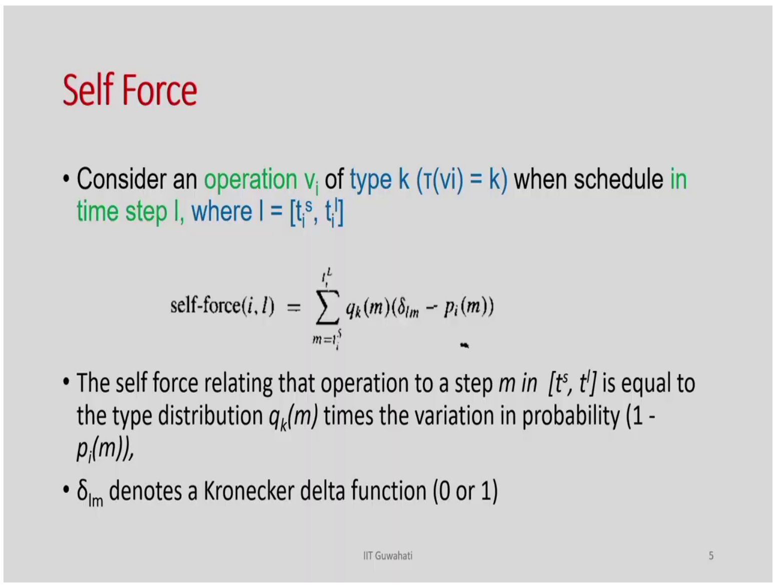
C-Based VLSI Design

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

Module - 04
C-Based VLSI Design: Scheduling
Lecture - 13
Heuristic Scheduling: Forced-directed Scheduling

Welcome students. In today's class, we will continue discussion on Force-directed Scheduling. So, in the last class, we have understood what is force? And, how to calculate the force?

(Refer Slide Time: 01:02)



Self Force

- Consider an operation v_i of type k ($\tau(vi) = k$) when schedule in time step l, where $l = [t_i^s, t_i^l]$

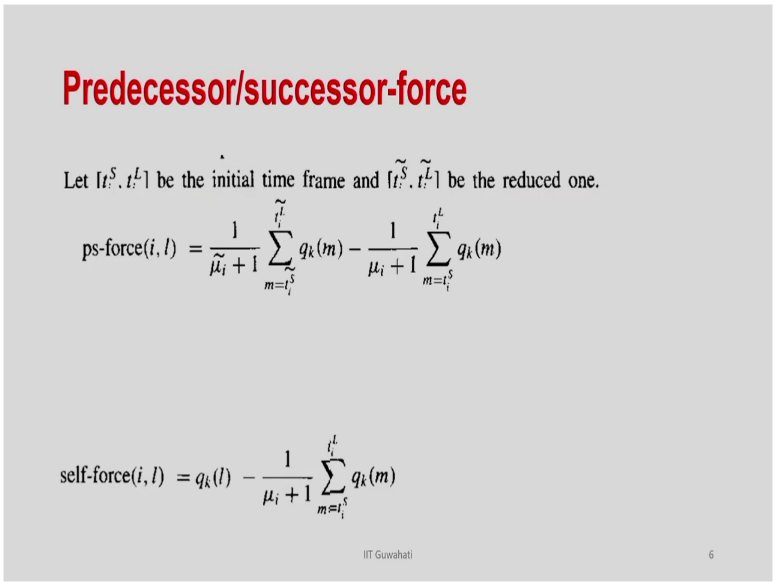
$$\text{self-force}(i, l) = \sum_{m=t_i^S}^{t_i^L} q_k(m)(\delta_{lm} - p_i(m))$$

- The self force relating that operation to a step *m in* $[t^s, t^l]$ is equal to the type distribution $q_k(m)$ times the variation in probability ($1 - p_i(m)$),
- δ_{lm} denotes a Kronecker delta function (0 or 1)

IIT Guwahati 5

So, just to recall there are two types of forces. One is the self-force.

