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){

$l = 1$;

repeat {

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

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

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

Select $S_k \subset U_{l,k}$ vertices, such that $|S_k| + |T_{l,k}| <= a_k$;

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

}

$l = l + 1$;

} until (v_n is scheduled);

return t;

}

Vector a indicates the number of each type of resource available

indicates the time step

Operations of type k whose predecessors are completed by time l

Unfinished operations that are already scheduled but have not completed yet

Select a subset S so that the number of new operations and unfinished operations are <= to number of resources of that type

Schedule operations in S to run at time step l

update l to next time step

Keep going until we have scheduled the sink node v_n

List Scheduling (LIST_L)

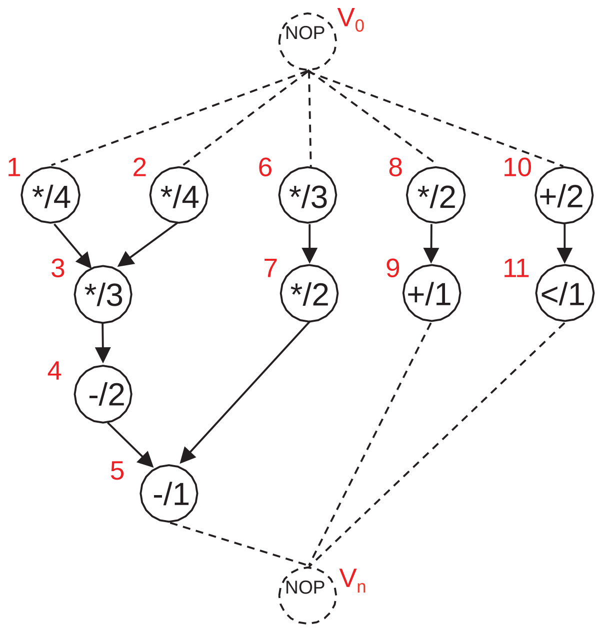

```
LIST_L( G_S(V,E), a ){
    l = 1;
    repeat {
        for each resource type k = 1, 2, . . , n_res {
            Determine candidate operations U_{l,k};
            Determine unfinished operations T_{l,k};
            Select S_k ⊆ U_{l,k} vertices, such that |S_k| + |T_{l,k}| <= a_k;
            Schedule the S_k operations at step l by setting t_i = l   i : v_i ϵ S;
        }
        l = l + 1;
    } until (v_n is scheduled);
    return t;
}
```

Select a subset S so that the number of new operations and unfinished operations are <= to number of resources of that type

- 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

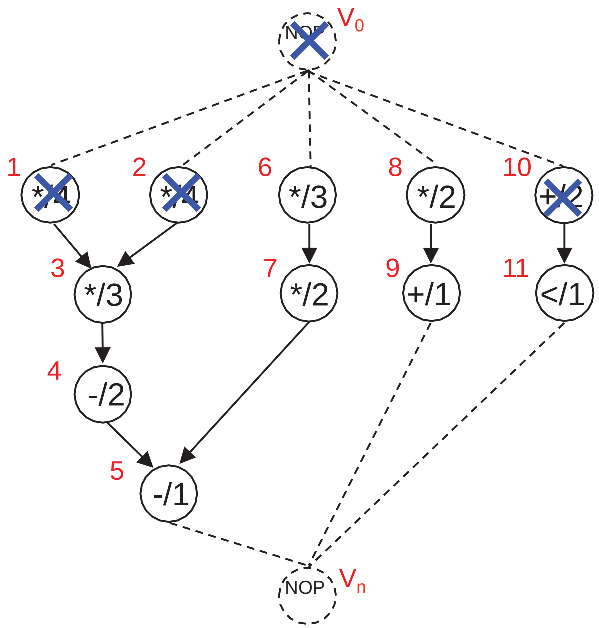
a_1 = 2 multipliers
a_2 = 2 ALUs

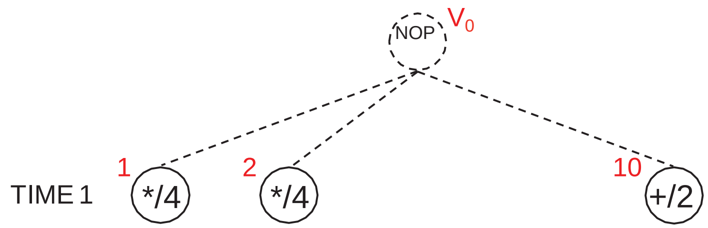
$\boxed{l = 1}$

<u>Step1</u>
l = 1

LIST_L Scheduling

Example 1





Assume all operations take 1 cycle

a_1 = 2 multipliers
a_2 = 2 ALUs

$\boxed{l = 1}$

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
U = { v_1, v_2, v_6, v_8 }
T = {}

S = { v_1, v_2 }

Set vertices in S to start at 1

ALUs
U = { v_{10} }
T = {}

S = { v_{10} }

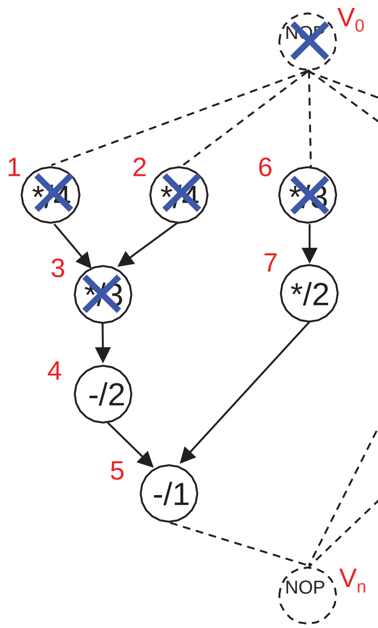
Set vertices in S to start at 1

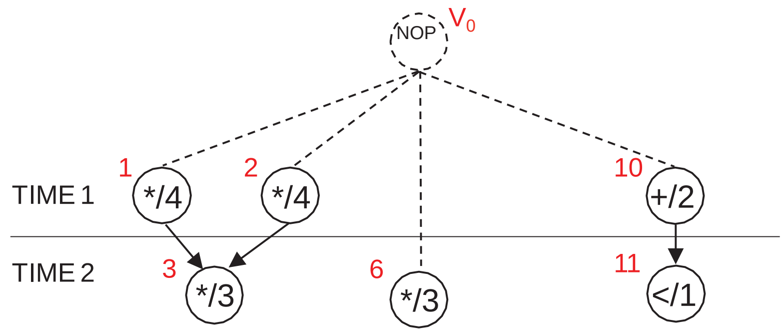
l = 1 + 1 = 2

No. Repeat loop.

