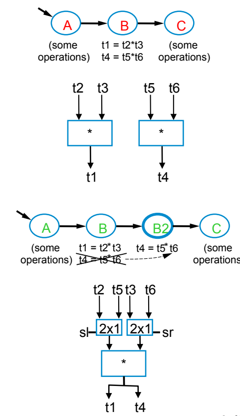
ECE 474A/57A
Computer-Aided Logic Design

Behavioral Synthesis
Scheduling Algorithms

Scheduling

- What have we done?
  - Specified the order in which operations are performed
- What's next - Operator Scheduling
  - Assigning operations performed in each states
  - Generally speaking, determining the start time of each task/operation
- Why is scheduling important?
  - Determines the amount of concurrency of the resulting implementation – effects performance
  - Maximum amount of concurrent operations of a given type at any time step also determines the amount of hardware resources of that type required – effects area



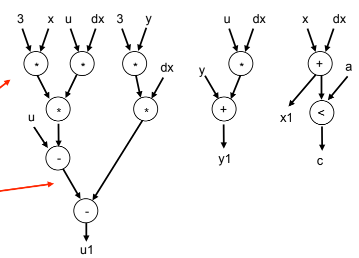
Control/Data flow graph (CDFG)

- How do we represent a scheduling?
- Dataflow graph – represents the way data flows through a computation
  - Computations limited to 2-input

Code fragment we want to implement

x1 = x + dx;
u1 = u – (3 * x * u * dx) – (3 * y * dx)
y1 = y + u * dx
c = x1 < a

Corresponding CDFG



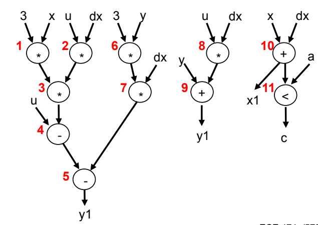
Vertices correspond to the operations we perform

Edges correspond to the dependencies

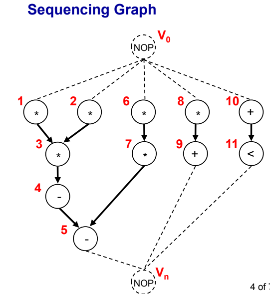
Task Representation - Sequencing graph

- Determining a schedule
  - Don't really care about the actual input/output values
  - Just want to know the task we perform (add, subtract, compare, etc..) and the dependencies among the task
- Utilize a Sequencing graph

CDFG



Sequencing Graph



Sequencing graph

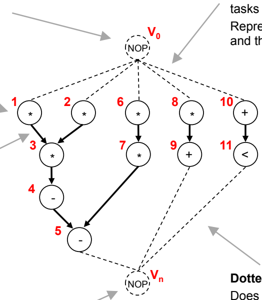
Source node
Represented by a NOP (no operation) node
Indicates start of computation

Vertices
Represents the task or operation to be performed

Edges
Represents dependencies among operations
Cannot perform operation 3 until operation 1 and 2 are completed

Sink node
Represented by a NOP (no operation) node
Indicates completion of computation

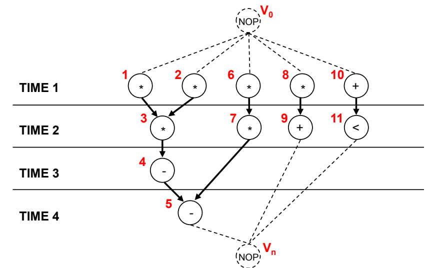
Dotted lines
Does NOT represent a dependency between tasks or operations
Represents a connection between the source and the initial task or operations



Dotted lines
Does NOT represent a dependency between tasks or operations
Represents a connection between the sink and the final task or operations

Scheduling - Sequencing Graphs

- Sequencing graph itself only specifies the dependencies among tasks
- Scheduling requires we associate a start time for each task/operation in the sequencing graph
  - Introduce concept of time



Operation 1, 2, 6, 8, and 10 are performed in the first clock cycle

Operation 3, 7, 9, and 11 are performed in the second clock cycle

Operation 4 is performed in the third clock cycle

Operation 5 is performed in the fourth clock cycle

Notice all operations require only 1 clock cycle to execute

ASAP Scheduling

- Unconstrained minimum-latency scheduling problem
  - We have infinite resources, all we want is the minimum time to perform the computation
  - Commonly referred to as ASAP (as soon as possible) scheduling

ASAP($G_S(V,E)$){
Schedule v_0 by setting $t_0 = 1$
repeat{
Select a vertex v_i whose predecessors are all scheduled;
Schedule v_i by setting $t_i = \max_{j:(v_j, v_i) \in E} t_j + d_j$
}
until (v_n is scheduled);
return t;
}

Perform ASAP scheduling on the sequencing graph

Schedule the source node v_0 for time 1

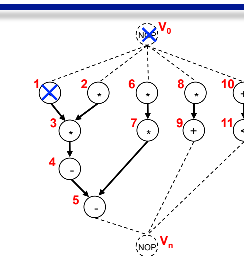
Look for tasks/operations that are not dependent on a task/operation that hasn't been scheduled yet

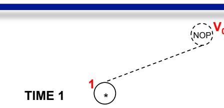
Schedule the task/operation to time = time predecessor scheduled for + time required for predecessor to execute

may have multiple predecessors, take maximum time

Keep going until we have scheduled the sink node v_n

ASAP Scheduling
Example 1





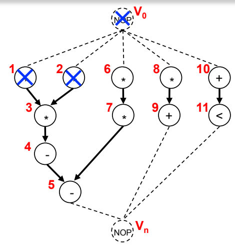
Step1
Schedule v_0 at time 1

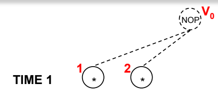
Step2
Select a vertex v_i whose predecessors are all scheduled — **All of v_1 predecessors are scheduled**

Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute — **Time = v_0 start time + v_0 execution time = 1 + 0 = 1**

Step4
Has v_n been scheduled yet? — **No. Repeat loop.**

ASAP Scheduling
Example 1



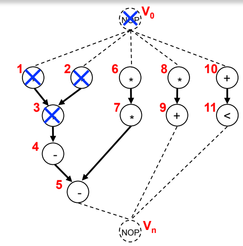


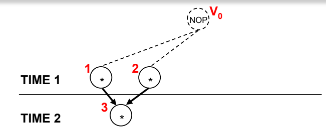
Step2
Select a vertex v_i whose predecessors are all scheduled — All of v_2 predecessors are scheduled

Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute — Time = v_0 start time + v_0 execution time
= 1 + 0
= 1

Step4
Has v_n been scheduled yet? — No. Repeat loop.

ASAP Scheduling
Example 1





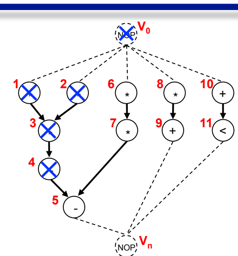
Step2
Select a vertex v_i whose predecessors are all scheduled — All of v_3 predecessors are scheduled

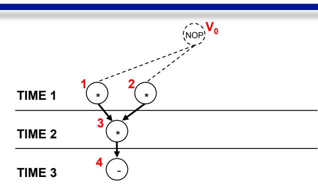
Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute — Time = v_1 start time + v_1 execution time
= 1 + 1
= 2

Time = v_2 start time + v_2 execution time
= 1 + 1
= 2

Step4
Has v_n been scheduled yet? — No. Repeat loop.

ASAP Scheduling
Example 1



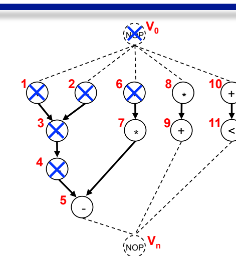


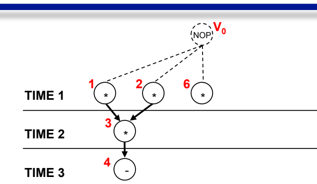
Step2
Select a vertex v_i whose predecessors are all scheduled — All of v_4 predecessors are scheduled

Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute — Time = v_3 start time + v_3 execution time
= 2 + 1
= 3

Step4
Has v_n been scheduled yet? — No. Repeat loop.

ASAP Scheduling
Example 1



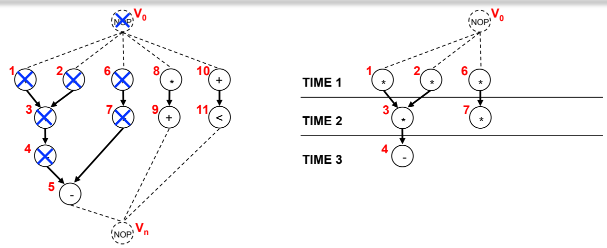


Step2
Select a vertex v_i whose predecessors are all scheduled — v_5 still has predecessors not scheduled (v_7), skip for now
All of v_6 predecessors are scheduled

Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute — Time = v_0 start time + v_0 execution time
= 1 + 0
= 1

Step4
Has v_n been scheduled yet? — No. Repeat loop.

ASAP Scheduling
Example 1

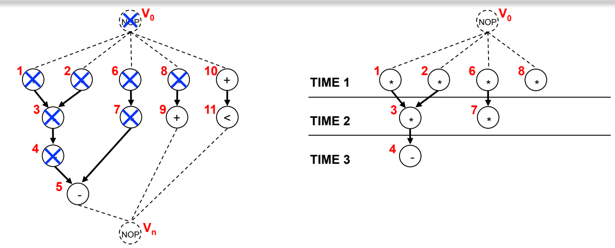


Step2
Select a vertex v_i whose predecessors are all scheduled — All of v_7 predecessors are scheduled

Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute — Time = v_6 start time + v_6 execution time = 1 + 1 = 2

Step4
Has v_n been scheduled yet? — No. Repeat loop.

ASAP Scheduling
Example 1

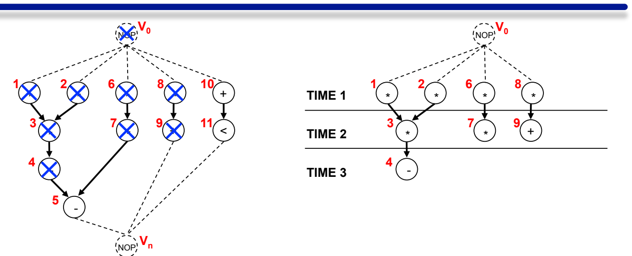


Step2
Select a vertex v_i whose predecessors are all scheduled — All of v_8 predecessors are scheduled

Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute — Time = v_0 start time + v_0 execution time = 1 + 0 = 1

Step4
Has v_n been scheduled yet? — No. Repeat loop.

