Instructor : Haidar M. Harmanani

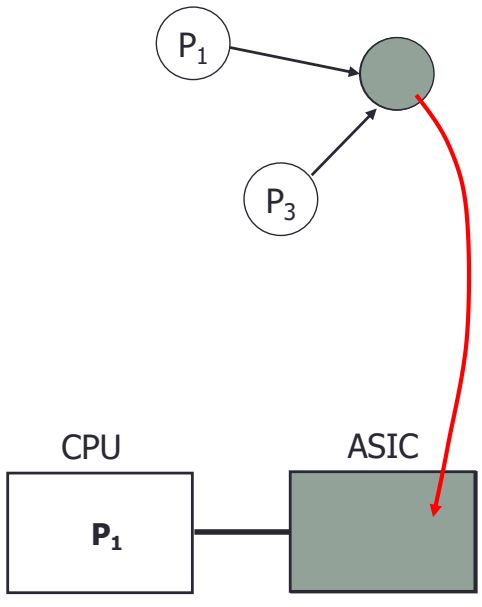
Fall 2020

VLSI DESIGN AUTOMATION
HIGH-LEVEL SYNTHESIS

LAU Computer Science

Hardware Synthesis

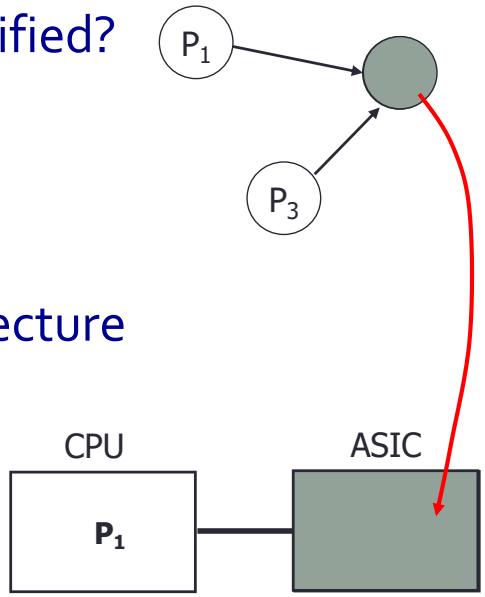
- Starts from an abstract behavioral description
- Generates an RTL description
- Need to restrict the target hardware – otherwise search space is too large



LAU Computer Science

Hardware Synthesis

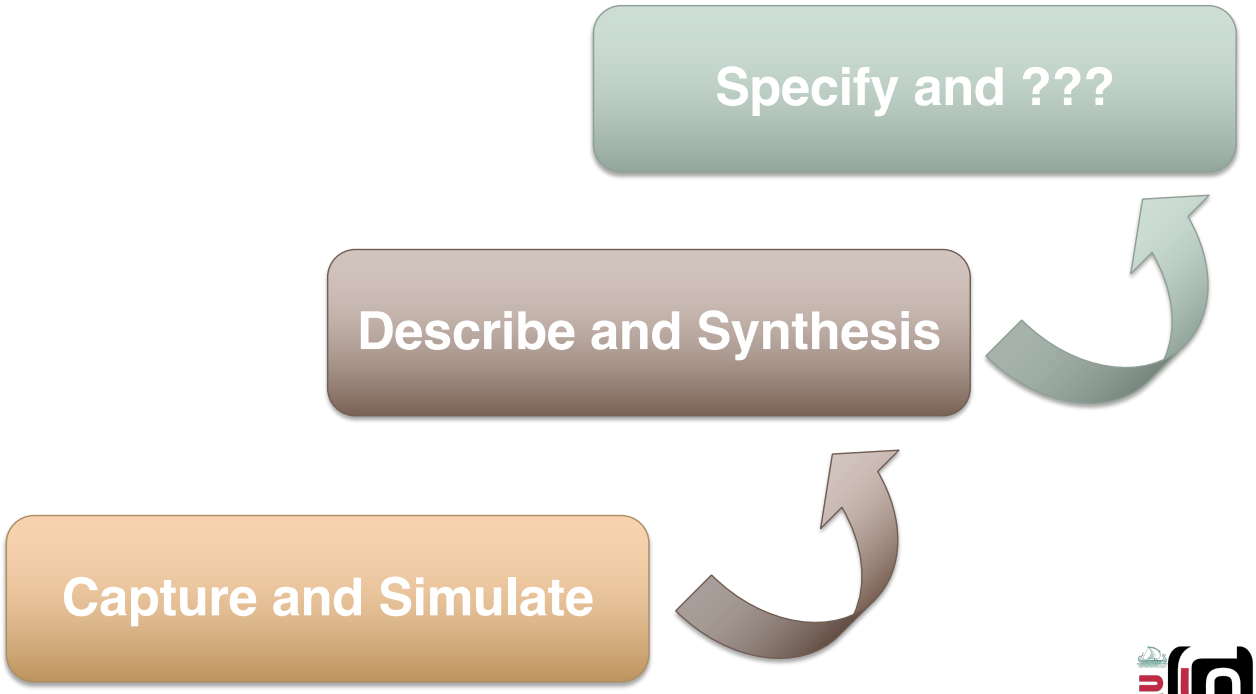
- How is the behavior specified?
  - Natural languages
  - C/C++
  - VHDL/Verilog
- What is the target architecture of the ASIC?



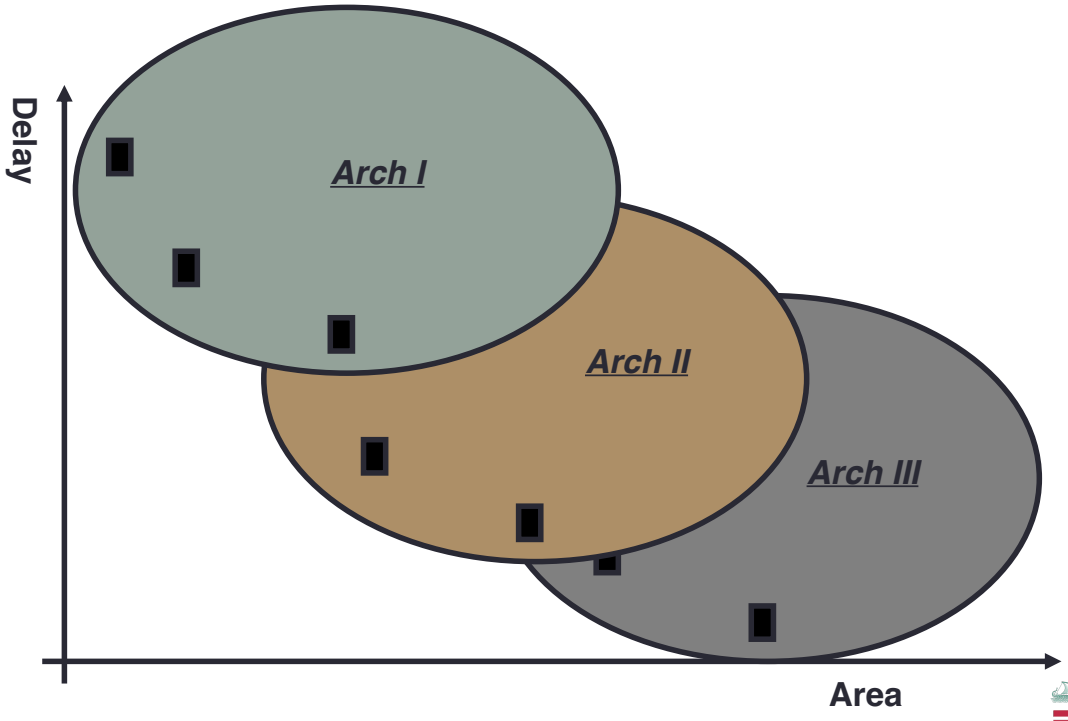
VLSI Design Tools

- Design Capturing/Entry
- Analysis and Characterization
- Synthesis/Optimization
  - Physical (Floor planning, Placement, Routing)
  - Logic (FSM, Retiming, Sizing, DFT)
  - High Level(RTL, Behavioral)
- Management

Design Methodology Progress



Design Space Exploration

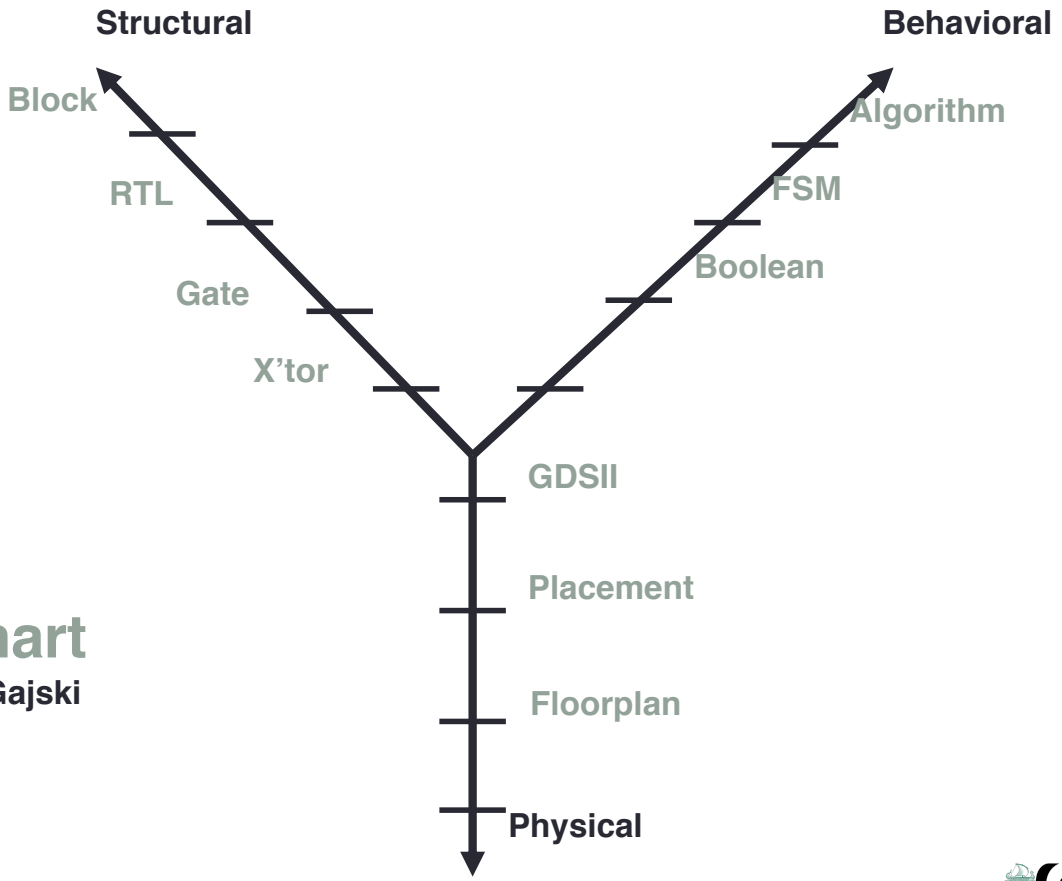


Why Synthesis?

- Productivity
- Correctness
- Re-Targetability

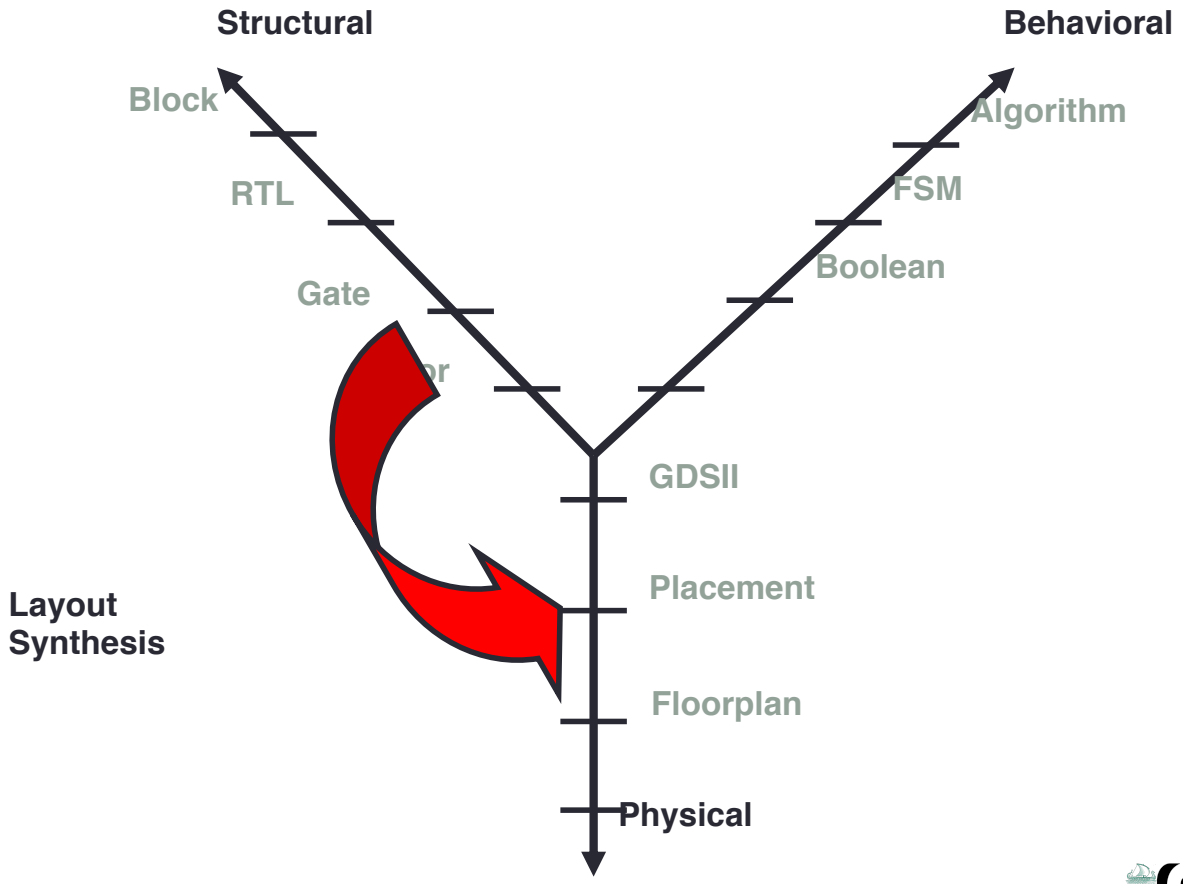
Why not Synthesis?

