Instruction Scheduling - Part 1

Y.N. Srikant

Department of Computer Science
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Compiler Design

Outline

- Instruction Scheduling
  - Simple Basic Block Scheduling
  - Automaton-based Scheduling
  - Integer programming based scheduling
  - Optimal Delayed-load Scheduling (DLS) for trees
  - Trace, Superblock and Hyperblock scheduling

Instruction Scheduling

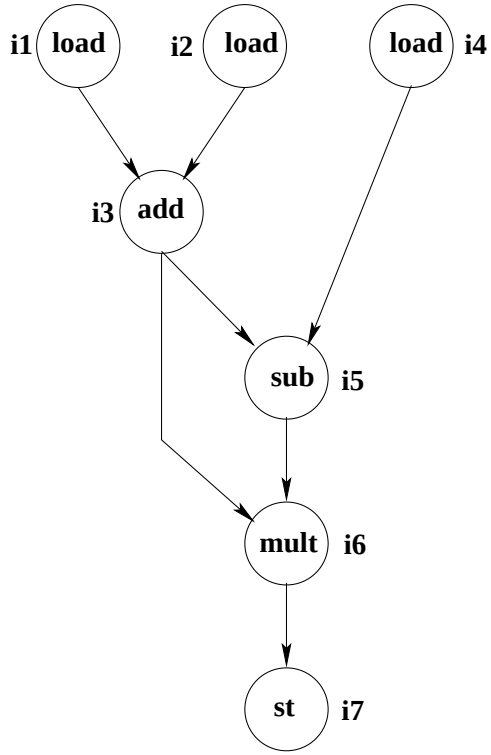
- Reordering of instructions so as to keep the pipelines of functional units full with no stalls
- NP-Complete and needs heuristcs
- Applied on basic blocks (local)
- Global scheduling requires elongation of basic blocks (super-blocks)

Instruction Scheduling - Motivating Example

- time: load - 2 cycles, op - 1 cycle
- This code has 2 stalls, at i3 and at i5, due to the loads

i1:	r1	←	load a
i2:	r2	←	load b
i3:	r3	←	r1 + r2
i4:	r4	←	load c
i5:	r5	←	r3 - r4
i6:	r6	←	r3 * r5
i7:	d	←	st r6

(a) Sample Code Sequence



(b) DAG

Scheduled Code - no stalls

- There are no stalls, but dependences are indeed satisfied

i1:	r1	←	load a
i2:	r2	←	load b
i4:	r4	←	load c
i3:	r3	←	r1 + r2
i5:	r5	←	r3 - r4
i6:	r6	←	r3 * r5
i7:	d	←	st r6

Definitions - Dependences

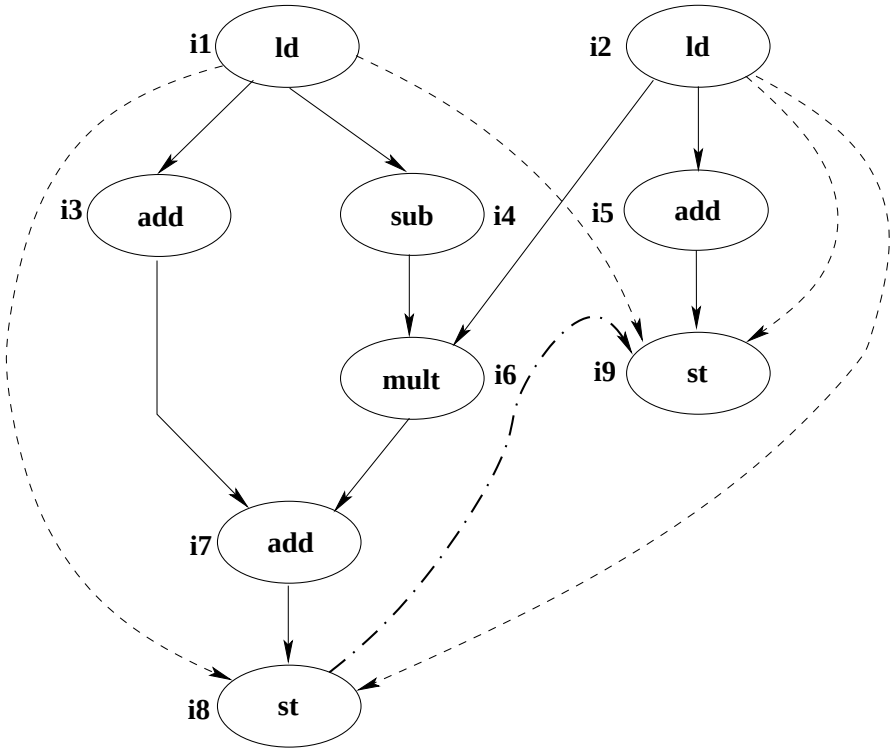
- Consider the following code:
  $i_1 : \ r1 \leftarrow load(r2)$
  $i_2 : \ r3 \leftarrow r1 + 4$
  $i_3 : \ r1 \leftarrow r4 + r5$
- The dependences are
  $i_1 \ \delta \ i_2$ (flow dependence) $i_2 \ \overline{\delta} \ i_3$ (anti-dependence)
  $i_1 \ \delta^o \ i_3$ (output dependence)
- anti- and ouput dependences can be eliminated by register renaming

