332 Parallel Architectures

Unassessed tutorial exercise 3

Pipelining and the SAXPY loop

Many important applications make heavy use of a loop with the following form:

```
for i := 1 to 100
  Y[i] := a*X[i] + Y[i];
```

For example this is the central operation in Gauss elimination, and is called "SAXPY" (Single-precision aX Plus Y). Here is a DLX implementation:

```
foo:  LD     F2, 0(R1)        ; load X[i]
      MULTD  F4,F2,F0         ; multiply a*X[i]
      LD     F6, 0(R2)        ; load Y[i]
      ADDD   F6,F4,F6         ; add aX[i] + Y[i]
      SD     0(R2),F6         ; store Y[i]
      ADDI   R1,R1,8          ; increment X index
      ADDI   R2,R2,8          ; increment Y index
      SGTI   R3,R1,done       ; test if finished (R3 := (R1>done))
      BEQZ   R3,foo           ; loop if not finished.
```

Make the following assumptions:

- Integer operations issue and complete in one clock cycle.
- There are no memory system delays.
- There is no branch delay.
- FP addition takes 2 cycles, FP multiplication takes 5 cycles, and FP division takes 19 cycles (although you will not need division).

Exercise 3.1. Static pipeline timing

Assume a static pipeline with separate, non-pipelined FP units, one for addition/subtraction, and one for multiplication/division. Assume full bypassing. Show a timing diagram for the loop. How many clock cycles does each iteration of the loop take?

Exercise 3.2. Loop unrolling

Unwind the loop given above three times. Change the registers used in each of the three versions of the loop body, so that no registers are reused. Eliminate any obviously redundant computations. Notice that you can use the displacement addressing mode instead of repeatedly incrementing R1 and R2.

On the static-schedule machine (Question 3.1 above), how many clock cycles does each iteration of the loop take now, on average? How many clock cycles would each iteration take if you unrolled the loop completely?

Exercise 3.3. Compile-time pipeline scheduling

See if you can modify the sequence of instructions in the unrolled loop to reduce the number of stalls. How many clock cycles does each iteration take now?

(These exercises are based on Hennessy and Patterson, first edition, Exercises 6.11–19).

Paul Kelly, Imperial College, 2001

Notes on the solutions to unassessed exercise 3

Exercise 3.1. Static pipeline timing

Assume a static pipeline with separate, non-pipelined FP units, one for addition/subtraction, and one for multiplication/division. Assume full bypassing. Show a timing diagram for the loop. How many clock cycles does each iteration of the loop take?

Clock:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
`LD F2, 0(R1)`	IF	ID	EX	M	WB													
`MULTD F4,F2,F0`		IF	ID	*stall*	EX	EX	EX	EX	EX	WB								
`LD F6, 0(R2)`			IF	*stall*	ID	EX	M	WB										
`ADDD F6,F4,F6`					IF	ID	*stall*	*stall*	*stall*	EX	EX	WB						
`SD 0(R2),F6`						IF	*stall*	*stall*	*stall*	ID	EX	M						
`ADDI R1,R1,8`										IF	ID	EX	M	WB				
`ADDI R2,R2,8`											IF	ID	EX	M	WB			
`SGTI R3,R1,done`												IF	ID	EX	M	WB		
`BEQZ R3,foo`													IF	ID	EX	M	WB	

Since we are asked to ignore the branch delay, the first instruction of the next iteration of the loop can begin after 13 cycles. Finally we check to see whether the instructions still being executed then will cause any hazards which could stall instructions in the next iteration. There are none, so the iteration time is 13 cycles.

Exercise 3.2. Loop unrolling

Unwind the loop given above three times. Do not reuse any registers. Eliminate any obviously redundant computations. Notice that you can use the displacement addressing mode instead of repeatedly incrementing R1 and R2.