ASAP Scheduling
Example 1

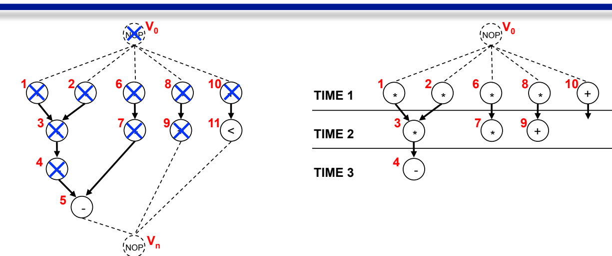


Step2
Select a vertex v_i whose predecessors are all scheduled — All of v_9 predecessors are scheduled

Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute — Time = v_8 start time + v_8 execution time = 1 + 1 = 2

Step4
Has v_n been scheduled yet? — No. Repeat loop.

ASAP Scheduling
Example 1

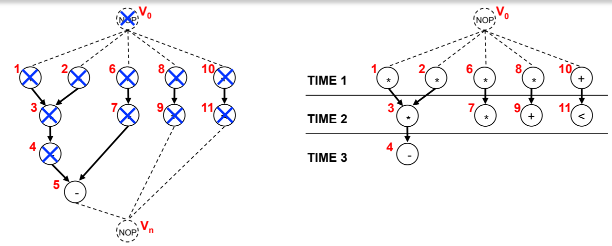


Step2
Select a vertex v_i whose predecessors are all scheduled — All of v_{10} predecessors are scheduled

Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute — Time = v_0 start time + v_0 execution time = 1 + 0 = 1

Step4
Has v_n been scheduled yet? — No. Repeat loop.

ASAP Scheduling
Example 1



Step2
Select a vertex v_i whose predecessors are all scheduled

All of v_{11} predecessors are scheduled

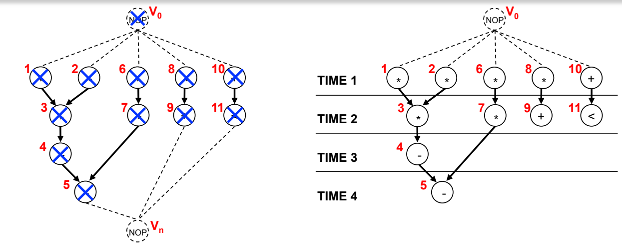
Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute

Time = v_{10} start time + v_{10} execution time
= 1 + 1
= 2

Step4
Has v_n been scheduled yet?

No. Repeat loop.

ASAP Scheduling
Example 1



Step2
Select a vertex v_i whose predecessors are all scheduled

v_n still has predecessors not scheduled (v_5), skip for now
Return to v_5, all predecessors are scheduled

Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute

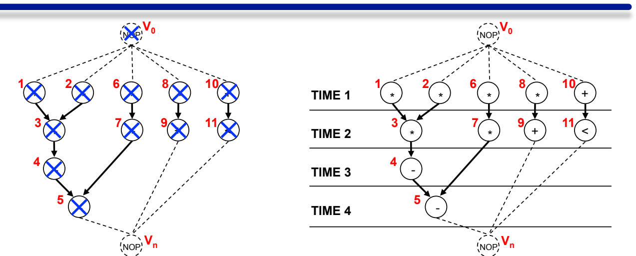
Time = v_4 start time + v_4 execution time
= 3 + 1
= 4

Time = v_7 start time + v_7 execution time
= 2 + 1
= 3

Step4
Has v_n been scheduled yet?

No. Repeat loop.

ASAP Scheduling
Example 1



Step2
Select a vertex v_i whose predecessors are all scheduled

Return to v_n, all predecessors are scheduled

Step3
Schedule v_i to time = predecessor's scheduled time + time required for predecessor to execute

Time = v_5 start time + v_5 execution time
= 4 + 1
= 5

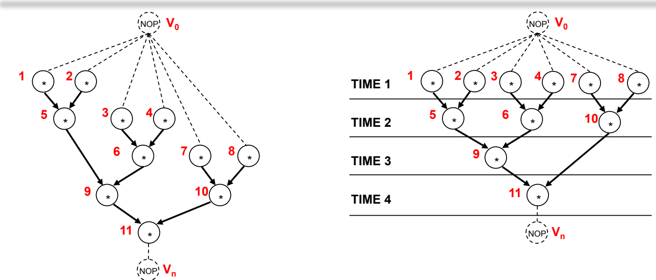
Time = v_9 start time + v_9 execution time
= 2 + 1
= 3

Time = v_{11} start time + v_{11} execution time
= 2 + 1
= 3

Step4
Has v_n been scheduled yet?

Yes. We are done!

ASAP Scheduling
Example 2

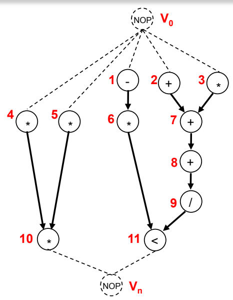


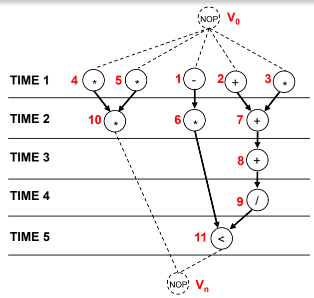
ASAP Scheduling goal is to schedule tasks/operations to perform as soon as possible

We can skip the algorithm and visually move vertices "up" as far as possible

ASAP Scheduling

Example 3





ALAP Scheduling

- Latency constrained scheduling problem
  - Schedule must satisfy an upper bound on latency
  - Commonly referred to as ALAP (as late as possible) scheduling

```
ALAP( G_S(V,E), λ ){
   Schedule v_n by setting t_n = λ + 1
   repeat{
      Select a vertex v_i whose successors are all scheduled;
      Schedule v_i by setting t_i = min          t_j - d_j
                                   j:(v_i, v_j) єE
   }
   until (v_0 is scheduled);
   return t;
}
```

Perform ALAP scheduling on the sequencing graph, λ is the upper time bound

Schedule the sink node v_n for upper latency bound + 1

Look for tasks/operations whose successors are already scheduled

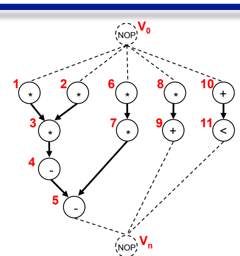
Schedule the task/operation to time = time successor scheduled for - time required for successor to execute

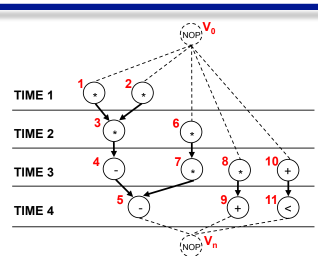
may have multiple successors, take minimum time

Keep going until we have scheduled the source node v_0

ALAP Scheduling

Example 1



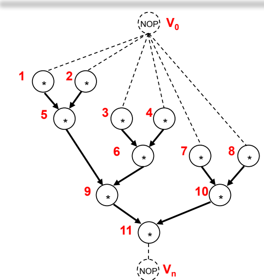


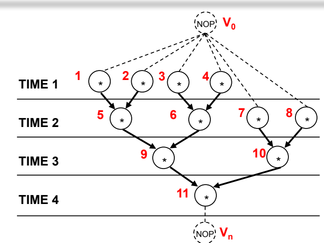
ALAP Scheduling goal is to schedule tasks/operations to perform as late as possible

We can skip the algorithm and visually move vertices “down” as far as possible

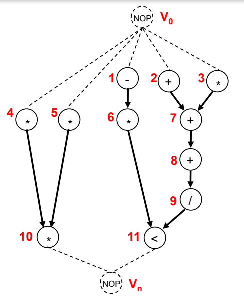
ALAP Scheduling

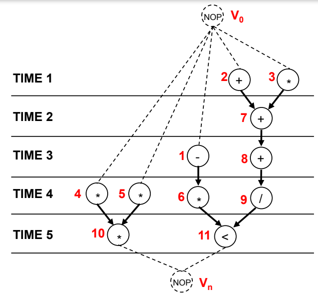
Example 2





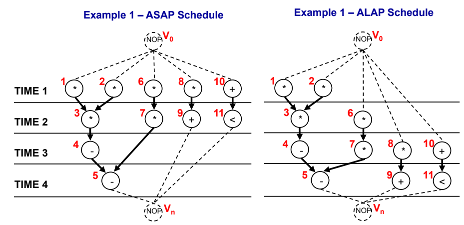
ALAP Scheduling
Example 3





Mobility

- Mobility (or slack) important quantity used by some scheduling algorithms
  - Mobility = start time $_{\text{ALAP scheduling}}$ – start time $_{\text{ASAP scheduling}}$
  - Mobility = 0, task/operation can only be started at the given time in order to meet overall latency constraint
  - Mobility > 0, indicates span of possible start times
    - Helps with minimizing resources (adders, multipliers, etc.)



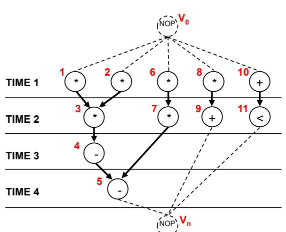
V_1 mobility = $\text{time}_{ALAP}(V_1)$ - $\text{time}_{ASAP}(V_1)$
= 1 – 1
= 0

V_6 mobility = $\text{time}_{ALAP}(V_6)$ - $\text{time}_{ASAP}(V_6)$
= 2 – 1
= 1

V_{11} mobility = $\text{time}_{ALAP}(V_{11})$ - $\text{time}_{ASAP}(V_{11})$
= 4 – 2
= 2

Mobility
Example 1 – ASAP Only

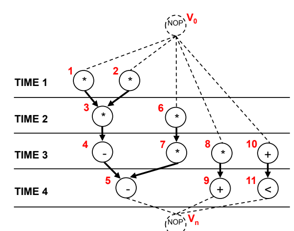
- What do we get with the ASAP Schedule?
  - Latency = 4
  - Resource requirement = 4 multipliers, 2 ALUs



4 multiply operations, 1 ALU operation
2 multiply operations, 2 ALU operations
1 ALU operation
1 ALU operation

Mobility
Example 1 – ALAP Only

- What do we get with the ALAP Schedule?
  - Latency = 4
  - Resource requirement = 2 multipliers, 3 ALUs



2 multiply operations
2 multiply operations
2 multiply operations, 2 ALU operations
3 ALU operations

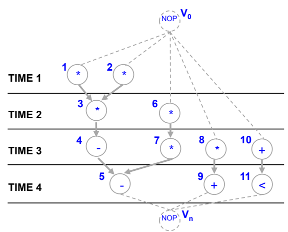
Mobility
Example 1 – Modify ALAP

- Start with ALAP schedule
- Use mobility to try to improve resource requirements

