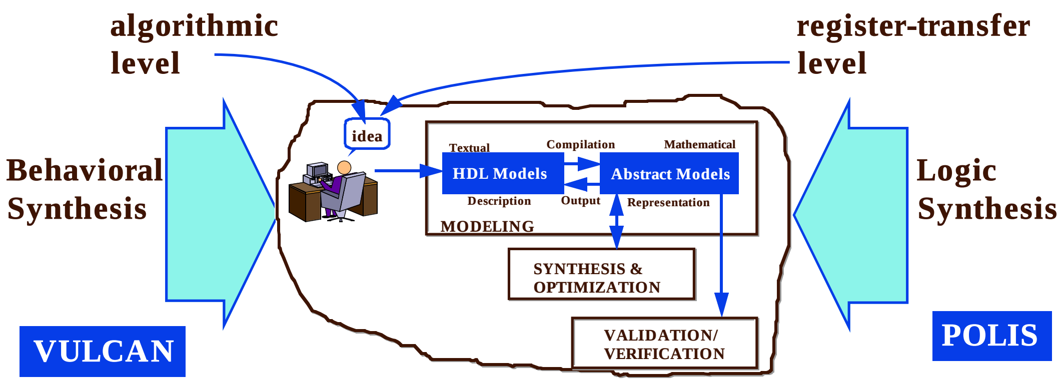
Constraint Analysis in Vulcan

by
Margarida F. Jacome

The University of Texas at Austin

The Co-Synthesis Approach



The hardware software co-design problem is posed as an evolution of existing synthesis Methods

Margarida Jacome - UT Austin - Spring 98

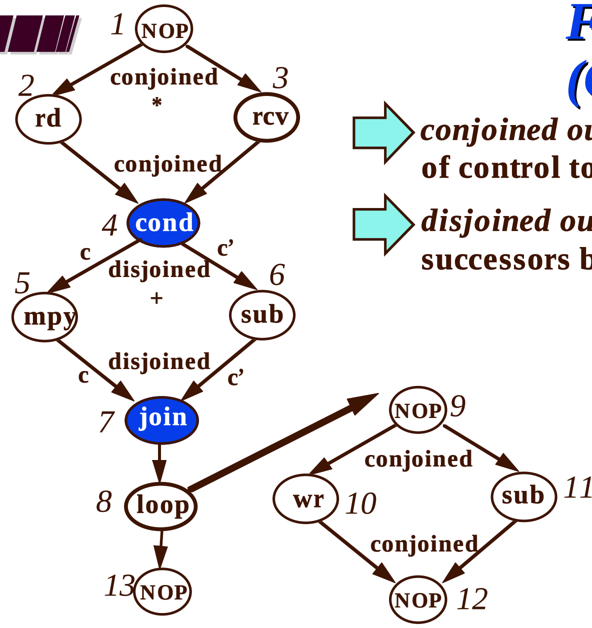
Vulcan

- Specification
- Modeling
- Constraint Analysis
- Software and Runtime Environment
- Target Architecture - H/S Interface
- Partitioning
- Co-simulation

Model -- Flow Graph

- Hierarchical control/data-flow graph
  - control flow primitives (iteration and model call modeled though hierarchy
- Acyclic
  - models partial order of tasks/operations
  - iteration is modeled outside the graph
- Polar
  - source and sink vertices model No-Operatios

Flow Graph (CDFG)



- ***conjoined output***: directs the flow of control to all its branches
- ***disjoined output***: selects one of the successors based on a condition index

```
process example (a, b, c)
  in port a[8],
  in channel b[8];
  out port c[8];
{
   boolean x[8], y[8], z[8];
   x = read(a);
   y = receive(b);
   if (x > y)
       z = x - y;
  else
       z = x * y;
  while (z >= 0)
      write c = y;
      z = z - 1; }
```

Operation vertices in a Flow Graph

- ***no-op***: no operation
- ***cond***: conditional fork
- ***join***: conditional join
- ***op-logic***: logical operations
- ***op-arithmetic***: arithmetic operations
- ***op-relational***: relational operations
- ***op-io:*** I/O operations
- ***wait***: wait on a signal variable
- ***link***: hierarchical operations
  - ***call***: procedure call (invocation times = 1)
  - ***loop***: iteration (invocation times $\geq$ 1)

System Model

- Consists of one or more flow graphs that may be hierarchically linked to other flow graphs

System Model: $\Phi = \{G_1^*, G_2^*, \ldots, G_n^*\}$
where

G_i^* **represents the process graph model** G_i **and all the flow graphs that are hierarchically linked to** G_i**.**

☆ **A flow graph model that is common to two hierarchies of a system model is called a *shared model***

Flow Graphs: Execution Semantics

- At any time, an operation may be
  - waiting for execution
  - presently executing
  - having completed its execution

The state of a vertex is defined as being one of $\{s_r, s_e, s_d\}$

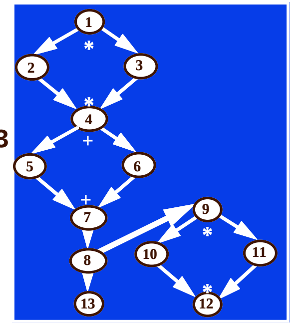
s_r: **reset state ==> waiting for execution**

s_e: **enable state ==> presently executing**

s_d: **done state ==> completed execution**

Example

1	2	3	4	5	6	7	8	9	10	11	12	13
-	-	-	-	-	-	-	-	-	-	-	-	-
e	-	-	-	-	-	-	-	-	-	-	-	-
d	e	e	-	-	-	-	-	-	-	-	-	-
-	d	d	e	-	-	-	-	-	-	-	-	-
-	-	-	d	e	-	-	-	-	-	-	-	-
-	-	-	-	d	-	e	-	-	-	-	-	-
-	-	-	-	-	-	d	e	-	-	-	-	-
-	-	-	-	-	-	-	e	e	-	-	-	-
-	-	-	-	-	-	-	e	d	e	e	-	-
-	-	-	-	-	-	d	e	-	d	d	e	-
-	-	-	-	-	-	-	d	-	-	-	d	e
e	-	-	-	-	-	-	-	-	-	-	-	d



