EE/CE 6301: Advanced Digital Logic

Bill Swartz

Dept. of EE
Univ. of Texas at Dallas

Scheduling

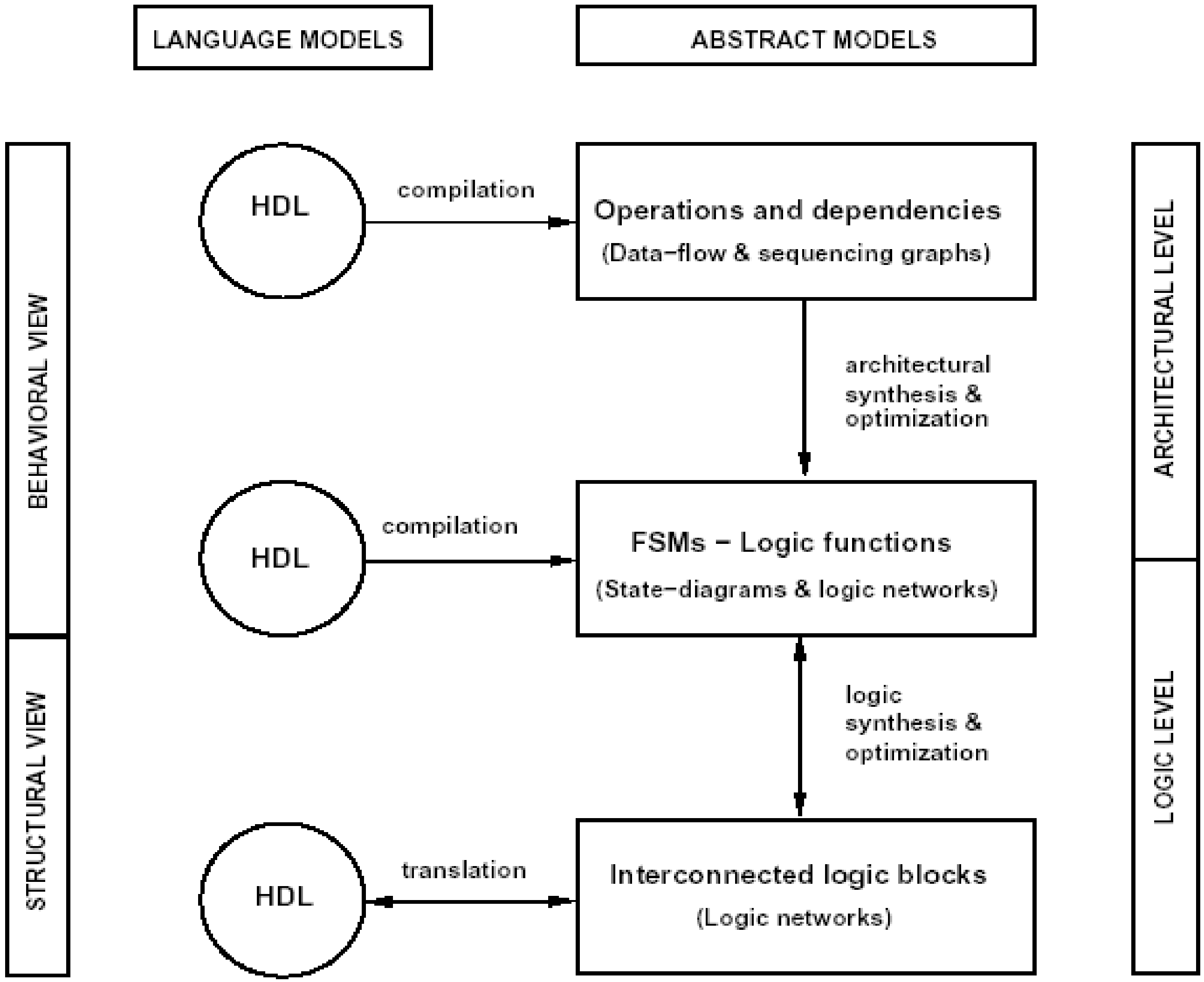
Synthesis and Design Automation

Architectural Synthesis

Synthesis

- Transform behavioral into structural view.
- Architectural-level synthesis
  - Architectural abstraction level.
  - Determine macroscopic structure.
  - Example: major building blocks like adder, register, mux.
- Logic-level synthesis
  - Logic abstraction level.
  - Determine microscopic structure.
  - Example: logic gate interconnection.

Synthesis and Optimization



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.

Architectural-Level Synthesis

- Translate HDL models into sequencing graphs.
- Behavioral-level optimization
  - Optimize abstract models independently from the implementation parameters.
- Architectural synthesis and optimization
  - Create macroscopic structure
    - data-path and control-unit.
  - Consider area and delay information of the implementation.

Example - Pseudo Code

- Second-order differential equation solver

```
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 = x < a;
                x = xl; u = ul; y = yl;
                }
        until ( c ) ;
write (y);
}
```

Example - VHDL

```
package mypack is
  subtype bit8 is integer range 0 to 255;    e.g. 8-bit signal/wire/port
end mypack;

use work.mypack.all;
entity DIFFEQ is
port(
        dx_port, a_port, x_port,  u_port : in  bit8;    Port definition
        y_port : inout  bit8;
        clock, start   : in bit );
end diffeq;

architecture BEHAVIOR of DIFFEQ is
begin
process
        variable x, a, y, u, dx, xl, ul, yl:  bit8;
begin
        wait until start'event and start = '1';    wait for (transition)
        x := x_port; y := y_port; a := a_port;
        u := u_port; dx := dx_port;
        DIFFEQ_LOOP:
        while ( x < a ) loop
                wait until clock'event and clock = '1';
                xl := x + dx;
                ul := u - (3 * x * u * dx) - (3 * y * dx);    Main computation.
                yl := y + (u * dx);
                x  := xl; u := ul ; y := yl;
        end loop DIFFEQ_LOOP;
        y_port <=  y;
end process;
end BEHAVIOR;
```

Example - Verilog

8-bit signal/wire/port

```
module DIFFEQ (xp,yp,up,dx,a,clock,start);
input [7:0] a,dx,xp,up;
inout [7:0] yp;
input       clock, start;
reg   [7:0] x1,u1,y1,x,y,u;
always   @(posedge start)
begin
     x = xp;  y = yp;  u = up;
     while (x < a)
     begin
       x1 = x + dx;
       u1 = u - (3 * x * u * dx) - (3 * y * dx);
       y1 = y + (u * dx);
       @(posedge clock);
       x = x1; u = u1; y = y1;
     end
end
assign yp = y;
endmodule
```

port defn.

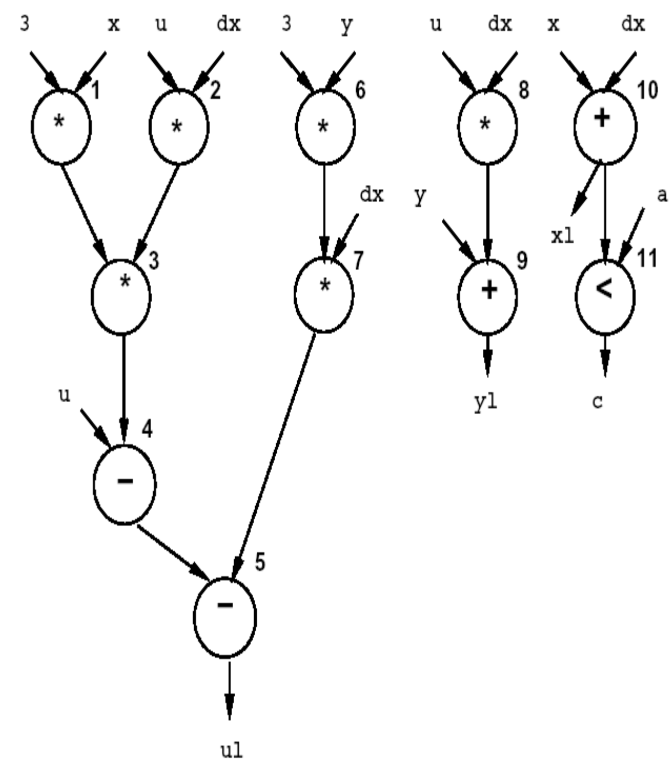
wait for S

dependencies is just like hazards in pipelining

Dataflow Graphs

- Behavioral views of architectural models.
- Useful to represent data-paths.
- Graph
  - Vertices = operations.
  - Edges = dependencies.
- Dependencies arise due
  - Input to an operation is result of another operation.
  - Serialization constraints in specification.
  - Two tasks share the same resource.

$$
\begin{aligned}
xl &= x + dx \\
ul &= u - (3 \cdot x \cdot u \cdot dx) - (3 \cdot y \cdot dx) \\
yl &= y + u \cdot dx \\
c &= xl < a
\end{aligned}
$$

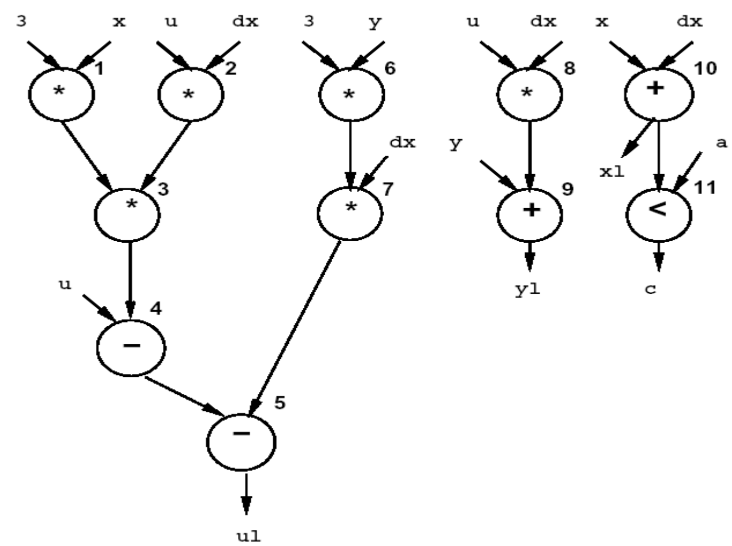


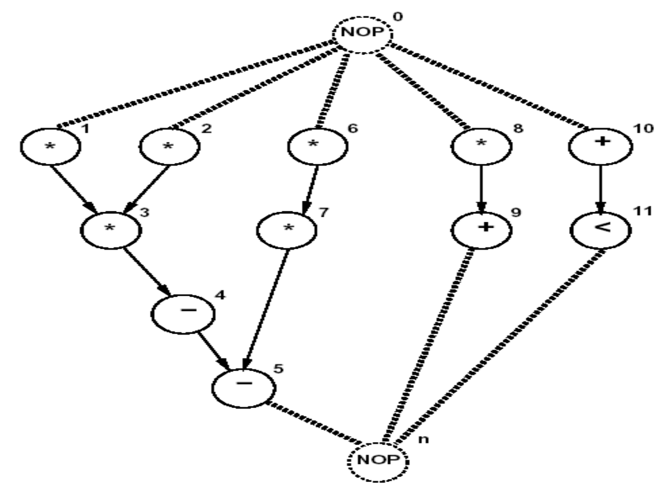
Dataflow Graphs (cont.)

- Assumes the existence of variables who store information required and generated by operations.
- Each variable has a *lifetime* which is the interval from *birth* to *death.*
- *Variable birth* is the time at which the value is generated.
- *Variable death* is the latest time at which the value is referenced as input to an operation.
- Values must be preserved during life-time.

Sequencing Graphs

- Useful to represent data-path and control.
- Extended dataflow graphs
  - —Control Data Flow Graphs (CDFGs).
  - —Polar: source and sink.
  - —Operation serialization.
  - —Hierarchy.
  - —Control-flow commands
    - – branching and iteration.
- Paths in the graph represent concurrent streams of operations.