Dependence DAG

- full line: *flow* dependence, dash line: *anti*-dependence
  dash-dot line: *output* dependence
- some anti- and output dependences are because memory disambiguation could not be done

i1:	t1	←	load a
i2:	t2	←	load b
i3:	t3	←	t1 + 4
i4:	t4	←	t1 - 2
i5:	t5	←	t2 + 3
i6:	t6	←	t4 * t2
i7:	t7	←	t3 + t6
i8:	c	←	st t7
i9:	b	←	st t5

(a) Instruction Sequence



(b) DAG

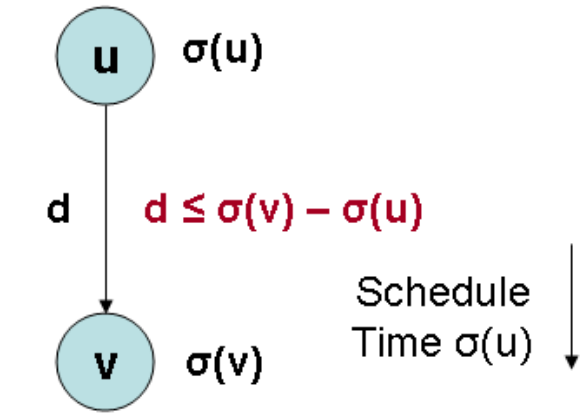
Basic Block Scheduling

- Basic block consists of micro-operation sequences (MOS), which are indivisible
- Each MOS has several steps, each requiring resources
- Each step of an MOS requires one cycle for execution
- Precedence constraints and resource constraints must be satisfied by the scheduled program
  - PC's relate to data dependences and execution delays
  - RC's relate to limited availability of shared resources

The Basic Block Scheduling Problem

- Basic block is modelled as a digraph, $G = (V, E)$
  - R: number of resources
  - Nodes (V): MOS; Edges (E): Precedence
  - Label on node v
    - resource usage functions, $\rho_v(i)$ for each step of the MOS associated with v
    - length $l(v)$ of node v
  - Label on edge e: Execution delay of the MOS, $d(e)$
- Problem: Find the shortest schedule $\sigma : V \rightarrow N$ such that $\forall e = (u, v) \in E,\ \sigma(v) - \sigma(u) \geq d(e)$ and $\forall i, \sum_{v \in V} \rho_v(i - \sigma(v)) \leq R$, where length of the schedule is $\max_{v \in V}\{\sigma(v) + l(v)\}$

Instruction Scheduling - Precedence and Resource Constraints



Precedence constraint

Schedule Time σ(u)

MOS substeps (time)

	0	1	2	3	4
$\sigma(v_1)=0$	1	1	2	2	
$\sigma(v_2)=1$	2	3	1	1	2
$\sigma(v_3)=2$	3	1	2		
$\sigma(v_4)=3$	1	2	3	2	

Resource constraint

Consider R = 5. Each MOS substep takes 1 time unit.

At i=4, $\varsigma_{v4}(1)+\varsigma_{v3}(2)+\varsigma_{v2}(3)+\varsigma_{v1}(4) = 2+2+1+0 = 5 \leq R$, satisfied

