Solutions to exercises on instruction level parallelism exploitation

J. Daniel García Sánchez (coordinator)
David Expósito Singh
Javier García Blas

Computer Architecture
ARCOS Group
Computer Science and Engineering Department
University Carlos III of Madrid

1 Exam exercises

Exercise 1

In the following code, each instruction has as an associated cost of one cycle, besides those included in table 1. Additionally, assume that the machine is able to issue one instruction per cycle, except waitngs due to stalls, and that the processor is pipelined with a single data path.

Due to structural hazards, stalls happen always independently that there are (or not) data dependencies with the next instructions. Initially, the following values are stored in registers **R1=0**, **R2=24**, **R3=16**.

```
Loop1: LD F2, 0(R1)
       ADD F2,F0,F2
Loop2: LD F4,0(R3)
       MULTD F4,F0,F4
       DIVD F10,F4,F0
       ADDD F12,F10,F4
       ADDI R1,R1,#8
       SUB R18,R2,R1
       BNZ R18,Loop2
       SD F2,0(R3)
       ADDI R3, R3, #8
       SUB R20,R2,R3
       BNZ R20,Loop1
```

Table 1: Additional costs per instruction

Instruction	Additional cost
LD	3
SD	1
ADD	2
MULTD	4
DIVD	10
ADDI	0
SUB	0
BNZ	1

1. Compute the number of needed cycles to run one iteration of the external loop and three of the internal loop.

2. Unroll three iterations from the internal loop, do not unroll the external loop, and perform again the computation from the previous section.

3. Compare the timing results (number of cycles) from the first and second sections. Justify your answers.

Solution 1

Section 1 Analysis:

```
Loop1: LD F2, 0(R1)          1+3
ADD F2,F0,F2                 1+2

Loop2: LD F4,0(R3)           1+3
MULTD F4,F0,F4               1+4
DIVD F10,F4,F0               1+10
ADD F12,F10,F4               1+2
ADDI R1,R1,#8                1
SUB R18,R2,R1                1
BNZ R18,Loop2                1+1

Loop2:                       27

SD F2,0(R3)                  1+1
ADDI R3,R3,#8                1
SUB R20,R2,R3                1
BNZ R20,Loop1                1+1

Loop1:                       13
```

Thee inner-loop iterations and one outer-loop iteration: $= 13 + (27 \cdot 3) = 94$ cycles

Section 2 Analysis:

```
Loop1: LD F2, 0(R1)          1+3
ADD F2,F0,F2                 1+2
Loop2: LD F4,0(R3)           1+3
LD F6,0(R3)                  1+3
LD F8,0(R3)                  1+3
MULTD F4,F0,F4               1+4
MULTD F6,F0,F6               1+4
MULTD F8,F0,F8               1+4
DIVD F10,F4,F0               1+10
DIVD F12,F6,F0               1+10
DIVD F14,F8,F0               1+10
ADDD F16,F10,F4              1+2
ADDD F18,F12,F6              1+2
ADDD F20,F14,F8              1+2
ADDI R1,R1,#24               1
SUB R18,R2,R1                1
BNZ R18,Loop2                1+1

3 * Loop2:                   73

SD F2,0(R3)                  1+1
ADDI R3,R3,#8                1
SUB R20,R2,R3                1
BNZ R20,Loop1                1+1
```

Thee inner-loop iterations and one outer-loop iteration $= 13 + (73 \cdot 1) = 86$ cycles

Section 3 It can be seen that there is no significant reduction of cycles. That's because the inner loop has very few iterations and that the weight of the control of the loop (increase indices and jump) is relatively small compared to the rest of instructions.

Exercise 2

A given processor has latencies between instructions as given by Table 2.