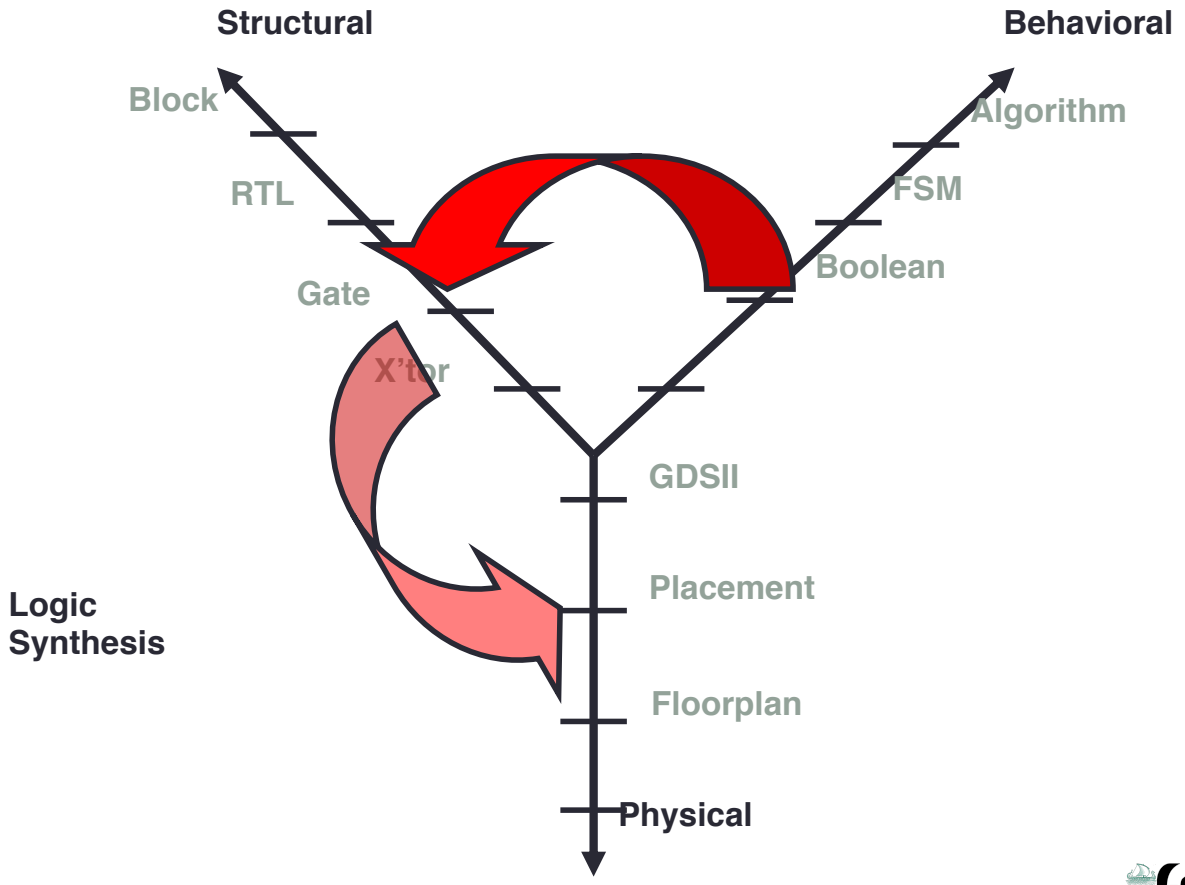
- Performance Loss
- Unsynthesizability
- Inertial

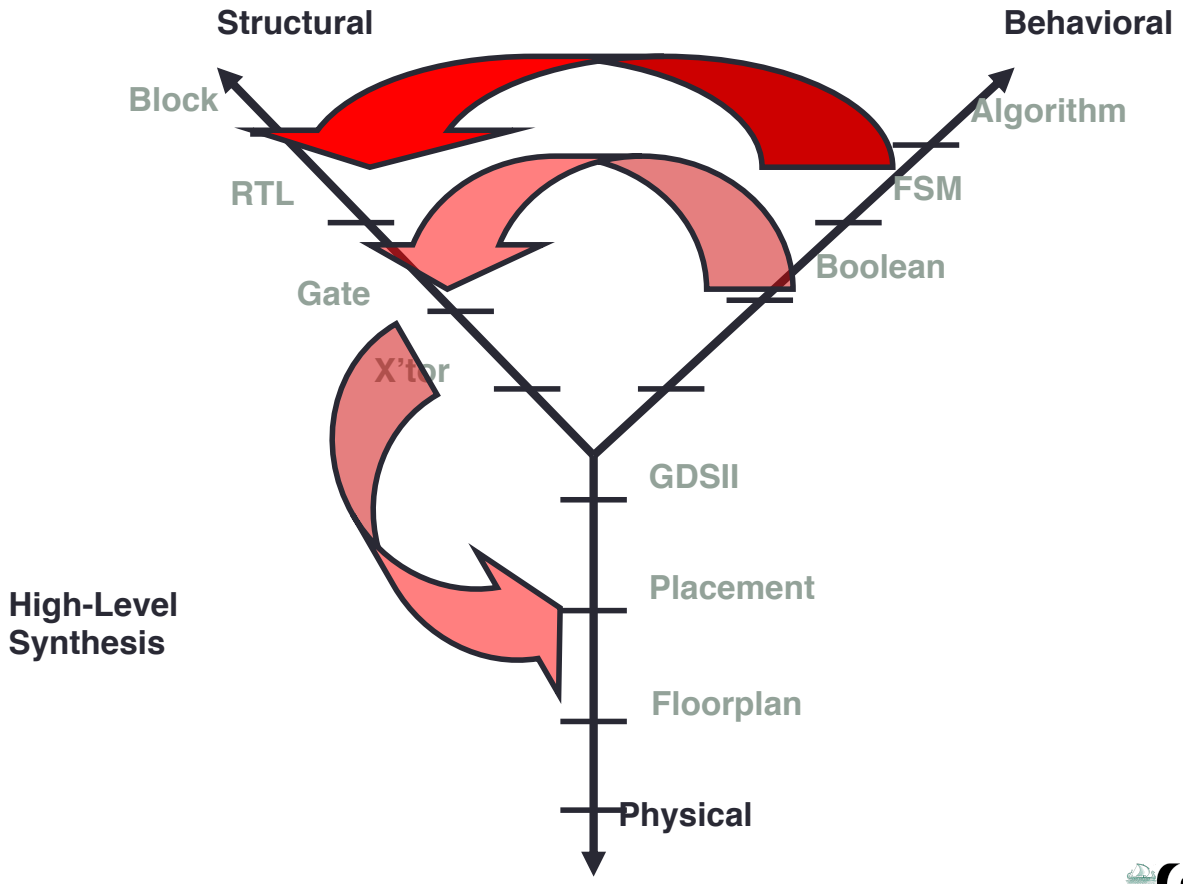


Y-Chart

Dan D Gajski


Structural
Behavioral
Block
RTL
Gate
X'tor
Algorithm
FSM
Boolean
GDSII
Placement
Floorplan
Physical
Layout
Synthesis

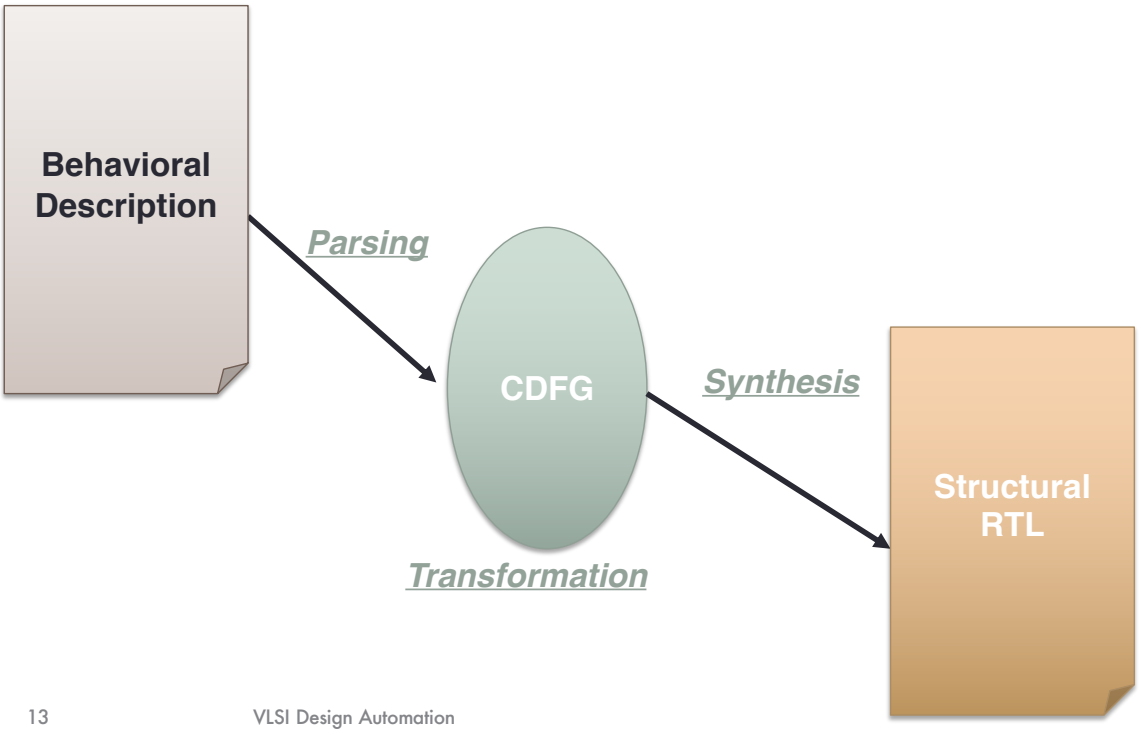

Structural
Behavioral
Block
RTL
Gate
X'tor
Algorithm
FSM
Boolean
GDSII
Placement
Floorplan
Physical
Logic
Synthesis



High Level Synthesis (HLS)

- The process of converting a high-level description of a design to a netlist
  - Input:
    - Behavioral description of a system
    - A set of constraints
      - Area constraints (e.g., # modules of a certain type)
      - Delay constraints (e.g., set of operations should finish in l clock cycles)
  - Output:
    - Operation scheduling (time) and binding (resource)
    - Control generation and detailed interconnections

High Level Synthesis



What Went Wrong?

- Too much emphasis on incremental work on algorithms and point tools
- Unrealistic assumption on component capability, architectures, timing, etc
- Lack of quality-measurement from the low level
- Too much promising on fully automation (silicon compiler??)

Essential Issues

- Behavioral Specification Languages
- Target Architectures
- Intermediate Representation
- Operation Scheduling
- Allocation/Binding
- Control Generation

Behavioral Specification Languages

- Add hardware-specific constructs to existing languages
  - HardwareC
- Popular HDL
  - Verilog, VHDL
- Synthesis-oriented HDL
  - UDL/I

Target Architectures

- Bus-based
- Multiplexer-based
- Register file
- Pipelined
- RISC, VLIW
- Interface Protocol

Hardware Model

- Data path
  - Network of functional units, registers, multiplexers and buses
- Control
  - Takes care of having the data present at the right place at a specific time
  - Takes care of presenting the right instructions to a programmable unit
- Often high-level synthesis concentrates on data path synthesis

Hardware Model - Components

- Most synthesis systems are targeted towards synchronous hardware
- Functional units:
  - Can perform one or more computations
  - Addition, multiplication, comparison, ALU, etc.
- Registers:
  - Store inputs, intermediate results and outputs
  - May be organized as a register file

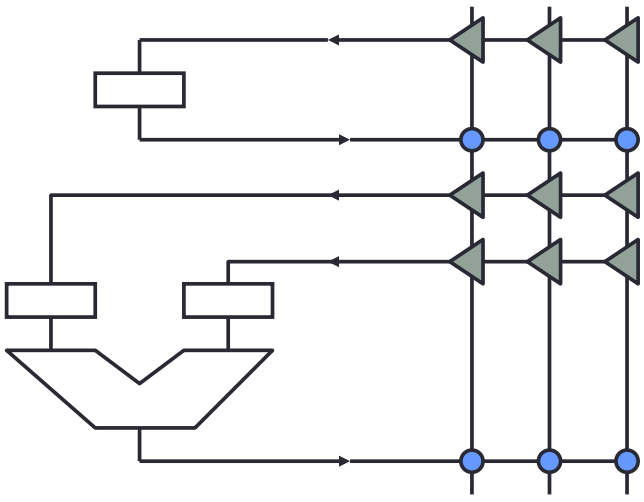
Hardware Model - Interconnection

- Multiplexers:
  - Select one output from several inputs
- Busses:
  - Connection shared between several components
  - Only one component can write data at a specific time
  - Exclusive writing may be controlled by tri-state drivers

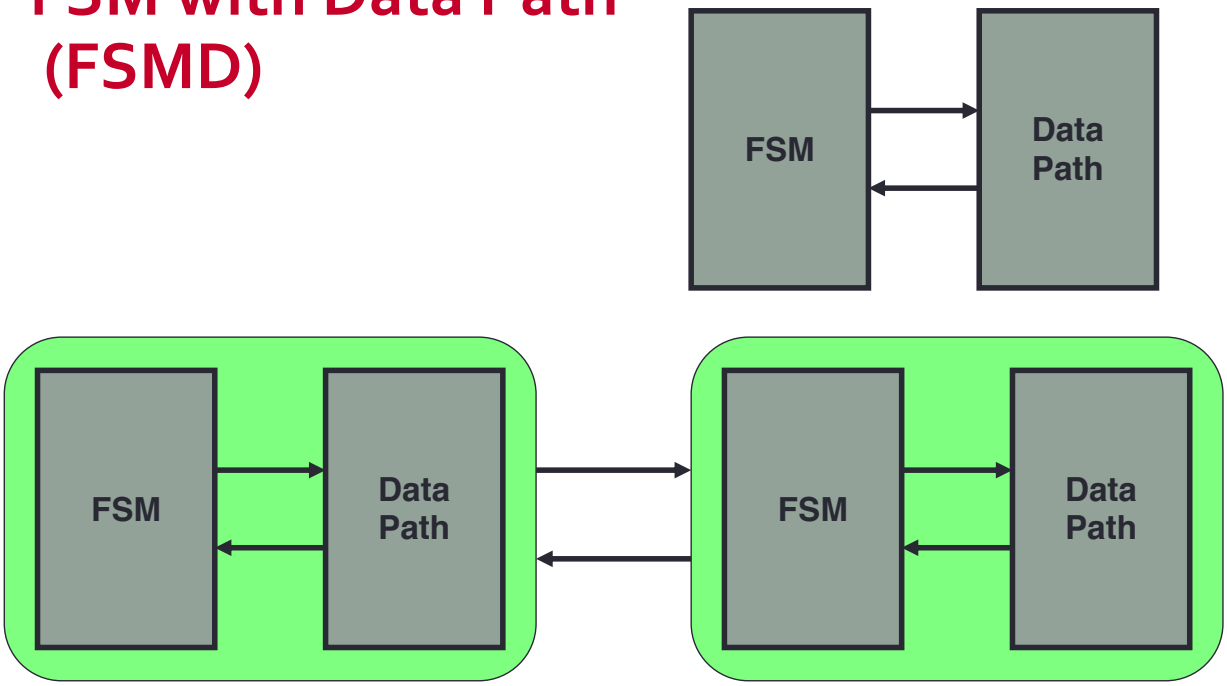
Hardware Model – Parameters

- Clocking strategy
  - Single or multiple phase clocks
- Interconnect
  - Allowing or disallowing busses
- Clocking of functional units
  - Multicycle operations
  - Chaining
  - Pipelined units

Hardware Model – Example

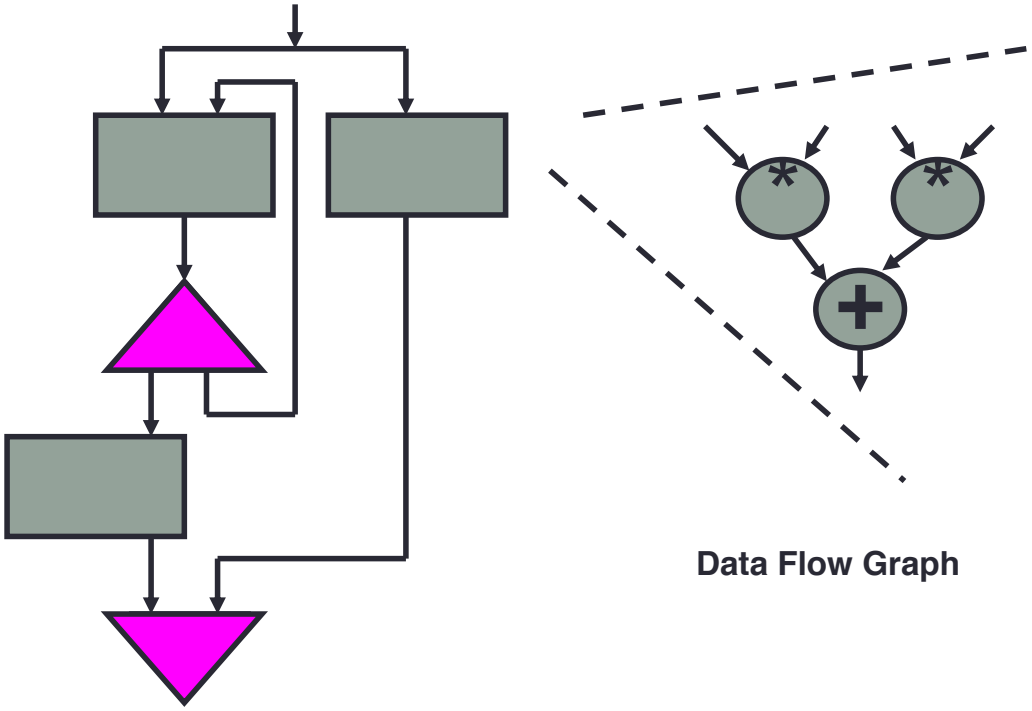


FSM with Data Path (FSMD)