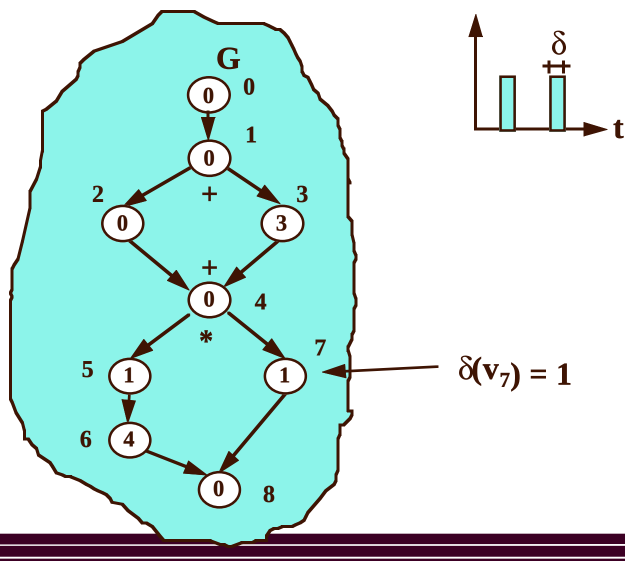
- reset
e enable
d done

No assumption about timing of the operations is made =>
(consecutive rows can be spaced arbitrarily over the time axis)

Timing Properties

- Operation delay
- Graph Latency
- Rate of Execution (operations)
- Rate of Reaction (graphs)

Operation Delay

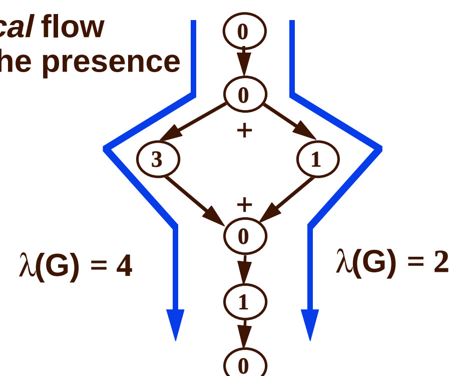


Latency

- Latency, $\lambda(G)$, of a graph model *G* refers to the execution delay of *G*

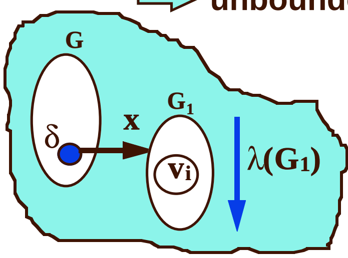
$$\lambda_k(G) = t_k(v_n) - t_k(v_0)$$

- the latency of a *non-hierarchical* flow graph may be variable due to the presence of conditional paths



Execution Delay of Link Vertices

- Given by
  - latency of the corresponding graph model times the number of times the called graph is invoked
  - execution delay of a link vertex can be
    - ⇨ variable
    - ⇨ unbounded (loop vertices with unbounded indices)



⇨ ***L**ink* **vertices: `call` and/or `loop` (point to other flow graphs in the hierarchy)**

Timing Properties

- Operation delay
- Graph Latency
- ⇨ Rate of Execution (operations)
- ⇨ Rate of Reaction (graphs)

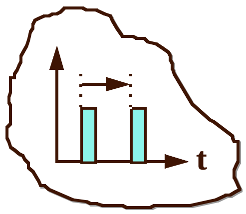
Rate of Execution (operations)

- assuming a *synchronous* execution model with cycle time τ,

⇨ the rate of execution at invocation *k* of operation v_i is given by the time interval between its current and previous execution

$$\rho_i(k) := \frac{1}{t_k(v_i) - t_{k-1}(v_i)} \quad (sec^{-1})$$

$$= \frac{\tau}{t_k(v_i) - t_{k-1}(v_i)} \quad (cycle^{-1})$$



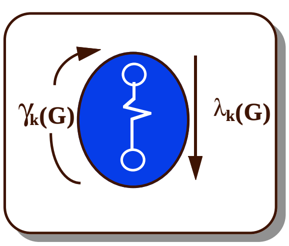
Rate of Reaction (Graphs)

- For a graph model, G, its *rate of reaction* is defined as the <u>rate of execution of its source operation</u>

$$\rho_G(k) := \rho_0(k)$$

The reaction rate is used to capture the effect on the run-time system of the type of implementation chosen for the graph model

- e.g., the choice of a non-pipelined implementation leads to

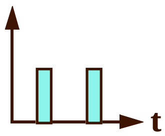


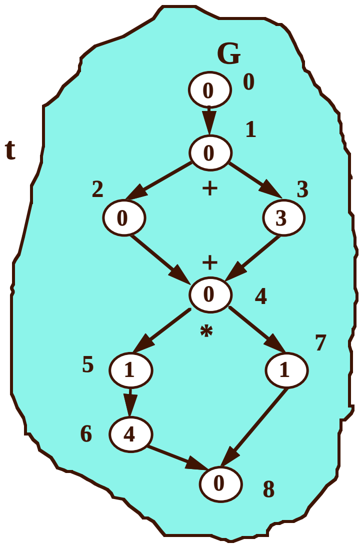
$$\rho_G(k)^{-1} = \lambda_k(G) + \gamma_k(G)$$

where $\gamma_k(G)$ represents the *overhead delay* (delay of reinvocation of G).