Table 2: Latencies between instructions

Instruction producing the result	Instruction using the result	Latency
FP ALU operation	Other FP ALU operation	6
FP ALU operation	Store double	3
Load double	FP ALU operation	2
Load double	Store double	0

In that machine, the following fragment of code will be executed:

```
LOOP:   L.D F0, 0(R1)
        L.D F2, 0(R2)
        ADD.D F4, F0, F2
        S.D F4, 0(R3)
        DADDUI R1, R1, #-8
        DADDUI R2, R2, #-8
        DADDUI R3, R3, #-8
        BNE R1, R4, LOOP
```

Initially registers have the following values:

- **R1**: Address of last element in first source array.
- **R2**: Address of last element in second source array.
- **R3**: Address of last element in target array.
- **R4**: Precomputed with **8(R4)** being first element in first source array.

All arrays have a size of 4,000 elements.
Complete the following tasks:

1. Determine how many cycles are required to execute all loop iterations without modifications.
2. Determine how many cycles are required to execute all loop iterations if loop scheduling is performed.
3. Determine how many cycles are required to execute all loop iterations if loop unrolling is performed for every two iterations.
4. Determine how many cycles are required to execute all loop iterations if loop unrolling is performed for every four iterations.

Solution 2

Section 1 The execution of an iteration of the loop would be:

```
L.D F0, 0(R1)
L.D F2, 0(R2)
<stall> x 2
ADD.D F4, F0, F2
<stall> x 3
S.D F4, 0(R3)
DADDUI R1, R1, #-8
DADDUI R2, R2, #-8
DADDUI R3, R3, #-8
BNE R1, R4, LOOP
```

In total, each iteration requires 13 cycles to start all the instructions. This gives a total of 52,000 cycles.

Section 2 DADDUI instruction can be executed before

```
L.D F0, 0(R1)
L.D F2, 0(R2)
DADDUI R1, R1, #-8
DADDUI R2, R2, #-8
ADD.D F4, F0, F2
DADDUI R3, R3, #-8
<stall> x 2
S.D F4, 8(R3)
BNE R1, R4, LOOP
```

In total, each iteration now requires 10 cycles. This results in a total of 40,000 cycles.

```
L.D F0, 0(R1)
L.D F2, 0(R2)
L.D F6, -8(R1)
L.D F8, -8(R2)
DADDUI R1, R1, #-16
ADD.D F4, F0, F2
ADD.D F10, F6, F8
DADDUI R2, R2, #-16
DADDUI R3, R3, #-16
S.D F4, 16(R3)
S.D F10, 8(R3)
BNE R1, R4, LOOP
```

A total of 12 cycles per iteration are required. This results in a total of 24,000 cycles

```
L.D F0, 0(R1)
L.D F2, 0(R2)
L.D F6, -8(R1)
L.D F8, -8(R2)
L.D F12, -16(R1)
L.D F14, -16(R2)
L.D F18, -24(R1)
L.D F20, -24(R2)
DADDUI R1, R1, #-32
DADDUI R2, R2, #-32
DADDUI R3, R3, #-32
ADD.D F4, F0, F2
ADD.D F10, F6, F8
ADD.D F16, F10, F12
ADD.D F22, F18, F20
S.D F4, 32(R3)
S.D F10, 24(R3)
S.D F16, 16(R3)
S.D F22, 8(R3)
BNE R1, R4, LOOP
```

A total of 20 cycles per iteration are required. This results in a total of 20,000 cycles

Exercise 3

Let MIPS processor with a pipeline that has two separate register files (one for integer and the other one for floating point values). The integer register file has 32 registers. The floating-point register file has 16 double-precision registers (**$f0**, **$f2**, ..., **$f30**).

The processor is assumed to have enough fetch and decode bandwidth to execute one instruction per cycle without stalls (with the exception of stalls derived from data dependencies).

Table 3 shows the extra latencies for some types of instructions. Those latencies have to be considered only when there are data dependencies. When there are no dependencies, those extra latencies are not applicable.

