High Level Synthesis

Shankar Balachandran
Assistant Professor, Dept. of CSE
IIT Madras
shankar@cse.iitm.ac.in

References and Copyright

- Textbooks referred (none required)
  - [Mic94] G. De Micheli
    "Synthesis and Optimization of Digital Circuits"
    McGraw-Hill, 1994.

- Slides used:
  - Giovanni de Micheli's Slides on Synthesis
  - Kia Bazargan's Material on High Level Synthesis
  - [©Gupta] © Rajesh Gupta
    UC-Irvine
    http://www.ics.uci.edu/~rgupta/ics280.html

High Level Synthesis (HLS)

- The process of converting a high-level description of a design to a netlist
  - Input:
    - High-level languages (e.g., C)
    - Behavioral hardware description languages (e.g., VHDL)
    - Structural HDLs (e.g., VHDL)
    - State diagrams / logic networks
  - Tools:
    - Parser
    - Library of modules
  - Constraints:
    - Area constraints (e.g., # modules of a certain type)
    - Delay constraints (e.g., set of operations should finish in λ clock cycles)
  - Output:
    - Operation scheduling (time) and binding (resource)
    - Control generation and detailed interconnections

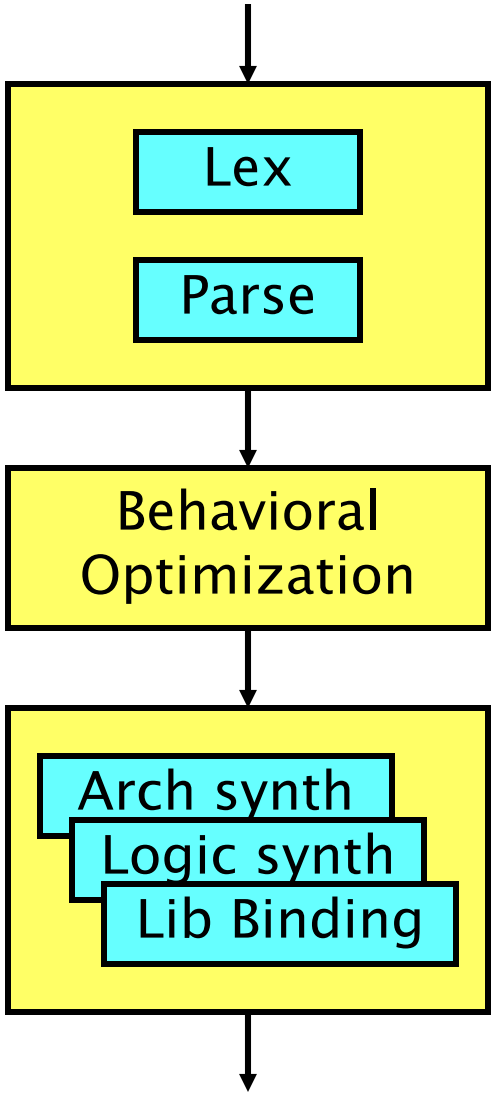
Architectural-level synthesis motivation

- Raise input abstraction level
  - Reduce specification of details
  - Extend designer base
  - Self-documenting design specifications
  - Ease modifications and extensions
- Reduce design time
- Explore and optimize macroscopic structure:
  - Series/parallel execution of operations

Synthesis

- Transform behavioral into structural view
- Architectural-level synthesis:
  - Architectural abstraction level
  - Determine *macroscopic* structure
  - Example: major building blocks
- Logic-level synthesis:
  - Logic abstraction level
  - Determine *microscopic* structure
  - Example: logic gate interconnection

High-Level Synthesis Compilation Flow

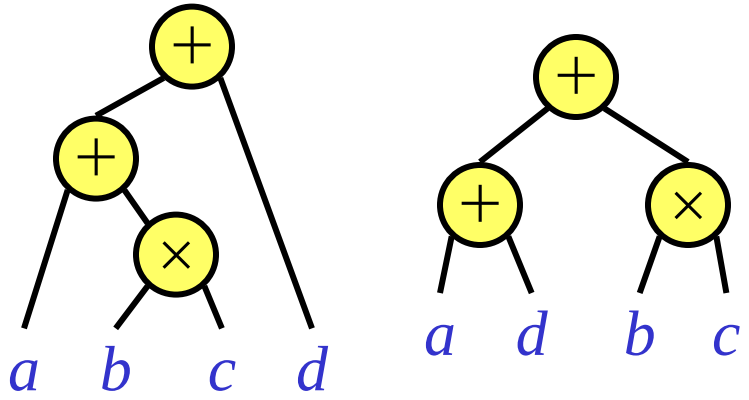


Compilation front-end

Intermediate form

HLS backend

$$x = a + (b \times c) + d$$



Example

```
diffeq {
    read (x; y; u; dx; a);
    repeat
        xl = x+dx;
        ul = u –(3 · x · u · dx) – (3 · y · dx)
        yl = y + u · dx ;
        c = xl < a;
        X = xl; u = ul; y = yl;
    until ( c )
    write (y);
}
```

Compilation and behavioral optimization

- Software compilation:
  - Compile program into intermediate form
  - Optimize intermediate form
  - Generate target code for an architecture
- Hardware compilation:
  - Compile programs/HDL into sequencing graph
  - Optimize sequencing graph
  - Generate gate-level interconnection for a cell library

Behavioral-level optimization

- Semantic-preserving transformations aiming at simplifying the model
- Applied to parse-trees or during their generation
- Taxonomy:
  - *Data-flow* based transformations
  - *Control-flow* based transformations

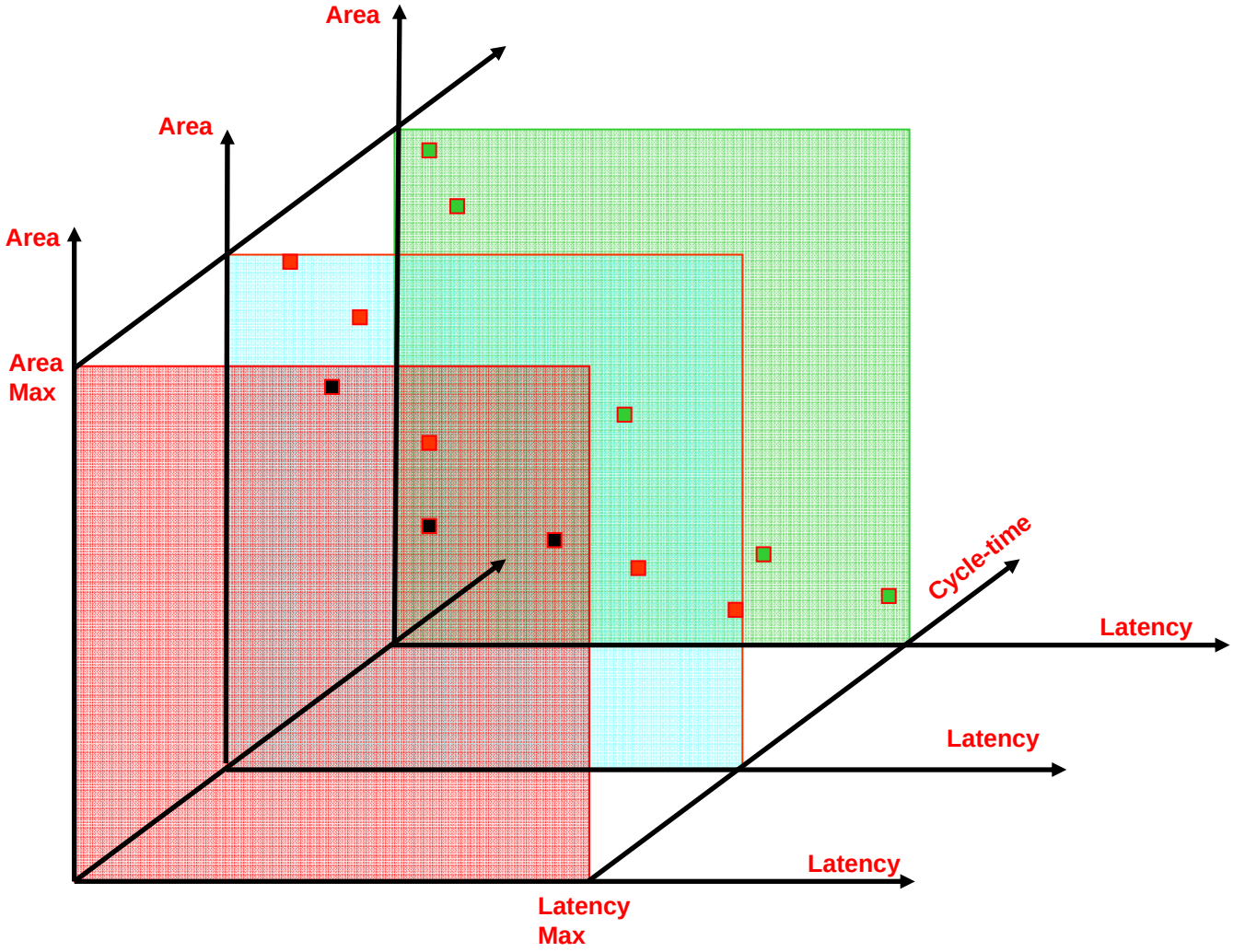
Architectural synthesis and optimization

- Synthesize macroscopic structure in terms of building-blocks
- Explore area/performance trade-off:
  - *maximize performance* of implementations subject to *area* constraints
  - *minimize area implementations* subject to *performance* constraints
- Determine an optimal implementation
- Create logic model for data-path and control

Design space and objectives

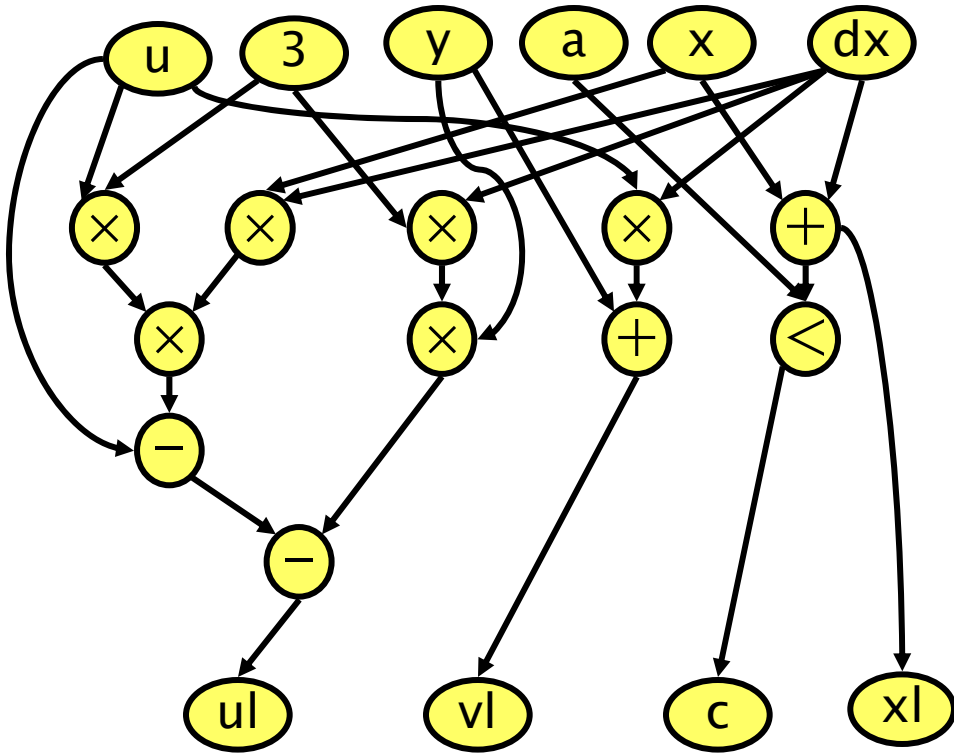
- Design space:
  - Set of all feasible implementations
- Implementation parameters:
  - Area
  - Performance:
    - Cycle-time
    - Latency
    - Throughput (for pipelined implementations)
  - Power consumption

Design evaluation space



Dependency Graph (Sequencing Graph)

```
diffeq {
    read (x; y; u; dx; a);
    repeat
        xl = x+dx;
        ul = u –(3 · x · u · dx) – (3 · y · dx)
        yl = y + u · dx ;
        c = xl < a;
        X = xl; u = ul; y = yl;
    until ( c )
    write (y);
}
```