Timing Properties

- Operation delay
- Graph Latency
- Rate of Execution (operations)
- Rate of Reaction (graphs)





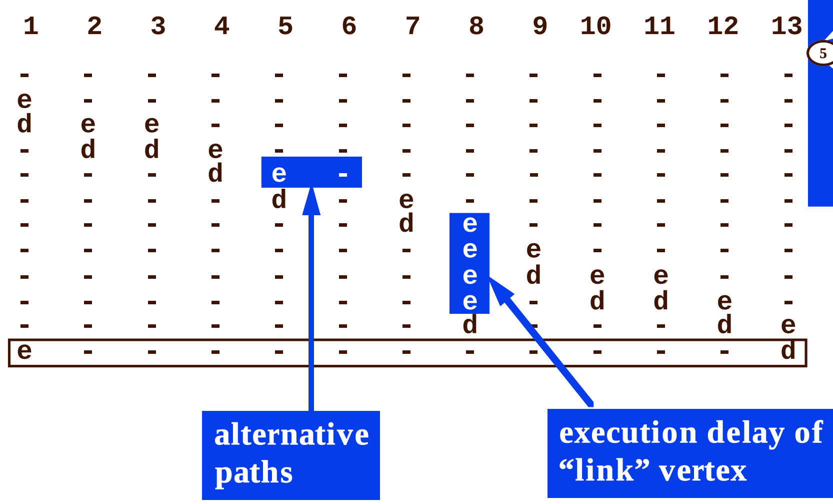
fixed, variable, bounded/unbounded

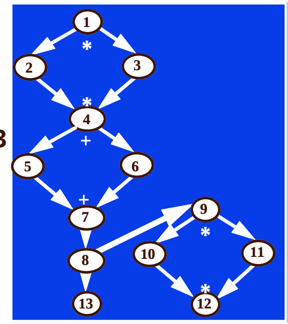
Non-Determinism

- the delay of an operation may be variable, depending on
  - the *value* of input data: e.g., loops with data dependent iteration counts, call vertices with conditionals
  - the *timing* of input data: e.g., wait operation
- the latency of a graph may be variable

Margarida Jacome - UT Austin - Spring 98

Data Dependent Delays





- ⇨ reset

e ⇨ enable

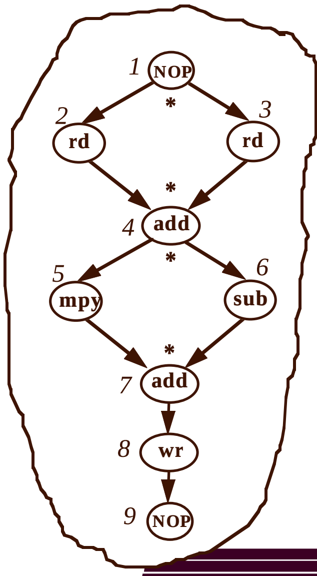
d ⇨ done

Non-determinism and Execution Rate

- *Data-dependent loop* and *synchronization* operations introduce *uncertainty* over
  - the precise *execution delay* of the model
  - the *order of execution* of the operations in the model

⇨ **Operations with *variable delays* are termed *non-deterministic delay* or *ND* operations.**

A Single Rate Model



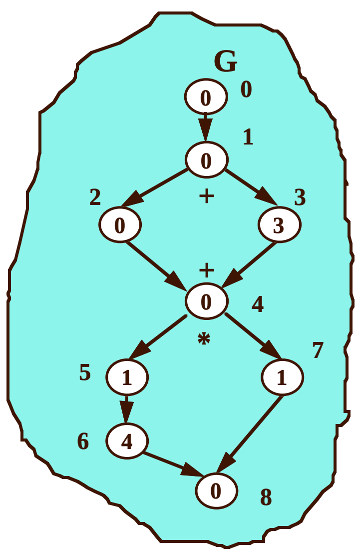
On each execution of the flow graph, each operation executes once

In this case, the *reaction rate of the graph G* is:

$\rho_G(k) := \rho_0(k) = \rho_{vi}(k)$, for all $v_i \in V(G)$ and for all $k \geq 0$

The execution of G proceeds at *single rate.*

A Multi Rate Model



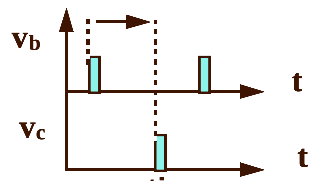
Timing Constraints and Constraint Analysis

- Timing Constraints
- Scheduling
- Constraint Satisfiability

Timing Constraints

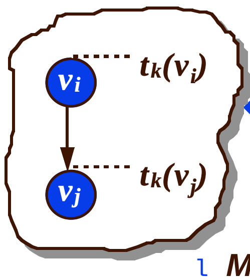
- *Operation delay* constraints
  - *unary*: bounds on the delay of an operation
  - *binary*: bounds on the delay between the starting time of two operations
- Execution rate constraints