Operations with mobility = 0
v_1, v_2, v_3, v_4, v_5

Operations with mobility = 1
v_6, v_7

Operations with mobility = 2
v_8, v_9, v_{10}, v_{11}



2 multiply operations
2 multiply operations
2 multiply operations, 2 ALU operations
3 ALU operations

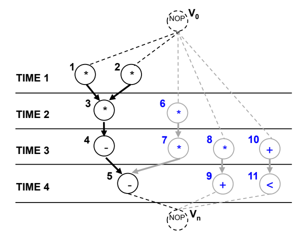
Mobility
Example 1 – Modify ALAP

- Start with ALAP schedule
- Use mobility to try to improve resource requirements
  - Vertices with mobility = 0 cannot be moved, they are part of the critical path

Operations with mobility = 0
v_1, v_2, v_3, v_4, v_5

Operations with mobility = 1
v_6, v_7

Operations with mobility = 2
v_8, v_9, v_{10}, v_{11}



2 multiply operations
2 multiply operations
2 multiply operations, 2 ALU operations
3 ALU operations

Mobility
Example 1 – Modify ALAP

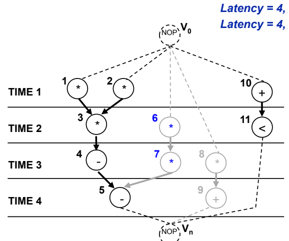
- Start with ALAP schedule
- Use mobility to try to improve resource requirements
  - Vertices with mobility = 0 cannot be moved, they are part of the critical path
  - Vertices with mobility > 0 can be moved to minimize resource requirements

Latency = 4, Resources = 2 multipliers, 2 ALUs
Latency = 4, Resources = 4 multipliers, 2 ALUs (ASAP)
Latency = 4, Resources = 2 multipliers, 3 ALUs (ALAP)

Operations with mobility = 0
v_1, v_2, v_3, v_4, v_5

Operations with mobility = 1
v_6, v_7

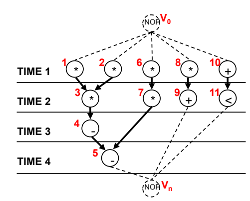
Operations with mobility = 2
v_8, v_9, v_{10}, v_{11}



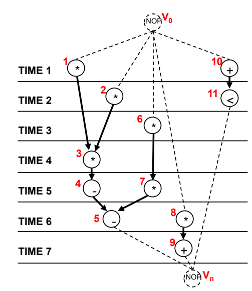
2 multiply operations, **1 ALU operations**
2 multiply operations, **1 ALU operations**
2 multiply operations, ~~2~~ 1 ALU operations
~~3~~ 2 ALU operations

Resource Constrained Scheduling

- Resource constrained scheduling problem
  - Resource usage determines circuit area
  - Consider area/latency tradeoff



ASAP schedule determines the minimum latency, we assumed infinite resources



We can determine a schedule to consider only minimizing resources – assuming latency doesn't matter

Likely we want something in between

Hu's Algorithm

- Exact (polynomial-time) algorithm for resource constrained scheduling
  - Assumes one resource handles all possible operations
  - Assumes all operations have 1 unit delay

HU($G_S(V,E)$, a){ ← Value of a indicates the number of resources we have available

Label the vertices; ← Label with distance of vertices to sink node

***l* = 1;** ← Indicates the time step

repeat {

***U* = unscheduled vertices in V without predecessors or whose predecessors have been scheduled;** ← Make a list of all vertices not waiting on another operation to be scheduled

Select S ⊆ U vertices, such that |S| ≤ a and labels in S are maximal; ← Select a subset of the vertices in U, no more than a, choosing vertices with largest labels

Schedule the S operations at step *l* by setting t = *l* ∀ *i* : $v_i \in S$; ← Schedule the vertices in the subset to start at time l

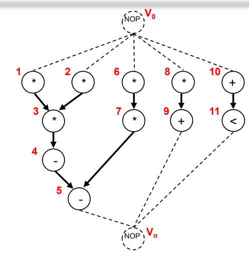
***l* = *l* + 1;** ← update l to next time step

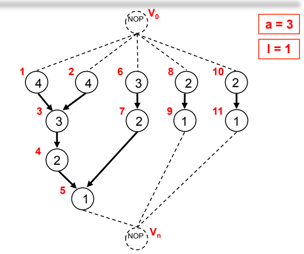
} until (v_n is scheduled); ← Keep going until we have scheduled the sink node v_n

return t;

}

Hu's Algorithm
Example 1

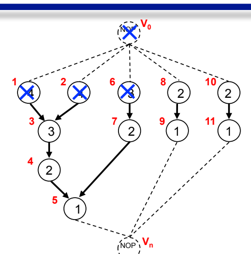


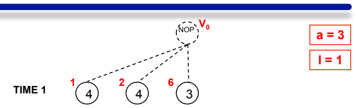


Step1
Label all vertices with distance to sink

Step2
l = 1

Hu's Algorithm
Example 1





Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — **U = { v_1, v_2, v_6, v_8, v_{10} }**

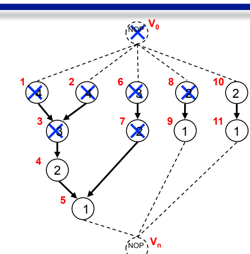
Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — **S = { v_1, v_2, v_6 }**

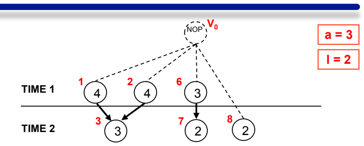
Step 5
Schedule vertices in S to time step l — **Set vertices in S to start at 1**

Step 6
l = l + 1 — **l = 1 + 1 = 2**

Step 7
Has v_n been scheduled yet? — **No. Repeat loop.**

Hu's Algorithm
Example 1





Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — **U = { v_3, v_7, v_8, v_{10} }**

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — **S = { v_3, v_7, v_8 }**

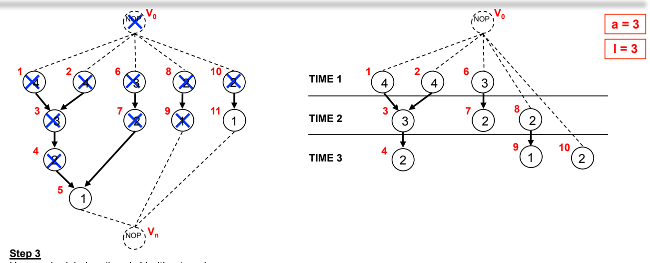
Step 5
Schedule vertices in S to time step l — **Set vertices in S to start at 2**

Step 6
l = l + 1 — **l = 2 + 1 = 3**

Step 7
Has v_n been scheduled yet? — **No. Repeat loop.**

Hu's Algorithm

Example 1



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — **U = { v_4, v_9, v_{10} }**

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — **S = { v_4, v_9, v_{10} }**

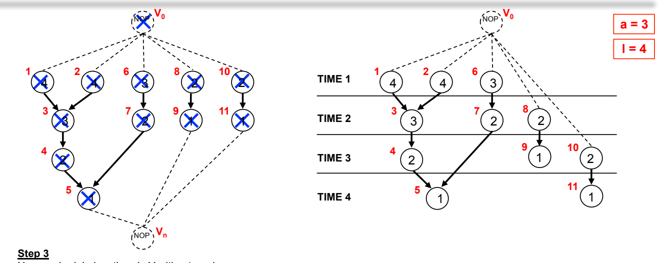
Step 5
Schedule vertices in S to time step l — **Set vertices in S to start at 3**

Step 6
l = l + 1 — **l = 3 + 1 = 4**

Step 7
Has v_n been scheduled yet? — **No. Repeat loop.**

Hu's Algorithm

Example 1



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — **U = { v_5, v_{11} }**

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — **S = { v_5, v_{11} }**

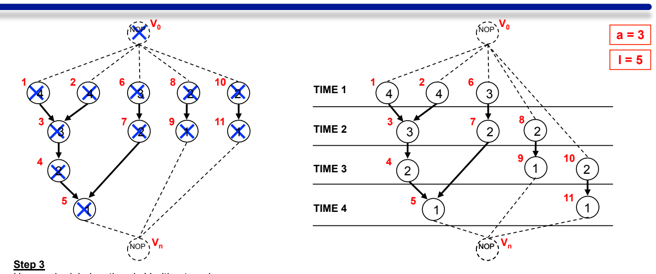
Step 5
Schedule vertices in S to time step l — **Set vertices in S to start at 4**

Step 6
l = l + 1 — **l = 4 + 1 = 5**

Step 7
Has v_n been scheduled yet? — **No. Repeat loop.**

Hu's Algorithm

Example 1



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — **U = { v_N }**

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — **S = { v_N }**

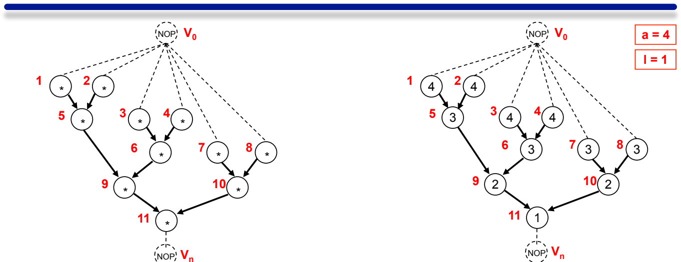
Step 5
Schedule vertices in S to time step l — **Set vertices in S to start at 5**

Step 6
l = l + 1 — **l = 5 + 1 = 6**

Step 7
Has v_n been scheduled yet? — **Yes. We are done!**

Hu's Algorithm

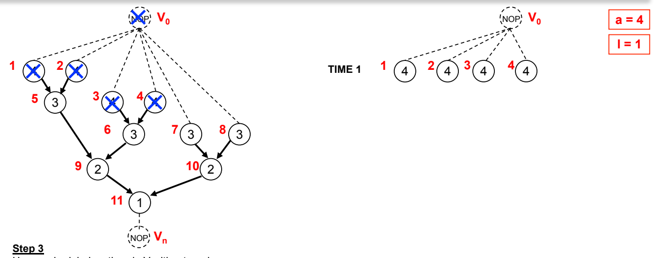
Example 2



Step1
Label all vertices with distance to sink

Step2
l = 1

Hu's Algorithm
Example 2



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — U = { v_1, v_2, v_3, v_4, v_7, v_8}