Hardware modeling

- Circuit behavior:
  - Sequencing graphs
- Building blocks:
  - Resources
- Constraints:
  - Timing and resource usage

Resources

- Functional resources:
  - Perform operations on data
  - Example: arithmetic and logic blocks
- Storage resources:
  - Store data
  - Example: memory and registers
- Interface resources:
  - Example: busses and ports

Resources and circuit families

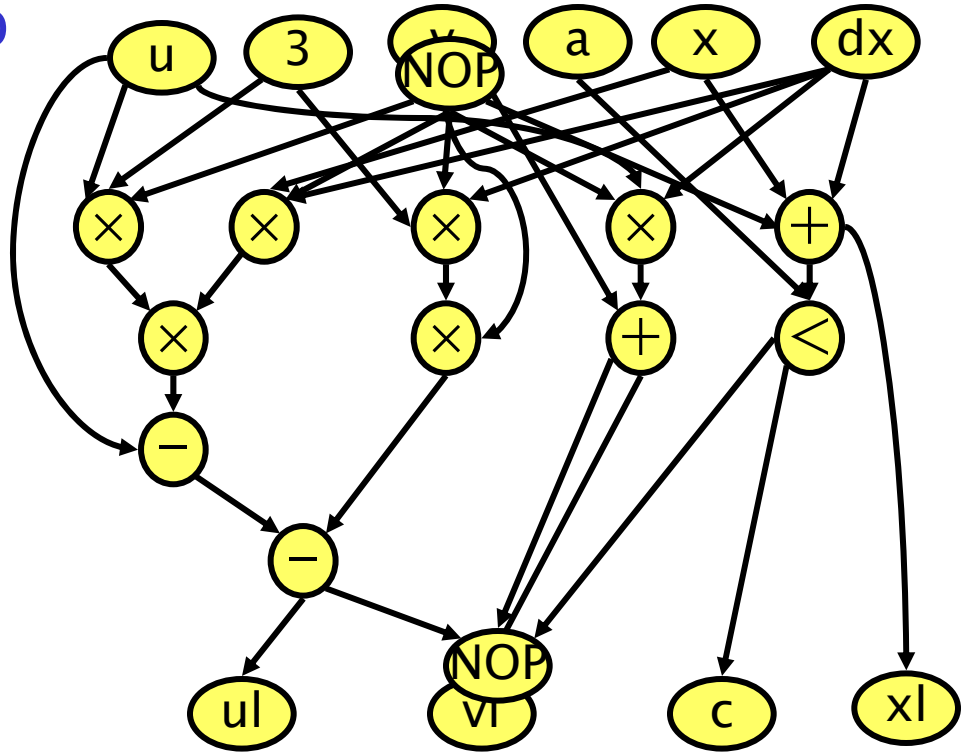
- *Resource-dominated* circuits.
  - Area and performance depend on few, well-characterized blocks
  - The most common in DSP Circuits
- *Non resource-dominated* circuits
  - Area and performance are strongly influenced by sparse logic, control and wiring
  - Example: some ASIC circuits

Implementation constraints

- Timing constraints:
  - Cycle-time
  - Latency of a set of operations
  - Time spacing between operation pairs
- Resource constraints:
  - Resource usage (or allocation)
  - Partial binding

Sequence Graph

- Remove all nodes corresponding to constants (with respect to the loop)
- Remove edges from those constants as well
- Remove nodes that are being written in the loop and the corresponding edges
- Add NOP nodes at both ends to represent reads and writes of the loop
- Such a graph is much simpler to operate on

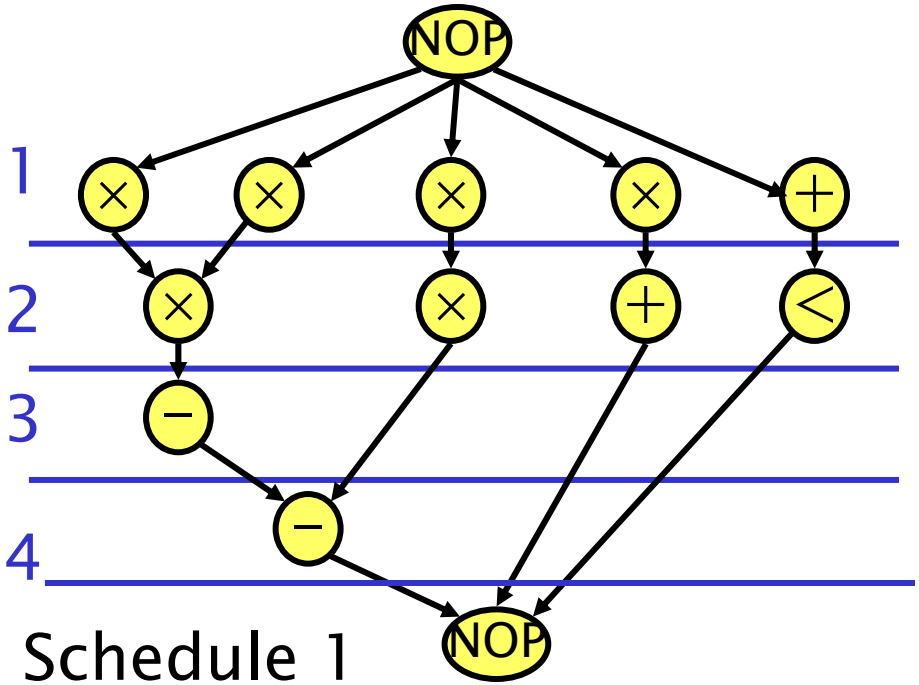


Synthesis in the temporal domain

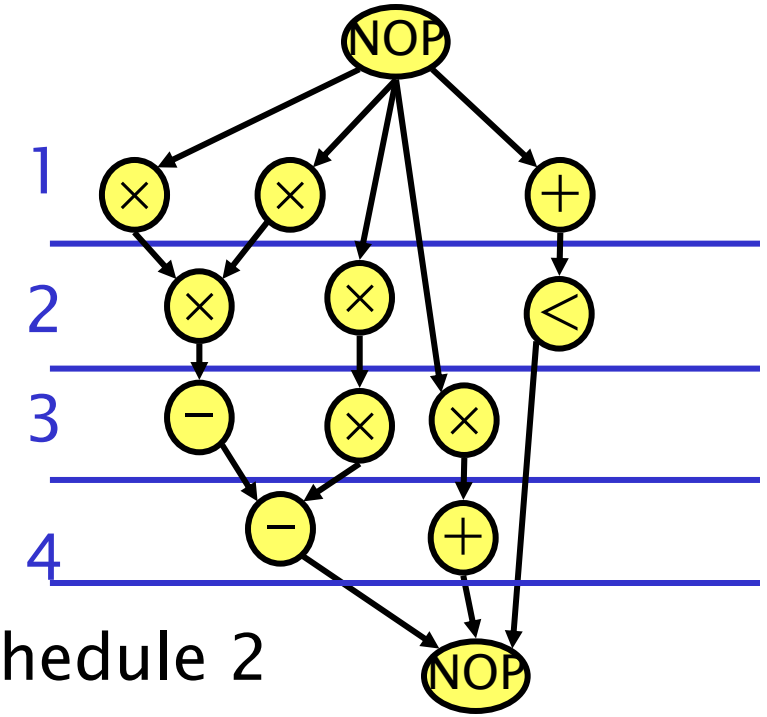
- *Scheduling:*
  - Associate a start-time with each operation
  - Determine latency and parallelism of the implementation
  - The schedule is called ϕ
- *Scheduled sequencing graph:*
  - Sequencing graph with start-time annotation

Synthesis in Temporal Domain

- Schedule:
  - Mapping of operations to time slots (cycles)
  - A scheduled sequencing graph is a labeled graph



Schedule 1



Schedule 2

Operation Types

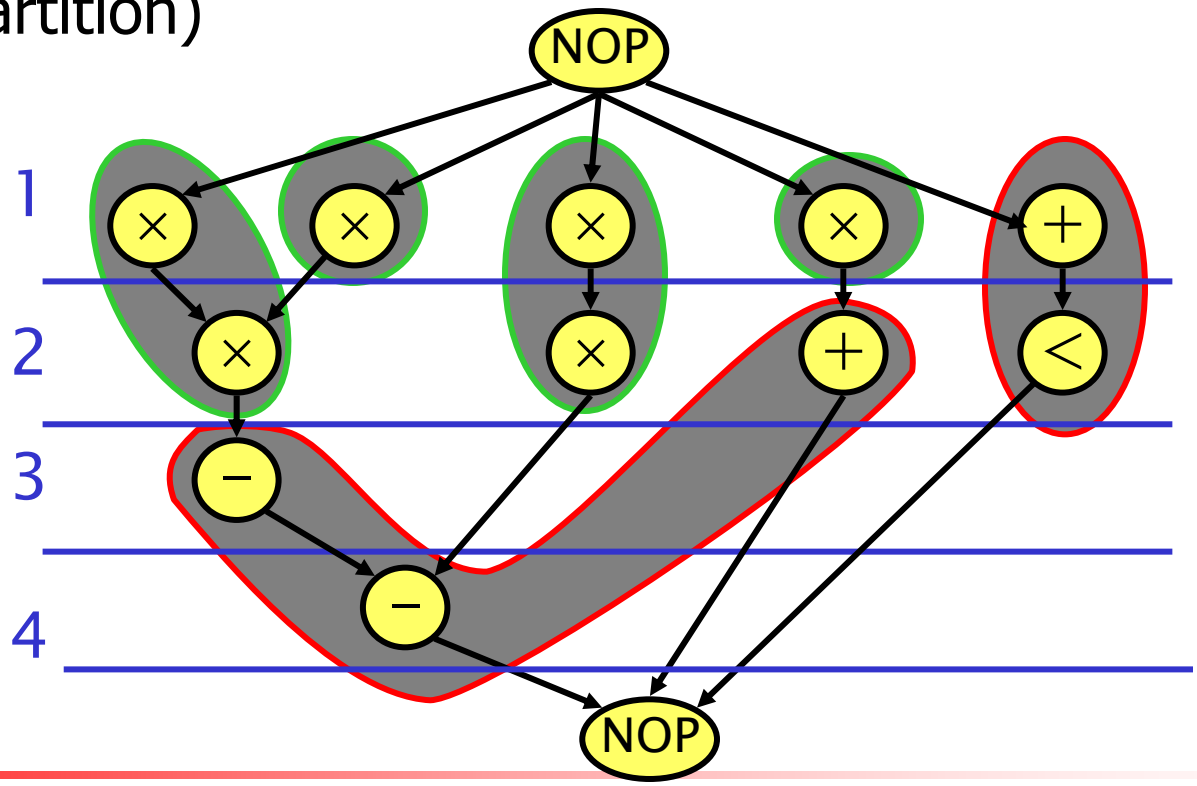
- For each operation, define its *type*.
- For each resource, define a resource type, and a delay (in terms of # cycles)
- T is a relation that maps an operation to a resource type that can implement it
  - $T : V \rightarrow \{1, 2, ..., n_{res}\}$.
- More general case:
  - A resource type may implement more than one operation type (e.g., ALU)
- Resource binding:
  - Map each operation to a resource with the same type
  - Might have multiple options

Synthesis in the spatial domain

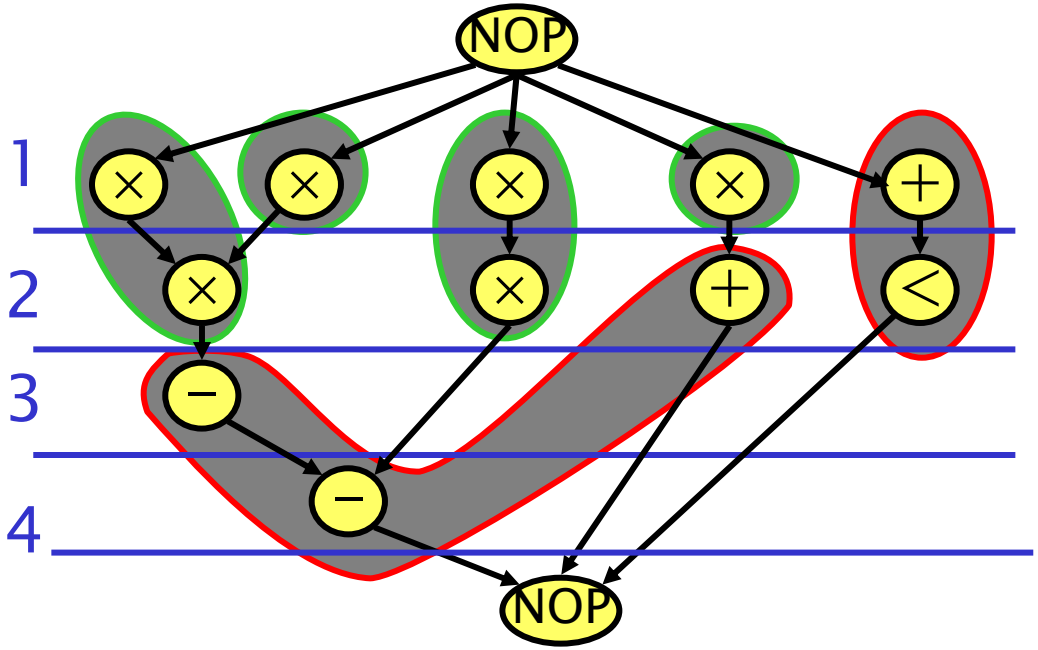
- *Binding:*
  - Associate a resource with each operation with the same type
  - Determine the area of the implementation
- *Sharing:*
  - Bind a resource to more than one operation
  - Operations must not execute concurrently
- *Bound sequencing graph:*
  - Sequencing graph with resource annotation

