Christoph Kessler with Mattias Eriksson

DF00100
Advanced Compiler Construction
VT1 / 2010

Exercise Set 3

Exercise 3.1

In the lecture on local instruction scheduling we described the list scheduling algorithm as a *forward scheduling* algorithm, that is, topological sorting in forward direction of the dependences.

Give the pseudocode of *backward* list scheduling.

What is the heuristic node priority criterion in backward list scheduling that corresponds to the *deepest-level-first* criterion in forward scheduling?

Exercise 3.2

Given the following basic block:

live on entry: a, b (assume for simplicity that they are in registers)

$c \leftarrow a + b$

$d \leftarrow a - c$

$e \leftarrow 2 * a$

$f \leftarrow c$

$g \leftarrow a/f$

live at exit: d, e, g (to be kept in registers)

Identify live ranges and draw the register interference graph. How many colors do you need? Apply coalescing where possible to eliminate copy operations. Given a processor with $R = 3$ physical registers, compute a register assignment using Chaitin's *degree*$< R$–heuristic and insert necessary spill code.

Exercise 3.3

Assume you have a VLIW processor with three functional units, a memory access unit for loads (latency 4cc) and stores (latency 1cc), a multiplier for multiplications (latency 2cc), and an ALU for all other operations (latency 1cc). All units have occupation time 1 cc. Hence, load has 3 delay slots, multiplication and branch has 1 delay slot, and the other operations have no delay slot. All multiplications must be done on the multiplier. You can issue one long instruction word per clock cycle, each containing (at most) one subinstruction per functional unit.

Given the following target level program representation (i.e., after instruction selection) of the main loop of an integer dot product program, shown here as naively scheduled pseudo-targetcode for the architecture described above:

`Loop:`		// invariant: loop counter in symb. reg. $v1$
S_1	$v2 \leftarrow$ `mult` #4, $v1$	// integer multiplication
	(`nop`)	// multiplication delay slot
S_2	$v3 \leftarrow$ `load 0x18f0(`$v2$`)`	// integer load $A[i]$
S_3	$v4 \leftarrow$ `load 0x1a20(`$v2$`)`	// integer load $B[i]$
	(`nop`)	// load delay slot
	(`nop`)	// load delay slot
	(`nop`)	// load delay slot
S_4	$v5 \leftarrow$ `mult` $v3$, $v4$	// integer multiplication
	(`nop`)	// multiplication delay slot
S_5	$v6 \leftarrow$ `add` $v6$, $v5$	// integer addition
S_6	$v1 \leftarrow$ `add` $v1$, #1	// integer addition
S_7	$v7 \leftarrow$ compare $v1$, #100	// integer compare to loop counter in $v1$
S_8	`branch` to `Loop` if $v7! = 0$	// differ $\rightarrow$ do next of 100 iterations
	(`nop`)	// branch delay slot – even if branch is taken
`Exit:`		// now the dot product result is in $v6$

(a) Draw the data dependence graph for the loop. (*Hint:* Do not forget anti-dependences, loop-carried dependences and control dependences.)

(b) Suggest (any) better (in terms of execution time) schedule for a single loop iteration than the naive schedule given above.

(c) Give the largest lower bound for the minimum initiation interval based on resource constraints.

(d) Give the largest lower bound for the minimum initiation interval based on recurrence constraints.

(e) Construct a modulo reservation table for the initiation interval $II = 5$ and construct a modulo schedule. (*Hint:* Note that v_1 can be overwritten by S_6 as soon as S_1 has executed for 1 cycle (early operand read phase). Begin placing instructions with the longest dependence *cycle*.)

Please prepare these exercises for the third lesson.