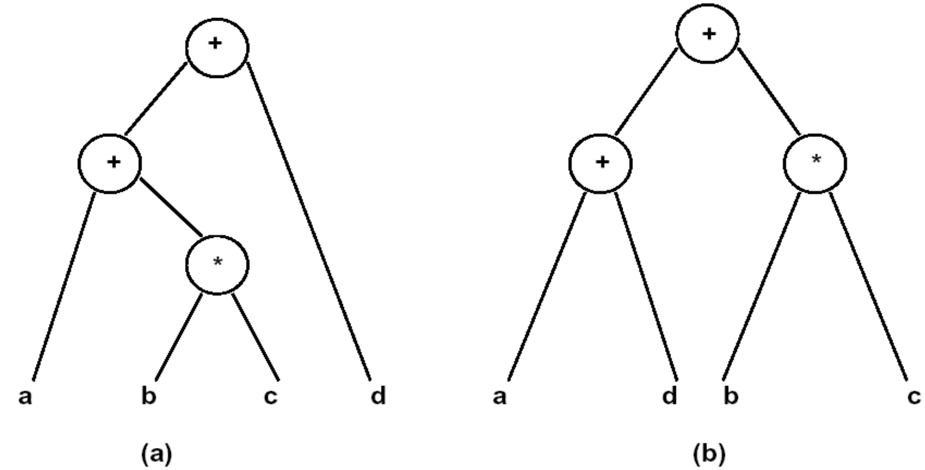


Behavioral-level optimization

- Tree-height reduction using commutative and associative properties
  - $x = a + b * c + d$

    =>

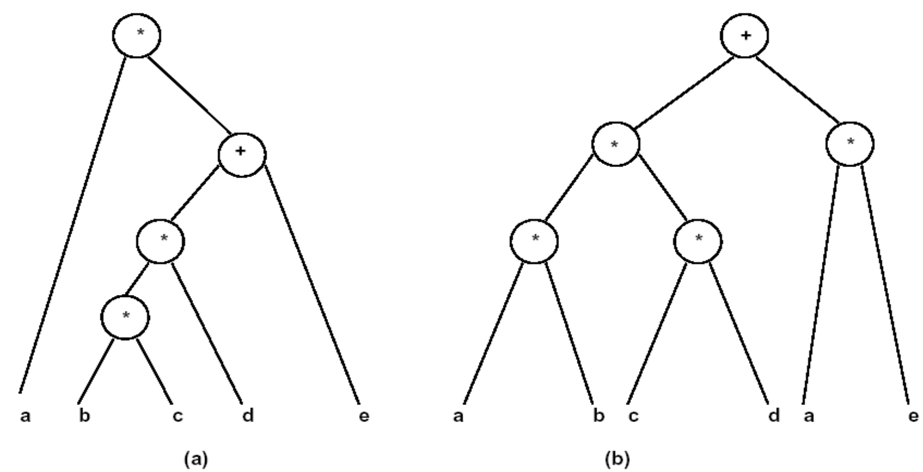
    $x = (a + d) + b * c$



- Tree-height reduction using distributive property
  - $x = a * (b * c * d + e)$

    =>

    $x = a * b * c * d + a * e$



Architectural Synthesis and Optimization

- Synthesize macroscopic structure in terms of building-blocks.
- Explore area/performance trade-offs
  - maximum performance implementations subject to area constraints.
  - minimum area implementations subject to performance constraints.
- Determine an optimal implementation.
- Create logic model for data-path and control.

Circuit Specification for Architectural Synthesis

- Circuit behavior
  - Sequencing graphs.
- Building blocks
  - Resources.
    - Functional resources: process data (e.g. ALU).
    - Memory resources: store data (e.g. Register).
    - Interface resources: support data transfer (e.g. MUX and Buses).
- Constraints
  - Interface constraints
    - Format and timing of I/O data transfers.
  - Implementation constraints
    - Timing and resource usage.
      - Area
      - Cycle-time and latency

Resources

- Functional resources: perform operations on data.
  - Example: arithmetic and logic blocks.
  - Standard resources
    - Existing macro-cells.
    - Well characterized (area/delay).
    - Example: adders, multipliers, ALUs, Shifters, ...
  - Application-specific resources
    - Circuits for specific tasks.
    - Yet to be synthesized.
    - Example: instruction decoder.
- Memory resources: store data.
  - Example: memory and registers.
- Interface resources
  - Example: busses and ports.

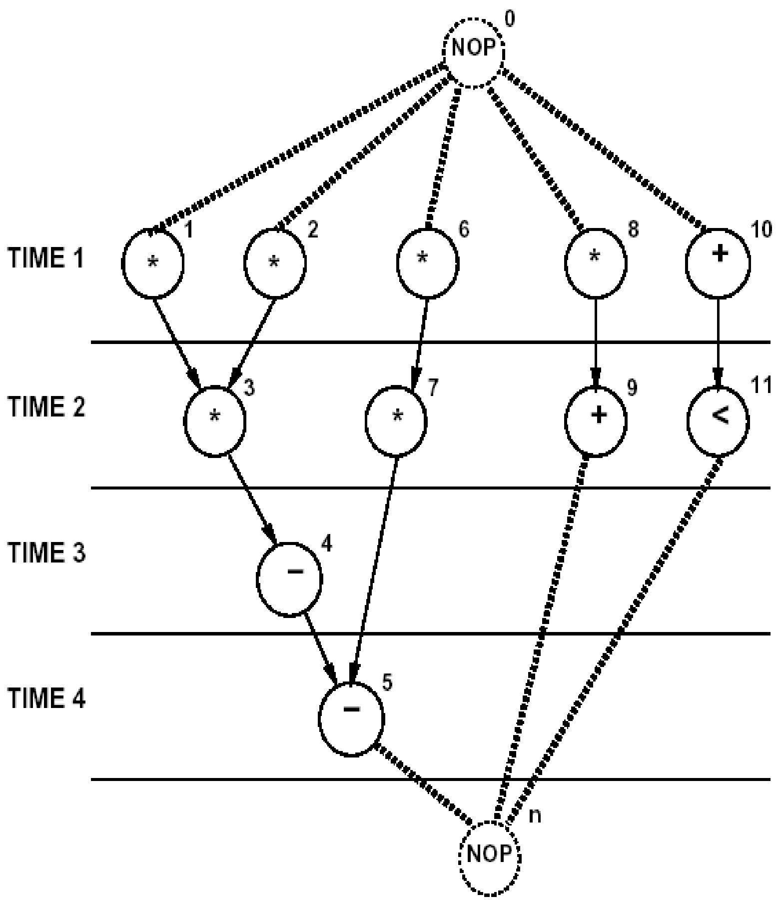
Resources and Circuit Families

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

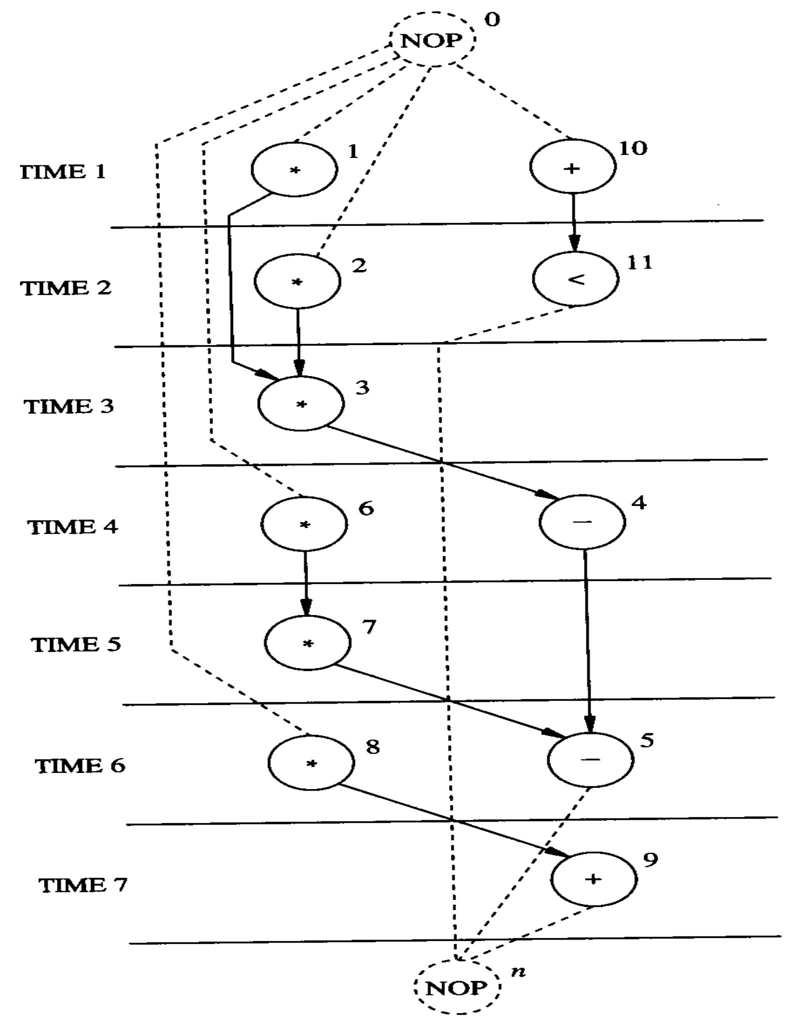
Synthesis in the Temporal Domain: Scheduling

- Scheduling
  - —Associate a start-time with each operation.
  - —Satisfying all the sequencing (timing and resource) constraint.
- Goal
  - —Determine area/latency trade-off.
  - —Determine latency and parallelism of the implementation.
- Scheduled sequencing graph
  - —Sequencing graph with start-time annotation.
- Unconstrained scheduling.
- Scheduling with timing constraints
- Scheduling with resource constraints.

Tradeoff in Scheduling

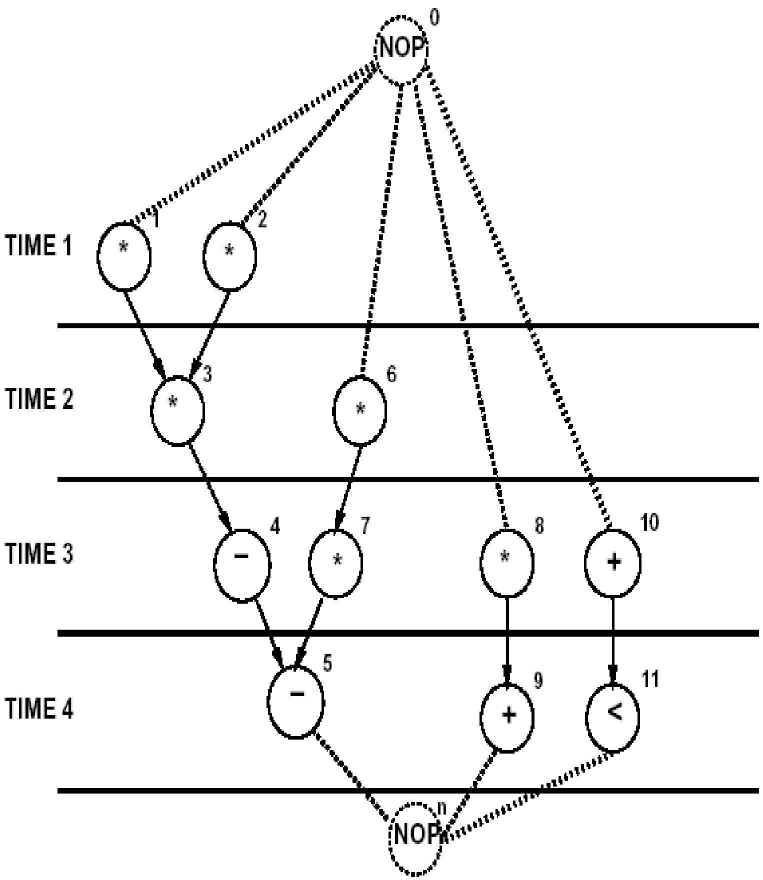


4 Multipliers, 2 ALUs

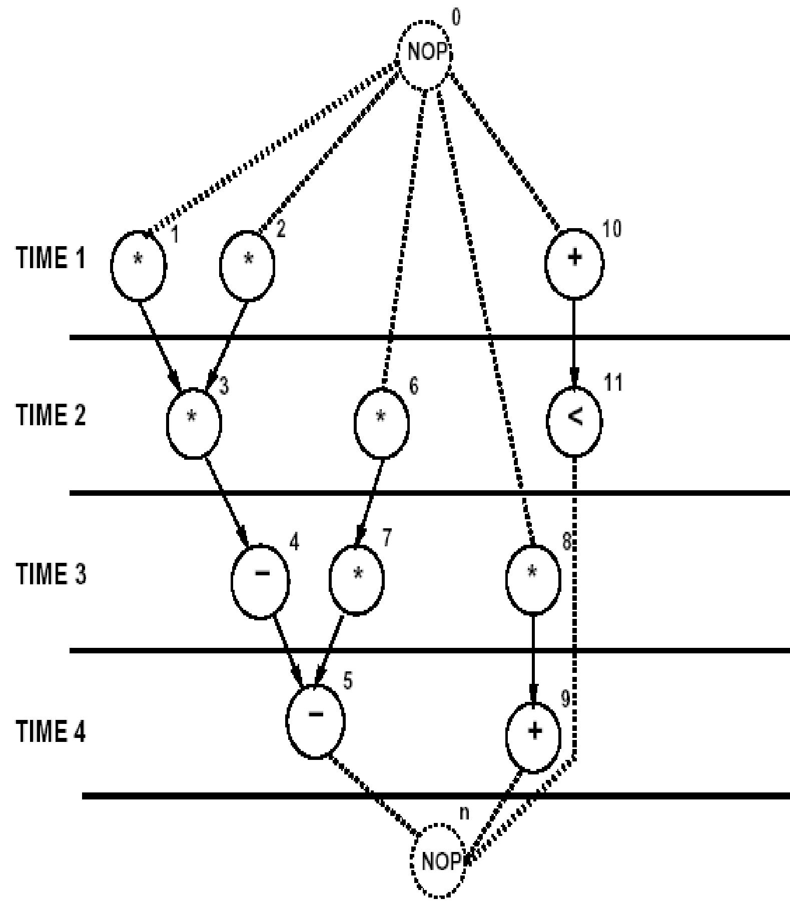


1 Multiplier , 1 ALU

Tradeoff in Scheduling (cont.)