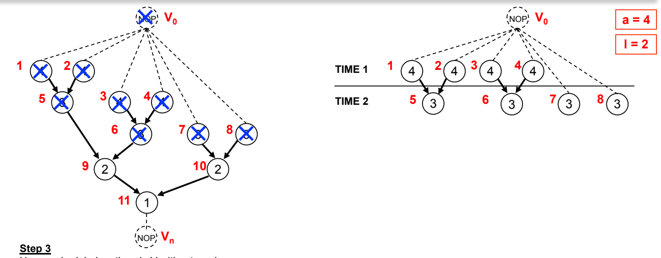
Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — S = { v_1, v_2, v_3, v_4}

Step 5
Schedule vertices in S to time step l — Set vertices in S to start at 1

Step 6
l = l + 1 — l = 1 + 1 = 2

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

Hu's Algorithm
Example 2



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — U = { v_5, v_6, v_7, v_8 }

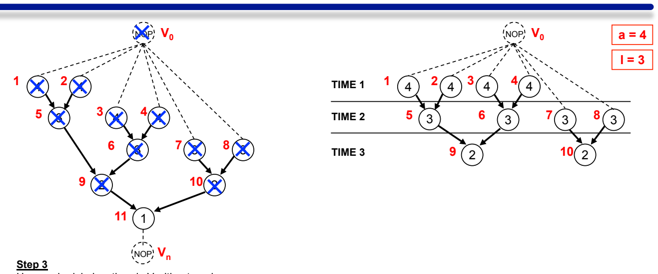
Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — S = { v_5, v_6, v_7, v_8}

Step 5
Schedule vertices in S to time step l — Set vertices in S to start at 2

Step 6
l = l + 1 — l = 2 + 1 = 3

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

Hu's Algorithm
Example 2



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — U = { v_9, v_{10} }

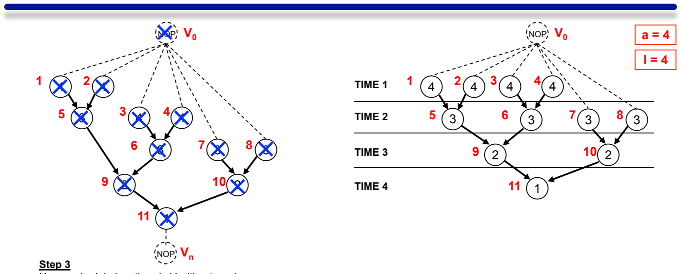
Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — S = { v_9, v_{10} }

Step 5
Schedule vertices in S to time step l — Set vertices in S to start at 3

Step 6
l = l + 1 — l = 3 + 1 = 4

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

Hu's Algorithm
Example 2



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — U = { v_{11} }

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — S = { v_{11} }

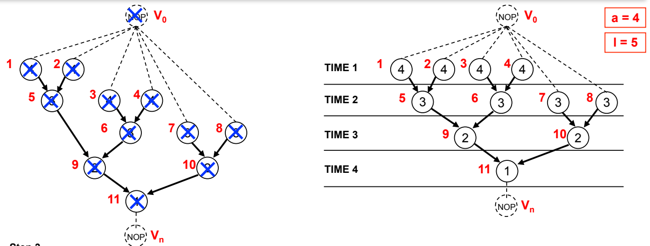
Step 5
Schedule vertices in S to time step l — Set vertices in S to start at 4

Step 6
l = l + 1 — l = 4 + 1 = 5

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

Hu's Algorithm

Example 2



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — U = { v_n }

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — S = { v_n }

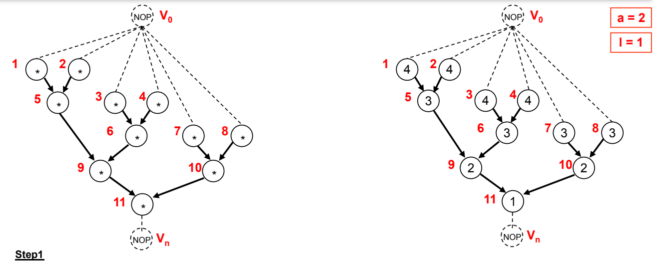
Step 5
Schedule vertices in S to time step l — Set vertices in S to start at 5

Step 6
l = l + 1 — l = 5 + 1 = 6

Step 7
Has v_n been scheduled yet? — Yes. We are done.

Hu's Algorithm

Example 3

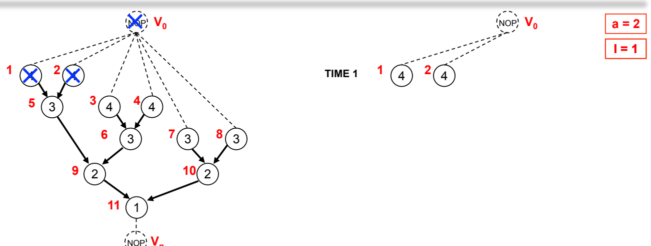


Step1
Label all vertices with distance to sink

Step2
l = 1

Hu's Algorithm

Example 3



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — U = { v_1, v_2, v_3, v_4, v_7, v_8 }

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — S = { v_1, v_2 }

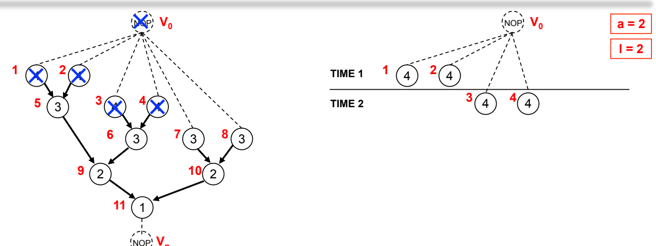
Step 5
Schedule vertices in S to time step l — Set vertices in S to start at 1

Step 6
l = l + 1 — l = 1 + 1 = 2

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

Hu's Algorithm

Example 3



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — U = { v_3, v_4, v_5, v_7, v_8 }

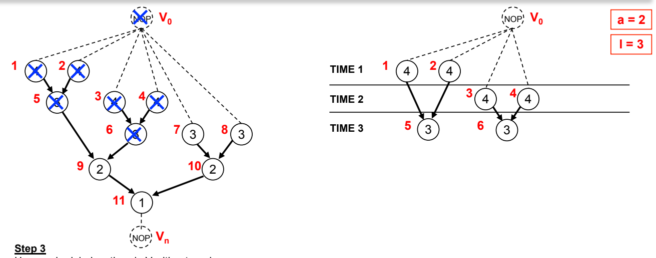
Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — S = { v_3, v_4 }

Step 5
Schedule vertices in S to time step l — Set vertices in S to start at 2

Step 6
l = l + 1 — l = 2 + 1 = 3

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

Hu's Algorithm
Example 3



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — U = { v_5, v_6, v_7, v_8 }

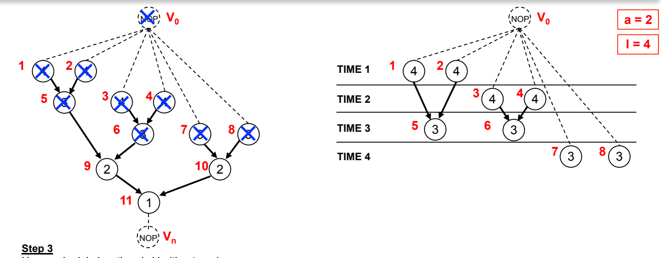
Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — S = { v_5, v_6 }

Step 5
Schedule vertices in S to time step l — Set vertices in S to start at 3

Step 6
l = l + 1 — l = 3 + 1 = 4

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

Hu's Algorithm
Example 3



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — U = { v_7, v_8, v_9 }

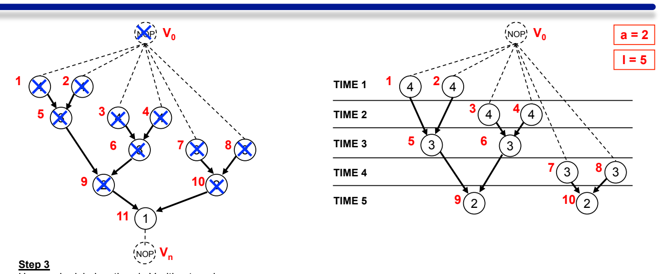
Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — S = { v_7, v_8 }

Step 5
Schedule vertices in S to time step l — Set vertices in S to start at 4

Step 6
l = l + 1 — l = 4 + 1 = 5

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

Hu's Algorithm
Example 3



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — U = { v_9, v_{10} }

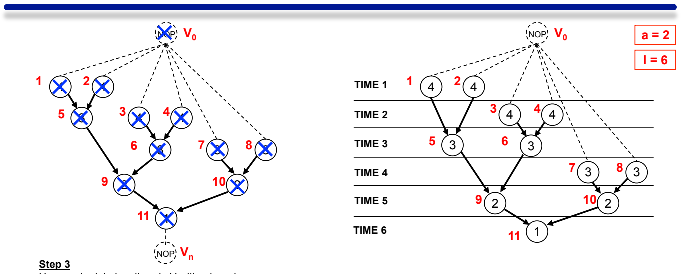
Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — S = { v_9, v_{10} }

Step 5
Schedule vertices in S to time step l — Set vertices in S to start at 5

Step 6
l = l + 1 — l = 5 + 1 = 6

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

Hu's Algorithm
Example 3



Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled — U = { v_{11} }

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal — S = { v_{11} }

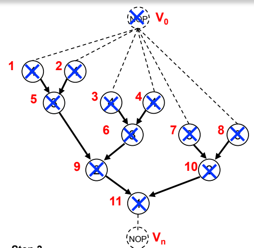
Step 5
Schedule vertices in S to time step l — Set vertices in S to start at 6

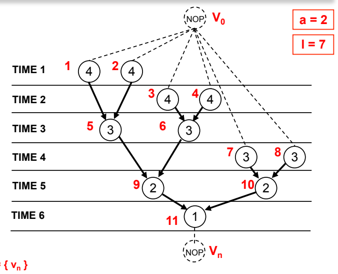
Step 6
l = l + 1 — l = 6 + 1 = 7

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

Hu's Algorithm

Example 3





Step 3
U = unscheduled vertices in V without predecessors or whose predecessors have been scheduled

U = { v_n }

Step 4
S = subset set of vertices in U, no more than a, where labels are maximal

S = { v_n }

Step 5
Schedule vertices in S to time step l

Set vertices in S to start at 7

Step 6
l = l + 1

l = 7 + 1 = 8

Step 7
Has v_n been scheduled yet?

Yes. We are done.