LIST_L Scheduling

Example 1





Assume all operations take 1 cycle

a_1 = 2 multipliers
a_2 = 2 ALUs

$\boxed{l = 2}$

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
U = { v_3, v_6, v_8 }
T = {}

S = { v_3, v_6 }

Set vertices in S to start at 2

ALUs
U = { v_{11} }
T = {}

S = { v_{11} }

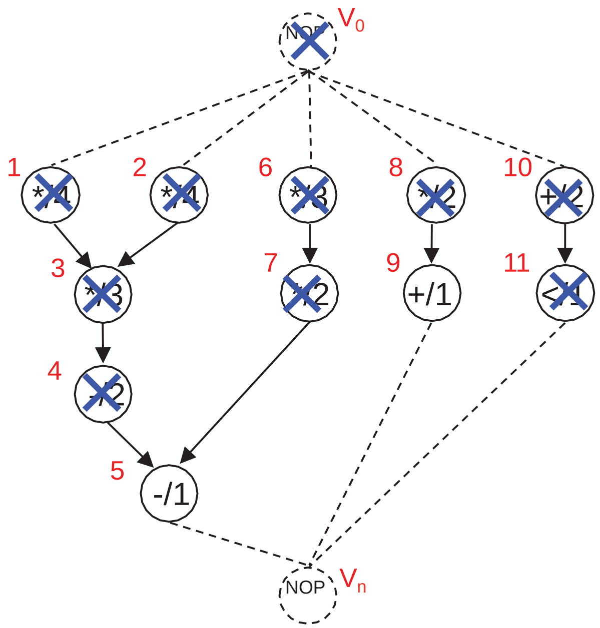
Set vertices in S to start at 2

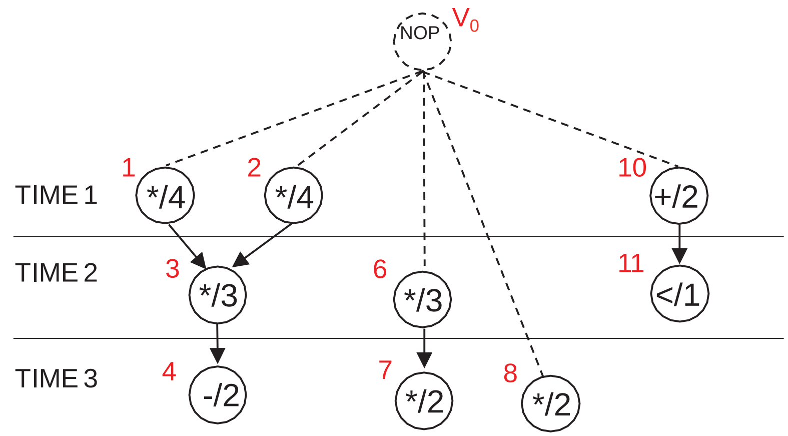
l = 2 + 1 = 3

No. Repeat loop.

LIST_L Scheduling

Example 1





Assume all operations take 1 cycle

a_1 = 2 multipliers
a_2 = 2 ALUs

$\boxed{l = 3}$

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

<u>Multipliers</u>
U = {v_7, v_8}
T = {}

<u>ALUs</u>
U = { v_4 }
T = {}

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

S = { v_7, v_8 }

S = { v_4 }

<u>Step 5</u>
Schedule vertices in S to time step l

Set vertices in S to start at 3

Set vertices in S to start at 3

<u>Step 6</u>
l = l + 1

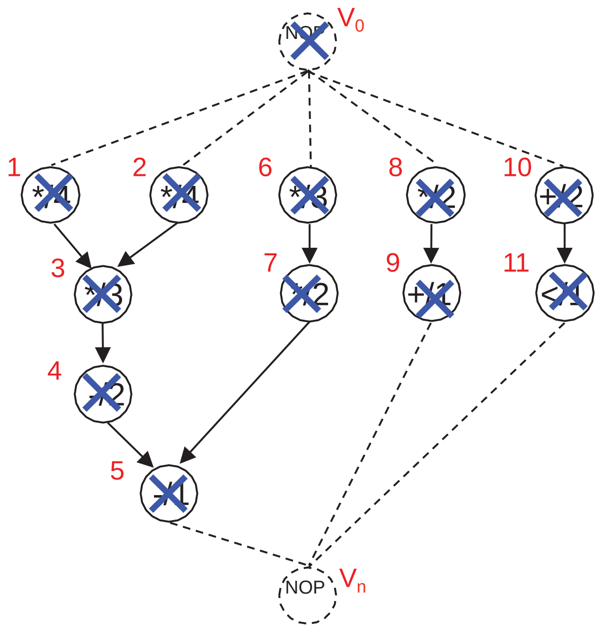
l = 3 + 1 = 4

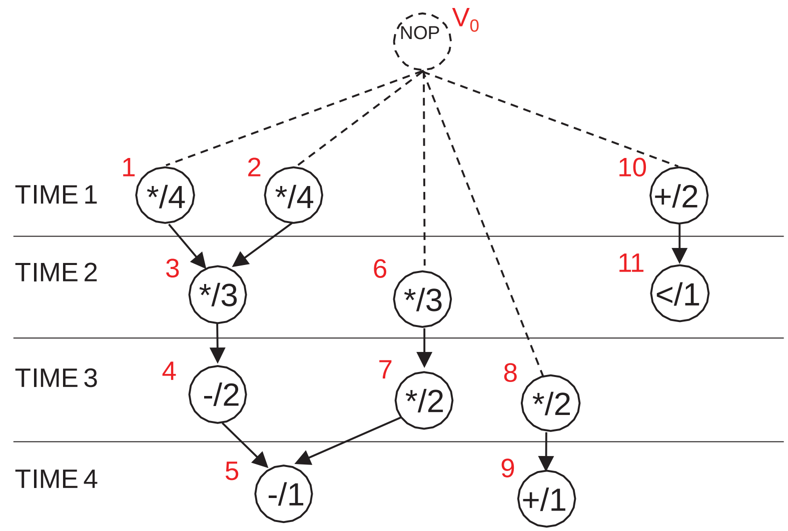
<u>Step 7</u>
Has v_n been scheduled yet?

No. Repeat loop.

