TU WIEN Informatics

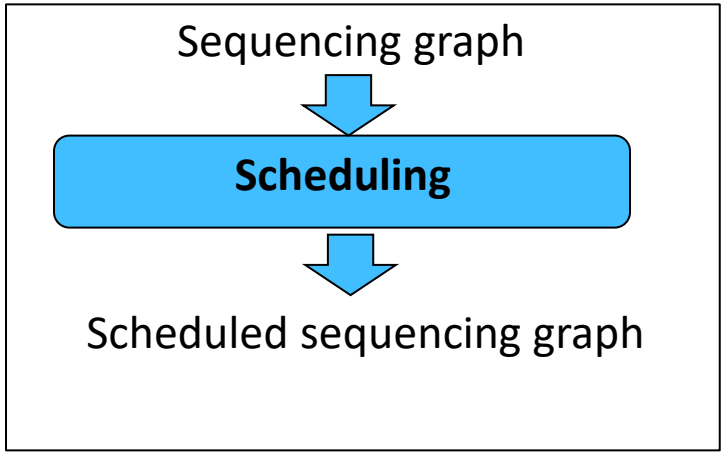
Advanced Computer Architecture

D2 –High Level Synthesis (HLS) – Scheduling Algorithms

Daniel Mueller-Gritschneder

D2-1 The Scheduling Task

Overview - Scheduling



- Unconstrained resources
- Resource constrained

Goal:
Minimize latency

- Timing constrained

Goal:
Minimize number of operational units

Problem Formulation

- Given:
  - Sequencing graph unit: $G_{s,u}(V_u, E_u) \quad V_u = v_x, ..., v_y \subset V_s$
  - x: Index of Source NOP node
  - y: Index of Sink NOP node with y>x
  - Execution delay of operations: $D = \{d_i : i = x, ..., y\}$
    - Known and data independent
    - **NOP have execution delay of zero**.

- Wanted:
  - Start time for each operation:

$$T = \{t_i : i = x, ..., y\}$$

Problem Formulation

- Scheduling is a function τ:

with

$$\tau : V_u \rightarrow \mathbb{Z}^+; \quad \tau(v_i) = t_i;$$
$$t_i \geq t_j + d_j \quad \forall_{i,j} : (v_j, v_i) \in E_u$$

  - **Constraints**: Starting time of an operation must be at least as large as the starting time of all predecessor operations plus their execution delay.
  - **Result**: Scheduled sequence graph. Each node is marked with its starting time.

- Latency of an schedule:

$$\Lambda = t_y - 1$$

Problem Formulation

- Example: DE-Solver

C-Code section

```
repeat {
        x1 = x+dx;
        u1 = u-3*x*u*dx-3*y*dx;
        y1 = y+u*dx;
        x=x1;u=u1;y=y1;
} until (x1 < a);
```

Three address code

```
B1: x1 = x+dx;
    t1 = 3*y;
    t2 = dx*t1;
    t3 = u*dx;
    t4 = 3*x;
    t5 = t3*t4;
    t6 = u-t5;
    u1 = t6-t2;
    t7 = u*dx;
    y1=  y+t7;
    x=x1;
    u=u1;
    y=y1;
    if x1 >= a goto B1;
```

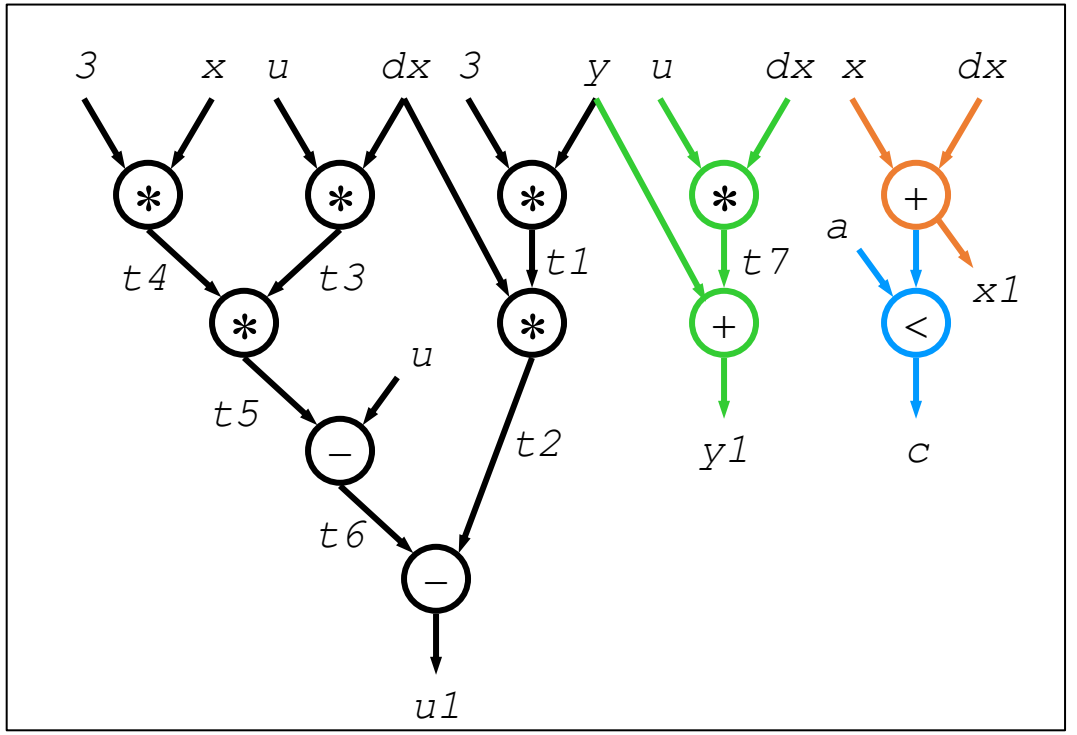
Problem Formulation

- Example: DE-Solver

Three address code

```
B1: x1 = x+dx;
    t1 = 3*y;
    t2 = dx*t1;
    t3 = u*dx;
    t4 = 3*x;
    t5 = t3*t4;
    t6 = u-t5;
    u1 = t6-t2;
    t7 = u*dx;
    y1=  y+t7;
    x=x1;
    u=u1;
    y=y1;
    if x1 >= a goto B1;
```