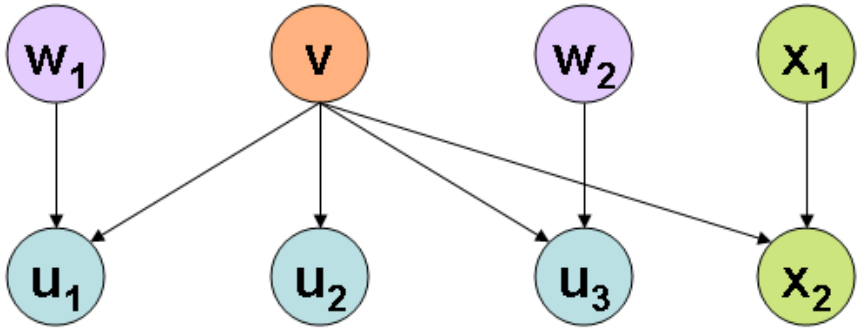
At i=2, $\varsigma_{v3}(0)+\varsigma_{v2}(1)+\varsigma_{v1}(2) = 3+3+2 = 8 > R$, NOT satisfied

A Simple List Scheduling Algorithm

Find the shortest schedule $\sigma : V \rightarrow N$, such that precedence and resource constraints are satisfied. Holes are filled with NOPs.

FUNCTION ListSchedule (V,E)
BEGIN
  Ready = root nodes of V; *Schedule* = ϕ;
  WHILE *Ready* $\neq \phi$ DO
  BEGIN
    v = highest priority node in *Ready*;
    Lb = SatisfyPrecedenceConstraints (v, *Schedule*, σ);
    σ(v) = SatisfyResourceConstraints (v, *Schedule*, σ, *Lb*);
    $Schedule = Schedule + \{v\}$;
    $Ready = Ready - \{v\} + \{u \mid NOT\ (u \in Schedule)$
      $AND\ \forall\ (w,u) \in E,\ w \in Schedule\}$;
  END
  RETURN σ;
END

List Scheduling - Ready Queue Update



Already scheduled nodes — w

Unscheduled nodes which will get into the Ready queue now — u

Currently scheduled node — v

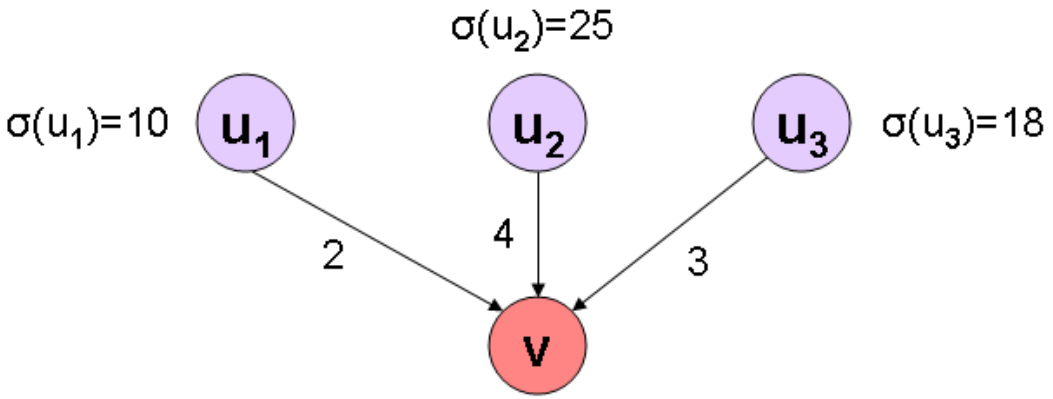
Unscheduled nodes — x

Constraint Satisfaction Functions

FUNCTION SatisfyPrecedenceConstraint(v, Sched, σ)
BEGIN
  RETURN ($\max_{u \in Sched} \sigma(u) + d(u, v)$)
END

FUNCTION SatisfyResourceConstraint(v, Sched, σ, Lb)
BEGIN
  FOR i := Lb TO ∞ DO
    IF $\forall 0 \leq j < l(v),\ \rho_v(j) + \sum_{u \in Sched} \rho_u(i + j - \sigma(u)) \leq R$ THEN
      RETURN (i);
END

Precedence Constraint Satisfaction



Lower bound for σ(v) = 29

Already scheduled nodes **u**

Node to be scheduled **v**

Precedence constraint satisfaction:

v can be scheduled only after all of u_1, u_2, and, u_3, finish

Lower bound for σ(v)
= max(10+2, 25+4, 18+3)
= max(12, 29, 21) = 29

Resource Constraint Satisfaction

Resource constraint satisfaction
Consider R = 5. Each MOS substep takes 1 time unit.