(Refer Slide Time: 01:07)



Predecessor/successor-force

Let $[t_i^S, t_i^L]$ be the initial time frame and $[\tilde{t}_i^S, \tilde{t}_i^L]$ be the reduced one.

$$\text{ps-force}(i,l) = \frac{1}{\tilde{\mu}_i + 1} \sum_{m=\tilde{t}_i^S}^{\tilde{t}_i^L} q_k(m) - \frac{1}{\mu_i + 1} \sum_{m=t_i^S}^{t_i^L} q_k(m)$$

$$\text{self-force}(i,l) = q_k(l) - \frac{1}{\mu_i + 1} \sum_{m=t_i^S}^{t_i^L} q_k(m)$$

IIT Guwahati 6

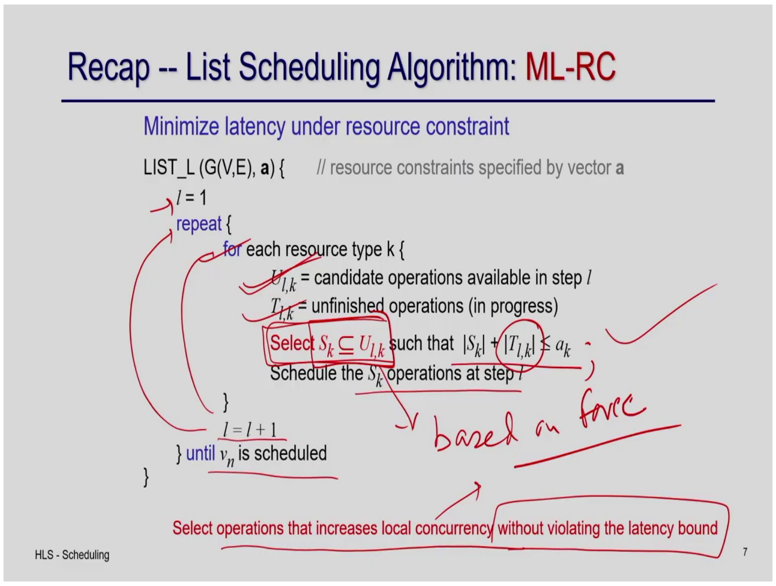
And there is another called this predecessor or successor-force. So, and then the basic idea is that if we try to schedule an operation vi in time step L. It will increase the concurrency in the time step, and it will increase the decrease the concurrency of the time other time step where it could have been scheduled.

So, this is I am going to calculate. And also once I am going to schedule this operation in time step L; it might impact the node that is its predecessor and the node that is its successor. So, how because the time frame of this the successor node might reduce because this is the last time step where this operation can be scheduled. So, that we have understood in detail in the last class.

The force I am going to calculate for each operation in each time step. Let us say it is already calculated. So, just to understand now. So, this force is something is the concurrency. And if the concurrency is high, that means, I am going to if I schedule this operation i in time step L, it will increase the concurrency and hence it will increase the resource usage.

If the concurrency is low, that means, the number of operation going to schedule in a time step will reduce or the hence it will reduce the concurrency, and hence also reduce the number of resource usage in the time step. So, these analogies let us say we already have.

(Refer Slide Time: 02:36)



