List Scheduling

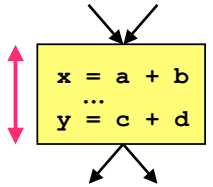
Slides by Todd C. Mowry
Updated by Peter Lee
CS745: Optimizing Compilers

List Scheduling

- The most common technique for scheduling instructions within a basic block

We don't need to worry about:
- control flow

We do need to worry about:
- data dependences
- hardware resources



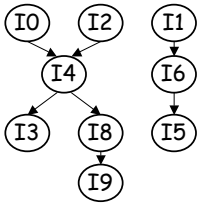
- Even without control flow, the problem is still NP-hard

List Scheduling Algorithm: Inputs and Outputs

Algorithm reproduced from:
- "*An Experimental Evaluation of List Scheduling*", Keith D. Cooper, Philip J. Schielke, and Devika Subramanian. Rice University, Department of Computer Science Technical Report 98-326, September 1998.

Inputs:

Data Precedence Graph (DPG)



Machine Parameters

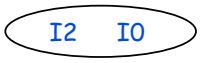
# of FUs:
2 INT, 1 FP
Latencies:
add = 1 cycle, …
Pipelining:
1 add/cycle, …

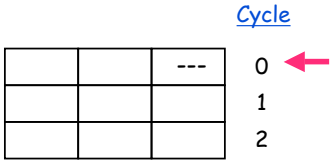
Output:

Scheduled Code			Cycle
I0	I2	---	0
---	I1	I4	1
I3	I8	I6	2
I10	---	I11	3
I7	I9	I5	4

List Scheduling: The Basic Idea

- Maintain a list of instructions that are ready to execute
  - data dependence constraints would be preserved
  - machine resources are available
- Moving cycle-by-cycle through the schedule template:
  - choose instructions from the list & schedule them
  - update the list for the next cycle





What Makes Life Interesting: Choice

Easy case:

- all ready instructions can be scheduled this cycle

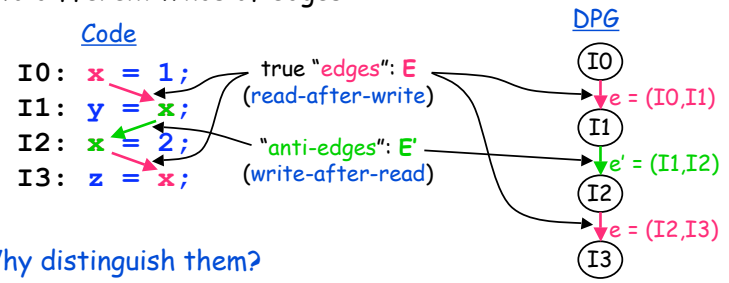
Interesting case:

- we need to pick a subset of the ready instructions

- List scheduling makes choices based upon *priorities*
  - assigning priorities correctly is a key challenge

Representing Data Dependences: The Data Precedence Graph (DPG)

- Two different kinds of edges:

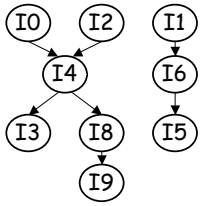


- Why distinguish them?
  - do they affect scheduling differently?
- What about output dependences?

Computing Priorities

- Let's start with just true dependences (i.e. "edges" in DPG)
- Priority = *latency-weighted depth* in the DPG

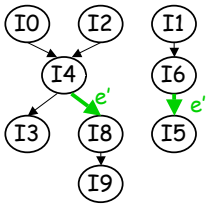
$$priority(x) = max(\forall_{l \in leaves(DPG)} \forall_{p \in paths(x,\ldots,l)} \sum_{p_i=x}^{l} latency(p_i))$$



Computing Priorities (Cont.)