Table 3: Additional latencies per instruction

Instruction	Additional latency	Operation
ldc1	+2	Loads a 64 bit value into a floating point register.
sdc1	+2	Stores a 64 bit value into main memory.
add.d	+4	Adds double precision floating point registers
mul.d	+6	Multiplies double precision floating point registers.
addi	+0	Adds a value and an integer register.
subi	+0	Subtracts a value from an integer register.
bnez	+1	Branches if a register value is zero.

Instruction **bnez** uses delayed branching with one *delay slot*.

On that processor, the following code fragmet is executed:

```
loop:   ldc1 $f0, ($t0)
        ldc1 $f2, ($t1)
        add.d $f4, $f0, $f2
        mul.d $f4, $f4, $f6
        sdc1 $f4, ($t2)
        addi $t0, $t0, 8
        addi $t1, $t1, 8
        subi $t3, $t3, 1
        bnez $t3, loop
        addi $t2, $t2, 8
```

The initial values for registers are:

- **$t0**: **0x00100000**.
- **$t1**: **0x00140000**.
- **$t2**: **0x00180000**.
- **$t3**: **0x00000100**.

Complete the following tasks:

1. Enumerate the RAW dependencies in the previous code.

2. Show all the stalls that are produced when one single loop iteration is executed. Show the overall number of cycles per iteration.

3. Schedule the instructions in order to reduce the number of stalls.

4. Unroll the loop in the following way: each unrolled iteration processes four array positions. Obtain the resulting speedup. Note: use real register names (**$f0**, **$f2**, ..., **$f30**).

IMPORTANT: The solutions that do not use real existing registers (e.g.: **$f2'** o **$f2"**) will not be considered valid.

Solution 3

Dependencies If the instructions are numbered sequentially starting at **I1** (Up to **I10**), the following RAW dependencies can be identified:

1. **$f0**: **I1** → **I3**.
2. **$f2**: **I2** → **I3**.
3. **$f4**: **I3** → **I4**.
4. **$f4**: **I4** → **I5**.
5. **$t3**: **I8** → **I9**.

Stalls The following stalls occur when executing the code:

```
ldc1 $f0, ($t0)          #I1
ldc1 $f2, ($t1)          #I2
<stall> x 2
add.d $f4, $f0, $f2      #I3
<stall> x 4
mul.d $f4, $f4, $f6      #I4
<stall> x 6
sdc1 $f4, ($t2)          #I5
addi $t0, $t0, 8         #I6
addi $t1, $t1, 8         #I7
subi $t3, $t3, 1         #I8
bnez $t3, loop  #I9
addi $t2, $t2, 8         #I10
```

In total, 22 cycles are required.

Loop scheduling Reordering instructions can reduce the number of stalls:

```
ldc1 $f0, ($t0)          #I1
ldc1 $f2, ($t1)          #I2
addi $t0, $t0, 8         #I6
addi $t1, $t1, 8         #I7
add.d $f4, $f0, $f2      #I3
subi $t3, $t3, 1         #I8
<stall> x 3
mul.d $f4, $f4, $f6      #I4
<stall> x 6
sdc1 $f4, ($t2)          #I5
bnez $t3, loop  #I9
addi $t2, $t2, 8         #I10
```

A total of 19 cycles are required.

Unrolled loop Unrolling the loop with a factor of four obtains:

```
ldc1 $f0, ($t0)
ldc1 $f2, ($t1)
ldc1 $f8, 8($t0)
ldc1 $f10, 8($t1)
ldc1 $f14, 16($t0)
ldc1 $f16, 16($t1)
ldc1 $f20, 24($t0)
ldc1 $f22, 24($t1)
add.d $f4, $f0, $f2
add.d $f12, $f8, $f10
add.d $f18, $f14, $f16
add.d $f24, $f20,$f22
subi $t3, 4
mul.d $f4, $f4, $f6
mul.d $f12, $f12, $f6
mul.d $f18, $f18, $f6
mul.d $f24, $f24, $f6
addi $t0, 32
addi $t1, 32
<stall>
sdc1 $f4, ($t2)
sdc1 $f12, 8($t2)
sdc1 $f18, 16($t2)
sdc1 $f24, 24($t2)
bnez $t3, loop
addi $t2, 32
```