Recap -- List Scheduling Algorithm: 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 $|S_k| + |T_{l,k}| \leq a_k$
            Schedule the $S_k$ operations at step $l$
        }
        $l = l + 1$
    } until $v_n$ is scheduled
}
```

based on force

Select operations that increases local concurrency without violating the latency bound

HLS - Scheduling 7

Now, I am going to think about how I can develop the both list scheduling algorithms ML-RC and MR-LC – Minimization of Latency under Resource Constraint and Minimization of Resource under Latency Constraint. So, let us now take the recap for the list scheduling algorithm for ML-RC. So, here the problem is that you already have a given the resource bound for each type of resources, and I am also given the sequence graph and I am going to schedule the operations. And if you just recap the list scheduling algorithm what was going to that? I am going to take the time step 1, I will identify for. So, I am going to take time step 1; I am going to take the each resource type. So, first say type 1, type 2, and type 3.

So, for each type of resource, I will identify what are the operations that are ready to be scheduled in this time step. And also for any other time step say l, I am going to identify the operation which is already running which is scheduled earlier and still running.

So, if the some operation is already running I have to say allocate resource for that, but and now suppose I have already allocate the resource for the running operations and still I have few resource available in a time step l. And now what I am going to do? And say there are some few operations are ready to be scheduled in this time step and which is larger than the available resource.

So, I have to take a subset of the operations from that available ready operations, and there I am actually use the priority function. The list scheduling algorithm that priority was from the distance from the sink node. So, and then so I am going to select this, and I am going to schedule this operation in the time step until my resource bound does not violate it.

So, if you recall this algorithm that for each time step basically I am starting from time step 1 for each type of resource, I will identify the ready operation, ready to be schedule. I will identify the operation that is running in this time step l of type k. And then I must schedule this operation T l, k, and I am going to select a subset of the ready operation such that my resource bound is not violated, so that something I am going to do.

And that particular this S k is the subset operations that actually start their execution in time step l. And then I will go to the next type of resource once all the resource type is already considered for the current time step, I will increase the current time step to next, and then I will go for the next time step. This is how the whole algorithm works and I am going to complete when my sink node is scheduled.

So, if I just think about the force directed scheduling for ML-RC, this is the same algorithm. The only difference is the selection. So, earlier for list scheduling whatever I have talked about, the huge series this priority function was the distance thus distance of the longest path from that node to the sink node that we have already understood.

But in force directed scheduling, this particular selection is based on the force. And we have to what is our force. So, I have a force, and force is basically concurrency. So, I am once I am going to select a subset I am going to select the operations that will increase the local concurrency that means, that has the highest force.

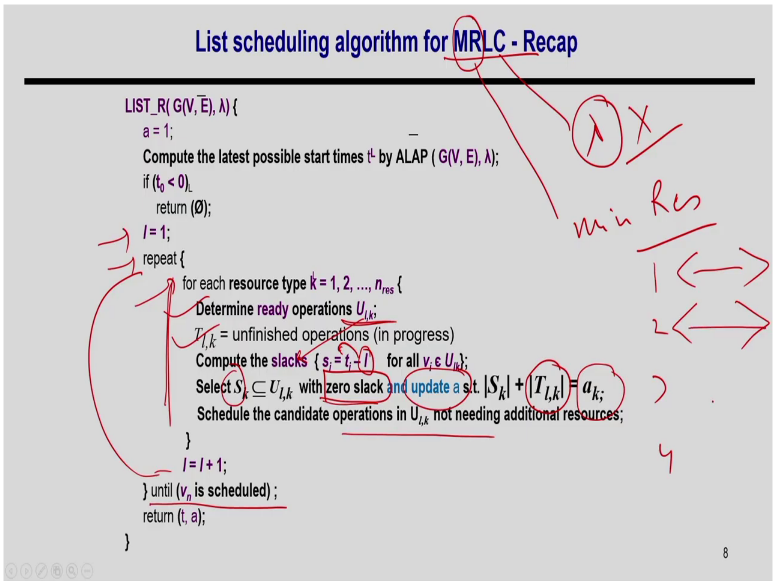
So, that means, I am going to order the nodes in terms of their force and I am going to select the node that has the high value of force. So, what is the analogy here? If I, node with high value, that means, it will increase the local concurrency, but I have to make sure that I am not violating the latency bound.

So, that means, if there are some node which is become critical to be scheduled in this time step, so that I cannot delay it further I am going to schedule it. Even if it has force is less, I am going to schedule it. And then if there are more resources available, I am going to take the nodes, which has highest this force.

So, this is so basically the just to summarize the algorithm remain the same only the selections I have is change it is based on force. And this is what I have already discuss in the previous class also that this here this algorithm is very generic, and your selection criteria can be differ. And based on the difference selection criteria, you can have different algorithm.

Once you actually select this subset of this operations from the ready list based on the force, this become force directed scheduling for minimization of latency and the resource constraint. So, I hope this you understand I am not going to in detail because I mean it is well understood now.

(Refer Slide Time: 07:09)



List scheduling algorithm for MRLC - Recap

