operation ready to be scheduled and T l, k. For time step 1 there is no T because there is no running operation because you are in time step 1. So, you can identify and this you have to do for each type of resource. I will do for multiplier and I have to do for ALU. So, you can understand that in time step 1 there are these 4 operations are ready to be scheduled.

So, v1, v2, v6 and v8 and since I have two multiplier available, so, which I am going to select, which is the distance highest distance. So, in this case this is you remember that this is 4, this is also 4, this is 3, this is 3; 1, 2, 3, 1 sorry this is 2. So obviously, I am going to select this 1 and 2. So, that is why I have schedule 1 and 2 in time step 1. So, this I have done. Similarly, in the first time step only this operation is available for the ALU type operations.

Only one available and there are two resource, I am going to schedule it. So, that is why I have scheduled v 10. So, this is my time step 1. And since there is no this all operations are single cycle T l, k is always 0, because there is no operation is carry forward to the next time step. So, this is a simplest scenario.

So, in time step 2, what I am going to do? So; that means, I have already done the scheduling of this 3 in time step 1. So, in time step 2, what are the operations available? This node 3, 6 and 8; so, 3 6 and 8 and this distance is 3. So, this is the distance is 3 from the sink node, for this node say this as 3 and this is 2. So, I am going to select 3 and 6 because I can select at most two. So, I have done 3 and 6 here.

And for ALU, now I am going for the next type of operator. So, this two are done in time step 2 and for ALU operations still this operation, only this operation is available now because the other ALU type operator is still not available because their predecessors are not scheduled yet. So, there is only one. So, I am going to schedule this v 11 in time step 2.

So, in time step 3, So, you can understand easily now the multiplier available is say this and this and this is also 2 and there are two available and two multiplier available. So, I am going to select schedule both 7 and 8 here; 7 and 8.

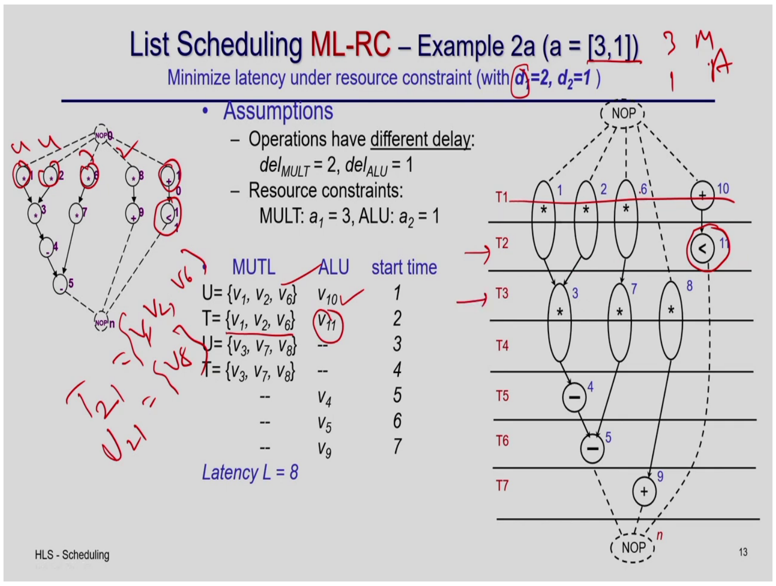
And for AL; this ALU type operation, since this was done earlier, so, this is available only one available. So, I am going to schedule that. So, this is my schedule in time step 3. So, in time step 4 I have there is no multiplier. So, there is no multiplication operation. So, that v U 1, 4 is 0. U 4 it is basically l k. So, l 1 is 0, there is no operations multiplication operations, but there are two ALU operations this and this and there are two resource available.

So, let us schedule them here. So, this is the schedule I am going to get for this particular constraint and you and it can schedule in time step 4 with 2 and 2 resource. So, remember whenever we discuss the ILP or so ILP solution. So, I got the same solution. So, I this is 4th time step and if you remember this is for this example the minimum lambda is lambda mean is equal to 4. So, you cannot schedule this behavior less than 4 time step when all operation is single cycle.

And we have already discussed that so far. So, if you have this 2, 2 with this my logic my MLRC can actually give you the optimal solution. So, this is all solution I am going to get it if I run the ILP also for this problem with resource constraint 2, 2 and we have already checked that you can actually cross check.

So, that is why I am saying that list scheduling may give you good solution, best solution sometime if your structure like this, but it may not give you the best solution the optimal solution all the time.

