Lecture: Static ILP

- Topics: loop scheduling, loop unrolling, software pipelines
Smart Schedule

- By re-ordering instructions, it takes 6 cycles per iteration instead of 10
- We were able to violate an anti-dependence easily because an immediate was involved
- Loop overhead (instrs that do book-keeping for the loop): 2
  Actual work (the ld, add.d, and s.d): 3 instrs
  Can we somehow get execution time to be 3 cycles per iteration?
Problem 1

```
for (i=1000; i>0; i--)
    x[i] = y[i] * s;
```

**Source code**

```
Loop:    L.D         F0, 0(R1)          ; F0 = array element
         MUL.D    F4, F0, F2        ; multiply scalar
         S.D         F4, 0(R2)          ; store result
         DADDUI  R1, R1,# -8      ; decrement address pointer
         DADDUI  R2, R2,#-8       ; decrement address pointer
         BNE        R1, R3, Loop    ; branch if R1 != R3
         NOP
```

**Assembly code**

- LD -> any : 1 stall
- FPMUL -> any: 5 stalls
- FPMUL -> ST : 4 stalls
- IntALU -> BR : 1 stall

• How many cycles do the default and optimized schedules take?
Problem 1

\[
\text{for (i=1000; i>0; i--)}
\text{x[i] = y[i] * s;}
\]

### Source code

#### Loop:
- \text{L.D} \quad F0, 0(R1) \quad ; F0 = array element
- \text{MUL.D} \quad F4, F0, F2 \quad ; multiply scalar
- \text{S.D} \quad F4, 0(R2) \quad ; store result
- \text{DADDUI} \quad R1, R1,# -8 \quad ; decrement address pointer
- \text{DADDUI} \quad R2, R2,#-8 \quad ; decrement address pointer
- \text{BNE} \quad R1, R3, Loop \quad ; branch if R1 != R3
- \text{NOP}

### Assembly code

- LD -> any : 1 stall
- FPMUL -> any: 5 stalls
- FPMUL -> ST : 4 stalls
- IntALU -> BR : 1 stall

- How many cycles do the default and optimized schedules take?

**Unoptimized:** \text{LD 1s} \quad \text{MUL 4s} \quad \text{SD} \quad \text{DA} \quad \text{DA} \quad \text{BNE 1s} \quad -- 12 cycles

**Optimized:** \text{LD} \quad \text{DA} \quad \text{MUL} \quad \text{DA} \quad 2s \quad \text{BNE} \quad \text{SD} \quad -- 8 cycles
Loop Unrolling

- Loop overhead: 2 instrs; Work: 12 instrs
- How long will the above schedule take to complete?
Scheduled and Unrolled Loop

Loop:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Source, Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0, 0(R1)</td>
</tr>
<tr>
<td>L.D</td>
<td>F6, -8(R1)</td>
</tr>
<tr>
<td>L.D</td>
<td>F10, -16(R1)</td>
</tr>
<tr>
<td>L.D</td>
<td>F14, -24(R1)</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F4, F0, F2</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F8, F6, F2</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F12, F10, F2</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F16, F14, F2</td>
</tr>
<tr>
<td>S.D</td>
<td>F4, 0(R1)</td>
</tr>
<tr>
<td>S.D</td>
<td>F8, -8(R1)</td>
</tr>
<tr>
<td>DADDUI</td>
<td>R1, R1, # -32</td>
</tr>
<tr>
<td>S.D</td>
<td>F12, 16(R1)</td>
</tr>
<tr>
<td>BNE</td>
<td>R1, R2, Loop</td>
</tr>
<tr>
<td>S.D</td>
<td>F16, 8(R1)</td>
</tr>
</tbody>
</table>

- Execution time: 14 cycles or 3.5 cycles per original iteration

LD -> any : 1 stall
FPALU -> any: 3 stalls
FPALU -> ST : 2 stalls
IntALU -> BR : 1 stall
Loop Unrolling

- Increases program size

- Requires more registers

- To unroll an n-iteration loop by degree k, we will need \((n/k)\) iterations of the larger loop, followed by \((n \mod k)\) iterations of the original loop
Automating Loop Unrolling

• Determine the dependences across iterations: in the example, we knew that loads and stores in different iterations did not conflict and could be re-ordered