Schedule in Spatial Domain

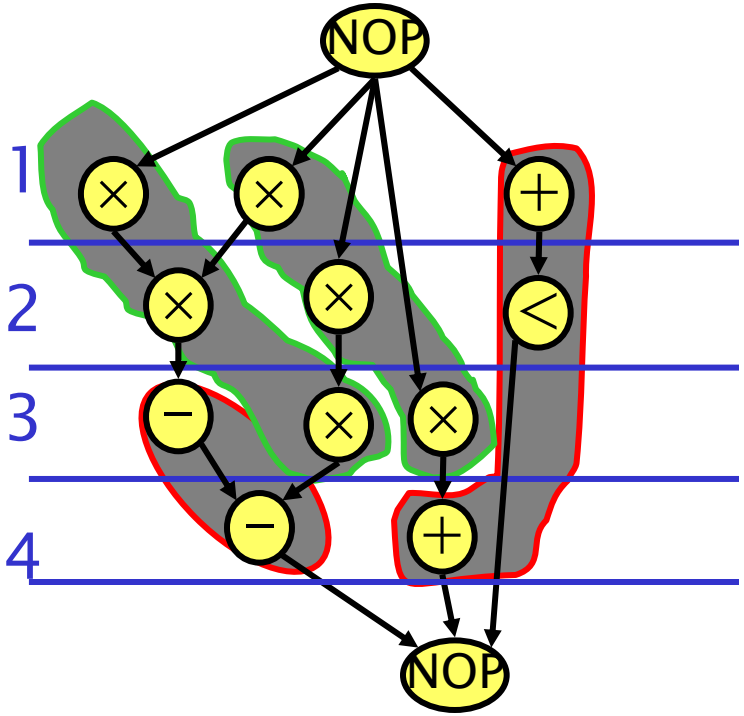
- Resource sharing
  - More than one operation bound to same resource
  - Operations have to be serialized
  - Can be represented using hyperedges (define vertex partition)



Binding Should Change with Schedules



4 Multipliers
2 Adders



2 Multipliers
2 Adders

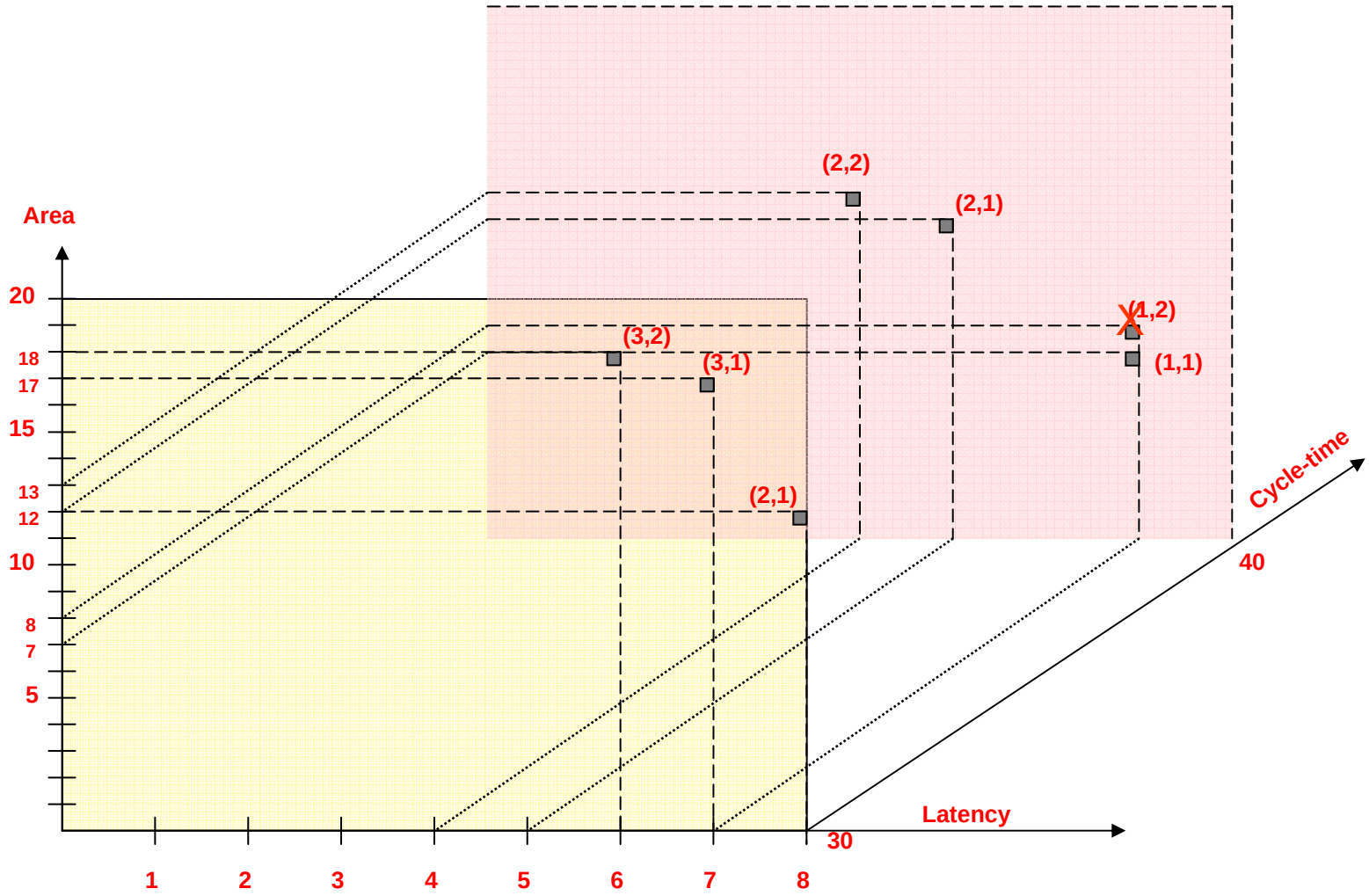
Scheduling and Binding

- Resource constraints:
  - Number of resource instances of each type $\{a_k : k=1, 2, ..., n_{res}\}$.
- Scheduling:
  - Labeled vertices $\phi(v_3)=1$.
- Binding:
  - Hyperedges (or vertex partitions) $\beta(v_2)=adder1$.
- Cost:
  - Number of resources $\approx$ area
  - Registers, steering logic (Muxes, busses), wiring, control unit
- Delay:
  - Start time of the "sink" node
  - Might be affected by steering logic and schedule (control logic) – resource-dominated vs. ctrl-dominated

Architectural Optimization

- Optimization in view of design space flexibility
- A multi-criteria optimization problem:
  - Determine schedule ϕ and binding β.
  - Under area A, latency λ and cycle time τ objectives
- Find non-dominated points in solution space
- Solution space tradeoff curves:
  - Non-linear, discontinuous
  - Area / latency / cycle time (more?)

Area/latency trade-off

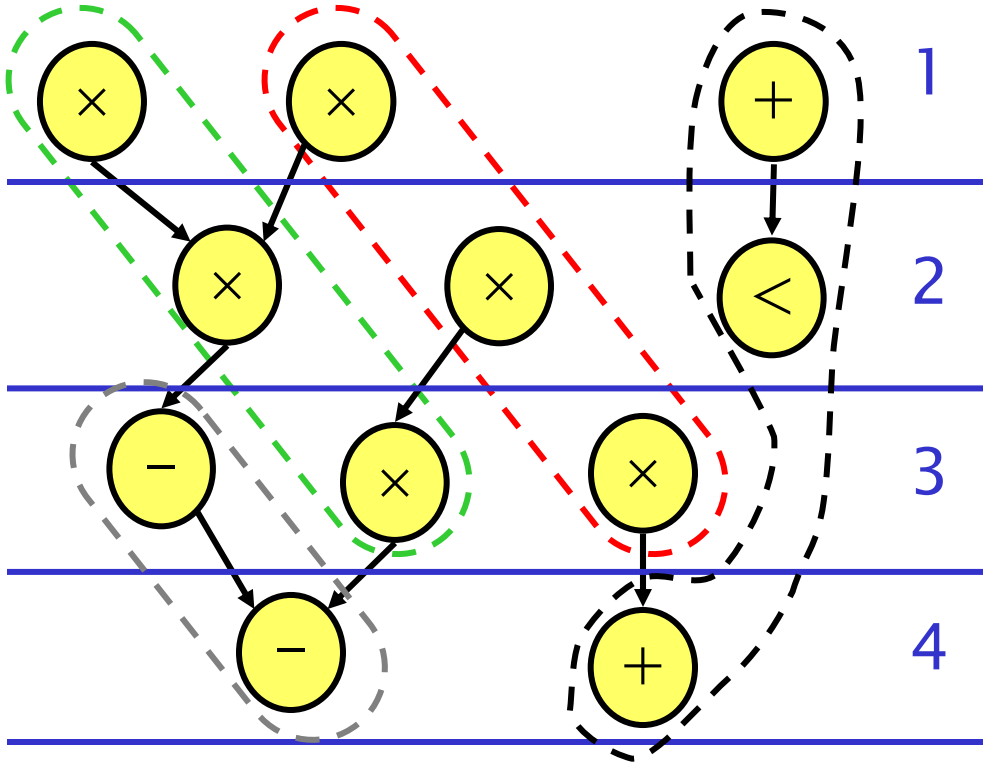


Scheduling and Binding

- Cost λ and A determined by both ϕ and β.
  - Also affected by floorplan and detailed routing
- β affected by ϕ:
  - Resources cannot be shared among concurrent ops
- ϕ affected by β:
  - Resources cannot be shared among concurrent ops
  - When register and steering logic delays added to execution delays, might violate cycle time.
- Order?
  - Apply either one (scheduling, binding) first

How Is the Datapath Implemented?