2 Multipliers, 3 ALUs

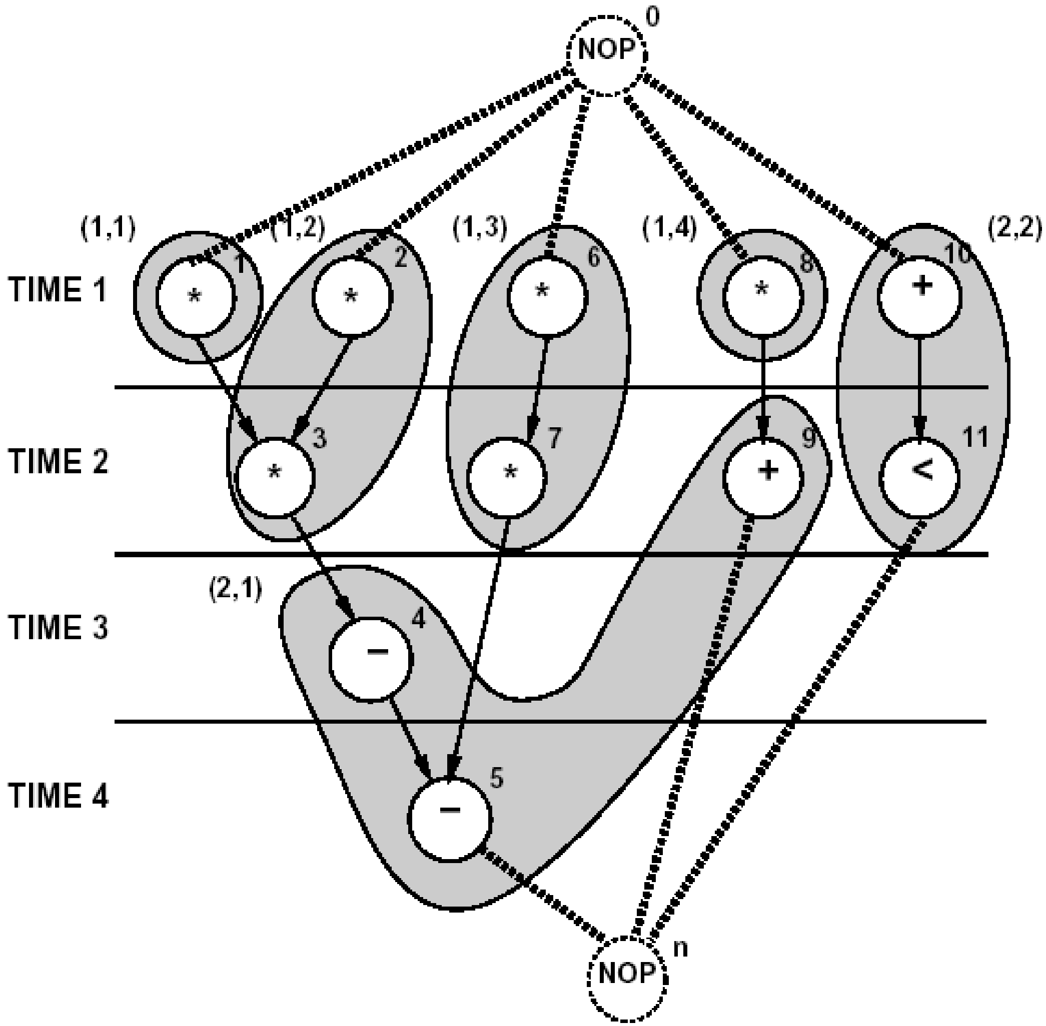


2 Multipliers, 2 ALUs

Synthesis in the Spatial Domain: Binding

- Binding
  - Associate a resource with each operation with the same type.
  - Determine 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.

Example: Bound Sequencing Graph



Performance and Area Estimation

- Resource-dominated circuits
  - Area = sum of the area of the resources bound to the operations.
    - Determined by binding.
  - Latency = start time of the sink operation (minus start time of the source operation).
    - Determined by scheduling
- Non resource-dominated circuits
  - Area also affected by
    - registers, steering logic, wiring and control.
  - Cycle-time also affected by
    - steering logic, wiring and (possibly) control.

Scheduling Algorithms

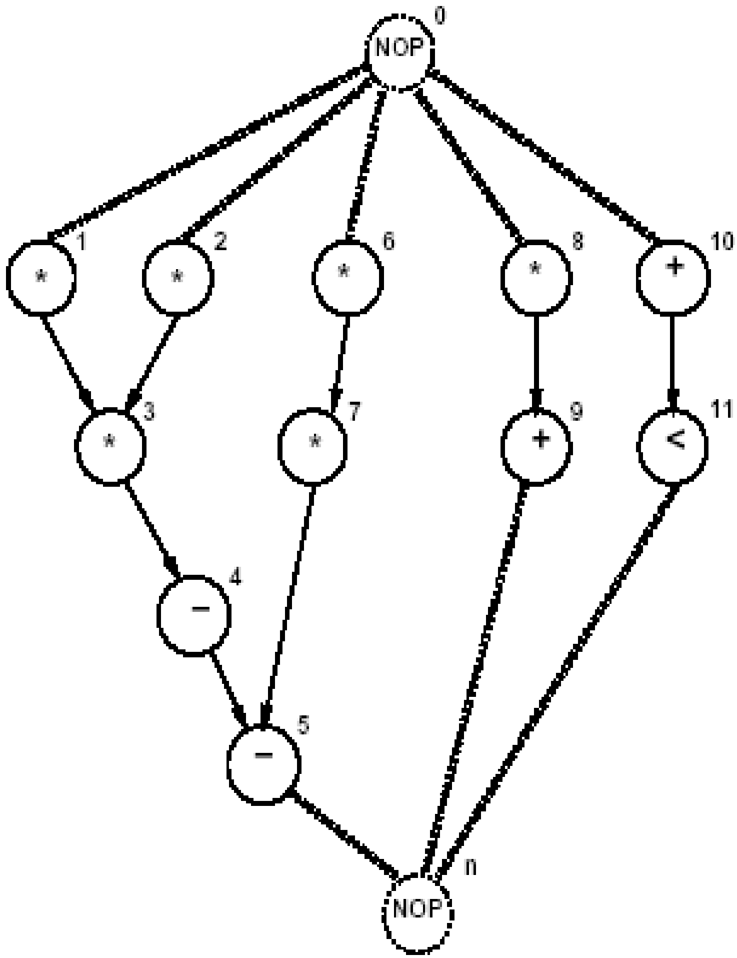
Outline

- The scheduling problem
- Scheduling without constraints
- Scheduling under timing constraints
  - Relative scheduling
- Scheduling under resource constraints
  - The ILP model
  - Heuristic methods
    - List scheduling
    - Force-directed scheduling

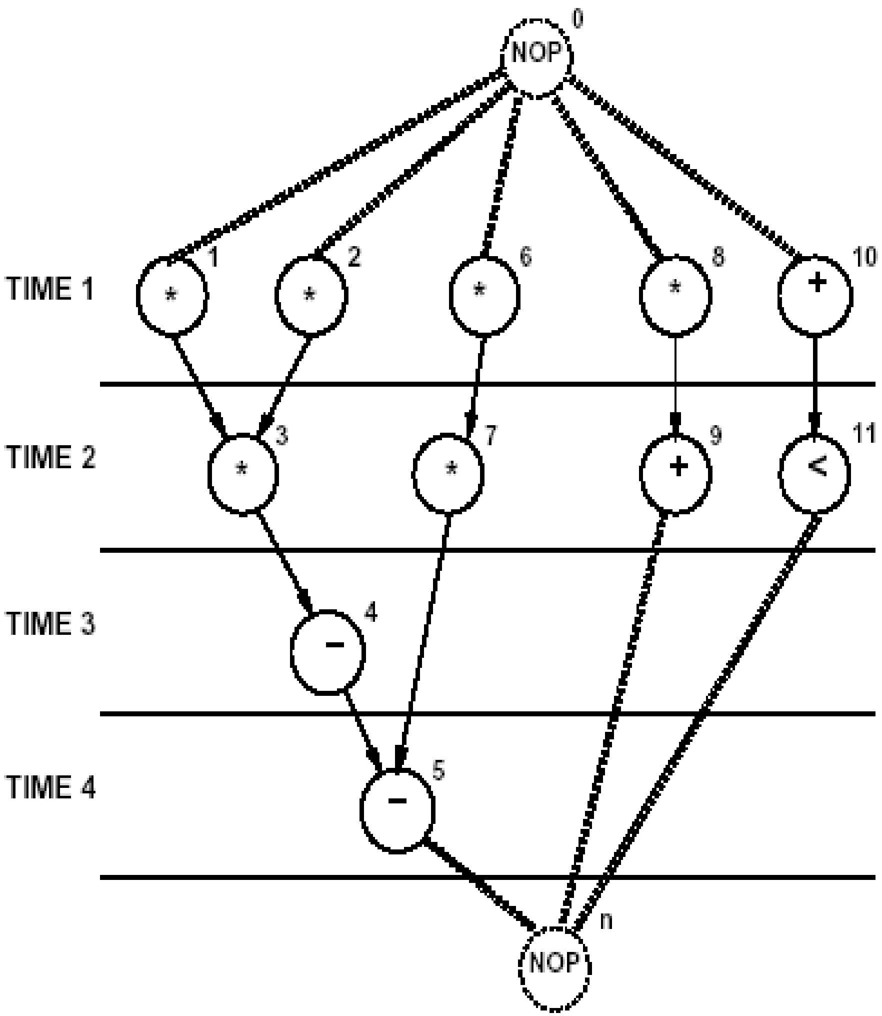
Scheduling

- Circuit model
  - Sequencing graph.
  - Cycle-time is given.
  - Operation delays expressed in cycles.
- Scheduling
  - Determine the start times for the operations.
  - Satisfying all the sequencing (timing and resource) constraint.
- Goal
  - Determine area/latency trade-off.
- Scheduling affects
  - Area: maximum number of concurrent operations of same type is a lower bound on required hardware resources.
  - Performance: concurrency of resulting implementation.

Scheduling Example



Sequencing Graph



A scheduled DFG

Scheduling Models

- Unconstrained scheduling.
- Scheduling with timing constraints
  - Latency
  - Detailed timing constraints
- Scheduling with resource constraints
- Simplest scheduling model
  - All operations have bounded delays.
  - All delays are in cycles.
    - Cycle-time is given
  - No constraints - no bounds on area.
  - Goal
    - Minimize latency

Minimum-Latency Unconstrained Scheduling

- Given a set of operations V with integer delays D and a partial order on the operations E
- Find an integer labeling of the operations $\varphi : V \rightarrow Z^+$, such that
  - $t_i = \varphi(v_i)$,
  - $t_i \geq t_j + d_j \quad \forall\, i, j$ s.t. $(v_j, v_i) \in E$
  - and t_n is *minimum*.
- Unconstrained scheduling used when
  - Dedicated resources are used.
  - Operations differ in type.
  - Operations cost is marginal when compared to that of steering logic, registers, wiring, and control logic.
  - Binding is done before scheduling: resource conflicts solved by serializing operations sharing same resource.
  - Deriving bounds on latency for constrained problems.