Binary Delay Constraints



- *Minimum timing constraint*, $l_{ij} \geq 0$ from operation vertex v_i to v_j is defined as

$$t_k(v_j) \geq t_k(v_i) + l_{ij}$$

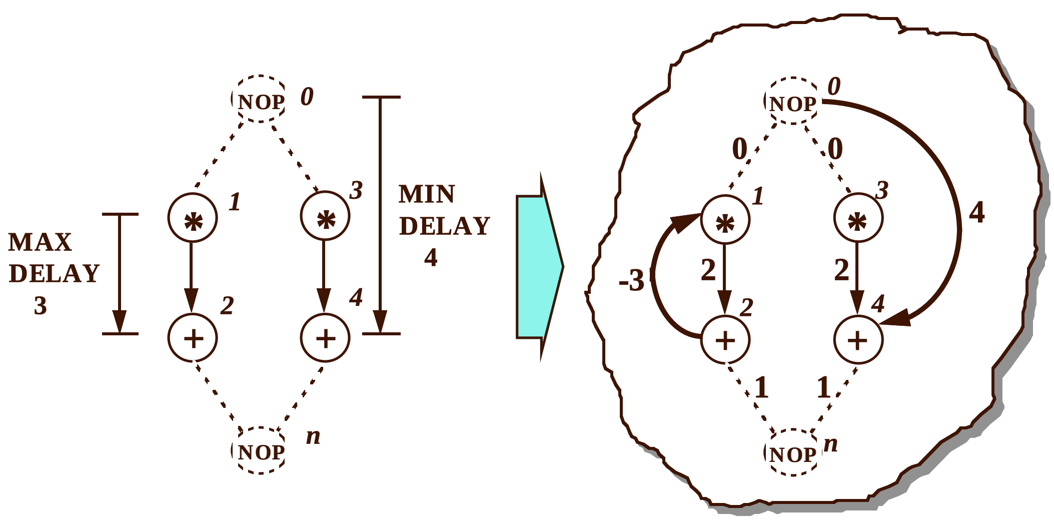


  - by default, any sequencing dependency between two operations induces a minimum timing constraint

- *Maximum timing constraint*, $u_{ij} \geq 0$ from operation vertex v_i to v_j is defined as

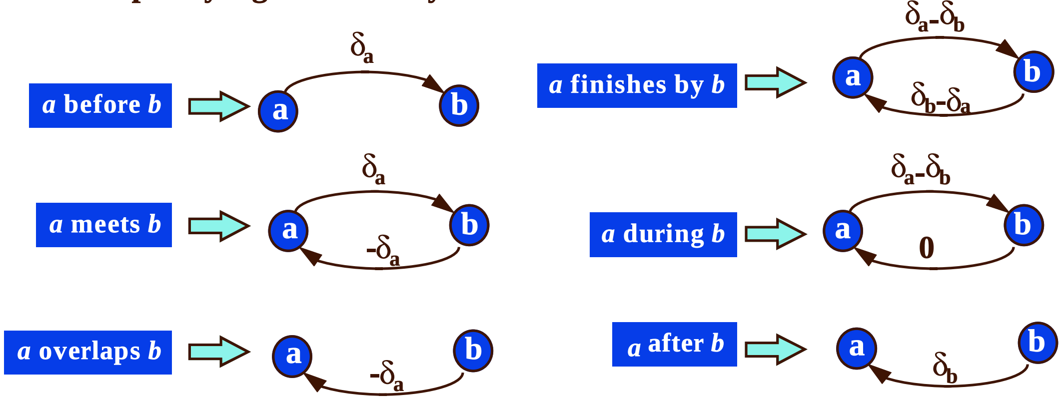
$$t_k(v_j) \leq t_k(v_i) + u_{ij}$$

Example



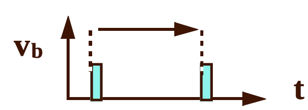
Operation Delay Constraints

- Can capture durational and deadline constraints in specifying real time systems



etc...

Timing Constraints



- Operation delay constraints
- Execution rate constraints
  - refer to constraints on the interval of time between successive executions of an operation
    - rate constraints on *input* (*output*) operations refer to the rates at which the data is required to be *produced* (*consumed*)

Margarida Jacome - UT Austin - Spring 98

Data Rate Constraints

- ***Minimum data rate constraint,*** r_i **(cycles^{-1}), on an input/output operation** v_i **defines a lower bound on the execution rate of the operation**

$$\rho_{v_i}(k) \geq r_i \qquad \forall k > 0 \qquad \text{[min rate]}$$

$$\Rightarrow t_k(v_i) - t_{k-1}(v_i) \leq \tau . r_i^{-1} \qquad \forall k > 0$$

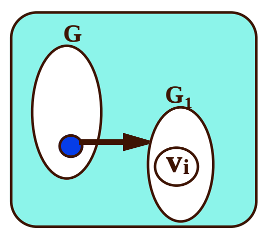
- ***Maximum data rate constraint,*** R_i **(cycles^{-1}), on an I/O operation** v_i **defines an upper bound on the execution rate of the operation**

$$\rho_{v_i}(k) \leq R_i \qquad \forall k > 0 \qquad \text{[max rate]}$$

$$\Rightarrow t_k(v_i) - t_{k-1}(v_i) \geq \tau . R_i^{-1} \qquad \forall k > 0$$

Ex.: Specification of Data Rate Constraints

```
process example (a,b,c)
      in port a[8],b[8];
      out port c[8];
{
       boolean x[8],y[8],z[8],w[8];
       tag A;
       x = read(a);
       y = read(b);
       z = x * y;
       w = x + y;
       while(z >= 0) {
          while(w >= 0)  {
A:                 write c = y;
                   w = w - 1; }
            z = z - w;
            write c = z;    }
 attribute "constraint minrate of A = 100 cycles/sample"
 attribute "constraint minrate 0 of A = 1 cycles/sample"
 attribute "constraint minrate 1 of A = 10 cycles/sample"
}
```