- In the following schedule and binding, every operation has two inputs. If an input is not shown explicitly, it comes from a unique register



Operation Scheduling

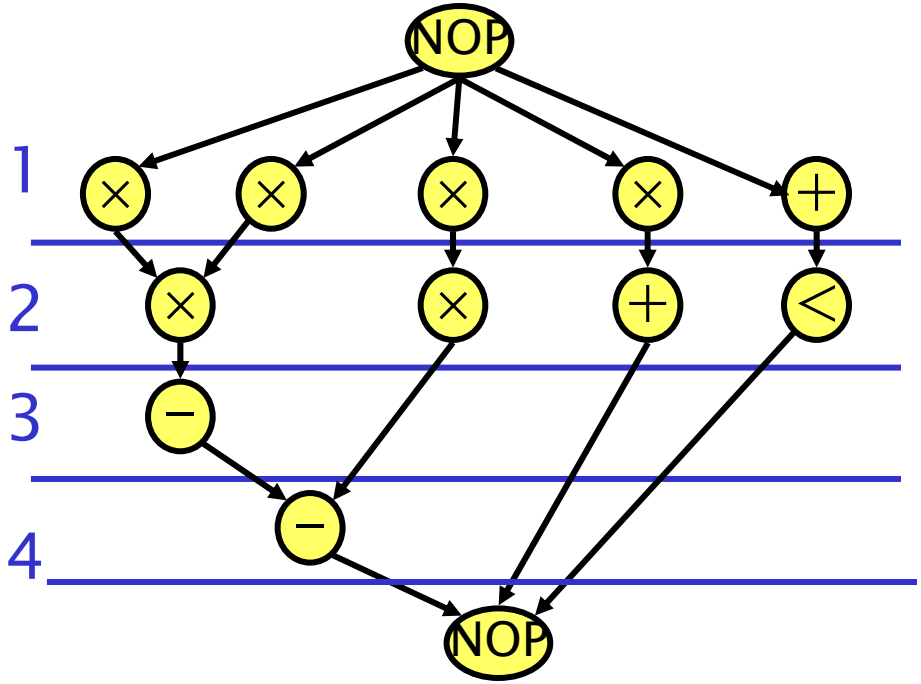
- Input:
  - Sequencing graph G(V, E), with n vertices
  - Cycle time τ.
  - Operation delays $D = \{d_i: i=0..n\}$.
- Output:
  - Schedule ϕ determines start time t_i of operation v_i.
  - Latency $\lambda = t_n - t_0$.
- Goal: determine area / latency tradeoff
- Classes:
  - Non-hierarchical and unconstrained
  - Latency constrained
  - Resource constrained
  - Hierarchical

Min Latency Unconstrained Scheduling

- Simplest case: no constraints, find min latency
- Given set of vertices V, delays D and a partial order > on operations E, find an integer labeling of operations $\phi: V \rightarrow Z^+$ Such that:
  - $t_i = \phi(v_i)$.
  - $t_i \geq t_j + d_j \qquad \forall (v_j, v_i) \in E.$
  - $\lambda = t_n - t_0$ is minimum.
- Solvable in polynomial time
- Bounds on latency for resource constrained problems
- ASAP algorithm used: topological order

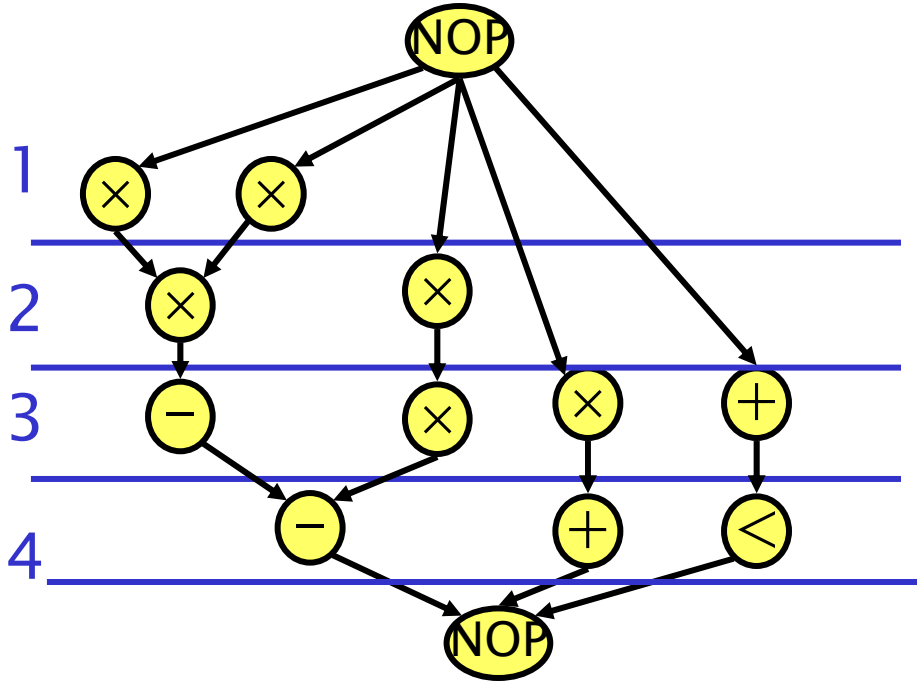
ASAP Schedules

- Schedule v_0 at $t_0=0$.
- While (v_n not scheduled)
  - Select v_i with all scheduled predecessors
  - Schedule v_i at $t_i = \max \{t_j+d_j\}$, v_j being a predecessor of v_i.
- Return t_n.



ALAP Schedules

- Schedule v_n at $t_n=\lambda$.
- While (v_0 not scheduled)
  - Select v_i with all scheduled successors
  - Schedule v_i at t_i = min $\{t_j-d_j\}$, v_j being a succecessor of v_i.

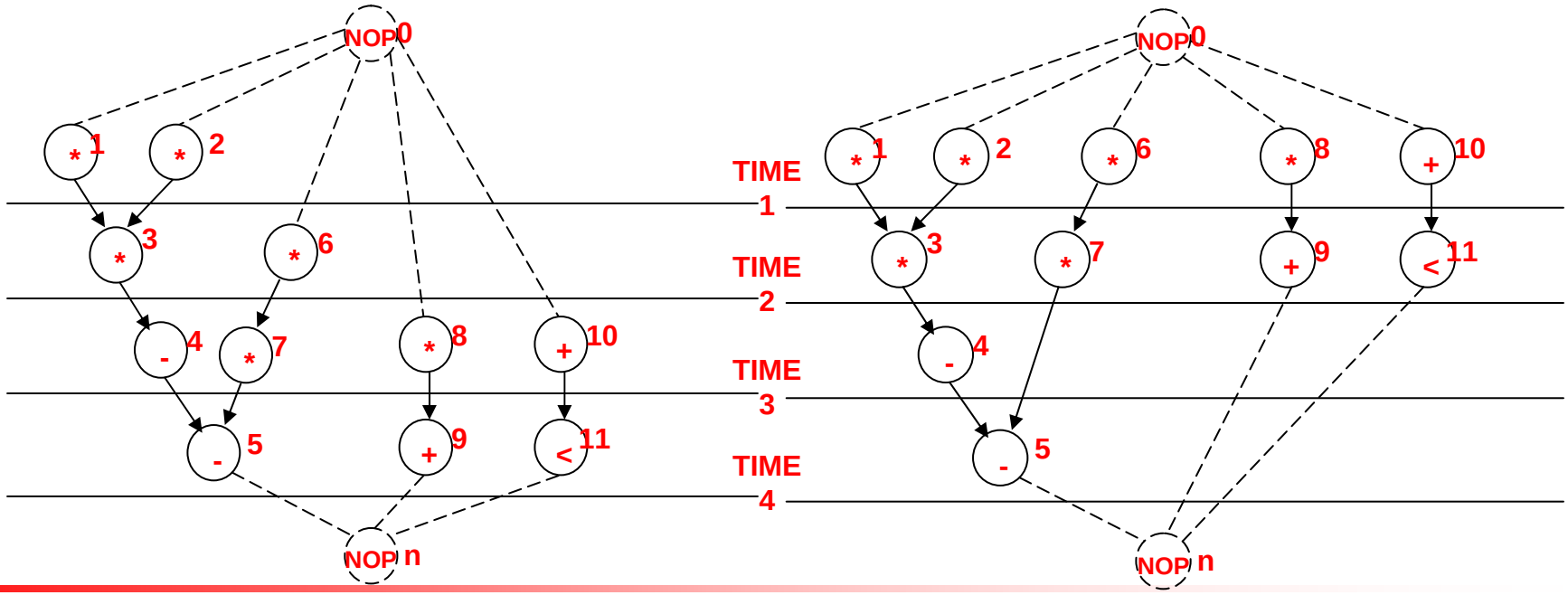


Remarks

- ALAP solves a latency-constrained problem
- Latency bound can be set to latency computed by ASAP algorithm
- Mobility:
  - Defined for each operation
  - Difference between ALAP and ASAP schedule
- Slack on the start time

Example

- Operations with zero mobility:
  - { v_1, v_2, v_3, v_4, v_5 }
  - Critical path
- Operations with mobility one: { v_6, v_7 }
- Operations with mobility two: { v_8, v_9, v_{10}, v_{11} }



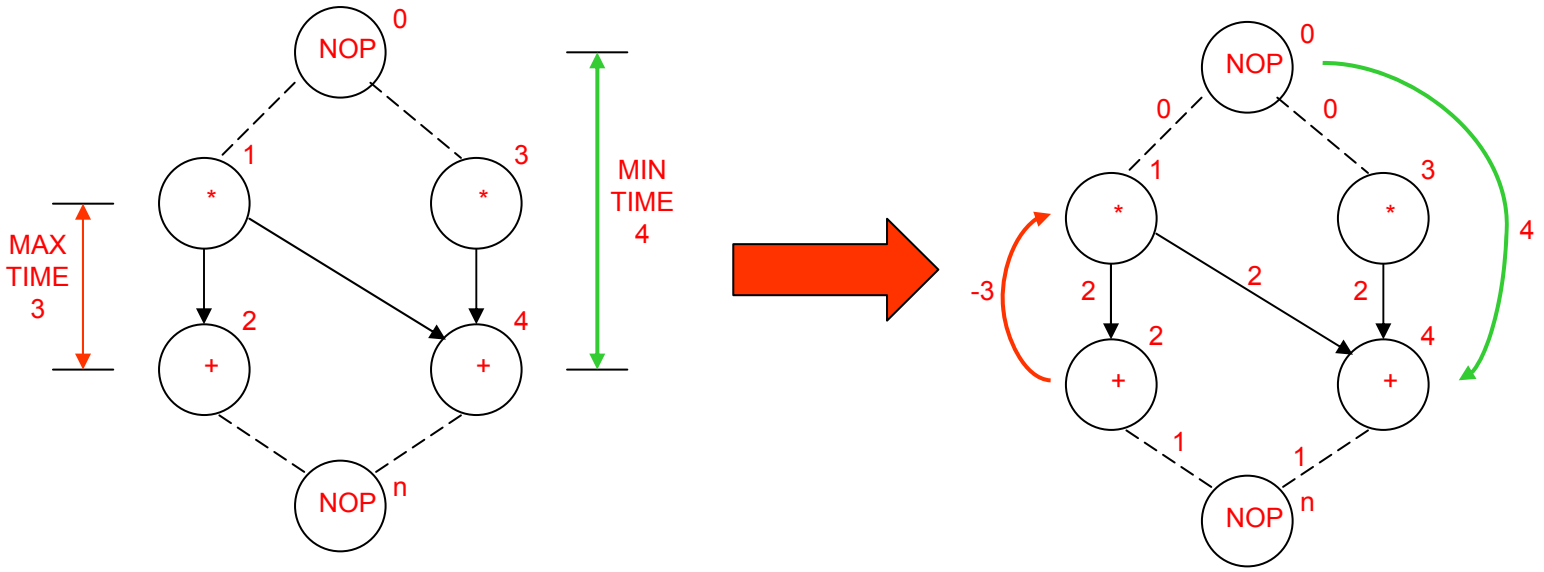
Scheduling under Relative Timing constraints

- Motivation:
  - Deadlines and Release times are absolute
  - Also makes sense to have relative constraints
    - Eg: Memory fetch must be done within 6 cycles and takes a minimum of 2 cycles
- Constraints:
  - Upper/lower bounds on start-time difference of any operation pair
  - A minimum timing constraint $l_{ij} \geq 0$ for specified i,j pairs
  - A maximum timing constraint $u_{ij} \geq 0$ for specified i,j pairs
- Feasibility of a solution

Constraint graph model

- Start from sequencing graph
  - Model delays as weights on edges
- Add forward edges for *minimum* constraints:
  - Edge (v_i, v_j) with weight $l_{ij} \rightarrow t_j \geq t_i + l_{ij}$
- Add backward edges for maximum constraints:
  - That is, for constraint from v_i to v_j
    add backward edge (v_j, v_i) with weight: $-u_{ij}$
    - because $t_j \leq t_i + u_{ij} \rightarrow t_i \geq t_j - u_{ij}$

Example



1. Ensure that there are no positive cycles in the graph
2. Find longest paths in the graph between 2 nodes *i* and *j* and use as delay separations

Vertex	Start time
v_0	1
v_1	1
v_2	3
v_3	1
v_4	5
v_n	6

Resource Constraint Scheduling

- Constrained scheduling
  - General case NP-complete
  - Minimize latency given constraints on area or the resources (ML-RCS)
  - Minimize resources subject to bound on latency (MR-LCS)
- Exact solution methods
  - ILP: Integer Linear Programming
  - Hu's heuristic algorithm for identical processors
- Heuristics
  - List scheduling
  - Force-directed scheduling

ILP Formulation of ML-RCS

- Use binary decision variables
  - $i = 0, 1, ..., n$
  - $l = 1, 2, ..., \lambda'+1$ λ' given upper-bound on latency
  - $x_{il} = 1$ if operation i starts at step l, 0 otherwise.
- Set of linear inequalities (constraints), and an objective function (min latency)
- Observations
  - $x_{il} = 0 \quad for \quad l < t_i^S \quad and \quad l > t_i^L$

    $(t_i^S = ASAP(v_i), t_i^L = ALAP(v_i))$
  - $t_i = \sum_l l \,.\, x_{il}$ t_i = start time of op i.
  - $\sum_{m=l-d_i+1}^{l} x_{im} \stackrel{?}{=} 1 \quad \Rightarrow$ is op v_i (still) executing at step l?