• Determine if unrolling will help – possible only if iterations are independent

• Determine address offsets for different loads/stores

• Dependency analysis to schedule code without introducing hazards; eliminate name dependences by using additional registers
Problem 2

for (i=1000; i>0; i--)
    x[i] = y[i] * s;

Source code

Assembly code

```
Loop:   L.D   F0, 0(R1)  ; F0 = array element
        MUL.D  F4, F0, F2  ; multiply scalar
        S.D    F4, 0(R2)  ; store result
        DADDUI R1, R1,#-8 ; decrement address pointer
        DADDUI R2, R2,#-8 ; decrement address pointer
        BNE    R1, R3, Loop ; branch if R1 != R3
        NOP
```

- How many unrolls does it take to avoid stall cycles?

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall
Problem 2

for (i=1000; i>0; i--)
    x[i] = y[i] * s;

Loop: L.D F0, 0(R1) ; F0 = array element
      MUL.D F4, F0, F2 ; multiply scalar
      S.D F4, 0(R2) ; store result
      DADDUI R1, R1,# -8 ; decrement address pointer
      DADDUI R2, R2,#-8 ; decrement address pointer
      BNE R1, R3, Loop ; branch if R1 != R3
      NOP

• How many unrolls does it take to avoid stall cycles?

  Degree 2: LD LD MUL MUL DA DA 1s SD BNE SD
  Degree 3: LD LD LD MUL MUL MUL DA DA SD SD BNE SD
          – 12 cyc/3 iterations

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall
### Superscalar Pipelines

<table>
<thead>
<tr>
<th>Integer pipeline</th>
<th>FP pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handles L.D, S.D, ADDUI, BNE</td>
<td>Handles ADD.D</td>
</tr>
</tbody>
</table>

- What is the schedule with an unroll degree of 5?
Superscalar Pipelines

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Integer pipeline</th>
<th>FP pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L.D F0,0(R1)</td>
<td>ADD.D F4,F0,F2</td>
</tr>
<tr>
<td></td>
<td>L.D F6,-8(R1)</td>
<td>ADD.D F8,F6,F2</td>
</tr>
<tr>
<td></td>
<td>L.D F10,-16(R1)</td>
<td>ADD.D F12,F10,F2</td>
</tr>
<tr>
<td></td>
<td>L.D F14,-24(R1)</td>
<td>ADD.D F16,F14,F2</td>
</tr>
<tr>
<td></td>
<td>L.D F18,-32(R1)</td>
<td>ADD.D F20,F18,F2</td>
</tr>
<tr>
<td></td>
<td>S.D F4,0(R1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S.D F8,-8(R1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S.D F12,-16(R1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DADDUI R1,R1,# -40</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S.D F16,16(R1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BNE R1,R2,Loop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S.D F20,8(R1)</td>
<td></td>
</tr>
</tbody>
</table>

- Need unroll by degree 5 to eliminate stalls (fewer if we move DADDUI up)
- The compiler may specify instructions that can be issued as one packet
- The compiler may specify a fixed number of instructions in each packet: Very Large Instruction Word (VLIW)
Problem 3

Source code

```
for (i=1000; i>0; i--)
    x[i] = y[i] * s;
```

Assembly code

```
Loop:
    L.D      F0, 0(R1)  ; F0 = array element
    MUL.D    F4, F0, F2  ; multiply scalar
    S.D      F4, 0(R2)  ; store result
    DADDUI   R1, R1,#-8  ; decrement address pointer
    DADDUI   R2, R2,#-8  ; decrement address pointer
    BNE      R1, R3, Loop  ; branch if R1 != R3
    NOP
```

• How many unrolls does it take to avoid stalls in the superscalar pipeline?

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall
Problem 3

for (i=1000; i>0; i--)
    x[i] = y[i] * s;

Source code

Loop: L.D F0, 0(R1) ; F0 = array element
      MUL.D F4, F0, F2 ; multiply scalar
      S.D F4, 0(R2) ; store result
      DADDUI R1, R1,# -8 ; decrement address pointer
      DADDUI R2, R2,#-8 ; decrement address pointer
      BNE R1, R3, Loop ; branch if R1 != R3
      NOP

Assembly code

• How many unrolls does it take to avoid stalls in the superscalar pipeline?