(Refer Slide Time: 35:54)



List Scheduling ML-RC – Example 2a (a = [3,1])

Minimize latency under resource constraint (with d_1=2, d_2=1)

- Assumptions
  - Operations have different delay: del_{MULT} = 2, del_{ALU} = 1
  - Resource constraints: MULT: a_1 = 3, ALU: a_2 = 1

MUTL	ALU	start time
U= {v_1, v_2, v_6}	v_{10}	1
T= {v_1, v_2, v_6}	v_{11}	2
U= {v_3, v_7, v_8}	--	3
T= {v_3, v_7, v_8}	--	4
--	v_4	5
--	v_5	6
--	v_9	7

Latency L = 8

HLS - Scheduling

13

So, let me take the same example, but now I assume that the delay of the multiplier is 2 and my resource bound is 3, 1 not 2, 2. So, I have 3 multipliers, 1 ALU. So, let me try to solve the problem again. So, you can understand that the first time step the multiplier available this 4; 4 multipliers are available and their delay is 4, 4, 3, 2 and I can select 3. So, I can obviously, going to select 1, 2 and 6. So, I have done.

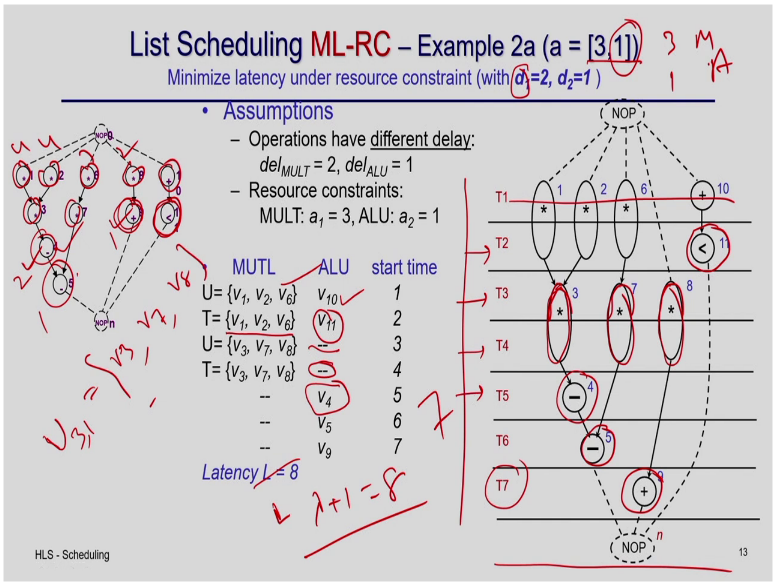
And this ALU operation 1 is available. So, I just schedule ALU 1, this is what is I have done. So, this is what I have done in time step 1. So, this is not scheduled yet, this all 3 done. In time step 2 my T is v1, v2 because these are two cycle operations. So, they are going to be running. So, there are my T 21 equal to this. So, v1, v2, and v 6 and although I have v 8 and then that U 21 is v8.

Although I have one operation available, but I cannot schedule because I have 3 multipliers, all 3 are occupied by my running operations. So, I am not going to take any new operation in this time step.

So, that is why I have not, see here I just did not schedule new multiply in time step 2. So, all these operations are running and they are occupying the multiplier and since there is no although there are some operation ready to be scheduled like v8, I cannot schedule them because the resource multiplier resource are already occupied.

But I can always schedule this v11 which is basically single cycle operation. So, this is only new operation schedule in this time series v11 and these are all running. In time step 3 this all are done, this 3 all 3 are done because they are already completed their execution. So, for now let us identify that.

(Refer Slide Time: 37:58)



List Scheduling ML-RC – Example 2a (a = [3,1])

Minimize latency under resource constraint (with $d_1=2, d_2=1$)

- Assumptions
  - Operations have <u>different delay</u>: $del_{MULT} = 2$, $del_{ALU} = 1$
  - Resource constraints: MULT: $a_1 = 3$, ALU: $a_2 = 1$

MUTL	ALU	start time
U= $\{v_1, v_2, v_6\}$	v_{10}	1
T= $\{v_1, v_2, v_6\}$	v_{11}	2
U= $\{v_3, v_7, v_8\}$	--	3
T= $\{v_3, v_7, v_8\}$	--	4
--	v_4	5
--	v_5	6
--	v_9	7

Latency L = 8

HLS - Scheduling

13

3 M
1 A
NOP
T1
T2
T3
T4
T5
T6
T7
1
2
6
10
11
3
7
8
4
5
9
n
NOP
7
λ+1 = 8

So, now whenever they are done my U is U 3 1; that means, in time step 3 for type multiplier 1 my v3 then v7 and v8 are available. So, I am going to schedule all of them in time step 3 because 3 multiplier available and 3 only resource. I do not have to check the priority I can schedule them and there is no such ALU available, because ALU is blank because there is no operation is ready to be scheduled.