[Mic94] p.198

Constraints

- Operations start only once

  $\Sigma\, x_{il} = 1 \quad i = 1, 2, \ldots, n$

- Sequencing relations must be satisfied

  $t_i \geq t_j + d_j \quad \rightarrow \quad t_i - t_j - d_j \geq 0 \quad$ for all $(v_j, v_i) \in E$

- Resource bounds must be satisfied

  Simple case (unit delay)

  $\sum_{i:T(v_i)=k} \sum_l x_{il} \leq a_k \quad k = 1,2,\ldots n_{res}$; for all l

- Equation 2 can be rewritten as

  $\Sigma\, l \bullet x_{il} - \Sigma\, l \bullet x_{jl} - d_j \geq 0 \quad$ for all $(v_j, v_i) \in E$

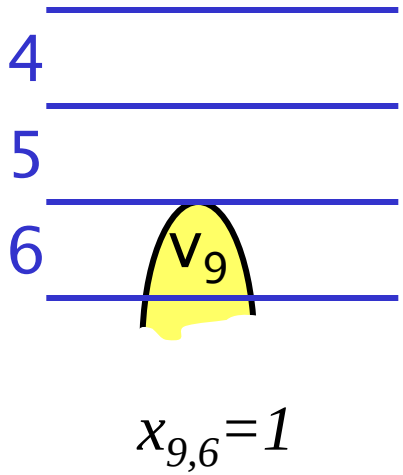
Start Time vs. Execution Time

- For each operation v_i, only one start time
- If $d_i=1$, then the following questions are the same:
  - Does operation v_i **start** at step l?
  - Is operation v_i **running** at step l?
- But if $d_i>1$, then the two questions should be formulated as:
  - Does operation v_i **start** at step l?
    - Does $x_{il} = 1$ hold?
  - Is operation v_i **running** at step l?
    - Does the following hold?

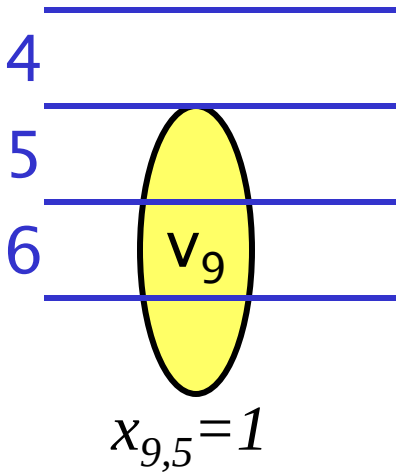
$$\sum_{m=l-d_i+1}^{l} x_{im} \stackrel{?}{=} 1$$

Operation v_i Still Running at Step l ?

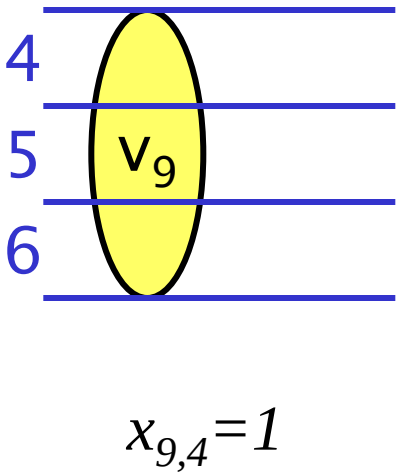
- Is v_9 running at step 6?
  - Is $x_{9,6} + x_{9,5} + x_{9,4} = 1$?



$x_{9,6}=1$



$x_{9,5}=1$

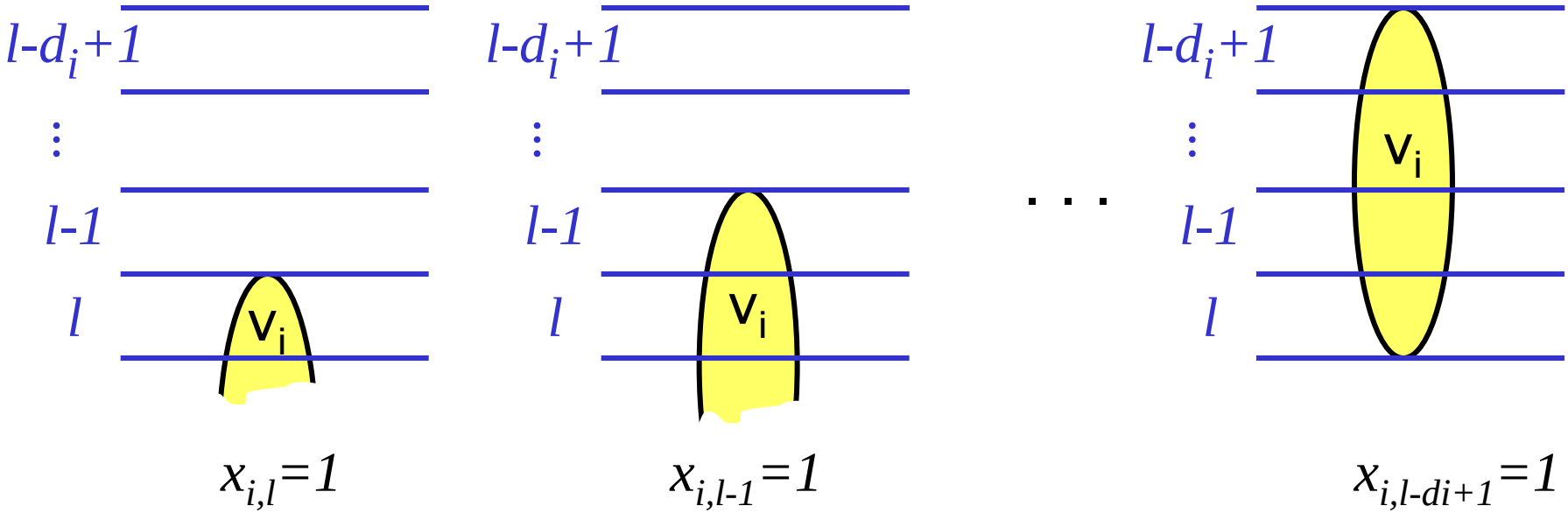


$x_{9,4}=1$

- Note:
  - Only one (if any) of the above three cases can happen
  - To meet resource constraints, we have to ask the same question for ALL steps, and ALL operations of that type

Operation v_i Still Running at Step l ?

- Is v_i running at step l ?
  - Is $x_{i,l} + x_{i,l-1} + \ldots + x_{i,l-di+1} = 1$?



ILP Formulation of ML-RCS (cont.)

- Constraints:
  - Unique start times: $\sum_{l} x_{il} = 1, \quad i = 0,1,\ldots,n$
  - Sequencing (dependency) relations must be satisfied

$$t_i \geq t_j + d_j \;\forall (v_j, v_i) \in E \Rightarrow \sum_{l} l \,.\, x_{il} \geq \sum_{l} l \,.\, x_{jl} + d_j$$

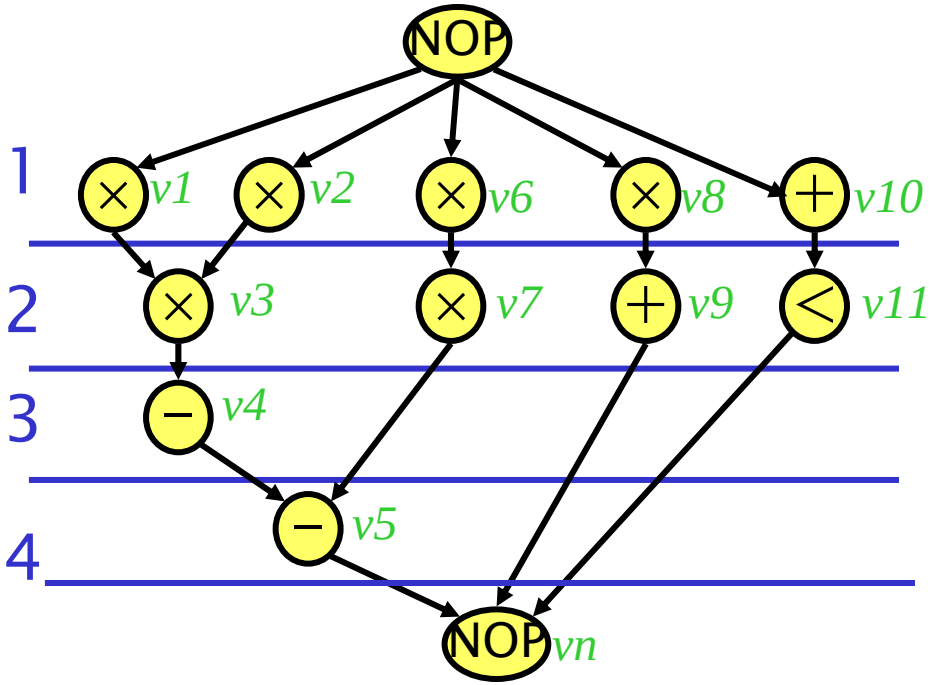
  - Resource constraints

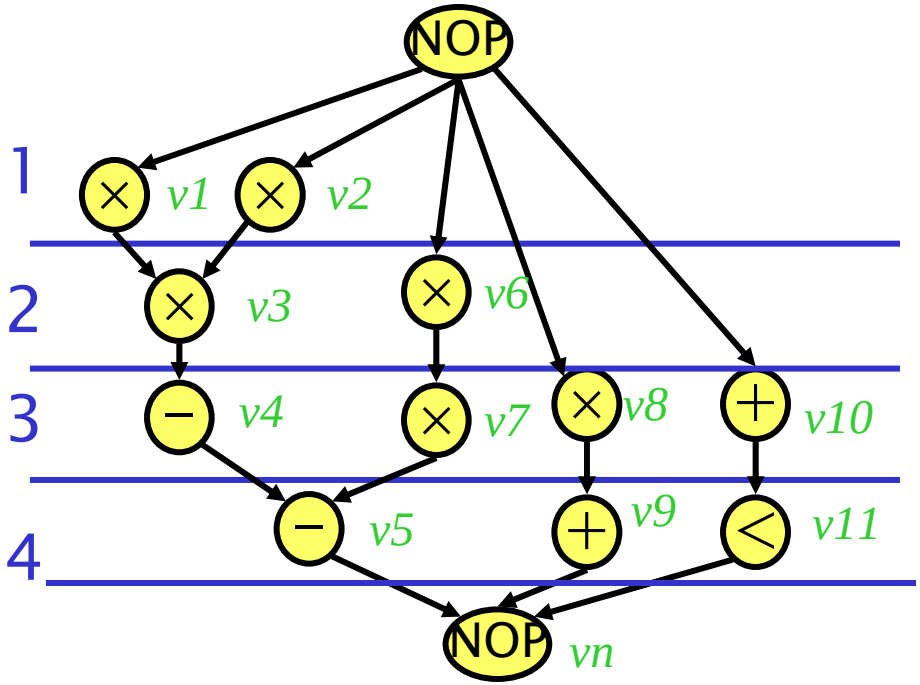
$$\sum_{i:T(v_i)=k} \sum_{m=l-d_i+1}^{l} x_{im} \leq a_k, \quad k = 1,\ldots,n_{res}, \quad l = 1,\ldots,\overline{\lambda} + 1$$

- Objective: min $\boldsymbol{c}^T\boldsymbol{t}$.
  - $\boldsymbol{t}$ =start times vector, $\boldsymbol{c}$ =cost weight (e.g., [0 0 ... 1])
  - When $\boldsymbol{c}$ =[0 0 ... 1], $\boldsymbol{c}^T\boldsymbol{t} = \sum_{l} l \,.\, x_{nl}$

ILP Example

- Assume $\overline{\lambda} = 4$
- First, perform ASAP and ALAP
  - (we can write the ILP without ASAP and ALAP, but using ASAP and ALAP will simplify the inequalities)





ILP Example: Unique Start Times Constraint

- Without using ASAP and ALAP values:

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1$$

$$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1$$

...

...