This is what you get if you duplicate the loop body three times, chop out the branches and labels leaving just those at the end, and use the displacement field of the load and store instructions instead of repeatedly incrementing R1 and R2:

```
foo: LD     F2, 0(R1)       ; load X[i]
     MULTD  F4,F2,F0        ; multiply a*X[i]
     LD     F6, 0(R2)       ; load Y[i]
     ADDD   F6,F4,F6        ; add aX[i] + Y[i]
     SD     0(R2),F6        ; store Y[i]
     LD     F12, 8(R1)      ; load X[i+1]
     MULTD  F14,F12,F0      ; multiply a*X[i+1]
     LD     F16, 8(R2)      ; load Y[i+1]
     ADDD   F16,F14,F16     ; add aX[i+1] + Y[i+1]
     SD     8(R2),F16       ; store Y[i+1]
     LD     F22, 16(R1)     ; load X[i+2]
     MULTD  F24,F22,F0      ; multiply a*X[i+2]
     LD     F26, 16(R2)     ; load Y[i+2]
     ADDD   F26,F24,F26     ; add aX[i+1] + Y[i+1]
     SD     16(R2),F26      ; store Y[i+1]
     ADDI   R1,R1,24        ; increment X index for all three iterations
     ADDI   R2,R2,24        ; increment Y index for all three iterations
     SGTI   R3,R1,done      ; test if finished (R3 := (R1>done))
     BEQZ   R3,foo          ; loop if not finished.
```

It is also important to use different working registers in each successive instance of the loop body, as otherwise unnecessary hazards are introduced. For simplicity I have used a systematic renumbering scheme.

On the static-schedule machine, how many clock cycles does each iteration of the loop take now, on average? How many clock cycles would each iteration take if you unrolled the loop completely?

Each instance of the non-unrolled loop body consists of 5 instructions and introduces the following stalls because of RAW hazards: MULTD is delayed by one cycle (due to F2), ADDD is delayed by three cycles (due to F4) and SD is delayed by one cycle (due to F6). Therefore, each instance of the original loop body costs 10 cycles.
At the end of the unrolled loop, we have four further instructions for loop management which don't cause any additional stalls. Hence, the unrolled loop takes $(3 \times 10 + 4) = 34$ instructions and each iteration of the original loop takes on average $34/3 = 11.33$ cycles.

If the loop is unrolled completely, we have no instructions for loop management overhead, so the completely unrolled loop will execute at a cost of 10 cycles per iteration.

Exercise 3.3. Compile-time pipeline scheduling

See if you can modify the sequence of instructions in the unrolled loop to reduce the number of stalls. How many clock cycles does each iteration take now?

The following schedule is one way improving the performance of the three times unrolled loop:

```
foo:  LD     F2, 0(R1)       ;
      ADDI   R1,R1,24        ; Done 'for free' in stall slot. Adjust addresses!
      MULTD  F4,F2,F0        ;
      LD     F6, 0(R2)       ;
      ADDD   F6,F4,F6        ;
      LD     F12, -16(R1)    ; Address adjusted. Swapping LD and SD gets rid
      SD     0(R2),F6        ; two stalls, assuming memory takes one cycle
      MULTD  F14,F12,F0      ;
      LD     F16, 8(R2)      ;
      ADDD   F16,F14,F16     ;
      LD     F22, -8(R1)     ; Address adjusted, LD and SD swapped
      SD     8(R2),F16       ;
      MULTD  F24,F22,F0      ;
      LD     F26, 16(R2)     ;
      ADDD   F26,F24,F26     ;
      ADDI   R2,R2,24        ; Done 'for free' in stall slot of ADDD
      SD     -8(R2),F26      ; Address adjusted
      SGTI   R3,R1,done      ;
      BEQZ   R3,foo          ;
```

This takes an average of 8.67 cycles per iteration. There are still three stall cycles on ADDD due to F4. By moving things around further, and with more unrolling and enough registers, it should be possible to eliminate all unnecessary stalls from the loop. The remaining stalls are then structural, and could be avoided by adding more functional units.

(Paul Kelly, Imperial College, October 2001)