Additional Scheduling Considerations

- Hu's algorithm
  - Assumes one resource handles all possible operations
  - Assumes all operations have 1 unit delay
- Most scheduling problems have additional considerations
  - What happens when we have more than one type of task/operation?
  - What happens when a task/operation takes more than 1 unit delay?
- Increased problem space, difficult problem to solve efficiently
  - Many heuristics have been developed to address these problems
    - Minimum-latency, resource-constrained scheduling
    - Minimum-resource, latency-constrained scheduling

We consider one such heuristic from a family of heuristics called *list scheduling* that looks at the minimum-latency, resource-constrained scheduling problem

List Scheduling (LIST_L)

- Extension of Hu's algorithm to handle multiple operation types and multiple-cycle execution delays
- Considers minimum-latency, resource-constrained scheduling problem

LIST_L($G_S(V,E)$, a){ ← Vector a indicates the number of each type of resource available

l = 1; ← indicates the time step

repeat {

for each resource type k = 1, 2, …, n_{res} {

Determine candidate operations $U_{l,k}$; ← Operations of type k whose predecessors are completed by time l

Determine unfinished operations $T_{l,k}$; ← Unfinished operations that are already scheduled but have not completed yet

Select $S_k \subseteq U_{l,k}$ vertices, such that $|S_k| + |T_{l,k}| <= a_k$; ← Select a subset S so that the number of new operations and unfinished operations are <= to number of resources of that type

Schedule the S_k operations at step l by setting $t_i = l \; \forall \; i : v_i \in S$; ← Schedule operations in S to run at time step l

}

l = l + 1; ← update l to next time step

} until (v_n is scheduled); ← Keep going until we have scheduled the sink node v_n

return t;

}

List Scheduling (LIST_L)

LIST_L($G_S(V,E)$, a){

l = 1;

repeat {

for each resource type k = 1, 2, …, n_{res} {

Determine candidate operations $U_{l,k}$;

Determine unfinished operations $T_{l,k}$;

Select $S_k \subseteq U_{l,k}$ vertices, such that $|S_k| + |T_{l,k}| <= a_k$; ← Select a subset S so that the number of new operations and unfinished operations are <= to number of resources of that type

Schedule the S_k operations at step l by setting $t_i = l \; \forall \; i : v_i \in S$;

}

l = l + 1;

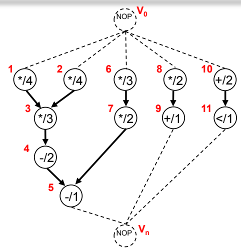
} until (v_n is scheduled);

return t;

}

- Selection of which operations to include is based on a priority list indicating some sort of urgency measure
  - We will utilize same method of labeling vertices with weights indicating path to sink, choose operations with highest weights

LIST_L Scheduling
Example 1



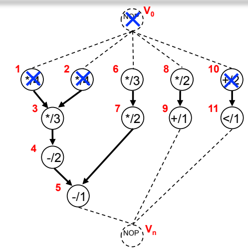
Assume all operations take 1 cycle

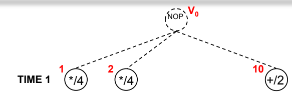
a_1 = 2 multipliers
a_2 = 2 ALUs

l = 1

Step1
l = 1

LIST_L Scheduling
Example 1





Assume all operations take 1 cycle

a_1 = 2 multipliers
a_2 = 2 ALUs

l = 1

Step 2/3
$U_{l,k}$ = candidate operations with predecessors finished at l
$T_{l,k}$ = unfinished operations

Multipliers: U = { v_1, v_2, v_6, v_8 }, T = { }
ALUs: U = { v_{10} }, T = { }

Step 4
S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

Multipliers: S = { v_1, v_2 }
ALUs: S = { v_{10} }

Step 5
Schedule vertices in S to time step l

Multipliers: Set vertices in S to start at 1
ALUs: Set vertices in S to start at 1

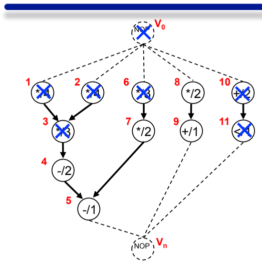
Step 6
l = l + 1

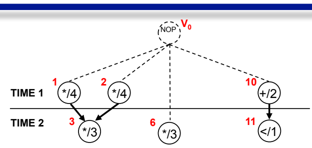
l = 1 + 1 = 2

Step 7
Has v_n been scheduled yet?

No. Repeat loop.

LIST_L Scheduling
Example 1





Assume all operations take 1 cycle

a_1 = 2 multipliers
a_2 = 2 ALUs

l = 2

Step 2/3
$U_{l,k}$ = candidate operations with predecessors finished at l
$T_{l,k}$ = unfinished operations

Multipliers: U = { v_3, v_6, v_8 }, T = { }
ALUs: U = { v_{11} }, T = { }

Step 4
S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

Multipliers: S = { v_3, v_6 }
ALUs: S = { v_{11} }

Step 5
Schedule vertices in S to time step l

Multipliers: Set vertices in S to start at 2
ALUs: Set vertices in S to start at 2

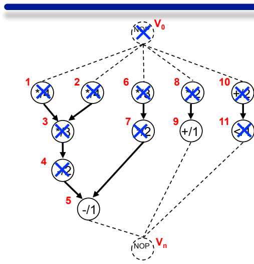
Step 6
l = l + 1

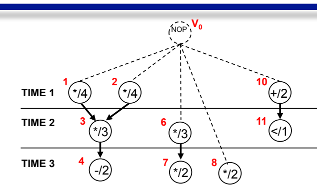
l = 2 + 1 = 3

Step 7
Has v_n been scheduled yet?

No. Repeat loop.

LIST_L Scheduling
Example 1





Assume all operations take 1 cycle

a_1 = 2 multipliers
a_2 = 2 ALUs

l = 3

Step 2/3
$U_{l,k}$ = candidate operations with predecessors finished at l
$T_{l,k}$ = unfinished operations

Multipliers: U = { v_7, v_8 }, T = { }
ALUs: U = { v_4 }, T = { }

Step 4
S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

Multipliers: S = { v_7, v_8 }
ALUs: S = { v_4 }

Step 5
Schedule vertices in S to time step l

Multipliers: Set vertices in S to start at 3
ALUs: Set vertices in S to start at 3

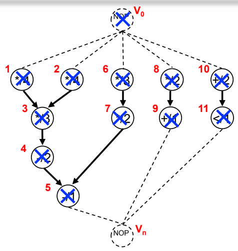
Step 6
l = l + 1

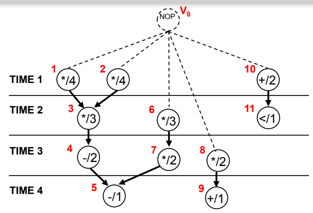
l = 3 + 1 = 4

Step 7
Has v_n been scheduled yet?

No. Repeat loop.

LIST_L Scheduling
Example 1





Assume all operations take 1 cycle

a_1 = 2 multipliers
a_2 = 2 ALUs

I = 4

Step 2/3
$U_{l,k}$ = candidate operations with predecessors finished at l
$T_{l,k}$ = unfinished operations

Step 4
S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

Step 5
Schedule vertices in S to time step l

Step 6
l = l + 1

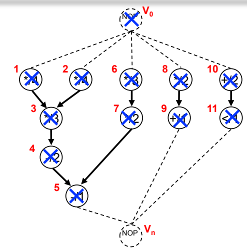
Step 7
Has v_n been scheduled yet?

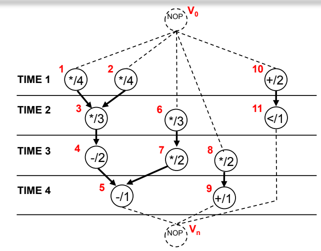
Multipliers	ALUs
U = { }	U = { v_5, v_9 }
T = { }	T = { }
S = { }	S = { v_5, v_9 }
	Set vertices in S to start at 4

l = 4 + 1 = 5

No. Repeat loop.

LIST_L Scheduling
Example 1





Assume all operations take 1 cycle

a_1 = 2 multipliers
a_2 = 2 ALUs

I = 5

Step 2/3
$U_{l,k}$ = candidate operations with predecessors finished at l
$T_{l,k}$ = unfinished operations

Step 4
S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

Step 5
Schedule vertices in S to time step l

Step 6
l = l + 1

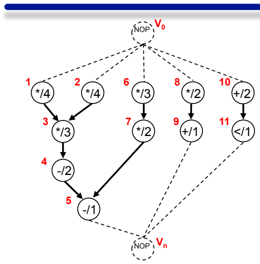
Step 7
Has v_n been scheduled yet?

Multipliers	ALUs	
U = { }	U = { }	U = { V_n }
T = { }	T = { }	T = { }
S = { }	S = { }	S = { V_n }
		Set vertices in S to start at 5

l = 5 + 1 = 6

Yes. We are done.

LIST_L Scheduling
Example 2



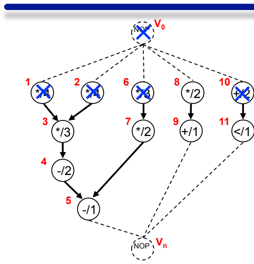
Mult. = 2 cycles
ALU = 1 cycle

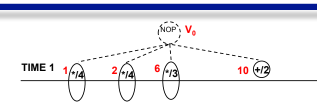
A_1 = 3 multipliers
A_2 = 1 ALU

I = 1

Step1
l = 1

LIST_L Scheduling
Example 2





Mult. = 2 cycles
ALU = 1 cycle

A_1 = 3 multipliers
A_2 = 1 ALU

I = 1

Step 2/3
$U_{l,k}$ = candidate operations with predecessors finished at l
$T_{l,k}$ = unfinished operations

Step 4
S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal

Step 5
Schedule vertices in S to time step l

Step 6
l = l + 1

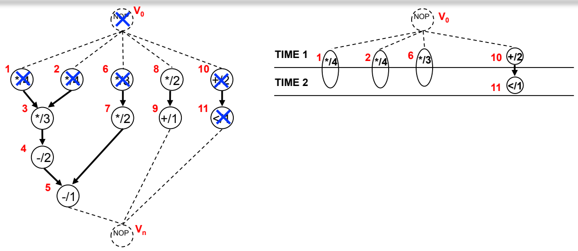
Step 7
Has v_n been scheduled yet?

Multipliers	ALUs
U = { v_1, v_2, v_6, v_8 }	U = { v_{10} }
T = { }	T = { }
S = { v_1, v_2, v_6 }	S = { v_{10} }
Set vertices in S to start at 1	Set vertices in S to start at 1

l = 1 + 1 = 2

No. Repeat loop.

