COSC 6385
Computer Architecture
- Instruction Level Parallelism
with Software Approaches (II)

Edgar Gabriel
Fall 2006
Finding dependencies

• Complex task, even when ignoring indirect access etc.
• Imagine the following case:
  – Two loop indices, \( j \) and \( k \)
  – Two instructions of the format:
    
    \[
    x[a*j +b] = ... \\
    ... = x[c*k+d]
    \]

  – Do the two accesses to \( x \) create a dependence?
• No general method to determine dependence at compile time
• A sufficient test for the absence of a dependence is the greatest common divisor (GDC) test
Finding dependencies (II)

- Greatest common divisor (GDC) test: a loop carried dependency does exist if the $GCD(c, a)$ divides $(d-b)$
  - Please note, that the test can succeed and still no dependence exists

- Example:
  ```c
  for ( i=1; i <=100 ; i++ ) {
  }
  a=2, b=3, c=2, d=0
  GCD (a,c) = 2 and d-b = -3
  
  Since 2 does not divide -3 no dependence is possible
  ```
Finding dependencies (III)

- Array oriented dependence analysis fails if
  - Objects are referenced via pointers instead of indices
  - Indirect array indices (e.g. \( a[b[i]] \))
  - Dependencies existing for certain value of input variables
Eliminating dependencies

- Eliminating dependent computations
  - *Copy propagation*
    
    \[
    \begin{align*}
    \text{DADDUI R1, R2, #4} & \implies \text{DADDUI R1, R2, #8} \\
    \text{DADDUI R1, R1, #4} & \implies \text{DADDUI R1, R2, #8}
    \end{align*}
    \]

- *Tree height reduction*
  
  \[
  \begin{align*}
  \text{ADD R1, R2, R3} & \implies \text{ADD R1, R2, R3} \\
  \text{ADD R4, R1, R6} & \implies \text{ADD R4, R6, R7} \\
  \text{ADD R8, R4, R7} & \implies \text{ADD R8, R1, R4}
  \end{align*}
  \]

  \[
  \begin{align*}
  \text{sum}=\text{sum}+x /* \text{suppose this is in a loop and we unroll it 5 times */} & \implies \text{sum}=\text{sum}+x1+x2+x3+x4+x5 \\
  \text{sum}=(\text{sum}+x1)+(x2+x3)+(x4+x5) & \implies \text{sum}=(\text{sum}+x1)+(x2+x3)+(x4+x5)
  \end{align*}
  \]

  Can be done in parallel

  Must be done sequentially

---

Slide based on a lecture by Jelena Mirkovic, University of Delaware

http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Software Pipelining

- Combining instructions from different loop iterations to separate dependent instructions within an iteration

Slide based on a lecture by Jelena Mirkovic, University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Software Pipelining

- Apply software pipelining technique to the following loop:

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

Startup code

```assembly
R1+16
L.D F0, 0(R1)
ADD.D F4, F0, F2
S.D F4, 0(R1)
```

Cleanup code

```assembly
R1+8
ADD.D F4, F0, F2
```

```assembly
R1
L.D F0, 0(R1)
ADD.D F4, F0, F2
S.D F4, 0(R1)
```

Slide based on a lecture by Jelena Mirkovic, University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Software Pipelining vs. Loop Unrolling

- Loop unrolling eliminates loop maintenance overhead exposing parallelism between iterations
  - Creates larger code
- Software pipelining enables some loop iterations to run at top speed by eliminating RAW hazards that create latencies within iteration
  - Requires more complex transformations

Slide based on a lecture by Jelena Mirkovic, University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Static Multiple Issue: VLIW

- Hardware checking for dependencies in issue packets may be expensive and complex
  - Compiler can examine instructions and decide which ones can be scheduled in parallel – group instructions into instruction packets – VLIW
  - Hardware can then be simplified
  - Processor has multiple functional units and each field of the VLIW is assigned to one unit
    - For example, VLIW could contain 5 fields and one has to contain ALU instruction or branch, two have to contain FP instructions and two have to be memory references

Slide based on a lecture by Jelena Mirkovic, University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Static Multiple Issue: VLIW

- Hardware checking for dependencies in issue packets may be expensive and complex
  - Compiler can examine instructions and decide which ones can be scheduled in parallel – group instructions into instruction packets – VLIW
  - Hardware can then be simplified
  - Processor has multiple functional units and each field of the VLIW is assigned to one unit
    - For example, VLIW could contain 5 fields and one has to contain ALU instruction or branch, two have to contain FP instructions and two have to be memory references

Slide based on a lecture by Jelena Mirkovic, University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Example

- Assume VLIW contains 5 fields: ALU instruction or branch, two FP instructions and two memory references
- Ignore branch delay slot