ASAP Scheduling Algorithm

- Denote by *t^S* the start times computed by the *as soon as possible* (ASAP) algorithm.
- Yields *minimum* values of start times.

$ASAP\ (\ G_s(V,E))\ \{$

  Schedule v_0 by setting $t_0^S = 1$;

  repeat {

    Select a vertex v_i whose pred. are all scheduled;

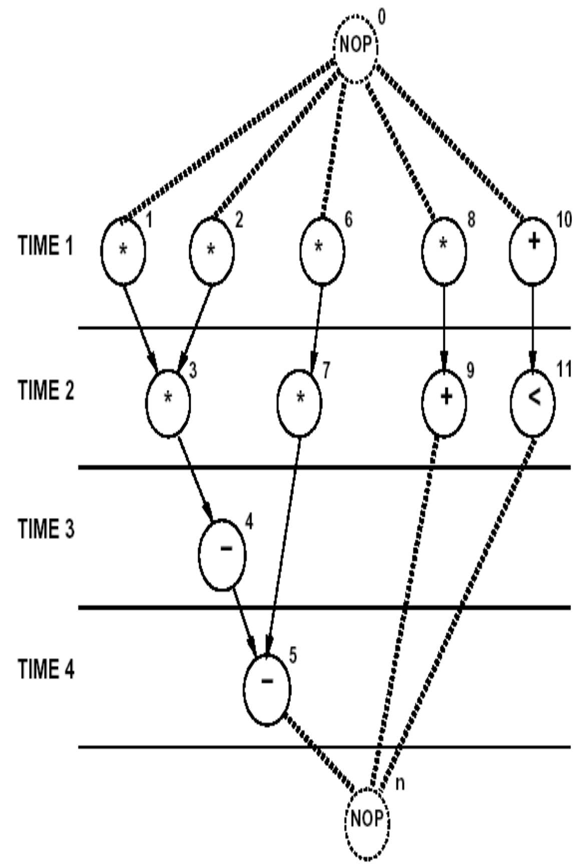
    Schedule v_i by setting $t_i^S = \max\limits_{j:(v_j,v_i)\in E} t_j^S + d_j$;

  }

  until (v_n is scheduled) ;

  return (t^S);

}



ALAP Scheduling Algorithm

- Denote by t^L the start times computed by the *as late as possible* (ALAP) algorithm.
- Yields *maximum* values of start times.
- Latency upper bound λ (i.e. t_n-t_0)

$ALAP(\, G_s(V,E), \bar{\lambda}) \; \{$

  Schedule v_n by setting $t_n^L = \bar{\lambda} + 1$;

  repeat $\{$

    Select vertex v_i whose succ. are all scheduled;

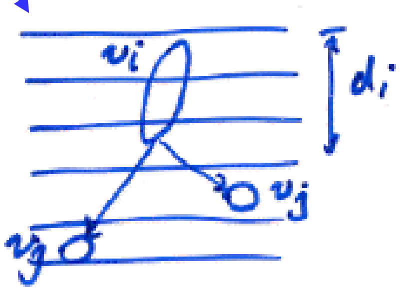
    Schedule v_i by setting $t_i^L = \min\limits_{j:(v_i,v_j)\in E} t_j^L - d_i$;

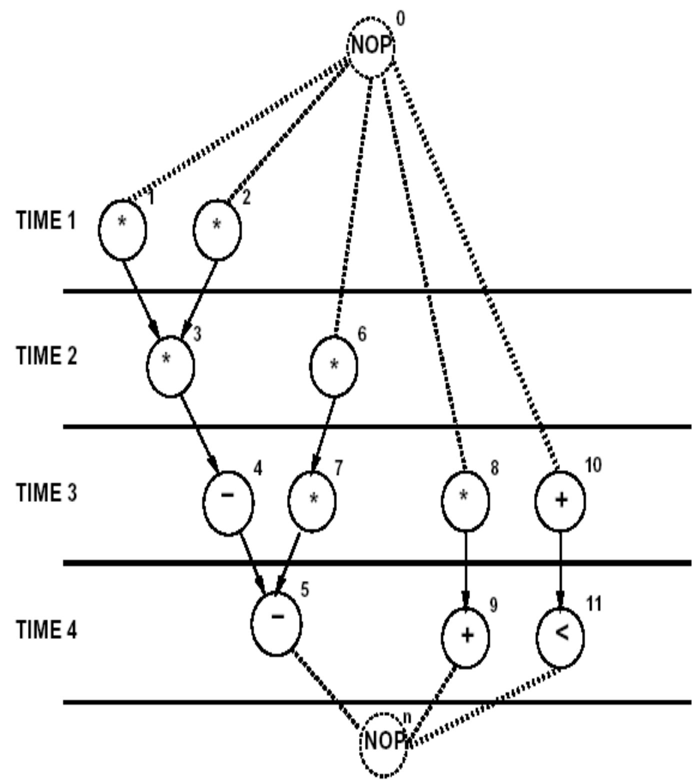
  $\}$

  until (v_0 is scheduled) ;

  return (t^L);

$\}$



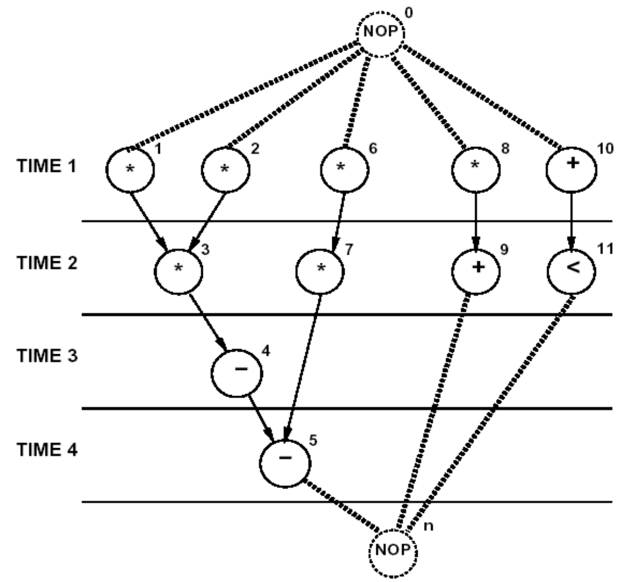


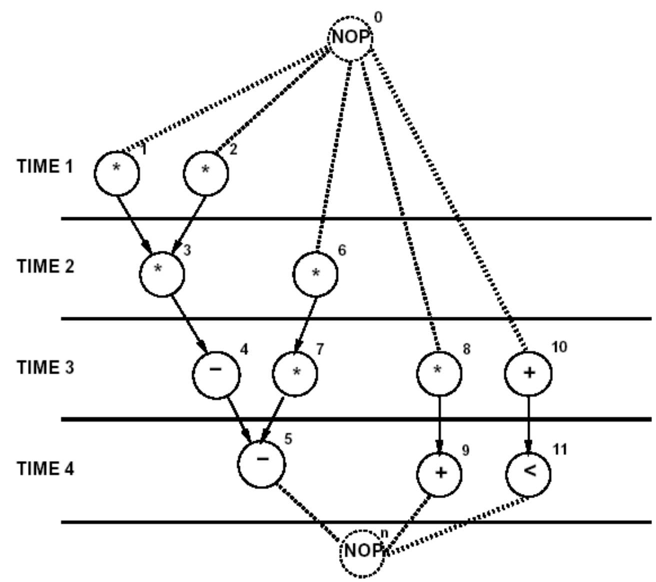
Latency-Constrained Scheduling

- 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.
  - Zero mobility implies that an operation can be started only at one given time step.
  - Mobility greater than 0 measures span of time interval in which an operation may start.
- Slack on the start time.

Example

- Operations with zero mobility
  - {v1, v2, v3, v4, v5}
  - Critical path
- Operations with mobility one
  - {v6, v7}
- Operations with mobility two
  - {v8, v9, v10, v11}



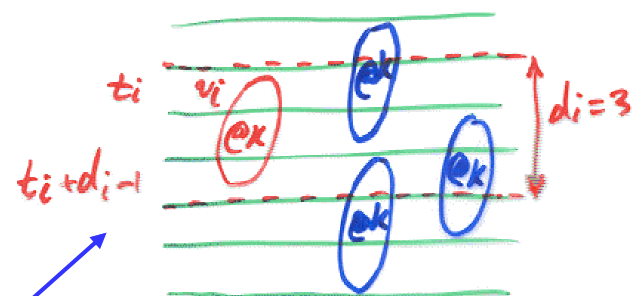


Scheduling under Resource Constraints

- Classical scheduling problem.
  - Fix area bound - minimize latency.
- The amount of available resources affects the achievable latency.
- Dual problem
  - Fix latency bound - minimize resources.
- Assumption
  - All delays bounded and known.

Minimum Latency Resource-Constrained Scheduling

- Given a set of ops V with integer delays D, a partial order on the operations E, and upper bounds $\{a_k;\ k = 1, 2, \ldots, n_{res}\}$
- Find an integer labeling of the operations $\varphi : V \to Z^+$, such that



  - $t_i = \varphi(v_i)$,
  - $t_i \geq t_j + d_j \quad \forall$ i, j s.t. $(v_j, v_i) \in E$

    $|\{v_i | \mathcal{T}(v_i) = k \text{ and } t_i \leq l < t_i + d_i\}| \leq a_k$
    $\forall$types $k = 1, 2, \ldots, n_{res}$ and $\forall$ steps l

    $\mathcal{T} : V \to \{1, 2, \ldots n_{res}\}$
  - and t_n is *minimum*.
- Number of operations of any given type in any schedule step does not exceed bound.

Scheduling under Resource Constraints

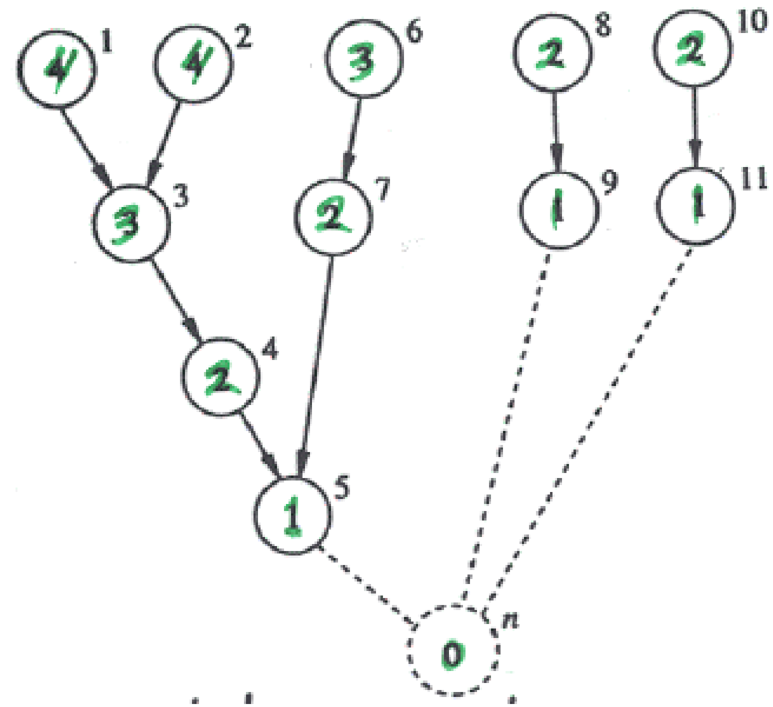
- Intractable problem
- Algorithms
  - Approximate
    - List scheduling
    - Force-directed scheduling
  - Exact
    - Integer linear program
    - Hu (restrictive assumptions)

List Scheduling

List Scheduling Algorithms

- Heuristic method for
  - Minimum latency subject to resource bound.
  - Minimum resource subject to latency bound.
- Greedy strategy.
- Priority list heuristics.
  - Assign a weight to each vertex indicating its scheduling priority
    - **Longest path to sink**.
    - Longest path to timing constraint.