So, I am not going to do anything here. So, in time step 4 it is very interesting that these 3 operations are running. So, I cannot consider any new multiplier and I still there is no other operation getting finished. So, none of the ALU operation is ready. So, I did not schedule any ALU operation in time step 4 as well. So, in time step 4 only this people are these 3 multipliers running, no new operation is scheduling in time step 4.

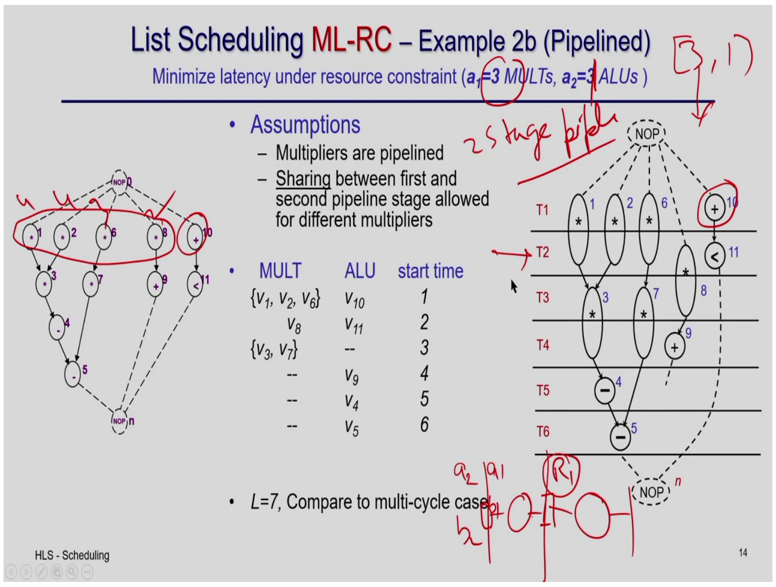
In time step 5, so, this multiplier is now done. So, now, these two ALU operations are there, but there is no multiplier. So, now, the multiplier will remain ideal for the rest of the time step. Although there are 3 multipliers they are all have been used only till time step 4 because there are 6 multiplications and they are all completed till time step 4.

So, from time step 5, 6 and 7 for the rest of the execution this multiplier will remain idle they have no use, but in time step 5 you can understand that this v4 this operation 4 and 9 is ready to be scheduled, but which one I am going to select because this distance is one this distance is 2? So, I should select this v4. So, I have selected v4, because I have only one ALU available.

So, I just schedule v4 here and in the next time step again v5 and v11 are available I can select any of them because both of the distance 1 from the sink node. So, let us say I have selected v5 and in the next time step I am going to select v9. So, this v9. So, what I got it here? So, with 3 multipliers which is 2 cycle operations multi cycle operation and 1 ALU I have I need 7 latencies basically 7.

And if you just assume this latency is basically lambda plus 1 is basically 8, but I need 7 cycles to complete this execution. So, this is for multi cycle operations.

(Refer Slide Time: 40:26)



List Scheduling ML-RC – Example 2b (Pipelined)

Minimize latency under resource constraint (**a_1=3** *MULTs*, **a_2=3** *ALUs*)

- Assumptions
  - Multipliers are pipelined
  - Sharing between first and second pipeline stage allowed for different multipliers

MULT	ALU	start time
$\{v_1, v_2, v_6\}$	v_{10}	1
v_8	v_{11}	2
$\{v_3, v_7\}$	--	3
--	v_9	4
--	v_4	5
--	v_5	6

- *L=7*, Compare to multi-cycle case

HLS - Scheduling

14

[3, 1]
2 stage pipeline
NOP
T1
T2
T3
T4
T5
T6
n
a_2
a_1
R_1

Now, let us take the same example I am going to quickly finish this, that now we assume that still I have 3 MULT and 1 ALU. So, basically that 3, 1 still available, but this is 2 stage pipeline. This is basically 2 stage pipeline multiplier. So, what is the difference between multi cycle and pipeline?