LIST_L Scheduling

Example 1





Assume all operations take 1 cycle

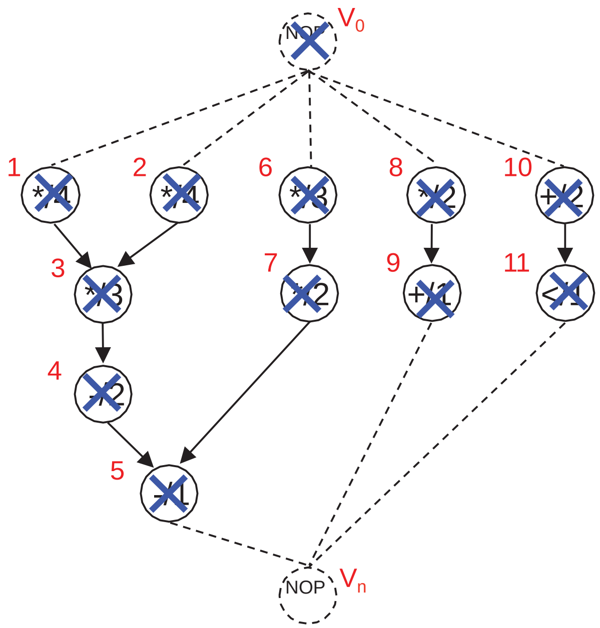
a_1 = 2 multipliers
a_2 = 2 ALUs

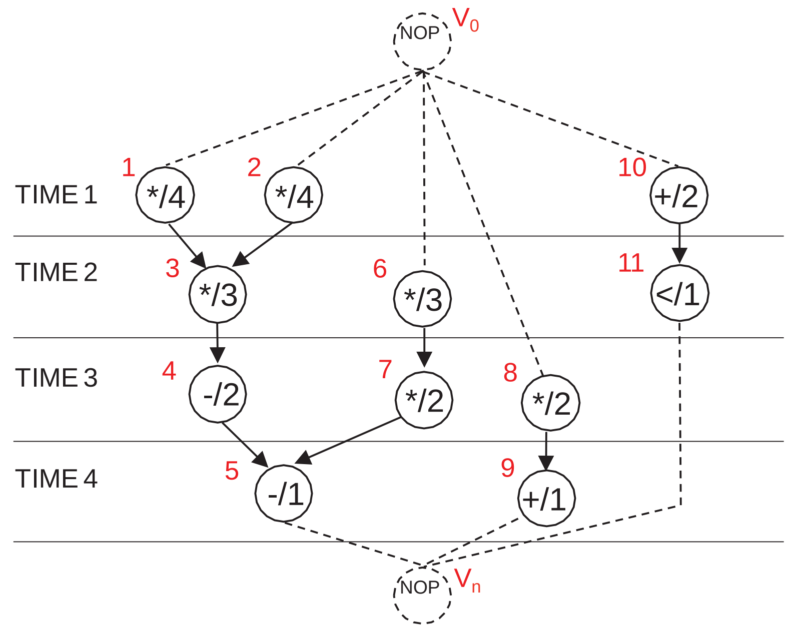
$\boxed{I = 4}$

	Multipliers	ALUs
<u>Step 2/3</u> $U_{l,k}$ = candidate operations with predecessors finished at l $T_{l,k}$ = unfinished operations	U = {} T = {}	U = { v_5, v_9 } T = {}
<u>Step 4</u> S = subset set of vertices in U and T such that U + T is <=a, where labels are maximal	S = { }	S = { v_5, v_9 }
<u>Step 5</u> Schedule vertices in S to time step l		Set vertices in S to start at 4
<u>Step 6</u> l = l + 1	l = 4 + 1 = 5	
<u>Step 7</u> 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

$\boxed{l = 5}$

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

	Multipliers	ALUs	
	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 5

Step 6
l = l + 1

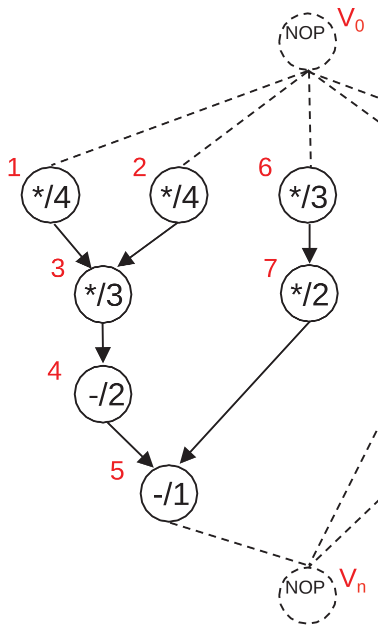
l = 5 + 1 = 6

Step 7
Has v_n been scheduled yet?

Yes. We are done.

LIST_L Scheduling

Example 2



Mult. = 2 cycles
ALU = 1 cycle
A_1 = 3 multipliers
A_2 = 1 ALU

List Scheduling (LIST_R)

- Considers minimum-resource, latency-constrained scheduling problem

```
LIST_R( G_S(V,E), λ ){
    a = 1;

    Compute the latest possible start times t^L by ALAP( G(V, E), λ);
    if( t^L_0 < 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^L_i - l   ∀v_i ϵ 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);
}
```

Vector a indicates the number of each type of resource available

Algorithm exits if ALAP detects no feasible solution with dedicated resources

Time step

Operations of type k whose predecessors are completed by time l

Compute slack of all candidates (ALAP time – current time)

Scheduled any operation with 0 slack to meet timing requirement, add resources if needed

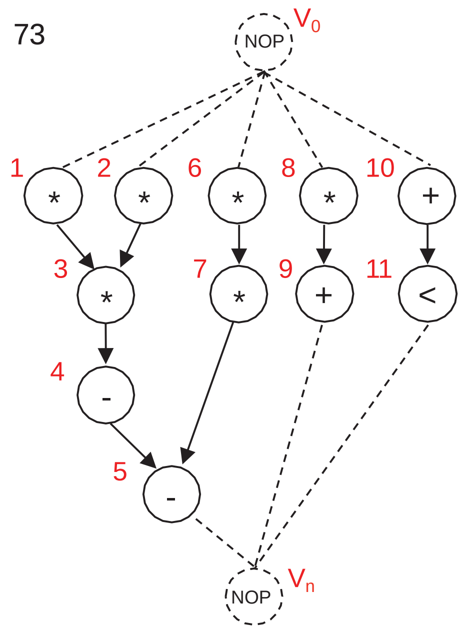
Fill in unused resources by scheduling any available operation

