ELE6XXE
Lectures

Prof. Dr. Müştak E. Yalçın

Istanbul Technical University

mustak.yalcin@itu.edu.tr

Scheduling Data Flow graph

Scheduling involves assigning every node of the DFG to control time steps or clock cycles.
Schedule:

- – Creating the sequence in which nodes fire
- – Determines number of clock cycles required

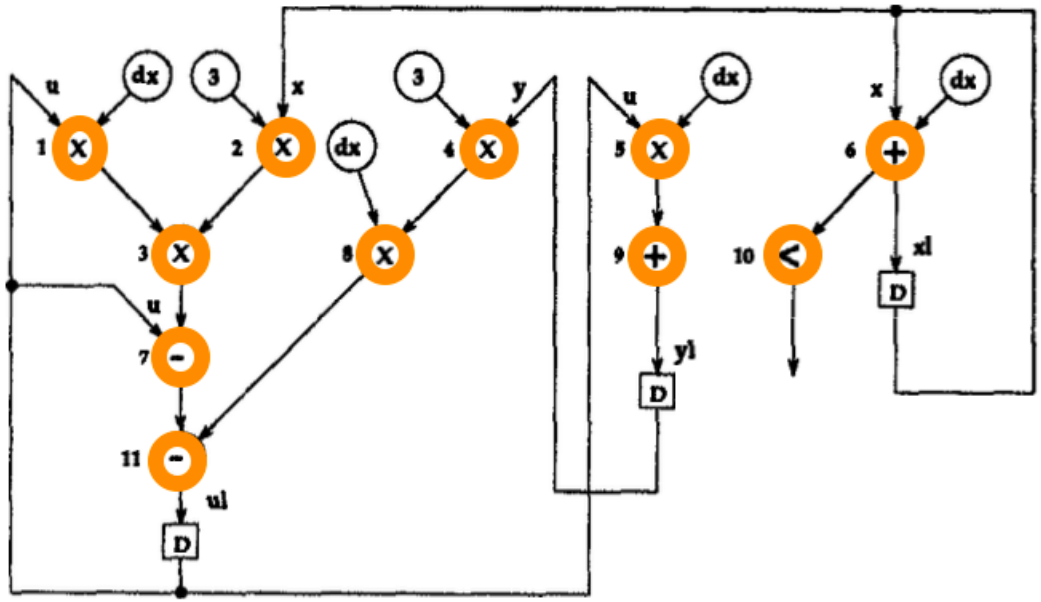
Two simple schedules:

- – As-soon-as-possible (ASAP) schedule puts every operation as early in time as possible
- – As-late-as-possible(ALAP)scheduleputs every operation as late in schedule as possible

Chapter 5: Giovanni De Micheli - Synthesis and Optimization of Digital Circuits-McGraw-Hill Science_Engineering_Math (1994)

Solve the differential equation $\ddot{y} + 3z\dot{y} + 3y = 0$.
This can be calculated using this iterative algorithm

```
while(z < a) {
zl := z + dz;
ul := u-(3 · z · u · dz)- (3 · y · dz);
yl := y + (u · dz);
z := zl;
u := ul;
y := yl;
}
```



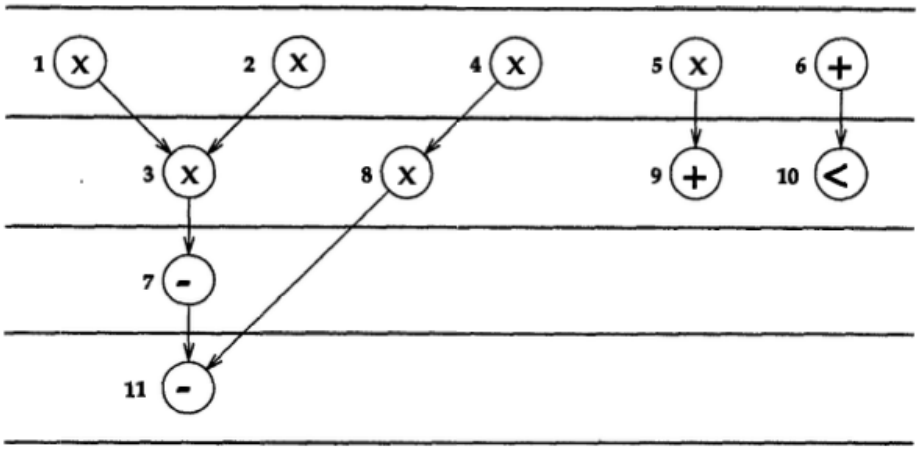
precedence graph

ASAP Scheduling

As-soon-as-possible (ASAP) schedule puts every operation as early in time as possible