Interactive FSMDs

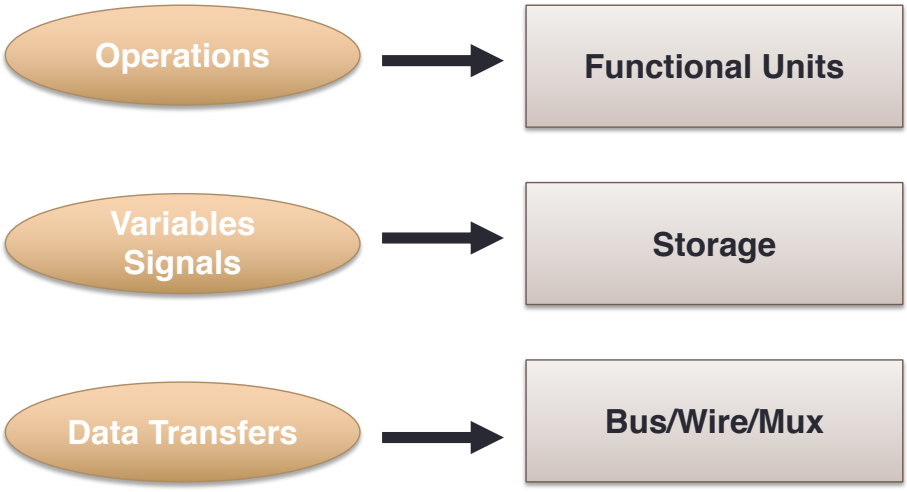
Intermediate Representation



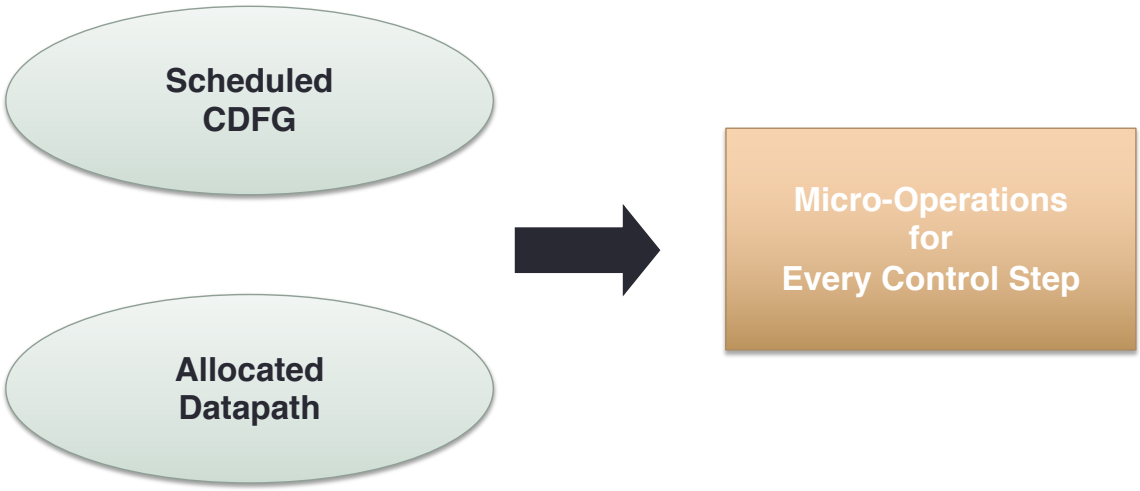
Scheduling (Temporal Binding)

- Time & Resource Tradeoff
- Time-Constrained
  - Integer Linear Programming (ILP)
  - Force-Directed
- Resource-Constrained
  - List Scheduling
- Other Heuristics
  - Simulated Annealing, Tabu Search, …

Allocation/Binding



Controller Specification Generation



HLS Quality Measures

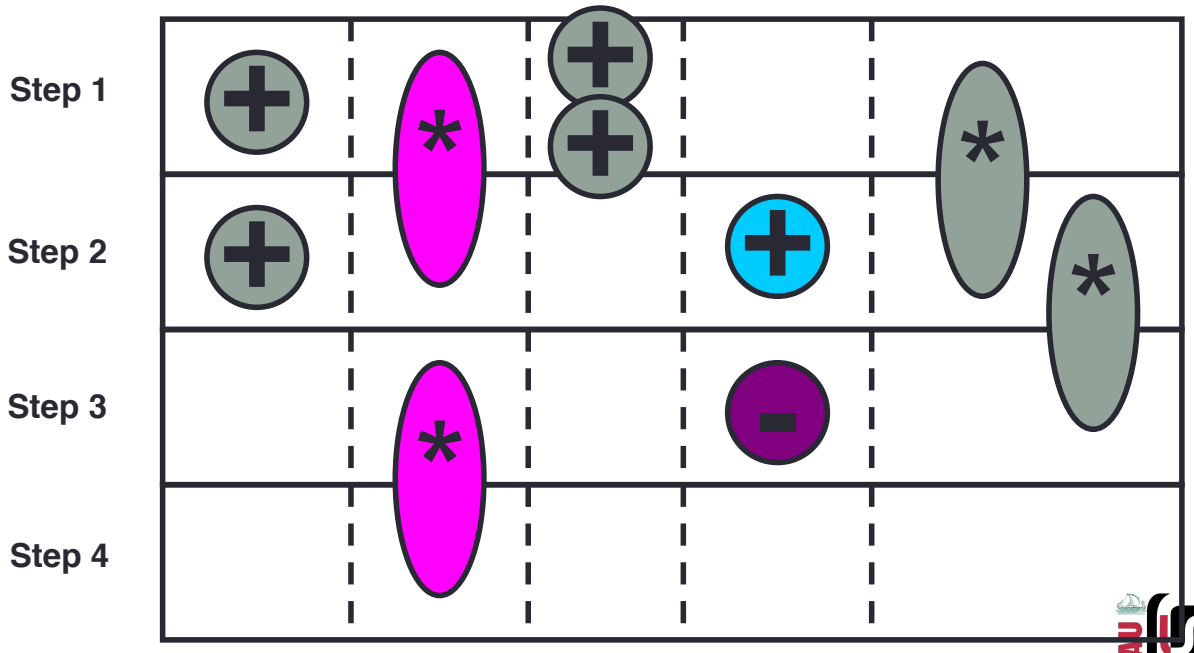
- Performance
- Area Cost
- Power Consumption
- Testability
- Reusability

Hardware Variations

- Functional Units
  - Pipelined, Multi-Cycle, Chained, Multi-Function
- Storage
  - Register, RF, Multi-Ported, RAM, ROM, FIFO, Distributed
- Interconnect
  - Bus, Segmented Bus, Mux, Protocol-Based

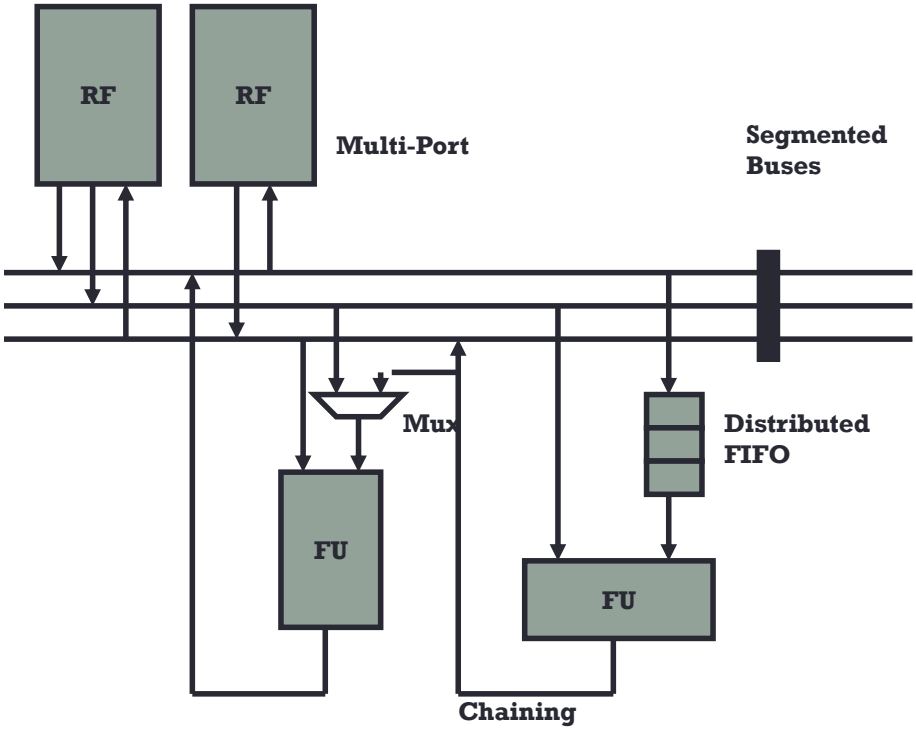
LAU Computer Science

Functional Unit Variations



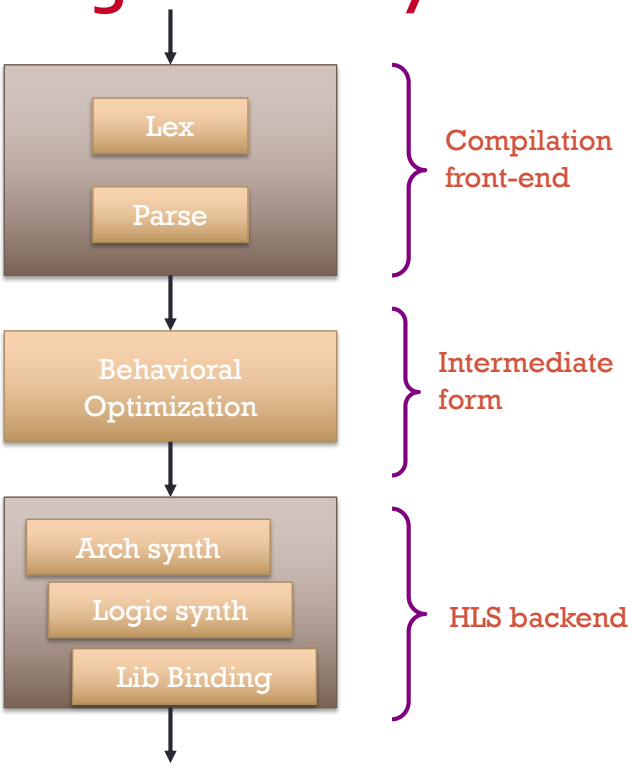
LAU Computer Science

Storage/Interconnect Variations

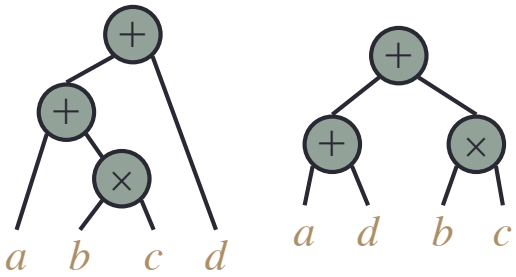


PART I: SYNTHESIS

High-Level Synthesis Compilation Flow

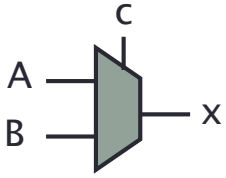


$$x = a + b \times c + d$$



Behavioral Optimization

- Techniques used in software compilation
  - Expression tree height reduction
  - Constant and variable propagation
  - Common sub-expression elimination
  - Dead-code elimination
  - Operator strength reduction (e.g., *4 → << 2)
- Typical Hardware transformations
  - Conditional expansion
    - If (c) then x=A else x=B
      → compute A and B in parallel, x=(C)?A:B
  - Loop expansion
    - Instead of three iterations of a loop, replicate the loop body three times



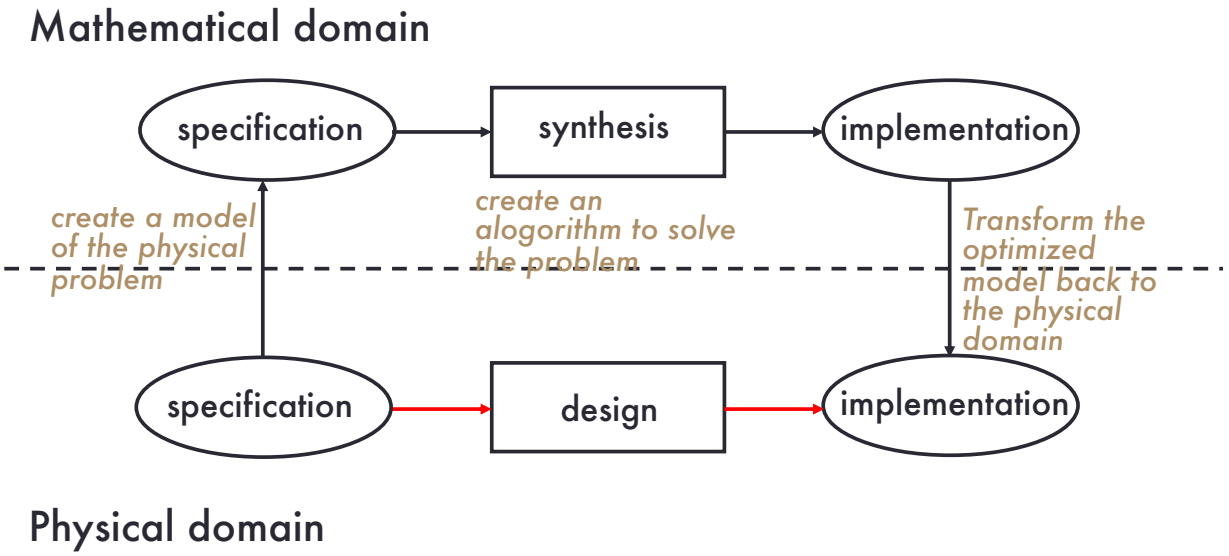
Architectural Synthesis

- Deals with "computational" behavioral descriptions
  - Behavior as sequencing graph data flow graph DFG)
  - Hardware resources as library elements
    - Pipelined or non-pipelined
    - Resource performance in terms of execution delay
  - Constraints on operation timing
  - Constraints on hardware resource availability
- Objective
  - Generate a synchronous, single-phase clock circuit
  - Might have multiple feasible solutions (explore tradeoff)
  - Satisfy constraints, minimize objective:
    - Maximize performance subject to area constraint
    - Minimize area subject to performance constraints

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?)
  - Evaluate (estimate) cost functions
- Unconstrained optimization problems for resource dominated circuits:
  - Min area: solve for minimal binding
  - Min latency: solve for minimum λ scheduling

Synthesis Methodology



Input Format

- Input
  - Behavior described in textual form
  - Conventional programming language
  - Hardware description language (HDL)
- Has to be parsed and transformed into an internal representation
- Conventional compiler techniques are used