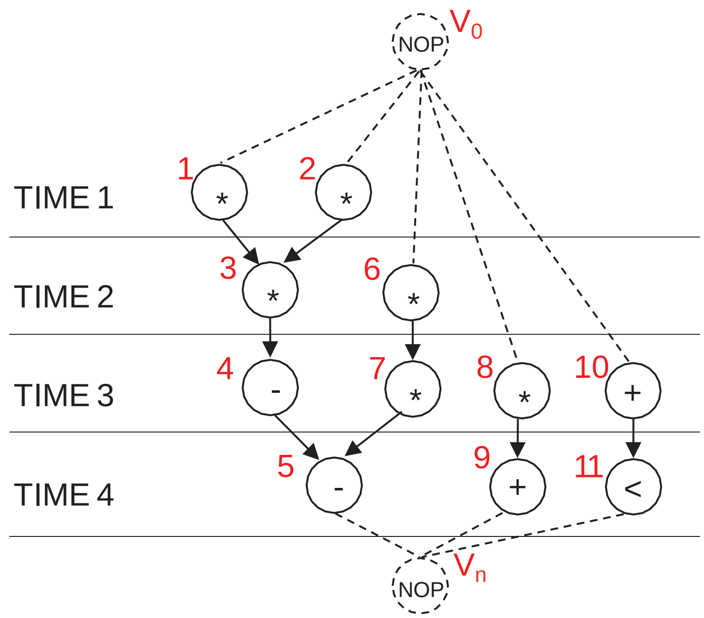
Keep going until we have scheduled the sink node v_n

LIST_R Scheduling

Example 1

73





l = 1

a_1 = 1 multiplier
a_2 = 1 ALU

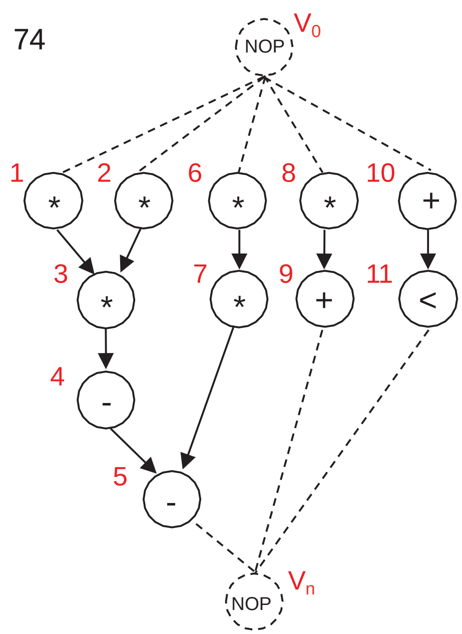
Node	Time
1	1
2	1
3	2
4	3
5	4
6	2
7	3
8	3
9	4
10	3
11	4

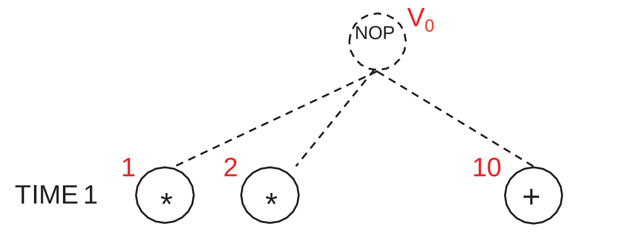
- Assume all operations have unit delay, latency of 4 is required
  - Initialize vector a so all entries have value of 1
  - Compute the latest start times of all vectors by using ALAP()
  - Set time step equal to 1

LIST_R Scheduling

Example 1

74





$\boxed{I = \cancel{1}\ 2}$

$a_1 = \cancel{1}\ 2$ 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	$v_1 = 1\text{-}1 = 0$ $v_2 = 1\text{-}1 = 0$ $v_6 = 2\text{-}1 = 1$ $v_8 = 3\text{-}1 = 2$	$v_{10} = 3\text{-}1 = 2$
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	$S = \{ v_{10} \}$

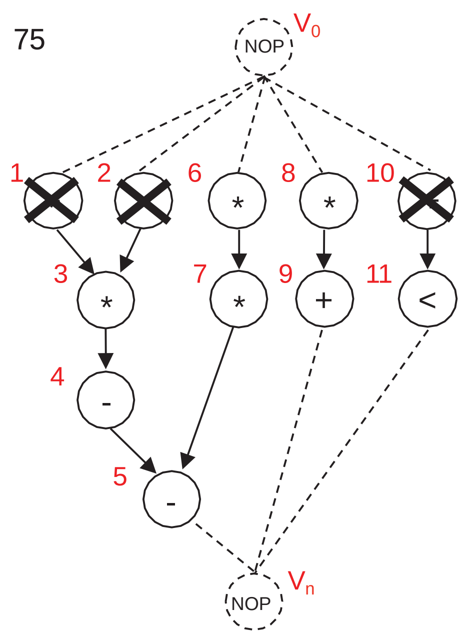
Increment time step: $I = 1 + 1 = 2$

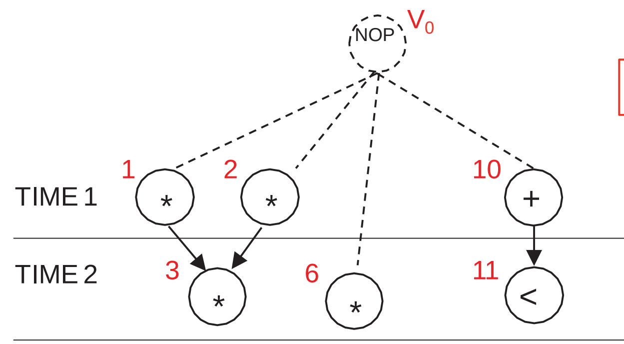
Has v_n been scheduled yet? No. Repeat loop.