MOS substeps (time)

Schedule Time σ(u) ↓

	0	1	2	3	4
$\sigma(v_1)=0$	1	1	2	2	
$\sigma(v_2)=1$	2	3	1	1	2
2					
3					
$\sigma(v_3)=4$	3	1	2		
$\sigma(v_4)=5$	1	2	3	2	

Time slots 2 and 3 are vacant because scheduling node v_3 in either of them violates resource constraints

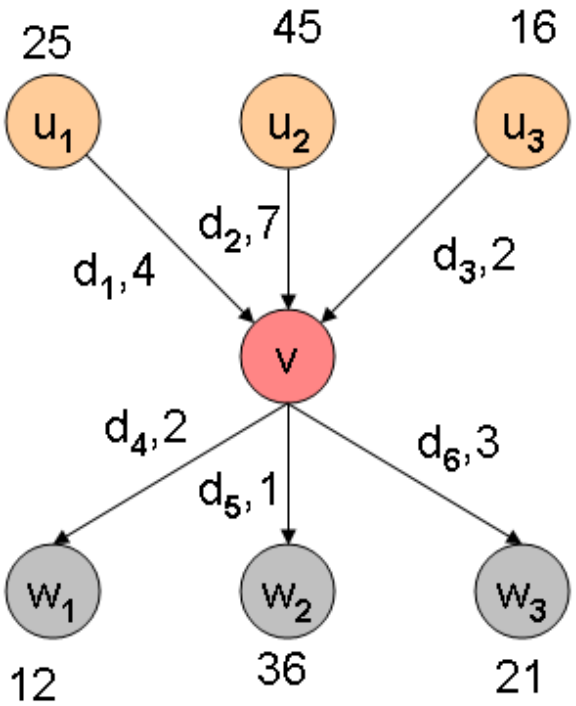
List Scheduling - Priority Ordering for Nodes

1. Height of the node in the DAG (*i.e.,* longest path from the node to a terminal node
2. *Estart*, and *Lstart*, the earliest and latest start times
   - Violating *Estart* and *Lstart* may result in pipeline stalls
   - $Estart(v) = \max_{i=1,\cdots,k}(Estart(u_i) + d(u_i, v))$

     where $u_1, u_2, \cdots, u_k$ are predecessors of v. *Estart* value of the source node is 0.
   - $Lstart(u) = \min_{i=1,\cdots,k}(Lstart(v_i) - d(u, v_i))$

     where $v_1, v_2, \cdots, v_k$ are successors of u. *Lstart* value of the sink node is set as its *Estart* value.
   - *Estart* and *Lstart* values can be computed using a top-down and a bottom-up pass, respectively, either statically (before scheduling begins), or dynamically during scheduling