```
LIST_R( G(V, $\bar{E}$), λ) {
    a = 1;
    Compute the latest possible start times $t^L$ by ALAP ( G(V, E), λ);
    if ($t_0$ < 0)$_L$
        return (Ø);
    $l$ = 1;
    repeat {
        for each resource type k = 1, 2, ..., $n_{res}$ {
            Determine ready operations $U_{l,k}$;
            $T_{l,k}$ = unfinished operations (in progress)
            Compute the slacks { $s_i = t_i - l$ } for all $v_i \in U_{lk}$};
            Select $S_k \subseteq U_{l,k}$ with zero slack and update a s.t. $|S_k| + |T_{l,k}| = a_k$;
            Schedule the candidate operations in $U_{l,k}$ not needing additional resources;
        }
        $l = l + 1$;
    } until ($v_n$ is scheduled) ;
    return (t, a);
}
```

8

So, now let us move for the other algorithm the MRLC. So, here the latency bound is given, and our objective was to minimization of resource. And if you recall the algorithm that I have already discuss for the list based scheduling algorithm. So because I do not know what is the resource bound what I have assumed that initially I have only one instance for each resource.

And whenever I found certain operation become very critical so if I do not schedule this operation in this time step, then it will increase the it would not it would not support this latency bound, then I by increase the resource and I schedule this operation there; otherwise I do not schedule. If in some time step, there are some resources available I am always going to schedule those operations if they are not critical.

So, if you just remember the operations, again I am starting from the time step 1. Now, for each time step, for each type of resource, I identify the operations that are ready to be scheduled, and I also identify the operation that running. And I am going to schedule this running operation must.

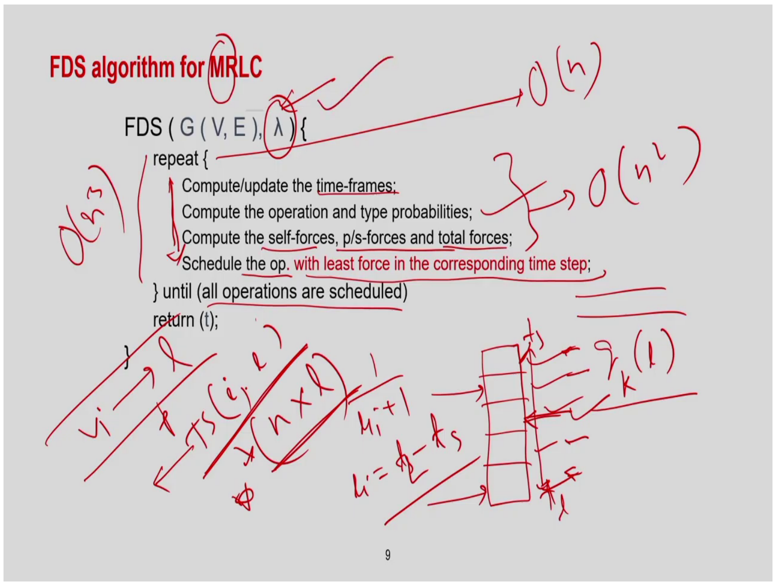
And then I identify among this ready list what is the slack value. Slack means what is the still what is the time frame here still can be scheduled which is basically the current time step minus t l the ALAP time. And if it becomes 0, it is 0 slack that mean it must be scheduled there. So, I have to sub select the subset of this S l k for where this slack was 0.

And if my resource bound is not supporting this, I am going to update I will increase my resource, so that this S k must be scheduled. And if say that is no zero slack operation and I have already have some resource, then I am going to can select the operations which is has non-zero slack also, but without violating my resource bound.

So, this is how I am going to do it for each type of resource, and then I will increase the time, and I will go for the next time and so on until the sink node is scheduled. So, this is the list scheduling algorithm for minimization of resource under latency constraint.

This is basically very greedy algorithm that I always assume that I have a minimum resource available. And whenever I am forced to increase the resource, I am going to increase the resource, and this way I am going to do it.

(Refer Slide Time: 09:33)



FDS algorithm for MRLC