Priority of V_i =label of V_i=α_i= #of edges in the longest path From $V_i \rightarrow V_n$



Labeled Sequencing Graph

List Scheduling Algorithm for Minimum Latency

```
LIST_L( G(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, s.t. |S_k| + |T_{l,k}| ≤ a_k;
            Schedule the S_k operations at step l;
        }
        l = l + 1;
    }
    until (v_n is scheduled) ;
    return (t);
}
```

Based on a "priority" metric (e.g. mobility, labeling, etc.)

List Scheduling for Minimum Latency (cont.)

- Candidate Operations $U_{l,k}$
  - Operations of type k whose predecessors are scheduled and completed at time step before l

$$U_{l,k} = \{v_i \in V : Type(v_i) = k \text{ and } t_j + d_j \le l \ \forall j : (v_j, v_i) \in E\}$$

- Unfinished operations $T_{l,k}$ are operations of type k that started at earlier cycles and whose execution is not finished at time l

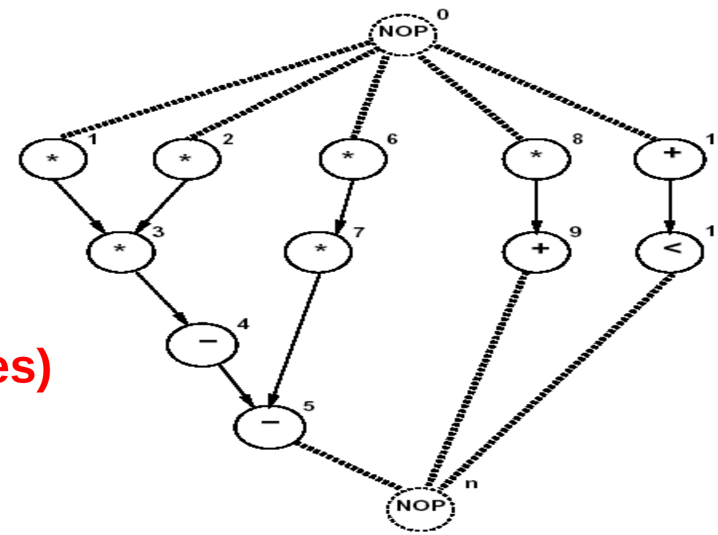
$$T_{l,k} = \{v_i \in V : Type(v_i) = k \ \textbf{and} \ t_j + d_j > l \ \forall j : (v_j, v_i) \in E\}$$

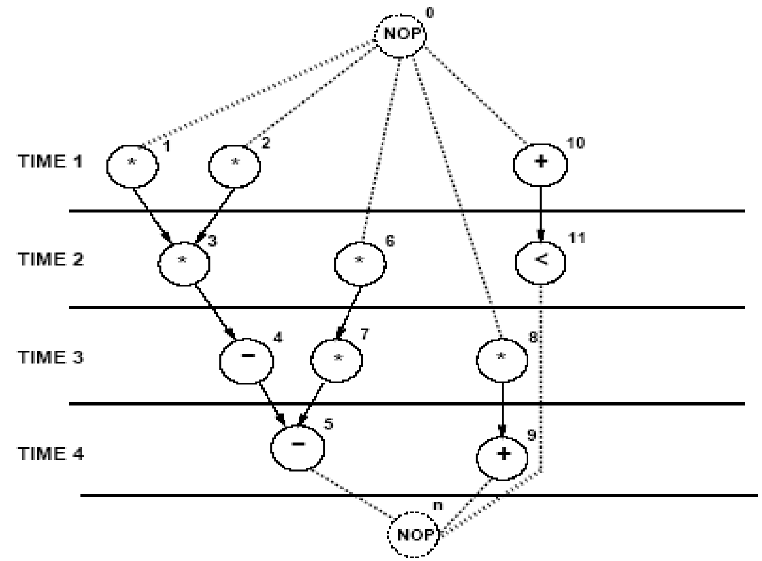
  - Note that when execution delays are 1, $T_{l,k}$ is empty.

Example I

- **Assumptions**
  - **a_1 = 2 multipliers with delay 1.**
  - **a_2 = 2 ALUs with delay 1.**
- First Step
  - 4, 4, 3, 2
  - $U_{1,1} = \{v_1, v_2, v_6, v_8\}$
  - Select $\{v_1, v_2\}$
  - $U_{1,2} = \{v_{10}\}$; selected
- Second step
  - 3, 3, 2
  - $U_{2,1} = \{v_3, v_6, v_8\}$
  - select $\{v_3, v_6\}$
  - $U_{2,2} = \{v_{11}\}$; selected
- Third step
  - 2, 2
  - $U_{3,1} = \{v_7, v_8\}$
  - Select $\{v_7, v_8\}$
  - $U_{3,2} = \{v_4\}$; selected
- Fourth step
  - $U_{4,2} = \{v_5, v_9\}$; selected

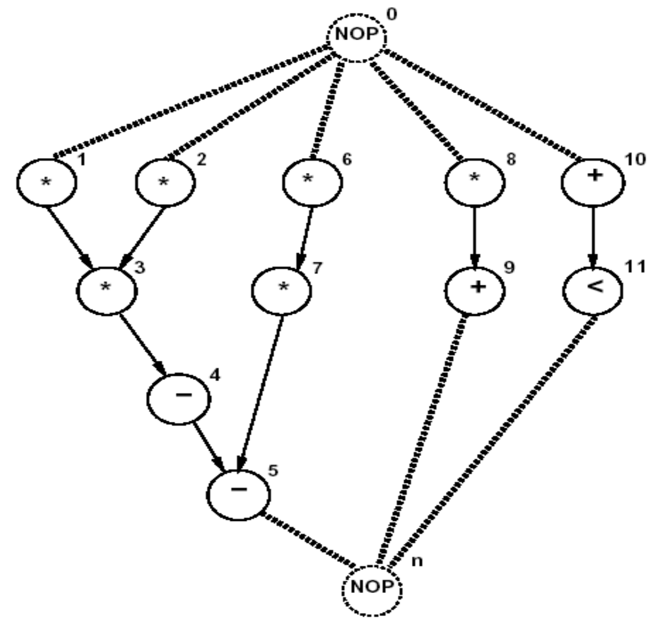
Labels (priorities)

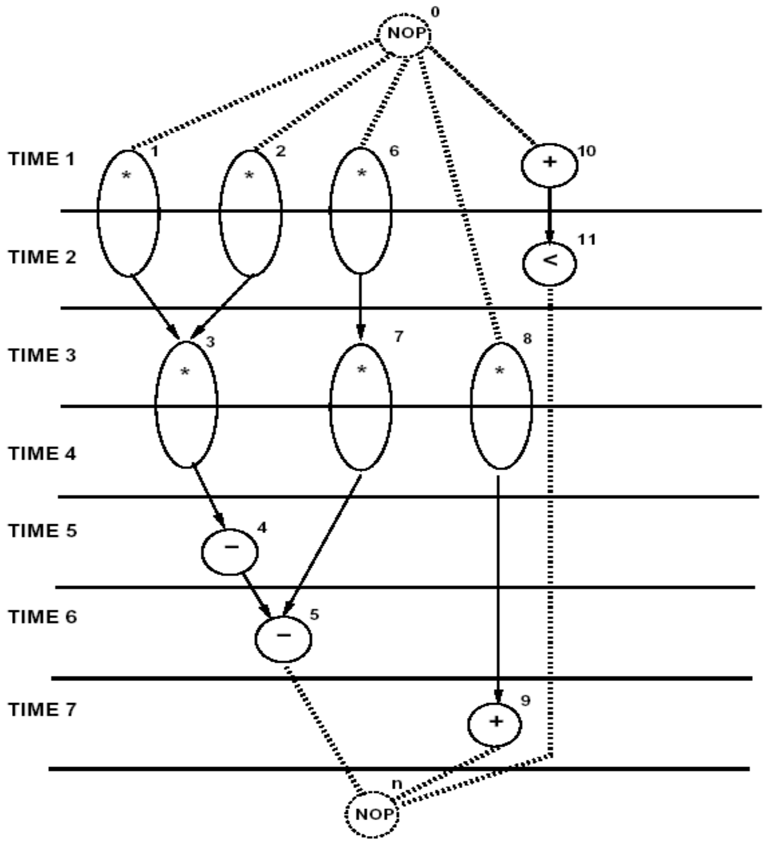




Example II

- **Assumptions**
  - a_1 = 3 multipliers with delay 2.
  - a_2 = 1 ALU with delay 1.





Operation		
Multiply	ALU	Start time
$\{v_1, v_2, v_6\}$	v_{10}	1
—	v_{11}	2
$\{v_3, v_7, v_8\}$	—	3
—	—	4
—	v_4	5
—	v_5	6
—	v_9	7

List Scheduling for Minimum Resource Usage

```
LIST_R( G(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 (∅);
    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 ∈ U_lk};
            Schedule the candidate operations
            with zero slack and update a;
            Schedule the candidate operations
            that do not require additional resources;
        }
        l = l + 1;
    }
    until (v_n is scheduled) ;
    return (t, a);
}
```

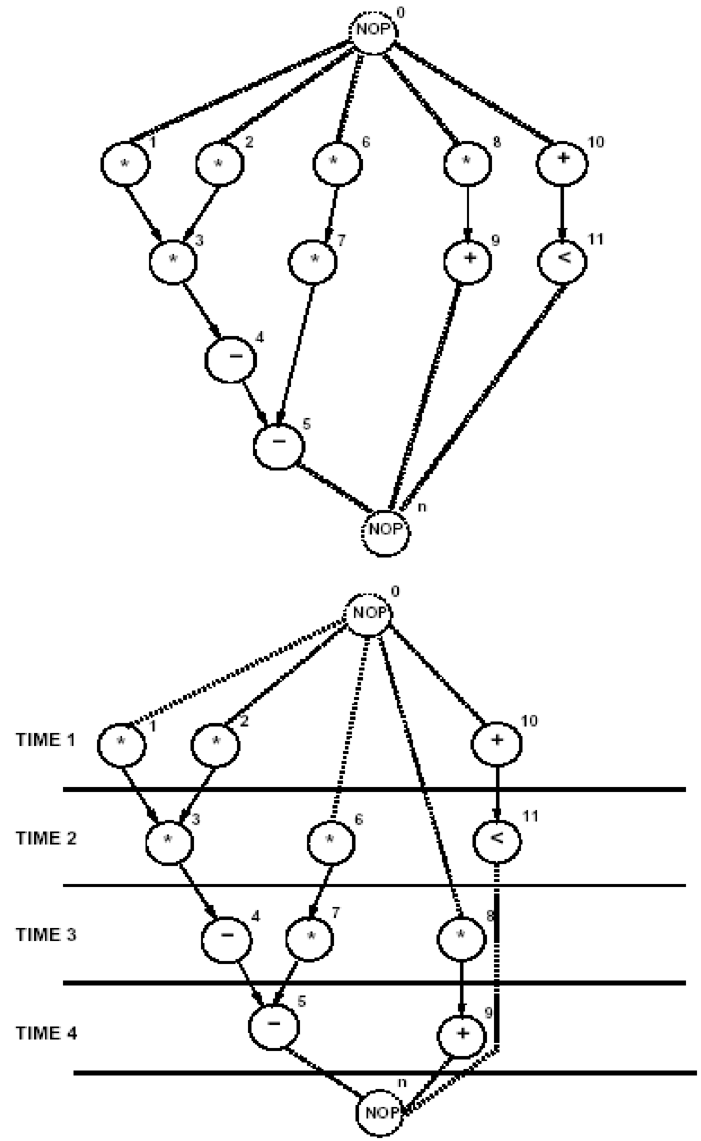
ALAP does not exist (Latency is too tight)

a_i will keep track of maximum concurrent operations of type i

Lower slack means higher urgency/priority

Example – (i)