LIST_L Scheduling
Example 2



Mult. = 2 cycles
ALU = 1 cycle

A_1 = 3 multipliers
A_2 = 1 ALU

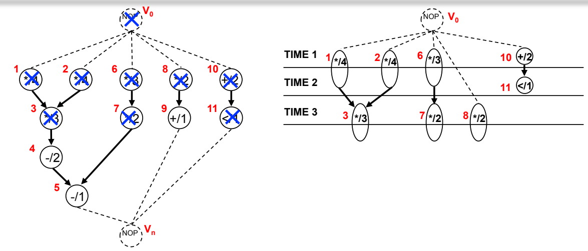
I = 2

	Multipliers	ALUs
Step 2/3 $U_{l,k}$ = candidate operations with predecessors finished at I $T_{l,k}$ = unfinished operations	U = { v_8 } T = {v_1, v_2, v_6 }	U = { v_{11} } T = { }
Step 4 S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal	S = { v_1, v_2, v_6 }	S = { v_{11} }
Step 5 Schedule vertices in S to time step I	Set vertices in S to start at 2	Set vertices in S to start at 2

Step 6
I = I + 1 — I = 2 + 1 = 3

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

LIST_L Scheduling
Example 2



Mult. = 2 cycles
ALU = 1 cycle

A_1 = 3 multipliers
A_2 = 1 ALU

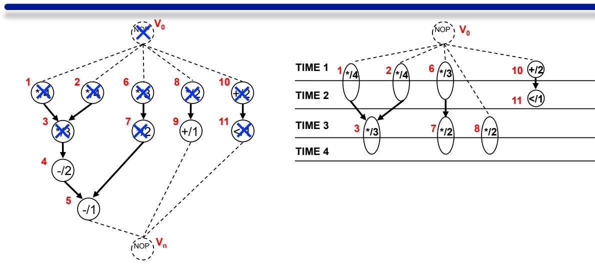
I = 3

	Multipliers	ALUs
Step 2/3 $U_{l,k}$ = candidate operations with predecessors finished at I $T_{l,k}$ = unfinished operations	U = { v_3, v_7, v_8 } T = { }	U = { } T = { }
Step 4 S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal	S = { v_3, v_7, v_8 }	S = { }
Step 5 Schedule vertices in S to time step I	Set vertices in S to start at 3	

Step 6
I = I + 1 — I = 3 + 1 = 4

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

LIST_L Scheduling
Example 2



Mult. = 2 cycles
ALU = 1 cycle

A_1 = 3 multipliers
A_2 = 1 ALU

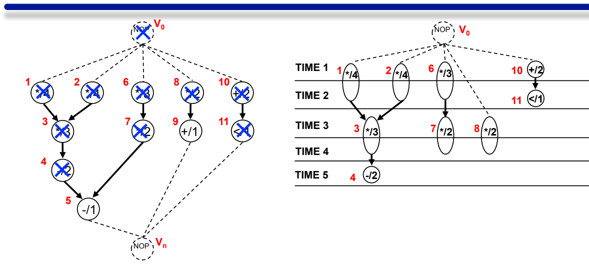
I = 4

	Multipliers	ALUs
Step 2/3 $U_{l,k}$ = candidate operations with predecessors finished at I $T_{l,k}$ = unfinished operations	U = { } T = { v_3, v_7, v_8 }	U = { } T = { }
Step 4 S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal	S = {v_3, v_7, v_8}	S = { }
Step 5 Schedule vertices in S to time step I	Set vertices in S to start at 4	

Step 6
I = I + 1 — I = 4 + 1 = 5

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

LIST_L Scheduling
Example 2



Mult. = 2 cycles
ALU = 1 cycle

A_1 = 3 multipliers
A_2 = 1 ALU

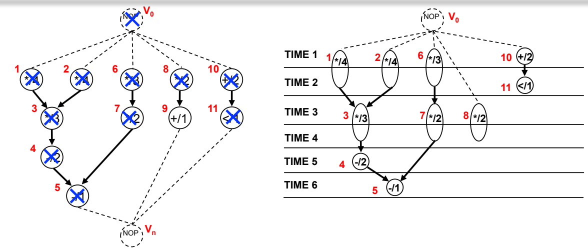
I = 5

	Multipliers	ALUs
Step 2/3 $U_{l,k}$ = candidate operations with predecessors finished at I $T_{l,k}$ = unfinished operations	U = { } T = { }	U = { v_4, v_9 } T = { }
Step 4 S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal	S = { }	S = { v_4 }
Step 5 Schedule vertices in S to time step I		Set vertices in S to start at 5

Step 6
I = I + 1 — I = 5 + 1 = 6

Step 7
Has v_n been scheduled yet? — No. Repeat loop.

LIST_L Scheduling
Example 2



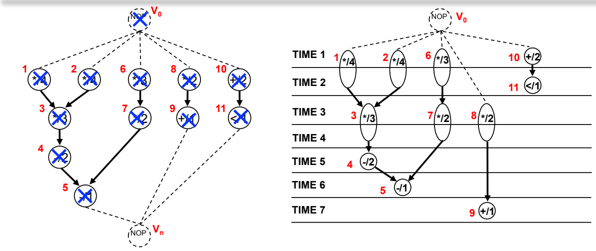
Mult. = 2 cycles
ALU = 1 cycle

A_1 = 3 multipliers
A_2 = 1 ALU

l = 6

	Multipliers	ALUs
Step 2/3 $U_{l,k}$ = candidate operations with predecessors finished at l $T_{l,k}$ = unfinished operations	U = { } T = { }	U = { v_5, v_9 } T = { }
Step 4 S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal	S = { }	S = { v_5 }
Step 5 Schedule vertices in S to time step l		Set vertices in S to start at 6
Step 6 l = l + 1	l = 6 + 1 = 7	
Step 7 Has v_n been scheduled yet?	No. Repeat loop.	

LIST_L Scheduling
Example 2



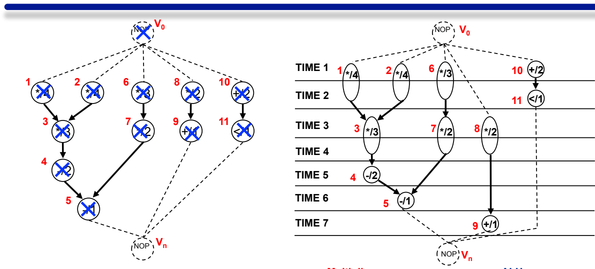
Mult. = 2 cycles
ALU = 1 cycle

A_1 = 3 multipliers
A_2 = 1 ALU

l = 7

	Multipliers	ALUs
Step 2/3 $U_{l,k}$ = candidate operations with predecessors finished at l $T_{l,k}$ = unfinished operations	U = { } T = { }	U = { v_9 } T = { }
Step 4 S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal	S = { }	S = { v_9 }
Step 5 Schedule vertices in S to time step l		Set vertices in S to start at 7
Step 6 l = l + 1	l = 7 + 1 = 8	
Step 7 Has v_n been scheduled yet?	No. Repeat loop.	

LIST_L Scheduling
Example 2



Mult. = 2 cycles
ALU = 1 cycle

A_1 = 3 multipliers
A_2 = 1 ALU

l = 8

	Multipliers	ALUs	
Step 2/3 $U_{l,k}$ = candidate operations with predecessors finished at l $T_{l,k}$ = unfinished operations	U = { } T = { }	U = { } T = { }	U = { V_n } T = { }
Step 4 S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal	S = { }	S = { }	S = { V_n }
Step 5 Schedule vertices in S to time step l			Set vertices in S to start at 8
Step 6 l = l + 1	l = 8 + 1 = 9		
Step 7 Has v_n been scheduled yet?	Yes. We are done.		

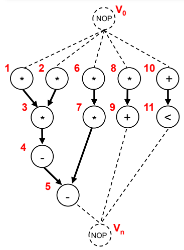
List Scheduling (LIST_R)

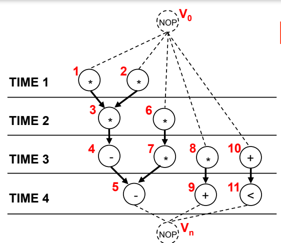
- Considers minimum-resource, latency-constrained scheduling problem

```
LIST_R( G_S(V,E), λ̄ ){
   a = 1;

   Compute the latest possible start times t^L by ALAP( G(V, E), λ̄);
   if( t_0^L < 0)
      return (Φ);

   l = 1;

   repeat {
      for each resource type k = 1, 2, …, n_res {
         Determine candidate operations U_lk;
         Compute the slacks {s_i = t_i^L - l  ∀v_i є U_lk};
         Schedule the candidate operations with zero slack and update a;
         Schedule the candidate operations requiring no additional resources;
      }
      l = l + 1;
   } until (v_n is scheduled);
   return (t, a);
}
```

- `a = 1;` — Vector a indicates the number of each type of resource available
- `if( t_0^L < 0) return (Φ);` — Algorithm exits if ALAP detects no feasible solution with dedicated resources
- `l = 1;` — Time step
- `Determine candidate operations U_lk;` — Operations of type k whose predecessors are completed by time l
- `Compute the slacks …;` — Compute slack of all candidates (current time – ALAP time)
- `Schedule the candidate operations with zero slack and update a;` — Scheduled any operation with 0 slack to meet timing requirement, add resources if needed
- `Schedule the candidate operations requiring no additional resources;` — Fill in unused resources by scheduling any available operation
- `} until (v_n is scheduled);` — Keep going until we have scheduled the sink node v_n

LIST_R Scheduling
Example 1





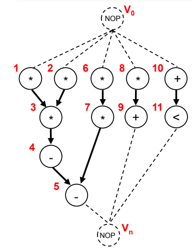
I = 1

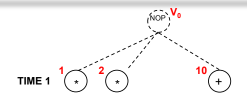
a_1 = 1 multiplier
a_2 = 1 ALU

Node	Time
1	1
2	1
3	2
4	3
5	4
6	2
7	3
8	3
9	4
10	3
11	4

- Assume all operations have unit delay, latency of 4 is required
  - Initialize vector a so all entries have value of 1
  - Compute the latest start times of all vectors by using ALAP()
  - Set time step equal to 1

LIST_R Scheduling
Example 1





I = ~~1~~ 2

a_1 = ~~1~~ 2 multiplier
a_2 = 1 ALU

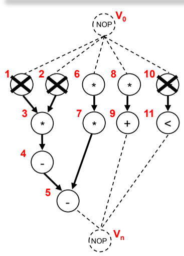
Node	Time
1	1
2	1
3	2
4	3
5	4
6	2
7	3
8	3
9	4
10	3
11	4