```
FDS ( G ( V, E ), λ ) {
    repeat {
        Compute/update the time-frames;
        Compute the operation and type probabilities;
        Compute the self-forces, p/s-forces and total forces;
        Schedule the op. with least force in the corresponding time step;
    } until (all operations are scheduled)
    return (t);
}
```

$O(n)$
$O(n^2)$
$O(n^3)$
$v_i \rightarrow \lambda$
$ts(i, \lambda)$
$n \times \lambda$
ϕ
$\frac{1}{\mu_i + 1}$
$\mu_i = t_l - t_s$
$q_k(l)$
t_s
t_l

9

But this force directed scheduling for MRLC is completely different and it is very interesting. Here if you remember that you basically I am going to take the time step 1, then 2, then 3 and 4. So, in time step 1, I identify the operation that can be schedule in time step 1, then I move on to time step 2, I will identify the operation that can be schedule in time step 2 and so on. But here I am not going to go by the at time step wise.

What I am going to do? I am going to schedule an operation in every iteration. So, it does not have two loops. Earlier it was kind of two loops. Now, what I am going to do here, I am going to in each time step I will identify a operation I am going to schedule it, and the next iteration I am going to again identify another operation and schedule it.

So, basically this will repeat for order of n because this loop will execute for order of n and because every time every iteration I am going to schedule one operation. And then what I am going to do inside this loop? So, the id is very interesting. So, what I am going to do you already remember that for every node there is a range where it can actually can be scheduled which is by t s to t l that we already understood.

So, this is the time frame. So, for each node, I will identify the time frame and then I will identify the operation distribution and type probability. So, for each operation, I can identify the probability what is the probability of the operation to be scheduled in each time step which is basically 1 by mu i plus 1. You recall that.

So, this so mu i basically nothing but t l minus t s, so that I already calculated. And then for each time step, we have already identified the terminology called q k l that each time step I will identify the total time distribution of operator type k. So, I am going to calculate all those things.

So, I will identify the probability, I will identify the probability distribution, and then I will identify the type distributions. And then I am going to calculate the self-force, and that predecessor or successor-force, and total forces.

So, you remember that self-force is something I will calculate because this self-force is this operation was could have been schedule anywhere, but if I try to schedule it here it will impact the concurrency here, here, here, here, and here, it will impact positively here and negatively here, so that I will calculate.

And then force predecessor and successor-force is that if I schedule it here it will impact the overall scheduling of the predecessors nodes and the successor node, I will calculate them. And then this is the total force is nothing but this plus this. So, the idea is that in each time step, I am going to calculate the total force for each node. And basically it is not only that.

So, it is basically it is the total force of node i in each time step while it is schedulable. So, for each node, it is not a single value also. For each node also there are many values because if it you will get a value if I schedule it here, I will get it total force if I try to decide to schedule it here. I will get another total value if I try to schedule it here.