In total, 26 cycles are required every 4 iterations. That is 6.5 cycles per iteration

Exercise 4

A given processor is intended to run the following code fragment:

```
i0: lw $r4, 0($r1)
i1: lw $r5, 0($r2)
i2: add $r4, $r4, $r5
i3: sw $r4, 0($r3)
i4: addi $r1, $r1, 4
i5: addi $r2, $r2, 4
i6: addi $r3, $r3, 4
i7: bne $r3, $r0, i0
```

Assume that the processor has a pipelined architecture with 5 stages (fetch, decode, execute, memory and write-back) without forwarding. All operations are executed in one cycle per stage, except:

- Load and store instructions, that require two cycles for the memory stage (an additional cycle).
- Branch instructions require an additional cycle in the execution stage. Assume that those instructions do not have any branch prediction.

Answer the following questions:

1. Determine the RAW data hazards in the code that have impact in code execution.
2. Show a timing diagram with the stages for each instruction in one iteration.
3. Determine how many cycles are required to execute one loop iteration if there is no branch prediction.
4. Propose a loop unrolling assuming that the loop runs for 1000 iterations. Unroll with a factor of four iterations.
5. Determine the obtained speedup obtained with unrolling proposed in the previous section.

Solution 4

Section 1

- **$r4**: **i0** → **i2**
- **$r5**: **i1** → **i2**
- **$r4**: **i2** → **i3**
- **$r3**: **i6** → **i7**

Section 2 Table 4 shows the corresponding timing diagram.

- I0: Requires two memory cycles.
- I1: It can not start memory stage until I0 does not end memory. Requires two memory cycles.
- I2: It can not start decoding until I1 does not do WB..
- I3: No puede empezar captación hasta que se libera unidad por I2.

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I0: lw $r4, 0($r1)	IF	ID	EX	M	M	WB								
I1: lw $r5, 0($r2)		IF	ID	EX	–	M	M	WB						
I2: add $r4, $r4, $r5			IF	ID	–	–	–	–	EX	M	WB			
I3: sw $r4, 0($r3)				IF	–	–	–	–	ID	–	–	EX	M	M
I4: addi $r1, $r1, 4									IF	–	–	ID	EX	–
I5: addi $r2, $r2, 4												IF	ID	–
I6: addi $r3, $r3, 4													IF	–
I7: bne $r3, $r0, I0														

Instr.	15	16	17	18	19	20	21	22
I0: lw $r4, 0($r1)								
I1: lw $r5, 0($r2)								
I2: add $r4, $r4, $r5								
I3: sw $r4, 0($r3)	WB							
I4: addi $r1, $r1, 4	M	WB						
I5: addi $r2, $r2, 4	EX	M	WB					
I6: addi $r3, $r3, 4	ID	EX	M	WB				
I7: bne $r3, $r0, I0	IF	ID	–	–	EX	EX	M	WB

Table 4: First timing diagram of the exercise 4.

- I4: It can not start pickup until unit is released by I2.
- I5: It can not start execution stage until I4 releases drive execution.
- I6: It can not start decode stage until I5 releases unit Of decoding.
- I7: It can not start decoding until I6 does WB.

Section 3 If the decoding stage is assumed to include a comparator ont the register file, the next instruction to I7 can begin after the decoding stage of I7 (that is, in cycle 19) is completed. Each iteration requires 18 clock cycles.

If the decoding step is not assumed to include a comparator, the comparison of two registers should be made with the general ALU. Therefore the decision cannot be taken until execution stage of I7 (in cycle 21) has been completed. In this case each iteration requires 20 clock cycles.