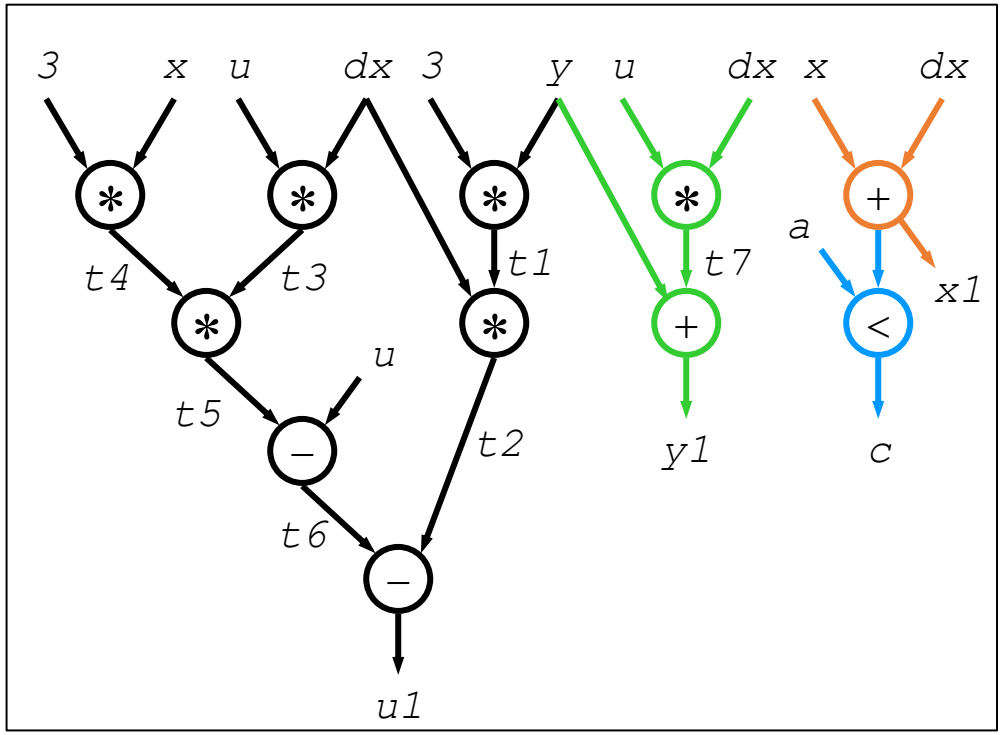
Data flow graph



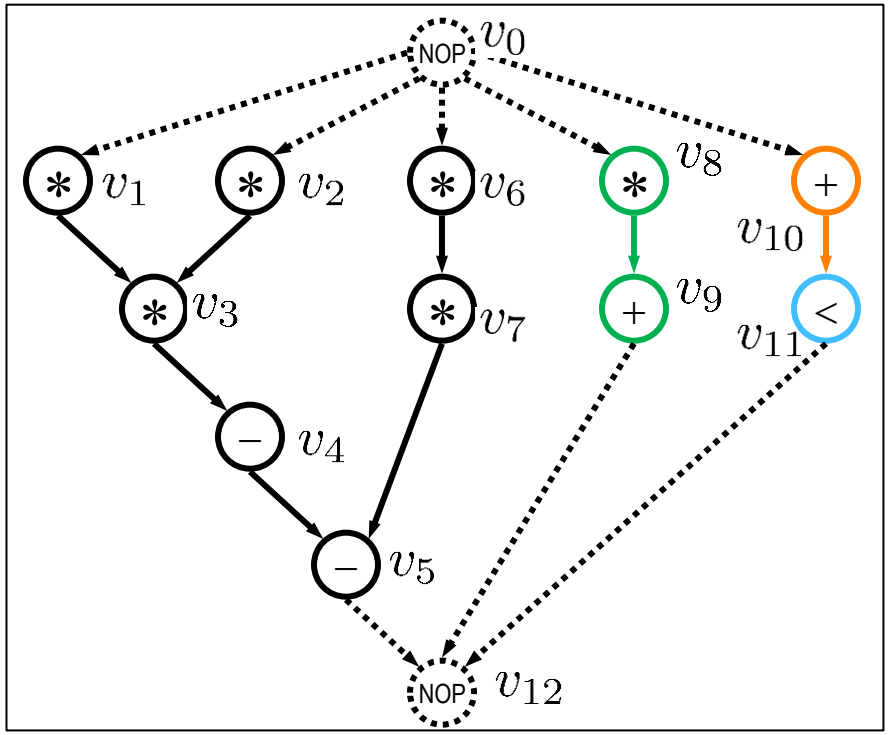
Problem Formulation

- Example: DE-Solver

Data flow graph



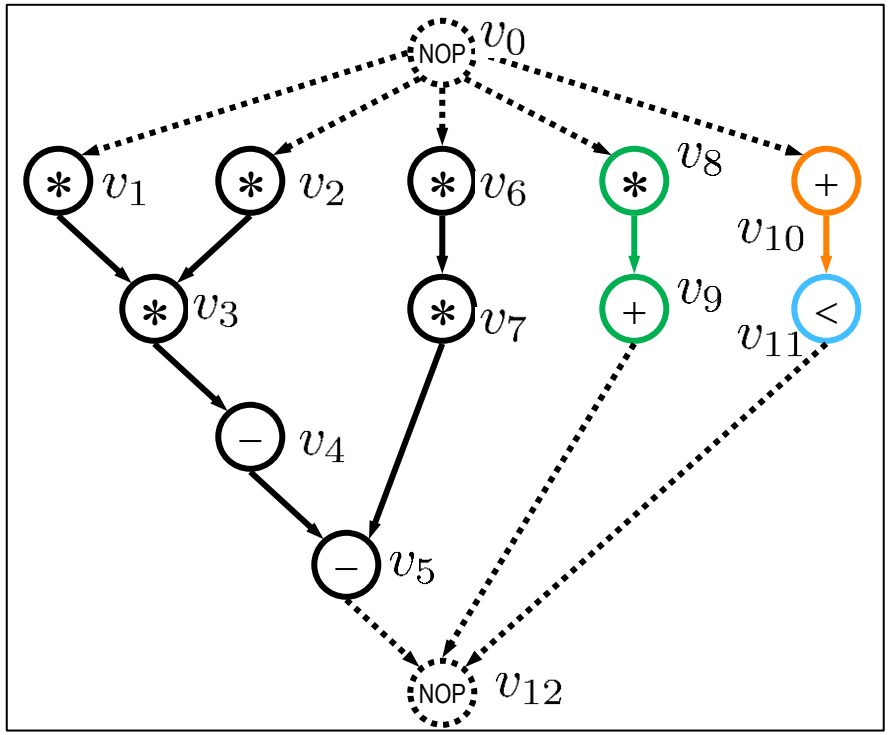
Sequencing graph unit



Problem Formulation

- Example: DE-Solver

Sequencing graph unit



Execution delays

$$d_1 = 1 \quad d_0 = 0$$
$$d_2 = 1 \quad d_{12} = 0$$
$$d_3 = 1$$
$$d_4 = 1$$
$$d_5 = 1$$
$$d_6 = 1$$
$$d_7 = 1$$
$$d_8 = 1$$
$$d_9 = 1$$
$$d_{10} = 1$$
$$d_{11} = 1$$

D2-2 As-soon-as-possible (ASAP) Schedule

As-soon-as-possible (ASAP) Schedule Constraint

- Schedule for unconstrained resources.
- Goal: Minimal latency
- Solution: Topological sorting of the sequencing graph.

- ASAP start time for node v_i :