I will have this basically v sorry n into l number of total forces. So, this is already I can calculate in this three time step. And now because what is my objective here? My objective here is to reduce the resource my minimization of resource. So, what I am going to do it now I am going to select the operation. So, say I am going to select the total force which is the minimum. So, I have n into l number of total forces.

And if I say select one such which is the minimum value among all the total forces. And what does it integrate? So, that suppose I have selected T i l, that means, now operation V i is going to schedule in time step l, so that is what is going to happen here. So, I am going to select an operation with least force among all these total forces.

So, that is the beauty of this algorithm that every time step, I am not going by what time step 1, 2, 3 and 4 what I am doing here, I will calculate the total force for all nodes in all possible time step. All possible time steps are basically the time zone or the time frame while it can be scheduled. And then I am going to find out among all these values for all nodes n into l values what is the minimum value.

So, if from what is the minimum value say suppose I got this minimum value for T s i l, what does it mean? That now I am going to schedule this operation i in time step l. So, basically in every time step, I decide the schedule of one operation. And then what is is once you fixed it again your time frame of all node successor predecessor node will change, the number of node will also change because one node is already fixed.

So, I have to recalculate the whole thing again for all remaining nodes. And again I am going to select the least one least force, and I am going to schedule that operation in the time step. So, this I am going to keep doing it until my all operations are scheduled.

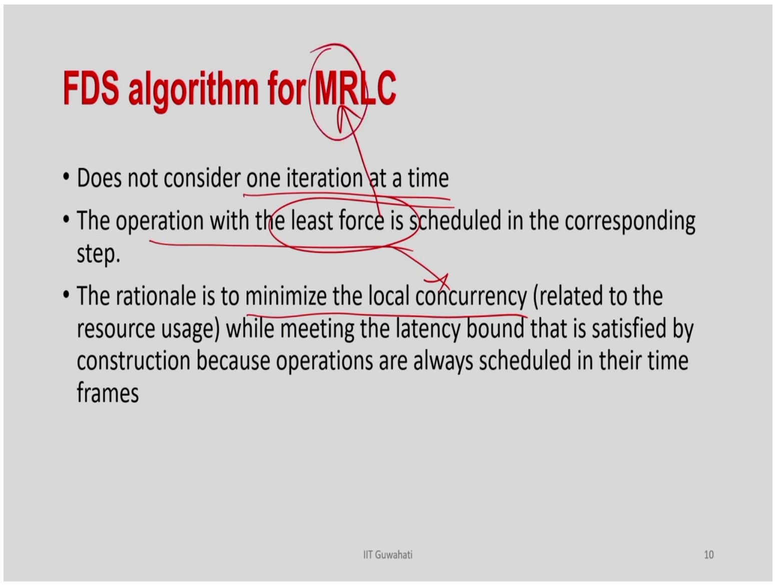
So, you already understood that in the previous class this is order of n square because this because of this predecessor calculation, it is predecessor force and successor force calculation was complex, so it is n square. So, the overall complexity of this algorithm is order of n cube, but this algorithm is completely different from the MRLC algorithm which is based on the list scheduling because here I am not going by time stepwise I am going operation wise.

And how this does this guarantee that it can say made by latency bound? Because all this value is calculated the time frame everything based on this lambda. Because this is I mean if a lambda must be greater than of its ALAP. If it is greater than of its ALAP or if it is the schedule is does not possible by lambda it will say it is not feasible by your ALAP start way.

So, since all this value is getting calculated, considering this lambda is feasible, and I am not scheduling anything any operation beyond its time frame. So, whatever the time frame is given by ASAP and ALAP, I am going to schedule this operation one of the time step.

So, it ensures that the way I am scheduling it never violate by lambda because this lambda is already validated by t l a p and t s a p and then whatever the time frame I am considering for the operations all are actually supporting this lambda. And hence what the schedule that is going to happen always may satisfy the time bound or the latency bound.