Section 4 A possible solution is presented below. Note, however, that more aggressive solutions are possible leading to better *speedups*.

```
I0: lw $r4, 0($r1)
i1: lw $r5, 0($r2)
i2: lw $r6, 4($r1)
i3: lw $r7, 4($r2)
i4: lw $r8, 8($r1)
i5: lw $r9, 8($r2)
i6: lw $r10, 12($r1)
i7: lw $r11, 12($r2)
i8: add $r4, $r4, $r5
i9: add $r6, $r6, $r7
i10: add $r8, $r8, $r9
i11: add $r10, $r10, $r11
```

```
i12: sw $r4, 0($r3)
i13: sw $r6, 4($r3)
i14: sw $r8, 8($r3)
i15: sw $r10, 12($r3)
i16: addi $r3, $r3, 16
i17: addi $r2, $r2, 16
i18: addi $r1, $r1, 16
i19: bne $r3, $r0, i0
```

Section 5 Table 5 shows the corresponding timing diagram.

Depending on the criterion chosen in Section 2, the number of cycles per iteration will be 33 or 35 and the number of cycles per iteration will be $\frac{33}{4} = 8.25$ or $\frac{35}{4} = 8.75$

Therefore the speedup will be either:

$$S = \frac{18}{8.25} = 2.18$$

or:

$$S = \frac{20}{8.75} = 2.28$$

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I0	IF	ID	EX	M	M	WB									
I1		IF	ID	EX	–	M	M	WB							
I2			IF	ID	–	EX	–	M	M	WB					
I3				IF	–	ID	–	EX	–	M	M	WB			
I4						IF	–	ID	–	EX	–	M	M	WB	
I5								IF	–	ID	–	EX	–	M	M
I6										IF	–	ID	–	EX	–
I7												IF	–	ID	–
I8														IF	–
I9															
I10															
I11															
I12															
I13															

Instrucción	16	17	18	19	20	21	22
I0							
I1							
I2							
I3							
I4							
I5	WB						
I6	M	M	WB				
I7	EX	–	M	M	WB		
I8	ID	–	EX	–	M	WB	
I9	IF	–	ID	–	EX	M	WB
I10			IF	–	ID	M	WB
I11					IF	ID	EX
I12						F	D
I13							F

Instrucción	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
I10	WB														
I11	M	WB													
I12	EX	M	M	WB											
I13	ID	EX	–	M	M	WB									
I14	IF	ID	–	EX	–	M	M	WB							
I15		IF	–	ID	–	EX	–	M	M	WB					
I16				IF	–	ID	–	EX	–	M	WB				
I17						IF	–	ID	–	EX	M	WB			
I18								IF	–	ID	EX	M	WB		
I19										IF	ID	EX	EX	M	WB

Table 5: Second timing diagram of the exercise 4.

Exercise 5 *May 2012 exam.*

Given the following code section:

```
        DADDUI R3, R1, #40      ; I1
LOOP:   L.D F0, 0(R1)           ; I2
        L.D F2, 0(R2)           ; I3
        ADD.D F4, F0, F2        ; I4
        S.D F4, 0(R1)           ; I5
        DADDUI R1, R1, #8       ; I6
        DADDUI R2, R2, #8       ; I7
        BLE R1, R3, LOOP        ; I8
```

And considering that it runs in a machine with additional latencies between instructions expressed in Table 6.

Table 6: Additional latencies

Instruction producing the result (previous)	Instruction using the result (subsequent)	Latency
FP ALU operation	FP ALU operation	5
FP ALU operation	Load/store double	4
FP ALU operation	Branch instruction	4
Load double	FP ALU operation	2
Load double	Load double	1
Store double	FP ALU operation	2

The *branch* instruction has a latency of one cycle and no *delay slot*.

Besides, assume that the machine is able to issue one instruction per cycle, except waiting due to stalls and that the processor has a pipeline with a single data path.

1. Identify all the data dependencies.
2. Determine the total number of cycles needed to run the complete section of code.
3. Modify the code to reduce stalls through the loop scheduling technique. Determine the obtained speedup versus to the non-scheduled version.
4. Modify the code performing a loop unrolling with two loop iterations. Determine the obtained speedup versus the non-scheduled version.