LIST_R Scheduling

Example 1

75





3

$\boxed{l = 2}$

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} }

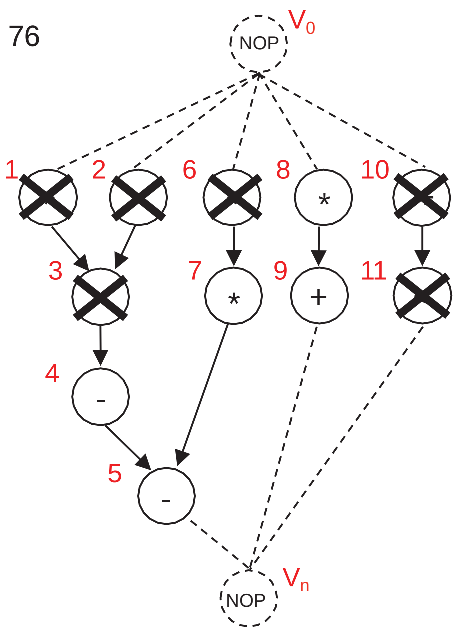
Increment time step — l = 2 + 1 = 3

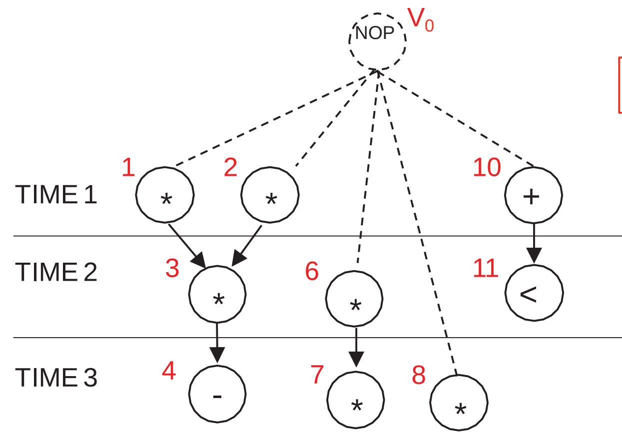
Has v_n been scheduled yet? — No. Repeat loop.

LIST_R Scheduling

Example 1

76





4

$\boxed{l = 3}$ ✕

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_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

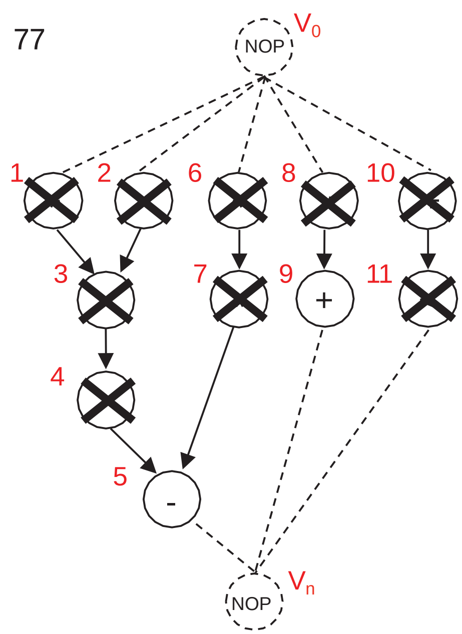
Increment time step l = 1 + 1 = 4

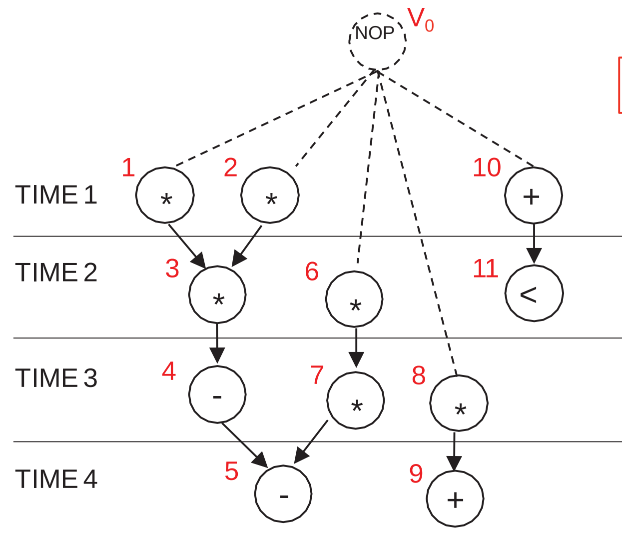
Has v_n been scheduled yet? No. Repeat loop.

LIST_R Scheduling

Example 1

77





5

$\boxed{I = 4 \times}$

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

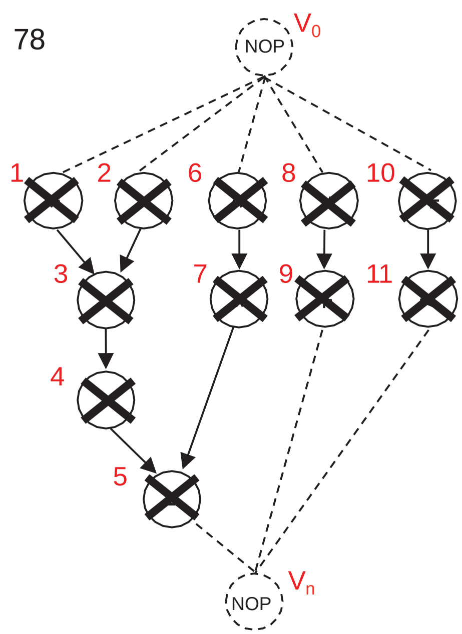
Increment time step — I = 4 + 1 = 5

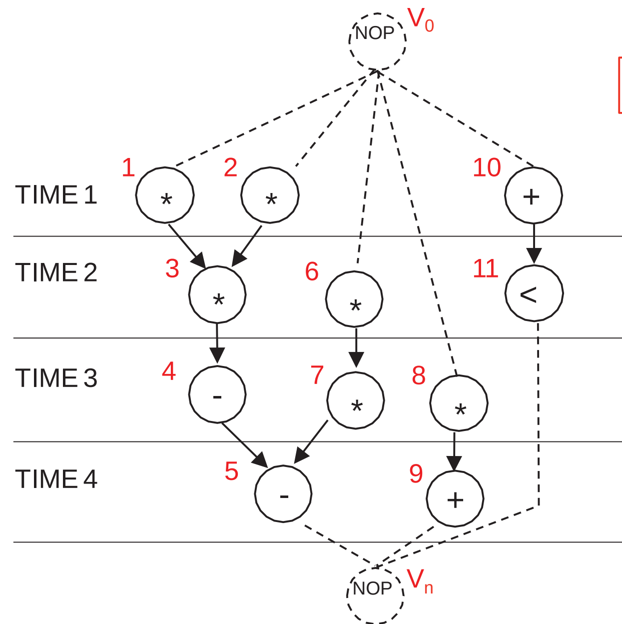
Has v_n been scheduled yet? — No. Repeat loop.