...

$$x_{11,1} + x_{11,2} + x_{11,3} + x_{11,4} = 1$$

- Using ASAP and ALAP:

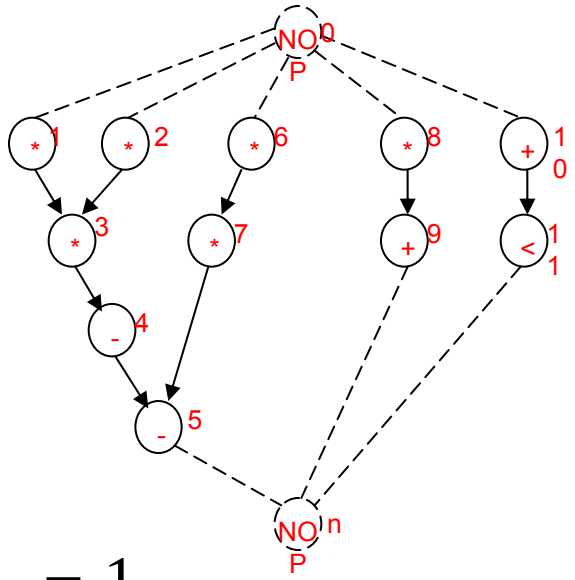
$$x_{1,1} = 1$$

$$x_{2,1} = 1$$

$$x_{3,2} = 1$$

$$x_{4,3} = 1$$

$$x_{5,4} = 1$$



$$x_{6,1} + x_{6,2} = 1$$

$$x_{7,2} + x_{7,3} = 1$$

$$x_{8,1} + x_{8,2} + x_{8,3} = 1$$

$$x_{9,2} + x_{9,3} + x_{9,4} = 1$$

....

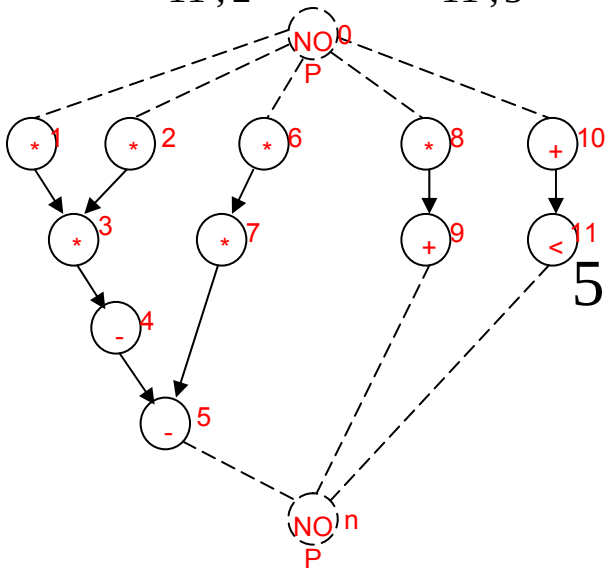
ILP Example: Dependency Constraints

- Using ASAP and ALAP, the non-trivial inequalities are: (assuming unit delay for + and *)

$$2.x_{7,2} + 3.x_{7,3} - x_{6,1} - 2.x_{6,2} - 1 \geq 0$$

$$2.x_{9,2} + 3.x_{9,3} + 4.x_{9,4} - x_{8,1} - 2.x_{8,2} - 3.x_{8,3} - 1 \geq 0$$

$$2.x_{11,2} + 3.x_{11,3} + 4.x_{11,4} - x_{10,1} - 2.x_{10,2} - 3.x_{10,3} - 1 \geq 0$$



$$4.x_{5,4} - 2.x_{7,2} - 3.x_{7,3} - 1 \geq 0$$

$$5.x_{n,5} - 2.x_{9,2} - 3.x_{9,3} - 4.x_{9,4} - 1 \geq 0$$

$$5.x_{n,5} - 2.x_{11,2} - 3.x_{11,3} - 4.x_{11,4} - 1 \geq 0$$

ILP Example: Resource Constraints

- Resource constraints (assuming 2 adders and 2 multipliers)

$$x_{1,1} + x_{2,1} + x_{6,1} + x_{8,1} \leq 2$$

$$x_{3,2} + x_{6,2} + x_{7,2} + x_{8,2} \leq 2$$

$$x_{7,3} + x_{8,3} \leq 2$$

$$x_{10,1} \leq 2$$

$$x_{9,2} + x_{10,2} + x_{11,2} \leq 2$$

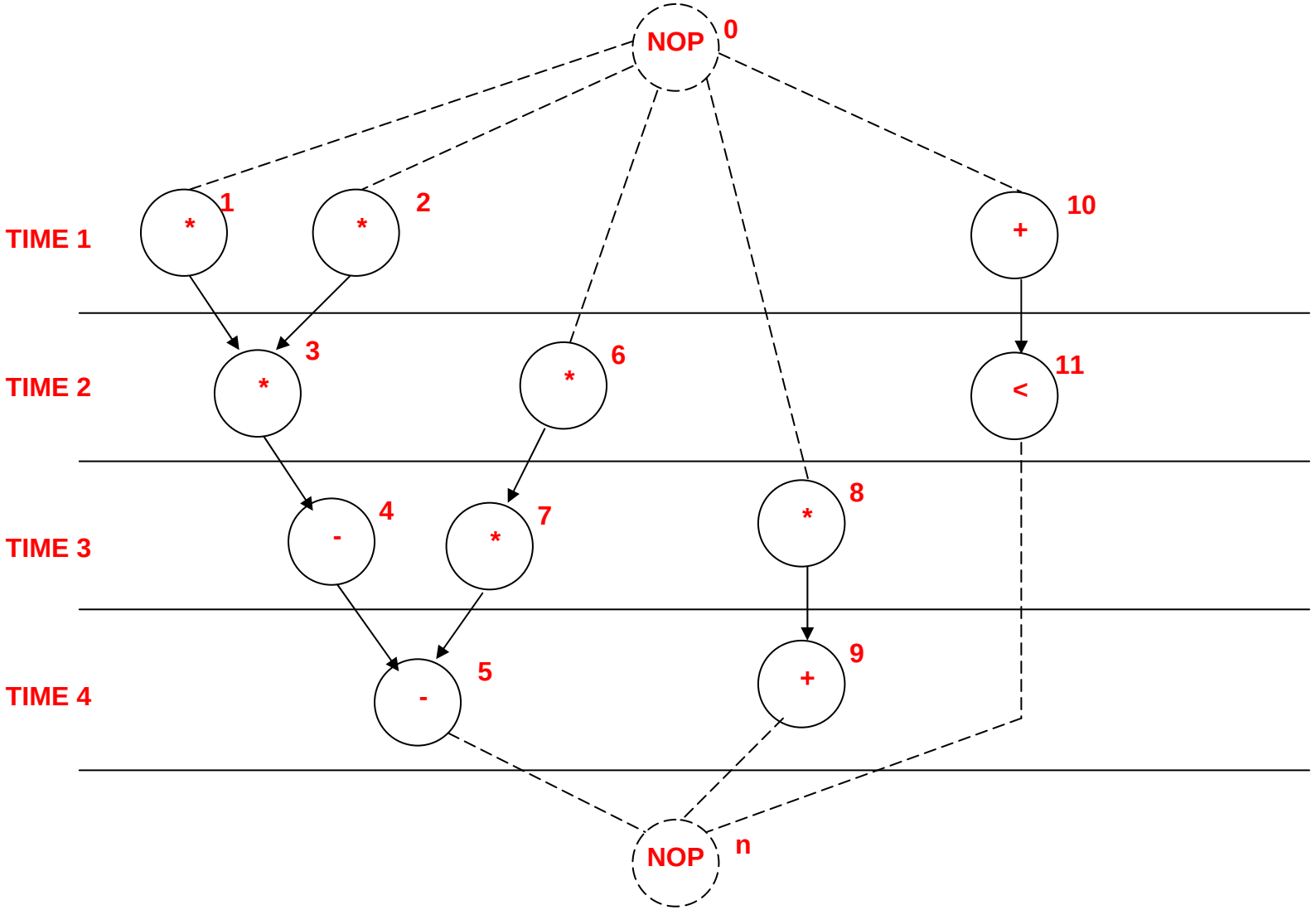
$$x_{4,3} + x_{9,3} + x_{10,3} + x_{11,3} \leq 2$$

$$x_{5,4} + x_{9,4} + x_{11,4} \leq 2$$

- Objective:
  - Since λ=4 and sink has no mobility, any feasible solution is optimum, but we can use the following anyway:

$$Min \quad x_{n,1} + 2.x_{n,2} + 3.x_{n,3} + 4.x_{n,4}$$

Example



Note that the schedule is different from both ALAP and ASAP schedules.

ILP Formulation of MR-LCS

- Dual problem to ML-RCS
- Objective:
  - Goal is to optimize total resource usage, $\mathbf{a}$.
  - Objective function is $\boldsymbol{c}^T\boldsymbol{a}$, where entries in $\boldsymbol{c}$ are respective area costs of resources
- Constraints:
  - Same as ML-RCS constraints, plus:
  - Latency constraint added:

$$\sum_l l \,.\, x_{nl} \leq \overline{\lambda} + 1$$

  - Note: unknown a_k appears in constraints.
- Resource usage is unknown in the constraints
- Resource usage is the objective to minimize

ILP Solution

- Use standard ILP packages
- Transform into LP problem
- Advantages:
  - Exact method
  - Others constraints can be incorporated
- Disadvantages:
  - Works well only up to few thousand variables

Hu's Algorithm

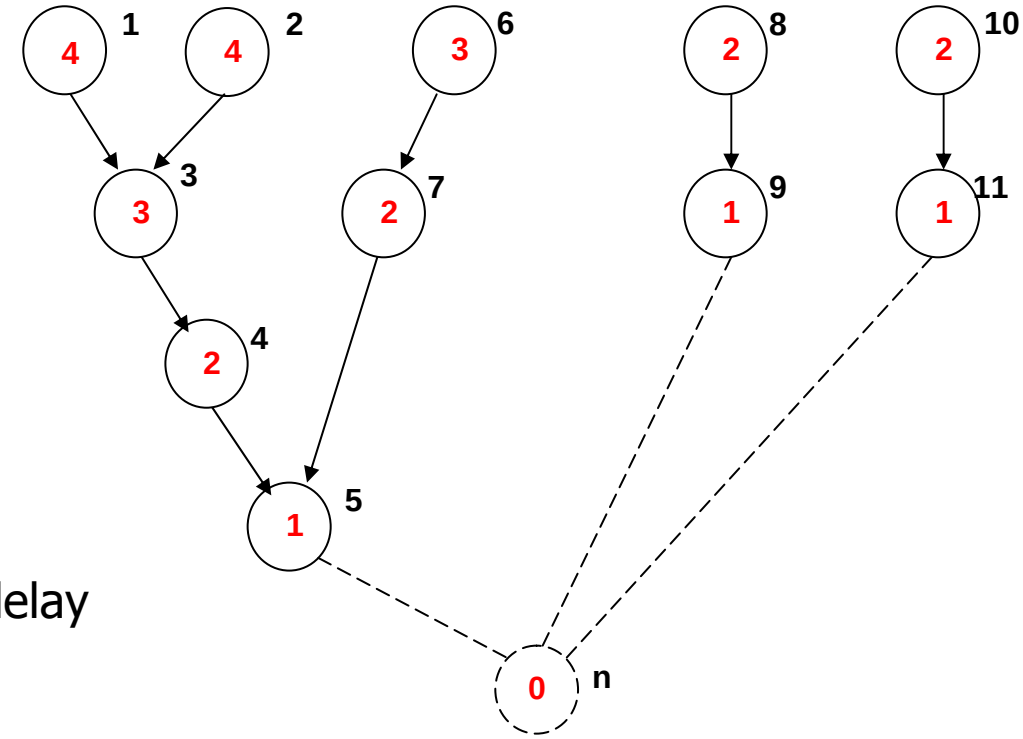
- Simple case of the scheduling problem
  - Operations of unit delay
  - Operations (and resources) of the same type
- Hu's algorithm
  - Greedy
  - Polynomial AND optimal
  - Computes lower bound on number of resources for a given latency
    OR: computes lower bound on latency subject to resource constraints
- Basic idea:
  - Label operations based on their distances from the sink
  - Try to schedule nodes with higher labels first
    (i.e., most "critical" operations have priority)

Hu's algorithm

- Assumptions:
  - Graph is a forest
  - All operations have unit delay
  - All operations have the same type
- Algorithm:
  - Greedy strategy
  - Exact solution

Example

- Assumptions:
  - One resource type only
  - All operations have unit delay
- Labels:
  - Distance to sink