- Now let's also take anti-dependences into account
  - i.e. anti-edges in the set E'

$$priority(x) = \begin{cases} latency(x) & \text{if } x \text{ is a leaf} \\ max(latency(x) + max_{(x,y) \in E}(priority(y)), & \\ \quad max_{(x,y) \in E'}(priority(y))) & otherwise. \end{cases}$$



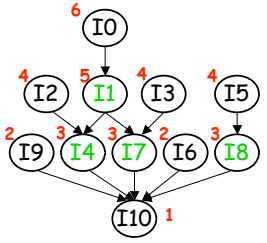
List Scheduling Algorithm

```
cycle = 0;
ready-list = root nodes in DPG; inflight-list = {};

while ((|ready-list|+|inflight-list| > 0) && an issue slot is available) {
    for op = (all nodes in ready-list in descending priority order) {
        if (an FU exists for op to start at cycle) {
            remove op from ready-list and add to inflight-list;
            add op to schedule at time cycle;
            if (op has an outgoing anti-edge)
                add all targets of op's anti-edges that are ready to ready-list;
        }
    }
    cycle = cycle + 1;
    for op = (all nodes in inflight-list)
        if (op finishes at time cycle) {
            remove op from inflight-list;
            check nodes waiting for op & add to ready-list if all operands available;
        }
    }
}
```


Example

```
I0: a = 1
I1: f = a + x
I2: b = 7
I3: c = 9
I4: g = f + b
I5: d = 13
I6: e = 19;
I7: h = f + c
I8: j = d + y
I9: z = -1
I10: JMP L1
```



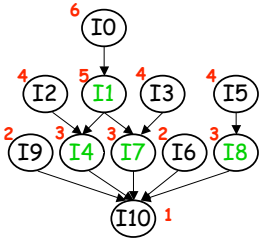
Cycle

		0
		1
		2
		3
		4
		5
		6

- 2 identical fully-pipelined FUs
- adds take 2 cycles; all other insts take 1 cycle

Example

```
I0: a = 1
I1: f = a + x
I2: b = 7
I3: c = 9
I4: g = f + b
I5: d = 13
I6: e = 19;
I7: h = f + c
I8: j = d + y
I9: z = -1
I10: JMP L1
```



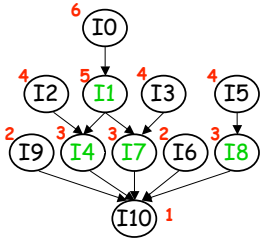
Cycle

I0	I2	0
I1	I3	1
I5	I9	2
I4	I7	3
I8	I6	4
---	---	5
I10		6

- 2 identical fully-pipelined FUs
- adds take 2 cycles; all other insts take 1 cycle

What if We Break Ties Differently?

```
I0: a = 1
I1: f = a + x
I2: b = 7
I3: c = 9
I4: g = f + b
I5: d = 13
I6: e = 19;
I7: h = f + c
I8: j = d + y
I9: z = -1
I10: JMP L1
```



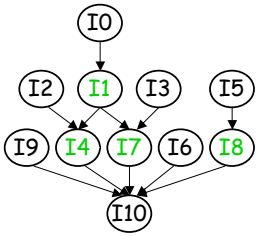
Cycle

		0
		1
		2
		3
		4
		5
		6

- 2 identical fully-pipelined FUs
- adds take 2 cycles; all other insts take 1 cycle

What if We Break Ties Differently?

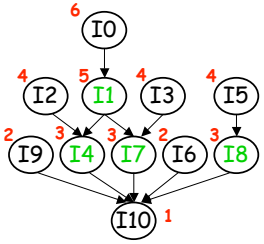
```
I0: a = 1
I1: f = a + x
I2: b = 7
I3: c = 9
I4: g = f + b
I5: d = 13
I6: e = 19;
I7: h = f + c
I8: j = d + y
I9: z = -1
I10: JMP L1
```



		Cycle
I0	I2	0
I1	I5	1
I3	I8	2
I4	I7	3
I9	I6	4
I10		5
		6

- 2 identical fully-pipelined FUs
- adds take 2 cycles; all other insts take 1 cycle

Contrasting the Two Schedules



		Cycle
I0	I2	0
I1	I3	1
I5	I9	2
I4	I7	3
I8	I6	4
---	---	5
I10		6

		Cycle
I0	I2	0
I1	I5	1
I3	I8	2
I4	I7	3
I9	I6	4
I10		5

- Breaking ties arbitrarily may not be the best approach

Backward List Scheduling

Modify the algorithm as follows:

- reverse the direction of all edges in the DPG
- schedule the *finish times* of each operation
  - start times must still be used to ensure FU availability

Impact of scheduling backwards:

- clusters operations near the end (vs. the beginning)
- may be either better or worse than forward scheduling