Internal Representation

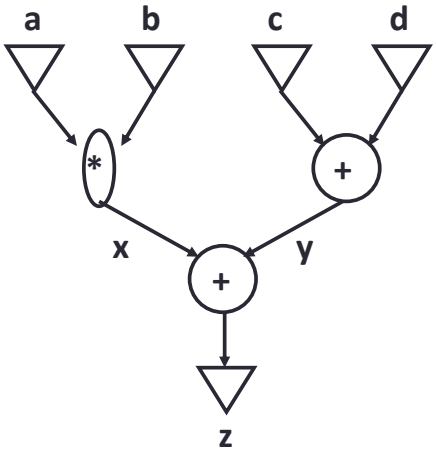
- Data-flow graph (DFG)
  - Used by most systems
  - May or may not contain information on control flow

vertex (node): represent computation

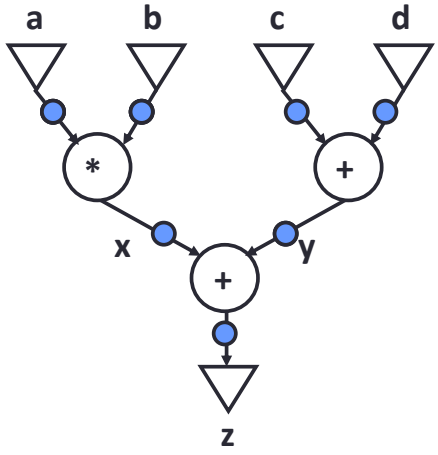
edge: represents *precedence* relations

Data Flow

```
x := a * b;
y := c + d;
z := x + y;
```



DFG Semantics

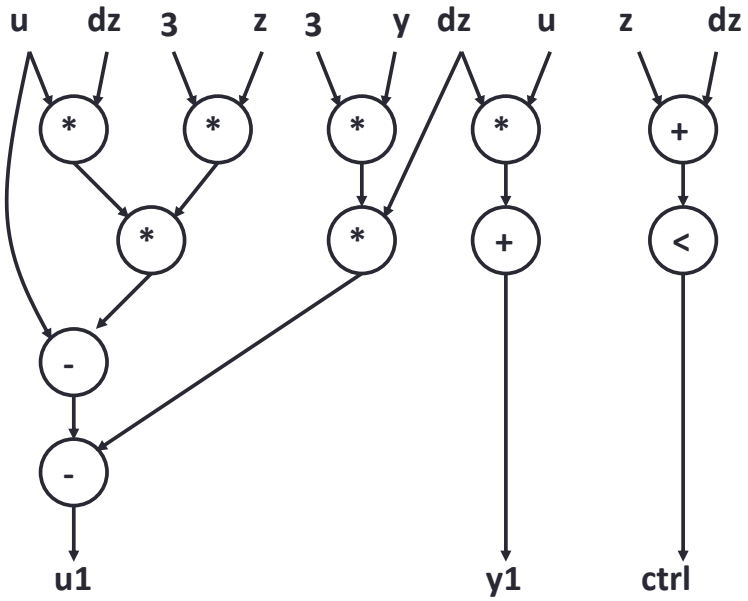


Example – Data Flow Graph of DiffEq

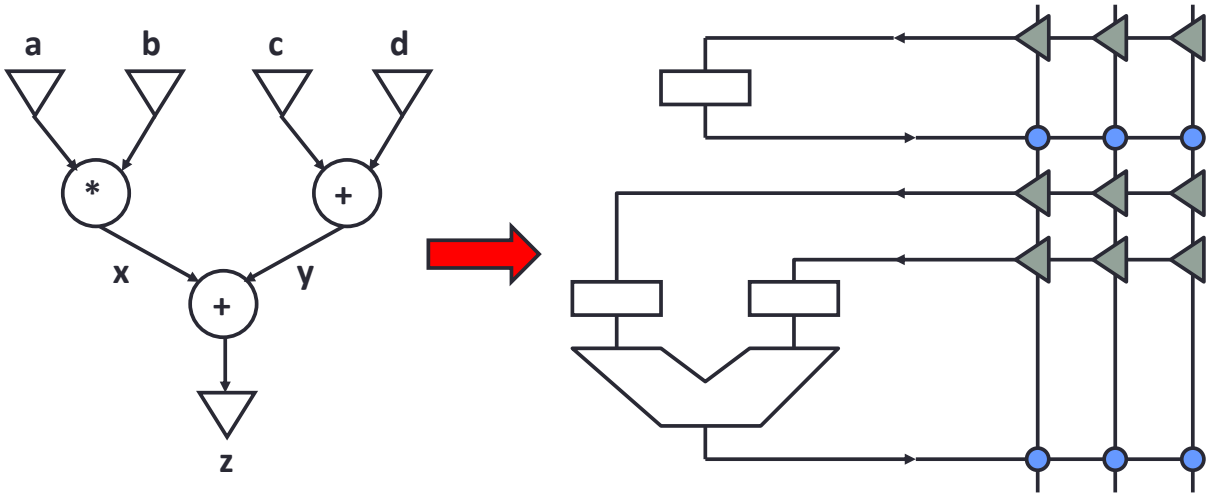
- Solve the second order differential equation
  - $y'' + 3zy' + 3y = 0$
- Iterative solution

```
While (z<a) {
    z1 := z + dz;
    u1 := u – (3*z*u*dz) – (3*y*dz);
    y1 := y + (u*dz);
    z := z1; u := u1; y := y1;
}
```

Example - Result



High-Level Synthesis



High-Level Synthesis Tasks

- Scheduling
  - Determine for each operation the time at which it should be performed such that no precedence contraint is violated
- Allocation
  - Specify the hardware resources that will be necessary
- Assignment
  - Provide a mapping from each operation to a specific functional unit and from each variable to a register

High-Level Synthesis Tasks

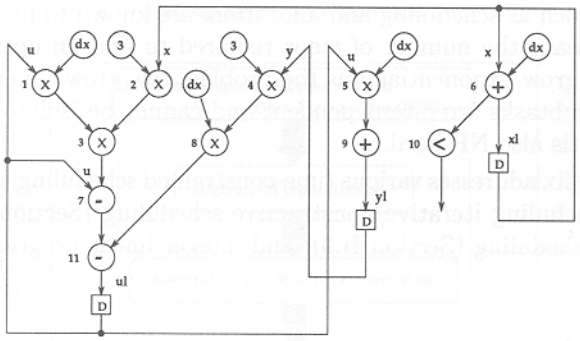
- Scheduling, allocation and assignment are strongly interrelated
  - Sometimes solved together but often separately!
- Scheduling is NP-complete
  - Heuristics have to be used!
- Datapath allocation involves various tasks that are also NP-complete

PART II: SCHEDULING

Algorithm Description → Data Flow Graph

$$y'' + 3xy' + 3y = 0 \qquad u = y' = \frac{dy}{dx}$$

$$\frac{du}{dx} = y'' = \frac{d^2y}{dx^2} = -3xy' - 3y = -3xu - 3y$$



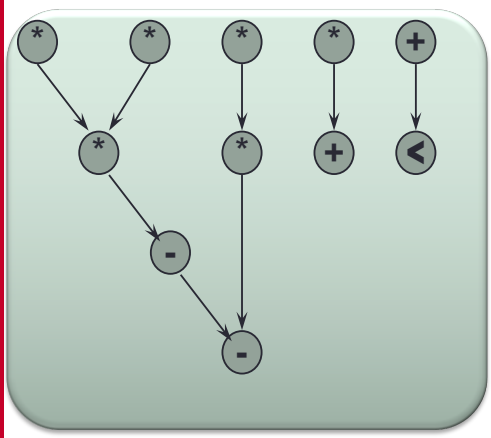

```
while (x < a) {
        xl = x + dx;
        ul = u - (3 * x * u * dx) - (3 * y * dx);
        yl = y + u * dx;
        x = xl; y = yl; u = ul;
}
```

Scheduling

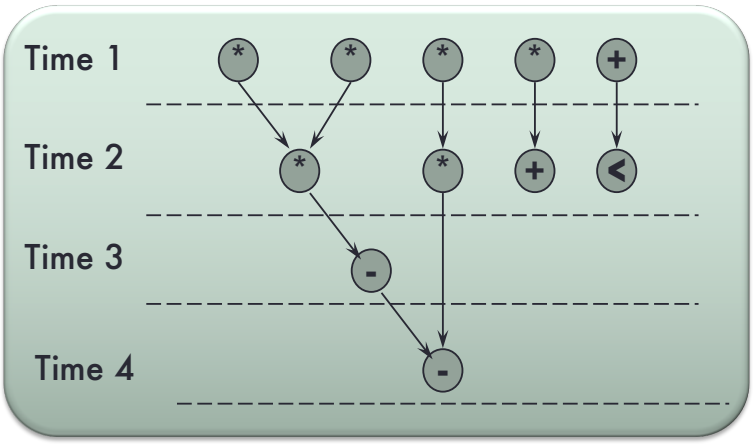
- Definition
  - Determine start times for behavioral operators while satisfying timing, power, testability, and/or resource constraints
- Input
  - Control DFG
  - Constraints
    - Cycle Time
    - Operations delays expressed in cycles
- Output
  - Temporal ordering of individual operations
- Goal
  - Exploit parallelism to achieve fastest design while meeting contsraints
    - Area/latency trade-off

LAU Computer Science

Example 1



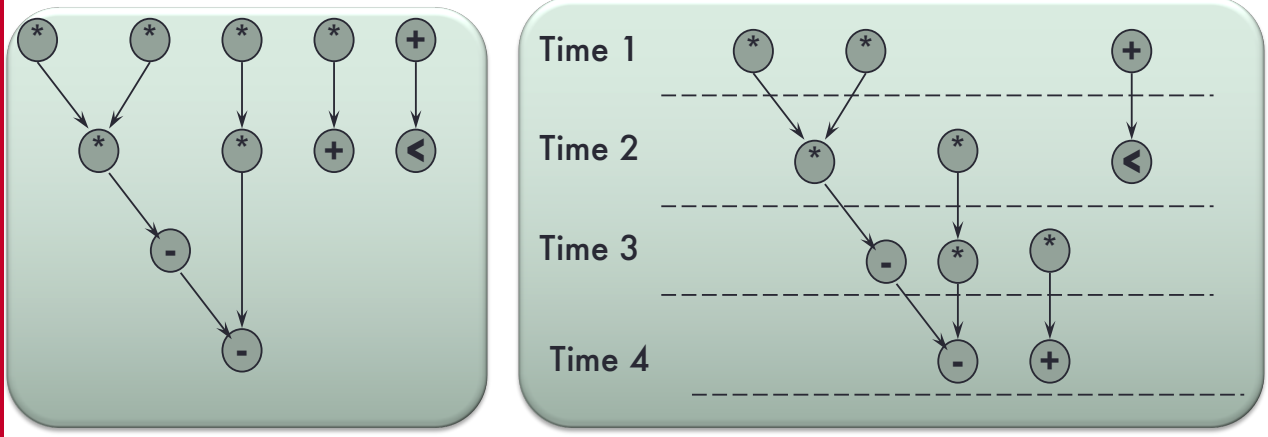
Input DFG



Scheduled DFG

LAU Computer Science

Example 2



Input DFG

Scheduled DFG

Taxonomy

- Unconstrained Scheduling
- Scheduling with timing constraints
  - Latency
  - Detailed Timing Constraints
- Scheduling with resource constraints
- Related Issues
  - Pipelining
  - Chaining
  - Multicycling
  - Synchronization

Minimum-Latency Unconstrained Scheduling

- Problem Description
  - Given a set of operations with delays D and a partial order on the operations E, find an integer labeling $\varphi: V \rightarrow Z^+$ of the operations such that:
    - $t_i = \varphi(v_i)$ and $t_i \geq t_j + d_j$
    - t_n is minimum
- Simplest model
  - Operations have bounded delays in cycles
  - No constraints or bounds on area
- Goal
  - Minimize latency

ASAP Scheduling

```
ASAP(G(V, E)) {
        Schedule v0 by setting t0 = 1
        do {
                Select a vertex vi whose predessors are all scheduled
                Schedule vi by setting ti = max tj + dj
        } while (vn is not scheduled
        return(t)
}
```

ASAP Scheduling Algorithm

Schedule an operation O_i into the earliest possible control step

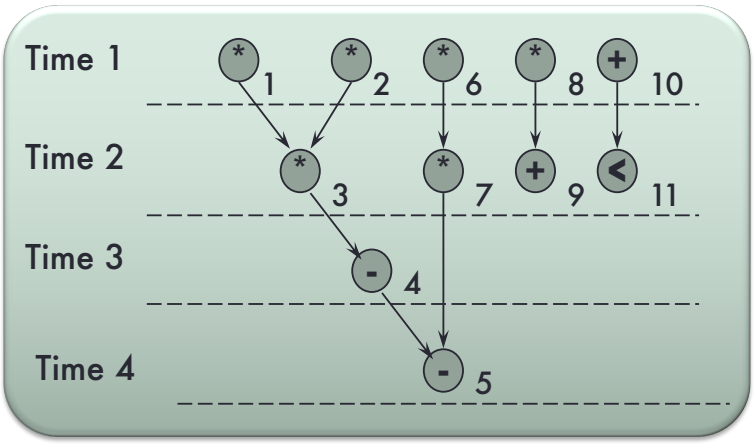
```
for each v_i ∈ V do
    if Pred = ϕ {
        E_i = 1
        V = V - {v_i}
    else
        E_i = 0
    }

While V ≠ ϕ do
    for each node v_i ∈ V {
        if all_pred_scheduled(v_i, E) + 1 {
            Ei = max(pred_vi, E) + 1
            V = V - {v_i}
        }
    }
```

Assign the nodes that do not have any predessors to state s1 and the rest to state s0

Determine the nodes that have all their predessors scheduled and assign them to the earliest possible state, max ($Pred_{vi}$, E) + 1

LAU Computer Science

Example



ASAP DFG

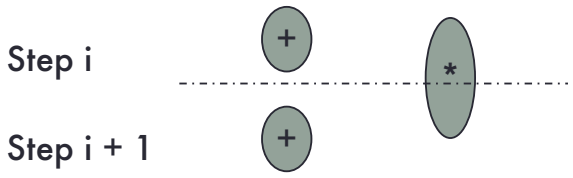
LAU Computer Science

Related Issues: Multicycle Operations

- Real functional units have different propagation delays based on their design
  - A floating point adder is slower than a fixed point adder
- Operations may not finish in one time step
  - ⇒ Increase clock cycle to accommodate slowest design unit
    - Slow units, with propagation delay shorter than the clock cycle, remain idle during part of the clock cycle
  - ⇒An alternative is to shorter the clock period to allow fast operations to execute in one clock cycle
    - Slower operations, *multicycle operations*, are scheduled across two or more control steps

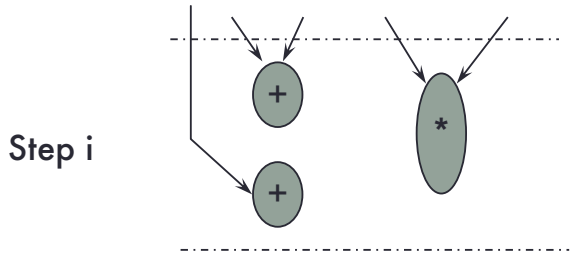
Related Issues: Multicycle Operations

- Tradeoff
  - Faster latency and Shorter clock cycle
  - Need latches to hold operands in front of the multicycle units to hold operands until the result is available in the next clock cycle(s)
  - Larger number of control steps
    - Bigger controller



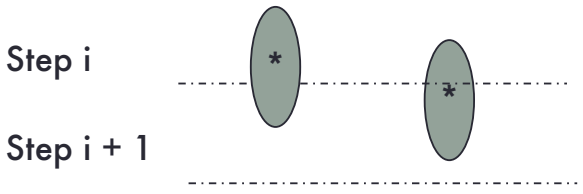
Related Issues: Chaining

- Allow two or more operations to be performed serially within one step
- Result is fed directly to the input of some other Functional unit



Related Issues: Pipelining

- Another alternative is to use pipelined functional units
  - An effective technique for increasing Parallelism
  - Two multipliers can share the same two stage pipelined multiplier despite the fact that the two operations are executing concurrently
    - Each multiplier uses a different stage of the pipelined multiplier
      - One pipelined multiplier is needed instead of two