LD
LD
LD MUL
LD MUL
LD MUL
LD MUL 7 unrolls. Could also make do with 5 if we moved up the DADDUIs.
LD MUL
LD MUL
SD MUL
Loop:  
- L.D  F0, 0(R1)  
- ADD.D  F4, F0, F2  
- S.D  F4, 0(R1)  
- DADDUI  R1, R1,# -8  
- BNE  R1, R2, Loop

Software Pipeline?!
Software Pipeline

Original iter 1
Original iter 2
Original iter 3
Original iter 4

New iter 1
New iter 2
New iter 3
New iter 4
Software Pipelining

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

Loop:    S.D     F4, 16(R1)
         ADD.D   F4, F0, F2
         L.D     F0, 0(R1)
         DADDUI  R1, R1,# -8
         BNE     R1, R2, Loop

• Advantages: achieves nearly the same effect as loop unrolling, but without the code expansion – an unrolled loop may have inefficiencies at the start and end of each iteration, while a sw-pipelined loop is almost always in steady state – a sw-pipelined loop can also be unrolled to reduce loop overhead

• Disadvantages: does not reduce loop overhead, may require more registers
Recall Superscalar Pipeline Example

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Integer pipeline</th>
<th>FP pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0,0(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F6,-8(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F10,-16(R1)</td>
<td>ADD.D F4,F0,F2</td>
</tr>
<tr>
<td>L.D</td>
<td>F14,-24(R1)</td>
<td>ADD.D F8,F6,F2</td>
</tr>
<tr>
<td>L.D</td>
<td>F18,-32(R1)</td>
<td>ADD.D F12,F10,F2</td>
</tr>
<tr>
<td>S.D</td>
<td>F4,0(R1)</td>
<td>ADD.D F16,F14,F2</td>
</tr>
<tr>
<td>S.D</td>
<td>F8,-8(R1)</td>
<td>ADD.D F20,F18,F2</td>
</tr>
<tr>
<td>S.D</td>
<td>F12,-16(R1)</td>
<td></td>
</tr>
<tr>
<td>DADDUI</td>
<td>R1,R1,# -40</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F16,16(R1)</td>
<td></td>
</tr>
<tr>
<td>BNE</td>
<td>R1,R2,Loop</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F20,8(R1)</td>
<td></td>
</tr>
</tbody>
</table>

- Need unroll by degree 5 to eliminate stalls (fewer if we move DADDUI up)
- The compiler may specify instructions that can be issued as one packet
- The compiler may specify a fixed number of instructions in each packet: Very Large Instruction Word (VLIW)
### Problem 4

Given the source code:

```c
for (i=1000; i>0; i--)
    x[i] = y[i] * s;
```

**Source code**

```
Loop:    L.D  F0, 0(R1)       ; F0 = array element
         MUL.D F4, F0, F2        ; multiply scalar
         S.D  F4, 0(R2)         ; store result
         DADDUI R1, R1,# -8     ; decrement address pointer
         DADDUI R2, R2,#-8      ; decrement address pointer
         BNE    R1, R3, Loop     ; branch if R1 != R3
         NOP
```

**Assembly code**

- `LD -> any : 1 stall`
- `FPMUL -> any: 5 stalls`
- `FPMUL -> ST : 4 stalls`
- `IntALU -> BR : 1 stall`

**Questions:**

- Show the SW pipelined version of the code and does it cause stalls?
Problem 4

for (i=1000; i>0; i--)
    x[i] = y[i] * s;

Loop:     L.D         F0, 0(R1)          ; F0 = array element
          MUL.D    F4, F0, F2        ; multiply scalar
          S.D      F4, 0(R2)          ; store result
          DADDUI  R1, R1,# -8      ; decrement address pointer
          DADDUI  R2, R2,#-8       ; decrement address pointer
          BNE        R1, R3, Loop    ; branch if R1 != R3
          NOP

• Show the SW pipelined version of the code and does it cause stalls?

Loop:    S.D      F4, 0(R2)
          MUL   F4, F0, F2
          L.D   F0, 0(R1)
          DADDUI R2, R2, #-8
          BNE   R1, R3, Loop
          DADDUI R1, R1, # -8

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

There will be no stalls