1

0

relative min constraints -- indexed by the corresponding loops

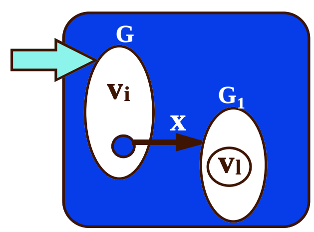
r = 0.01 per cycle

Timing Constraints and Constraint Analysis

- Timing Constraints
- ⇨ Scheduling
- Constraint Satisfiability

Margarida Jacome - UT Austin - Spring 98

Scheduling



- For each invocation of a flow graph model, an operation is invoked zero, one, or many times depending upon its *position on the hierarchy* of the flow model

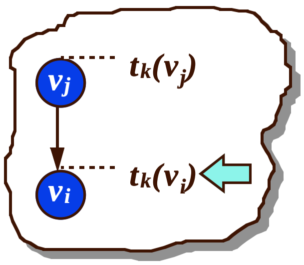
⇨ **The execution times $t_k(v)$ of an operation v are determined by two separate mechanisms**

- **The runtime scheduler, γ**
  - ⇨ **determines the invocation time of flow graphs**
- **The operation scheduler, Ω**

Margarida Jacome - UT Austin - Spring 98

Timing Constraints and Scheduling

- Given a scheduling function, a timing constraint is considered *satisfied* if
  - the operation starting times determined by the scheduling function satisfy the inequalities



$t_k(v_j) \geq t_k(v_i) + l_{ij}$ [min delay]

$t_k(v_j) \leq t_k(v_i) + u_{ij}$ [max delay]

$\rho_{vi}(k) \leq R_i$ [max rate]

$\rho_{vi}(k) \geq r_i$ [min rate]

Relative Scheduler

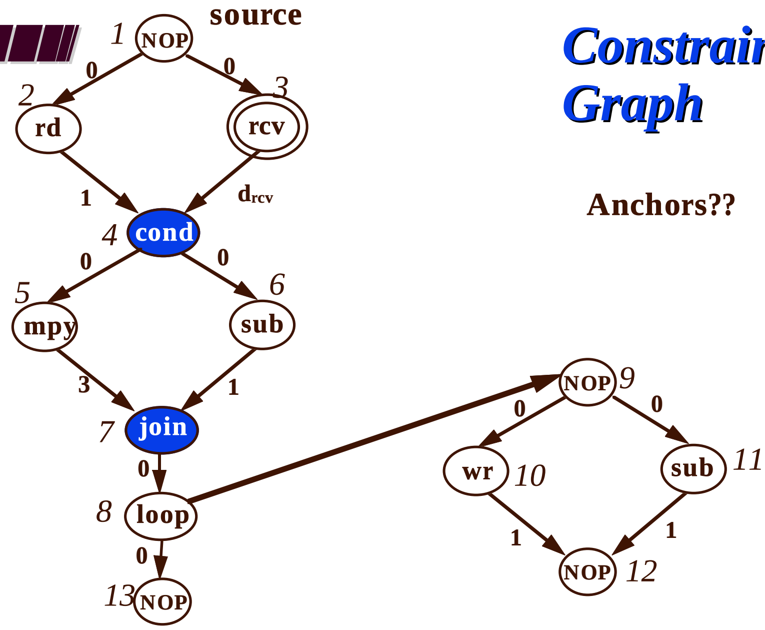
- For a given vertex v_i a set $A(v_i)$ of *anchor* vertices is defined as the set of conditional (CD) and loop, wait (ND) vertices that have a path to v_i

$$A(v_i) = \{v_j \in V : v_j >^* v_i,\ v_j \text{ is ND or CD}\}$$

- A *relative schedule function* Ω_r is defined as a set of offsets for each operation such that the operation start time satisfies

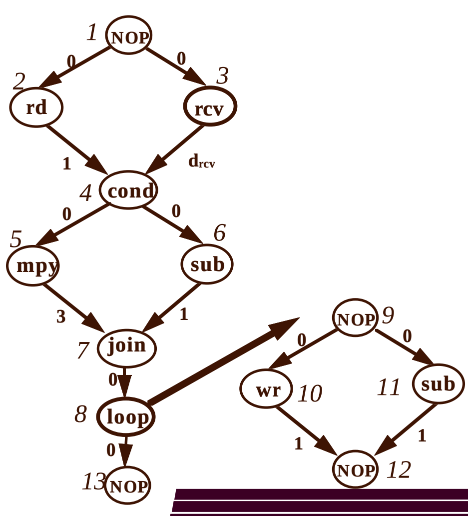
$$t_k(v_i) \geq \max_{a \in A(v_i)} [\, t_k(a) + \delta(a) + \theta_a(v_i)]$$

Constraint Graph



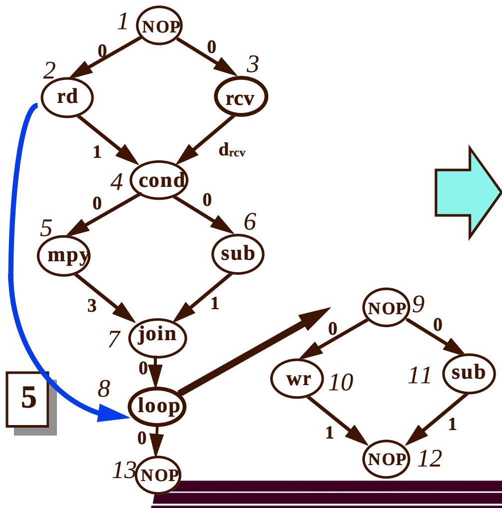
Anchors??