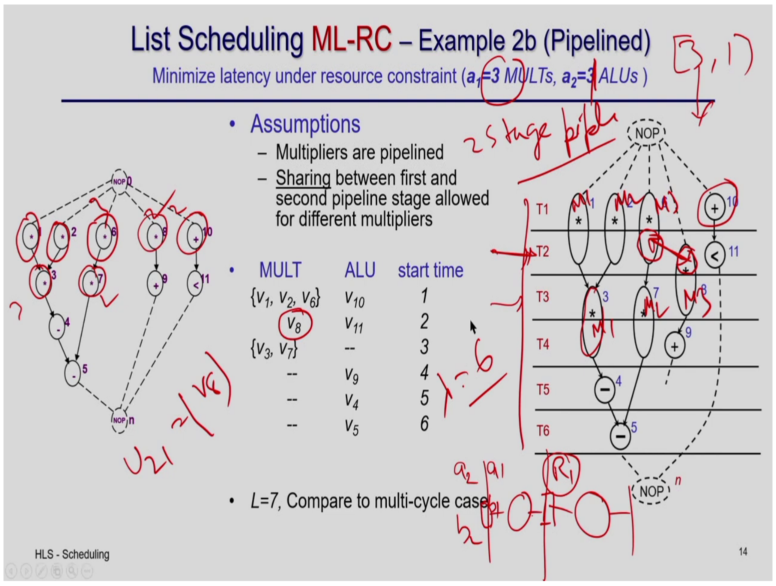
So, in the 2 stage the pipe line where I can actually run them, I can actually insert the next set of input to the same multiplier because it has 2 stages. So, basically just to finish. So, if this is the multiplier it has 2 stages. I have a pipeline here.

So, what I am going to do it is basically in the first clock I am going to give say a1 and b1. So, this is a1 and b1. So, it will do the multiplication and it will put the intermediate results into this say this which is basically say R 1 and in the next clock I am going to do give a2 and b2.

So, this part of the multiplier will do work on the a2, b2 and this part of the multiplier will work on a1, b1 because that intermediate results stored in R 1. So, it will work on the R1. So that means, when it has multiple stages and different stage can work on different stage, I mean different set of data. This is very standard pipeline logic. So, if I apply this concept here so, what I am going to do it?

You can understand that in 1st time step there are 4 multiplier available and you know their distance is 4, 4, 3, 2. So, I am going to select this 3 which is similar to the previous one that 1 to 6. The interesting point and for v10 because this is available I can schedule it here there is no problem here. So, you closely look into time step 2.

(Refer Slide Time: 42:19)



List Scheduling ML-RC – Example 2b (Pipelined)

Minimize latency under resource constraint (a_1=3 MULTs, a_2=3 ALUs)

- Assumptions
  - Multipliers are pipelined
  - Sharing between first and second pipeline stage allowed for different multipliers

-

MULT	ALU	start time
{v_1, v_2, v_6}	v_{10}	1
v_8	v_{11}	2
{v_3, v_7}	--	3
--	v_9	4
--	v_4	5
--	v_5	6

- L=7, Compare to multi-cycle case

HLS - Scheduling

14

So, in the time step 2 if you; so, basically I have already scheduled this nodes, so, how many multiplayer are available? So, U 21 is basically v 8. I have one of multiplayer can be scheduled. And since this multiplier I pipelined, so, what I can do? I can just start this execution and I can pipeline any of the multiplier where these two are running, let us say v6 and v8.

So that means, now these two multipliers are basically is a same multiplier which is. So, here this will work on the data for v8 and this will work on the data on v6. This is what the beauty of this. So, this is how we can actually do the pipeline things and in the next time stamp, so, these operations we will finish.

So, now I can actually assign. So, this is may be multiplier 1, this is multiplier 2, this is multiplier 3, this is also multiplier 3 because these are pipelined. In the next time stamp when these two operations are done what I can do is basically my 3, 7 is available. So, I can and also 8 are available. So, I can just put because 3 and 7 has the highest length the priority. So, I am going to run them again. So, this again will run on multiplier 1, this is on multiplier 2, still multiplier 3 is running.

This is how I will do it and. So, the availability of the multiplier here is all 3 actually available. So, in this time step all 3 are available. If there are 3 multiplication operations are there I could have start their executions by the pipelining that with the existing multiplier which is already running on the different set of data. So, that is the concept. If you just do it you can actually execute the whole behavior in 6 time step lambda equal to 6.

So, with the pipeline version with same type of resource 3 and 1, I can execute this behavior in 6 clocks. So, what is the pipeline version of this example? I hope now this list scheduling is very much clear to you and you must appreciate that this is something very fast algorithm, and actually can give you good results for most of the scheduling problem for generic sequence term.

So, with this I conclude today's class. In the next class I am going to discuss about the same list scheduling for the minimizations of resource under latency bound, so, that particular problem.

Thank you.