(Refer Slide Time: 16:20)



FDS algorithm for MRLC

- Does not consider one iteration at a time
- The operation with the least force is scheduled in the corresponding step.
- The rationale is to minimize the local concurrency (related to the resource usage) while meeting the latency bound that is satisfied by construction because operations are always scheduled in their time frames

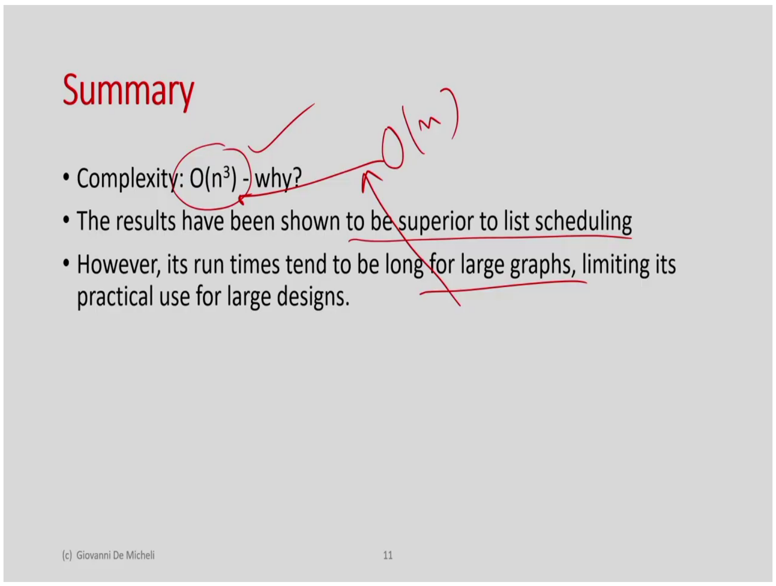
IIT Guwahati 10

So, what is understood that this does not take one iteration at a time rather it take one operation at a time. The operation with the least force is schedule in that corresponding time step. And what is the idea here, why I am taking the least one? Because, I want to reduce the resource. So, the minimization of the local concurrency, if I take the least force, it has the minimum impact on the concurrency.

And since my objective is to reduce the concurrency is basically the number of resource. If there are three operations running in a time step that means I need three resources, but if the concurrency is less say 1, that means, I need one resource. So, if I reduce the concurrency, so then my resource requirement will be less.

So, that is why I am going to select the operation with a least concurrent least force. So that means, it is actually giving the minimization of the local concurrency and is nothing but the minimization of resource. So, minimization of local concurrency is nothing but the minimization of resource. So, selecting the operations by the least force ensure that it will actually minimization of the resource.

(Refer Slide Time: 17:31)



Summary

- Complexity: $O(n^3)$ - why?
- The results have been shown to be superior to list scheduling
- However, its run times tend to be long for large graphs, limiting its practical use for large designs.

(c) Giovanni De Micheli 11

So, that is the overall idea. So, in summary, so we have understood two algorithms in today's class. One is the list forced-directed scheduling for MLRC Minimization of Latency under Resource Constraint. And we have understood that is the same algorithm which I have talked about for list scheduling.

And only thing is that my selection criteria here based on the force, and I am going to select the node which has the highest force because my objective is to reduce the latency and hence increase the concurrency. If you run many nodes in parallel, my latency will automatically decreases. So, that is the analogy there.

But for this MRLC whenever the latency bound is given, you want to reduce the minimize the resource. The idea here is that you take you schedule one operation at a time which has the least force, so that means, every iteration I am making sure that I am increasing this concurrency by minimum way. So, and hence my algorithm will give me the minimization of resource. So, and we have seen that complexities order of n cube.

So, whatever is found that this results given by this force scheduling is much better than the list scheduling, but only thing is that the complexity is quite high because in list scheduling it was order of n and here it is order of n cube so for this MRLC. So, for large graph, it will take more time compared to this.

Although it is give very good results better result than scheduling, but sometime this because of the complexity it may not be applicable to for very large design, but for molar design it always give me better result than list scheduling. So, with this, I conclude today's class.

Thank you.