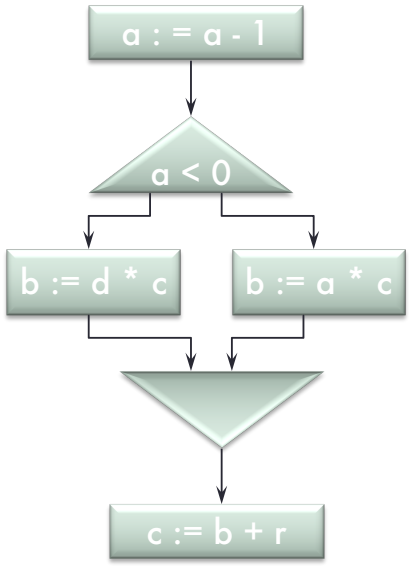
Related Issues: Conditionals

- Conditionals result in several branches that are mutually exclusive
  - During execution, only one branch gets executed based on the outcome of an evaluated condition
- An effective scheduling algorithm shares resources among mutual exclusive operations

LAU Computer Science

Related Issues: Conditionals

```
a := a - 1;
if (a < 0)
   b := d * c;
else
   b := a * c;
c := b + r;
```



- Schedule both multiplications in the same clock cycle even if one multiplier is available in each step

LAU Computer Science

ALAP Scheduling

```
ALAP(G(V, E), "λ) {
        Schedule v_n by setting t_n = λ + 1
        do {
            Select vertex v_i whose successors are all scheduled
            Schedule vi by setting ti = min t_i - d_i
        } while (v_0 is not scheduled)
        return(t)
}
```


LAU Computer Science

ALAP Scheduling Algorithm

```
for each node v_i ∈ V {
  if succ_vi = Φ {
    L_i = T;
    V = V - {v_i}
  else
    L_i = 0
  }
while V ≠ Φ {
  for each node v_i ≠ V {
    if all_pred_sched(v_i, L) {
      L_i = min(succ_vi, L) - 1;
      V = V - {v_i};
    }
  }
}
```

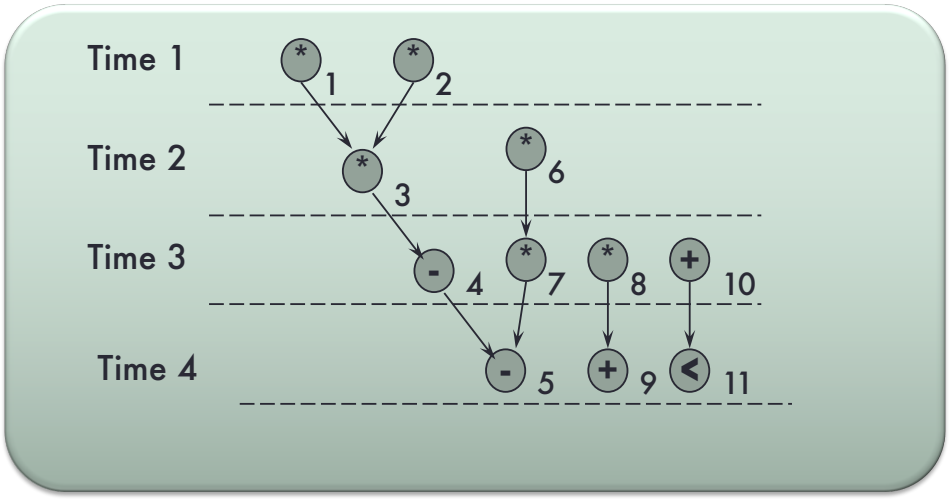
Assign the nodes that do not have any successors to the last possible state

Determine the nodes that have all their predessors scheduled and assign them to the latest possible state

Question: How to get T?

LAU Computer Science

Example



ALAP DFG

General Remarks

- ALAP solves a latency-constrained problem
  - Latency bound can be computed by ASAP
- Given a schedule, can we determine the number of functional units required to implement the design?

Mobility

- Mobility
  - Defined for each operation as the difference between ALAP and ASAP schedules
  - Operations with zero mobility are operations on the critical path ($\{v_1, v_2, v_3, v_4, v_5\}$)
  - Operations with mobility one are $\{v_6, v_7\}$
  - Operations with mobility two are $\{v_6, v_9, v_{10}, v_{11}\}$
- Question
  - How can we use mobility to improve our scheduling algorithms?

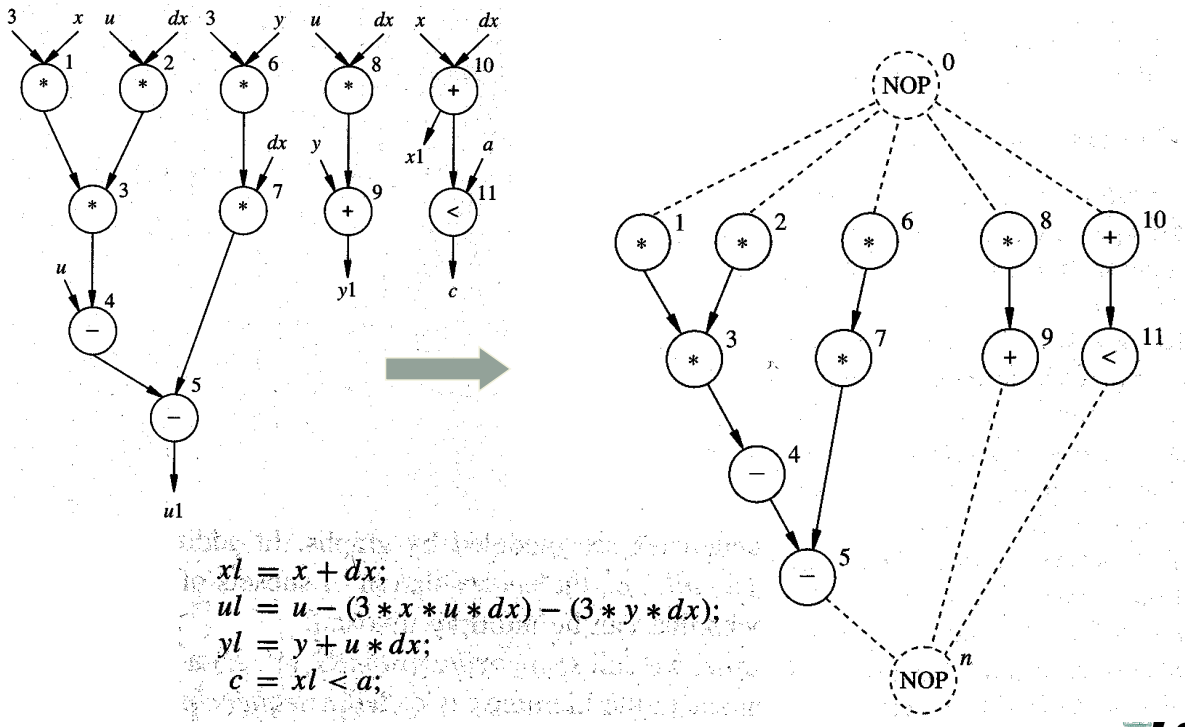
LAU Computer Science

Detailed Timing Constraints Scheduling

- Motivation
  - Control over operation start time
- Procedure
  - Start with a CDFG
  - 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
    - Edge (v_j, v_i) with weight $-u_{ij} \Rightarrow t_j \leq t_i + u_{ij}$ since $t_j \leq t_i + u_{ij} \Rightarrow t_i \geq t_j - u_{ij}$

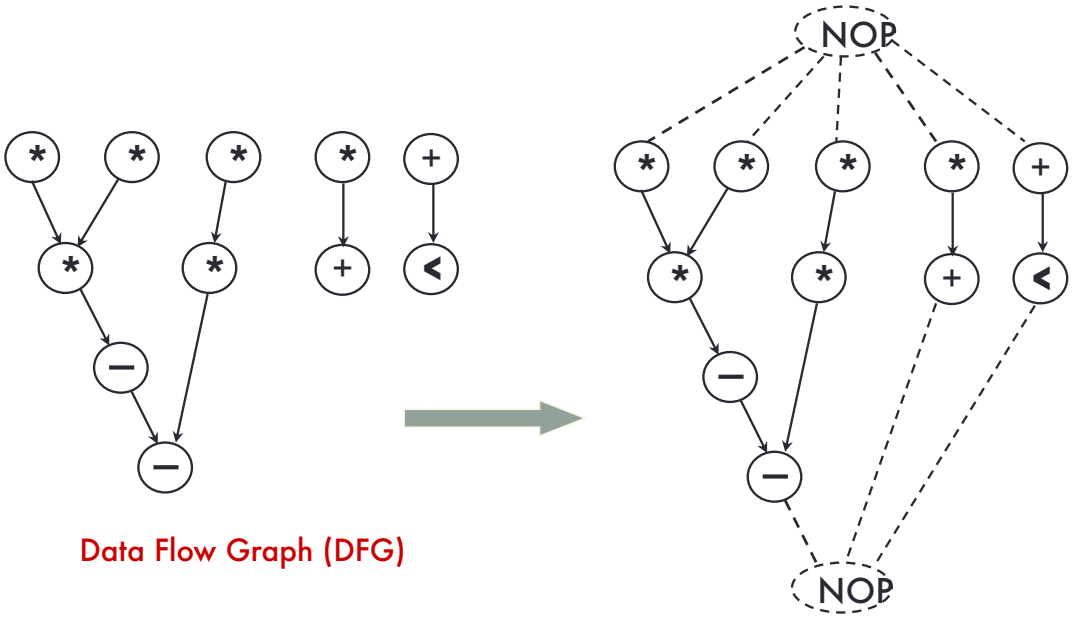
LAU Computer Science

Sequencing Graph



Sequencing Graph

- Add *source* and *sink* nodes (NOP) to the DFG



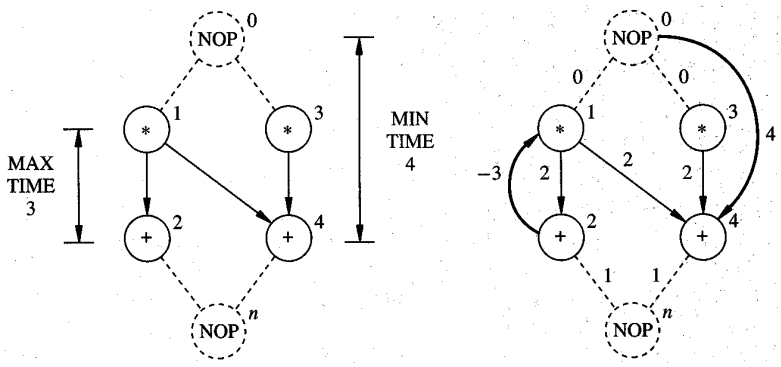
Data Flow Graph (DFG)

Sequencing Graph

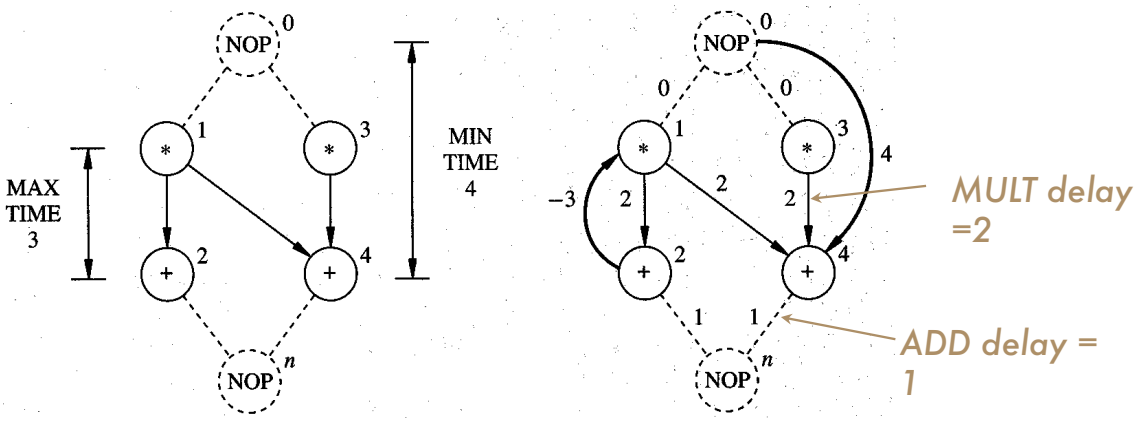
Timing Constraints

- Time measured in *cycles* or *control steps*
- Imposing relative timing constraints between operators *i* and *j*
  - max & min timing constraints

A **minimum** timing constraint $l_{ij} \geq 0$ requires: $t_j \geq t_i + l_{ij}$.
A **maximum** timing constraint $u_{ij} \geq 0$ requires: $t_j \leq t_i + u_{ij}$.

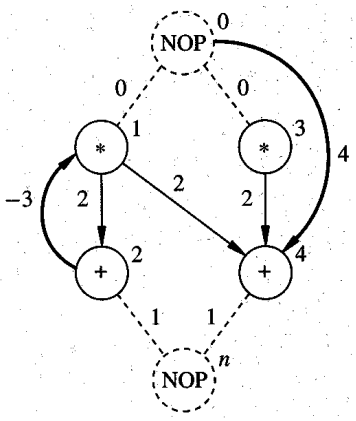


Constraint Graph $G_c(V,E)$



Existence of Schedule under Timing Constraints

- Upper bound (max timing constraint) is a problem
- Examine each *max* timing constraint (i, j):
  - Longest weighted path between nodes i and j must be $\leq$ max timing constraint u_{ij}.
  - Any cycle in G_c including edge (i, j) must be negative or zero
- Necessary and sufficient condition:
  - The constraint graph G_c must not have positive cycles



- Example:
  - Assume delays: ADD=1, MULT=2
  - Path {1 → 2} has weight $2 \leq u_{12}=3$, that is cycle {1,2,1} has weight = -1, OK
  - No positive cycles in the graph, so it has a consistent schedule

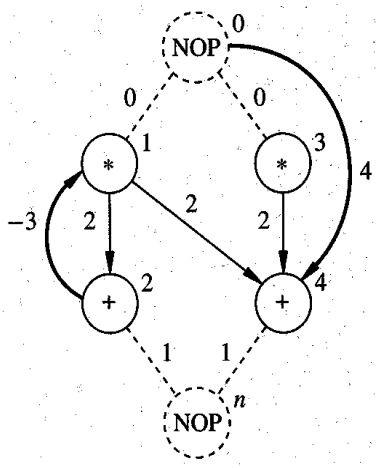
LAU Computer Science

Existence of schedule under timing constraints

- Example: satisfying assignment
  - Assume delays: ADD=1, MULT=2
  - Feasible assignment:

Vertex Start time

- v_0 → step 1
- v_1 → step 1
- v_2 → step 3
- v_3 → step 1
- v_4 → step 5
- v_n → step 6



LAU Computer Science

Conclusion 1: Scheduling

- NP-complete Problem
- Optimal solutions for special cases and ILP
- Heuristics - iterative Improvements
- Heuristics – constructive
- Various versions of the problem
  - Unconstrained, minimum latency
  - Resource-constrained, minimum latency
  - Timing-constrained, minimum latency
  - Latency-constrained, minimum resource
- If all resources are identical, problem is reduced to multiprocessor scheduling (Hu's algorithm)
  - Minimum latency multiprocessor problem is intractable

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 — minimize resources
- Assumptions
  - All delays bounded and known

Minimum Latency Resource Constrained

- Given a set of operations *V* with integer delays *D*, a partial order on the operations *E*, and upper bounds a_k, $\{k = 1, 2, \ldots, n_{resources}\}$
- Find an integer labeling of the operations
  - $\Phi: V \rightarrow Z^+$
  - Such that
    - $t_i = \Phi(v_i)$
    - $t_i \geq t_j + d_j \; (v_i, v_j) \in E$
  - $|\{v_i | T(v_i) = k \text{ and } t_i \leq l < t_i + d_i\}| \leq a_k \; \forall k$ and $\forall$ steps l
    - t_n is minimum

Scheduling Under Resource Constraints

- Problem is NP Hard
  - Also called *intractable*
- Algorithms
  - Exact
    - Integer Linear Program
    - Hu
  - Approximate
    - List Scheduling
    - Force-Directed Scheduling