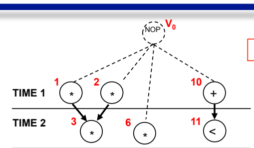
	Multipliers	ALUs
Determine candidate operations	U = { v_1, v_2, v_6, v_8 }	U = { v_{10} }
Compute the slacks	v_1 = 1-1 = 0 v_2 = 1-1 = 0 v_6 = 2-1 =1 v_8 = 3-1 = 2	v_{10} = 3-1 = 2
Schedule candidate operations with zero slack and update a	S = { v_1, v_2 }, a_1 = 2	no zero slack operations
Schedule candidate operations requiring no additional resources	no spare multipliers	S = { v_{10} }

Increment time step — **I = 1 + 1 = 2**

Has v_n been scheduled yet? — **No. Repeat loop.**

LIST_R Scheduling
Example 1





I = ~~2~~ 3

a_1 = 2 multiplier
a_2 = 1 ALU

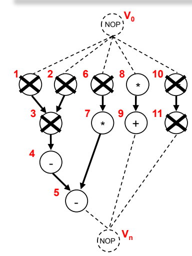
Node	Time
1	1
2	1
3	2
4	3
5	4
6	2
7	3
8	3
9	4
10	3
11	4

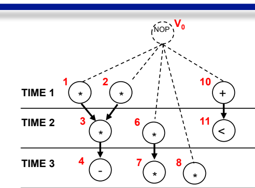
	Multipliers	ALUs
Determine candidate operations	U = { v_3, v_6, v_8 }	U = { v_{11} }
Compute the slacks	v_3 = 2-2 = 0 v_6 = 2-2 =0 v_8 = 3-2 = 1	v_{11} = 4-2 = 2
Schedule candidate operations with zero slack and update a	S = { v_3, v_6 }	no zero slack operations
Schedule candidate operations requiring no additional resources	no spare multipliers	S = { v_{11} }

Increment time step — **I = 2 + 1 = 3**

Has v_n been scheduled yet? — **No. Repeat loop.**

LIST_R Scheduling
Example 1





I = ~~3~~ 4

a_1 = 2 multiplier
a_2 = 1 ALU

Node	Time
1	1
2	1
3	2
4	3
5	4
6	2
7	3
8	3
9	4
10	3
11	4

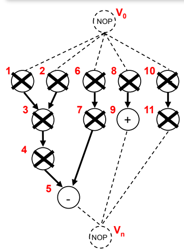
	Multipliers	ALUs
Determine candidate operations	U = { v_7, v_8 }	U = { v_4 }
Compute the slacks	v_7 = 3-3 = 0 v_8 = 3-3 =0	v_4 = 3-3 = 0
Schedule candidate operations with zero slack and update a	S = { v_7, v_8 }	S = { v_4 }
Schedule candidate operations requiring no additional resources	no spare multipliers	no spare ALUs

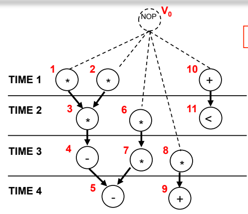
Increment time step — **I = 1 + 1 = 4**

Has v_n been scheduled yet? — **No. Repeat loop.**

LIST_R Scheduling

Example 1





5
I = ~~4~~

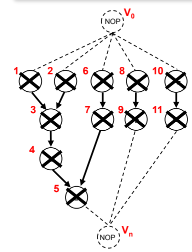
a_1 = 2 multiplier
a_2 = ~~1~~ ALU
2

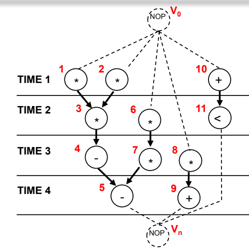
Node	Time
1	1
2	1
3	2
4	3
5	4
6	2
7	3
8	3
9	4
10	3
11	4

	Multipliers	ALUs
Determine candidate operations	U = { Φ }	U = { v_5, v_9}
Compute the slacks		v_5 = 4-4 = 0 v_9 = 4-4 = 0
Schedule candidate operations with zero slack and update a	S = { Φ }	S = {v_5, v_9}; a = 2
Schedule candidate operations requiring no additional resources	no multiplier operations	no spare ALUs
Increment time step	I = 4 + 1 = 5	
Has v_n been scheduled yet?	No. Repeat loop.	

LIST_R Scheduling

Example 1





I = 5

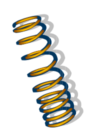
a_1 = 2 multiplier
a_2 = 2 ALU

Node	Time
1	1
2	1
3	2
4	3
5	4
6	2
7	3
8	3
9	4
10	3
11	4

	Multipliers	ALUs	
Determine candidate operations	U = { Φ }	U = { Φ}	U = { V_n}
Compute the slacks			
Schedule candidate operations with zero slack and update a			
Schedule candidate operations requiring no additional resources			
Increment time step			
Has v_n been scheduled yet?	Yes. Done		

Force-Directed Scheduling (FDS)

- Heuristic scheduling algorithms
  - Consider the unscheduled CDFG under a physics-based spring model
  - Operators are subjected to physical 'forces', both repelling and attracting them to particular time slices
    - Larger the force, the larger the concurrency
  - Goal is to find the optimal placement of vertices into a schedule, when subject to these 'forces'



- Minimum latency under resource-constraint
  - Force directed list scheduling
  - Extension of list scheduling algorithms
- Minimum resource under latency-constraint
  - Force directed scheduling

This is the one we will consider

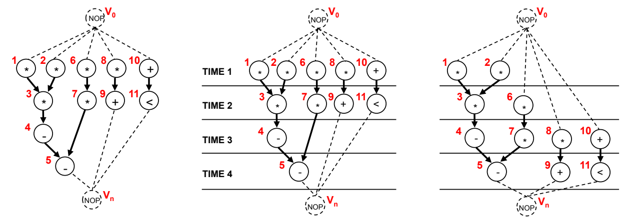
Force-Directed Scheduling (FDS)

- Force-Directed Scheduling
  - Minimum resource under latency constraint

```
FDS( G(V,E), λ̄ ){
   repeat {
      Compute the time frames;
      Compute the operations and type probabilities;
      Compute the self-forces, predecessor/successor forces and total forces;
      Schedule the operation with least force and update its time-frame;
   } until (all operations scheduled);
   return (t);
}
```

Force-Directed Scheduling (FDS)
Time Frames

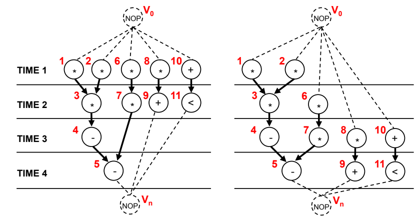
- Time frame of an operation is the time interval where it can be scheduled
  - Denoted by $\{[t_i^S, t_i^L]; i = 0, 1, \ldots, n\}$
  - Earliest and latest start times can be computed by ASAP and ALAP algorithms



- Width of time frame of an operation is equal to its mobility plus 1

Force-Directed Scheduling (FDS)
Example 2

- Time frames for various operation assuming a latency bound of 4
  - Latency bound needed for ALAP scheduling



operation v_1	operation v_2	operation v_6	operation v_8
ASAP time = 1	ASAP time = 1	ASAP time = 1	ASAP time = 1
ALAP time = 1	ALAP time = 1	ALAP time = 2	ALAP time = 3
time frame = [1, 1]	time frame = [1, 1]	time frame = [1, 2]	time frame = [1, 3]

Force-Directed Scheduling (FDS)

- Force-Directed Scheduling
  - Minimum resource under latency constraint

```
FDS( G(V,E), λ̄ ){
   repeat {
      Compute the time frames;
      Compute the operations and type probabilities;
      Compute the self-forces, predecessor/successor forces and total forces;
      Schedule the operation with least force and update its time-frame;
   } until (all operations scheduled);
   return (t);
}
```

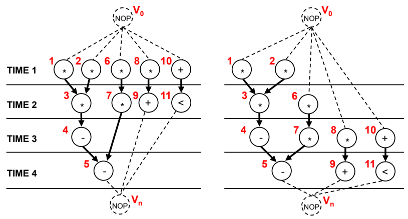

Force-Directed Scheduling (FDS)
Operation Probability

- Operation Probability is a function
  - Equal to zero outside of the corresponding time frame
  - Equal to reciprocal of the frame width inside the time frame
- Denoted the probability of the operations at time l by $\{p_i(l); i = 0, 1, \ldots, n\}$
- What is the significance?
  - Operations whose time frame is one unit wide are bound to start in one specific time
  - For remaining operations, the larger the width, the lower the probability that the operation is scheduled in any given step inside the corresponding time frame

Force-Directed Scheduling (FDS)

Example 3

- Operation Probability for various operations
  - Equal to zero outside of the corresponding time frame
  - Equal to reciprocal of the frame width inside the time frame



operation v_1	operation v_2	operation v_6	operation v_8
time frame = [1, 1]	time frame = [1, 1]	time frame = [1, 2]	time frame = [1, 3]
frame width = 1	frame width = 1	frame width = 2	frame width = 3
$p_1(1) = 1, p_1(2) = 0$	$p_2(1) = 1, p_2(2) = 0$	$p_6(1) = 0.5, p_6(2) = 0.5$	$p_8(1) = 0.3, p_8(2) = 0.3$
$p_1(3) = 0, p_1(4) = 0$	$p_2(3) = 0, p_2(4) = 0$	$p_6(3) = 0, p_6(4) = 0$	$p_8(3) = 0.3, p_8(4) = 0$

Force-Directed Scheduling (FDS)

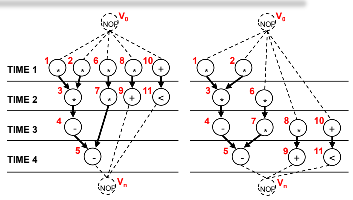
Type Distribution

- Type Distribution is the sum of probabilities of the operations implemented by a specific resource at any time step of interest
  - Denote distribution at time *l* by $\{q_k(l); k = 1, 2, \ldots, n_{res}\}$
- Distribution graph is a plot of any operation-type distribution over the scheduled steps
  - Shows likelihood that a resource is used at each scheduled step
  - Uniform plot in a distribution graph means that a type is evenly scattered in the schedule and a good measure of utilization

Force-Directed Scheduling (FDS)

Example 4

- Distribution graph for ALU
  - Sum of probabilities of the operations implemented by a specific resource at any time step of interest



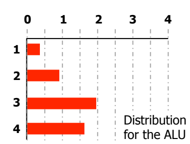
	p(1)	p(2)	p(3)	p(4)
v_4 = [3, 3], width = 1	0	0	1	0
v_5 = [4, 4], width = 1	0	0	0	1
v_9 = [2, 4], width = 3	0	0.3	0.3	0.3
v_{10} = [1, 3], width = 3	0.3	0.3	0.3	0
v_{11} = [2, 4], width = 3	0	0.3	0.3	0.3