- Assume $\lambda=4$
- Let a = $[1, 1]^T$
- First Step
  - $U_{1,1} = \{v_1, v_2, v_6, v_8\}$
  - Operations with zero slack $\{v_1, v_2\}$
  - a = $[2, 1]^T$
  - $U_{1,2} = \{v_{10}\}$
- Second step
  - $U_{2,1} = \{v_3, v_6, v_8\}$
  - Operations with zero slack $\{v_3, v_6\}$
  - $U_{2,2} = \{v_{11}\}$
- Third step
  - $U_{3,1} = \{v_7, v_8\}$
  - Operations with zero slack $\{v_7, v_8\}$
  - $U_{3,2} = \{v_4\}$
- Fourth step
  - $U_{4,2} = \{v_5, v_9\}$
  - Both have zero slack; a = $[2, 2]^T$



Example – (ii)

- Assume $\lambda=7$
- Let a = $[1, 1]^T$

slack: 3 3 4 5

(i) $\begin{cases} k=1 \Rightarrow U_{1,1}=\{v_1, v_2, v_6, v_8\} \Rightarrow \begin{cases} slack > 0 \\ v_1 \rightarrow T_1 \end{cases} \\ k=2 \Rightarrow U_{1,2}=\{\overset{5}{v_{10}}\} \Rightarrow \begin{cases} slack > 0 \\ v_{10} \rightarrow T_1 \end{cases} \end{cases}$

(ii) $\begin{cases} k=1 \Rightarrow U_{2,1}=\{\overset{2}{v_2}, \overset{3}{v_6}, \overset{4}{v_8}\} \Rightarrow v_2 \rightarrow T_2 \\ k=2 \Rightarrow U_{2,2}=\{\overset{5}{v_{11}}\} \Rightarrow v_{11} \rightarrow T_2 \end{cases}$

(iii) $\begin{cases} k=1 \Rightarrow U_{3,1}=\{\overset{2}{v_3}, \overset{2}{v_6}, \overset{3}{v_8}\} \Rightarrow v_3 \rightarrow T_3 \\ k=2 \Rightarrow U_{3,2}=\{\ \} \end{cases}$

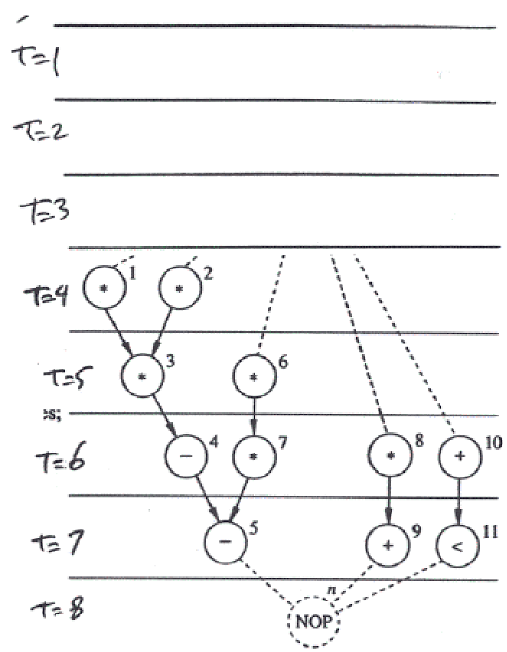
(iv) $\begin{cases} k=1 \Rightarrow U_{4,1}=\{\overset{1}{v_6}, \overset{2}{v_8}\} \Rightarrow v_6 \rightarrow T_4 \\ k=2 \Rightarrow U_{4,2}=\{\overset{2}{v_4}\} \Rightarrow v_4 \rightarrow T_4 \end{cases}$

(v) $\begin{cases} k=1 \Rightarrow U_{5,1}=\{\overset{1}{v_7}, \overset{1}{v_8}\} \Rightarrow v_7 \rightarrow T_5 \\ k=2 \Rightarrow U_{5,2}=\{\ \} \end{cases}$

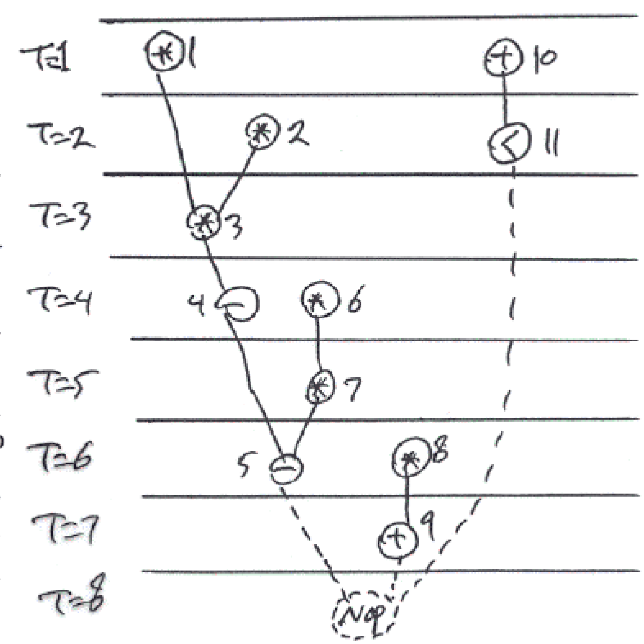
(vi) $\begin{cases} k=1 \Rightarrow U_{6,1}=\{\overset{0}{v_8}\} \Rightarrow v_8 \rightarrow T_6 \\ k=2 \Rightarrow U_{6,2}=\{\overset{1}{v_5}\} \Rightarrow v_5 \rightarrow T_6 \end{cases}$

(vii) $\begin{cases} k=1 \Rightarrow U_{7,1}=\{\ \} \\ k=2 \Rightarrow U_{7,2}=\{\overset{0}{v_9}\} \Rightarrow v_9 \rightarrow T_7 \end{cases}$

ALAP Schedule



Final Schedule



Scheduling Based on Integer Linear Programming (ILP)

ILP Solution

- Use standard ILP packages.
- Transform into LP problem [Gebotys].
- Advantages
  - Exact method.
  - Other constraints can be incorporated easily
    - Maximum and minimum timing constraints
- Disadvantages
  - Works well up to few thousand variables.

ILP Formulation

- Binary decision variables
  - $X = \{ x_{il};\ i = 1, 2, \ldots, n;\ l = 1, 2, \ldots, \lambda+1\}$.
  - x_{il} is TRUE ("1") only when operation v_i starts in step l of the schedule (i.e. $l = t_i$).
  - λ is an upper bound on latency.
- Start time of operation v_i

$$t_i = \sum_l l \cdot x_{il}$$

- Operations start only once

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

ILP Formulation (cont.)

- Sequencing relations must be satisfied
  - $t_i \geq t_j + d_j \qquad \forall (v_j, v_i) \in E$
  - $\sum_l l \cdot x_{il} - \sum_l l \cdot x_{jl} - d_j \geq 0 \quad \forall (v_j, v_i) \in E$
- Resource bounds must be satisfied
  - Simple case (unit delay)
  - $\sum_{i:\mathcal{T}(v_i)=k} x_{il} \leq a_k \quad k = 1, 2, \ldots, n_{res}; \ \forall l$

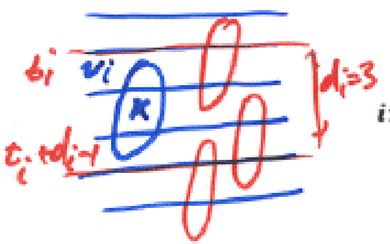
ILP Formulation (cont.)

- Minimize $c^T\, t$ such that

$$\sum_l x_{il} = 1, \quad i = 0, 1, \ldots, n$$ } uniqueness

$t_i \geq t_j + d_j$

$$\sum_l l \cdot x_{il} - \sum_l l \cdot x_{jl} - d_j \geq 0, \quad i, j = 0, 1, \ldots, n \;:\; (v_j, v_i) \in E$$ } dependencies



$$\sum_{i:\mathcal{T}(v_i)=k} \; \sum_{m=l-d_i+1}^{l} x_{im} \leq a_k, \quad k = 1, 2, \ldots, n_{res}, \quad l = 1, 2, \ldots, \bar{\lambda}+1$$ } Resource constraints

$$x_{il} \in \{0, 1\}, \quad i = 0, 1, \ldots, n, \quad l = 1, 2, \ldots, \bar{\lambda}+1$$ } Acceptable values.

$t_i \leq m \leq t_i + d_i - 1$

$t_i - d_i + 1 \leq m \leq t_i$

ILP Formulation (cont.)

- About the objective function ($c^T t$)
  - —$c^T=[0,0,\ldots,0,1]^T$ corresponds to minimizing the latency of the schedule.

$$c^T t = \sum_{l=1}^{\bar{\lambda}+1} l \cdot x_{nl} = t_n$$

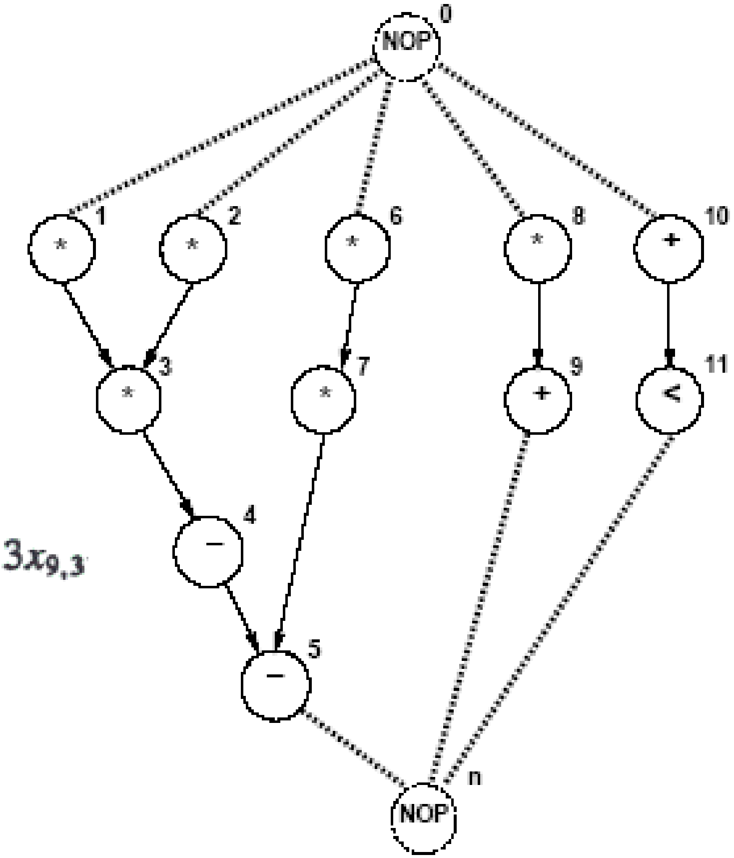
  - —$c^T=[1,1,\ldots,1,1]^T$ corresponds to finding the earliest start times of all operations under the given constraints.

$$c^T t = \sum_{i=1}^{n} \left(\sum_{l=1}^{\bar{\lambda}+1} l \cdot x_{il} \right) = \sum_{i=1}^{n} t_i$$

Example

- Resource constraints
  - 2 ALUs; 2 Multipliers.
  - $a_1 = 2$; $a_2 = 2$.
- Single-cycle operation.
  - $d_i = 1 \ \forall i$.
- λ=4 is given as upper bound
- Objective function:

$$x_{6,1} + 2x_{6,2} + 2x_{7,2} + 3x_{7,3} + x_{8,1} + 2x_{8,2} + 3x_{8,3} + 2x_{9,2} + 3x_{9,3} + 4x_{9,4} + x_{10,1} + 2x_{10,2} + 3x_{10,3} + 2x_{11,2} + 3x_{11,3} + 4x_{11,4}$$

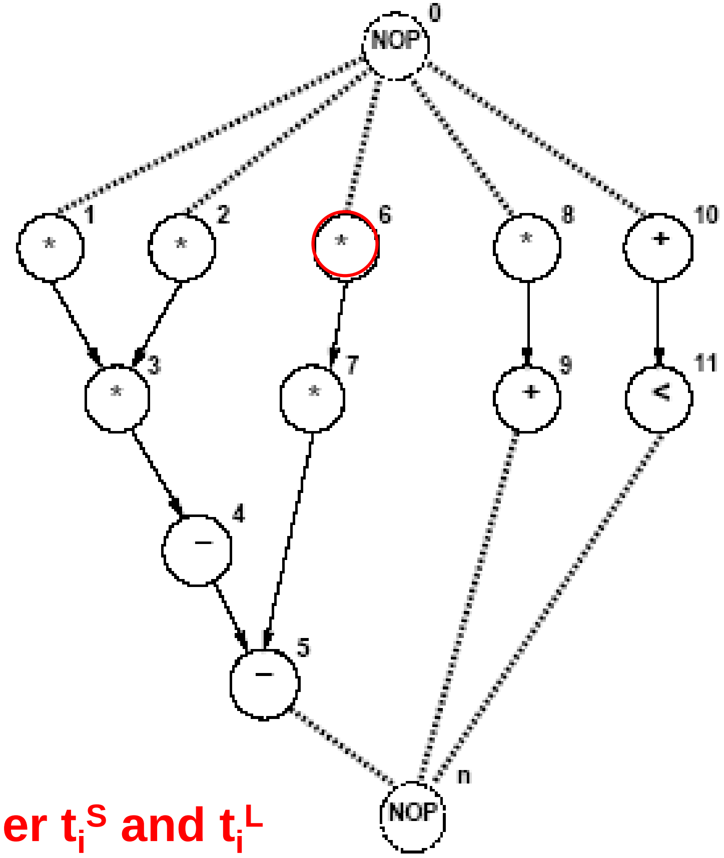


  - No need to consider $x_{1,1}+x_{2,1}+2x_{3,2}+3x_{4,3}+4x_{5,4}$ because operations 2, 2, 3, 4 and 5 have zero mobility.