Scheduling – a Combinatorial Optimization Problem

- NP-complete Problem
- Optimal solutions for special cases and for ILP
  - Integer linear program (ILP)
  - Branch and bound
- Heuristics
  - iterative Improvements, constructive
- Various versions of the problem
  - Minimum latency, unconstrained (ASAP)
  - Latency-constrained scheduling (ALAP)
  - Minimum latency under resource constraints (ML-RC)
  - Minimum resource schedule under latency constraint (MR-LC)
- If all resources are identical, problem is reduced to multiprocessor scheduling (Hu's algorithm)
  - In general, minimum latency multiprocessor problem is intractable under resource constraint
  - Under certain constraints (G(VE) is a tree), greedy algorithm gives optimum solution

ILP Model of Scheduling

- Binary decision variables x_{il}

$x_{il} = 1$ if operation v_i starts in step l,

otherwise $x_{il} = 0$

$i = 0, 1, \ldots, n$ (operations)

$l = 1, 2, \ldots \lambda+1$ (steps, with limit λ)

- Start time o

$$\sum_{l} x_{il} = 1, \quad i = 0, 1, \ldots, n$$

Note: $\sum_{l} x_{il} = \sum_{l=t_i^S}^{l=t_i^L}$

where: x_{il}

t_i^S = time of operation I computed with *ASAP*

t_i^L = time of operation I computed with *ALAP*

ILP Model of Scheduling - constraints

- Start time for v_i:

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

- Precedence relationships must be satisfied

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

- Resource constraints must be met
  - let upper bound on number of resources of type k be a_k

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

Latency Minimization - Objective Function

- Function to be minimized: $F = c^T t$, where $t_i = \sum_l l \cdot x_{il}$
- Minimum latency schedule: $c = [0, 0, \ldots, 1]^T$
  - $F = t_n = \sum_l l\, x_{nl}$
  - if sink has no mobility ($x_{n,s} = 1$), any feasible schedule is optimum
- ASAP: $c = [1, 1, \ldots, 1]^T$
  - finds earliest start times for all operations $\sum_i \sum_l x_{il}$
  - or equivalently:

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

Minimum-Latency Scheduling under Resource Constraints (ML-RC)

- Let ***t*** be the vector whose entries are *start times*

$$t = [t_0, t_1, \ldots, t_n]$$

- Formal ILP model

$$\text{minimize } \mathbf{c}^T\mathbf{t} \text{ such that}$$

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

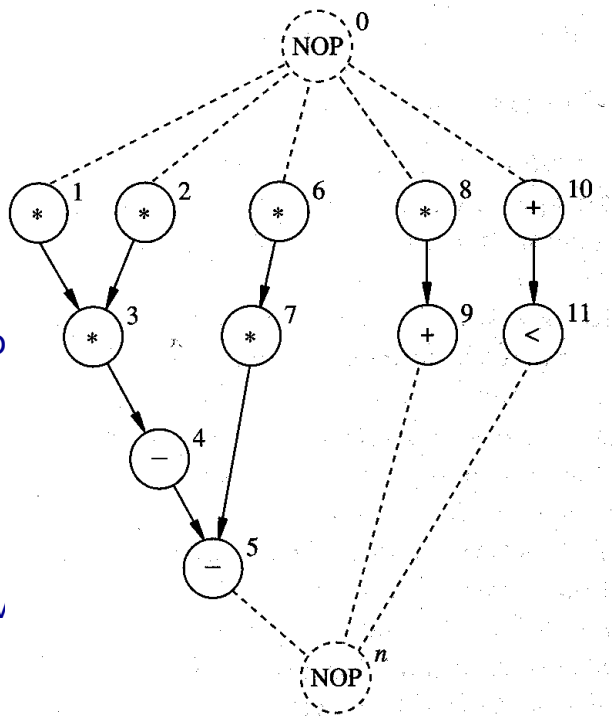
$$\sum_l l \cdot x_{il} - \sum_l l \cdot x_{jl} - d_j \geq 0, \quad i, j = 0, 1, \ldots, n \ : (v_j, v_i) \in E$$

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

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

Example 1 – multiple resources

- Two types of resources
  - MULT
  - ALU
    - Adder, Subtractor
    - Comparator
- Each take 1 cycle of executio time
- Assume upper bound on latency, L = 4
- Use ALAP and ASAP to deriv bounds on start times for each operator

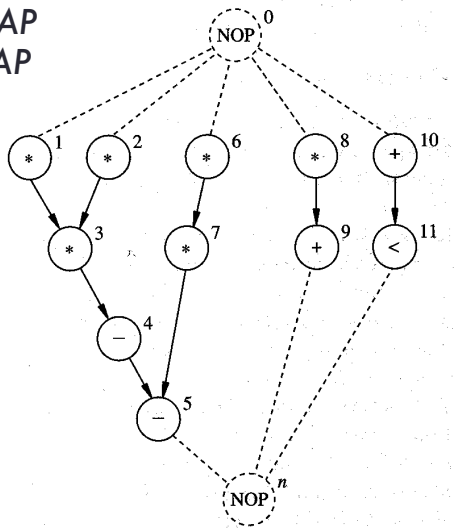


Example 1 (cont'd.)

- Start time must be unique

Recall: $\sum_{l} x_{il} = \sum_{l=t_i^S}^{l=t_i^L} x_{il}$

where:
t_i^S = t_i computed with ASAP
t_i^L = t_i computed with ALAP

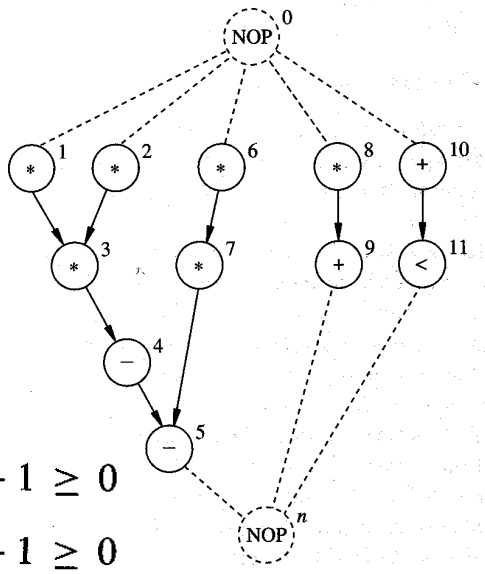


$$x_{0,1} = 1$$
$$x_{1,1} = 1$$
$$x_{2,1} = 1$$
$$x_{3,2} = 1$$
$$x_{4,3} = 1$$
$$x_{5,4} = 1$$
$$x_{6,1} + x_{6,2} = 1$$
$$x_{7,2} + x_{7,3} = 1$$
$$x_{8,1} + x_{8,2} + x_{8,3} = 1$$
$$x_{9,2} + x_{9,3} + x_{9,4} = 1$$
$$x_{10,1} + x_{10,2} + x_{10,3} = 1$$
$$x_{11,2} + x_{11,3} + x_{11,4} = 1$$
$$x_{n,5} = 1$$

LAU Computer Science

Example 1 (cont'd.)

- Precedence constraints
  - Note: only non-trivial ones listed

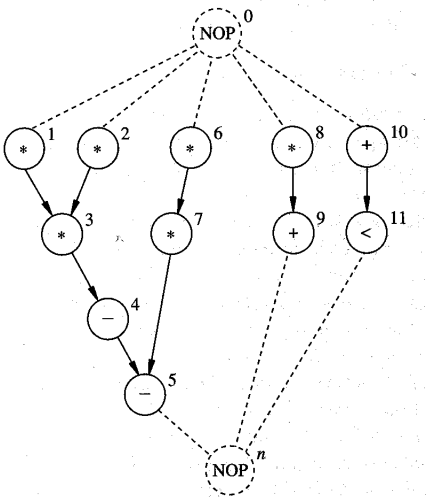


$$2x_{7,2} + 3x_{7,3} - x_{6,1} - 2x_{6,2} - 1 \geq 0$$
$$2x_{9,2} + 3x_{9,3} + 4x_{9,4} - x_{8,1} - 2x_{8,2} - 3x_{8,3} - 1 \geq 0$$
$$2x_{11,2} + 3x_{11,3} + 4x_{11,4} - x_{10,1} - 2x_{10,2} - 3x_{10,3} - 1 \geq 0$$
$$4x_{5,4} - 2x_{7,2} - 3x_{7,3} - 1 \geq 0$$
$$5x_{n,5} - 2x_{9,2} - 3x_{9,3} - 4x_{9,4} - 1 \geq 0$$
$$5x_{n,5} - 2x_{11,2} - 3x_{11,3} - 4x_{11,4} - 1 \geq 0$$

LAU Computer Science

Example 1 (cont'd.)

- Resource constraints



MULT
a1=2

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

ALU
a2=2

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

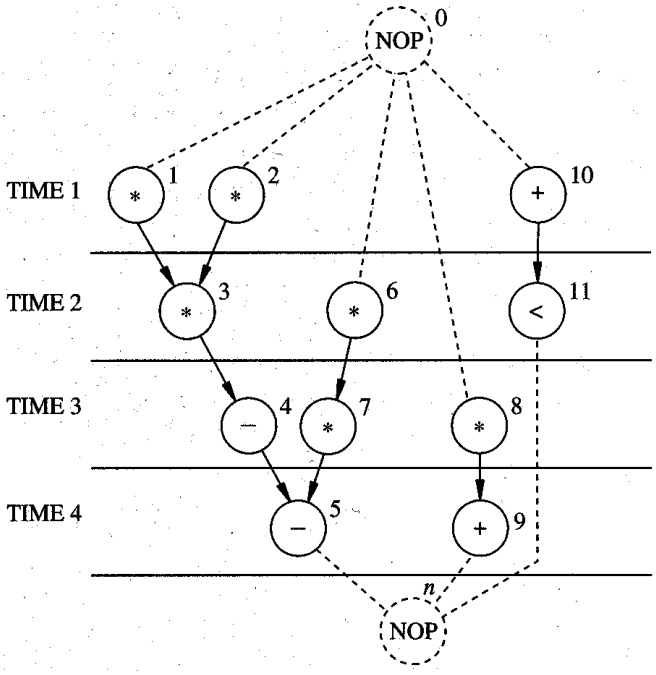
Example 1 (cont'd.)

- Objective function (some possibilities): $F = c^T t$
- *F1*: $\mathbf{c} = [0, 0, \ldots, 1]^T$
  - Minimum latency schedule
  - since sink has no mobility ($x_{n,5} = 1$), any feasible schedule is optimum
- *F2*: $\mathbf{c} = [1, 1, \ldots, 1]^T$
  - finds earliest start times for all operations $\sum_i \sum_l x_{il}$
  - or equivalently:

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

Example Solution 1:

Min. Latency Schedule Under Resource Constraint



Minimum Resource Scheduling under Latency Constraint

- Special case
  - Identical operations, each executing in one cycle time
- Given a set of operations $\{v_1, v_2, \ldots, v_n\}$,
  - find the *minimum number* of operation units needed to complete the execution in k control steps (*MR-LC problem*)
- Integer Linear Programming (ILP):
  - Let y_o be an integer variable (# units to be minimized)
  - for each control step $l = 1, \ldots, k$, define variable x_{il} as

$x_{il} = 1$, if computation v_i is executed in the l-*th* control step

0, otherwise

  - define variable y_l (number of units in control step l)

$$y_l = x_{1l} + x_{2l} + \ldots + x_{nl} = \Sigma_i x_{il}$$

ILP Scheduling – simple MR-LC

- Minimize: y_o

Subject to:

- Each computation v_i can start only once: $\sum_l x_{il} = 1, \quad i = 0, 1, \ldots, n$

  x_{il}= 1 *for only one value of* l *(control step)*
- For each precedence relation:
  - *If* v_j has to be executed *after* v_i

    $x_{j1} + 2\,x_{j2} + \ldots + k\,x_{jk} \geq x_{i1} + 2\,x_{i2} + \ldots + k\,x_{ik} + d(i)$
- $y_l \leq y_o$ for all $l = 1, \ldots, k$ (steps)

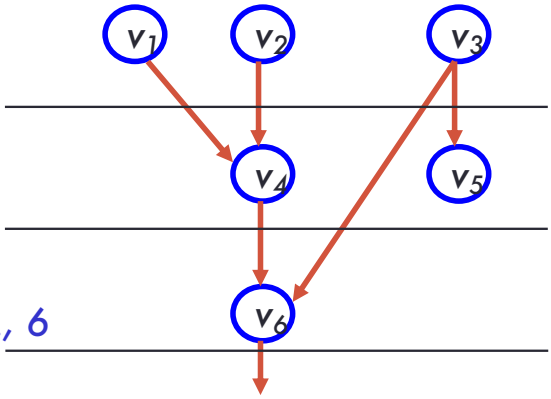
- Meaning of *yo:*

  upper bound on the number of units, to be minimized

LAU Computer Science

Example 2 - Formulation

n = 6 computations
k = 3 control steps
d(i) = 1



- Execution constraints:

$x_{i1} + x_{i2} + x_{i3} = 1$ for $i = 1, \ldots, 6$

- Resource constraints:

$y_l = x_{1l} + x_{2l} + x_{3l} + x_{4l} + x_{5l} + x_{6l}$ for $l = 1, \ldots, 3$ (steps)

- Dependency constraints: e.g. v_4 executes after v_1

$x_{41} + 2x_{42} + 3x_{43} \geq x_{11} + 2x_{12} + 3x_{13} + 1$

. . . . . etc.

LAU Computer Science

Example 2 - Solution

- Minimize: y_0
- Subject to:

  $y_l \leq y_0$ for $l = 1, \ldots, 3$

  - *Starting time constraints …*
  - *Precedence constraints …*
- One possible solution:

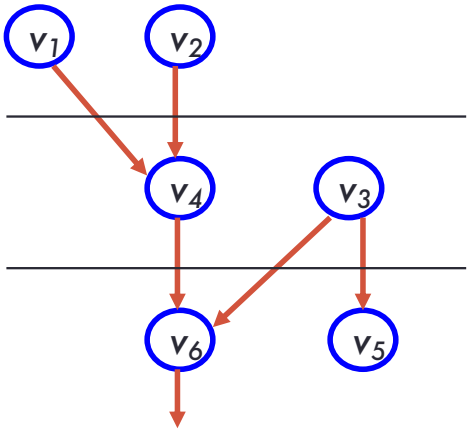
  $y_0 = 2$

  $x_{11} = 1, x_{21} = 1,$

  $x_{32} = 1, x_{42} = 1,$

  $x_{53} = 1, x_{63} = 1.$

  all other $x_{il} = 0$



Minimum Resource Scheduling under Latency Constraint – general case

- General case: several operation units (resources)
- Given
  - vector $c = [c_1, \ldots, c_r]$ of resource costs (areas)
  - vector $a = [a_1, \ldots, a_r]$ of number of resources (unknown)
- Minimize total cost of resources

  $\min \boldsymbol{c^T a}$
- Resource constraints are expressed in terms of variables

  a_k = number of operators of type k

Example 3 – Min. Resources under Latency Constraint

- Let $c = [5, 1]$
  - MULT costs = 5 units of area, $c_1 = 5$
  - ALU costs = 1 unit of area, $c_2 = 1$
- Starting time constraint – as before
- Sequencing constraints - as before

- Resource constraints – similar to previous ones, but expressed in terms of unknown variables a_1 and a_2

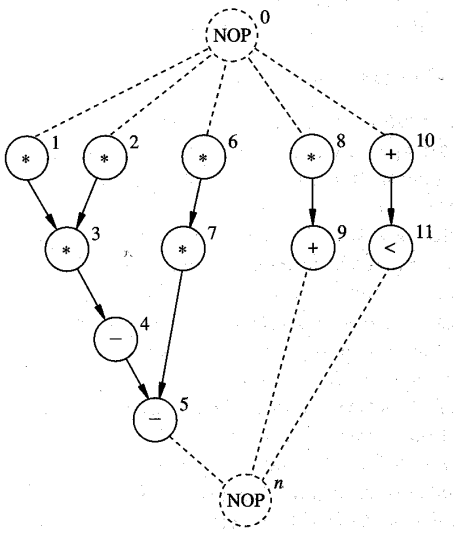
  a_1 = number of multipliers

  a_2 = number of ALUs (add/sub)
- Objective function:

  $\boldsymbol{c^T a} = 5 \cdot a_1 + 1 \cdot a_2$

Example 3 (contd.)