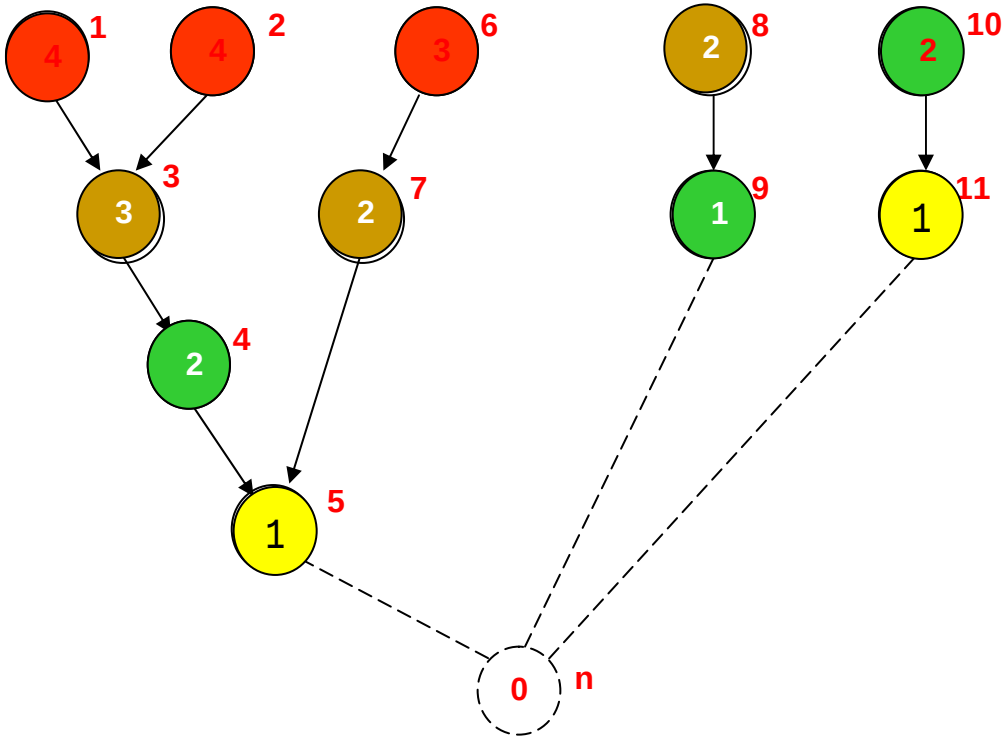


Hu's Algorithm

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

Example

$\bar{a} = 3$



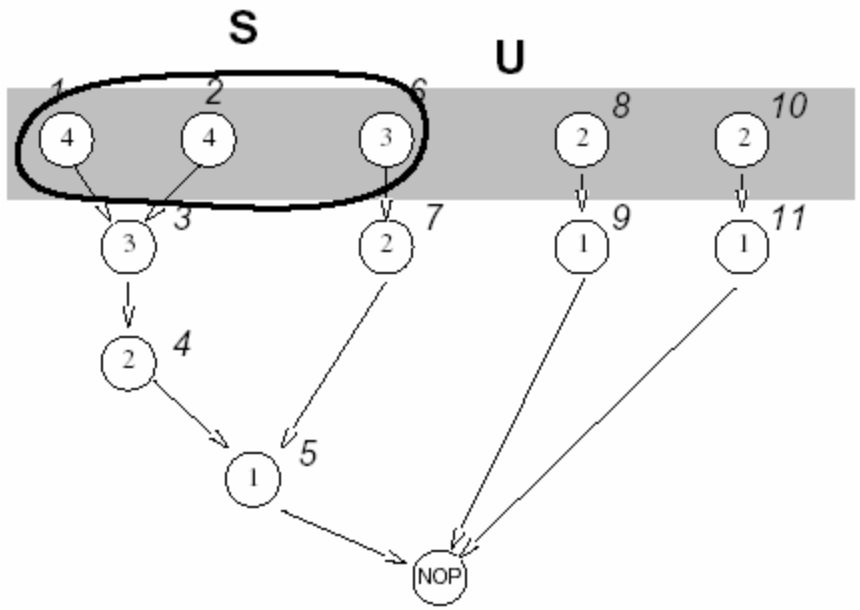
Step 1: Op 1,2,6

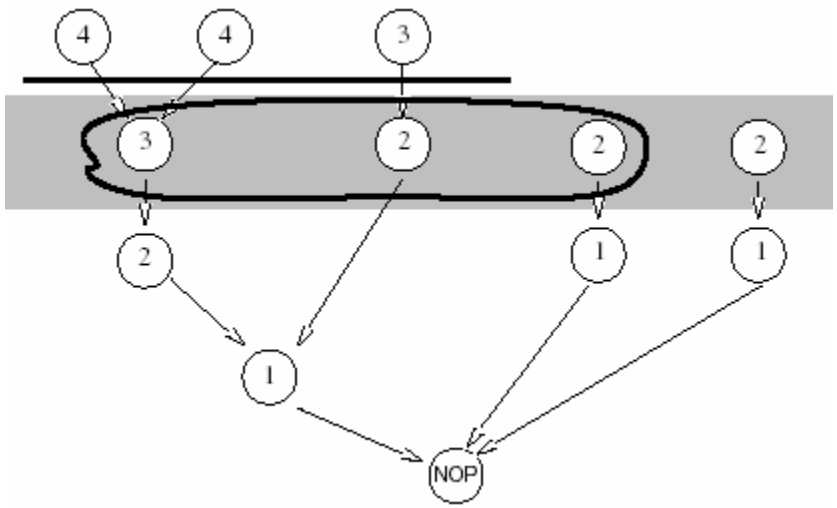
Step 2: Op 3,7,8

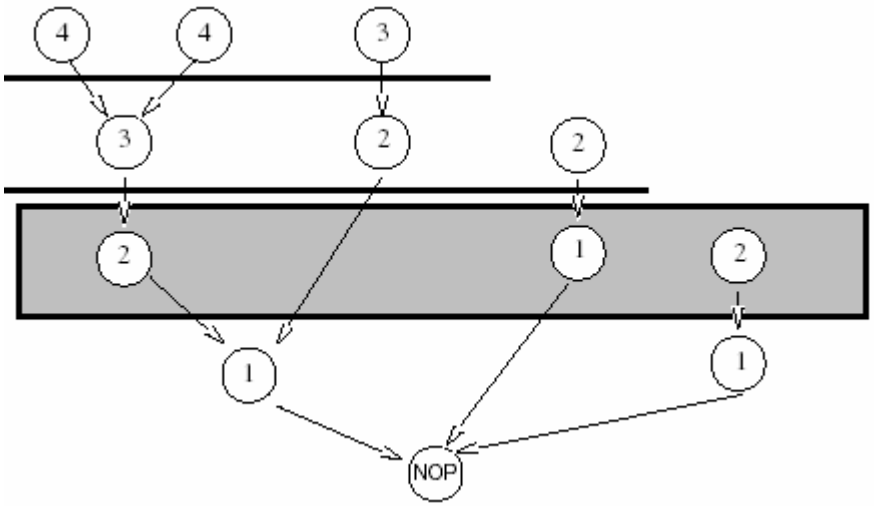
Step 3: Op 4,9,10

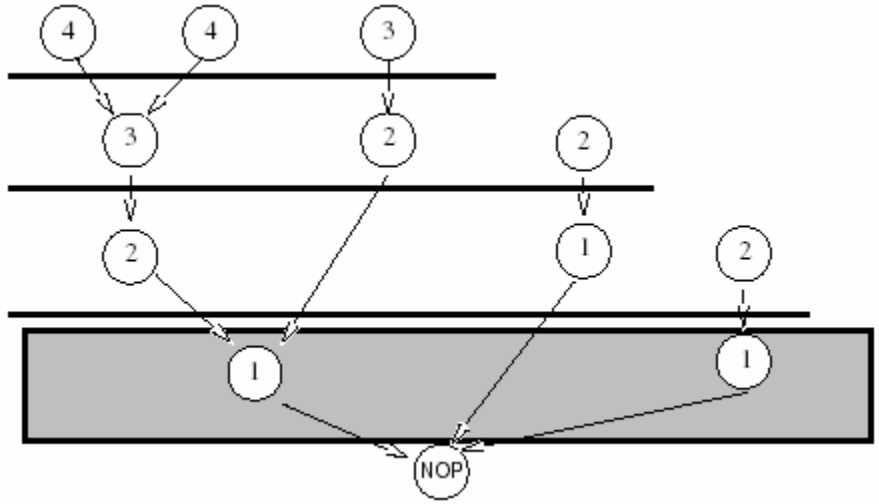
Step 4: Op 5,11

Hu's Algorithm: Example (a=3)









List Scheduling

- Greedy algorithm for ML-RCS and MR-LCS
  - Does NOT guarantee optimum solution
- Similar to Hu's algorithm
  - Operation selection decided by criticality
  - O(n) time complexity
- More general input
  - Resource constraints on different resource types

List Scheduling Algorithm: ML-RCS

LIST_L (G(V,E), **a**) {

l = 1

repeat {

for each resource type k {

$U_{l,k}$ = available vertices in V.

$T_{l,k}$ = operations in progress.