Modified Relative Schedule



Vertex	Relative Offset v1	v3	v8	v9
v1	--	--	--	
v2	0	--	--	
v3	0	--	--	
v4	1	0	--	
v5	1	0	--	
v6	1	0	--	
v7	(2,4)	(1,3)	--	
v8	(2,4)	(1,3)	--	
v9	(2,4)	(1,3)	--	
v10	--	--	--	0
v11	--	--	--	0
v12	--	--	--	1
v13	--	--	0	--

Modified Relative Schedule



Vertex	Relative Offset v_1	v_3	v_8	v_9
v_1	--	--	--	
v_2	0	--	--	
v_3	0	--	--	
v_4	1	0	--	
v_5	1	0	--	
v_6	1	0	--	
v_7	4	3	--	
v_8	~~4~~ 5	3	--	
v_9	~~4~~ 5	3	--	
v_{10}	--	--	--	0
v_{11}	--	--	--	0
v_{12}	--	--	--	1
v_{13}	--	--	0	--

Constraint Satisfiability

- For constraint analysis purposes, it is not necessary to determine a schedule of the operations, but only to *verify* the *existence* of a schedule

Constraint satisfiability ⇨ **identifying conditions under which no solution (i.e., schedule) exists**

ND operations

- In the presence of *ND* operations
  - satisfiability analysis attempts to determine the existence of a schedule of operations for all possible (and conceivable) values of the delay of the *ND* operations

Modified relative scheduling - Min/Max Delay Constraints

A minimum delay constraint is always satisfiable

$$t_k(v_i) \geq \max_{a \in A_b(v_i)} [\, t_a + \delta(a) + |\theta_a(v_i)|_\infty]$$

For each constraint l*ij* solution can be constructed such that

$$\theta_{vj}(v_i) \geq \max(l(v_j, v_j), l_{ij})$$

A maximum delay constraint may not always be satisfiable

Satisfiability - Delay Constraints

Feasibility:

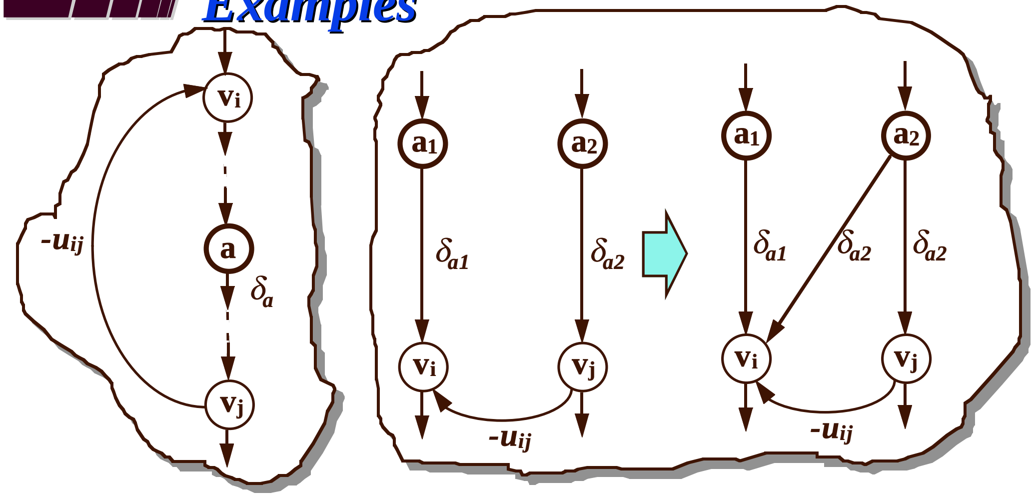
A constraint graph is considered *feasible* if it contains *no positive cycle* when the delay of the ND operations is assigned to zero.

Condition necessary and sufficient to determine the *satisfiability* of constraints in the presence of *ND* operations:

Operation delay constraints are *satisfiable* if and only if

- **the constraint graph is feasible**
- **there exists no cycles with ND operations**

Examples



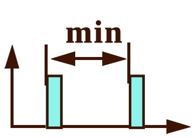
Constraints are not satisfiable (maybe feasible) | can be modified such that... ⇒ Constraints are satisfiable

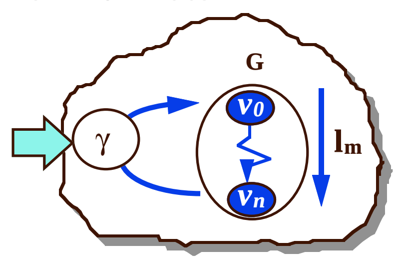
Max Rate Constraints

- A max-rate constraint, R_i, in G is satisfied if

$$l_m(G) \geq R_i^{-1}$$

- As with minimum delay constraints, maximum rate constraints are always satisfiable





- when the lower bound $l_m(G) \leq R_i^{-1}$ the max-rate constraint can still be satisfied by an appropriate choice of *overhead delay* that is applicable to every execution of G