- Resource constraints

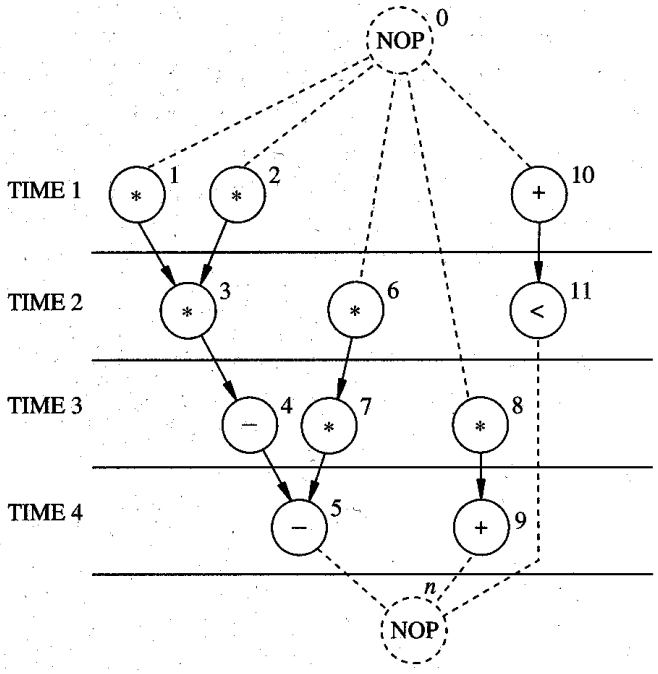


$$\textit{MULT} \begin{cases} x_{1,1} + x_{2,1} + x_{6,1} + x_{8,1} - a_1 \leq 0 \\ x_{3,2} + x_{6,2} + x_{7,2} + x_{8,2} - a_1 \leq 0 \\ x_{7,3} + x_{8,3} - a_1 \leq 0 \end{cases}$$

$$\textit{ALU} \begin{cases} x_{10,1} - a_2 \leq 0 \\ x_{9,2} + x_{10,2} + x_{11,2} - a_2 \leq 0 \\ x_{4,3} + x_{9,3} + x_{10,3} + x_{11,3} - a_2 \leq 0 \\ x_{5,4} + x_{9,4} + x_{11,4} - a_2 \leq 0 \end{cases}$$

Example 3 - Solution

- Minimize
  $\mathbf{c}^T\mathbf{a} = 5 \cdot a_1 + 1 \cdot a_2$
- Solution with *cost = 12*
  - *a1 = 2*
  - *a2 = 2*



LAU Computer Science

Precedence-constrained Multiprocessor Scheduling

- All operations performed by the same type of resource
  - intractable problem; even if operations have unit delay
  - except when the G_c is a tree (then it is optimal and $O(n)$)

$$\text{minimize } \mathbf{c}^T\mathbf{t} \text{ such that}$$

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

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

$$\sum_i x_{il} \leq a, \quad l = 1, 2, \ldots, \bar{\lambda} + 1$$

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

LAU Computer Science

Scheduling Under Resource Constraints

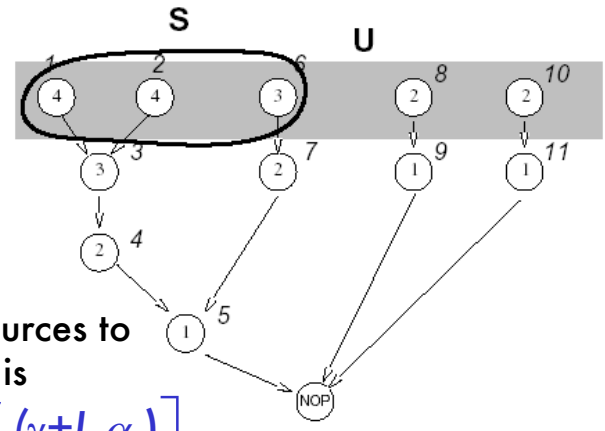
- Hu's Algorithm
  - Label vertices with distance from sink
  - Greedy strategy
  - Exact solution
- Assumptions
  - Graph is a forest
  - All operations have unit delay
  - All operations have the same type

Hu's Algorithm

- Simple case of the scheduling problem
  - Operations of unit delay
  - Operations (and resources) of the same type ("multiprocessor" operation)
- Hu's algorithm
  - Greedy
  - Polynomial time and optimal for trees
  - Computes lower bound on number of resources for a given latency (MR-LCS), or
  - computes lower bound on latency subject to resource constraints (ML-RCS)
- 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

- Labeling of nodes
  - Label operations based on their distances from the sink
- Notation
  - α_i = label of node i
  - $\alpha = max_i\ \alpha_i$
  - $p(j)$ = # vertices with label j



- Theorem (Hu)

  Lower bound on the number of resources to complete schedule with latency L is

  $a_{min} = max_\gamma \lceil \sum_{j=1}^{\gamma} p(\alpha+1-j) / (\gamma+L-\alpha) \rceil$

  where γ is a positive integer ($1 \le \gamma \le \alpha+1$)
- In this case: $a_{min} = 3$

LAU Computer Science

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 $\subseteq$ U such that |S| $\le$ a and labels in S are maximal

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

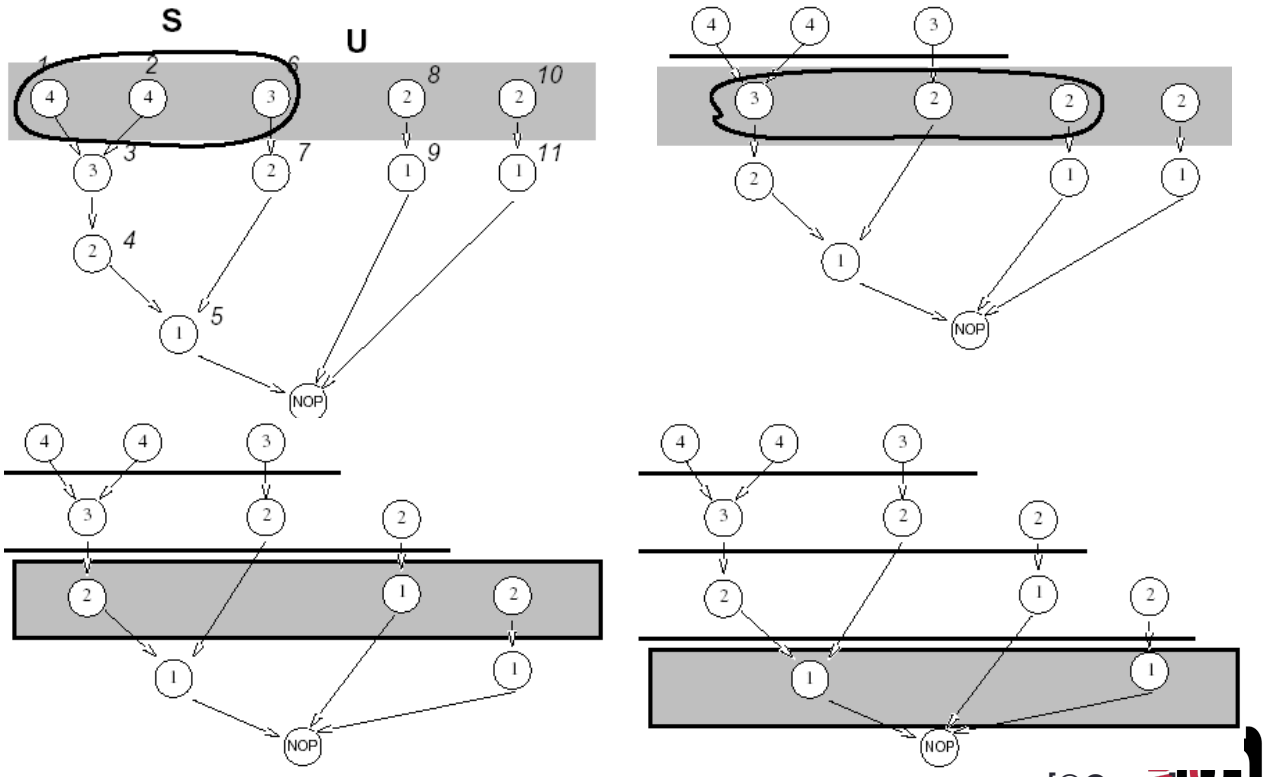
$l = l + 1$;

} until v_n is scheduled.

}

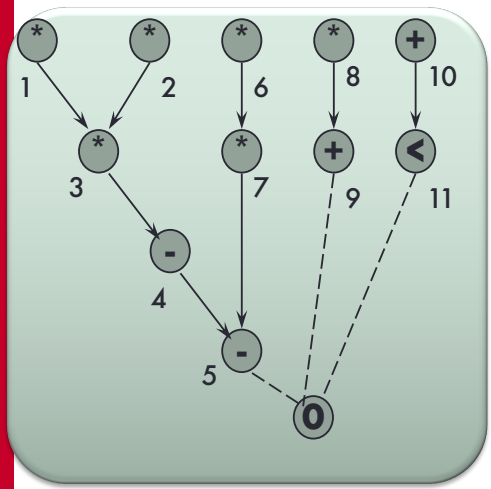
LAU Computer Science

Hu's Algorithm: Example (a=3)



[©Gupta]

Example



Set step *l* = 1;
do {
Select s ≤ *a* resources with:
- All predecessors scheduled
- Maximal labels
Schedule the s operations at step *l*
l = *l* + 1;
} **while** all operations are not scheduled;

Minimum latency with a = 3 resources

Step 1: Select $\{v_1, v_2, v_6\}$

Step 2: Select $\{v_3, v_7, v_8\}$

Step 3: Select $\{v_4, v_9, v_{10}\}$

Step 4: Select $\{v_5, v_{11}\}$

List Scheduling Algorithms

- Heuristic Method for
  - Minimum latency subject to resource bound
  - Minimum resource subject to latency bound
- Greedy strategy
- General Graphs
- Use priority list heuristics
  - Longest path to sink
  - Longest path to timing constraint
- Similar to Hu's algorithm
  - Operation selection decided by criticality
  - *O(n)* time complexity

LAU Computer Science

List Scheduling Algorithms

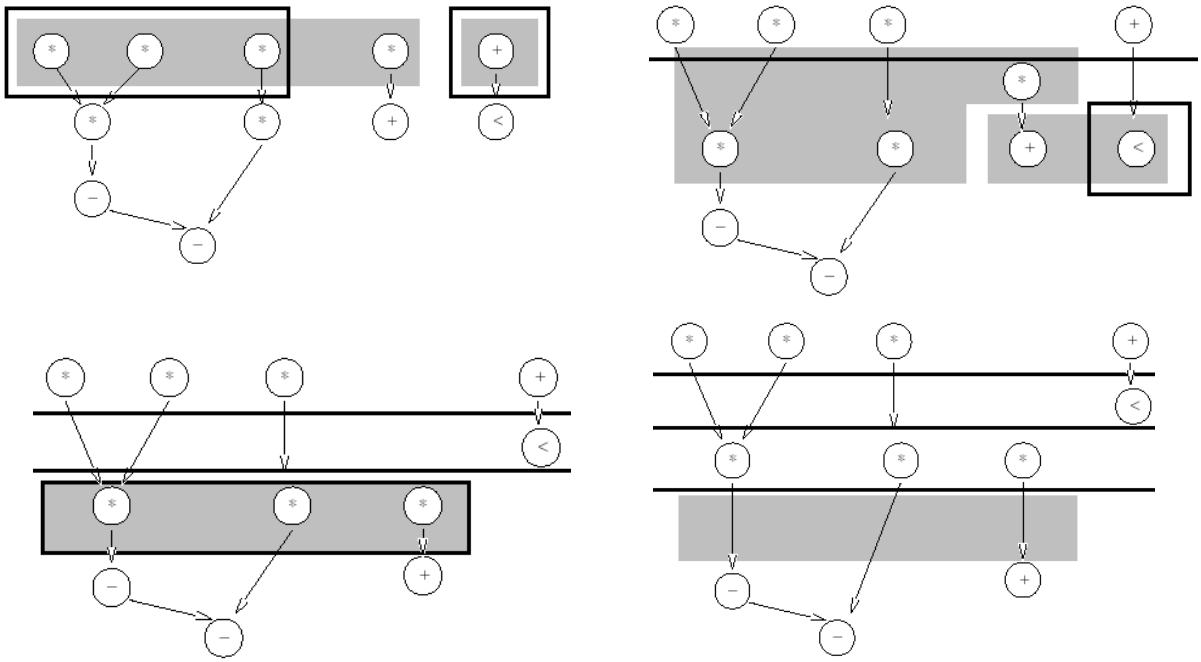
- Algorithm 1: Minimize latency under resource constraint (ML-RC)
  - Resource constraint represented by vector $\boldsymbol{a}$ (indexed by resource type)
    - Example: two resources, MULT, ADD; $a_1=1$, $a_2=2$
- The candidate operations $U_{l,k}$
  - those operations of type k whose predecessors have already been scheduled early enough so that they are completed at step l:

    $U_{l,k} = \{ v_i \subseteq V: type(v_i) = k$ and $t_j + d_j \leq l$, for all j: $(v_i, v_j) \subseteq E$
- The unfinished operations $T_{l,k}$
  - those operations of type k that started at earlier cycles but whose execution is not finished at step l:

    $T_{l,k} = \{ v_i \subseteq V: type(v_i) = k$ and $t_i + d_i > l$
- Priority list
  - List operators according to some heuristic urgency measure
  - Common priority list: labeled by position on the longest path in decreasing order
- Algorithm 2: Minimize resources under latency constraint (MR-LC)

LAU Computer Science

ML-RC Scheduling: Example (a=[3,1])

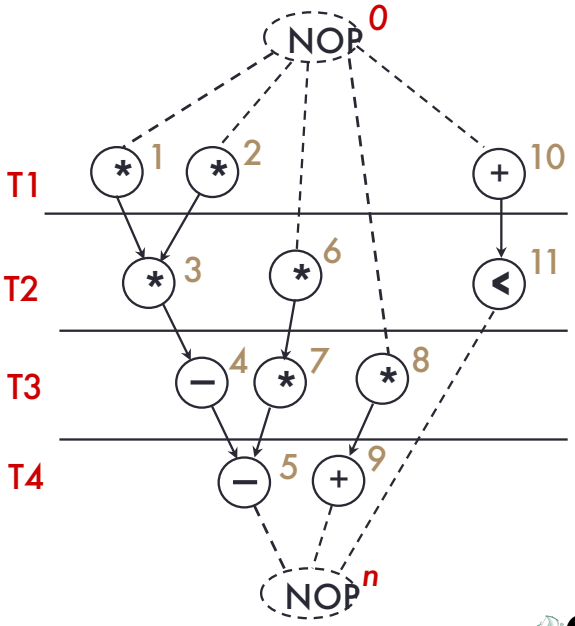


List Scheduling – Example 1a (a=[2,2])

Minimize latency under resource constraint (with d = 1)

- Assumptions
  - All operations have unit delay (d_i=1)
  - Resource constraints:

    MULT: a_1 = 2, ALU: a_2 = 2
- Step 1:
  - $U_{1,1} = \{v_1, v_2, v_6, v_8\}$, select $\{v_1, v_2\}$
  - $U_{1,2} = \{v_{10}\}$, select + schedule
- Step 2:
  - $U_{2,1} = \{v_3, v_6, v_8\}$, select $\{v_3, v_6\}$
  - $U_{2,2} = \{v_{11}\}$, select + schedule
- Step 3:
  - $U_{3,1} = \{v_7, v_8\}$, select + schedule
  - $U_{3,2} = \{v_4\}$, select + schedule
- Step 4:
  - $U_{4,2} = \{v_5, v_9\}$, select + schedule