$$t_i^S = \max_{j:(v_j,v_i)\in E_u} (t_j^S + d_j)$$

- Quadratic complexity: $O(|V|^2)$

As-soon-as-possible (ASAP) Scheduling Algorithm

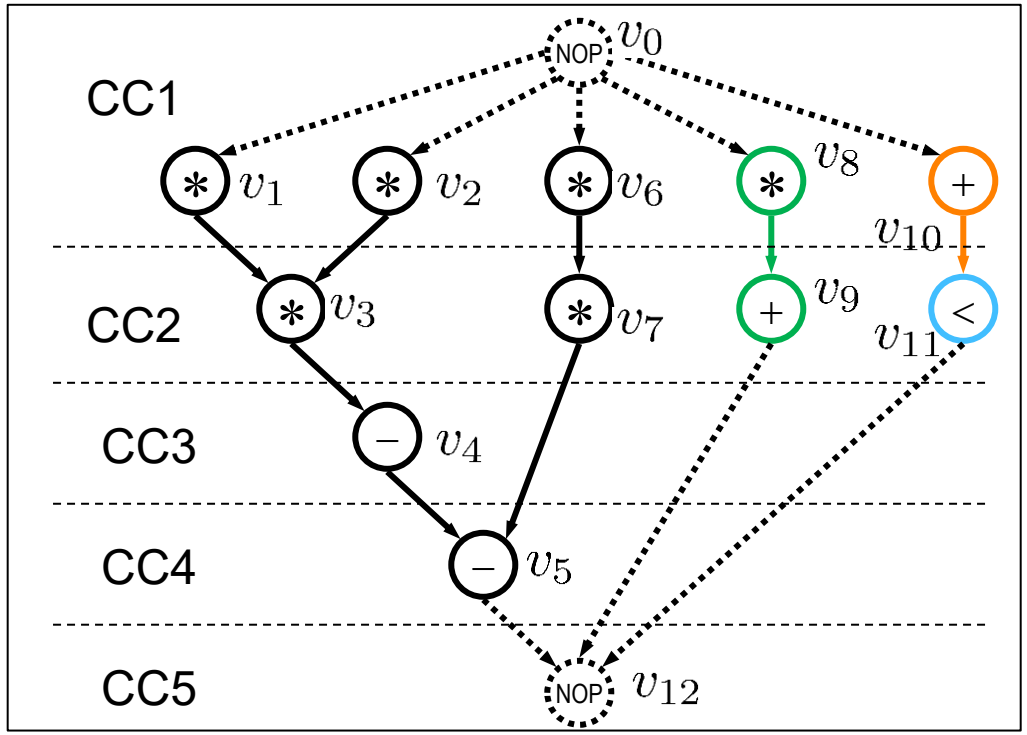
- Algorithm:

```
ASAP_schedule(G_s,u(V_u,E_u)) {
    Start time of node v[x]: t_S[x]=1;
    repeat {
      Select node v[i], whose direct predecessors v[j] all
        have been assigned a starting time.
      Set start time for node v[i]:
        t_S[i]=max(t_S[j]+d[j]);
    } until node v[y] has been assigned a starting time.
    return (t_S);
}
```

As-soon-as-possible (ASAP) Schedule Example

- Example: DE-Solver

Sequencing graph unit with ASAP schedule



Starting times

$$t_0^S = t_1^S = t_2^S = t_6^S = t_8^S = t_{10}^S = 1$$

$$t_3^S = t_7^S = t_9^S = t_{11}^S = 2$$

$$t_4^S = 3$$

$$t_5^S = 4$$

$$t_{12}^S = 5$$

Latency:

$$\Lambda = t_{12}^S - 1 = 4$$

Resources:

4xMultipler, 2xALUs

D2-2 As-late-as-possible (ALAP) Schedule

As-late-as-possible (ALAP) schedule constraint

- Schedule with fixed latency (Timing constrained)
- Given Latency:

$$\Lambda^L = t_y^L - 1 = \Lambda_{max}$$

- Goal: Find latest starting time for all operations such that maximal latency constraint is met:

$$t_i^L = \min_{j:(v_i,v_j)\in E_u} (t_j^L - d_i)$$

- Same complexity as ASAP

As-late-as-possible (ALAP) Scheduling Algorithm

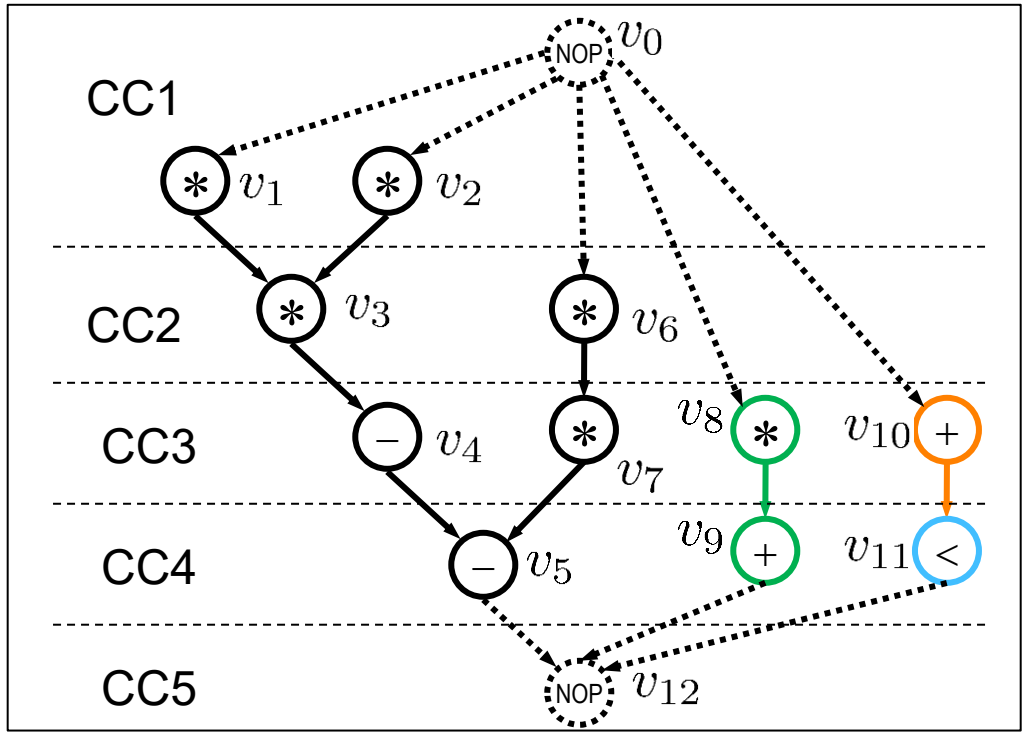
- Algorithm:

```
ALAP_schedule(G_s,u(V,E),Lambda_max) {
    Start time for node v[y]: t_L[y]=Lambda_max+1
    repeat {
        Select node v[i], whose direct successors v[j] all
          have been assigned a starting time.
        Set start time for node v[i]:
          t_L[i]=min(t_L[j]-d[i])
    } until node v[x] has been assigned starting time
    return (t_L)
}
```

As-late-as-possible (ALAP) Schedule

- Example: DE-Solver

Sequencing graph unit with ALAP schedule



Starting times

Given Latency:

$$\Lambda^L = 4$$

Resources:
2xMultipler, 3xALUs

$$t_0^L = t_1^L = t_2^L = 1$$

$$t_3^L = t_6^L = 2$$

$$t_4^L = t_7^L = t_8^L = t_{10}^L = 3$$

$$t_5^L = t_9^L = t_{11}^L = 4$$

$$t_{12}^L = 5$$

D2-3 Mobility of Operations

Mobility of Operations

- Given is upper constraint on latency:

$$\Lambda = t_y - 1 \leq \Lambda_{max}$$

- ASAP Schedule: Minimal start times for operations
- ALAP Schedule: Maximal start times for operations

- Mobility of operations on time axis:

$$\mu_i = t_i^L - t_i^S, \quad i = x, ..., y$$

Mobility of Operations

- For operations with $\mu_i = 0$
  - The start time is fixed: $t_i = t_i^L = t_i^S$
  - Operations are located on critical path.
    (Not the same as critical path in logic circuits)