- Unlimited resources (No limit for the number of registers, adders, etc.)
- Longest path through data flow determines minimum schedule length

Input: DFG $G = (N, E)$.
Output: ASAP Schedule.
1. $TS_0 = 1$; /* Set initial time step */
2. While (Unscheduled nodes exist) {
   2.1 Select a node n_j whose predecessors have already been scheduled;
   2.2 Schedule node n_j to time step $TS_j = \max\{TS_i + (C_i)\}$ $\forall\ n_i \rightarrow n_j$;
   }

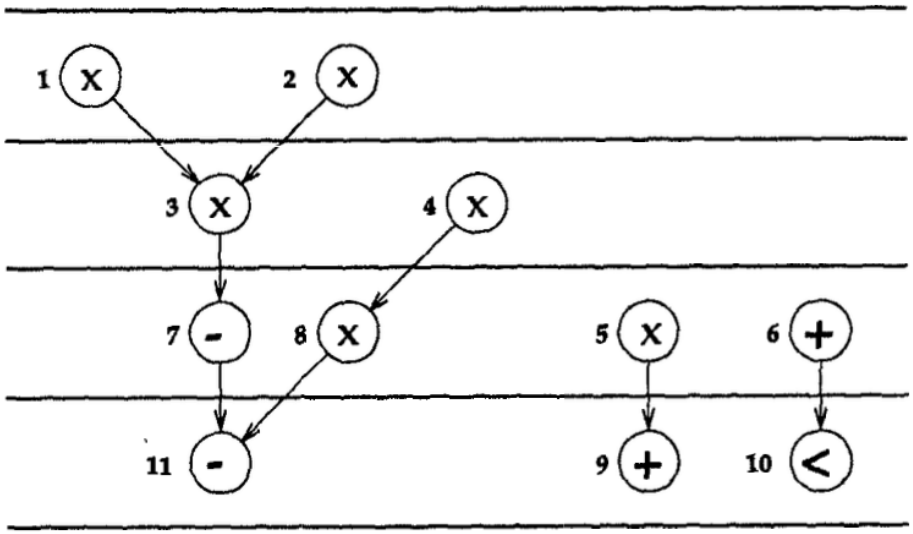


$C_i = 1$, draw when the execution time of mul is 2 and the rest is 1.

ALAP Scheduling

As-late-as-possible(ALAP)schedule puts every operation as late in schedule as possible

Input: DFG $G = (N, E)$, $IterationPeriod = T$.
Output: ALAP Schedule.
1. $TS_0 = T$; /* Set initial time step */
2. While (Unscheduled nodes exist) {
   2.1 Select a node n_i whose successors have already been scheduled;
   2.2 Schedule node n_i to time step $TS_i = \min\{ TS_j - (C_i)\}$ $\forall\ n_i \rightarrow n_j$;
   }



ALAP & ASAP Scheduling

- No consideration given to resource constraints
- No priority is given to nodes on critical path
- As a result, less critical nodes may be scheduled ahead of critical nodes
  - No problem if unlimited hardware is available
  - However if the resources are limited, the less critical nodes may block the critical nodes and thus produce inferior schedules
- List scheduling techniques overcome this problem by utilizing a more global node selection criterion
- Mobility (slack): The node mobility represents its flexibility in the fire sequence the differnce of the start times computed by ALAS and ASAP alg. $(t_L - t_S)$.