Select $S_k \subseteq U_{l,k}$ such that $|S_k| + |T_{l,k}| \leq a_k$

Schedule the S_k operations at step l

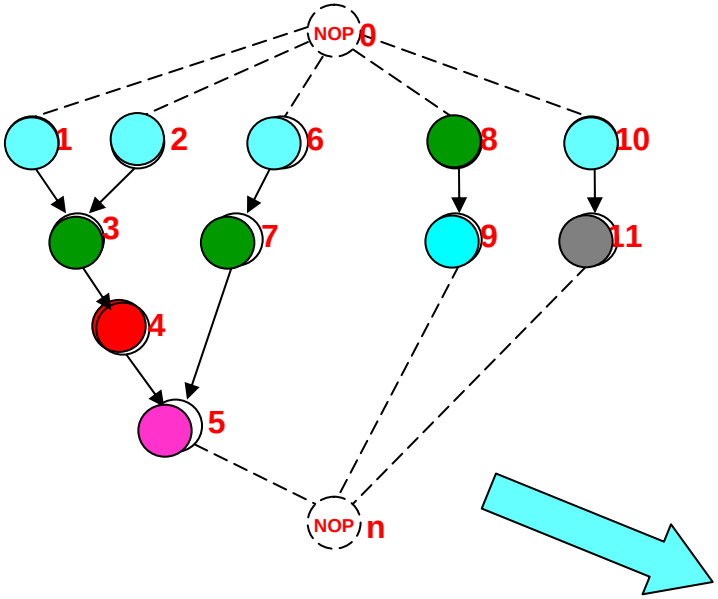
}

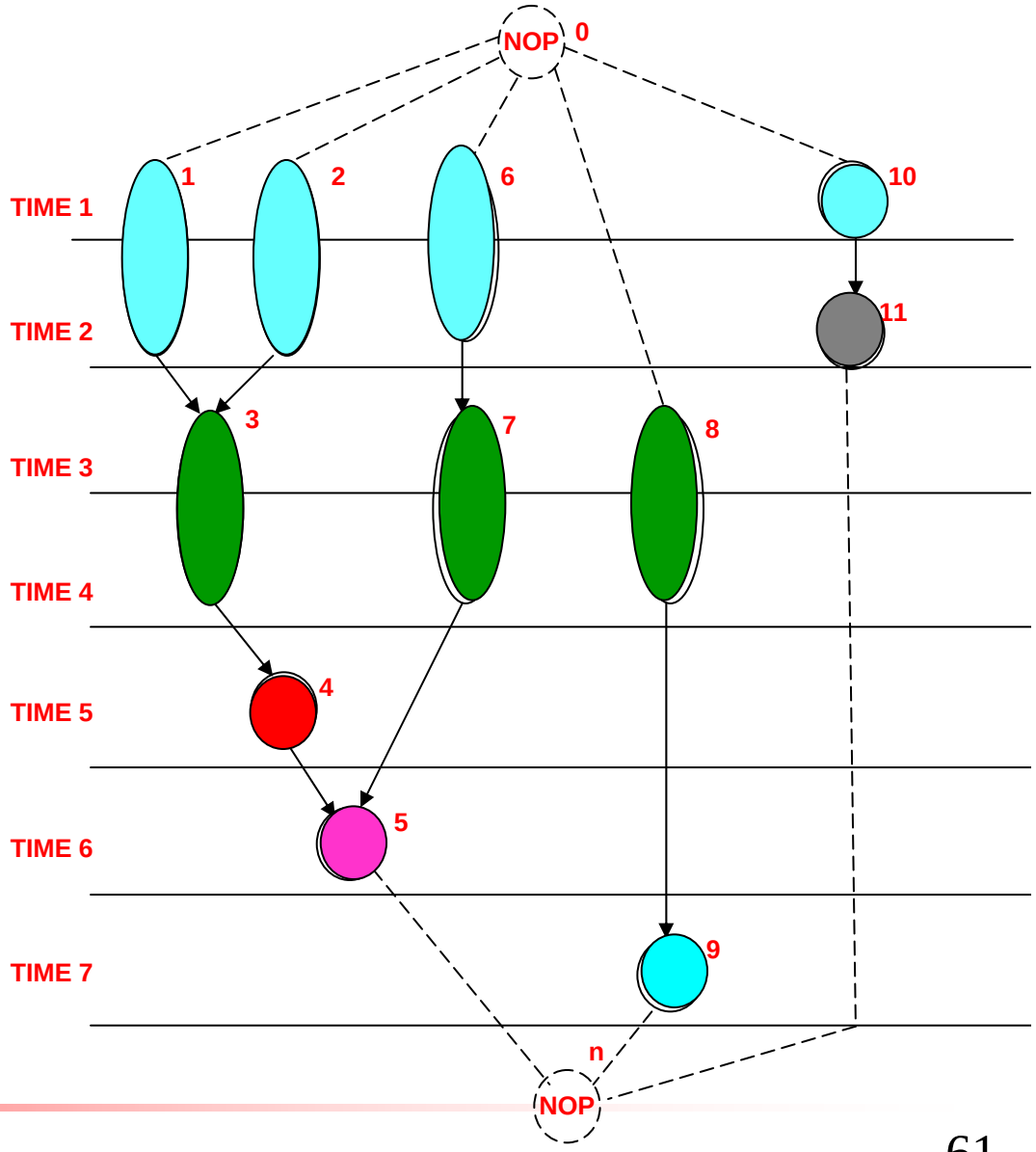
$l = l + 1$

} until v_n is scheduled.

}

Example





Resource bounds:

- **3** multipliers with delay **2**
- **1** ALU with delay **1**

List Scheduling Algorithm: MR-LCS

LIST_R (G(V,E), λ') {

$\boldsymbol{a}$ = **1,** $l = 1$

Compute the ALAP times $\boldsymbol{t}^L$.

if $\boldsymbol{t}_0^L < 0$

return (not feasible)

repeat {

for each resource type k {

$U_{l,k}$ = available vertices in V.

Compute the slacks $\{ s_i = \boldsymbol{t}_i^L - l, \ \forall v_i \in U_{l,k} \}$.

Schedule operations with zero slack, update $\boldsymbol{a}$

Schedule additional $S_k \subseteq U_{l,k}$ under $\boldsymbol{a}$ constraints

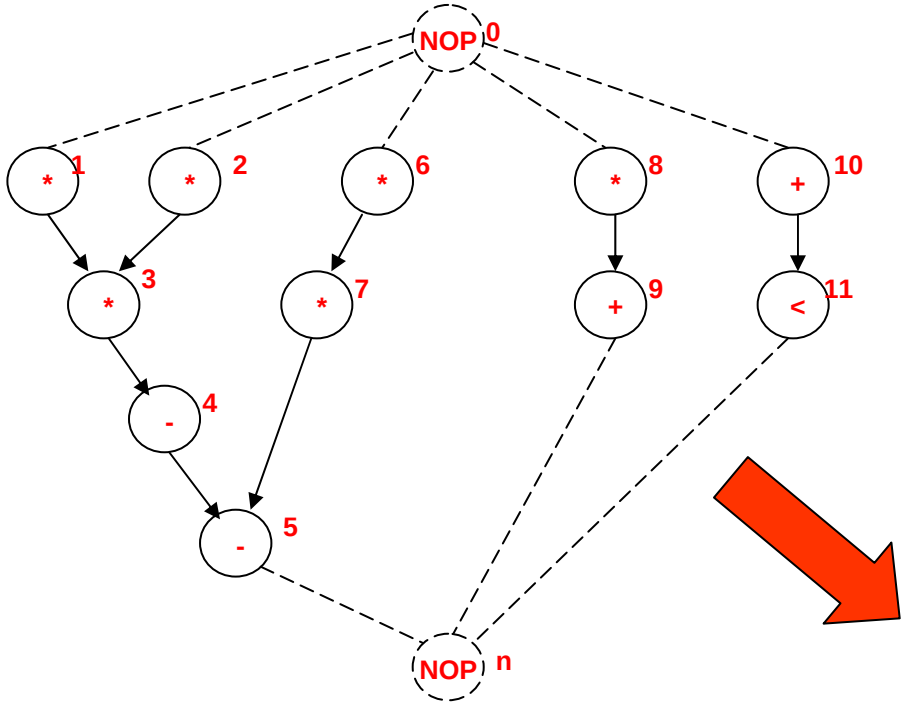
}

$l = l + 1$

} until v_n is scheduled.

}

Example

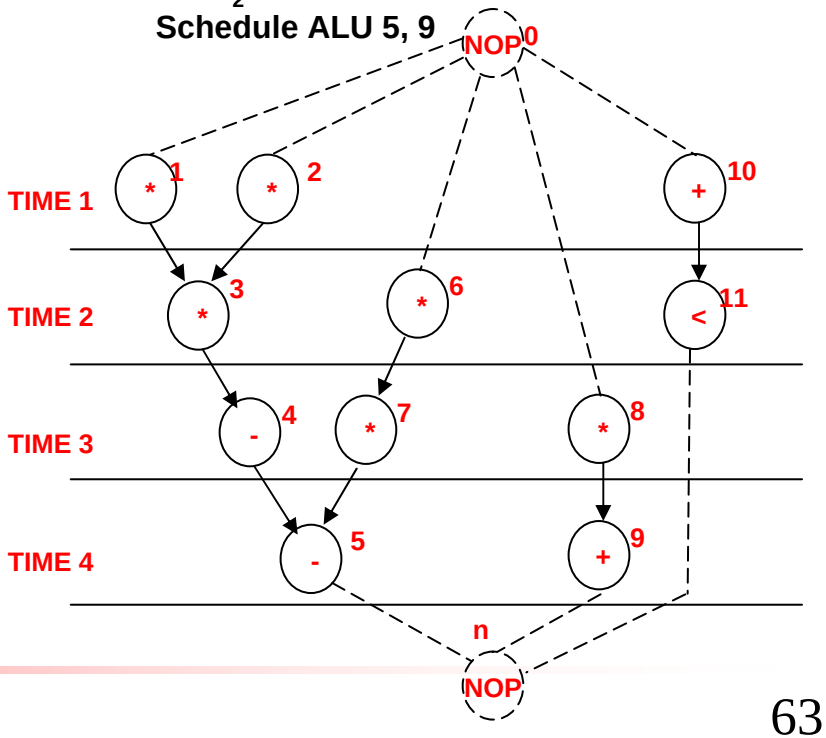


Step 1
Two multiplications on CP
Set a_1 = 2
Schedule Mult 1,2
Schedule ALU 10

Step 2
Schedule Mult 3, 6
Schedule ALU 11

Step 3
Schedule Mult 7,8
Schedule ALU 4

Step 4
Set a_2=2
Schedule ALU 5, 9



Assumptions

Unit-delay resources

Maximum latency = 4

Start with :

a_1 = 1 multiplier

a_2 = 1 ALUs

Summary

- Scheduling algorithms are used by tools
  - Compilers use them when you write code for DSP processors
  - Tools like Xilinx ISE, Synopsys DC etc. use them when you compiler HDL models
- Good understanding of "under the hood" operations of tools is useful
- The constraint solving techniques can be used directly for your custom designs
  - Eg: In DSP software, if you know the resources, you write assembly code to minimize latency

Force-Directed Scheduling

- Similar to list scheduling
  - Can handle ML-RCS and MR-LCS
  - For ML-RCS, schedules step-by-step
  - BUT, selection of the operations tries to find the *globally* best set of operations
- Idea:
  - Find the mobility $\mu_i = t_i^L - t_i^S$ of operations
  - Look at the operation type probability distributions
  - Try to flatten the operation type distributions
- Definition: operation probability density
  - $p_i(l)$ = Pr { v_i starts at step l }.
  - Assume uniform distribution:

$$p_i(l) = \frac{1}{\mu_i + 1} \quad for\ l \in [t_i^S, t_i^L]$$

Force-Directed Scheduling: Definitions

- Operation-type distribution (NOT normalized to 1)
  - $q_k(l) = \sum_{i:T(v_i)=k} p_i(l)$
- Operation probabilities over control steps:
  - $p_i = \{p_i(0), p_i(1), \mathrm{K}\ , p_i(n)\}$
- Distribution graph of type k over all steps:
  - $\{q_k(0), q_k(1), \mathrm{K}\ , q_k(n)\}$
  - $q_k(l)$ can be thought of as *expected* operator cost for implementing operations of type k at step l.

Example

$$q_{add}(1) = \frac{1}{3} = 0.33$$

$$q_{add}(2) = \frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 1$$

$$q_{add}(3) = 1 + \frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 2$$

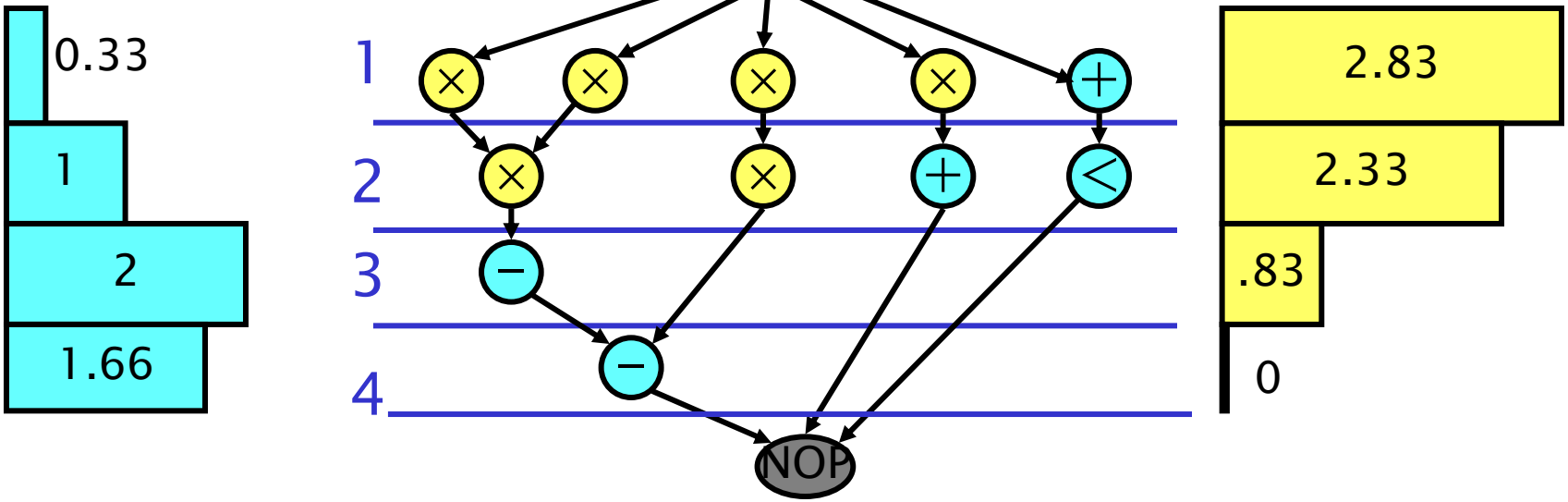
$$q_{add}(4) = 1 + \frac{1}{3} + \frac{1}{3} = 1.66$$

$$q_{mult}(1) = 1 + 1 + \frac{1}{2} + \frac{1}{3} = 2.83$$

$$q_{mult}(2) = 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{3} = 2.33$$

$$q_{mult}(3) = \frac{1}{2} + \frac{1}{3} = 0.83$$

$$q_{mult}(4) = 0$$



Force-Directed Scheduling Algorithm: Idea

- Very similar to LIST_L(G(V,E), **a**)
  - Compute mobility of operations using ASAP and ALAP
  - Computer operation probabilities and type distributions
  - Select and schedule operations
  - Update operation probabilities and type distributions
  - Go to next control step
- Difference with list sched in selecting operations
  - Select operations with least force
  - Consider the effect on the type distribution
  - Consider the effect on successor nodes and their type distributions