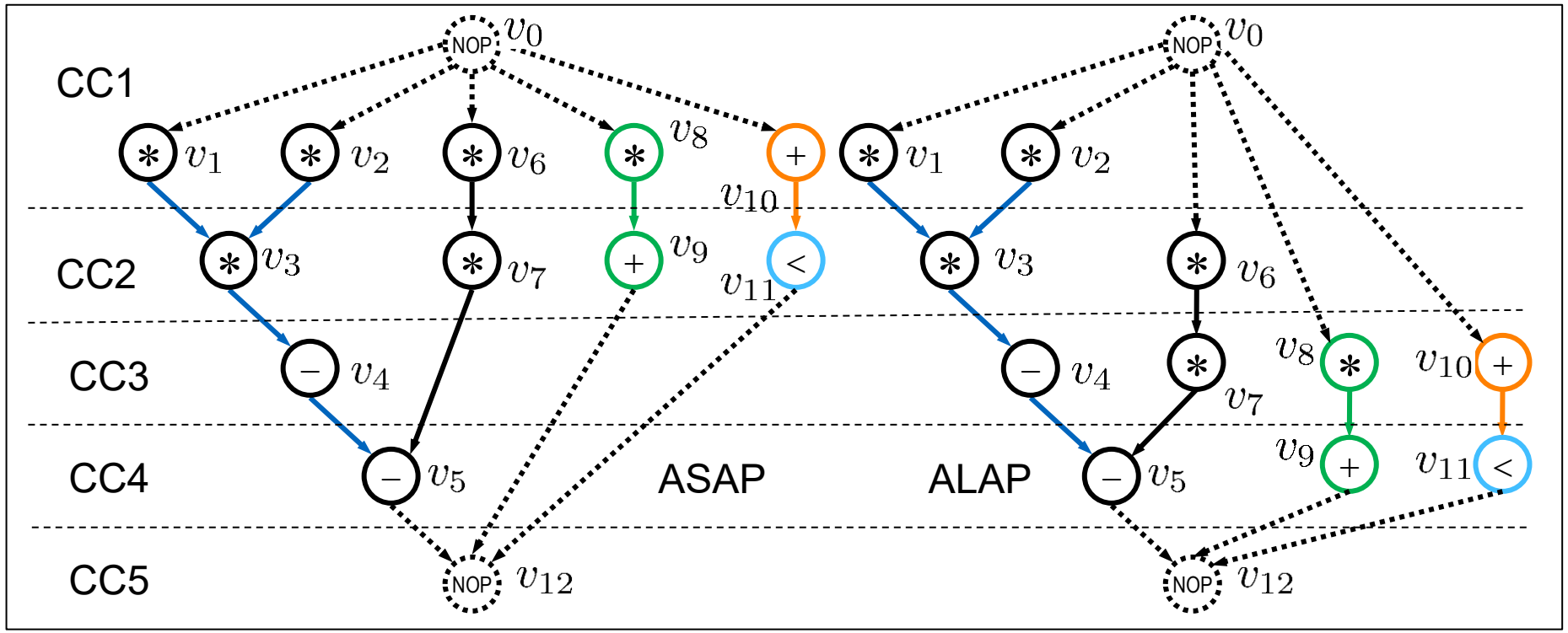
- There is no schedule for latency constraint $\Lambda \leq \Lambda_{max}$

  possible, if $t_y^S > \Lambda_{max} + 1$

  or, if $t_x^L < 1$

Mobility of Operations

- Example: DE-Solver



Operation i	1	2	3	4	5	6	7	8	9	10	11
ASAP	1	1	2	3	4	1	2	1	2	1	2
ALAP	1	1	2	3	4	2	3	3	4	3	4
Mobilität μ_i	0	0	0	0	0	1	1	2	2	2	2

D2-4 Hu’s Algorithm

Hu‘s Algorithm

- Goal: Minimize latency
- Resource constraint: Maximal number of resources = a
- Requirements:
  - Only one type of resource
  - All execution delays = 1
  - Operations with larger execution delay can be split into several operations with execution delay=1.
- Properties:
  - Linear Complexity: $O(n)$
  - Greedy algorithm
  - Optimal: Finds schedule with minimal latency.

Hu‘s Algorithm

- Set of ready operations:

$$U_{act} = \{v_i \mid \forall_{j:(v_j,v_i)\in E_u}\ t_j + d_j \leq t_{act}\} \quad \text{Direct predecessors finished}$$

- Label each node with length of longest path from this node to the sink NOP node: α_i

- Set of operations to start: S_{act}
  - Must be operations that are ready
  - Must be less or equal number as available resources a
  - The label α_i should be maximal

Hu‘s Algorithm

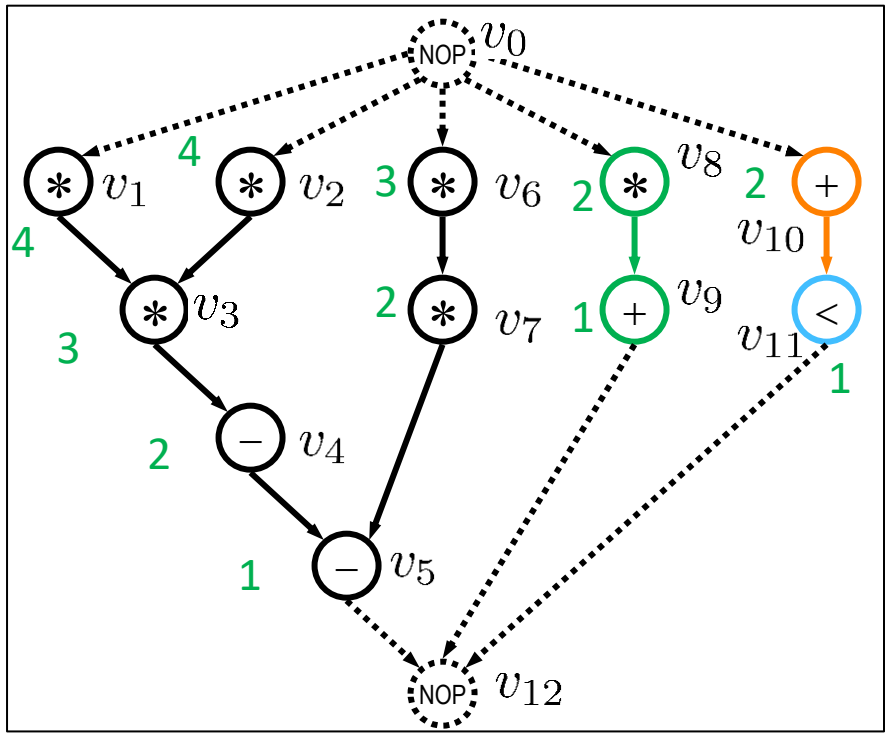
- Algorithm:

```
HU(G_s,u(V,E),a) {
    Label nodes v[i] with max. path length alpha[i] to sink v[y]
    Set start time for source node v[x]: t_HU[x]=1
    Set t_act=1
    repeat {
        Select set of nodes S_act, such that for v[i] in S_act:
            1.  v[i] is in U_act
            2.  alpha[i] of v[i] in S_act is maximal
            3.  Number of elements in S_act: |S|<=a
        Set start time of all v[i] in S_act: t_HU[i]=t_act
        Set t_act=t_act+1
    } until sink node v[y] was assigned a start time
    return (t_HU)
}
```

Hu‘s Algorithm

- Example: DE-Solver

3 ALUs for all operations (+,-,*,<)



0. Label nodes
1. Iteration:
   $S_{act}=\{v_1, v_2, v_6\}$
   Start times: $t_1^{Hu} = t_2^{Hu} = t_6^{Hu} = 1$
2. Iteration:
   $S_{act}=\{v_3, v_7, v_8\}$
   Start times: $t_3^{Hu} = t_7^{Hu} = t_8^{Hu} = 2$
3. Iteration:
   $S_{act}=\{v_4, v_9, v_{10}\}$
   Start times: $t_4^{Hu} = t_9^{Hu} = t_{10}^{Hu} = 3$
4. Iteration:
   $S_{act}=\{v_5, v_{11}\}$
   Start times: $t_5^{Hu} = t_{11}^{Hu} = 4$

D2-5 List Scheduling

List Scheduling Priorities

- Resource Constrained (Goal: Minimize latency)
  - Number of resources of type k: a_k
  - Priority equals maximal sum of execution delays on paths to sink

$$\mathrm{Prio}(v_i) = \max_v(\sum_{w \in H_v} d_w) \text{ with } H_v \text{ equals path from } v_i \text{ to sink}$$