List Scheduling Algorithms

- Algorithm 1: Minimize latency under resource constraint (ML-RC)
  - Resource constraint represented by vector a (indexed by resource type)
    Example: two types of resources, MULT (a1=1), ADD (a2=2)
- Algorithm 2: Minimize resources under latency constraint (MR-LC)
  - Latency constraint is given and resource constraint vector a to be minimized

Define (V: vertex, E: edge):

- The candidate operations $U_{l,k}$
  - those operations of type k whose predecessors have already been scheduled early enough (completed at step l):
    $U_{l,k} = \{v_i \subseteq V : \text{type}(v_i) = k\}$ and $t_j + d_j \leq l$, for all $j : (v_j, v_i) \subseteq E$
- The unfinished operations $T_{l,k}$
  - those operations of type k that started at earlier cycles but whose execution has not finished at step l (multi-cycle opearations):
    $T_{l,k} = \{v_i \subseteq V : \text{type}(v_i) = k\}$ 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

List Scheduling Algorithm 1: ML-RC

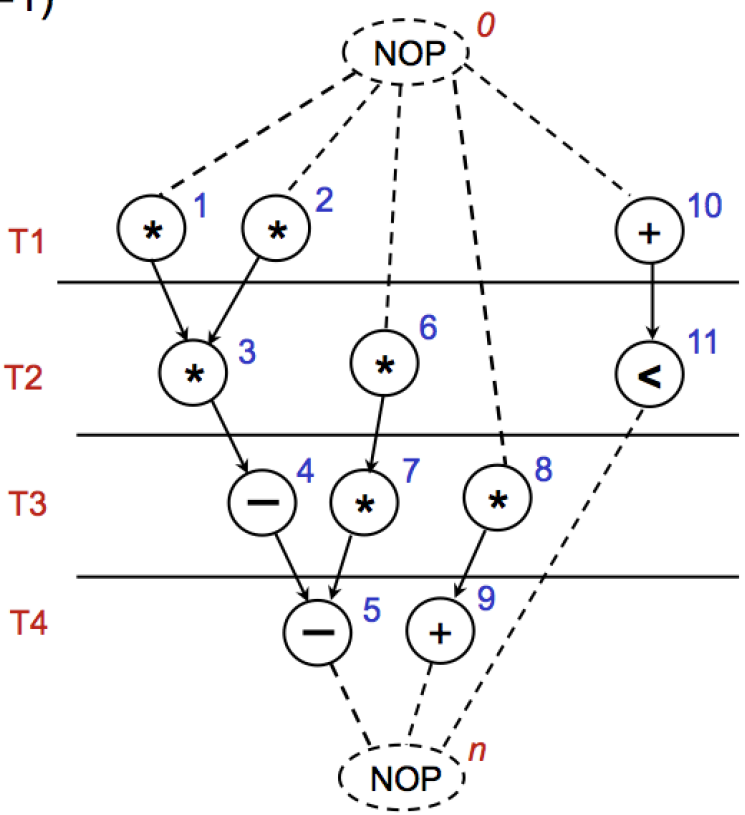
Minimize latency under resource constraint

```
LIST_L( G_s(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, such that |S_k| + |T_{l,k}| ≤ a_k;
            Schedule the S_k operations at step l by setting t_i = l ∀i : v_i ∈ S_k;
        }
        l = l + 1;
    }
    until (v_n is scheduled);
    return (t);
}
```