Loop:  

L.D F0,0(R1)  
memory reference 
stall, wait for F0 value to propagate 
ADD.D F4, F0, F2  
FP instruction 
stall, wait for FP add to be completed  
stall, wait for FP add to be completed 
S.D F4, 0(R1)  
memory reference 
DADDUI R1, R1, #-8  
ALU instruction  
stall, wait for R1 value to propagate 
BNE R1, R2, Loop  
ALU instruction
Example

- Unroll seven times and rearrange

<table>
<thead>
<tr>
<th>Loop:</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F0,0(R1)</td>
<td>S.D F4, 0(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D F6,-8(R1)</td>
<td>S.D F8, -8(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D F10,-16(R1)</td>
<td>S.D F12, -16(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D F14,-24(R1)</td>
<td>S.D F16, -24(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D F18,-32(R1)</td>
<td>S.D F20, -32(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D F22,-40(R1)</td>
<td>DADDUI R1, R1, #-56</td>
<td></td>
</tr>
<tr>
<td>L.D F26,-48(R1)</td>
<td>S.D F24, 16(R1)</td>
<td></td>
</tr>
<tr>
<td>ADD.D F4, F0, F2</td>
<td>BNE R1, R2, Loop</td>
<td></td>
</tr>
<tr>
<td>ADD.D F8, F6, F2</td>
<td>S.D F28, 8(R1)</td>
<td></td>
</tr>
<tr>
<td>ADD.D F12, F10, F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F16, F14, F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F20, F18, F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F24, F22, F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F28, F26, F2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Slide based on a lecture by Jelena Mirkovic,
University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Example

Loop:

L.D F0,0(R1)
L.D F6,-8(R1)
L.D F10,-16(R1)
L.D F14,-24(R1)
L.D F18,-32(R1)
L.D F22,-40(R1)
L.D F26,-48(R1)
ADD.D F4, F0, F2
ADD.D F8, F6, F2
ADD.D F12, F10, F2
ADD.D F16, F14, F2
ADD.D F20, F18, F2
ADD.D F24, F22, F2
ADD.D F28, F26, F2

S.D F4, 0(R1)
S.D F8, -8(R1)
S.D F12, -16(R1)
S.D F16, -24(R1)
S.D F20, -32(R1)
DADDUI R1, R1, #-56
S.D F24, 16(R1)
S.D F28, 8(R1)

BNE R1, R2, Loop
Example

Loop:
L.D F0,0(R1)
L.D F6,-8(R1)
L.D F10,-16(R1)
L.D F14,-24(R1)
L.D F18,-32(R1)
L.D F22,-40(R1)
L.D F26,-48(R1)
ADD.D F4, F0, F2
ADD.D F8, F6, F2
ADD.D F12, F10, F2
ADD.D F16, F14, F2
ADD.D F20, F18, F2
ADD.D F24, F22, F2
ADD.D F28, F26, F2
S.D F4, 0(R1)
S.D F8, -8(R1)
S.D F12, -16(R1)
S.D F16, -24(R1)
S.D F20, -32(R1)
DADDUI R1, R1, #-56
S.D F24, 16(R1)
S.D F28, 8(R1)
BNE R1, R2, Loop

Slide based on a lecture by Jelena Mirkovic, University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Example

Loop:

L.D F0,0(R1)
L.D F6,-8(R1)
L.D F10,-16(R1)
L.D F14,-24(R1)
L.D F18,-32(R1)
L.D F22,-40(R1)
L.D F26,-48(R1)
ADD.D F4, F0, F2
ADD.D F8, F6, F2
ADD.D F12, F10, F2
ADD.D F16, F14, F2
ADD.D F20, F18, F2
ADD.D F24, F22, F2
ADD.D F28, F26, F2

S.D F4, 0(R1)
S.D F8, -8(R1)
S.D F12, -16(R1)
S.D F16, -24(R1)
S.D F20, -32(R1)
DADDUI R1, R1, #-56
S.D F24, 16(R1)
BNE R1, R2, Loop
S.D F28, 8(R1)

Slide based on a lecture by Jelena Mirkovic, University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Example

Loop:

L.D F0, 0(R1)
L.D F6, -8(R1)
L.D F10, -16(R1)
L.D F14, -24(R1)
L.D F18, -32(R1)
L.D F22, -40(R1)
L.D F26, -48(R1)
ADD.D F4, F0, F2
ADD.D F8, F6, F2
ADD.D F12, F10, F2
ADD.D F16, F14, F2
ADD.D F20, F18, F2
ADD.D F24, F22, F2
ADD.D F28, F26, F2