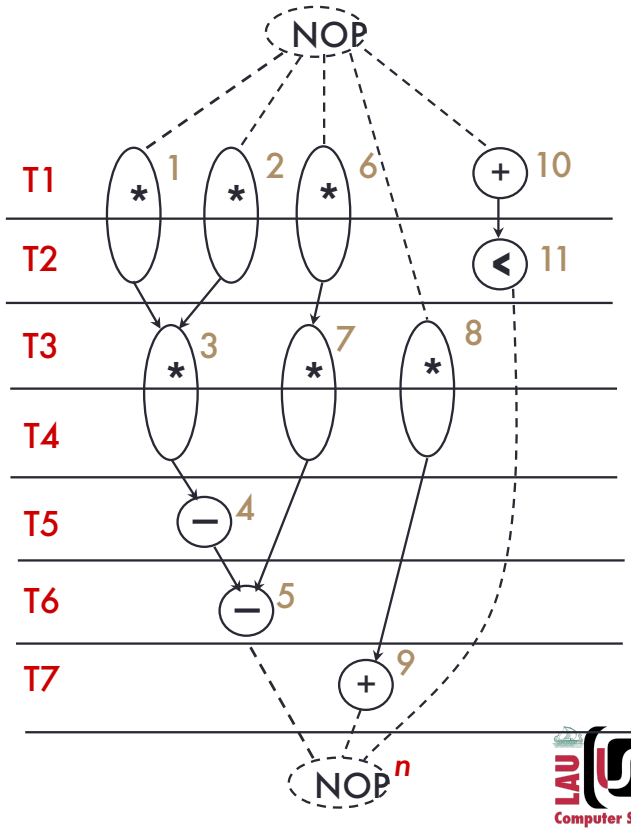
List Scheduling – Example 1b (a = [3,1])

Minimize latency under resource constraint (with d_1=2, d_2=1)

- Assumptions
  - Operations have different delay:
    $del_{MULT} = 2$, $del_{ALU} = 1$
  - Resource constraints:
    MULT: $a_1 = 3$, ALU: $a_2 = 1$

- MUTL 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

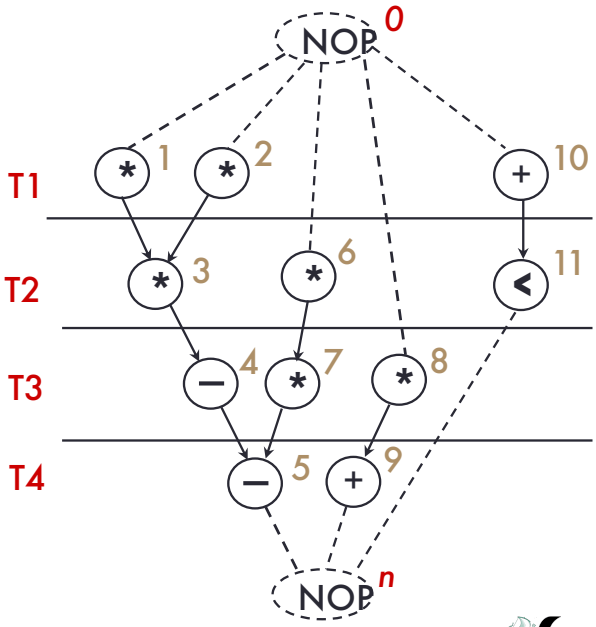


List Scheduling – Example 2

Minimize resources under latency constraint

- Assumptions
  - All operations have unit delay (d_i=1)
  - Latency constraint: $L = 4$

- Use slack information to guide the scheduling
  - Schedule operations with slack = 0 first

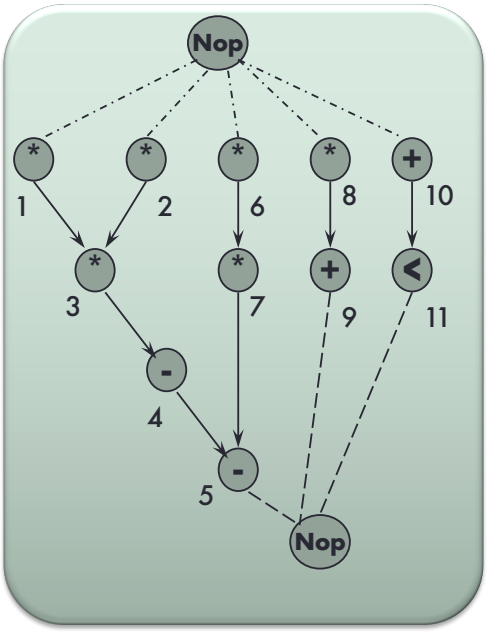


List Scheduling - Minimum Latency

```
List_Sch(G(V, E), a) {
l = 1;
do {
    for each resource type k = 1, 2, ... n_resources {
        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;
    } while v_n is not scheduled;
    return(t);
}
```


LAU Computer Science

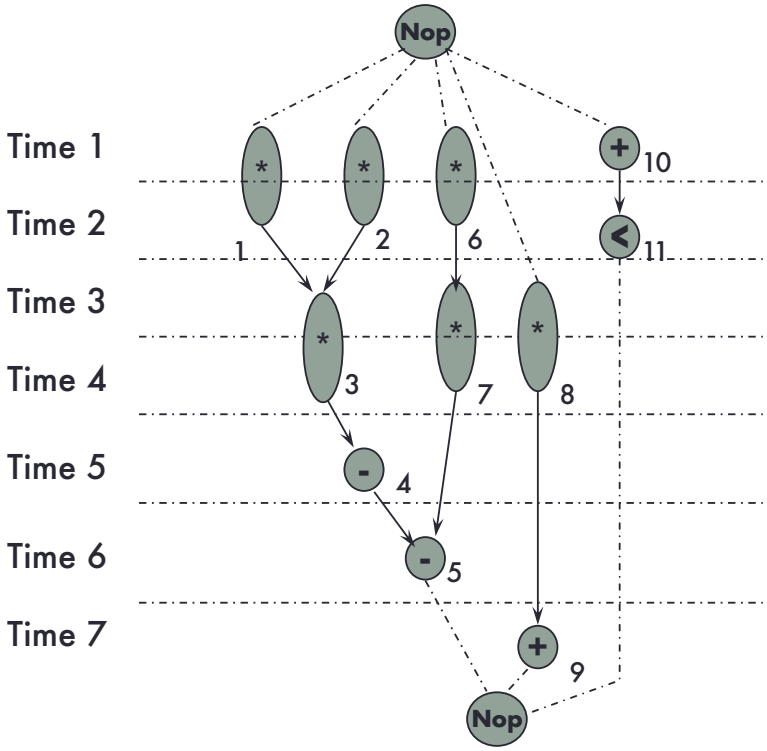
Example



- Assumptions
  - a_1 = 3 multipliers with delay 2
  - a_2 = 1 ALUs with delay 1

LAU Computer Science

Example



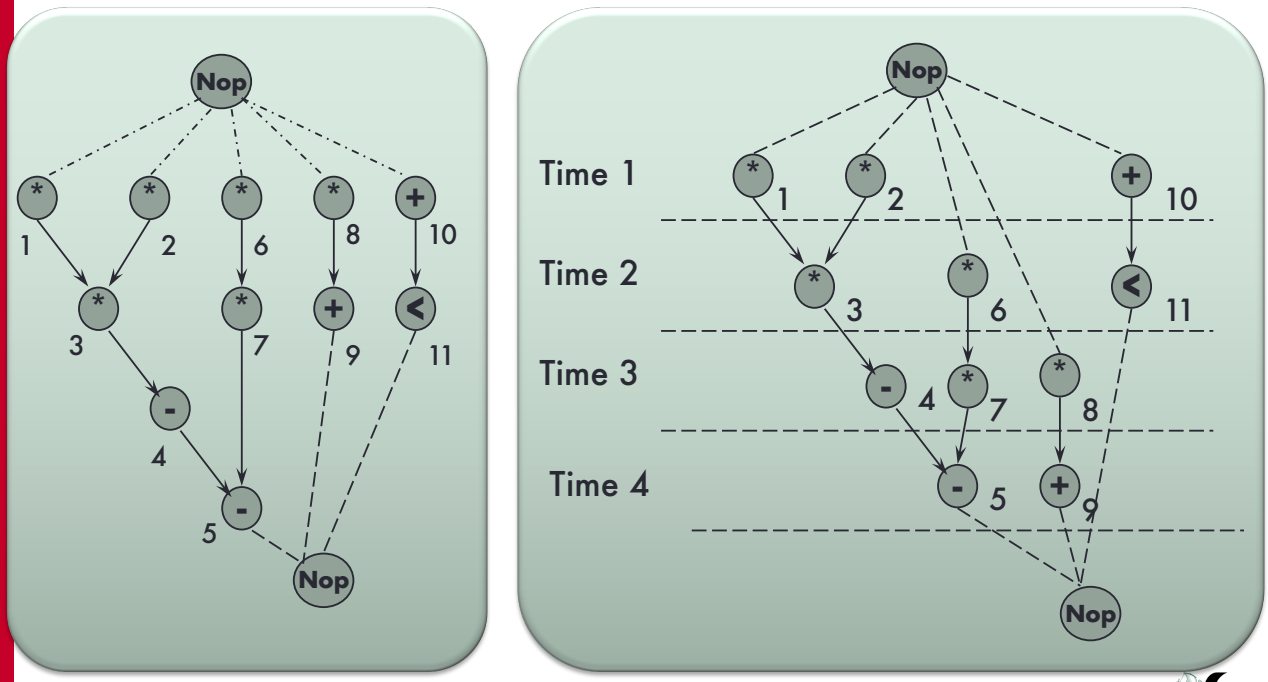
LAU Computer Science

List Scheduling - Minimum Resource

```
List_Sch_Resources(G(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;
do {
    foreach resource type k = 1, 2, ..., n_res {
        Determine candidate operations U_lk;
        Compute the slacks {s_i = t^L_i - l  ∀ v_i ∈ U_lk
        Schedule candidate operations with zero slack and update a
        Schedule candidate operations that do not require additional resources;
        }
    l = l + 1;
} while (v_n is not scheduled);
return(t, a);
}
```


LAU Computer Science

Example



Scheduling – a Combinatorial Optimization Problem

- NP-complete Problem
- Optimal solutions for special cases (trees) and ILP
- Heuristics
  - iterative Improvements
  - constructive
- Various versions of the problem
  - Minimum latency, unconstrained (ASAP)
  - Latency-constrained scheduling (ALAP)
  - Minimum latency under resource constraints (ML-RC)
  - Minimum resource schedule under latency constraint (MR-LC)
- If all resources are identical, problem is reduced to multiprocessor scheduling (Hu's algorithm)
  - Minimum latency multiprocessor problem is intractable for general graphs
  - For trees greedy algorithm gives optimum solution

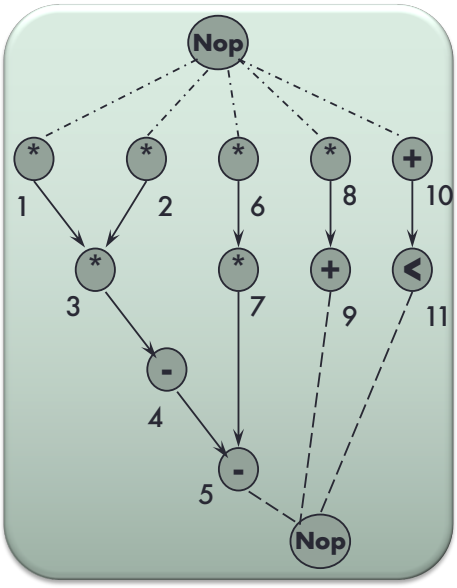
Force Directed Scheduling

- Heuristics scheduling methods
  - Minimum latency subject to resource constraints
    - A variation of list scheduling where the force is used as a priority function
  - Minimum resource subject to latency constraint
    - Schedule one operation at a time
- Goal
  - Achieve uniform distribution of operations across schedule steps

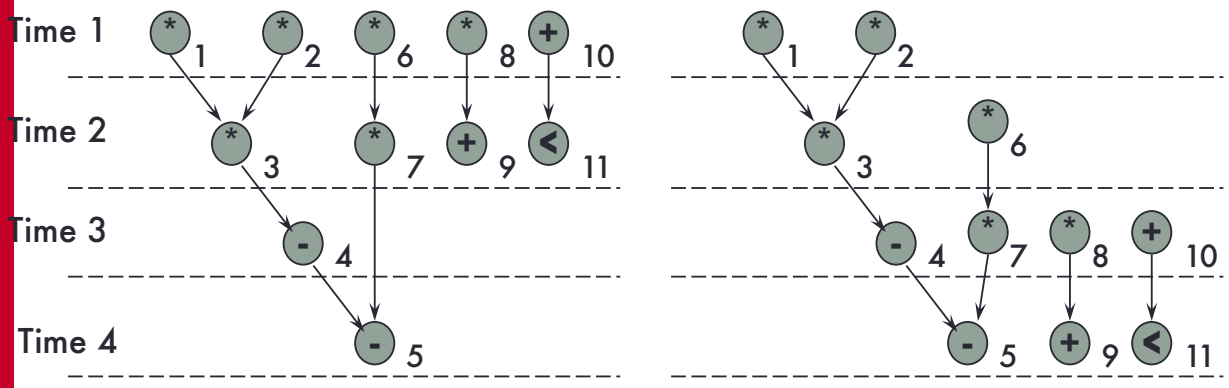
Force-Directed Scheduling

- Definitions
  - Operations interval, computed using ASAP and ALAP
    - Mobility plus one
  - Operation Probability $p_i(l)$
    - Probability of an operation to execute in a given step
    - 1/(mobility+1) inside interval; 0 elsewhere
  - Operation type distribution, $q_k(l)$
  - Sum of the operations probability for each type

Example



FDS: Example



mobility of operations v_i; i = 1, 2, 3, 4, 5, is zero
mobility of operations $\{v_6, v_7\}$ is one
mobility of remaining nodes is two

Example

v_1 has zero mobility $\Rightarrow p_1(1) = 1; p_1(2) = p_1(3) = p_1(4) = 0$

v_2 has zero mobility $\Rightarrow p_2(1) = 1; p_2(2) = p_2(3) = p_2(4) = 0$

v_6 has mobility one. Time frame is [1, 2]
$\Rightarrow$
$p_6(1) = p_6(2) = 0.5$
$p_6(3) = p_6(4) = 0$

v_8 has mobility two. Time frame is [1, 3]
$\Rightarrow$
$p_8(1) = p_8(2) = p_6(3) = 0.3$
$p_8(4) = 0$

Type distribution for the multiplier (k = 1) at step 1 is $q_1(1) = 1 + 1 + 0.5 + 0.3 = 2$

Example: Distribution Graph

Force

- Used as a *priority* function and related to concurrency
  - Sort operations for least force
- Mechanical analogy
  - Force = constant x displacement
    - Constant = operation-type distribution
    - displacement = change in probability
- Self-Force
  - Sum of forces to other steps
  - Self force operation v_i in step l:

$$\sum_{m=t_i^S}^{t_i^L} q_k(m)(\delta_{lm} - p_i(m))$$

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

LAU Computer Science

Example: operation v_6

- It can be scehduled 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
- Assign v_6 to step 1
  - Variation in probability 1 - 0.5 = 0.5 for step 1
  - Variation in probability 0 - 0.5 = -0.5 for step 2
- Self-Force
  - 2.8 * 0.5 -2.3*0.5 = +0.25

LAU Computer Science

Example: operation v_6

- Assign v_6 to step 2
  - Variation in probability 0 - 0.5 for step 1
  - Variation in probability 1 - 0.5 = 0.5 for step 2
- Self-Force
  - -2.8*0.5+ 2.3*0.5= -0.25
- Successor force
  - Operation v_7 assigned to step 3
    - 2.3 * (0 - 0.5) + 0.8 * (1 - 0.5) = - 0.75
- Total Force is -1
- Conclusion
  - Least force is for step 2
  - Assigning v_6 to step 2 reduces concurrency

FDS Algorithm

```
FDS( G(V, E), λ) {
        do {
            Compute 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 the time-frame:
        } while (not all operations are scheduled);
return (t);
}
```