Note: If for all operators i, di = 1 (unit delay), the set Tl,k is empty

List Scheduling Algorithm 1: ML-RC

- **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



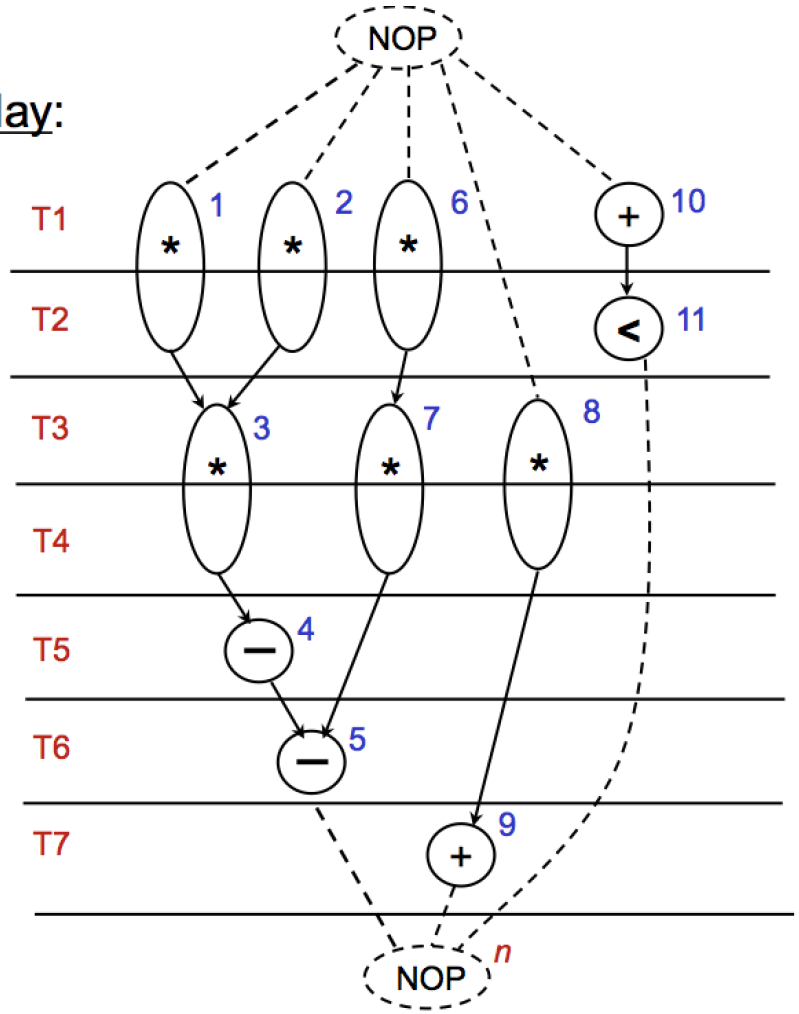
Minimize latency under resource constraint (with d = 1)

List Scheduling Algorithm 1: ML-RC

- **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
U= $\{v_1, v_2, v_6\}$	v_{10}	1
T= $\{v_1, v_2, v_6\}$	v_{11}	2
U= $\{v_3, v_7, v_8\}$	--	3
T= $\{v_3, v_7, v_8\}$	--	4
--	v_4	5
--	v_5	6
--	v_9	7

Latency L = 8



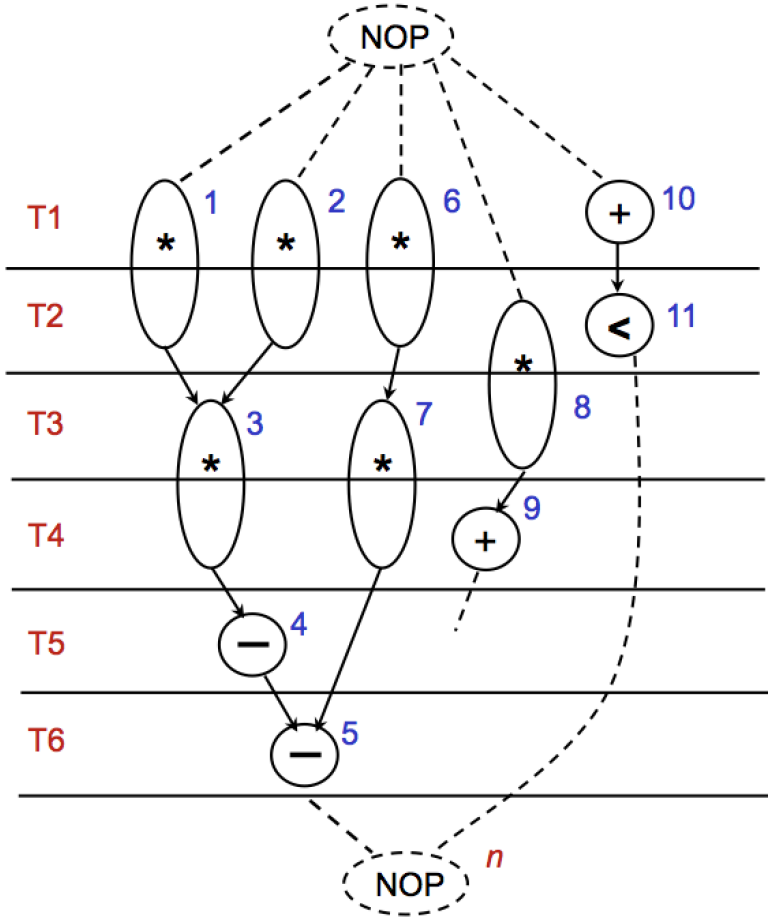
Minimize latency under resource constraint (with $d_1 = 2$, $d_2 = 1$)