- Time constrained (Goal: Minimize resources)
  - Maximal latency: $\Lambda \leq \Lambda_{max}$
  - Slack of a node (distance to ALAP start time):
  - Priority at time t_{act} equals slack: $s_i = t_i^L - t_{act}$

- Heuristic and greedy algorithm based on priorities

$$\mathrm{Prio}(v_i) = s_i = t_i^L - t_{act}$$

List Scheduling Operation Sets

- Set of candidates ready to be executed on resource of type k:

$$U_{act,k} = \left\{ v_i \;\middle|\; \begin{array}{ll} v_i \text{ of type } k & \wedge \\ \forall_{j:(v_j,v_i)\in E_u}\ t_j + d_j \leq t_{act} & \text{Direct predecessors finished} \end{array} \right\}$$

- Set of running operations on resources of type k:

$$T_{act,k} = \{v_l \mid v_l \text{ of type } k \ \wedge \ t_l + d_l > t_{act}\}$$

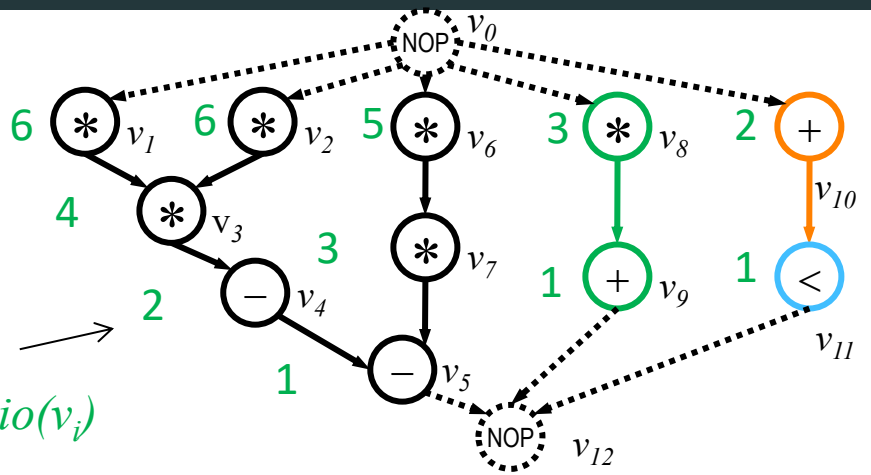
List Scheduling with Resource Constraint - Algorithm

- Algorithm for Resource constrained latency minimization

```
LIST_L(G_s,u(V,E),a) {
    Set start time of source node v[x]: t_LR[x]=1
    t_act=1
    repeat {
      foreach type of resource k=1,2,… {
          Find set of candidate operations U_act[k]
          Find set of running operations T_act[k]
          Select starting operations v[i] in S_act[k] such that:
            1. v[i] in U_act[k]
            2. Priorities Prio(v[i]) maximal
            3. Number of running and starting operations smaller than
                resource number:|S_act[k]| + |T_act[k]| <= a[k]
            Set start time of v[i] in S_act[k]: t_LR[i]=t_act
        }
        t_act=t_act+1
    } until sink node v[y] was assigned a start time
    return (t_LR)
```

List Scheduling with Resource Constraint - Example

- Example DE-Solver
- Resource Constrained (2xMULT, 1xALU)



Priorities $Prio(v_i)$

- Schedule:

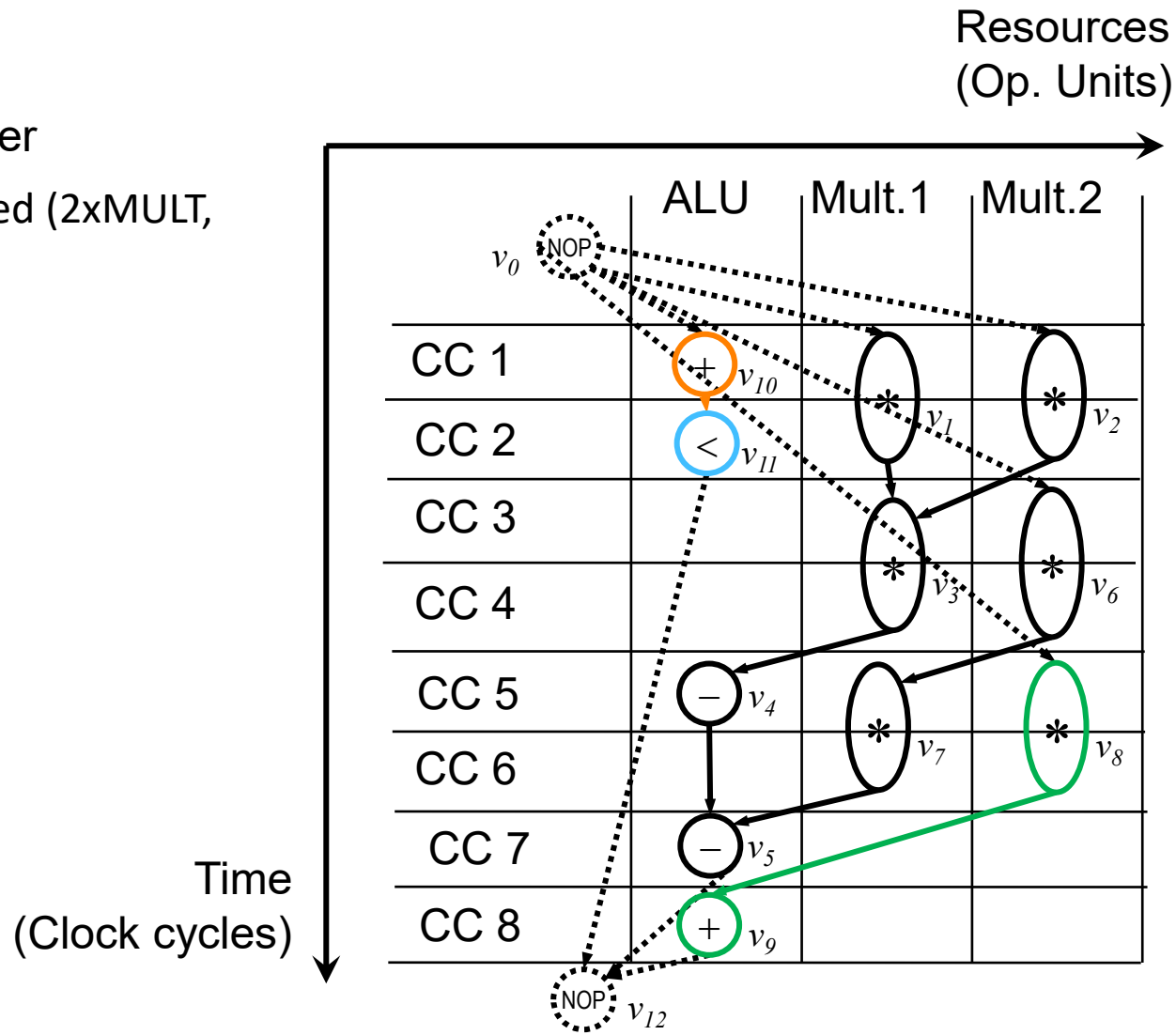
It.	2xMULT d=2 Cycles			1xALU d=1 Cycle		Start time
t_{act}	$U_{akt,mult}$	$T_{act,mult}$	$S_{act,mult}$	$U_{act,alu}$	$S_{act,alu}$	t_i
1	$\{v_1,v_2,v_6,v_8\}$	$\{\}$	$\{v_1,v_2\}$	$\{v_{10}\}$	$\{v_{10}\}$	$t_1=t_2=t_{10}=1$
2	$\{v_6,v_8\}$	$\{v_1,v_2\}$	$\{\}$	$\{v_{11}\}$	$\{v_{11}\}$	$t_{11}=2$
3	$\{v_3,v_6,v_8\}$	$\{\}$	$\{v_3,v_6\}$	$\{\}$	$\{\}$	$t_3=t_6=3$
4	$\{v_8\}$	$\{v_3,v_6\}$	$\{\}$	$\{\}$	$\{\}$	
5	$\{v_7,v_8\}$	$\{\}$	$\{v_7,v_8\}$	$\{v_4\}$	$\{v_4\}$	$t_4=t_7=t_8=5$
6	$\{\}$	$\{v_7,v_8\}$	$\{\}$	$\{\}$	$\{\}$	
7	$\{\}$	$\{\}$	$\{\}$	$\{v_5,v_9\}$	$\{v_5\}$	$t_5=7$
8	$\{\}$	$\{\}$	$\{\}$	$\{v_9\}$	$\{v_9\}$	$t_9=8$