S.D F4, 0(R1)
S.D F8, -8(R1)
S.D F12, -16(R1)
S.D F16, -24(R1)
S.D F20, -32(R1)
S.D F24, 16(R1)
S.D F28, 8(R1)

DADDUI R1, R1, #-56
BNE R1, R2, Loop

Slide based on a lecture by Jelena Mirkovic, University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Example

Loop:

- L.D F0,0(R1)
- L.D F6,-8(R1)
- L.D F10,-16(R1)
- L.D F14,-24(R1)
- L.D F18,-32(R1)
- L.D F22,-40(R1)
- L.D F26,-48(R1)
- ADD.D F4, F0, F2
- ADD.D F8, F6, F2
- ADD.D F12, F10, F2
- ADD.D F16, F14, F2
- ADD.D F20, F18, F2
- ADD.D F24, F22, F2
- ADD.D F28, F26, F2

- S.D F4, 0(R1)
- S.D F8, -8(R1)
- S.D F12, -16(R1)
- S.D F16, -24(R1)
- S.D F20, -32(R1)
- DADDUI R1, R1, #-56
- S.D F24, 16(R1)
- BNE R1, R2, Loop
- S.D F28, 8(R1)

Slide based on a lecture by Jelena Mirkovic, University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Example

Loop:

L.D F0,0(R1)
L.D F6,-8(R1)
L.D F10,-16(R1)
L.D F14,-24(R1)
L.D F18,-32(R1)
L.D F22,-40(R1)
L.D F26,-48(R1)
ADD.D F4, F0, F2
ADD.D F8, F6, F2
ADD.D F12, F10, F2
ADD.D F16, F14, F2
ADD.D F20, F18, F2
ADD.D F24, F22, F2
ADD.D F28, F26, F2

S.D F4, 0(R1)
S.D F8, -8(R1)
S.D F12, -16(R1)
S.D F16, -24(R1)
S.D F20, 24(R1)
DADDUI R1, R1, #-56
S.D F24, 16(R1)
BNE R1, R2, Loop
S.D F28, 8(R1)

Slide based on a lecture by Jelena Mirkovic, University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Example

Loop:
L.D F0,0(R1)
L.D F6,-8(R1)
L.D F10,-16(R1)
L.D F14,-24(R1)
L.D F18,-32(R1)
L.D F22,-40(R1)
L.D F26,-48(R1)
ADD.D F4, F0, F2
ADD.D F8, F6, F2
ADD.D F12, F10, F2
ADD.D F16, F14, F2
ADD.D F20, F18, F2
ADD.D F24, F22, F2
ADD.D F28, F26, F2

S.D F4, 0(R1)
S.D F8, -8(R1)
S.D F12, -16(R1)
S.D F16, -24(R1)
S.D F20, 24(R1)
DADDUI R1, R1, #-56
S.D F24, 16(R1)
BNE R1, R2, Loop
S.D F28, 8(R1)
Example

Loop:  
L.D F0, 0(R1)  
L.D F6, -8(R1)  
L.D F10, -16(R1)  
L.D F14, -24(R1)  
L.D F18, -32(R1)  
L.D F22, -40(R1)  
L.D F26, -48(R1)  
ADD.D F4, F0, F2  
ADD.D F8, F6, F2  
ADD.D F12, F10, F2  
ADD.D F16, F14, F2  
ADD.D F20, F18, F2  
ADD.D F24, F22, F2  
ADD.D F28, F26, F2  
S.D F4, 0(R1)  
S.D F6, -8(R1)  
S.D F10, -16(R1)  
S.D F14, -24(R1)  
S.D F18, -32(R1)  
DADDUI R1, R1, #-56  
S.D F20, 24(R1)  
S.D F24, 16(R1)  
BNE R1, R2, Loop  
S.D F28, 8(R1)

Overall 9 cycles for 7 iterations 1.29 per iteration
But VLIW was always half-full

<table>
<thead>
<tr>
<th>ALU /branch</th>
<th>FP</th>
<th>FP</th>
<th>mem</th>
<th>mem</th>
</tr>
</thead>
</table>

Slide based on a lecture by Jelena Mirkovic, University of Delaware
http://www.cis.udel.edu/~sunshine/courses/F04/CIS662/class16.pdf
Some other techniques (I)

• Global scheduling:
  – Compact code a fragment with internal control structure (branches) into the shortest possible sequence that preserves the data and control dependences

• Trace scheduling:
  – Based on loop unrolling
    • Trace selection: determines highly probable outcome of branches and generate sequential portion of the code
    • Trace compaction: parallelize the sequential portion of the code generated in the Trace selection step.
Some other techniques (II)

- Superblocks:
  - Based on loop unrolling
  - Determine the most frequently used path (critical path)
  - Form only one entrance in the path but many exits
  - On exit, some code duplication required