Estart and Lstart Computation



$$Estart\ (v) = \max_{i=1,\ldots,3} (Esart\ (u_i) + d_i)$$

$= \max(25+4,\ 45+7,\ 16+2)$

$= \max(29,\ 52,\ 18) = 52$

$$Lstart\ (v) = \min_{i=4,\ldots,6} (Lsart\ (w_i) - d_i)$$

$= \min(12-2,\ 36-1,\ 21-3)$

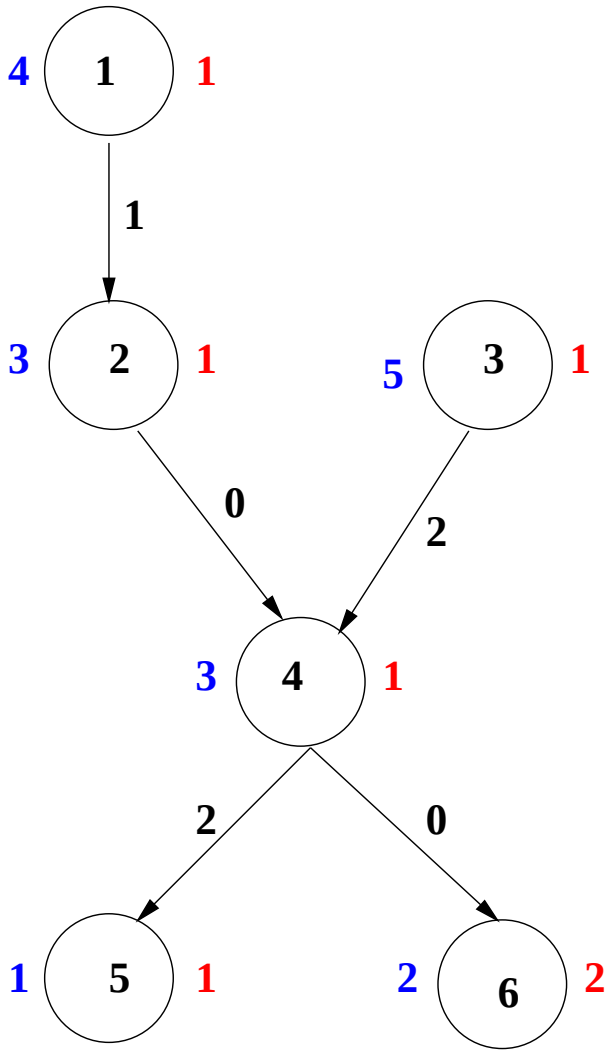
$= \min(10,\ 35,\ 18) = 10$

List Scheduling - Slack

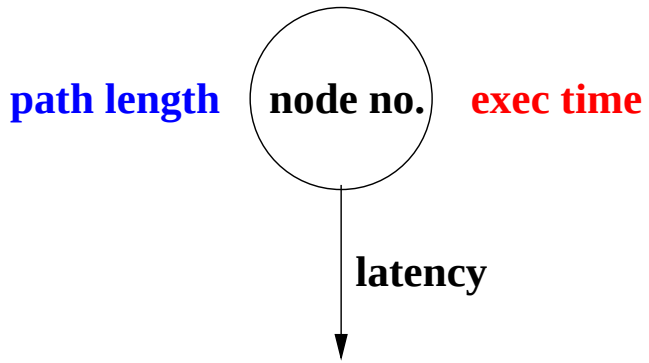
1. A node with a lower *Estart* (or *Lstart*) value has a higher priority
2. $Slack = Lstart - Estart$
   - Nodes with lower slack are given higher priority
   - Instructions on the critical path may have a slack value of zero and hence get priority

Simple List Scheduling - Example - 1

INSTRUCTION SCHEDULING - EXAMPLE



LEGEND



path length (n) = exec time (n) , if n is a leaf

= max { latency (n,m) + path length (m) }
m ε succ (n)

Schedule = {3, 1, 2, 4, 6, 5}

Simple List Scheduling - Example - 2

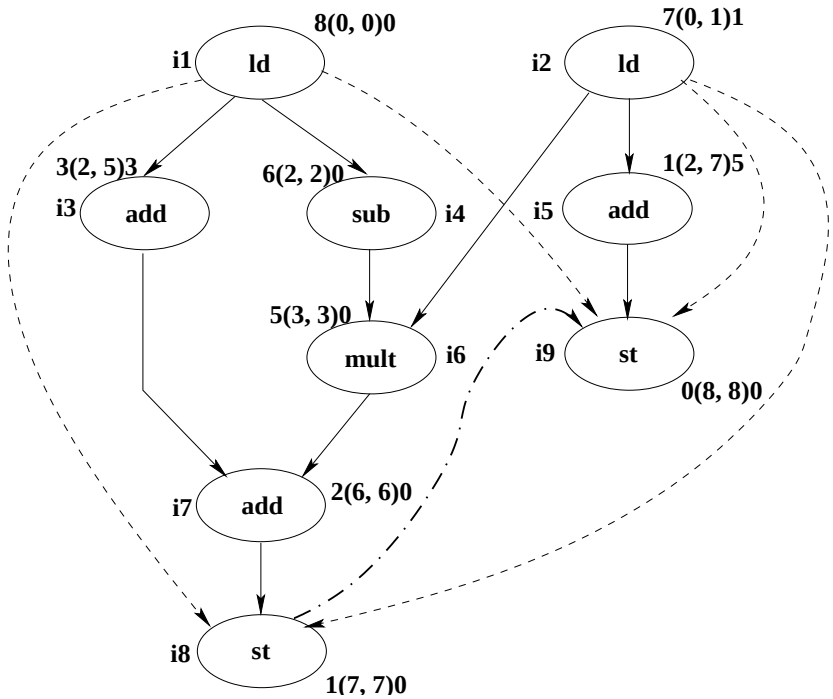
- latencies
  - *add,sub,store*: 1 cycle; *load*: 2 cycles; *mult*: 3 cycles
- *path length* and *slack* are shown on the left side and right side of the pair of numbers in parentheses

```
c = (a+4)+(a-2)*b;
b = b+3;
```

(a) High-Level Code

i1:	t1	←	load a
i2:	t2	←	load b
i3:	t3	←	t1 + 4
i4:	t4	←	t1 - 2
i5:	t5	←	t2 + 3
i6:	t6	←	t4 * t2
i7:	t7	←	t3 + t6
i8:	c	←	st t7
i9:	b	←	st t5

(b) 3-Address Code



(c) DAG with *(Estart, Lstart)* Values