Latency

$$\Lambda = 8$$

List Scheduling with Resource Constraint – Example in TRP Plane

- Example: De-Solver
- Resource Constrained (2xMULT, 1xALU)
- Possible Binding



List Scheduling with Timing Constraint – Algorithm Part 1

- Algorithm: Timing constrained resource minimization – Part 1

```
LIST_R(G_s,u(V,E),Lambda_max) {
    Set Number of resources: a[k]=1 for all k
    t_L = ALAP_Schedule(G_s,u(V,E),Lambda_max)
    if t_L[x] < 1 then return(„No schedule possible“)
    Set start time of node v[x]: t_LT[x]=1
    t_act=1
    repeat {
      foreach type of resource k {
        Find set of candidate nodes U_act[k]
        Find set of running nodes T_act[k]
        Compute slack s[i] = t_L[i] – t_act for v[i] in U_act[k]
        Place all v[i] from U_act[k] into S_act[k], with slack s[i]=0
        Set start time of v[i] in S_act[k]: t_LT[i]=t_act
        ...
```

List Scheduling with Timing Constraint – Algorithm Part 2

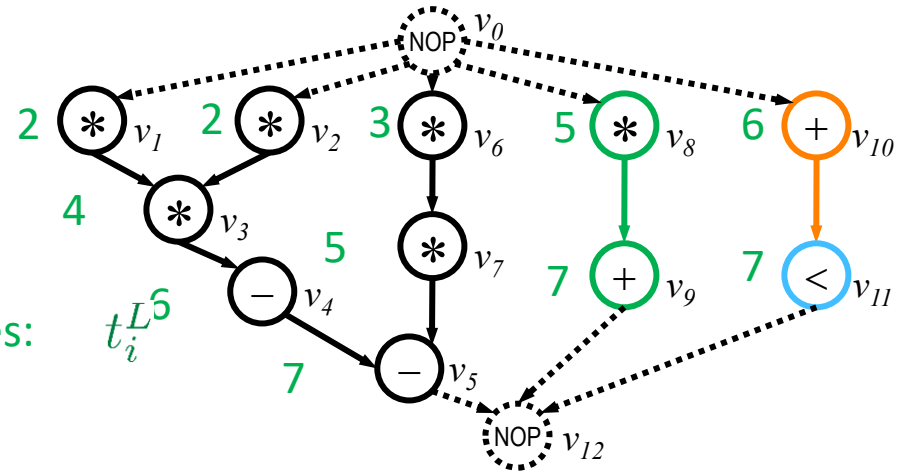
- Algorithm: Timing constrained resource minimization – Part 2

```
          if |S_act[k]| + |T_act[k]| > a[k] then {
            Update a[k]: a[k] = |S_act[k]| + |T_act[k]|
          }
          if |S_act[k]| + |T_act[k]| < a[k] then {
             { Place nodes v[l] from U_act[k] without S_act[k]
               into R_act[k],
                Such that slack s[l] for v[l] in R_act[k] minimal }
                until |S_act[k]| + |T_act[k]| +  | R_act[k] | = a[k] or
                       no more nodes in U_act[k]
             Set start time of nodes v[l] in R_act[k]:
                t_LT[l]=t_act
          }
       }
       t_act=t_act+1
    } until sink node v[y] was assigned a start time
    return (t_LT);
}
```

List Scheduling with Timing Constraint – Example

- Example: DE-Solver
- Resource minimization
- Timing constrained: $\Lambda_{max} = 7$

ALAP times: t_i^L



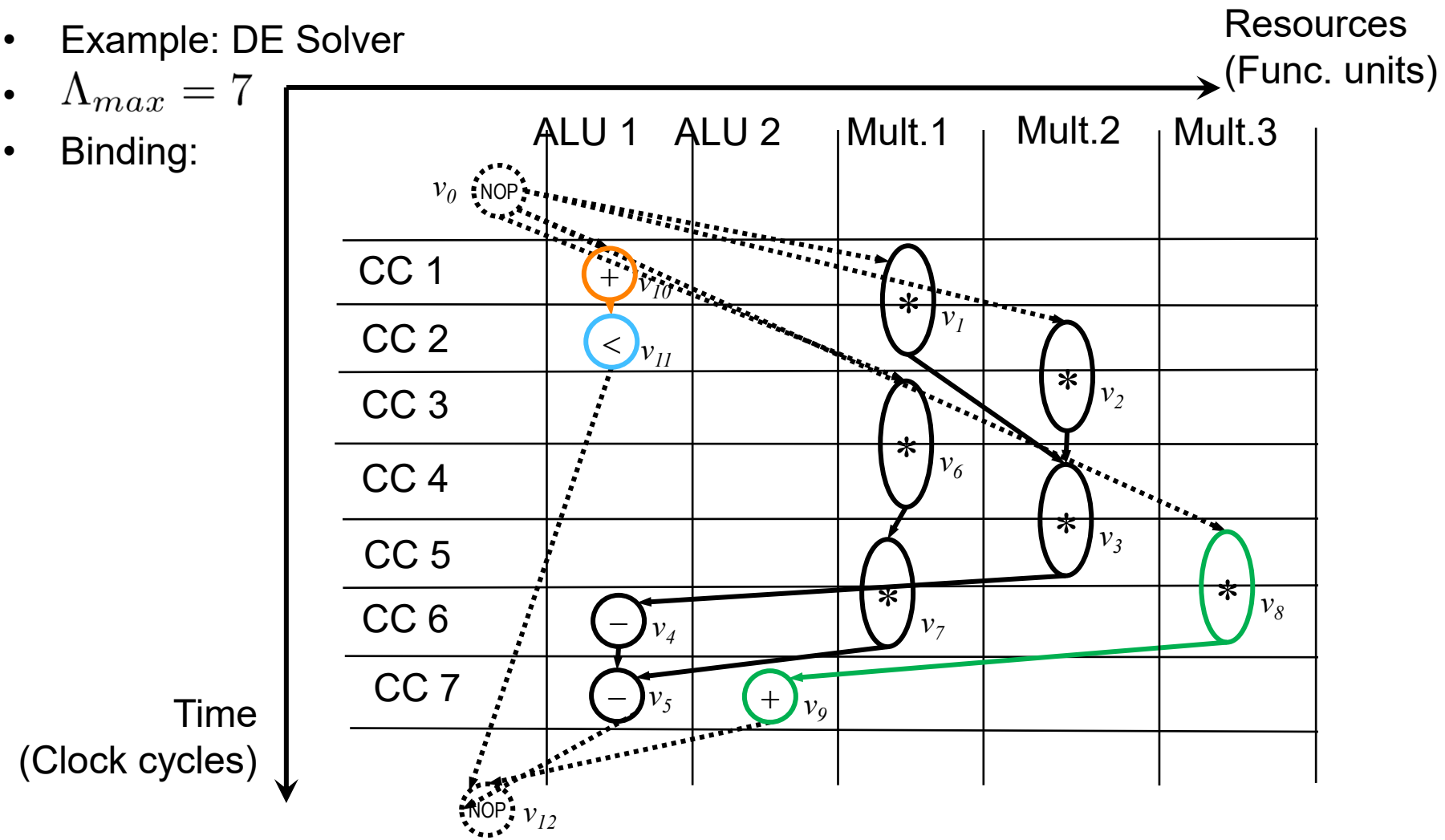
- Schedule:

3 x MULT und 2 x ALU

lt.	MULT d=2 Takte						ALU d=1Takt					
t_{act}	$U_{akt,mult}$	Slack	$T_{act,mult}$	$S_{act,mult}$	$R_{act,mult}$	a_{mult}	$U_{act,alu}$	Slack	$S_{act,alu}$	$R_{act,alu}$	a_{alu}	t_i
1	{v1,v2, v6,v8}	s_1=1,s_2=1,s_6 =2,s_8=4	{}	{}	{v1}	1	{v10}	s_{10}=5	{}	{v10}	1	t_1=t_{10}=1
2	{v2,v6,v8}	s_2=**0**, s_6=1, s_8=3	{v1}	{v2}	{}	2	{v11}	s_{11}=5	{}	{v11}	1	t_2=t_{11}=2
3	{v6,v8}	s_6=**0**, s_8=2	{v2}	{v6}	{}	2	{}	( )	{}	{}	1	t_6=3
4	{$v_{3,v8}$}	s_3=**0**, s_8=1	{v6}	{v3}	{}	2	{}	( )	{}	{}	1	t_3=4
5	{v_7,v_8}	s_7=**0**, s_8=**0**	{v3}	{v_7,v_8}	{}	3	{}	( )	{}	{}	1	t_7=t_8=5
6	{}	( )	{v_7,v_8}	{}	{}	3	{v4}	s_4=**0**	{v4}	{}	1	t_4=6
7	{}	( )	{}	{}	{}	3	{v_5,v_9}	s_5= s_9=**0**	{v_5,v_9}	{}	2	t_5=t_9=7

List Scheduling with Timing Constraint – TRP Plane

- Example: DE Solver
- $\Lambda_{max} = 7$
- Binding:



List Scheduling with Timing Constraint – Improved Version with Restart

- Improved Algorithm: Timing-constrained resource minimization

- Restart algorithms each time number of resources was increased, do not reset number of resources to 1, but start with the last value.

D2-6 Force Directed Scheduling

Force-directed Scheduling - Introduction

- Heuristic based on a force-based model
- Timing constrained resource minimization
- Published: Paul & Knight, TCAD 1989

Force-directed Scheduling – Distribution of Start Times

- Time frame of possible starting times for node v_i:

$$T_i = [t_i^{ASAP} \quad t_i^{ALAP}]$$

with width of time frame equals $\mu_i + 1$

- Distribution for starting time of node v_i at time t_{act}

$$p_i(t_{act}) = \begin{cases} \frac{1}{\mu_i+1} & \forall_{t_{act} \in T_i} \\ 0 & \forall_{t_{act} \notin T_i} \end{cases}$$

  - Uniform distribution in the time frame.

Force-directed Scheduling – Resource Demand

- Demand for resources of type k at clock cycle t_{act} :

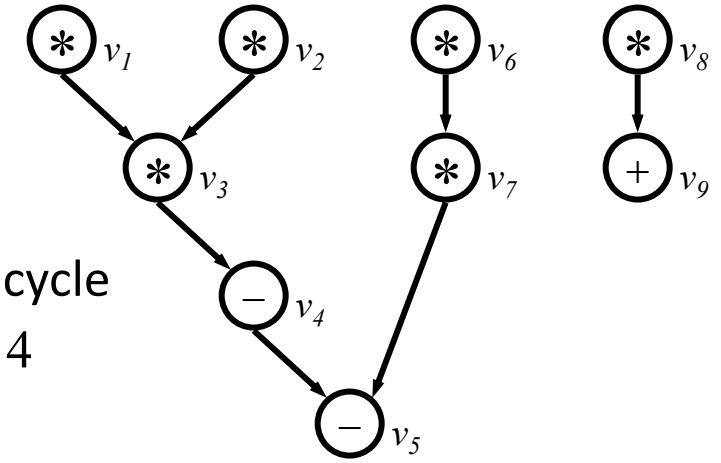
$$q_k(t_{act}) = \sum_{\{i:op_i \text{ ist von Typ } k\}} p_i(t_{act})$$

- Mean demand for resources of type k in the time frame T_i:

$$m_{k,i} = \frac{1}{\mu_i + 1} \sum_{t_p = t_i^{ASAP}}^{t_i^{ALAP}} q_k(t_p)$$

Force-directed Schedule – Example Distribution of Starting Times

- Example: DE-Solver
  - Multipliers (k=MULT)
  - Execution delay = 1 clock cycle
  - Timing constraint: $\Lambda_{max} = 4$



CC1 2,83
CC2 2,33
CC3 0,83
CC4 0

Distribution for starting times

	v_1	v_2	v_3	v_6	v_7	v_8	
T_i	[1,1]	[1,1]	[2,2]	[1,2]	[2,3]	[1,3]	$\downarrow q_{MULT}(t_{act}) \downarrow$
$p_i(1)$	1	1	0	1/2	0	1/3	17/6=2,83
$p_i(2)$	0	0	1	1/2	1/2	1/3	7/3=2,33
$p_i(3)$	0	0	0	0	1/2	1/3	5/6=0,83
$p_i(4)$	0	0	0	0	0	0	0

Force-directed Scheduling – Self Force

- Self force:

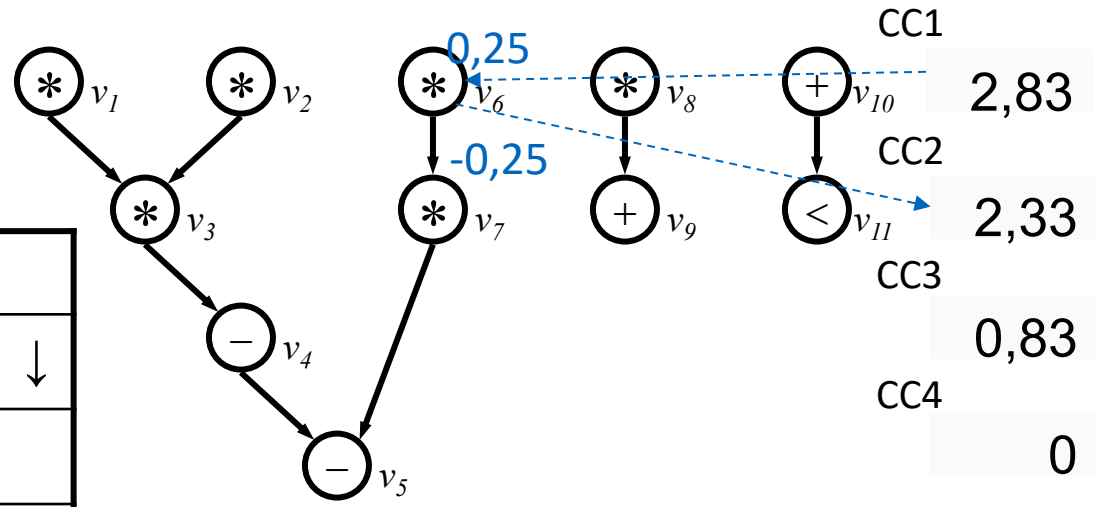
$$F_i^S(t_{act}) = q_k(t_{act}) - m_{k,i}$$

  - Difference between demand of resource of type k in clock cycle t_{act} and mean demand for resource of type k in the time frame T_i of the node.
  - $F_i^S(t_{act}) > 0$: In this clock cycle demand for this resource type is high. Push node v_i away from t_{act} by positive self force.
  - $F_i^S(t_{act}) < 0$: In this clock cycle demand for this resource type is low. Pull node v_i near to t_{act} by negative self force.