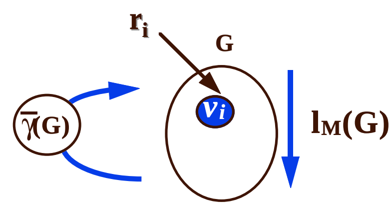
Min Rate Constraint

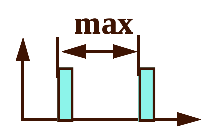
- A minimum rate constraint r_i on an operation $v_i \in V(G)$, where G contains no ND operations is satisfiable if

$$\bar{\gamma}(G) + l_M(G) \leq (\tau / r_i)$$

bound on latency

overhead delay



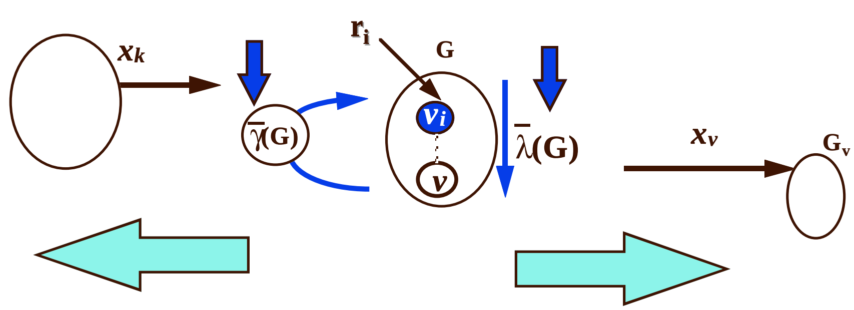


- A minimum rate constraint places an <u>upper bound</u> on the interval of successive executions of an operation

Min Rate Constraints

- General case: involves two bounds

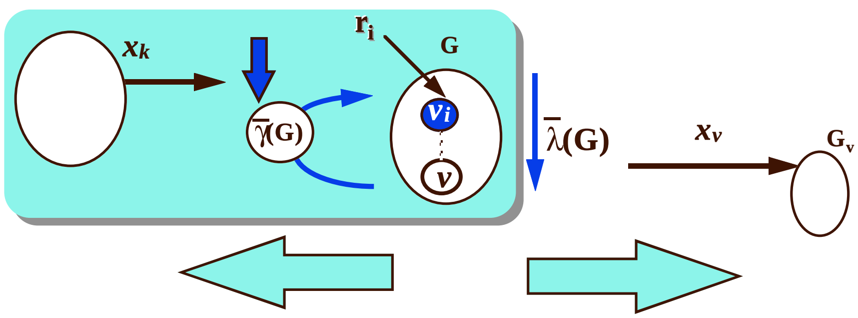
$$\bar{\gamma}(G) + \bar{\lambda}(G) \leq (\tau / r_i)$$