Solution 5

Section 1 The following dependencies are produced:

- **I4** → **I2** (**F0**: RAW), **I4** → **I3** (**F2**: RAW)
- **I5** → **I4** (**F4**: RAW)
- **I6** → **I1** (**R1**: WAR), **I6** → **I2** (**R1**: WAR), **I6** → **I5** (**R1**: WAR)
- **I7** → **I3** (**R2**: WAR)
- **I8** → **I7** (**R2**: RAW), **I8** → **I1** (**R3**: RAW)

Section 2 The following are the stalls in the execution:

```
        DADDUI R3, R1, #40
LOOP:   L.D F0, 0(R1)
        L.D F2, 0(R2)
        <stall>
        <stall>
        ADD.D F4, F0, F2
        <stall>
        <stall>
        <stall>
        <stall>
        S.D F4, 0(R1)
        DADDUI R1, R1, #8
        DADDUI R2, R2, #8
        BLE R2, R3, LOOP
```

A cycle is required for the initiation code. Each iteration needs 13 cycles. As the loop is executed 5 times, we have a total of:

$$1 + 5 \cdot 13 = 66 \text{ cycles}$$

Because there is no delay slot it is not necessary to add a stall cycle after the branch instruction (**BLE**).

Section 3 Next, the modified code:

```
        DADDUI R3, R1, #40
LOOP:   L.D F0, 0(R1)
        L.D F2, 0(R2)
        DADDUI R1, R1, #8
        DADDUI R2, R2, #8
        ADD.D F4, F0, F2
        <stall>
        <stall>
        <stall>
        <stall>
        S.D F4, -8(R1)
        BLE R1, R3, LOOP
```

A total time of $1 + 5 \cdot 11 = 56$. Again, we need a cycle for the initiation code. Each iteration needs 13 cycles. Because there is no delay slot, it is not necessary to add a stall after the jump instruction (**BLE**).

$$\text{Speedup} = 66/56 = 1.17$$

Section 4 Next, the modified code:

```
        DADDUI R3, R1, #32
LOOP:   L.D F0, 0(R1)
        L.D F2, 0(R2)
        <stall>
        <stall>
        ADD.D F4, F0, F2
        <stall>
        <stall>
        <stall>
        <stall>
        S.D F4, 0(R1)
        L.D F6, 8(R1)
        L.D F8, 8(R2)
        <stall>
        <stall>
        ADD.D F10, F6, F8
```

```
        <stall>
        <stall>
        <stall>
        <stall>
        S.D F10, 8(R1)
        DADDUI R1, R1, #16
        DADDUI R2, R2, #16
        BLE R2, R3, LOOP
        L.D F0, 0(R1)
        L.D F2, 0(R2)
        <stall>
        <stall>
        ADD.D F4, F0, F2
        <stall>
        <stall>
        <stall>
        <stall>
        S.D F4, 0(R1)
```

In total, we now have 2 iterations inside the loop. In addition, the fifth iteration of the original loop is now done at the end. The time required is:

$$1 + 2 \cdot 23 + 10 = 57$$

If you also re-schedule the instruction, you can have:

```
        DADDUI R3, R1, #32
LOOP:   L.D F0, 0(R1)
        L.D F2, 0(R2)
        L.D F6, 8(R1)
        L.D F8, 8(R2)
        ADD.D F4, F0, F2
        Detención
        ADD.D F10, F6, F8
        DADDUI R1, R1, #16
        DADDUI R2, R2, #16
        S.D F4, -16(R1)
        Detención
        S.D F10, -8(R1)
        BLE R2, R3, LOOP
        L.D F0, 0(R1)
        L.D F2, 0(R2)
        Detención
        Detención
        ADD.D F4, F0, F2
        Detención
        Detención
        Detención
        Detención
        S.D F4, 0(R1)
```

The new time is:

$$1 + 2 \cdot 13 + 10 = 37 \text{ cycles}$$