Force-directed Scheduling – Self Force for Example

- Example: DE-Solver

	v_6	
T_i	[1,2]	$\downarrow\ q_{MULT}(t_{act})\ \downarrow$
$p_i(1)$	1/2	17/6
$p_i(2)$	1/2	7/3



$$m_{1,6} = \frac{1}{\mu_6+1}(q_{MULT}(1) + q_{MULT}(2)) = \frac{1}{1+1} * (\frac{17}{6} + \frac{7}{3}) = \frac{31}{12}$$

$$F_6^S(1) = q_{MULT}(1) - m_{MULT,6} = \frac{17}{6} - \frac{31}{12} = \frac{3}{12} = \frac{1}{4}$$

$$F_6^S(2) = q_{MULT}(2) - m_{MULT,6} = \frac{14}{6} - \frac{31}{12} = -\frac{3}{12} = -\frac{1}{4}$$

- Self force for clock cycle 1 positive because the demand for multipliers is high and negative for clock cycle 2, because demand is lower.

Force-directed Scheduling – Shift of Time Frames of Successors / Predecessors

- Selection of start time for node changes time frames for direct predecessor and successor nodes.
  - Node v_j is direct predecessor or successor of v_i.
  - Start time for node v_i is selected to:
  - > New time frame and mobility for nodes v_j: $t_i = t_{act}$

$$\tilde{T}_j = [\tilde{t}_j^{ASAP} \ \tilde{t}_j^{ALAP}] \qquad \tilde{\mu}_j = \tilde{t}_j^{ALAP} - \tilde{t}_j^{ASAP}$$

  - > New mean demands for resources:

$$\tilde{m}_{k,i} = \frac{1}{\tilde{\mu}_i + 1} \sum_{t_p = \tilde{t}_i^{ASAP}}^{\tilde{t}_i^{ALAP}} q_k(t_p)$$

Force-directed Scheduling - Predecessor and successor forces

- Predecessor and successor forces:

$$F_{i,j}^{V,N}(t_{act}) = \tilde{m}_{k,j} - m_{k,j}$$

  - Change of mean demand for resources of type k for predecessor and successor forces.
  - $F_{i,j}^{V,N}(t_{act}) > 0$ By setting start time of node v_i to t_{act} the successor/predecessor node $v_i$can only be scheduled in clock cycles with higher demand for resources of type k. Push $v_i$away from t_{act} by positive predecessor/successor force.
  - $F_{i,j}^{V,N}(t_{act}) < 0$ Other way around. Pull $v_j$tot_{act} by negative predecessor/successor force.

Force-directed Scheduling - Predecessor and successor forces for Example

- Example 3: DE-Solver
  - v_7 is direct successor of v_6
  - For

$$t_6 = 1 \rightarrow \tilde{T}_7 = T_7 = [2\ 3] \rightarrow F^N_{6,7}(1) = 0$$

	v_7	
T_i	[2,3]	$\downarrow\ q_{MULT}(t_{act})\ \downarrow$
$p_i(2)$	1/2	7/3
$p_i(3)$	1/2	5/6

  - For

$$t_6 = 2 \rightarrow \tilde{T}_7 = [3\ 3] \rightarrow \tilde{\mu}_7 = 0 \rightarrow \tilde{m}_{MULT,7} = \frac{1}{1+\tilde{\mu}_7} q_1(3) = \frac{5}{6}$$

$$m_{1,7} = \frac{1}{1+\mu_7}(q_{MULT}(2) + q_{MULT}(3)) = \frac{1}{2}(\frac{7}{3} + \frac{5}{6}) = \frac{19}{12}$$

$$F^N_{6,7}(2) = \tilde{m}_{MULT,7} - m_{MULT,7} = \frac{5}{6} - \frac{19}{12} = -\frac{3}{4}$$

Force-directed Scheduling – Total Force

- Total force

$$F_i^T(t_{act}) = F_i^S(t_{act}) + \sum_{\{j:(op_i,op_j)\in E\}} F_{i,j}^N(t_{act}) + \sum_{\{j:(op_j,op_i)\in E\}} F_{i,j}^V(t_{act})$$

  - Sum of self force, predecessor forces and successor forces.

- To minimize resources select starting times with minimal force, which should lead to minimal mean demand of resources of all types for the schedule.

Force-directed Scheduling - Example

- Example: DE-Solver

Higher than average demand for multipliers.

$$t_6 = 1$$

$$F_6^T(1) = F_6^S(1) = F_{6,7}^N(1) = \frac{1}{4} + 0 = \frac{1}{4}$$

Successor node is not influenced.

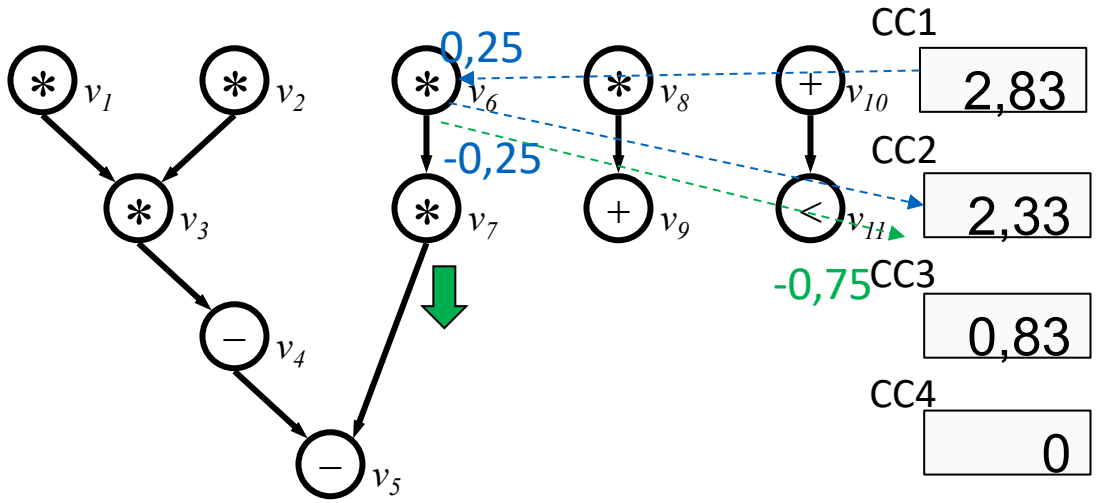
Lower than average demand for multipliers.

$$t_6 = 2$$

$$F_6^T(2) = F_6^S(2) = F_{6,7}^N(2) = -\frac{1}{4} - \frac{3}{4} = -1$$

Successor node is shifted to time frame with lower than average demand for multipliers.

Force-directed Scheduling Example



- Force will push v6 towards start time t6=2 because there is less demand for MUL in CC2 and v7 is pushed to a later start time where there is also less demand for MUL.

Force-directed Scheduling - Algorithm

- Algorithm

```
FDS(G_s,u(V,E),Lambda_max) {
    repeat {
      Compute time frame for all nodes
      Compute distribution for starting time for all nodes and all mean demands
      Compute total force for each node
      Select node with minimal force and assign the starting time to it.
    } until Starting time has been assigned to all nodes.
  return (t-FDS)
}
```

Summary

Where we are

- HLS – Operations are scheduled to cycles
- We need to generate the RTL code for the accelerator