LIST_R Scheduling

Example 1

78





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

U = { Φ }

ALUs

U = { Φ}

U = { V_n}

Determine candidate operations

Compute the slacks

Schedule candidate operations with zero slack and update a

Schedule candidate operations requiring no additional resources

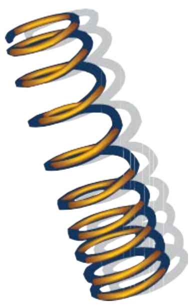
Increment time step

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)

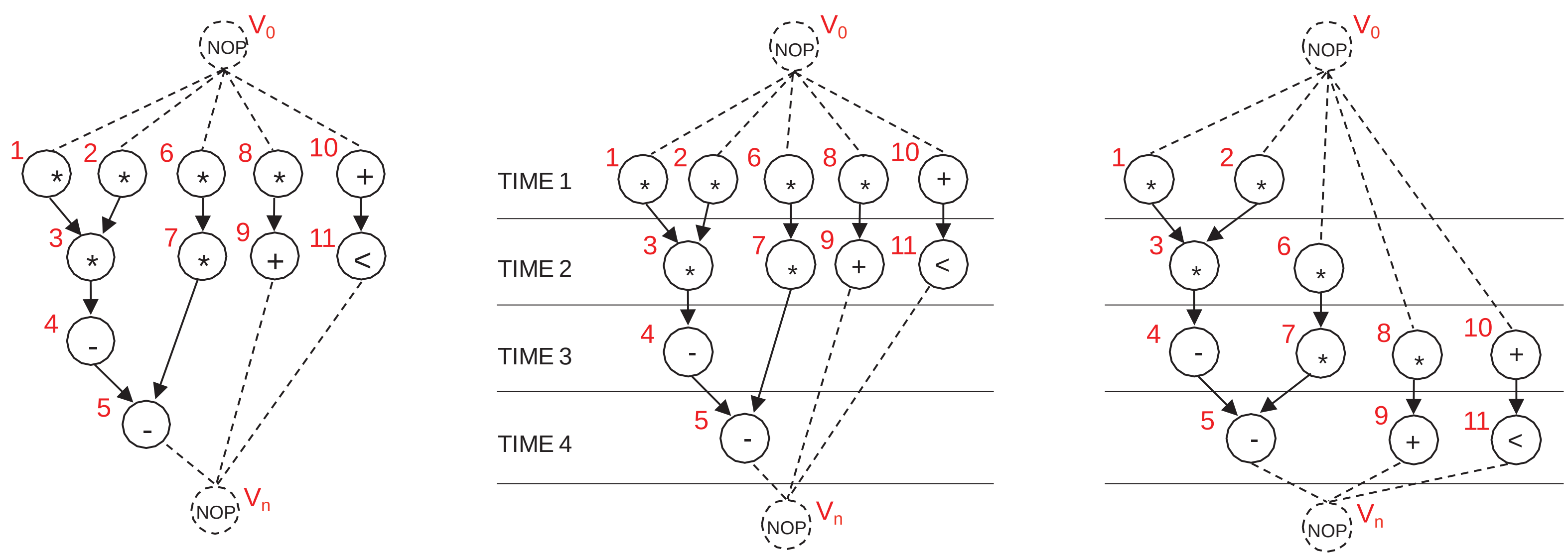
- Force-Directed Scheduling
  - Minimum resource under latency constraint

```
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 $\{[t^S_i, t^L_i];\ i = 0, 1, \ldots, n\}$
  - Earliest and latest start times can be computed by ASAP and ALAP algorithms

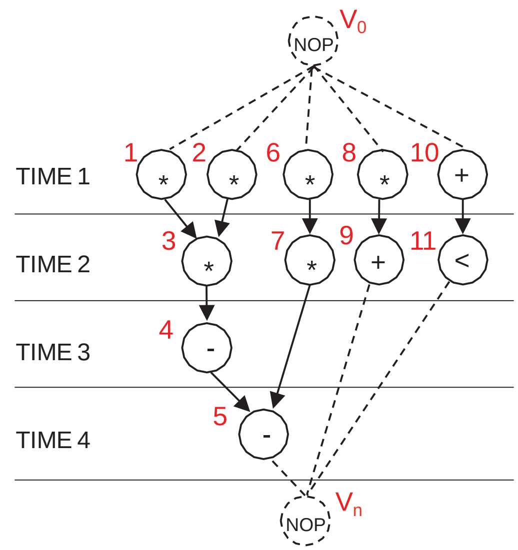


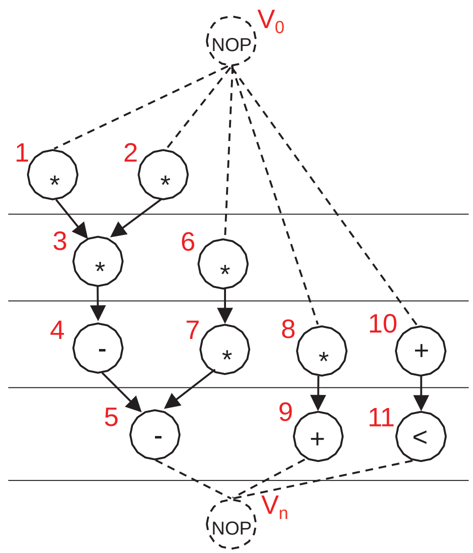
- Width of time frame of an operation is equal to its mobility plus 1

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

ASAP time = 1

ALAP time = 1

time frame = [1, 1]

operation v_2

ASAP time = 1

ALAP time = 1

time frame = [1, 1]

operation v_6

ASAP time = 1

ALAP time = 2

time frame = [1, 2]

operation v_8

ASAP time = 1

ALAP time = 3

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

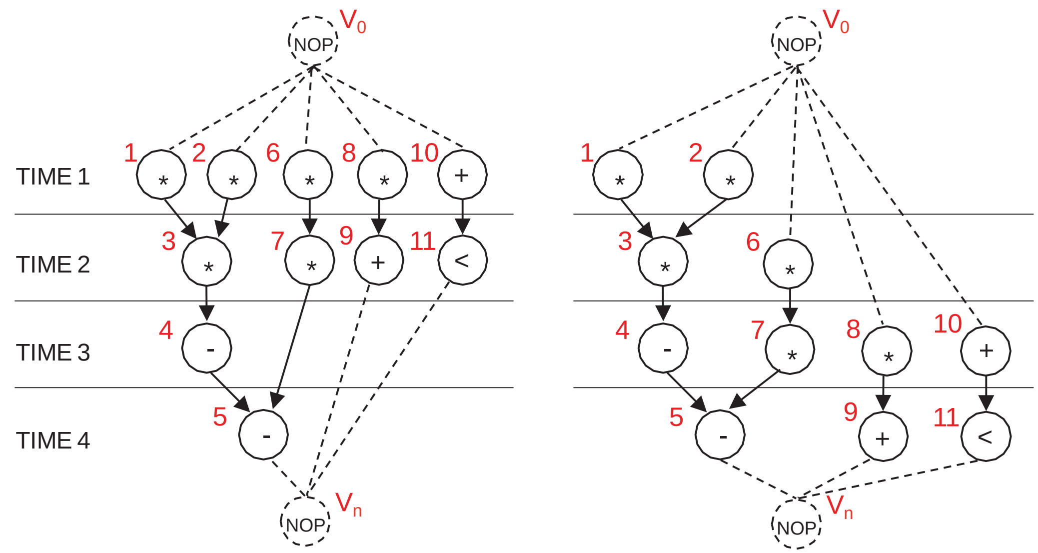
- Denoted the probability of the operations at time l by $\{p_i(l);\ i = 0, 1, \ldots, n\}$

- 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

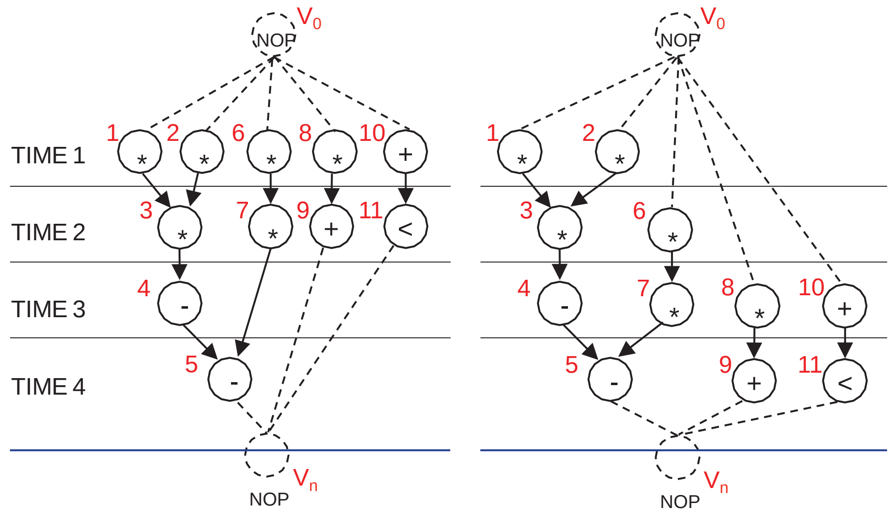
- Type Distribution is the sum of probabilities of the operations implemented by a specific resource at any time step of interest
  - Denote distribution at time l by $\{q_k(l);\ k = 1, 2, \ldots, n_{res}\}$

- 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



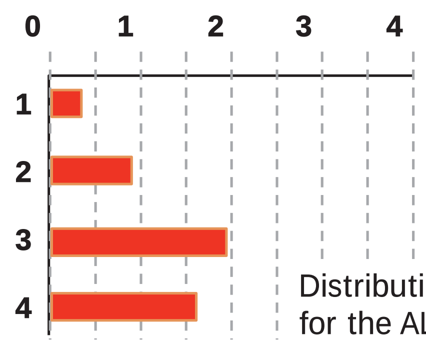
	p(1)	p(2)	p(3)	p(4)