Example (cont.)

- Resource constraints
  - 2 ALUs; 2 Multipliers.
  - $a_1 = 2; a_2 = 2.$
- Single-cycle operation.
  - $d_i = 1 \ \forall i.$
- Operations start only once
  - $x_{0,1}=1; x_{1,1}=1; x_{2,1}=1; x_{3,2}=1$
  - $x_{4,3}=1; x_{5,4}=1$
  - $x_{6,1}+ x_{6,2}=1$
  - $x_{7,2}+ x_{7,3}=1$
  - $x_{8,1}+ x_{8,2}+x_{8,3}=1$
  - $x_{9,2}+ x_{9,3}+x_{9,4}=1$
  - $x_{10,1}+ x_{10,2}+x_{10,3}=1$
  - $x_{11,2}+ x_{11,3}+x_{11,4}=1$
  - $x_{n,5}=1$

Consider t_i^S and t_i^L of operations (i.e. the mobility)



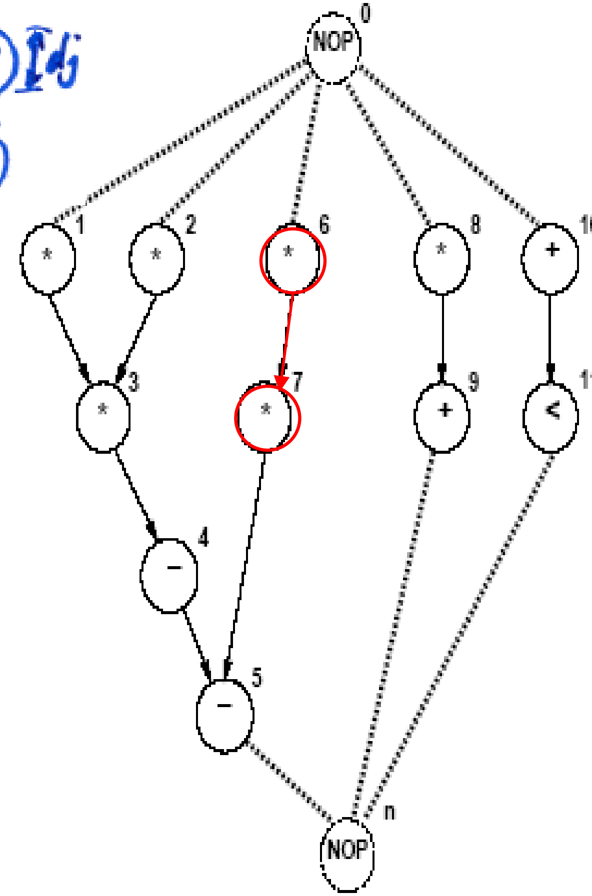
Example (cont.)

- Sequencing relations must be satisfied

Operation dependency

$t_i \geq t_j + d_j$

  - $2x_{3,2} - x_{1,1} \geq 1$
  - $2x_{3,2} - x_{2,1} \geq 1$
  - $\boxed{2x_{7,2} + 3x_{7,3}} - \boxed{x_{6,1} - 2x_{6,2}} \geq 1$
  - $2x_{9,2} + 3x_{9,3} + 4x_{9,4} - x_{8,1} - 2x_{8,2} - 3x_{8,3} \geq 1$
  - $2x_{11,2} + 3x_{11,3} + 4x_{11,4} - x_{10,1} - 2x_{10,2} - 3x_{10,3} \geq 1$
  - $4x_{5,4} - 2x_{7,2} - 3x_{7,3} \geq 1$
  - $4x_{5,4} - 3x_{4,3} \geq 1$
  - $5x_{n,5} - 2x_{9,2} - 3x_{9,3} - 4x_{9,4} \geq 1$
  - $5x_{n,5} - 2x_{11,2} - 3x_{11,3} - 4x_{11,4} \geq 1$
  - $5x_{n,5} - 4x_{5,4} \geq 1$



Example (cont.)

- Resource bounds must be satisfied:

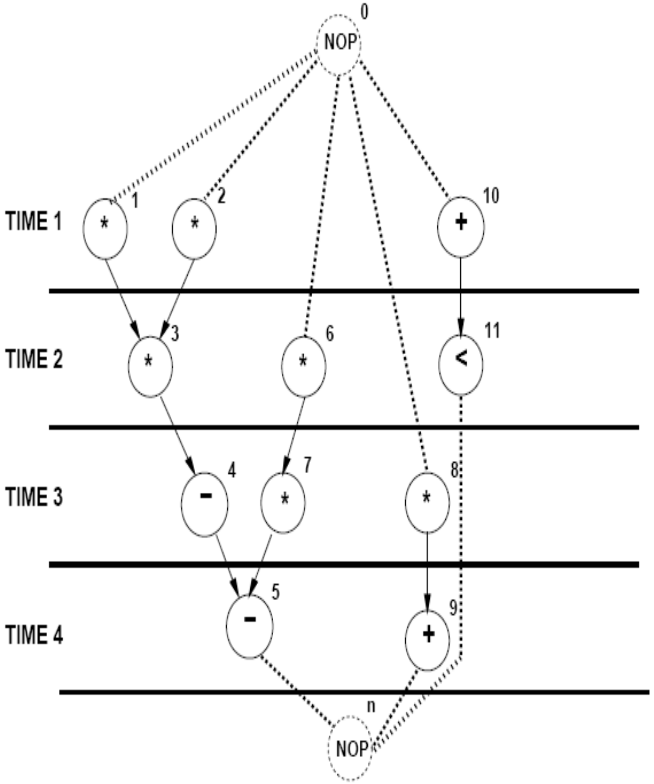
all that can be assigned to step 1.

$$x_{1,1} + x_{2,1} + x_{6,1} + x_{8,1} \le 2$$
$$x_{3,2} + x_{6,2} + x_{7,2} + x_{8,2} \le 2$$
$$x_{7,3} + x_{8,3} \le 2$$

for *'s

$$x_{10,1} \le 2$$
$$x_{9,2} + x_{10,2} + x_{11,2} \le 2$$
$$x_{4,3} + x_{9,3} + x_{10,3} + x_{11,3} \le 2$$
$$x_{5,4} + x_{9,4} + x_{11,4} \le 2$$

for +-<'s



- Any set of start times satisfying constraints provides a feasible solution.
- Any feasible solution is optimum since sink ($x_{n,5}$=1) mobility is 0.

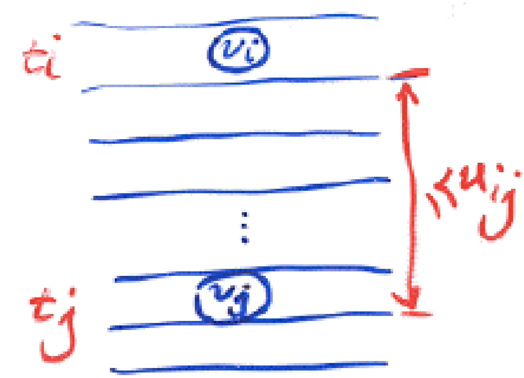
Relative Timing Constraints in ILP

- Relative timing constraints can be considered by adding new set of inequalities.
- Example:

$$t_j \le t_i + u_{ij}$$

e.g. v_j has to start at most u_{ij} steps after v_i.

$$\sum_{l=1}^{\bar{\lambda}+1} l \cdot x_{jl} \le \sum_{l=1}^{\bar{\lambda}+1} l \cdot x_{il} + u_{ij}$$



Dual ILP Formulation

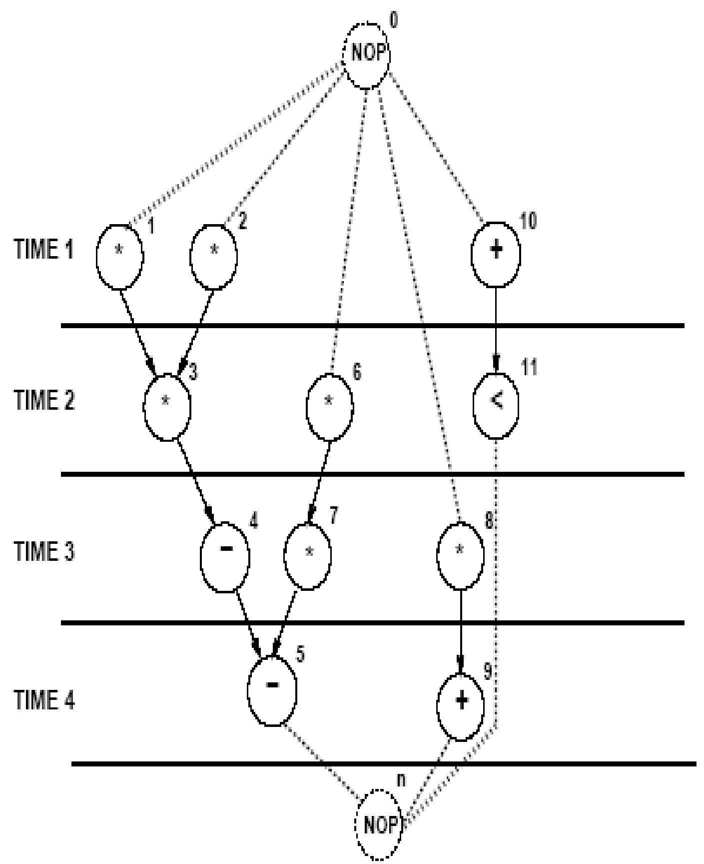
- Minimize resource usage under latency constraint.
- Same constraints as previous formulation.
- Additional constraint
  - Latency bound must be satisfied.

$$\sum_{l} l \; x_{nl} \leq \overline{\lambda} + 1$$

- Resource usage is unknown in the constraints.
- Resource usage is the objective to minimize.
  - Minimize ***c^T a***
    - ***a vector represents resource usage***
    - ***c^T vector represents resource costs***

Example

- Multiplier area = 5; ALU area = 1.
- Objective function: $5a_1 + a_2$
- Latency constraints: $\bar{\lambda} = 4$
- Start time constraints same.
- Sequencing dependency constraints same.
- Resource constraints
  - $x_{1,1}+x_{2,1}+x_{6,1}+x_{8,1} - a_1 \leq 0$
  - $x_{3,2}+x_{6,2}+x_{7,2}+x_{8,2} - a_1 \leq 0$
  - $x_{7,3}+x_{8,3} - a_1 \leq 0$
  - $x_{10,1} - a_2 \leq 0$
  - $x_{9,2}+x_{10,2}+x_{11,2} - a_2 \leq 0$
  - $x_{4,3}+x_{9,3}+x_{10,3}+x_{11,3} - a_2 \leq 0$
  - $x_{5,4}+x_{9,4}+x_{11,4} - a_2 \leq 0$



Force-Directed Scheduling

Force-Directed Scheduling

- Heuristic scheduling methods [Paulin]
  - Min latency subject to resource bound.
    - Variation of list scheduling: FDLS.
  - Min resource subject to latency bound.
    - Schedule one operation at a time.
- Rationale
  - Reward uniform distribution of operations across schedule steps.
- Operation interval: mobility plus one (μ_i+1).
  - Computed by ASAP and ALAP scheduling $[t_i^S, t_i^L]$

Force-Directed Scheduling (cont.)