Min Rate Constraints

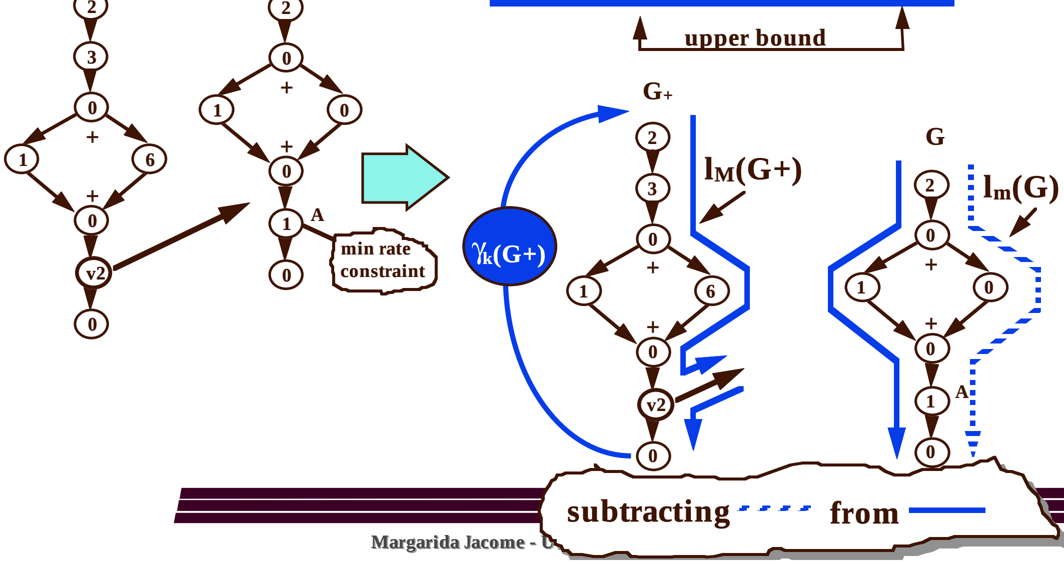
- General case: involves two bounds

$\overline{\gamma}(G) + \overline{\lambda}(G) \leq (\tau / r_i)$

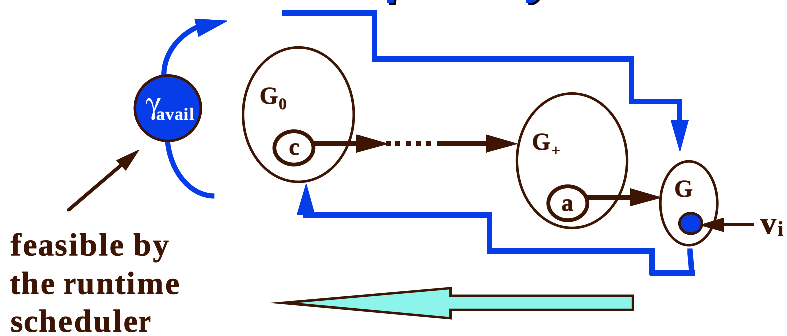


Upper Bound on Overhead Delay

$\overline{\gamma}(G) := [l_M(G+) + \gamma(G+)] - l_m(G)$



Min Rate: satisfiability

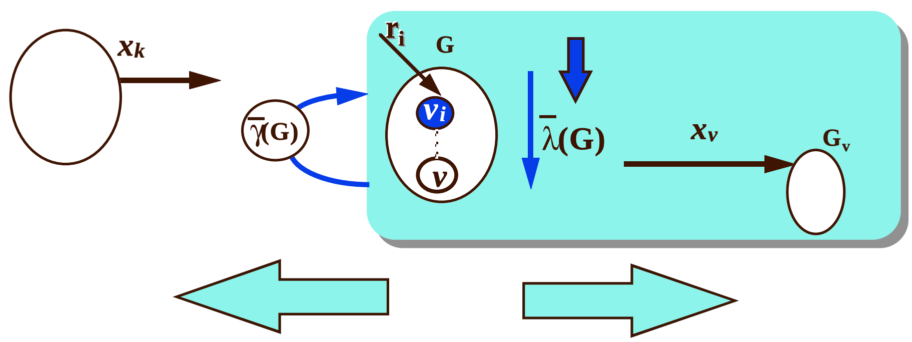


Max delay (min rate) between two executions of v_i occurs when the entire hierarchy is traversed with just one execution of the link operations that lead to v_i.

Min Rate Constraints

General case: involves two bounds

$\overline{\gamma}(G) + \overline{\lambda}(G) \leq (\tau / r_i)$

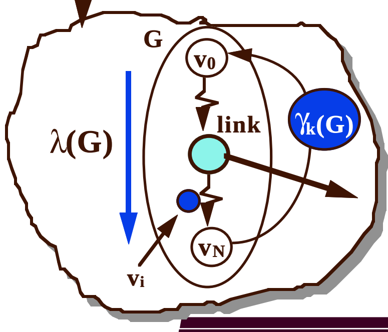


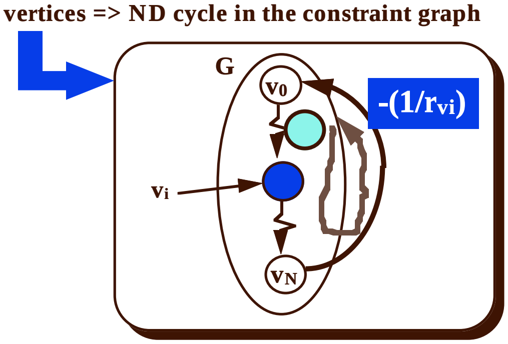
Min Rate Constraints (with ND operations)

In the presence of *ND* operations in G:

- **The latency $\overline{\lambda}(G)$ needs to be bounded**

  - relative rate constraints -- represented as a backward edge (i.e., max delay constraint) from G's sink to source vertices => ND cycle in the constraint graph





ND Operations: Data-dependent Loops

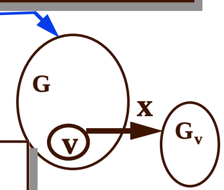
- **The *loop index* determines the number of times the loop body is invoked for each invocation of the loop link operation**

=> delay of the loop operation is its loop index times the latency of the loop body

If the constrained graph (G) contains <u>at most one</u> loop operation, v, on a path from source to sink

The *minimum rate constraint* can be seen as a bound on the number of times the loop body (G) is invoked.

bound on loop index, $\overline{x}$



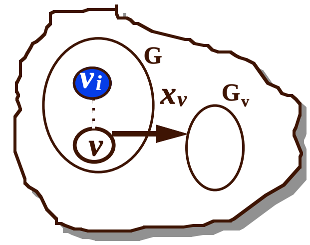
Satisfiability of Min Rate Constraints

Consider a flow graph *G* with an *ND* operation *v* representing a loop in the flow graph

- **A minimum rate constraint r_i on operation $v_i \in V(G)$ and $v_i \neq v$ is satisfiable if the loop index, x_v indicating the number of times G_v is invoked for each execution of *v* is less than the bound**

3

$$\bar{x}_v := \left\lfloor \frac{\tau \cdot r_i^{-1} - \bar{\gamma}(G) - l_M(G) + \mu(v)}{l_M(G_v)} \right\rfloor + 1$$

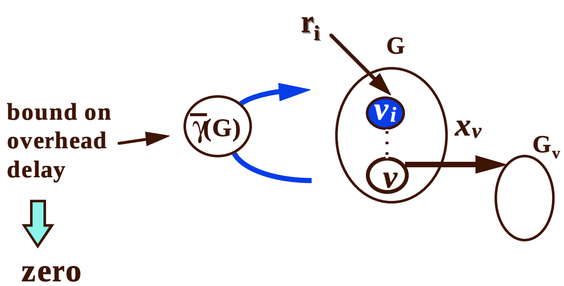


$\mu(v)$ ⇨ **mobility of operation v**

defined as the difference between the longest path that goes through v and l_M

Relative Min Rate Constraints

- **Relative min rate constraint relative to G ==> applied when G is enabled and executing**



zero

$$\bar{x}_v := \left\lfloor \frac{\tau \cdot r_i^{-1} - \bar{\gamma}(G) - l_M(G) + \mu(v)}{l_M(G_v)} \right\rfloor + 1$$

Constraint Satisfiability in Vulcan

1. Construct the Constraint Graph
   - add forward edges for *minimum delay* and *maximum rate* constraints
   - add backward edges for *maximum delay* and (relative) *minimum rate* constraints
2. Identify *cycles* by path enumeration for each of the *backward edges* in the constraint graph

   ⇒ check for constraint satisfiability, bound delays, etc.
3. Propagate *minimum rate* constraints up the graph hierarchy