List Scheduling Algorithm 1: ML-RC

- **Assumptions**
  - Multipliers are pipelined
  - Sharing between first and second pipeline stage allowed for different multipliers

MULT	ALU	start time
$\{v_1, v_2, v_6\}$	v_{10}	1
v_8	v_{11}	2
$\{v_3, v_7\}$	--	3
--	v_9	4
--	v_4	5
--	v_5	6

- *L=7*, Compare to multi-cycle case



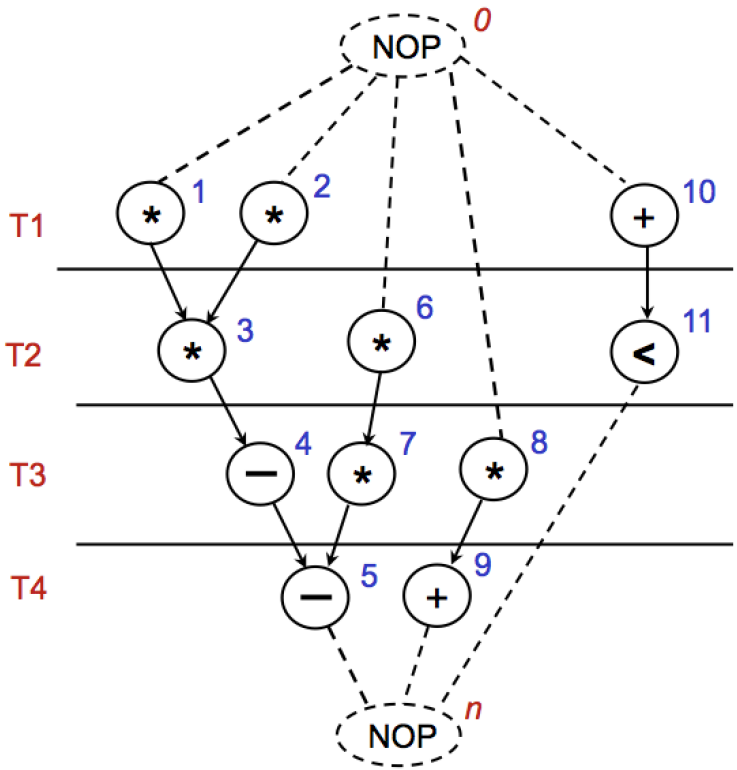
Minimize latency under resource constraint (a1=3 MULTs, a2=3 ALUs)

List Scheduling Algorithm 2: MR-LC

```
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 requiring no additional resources;
        }
        l = l + 1;
    }
    until (v_n is scheduled);
    return (t, a);
}
```

List Scheduling Algorithm 2: MR-LC

- 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
  - Add other operations only if current resource count allows
    - e.g. add *6 with *3, without exceeding current Mult=2
  - The lower the slack the more urgent it is to schedule the operation



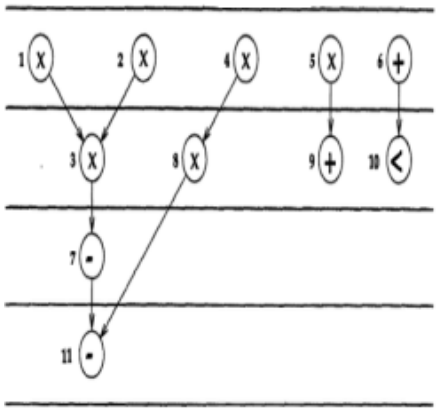
The Force-Directed Scheduling

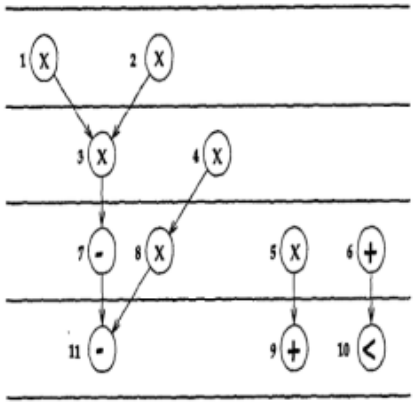
The intent of the ▸ force-directed scheduling algorithm
is to reduce the number of functional units, registers, and buses required by balancing the concurrency of the operations assigned to them but without lengthening the total execution time.
The Force-Directed Scheduling algorithm:

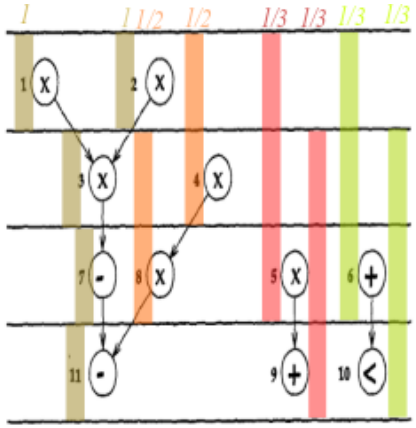
- Compute mobility of operations using ASAP and ALAP $\mu_i = {t_i}^L - {t_i}^S$
- Compute operation probabilities $(p_i(l) = \frac{1}{1+\mu_i}$ for $l \in [{t_i}^S, {t_i}^L])$
- Compute operation-type distribution $q_k(l) = \sum_{i \in T(v_i)=k} p_i(l)$ where $T(v_i)$ is the set of type of operation
- Calculate the total force
- Select and schedule operations with least force
- Update time-frame
- repeat until all op.s are scheduled.

The Force-Directed Scheduling

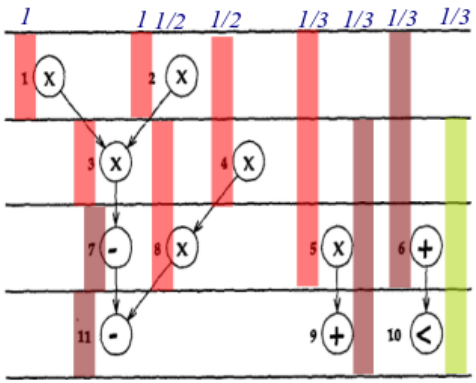
p_i

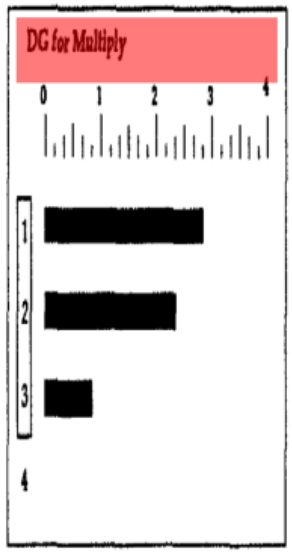


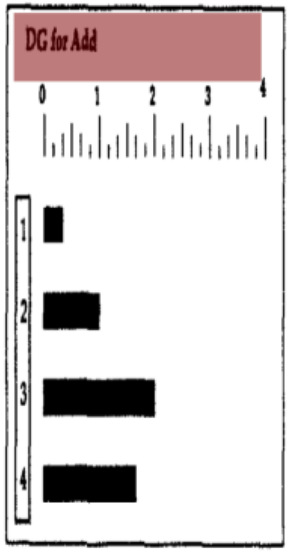




q_k







The Force-Directed Scheduling

Cost Function:Self Force (the effect on the overall concurrency by attempting to schedule a node i to the time step l)

$$F_{i,l} = \sum_{m={t_i}^S}^{{t_i}^L} q_k(m)(\delta_{l,m} - p_i(m))$$

$$F_{4,l} = \sum_{m={t_4}^S}^{{t_4}^L} q_k(m)(\delta_{l,m} - p_4(m)) = \sum_{m=1}^{2} q_k(m)(\delta_{l,m} - p_4(m))$$
$$= q_k(1)(\delta_{l,1} - p_4(1)) + q_k(2)(\delta_{l,2} - p_4(2))$$

$$F_{4,1} = q_k(1)(\delta_{1,1} - p_4(1)) + q_k(2)(\delta_{1,2} - p_4(2)) = 2.83(1 - .5) + 2.33(0 - 0.5)$$

$$F_{4,1} = 0.25, \quad F_{4,2} = -0.25$$

it means the concurrency at step 1 of the MULT is higher than at step 2.

The Force-Directed Scheduling

- If node 4 is scheduled into time step 1, its successor is scheduled into step 2 or 3!, $\sum F = F_{4,1} + F_{8,2} + F_{8,3} = F_{4,1} = 0.25$ (node 4 does not affect the time frame for its successor)
- If node 4 is scheduled into time step 2, its successor is scheduled into step 3, $\sum F = F_{4,2} + F_{8,3} = -0.25 - 0.75$