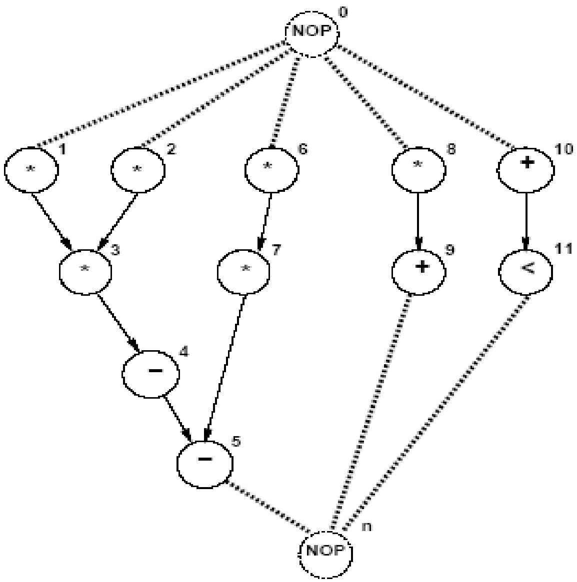
- Operation probability $p_i(l)$
  - —Probability of executing in a given step.
  - —$1/(\mu_i+1)$ inside interval; 0 elsewhere.

$$p_i(l) = \begin{cases} 0 & \text{outside of its time frame} \\ \frac{1}{\mu_i+1} & \text{inside " " " .} \end{cases}$$

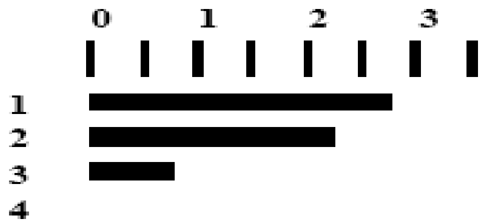
- Operation-Type Distribution $q_i(l)$
  - —Sum of the op. prob. for each type.
  - —Shows likelihood that a resource is used at each schedule step.

$$q_k(l) = \sum_{i=0}^{n} p_i(l)$$

Force-Directed Scheduling (cont.)



- Operation-type distribution $q_k(l)$
  - Sum of the op. prob. for each type.
  - Shows likelihood that a resource is used at each schedule step.
- Distribution graph for multiplier



- Distribution graph for adder

$p_1(1)=1,\ p_1(2)=p_1(3)=p_1(4)=0$
$p_2(1)=1,\ p_2(2)=p_2(3)=p_2(4)=0$
$\mu_6=1$; time frame [1,2]
$p_6(1)=0.5,\ p_6(2)=0.5,\ p_6(3)=p_6(4)=0$
$\mu_8=2$; time frame [1,3]
$p_8(1)=p_8(2)=p_8(3)=0.3,\ p_8(4)=0$
$q_{mul}(1)=1+1+0.5+0.3=2.8$

Force

- Used as priority function.
- Selection of operation to be scheduled in a time step is based on force.
- Forces attract (repel) operations into (from) specific schedule steps.
- Force is related to concurrency.
  - The larger the force the larger the concurrency
- Mechanical analogy
  - Force exerted by elastic spring is proportional to displacement between its end points.
  - Force = constant $\times$ displacement.
    - constant = operation type distribution.
    - displacement = change in probability.

Forces Related to the Assignment of an Operation

- **Self-force**
  - Sum of forces relating operation to all schedule steps in its time frame.
  - Self-force for scheduling operation v_i in step *l*

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

  - ***δ_{lm} denotes a Kronecker delta function; equal 1 when m=l.***

Constant Displacement

- **Successor-force**
  - Related to the successors.
  - Delaying an operation implies delaying its successors.

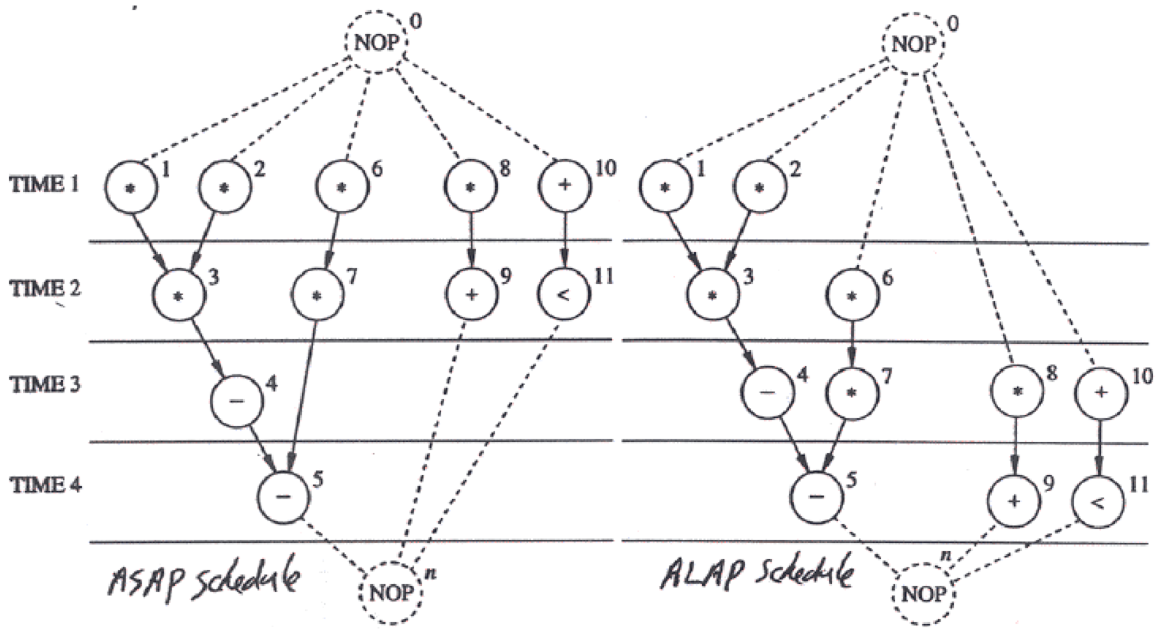
FDS Algorithm

- Analogy: All dynamic systems tend to go toward their minimum-energy (equilibrium) point

$FDS(\,G(V,E), \bar{\lambda}\,)$ {
    repeat {
        Compute the time-frames;
        Compute the operation 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 are scheduled);
    return (t);
}

Example

- Computing $P_i(l)$ and $q_i(l)$



v_i	t_i^L	t_i^S	μ_i	$P_i(l)$
1	1	1	0	1
2	1	1	0	1
3	2	2	0	1
4	3	3	0	1
5	4	4	0	1
6	2	1	1	0.5
7	3	2	1	0.5
8	3	1	2	0.33
9	4	2	2	0.33
10	3	1	2	0.33
11	4	2	2	0.33

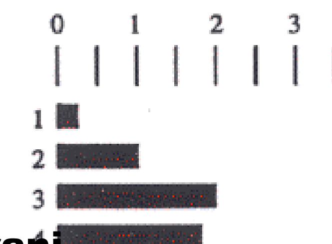
DG
For * (k=1)

v_1 v_2 v_6 v_8
$1+1+0.5+0.33$

$q_1(1)=2.8$
$q_1(2)=2.3$
$q_1(3)=0.8$
$q_1(4)=0$

DG
For +−< (k=2)

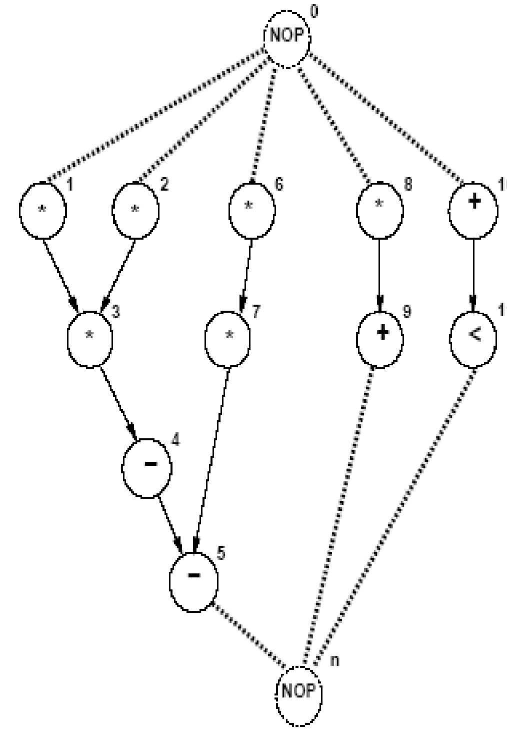
$q_2(1)=0.33$
$q_2(2)=1$
$q_2(3)=2$
$q_2(4)=1.66$



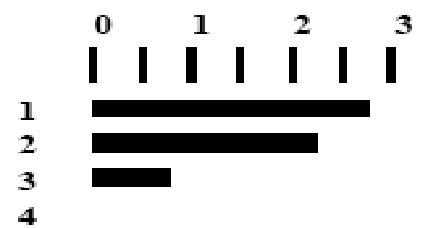
$1+0.33+0.33$
v_5 v_9 v_{11}

Example: Operation v_6

- It can be scheduled in the first two steps.
  - p(1) = 0.5; p(2) = 0.5; p(3) = 0; p(4) =0.
- Distribution: q(1) = 2.8; q(2) = 2.3; q(3)=0.8.



- Assign v_6 to step 1
  - variation in probability 1 – 0.5 = 0.5 for step1
  - variation in probability 0 – 0.5 = -0.5 for step2
  - Self-force: 2.8 * 0.5 + 2.3 * -0.5 = +0.25
- Assign v_6 to step 2
  - variation in probability 0 – 0.5 = -0.5 for step1
  - variation in probability 1 – 0.5 = 0.5 for step2
  - Self-force: 2.8 * -0.5 + 2.3 * 0.5 = -0.25



Example: Operation v6 (cont.)

- Successor-force
  - Assigning v_6 to step 2 implies operation v_7 assigned to step 3.
  - 2.3 (0-0.5) + 0.8 (1 -0.5) = -.75
  - Total-force on v_6 = (-0.25)+(-0.75)=-1.
- Conclusion
  - Least force is for step 2.
  - Assigning v_6 to step 2 reduces concurrency (i.e. resources).
- Total force on an operation related to a schedule step
  - = self force + predecessor/successor forces with affected time frame

$$ps\text{-}force(i,l)_j = \frac{1}{\tilde{\mu}_j + 1} \sum_{m=t_j^{\tilde{S}}}^{m=t_j^{\tilde{L}}} q_k(m) - \frac{1}{\mu_j + 1} \sum_{m=t_j^{S}}^{m=t_j^{L}} q_k(m)$$

Example: Operation v6 (cont.)

- Assignment of v_6 to step 2 makes v_7 assigned at step 3
  - Time frame change from [2, 3] to [3, 3]
  - Variation on force of $v_7 = 1*q(3) - ½ * (q(2)+q(3))$
    $= 0.8-0.5(2.3+0.8)= -0.75$
- Assignment of v_8 to step 2 makes v_9 assigned to step 3 or 4
  - Time frame change from [2, 3, 4] to [3, 4]
  - Variation on force of $v_9 = 1/2*(q(3)+q(4)) - 1/3 * (q(2)+q(3)+q(4)) = 0.5*(2+1.6)-0.3*(1+2+1.6)=0.3$

FDS - Minimum Resources

- Operations considered one a time for scheduling
- For each iteration
  - Time frames, probabilities and forces computed
  - Operation with least force scheduled

$FDS(\ G(V,E), \overline{\lambda}\ )\ \{$

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

FDS - Minimum Latency under Resource Constraints

- Outer structure of algorithm same as LIST-L.
- Selected candidates determined by
  - Reducing iteratively candidate set $U_{l,k}$.
  - Operations with least force are deferred.
  - Maximize local concurrency by selecting operations with large force.
  - At each outer iteration of loop, time frames updated.

```
LIST_L( G(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, s.t. |S_k| + |T_{l,k}| ≤ a_k;
            Schedule the S_k operations at step l;
        }
        l = l + 1;
    }
    until (v_n is scheduled) ;
    return (t);
}
```