$q_2(1) = 0 + 0 + 0 + 0.3 + 0 = 0.3$

$q_2(2) = 0 + 0 + 0.3 + 0.3 + 0.3 = 0.9$

$q_2(3) = 1 + 0 + 0.3 + 0.3 + 0.3 = 1.9$

$q_2(4) = 0 + 1 + 0.3 + 0 + 0.3 = 1.6$

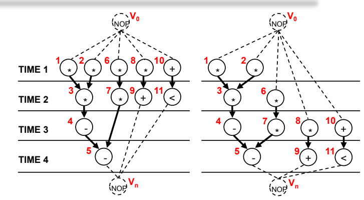


Distribution graph for the ALU

Force-Directed Scheduling (FDS)

Example 5

- Distribution graph for Multiplier
  - Sum of probabilities of the operations implemented by a specific resource at any time step of interest



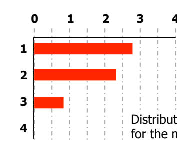
	p(1)	p(2)	p(3)	p(4)
v_1 = [1, 1], width = 1	1	0	0	0
v_2 = [1, 1], width = 1	1	0	0	0
v_3 = [2, 2], width = 1	0	1	0	0
v_6 = [1, 2], width = 2	0.5	0.5	0	0
v_7 = [2, 3], width = 2	0	0.5	0.5	0
v_8 = [1, 3], width = 3	0.3	0.3	0.3	0

$q_2(1) = 1 + 1 + 0 + 0.5 + 0 + 0.3 = 2.8$

$q_2(2) = 0 + 0 + 1 + 0.5 + 0.5 + 0.3 = 2.3$

$q_2(3) = 0 + 0 + 0 + 0 + 0.5 + 0.3 = 0.8$

$q_2(4) = 0 + 0 + 0 + 0 + 0 + 0 = 0$



Distribution graph for the multiplier

Force-Directed Scheduling (FDS)

- Force-Directed Scheduling
  - Minimum resource under latency constraint

```
FDS( G(V,E), λ̄ ){
   repeat {
      Compute the time frames;
      Compute the operations and type probabilities;
      Compute the self-forces, predecessor/successor forces and total forces;
      Schedule the operation with least force and update its time-frame;
   } until (all operations scheduled);
   return (t);
}
```


Force-Directed Scheduling (FDS)
Self Force

- Self Force
  - Scheduling an operation will effect overall concurrency
  - Every operation has "self force" for every C-step of its time frame
  - Desirable scheduling will have negative self force

Force(i) = DG(i) * x(i)

DG(i) = Current Distribution Graph value
x(i) = Change in operation's probability

$\text{Self Force(j)} = \sum_{i=t}^{b} \text{Force(i)}$

Force-Directed Scheduling (FDS)
Example 6

- Calculate Self Force for v_6
  - Assignment of v6 to time step 1
  - Assignment of v6 to time step 2

Force(i) = DG(i) * x(i)

DG(i) = Current Distribution Graph value
x(i) = Change in operation's probability

$\text{Self Force(j)} = \sum_{i=t}^{b} \text{Force(i)}$

<u>Assuming v6 assigned to time step 1</u>

Self force = <u>2.8</u>(<u>1</u>-<u>0.5</u>) + <u>2.3</u>(<u>0</u>-<u>0.5</u>)

Distribution graph values to time step 1 and 2

1 indicates that v6 schedule in time 1, minus the operator probability in time 1

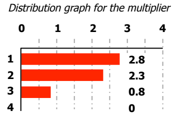
0 indicates that v6 is NOT scheduled in time 1, minus the operator probability in time 2

Time frame and operation probability for v_6

v_6 = [1, 2], width = 2

p(1)=0.5, p(2)=0.5, p(3)=0, p(4)=0

Distribution graph for the multiplier



Force-Directed Scheduling (FDS)
Example 6

- Calculate Self Force for v_6
  - Assignment of v6 to time step 1
  - Assignment of v6 to time step 2

Force(i) = DG(i) * x(i)

DG(i) = Current Distribution Graph value
x(i) = Change in operation's probability

$\text{Self Force(j)} = \sum_{i=t}^{b} \text{Force(i)}$

<u>Assuming v6 assigned to time step 1</u>

Self force = 2.8(1-0.5) + 2.3(0-0.5)
= 0.25

<u>Assuming v6 assigned to time step 2</u>

Self force = 2.8(0-0.5) + 2.3(1-0.5)
= -0.25

Want to reduce force (concurrency), time step 2 looks better

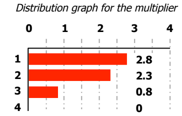
How does this impact other operations?

Time frame and operation probability for v_6

v_6 = [1, 2], width = 2

p(1)=0.5, p(2)=0.5, p(3)=0, p(4)=0

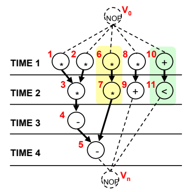
Distribution graph for the multiplier



Force-Directed Scheduling (FDS)

Predecessor/Successor Forces

- Predecessor/Successor Force
  - Scheduling an operation may affect the time frames of other linked operations
  - This may negate the benefits of the desired assignment
  - Predecessor/Successor Forces = Sum of Self Forces of any implicitly scheduled operations



If v_6 scheduled in time 2, then v_7 has to be scheduled in time 3

If v_{11} scheduled in time 3, then v_{10} has to be scheduled in time 1 or 2

Force-Directed Scheduling (FDS)

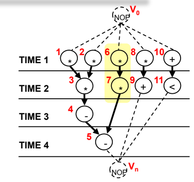
Example 7

- Calculate Predecessor/Successor Force for v_6
  - Assign of v6 to time step 1
  - Assign of v6 to time step 2

Force(i) = DG(i) * x(i)

DG(i) = Curr Distrb Graph value
x(i) = Change in op prob

Self Force(j) = $\sum_{i=t}^{b}$ **Force(i)**



Assuming v6 assigned to time step 1

no predecessor effected

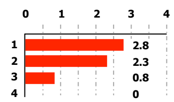
Predecessor force = 0

no successor effected
v_7 can be scheduled at time 2 or 3

Successor force = 0

Total force = Self Force + Predecessor force + Successor force
= 0.25 + 0 + 0
= 0.25

Distribution graph for the multiplier



Time frame and operation probability for v_6 and v_7

v_6 = [1, 2], width = 2
p(1)=0.5, p(2)=0.5, p(3)=0, p(4)=0

v_7 = [2, 3], width = 2
p(1)=0, p(2)=0.5, p(3)=0.5, p(4)=0

Force-Directed Scheduling (FDS)

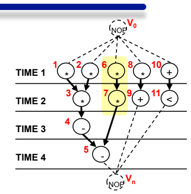
Example 7

- Calculate Predecessor/Successor Force for v_6
  - Assign of v6 to time step 1
  - Assign of v6 to time step 2

Force(i) = DG(i) * x(i)

DG(i) = Curr Distrb Graph value
x(i) = Change in op prob

Self Force(j) = $\sum_{i=t}^{b}$ **Force(i)**



Assuming v6 assigned to time step 2

no predecessor effected

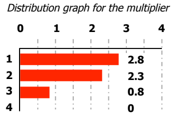
Predecessor force = 0

v_7 can only be scheduled at time 3

Successor force = sum of self forces of implicitly scheduled operations
= 2.3(0-0.5) + 0.8(1-0.5)
= -0.75

Total force = Self Force + Predecessor force + Successor force
= -0.25 + 0 + -0.75
= -1

Distribution graph for the multiplier



Time frame and operation probability for v_6 and v_7

v_6 = [1, 2], width = 2
p(1)=0.5, p(2)=0.5, p(3)=0, p(4)=0

v_7 = [2, 3], width = 2
p(1)=0, p(2)=0.5, p(3)=0.5, p(4)=0

Force-Directed Scheduling (FDS)

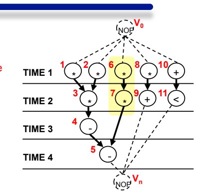
Example 7

- Calculate Predecessor/Successor Force for v_6
  - Assign of v6 to time step 1
  - Assign of v6 to time step 2

Force(i) = DG(i) * x(i)

DG(i) = Curr Distrb Graph value
x(i) = Change in op prob

Self Force(j) = $\sum_{i=t}^{b}$ **Force(i)**



Assuming v6 assigned to time step 1

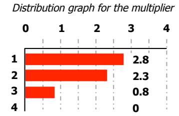
Total force = 0.25

Assuming v6 assigned to time step 2

Total force = -1

Better choice – want to reduce force in the minimum resource under latency-constraint

Distribution graph for the multiplier



Time frame and operation probability for v_6 and v_7

v_6 = [1, 2], width = 2
p(1)=0.5, p(2)=0.5, p(3)=0, p(4)=0

v_7 = [2, 3], width = 2
p(1)=0, p(2)=0.5, p(3)=0.5, p(4)=0

Force-Directed Scheduling (FDS)

- Force-Directed Scheduling
  - Minimum resource under latency constraint

```
FDS( G(V,E), λ̄ ){
   repeat {
      Compute the time frames;
      Compute the operations and type probabilities;
      Compute the self-forces, predecessor/successor forces and total forces;
      Schedule the operation with least force and update its time-frame;
   } until (all operations scheduled);
   return (t);
}
```

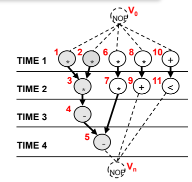
At each iteration time frame, probabilities, and forces need to be recalculated

Forces relate to concurrency – we choose lowest force so we can minimize number of resources

Results have shown FDS superior to list scheduling, but run time are long for larger graph (limited usage)

Force-Directed Scheduling (FDS)

- Previous example only looked at v6
- Algorithm tells us to calculate ALL unscheduled nodes, then schedule operation assignment with smallest force



Conclusion

- Considered several types of scheduling algorithms
  - Unconstrained Scheduling - ASAP
  - Latency-Constrained Scheduling – ALAP
  - Resource-Constrained Scheduling – Hu's Algorithm
- Practical Scheduling problems possibly include multiple-cycle operations with different types
  - Minimum-Latency, Resource-Constrained and Minimum-Resource, Latency-Constrained problems become difficult to solve efficiently
  - Heuristics developed
    - *List Scheduling (LIST_L)*
    - *List Scheduling (LIST_R)*
    - *Force-directed Scheduling*
    - *Trace Scheduling*
    - *Percolation Scheduling*