v_4 = [3, 3], width = 1	0	0	1	0
v_5 = [4, 4], width = 1	0	0	0	1
v_9 = [2, 4], width = 3	0	0.3	0.3	0.3
v_{10} = [1, 3], width = 3	0.3	0.3	0.3	0
v_{11} = [2, 4], width = 3	0	0.3	0.3	0.3

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

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

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

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

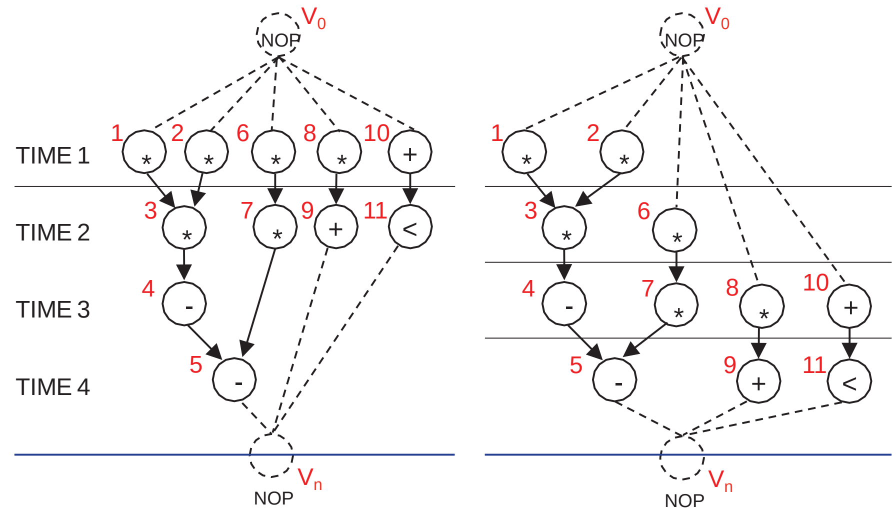


Distribution graph for the ALU

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



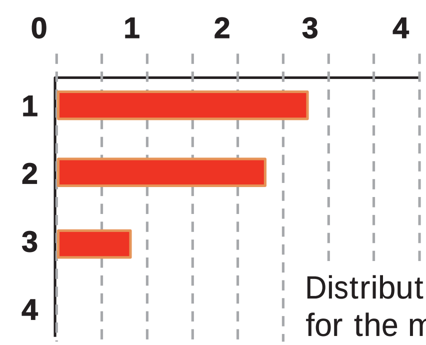
		p(1)	p(2)	p(3)	p(4)
v_1	= [1, 1], width = 1	1	0	0	0
v_2	= [1, 1], width = 1	1	0	0	0
v_3	= [2, 2], width = 1	0	1	0	0
v_6	= [1, 2], width = 2	0.5	0.5	0	0
v_7	= [2, 3], width = 2	0	0.5	0.5	0
v_8	= [1, 3], width = 3	0.3	0.3	0.3	0

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

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

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

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



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

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

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

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

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

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

Distribution graph values to time step 1 and 2

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

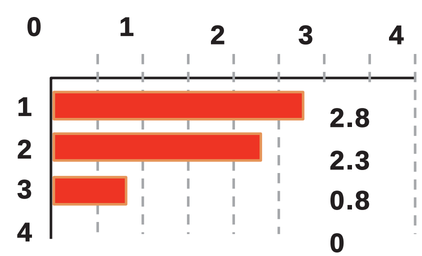
0 indicates that v6 is NOT scheduled in time 1, minus the operator probability in time 2

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

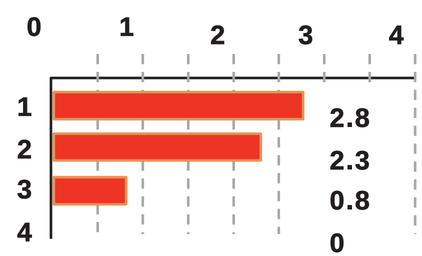
How does this impact other operations?

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

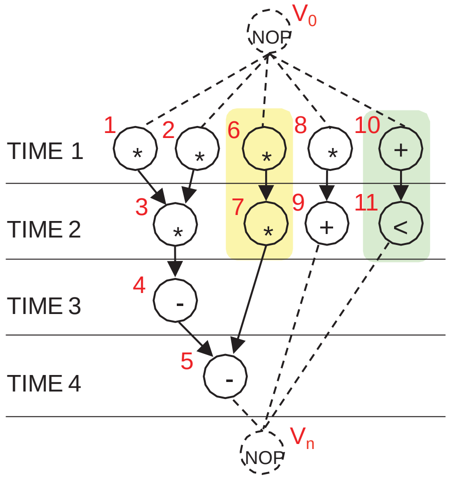
Distribution graph for the multiplier



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)

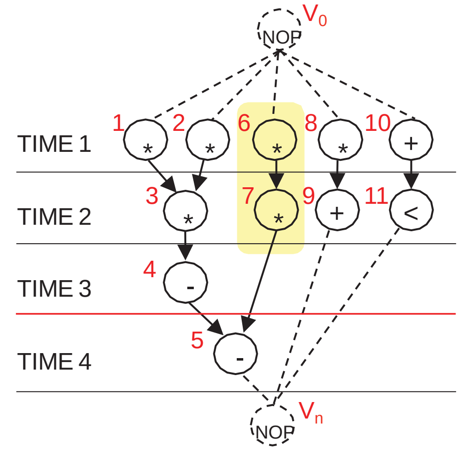
Example 7

- 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

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



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

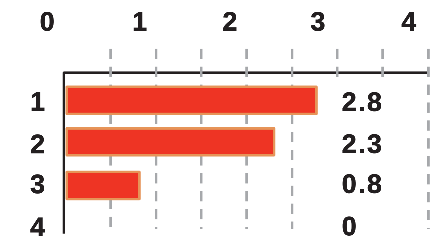
no predecessor effected

Predecessor force = 0

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

Successor force = 0

Distribution graph for the multiplier



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

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)

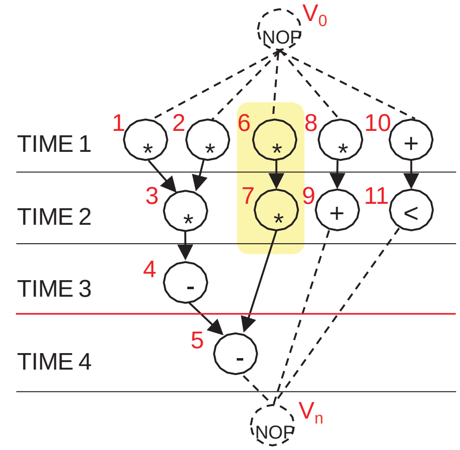
Example 7

- 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

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



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

no predecessor effected

Predecessor force = 0

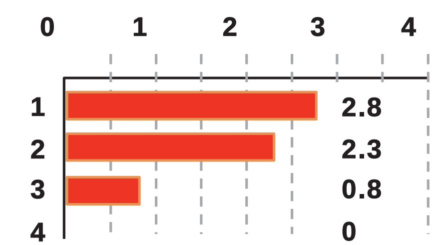
v_7 can only be scheduled at time 3

Successor force = sum of self forces of implicitly scheduled operations

= 2.3(0-0.5) + 0.8(1-0.5)

= -0.75

Distribution graph for the multiplier



Total force = Self Force + Predecessor force + Successor force

= -0.25 + 0 + -0.75

= -1

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)

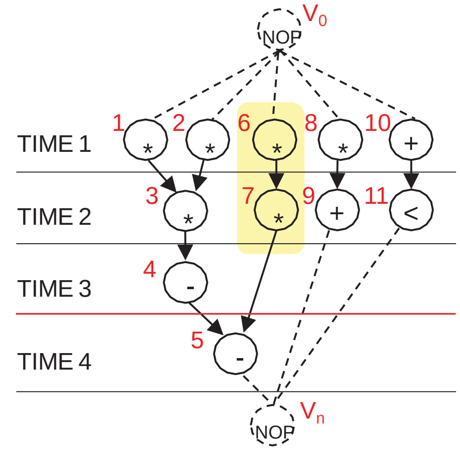
Example 7

- 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

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



Assuming v6 assigned to time step 1

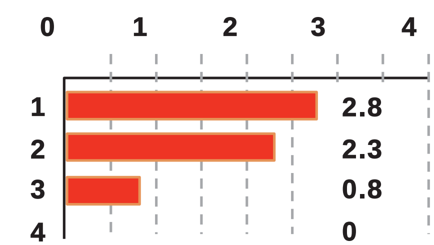
Total force = 0.25

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);
}
```

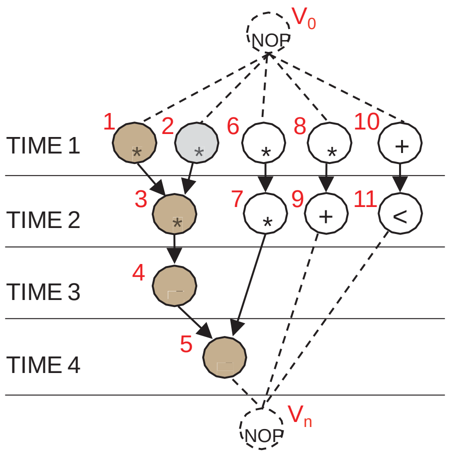
At each iteration time frame, probabilities, and forces need to be recalculated

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*