Decoupling Loads for Nano-Instruction Set Computers

Ziqiang (Patrick) Huang, Andrew Hilton, Benjamin Lee
Duke University
{ziqiang.huang, andrew.hilton, benjamin.c.lee}@duke.edu
ISCA-43, June 21, 2016
# In-order vs Out-of-order

<table>
<thead>
<tr>
<th></th>
<th>IO</th>
<th>OoO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td><strong>Area</strong></td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td><strong>Complexity</strong></td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>
What makes OoO perform better?

- **Scheduling**
  - Hardware support \(\rightarrow\) better schedules
    - ROB, Register renamer, Ld/St queue

- **Dynamism**
  - React to dynamic events
    - Cache misses
    - Branch mispredictions

\[ \text{Scheduling} = 88\% \]
\[ \text{Dynamism} = 12\% \]

[McFarlin13]
ISA Evolution: CISC $\rightarrow$ RISC

CISC ISAs are Great!

poly
insque
remque
ISA Evolution: CISC $\rightarrow$ RISC

High “RISC”, High Reward!

ld
add
bne
ISA Evolution: CISC → RISC → NISC

High “RISC”, High Reward!

Specify target
Compute condition
Transfer control
Compute address
Access memory
Write register

Prepare-to-Branch
Branch Delay Slots
Branch Vanguard

Decoupled Load
(Our Work)
Outline

• Motivation
• **Decoupled Loads**
• Microarchitecture
• Compiler
• Evaluation
• Conclusion
Decoupled Load in a Nutshell

• Conventional Load: \texttt{ld \textbackslash rD, I(rA)}
  
  • Compute address
  • Access memory
  • Write register
  • Order
Decoupled Load in a Nutshell

- **Conventional Load:** \( \text{ld } rD, \text{ I}(rA) \)
  - Compute address
  - Access memory
  - Write register
  - Order

- **Decoupled Load:**
  - \( \text{ld.D$ ltA, I}(rA) \)
  - \( \text{ld.wb} \)
  - \( \text{ld.D$ scheduled early} \)
  - \( \text{ld.wb at original position} \)
Key Rule

A decoupled load behaves just as if the load happens at ld.wb

• Stores
• Exceptions
• Coherence

Just follow this rule!
Compiler Challenge: May-Alias Stores

A: ld r1, 0(r2)

B: mul r3, r1, r4

C: st r3, 0(r5)

D: ld r6, 4(r2)

E: mul r3, r6, r4
Compiler Challenge: May-Alias Stores

A: ld r1, 0(r2)
B: mul r3, r1, r4
C: st r3, 0(r5)
D: ld r6, 4(r2)
E: mul r3, r6, r4

What if $r5 = r2 + 4$?

- Compiler can’t schedule loads above may-alias stores
- OoO addresses problem with load/store queue
Decoupled Load to the Rescue

A: ld r1, 0(r2)
B: mul r3, r1, r4
C: st r3, 0(r5)
D: ld r6, 4(r2)
E: mul r3, r6, r4

A: ld r1, 0(r2)
D.0: ld.D$ lt0, 4(r2)
B: mul r3, r1, r4
C: st r3, 0(r5)
D.1: ld.wb r6, lt0
E: mul r3, r6, r4

Remember our rule?

Alias? Not a problem
Compiler Challenge: Branches

A: ld r1, 0(r2)
B: mul r3, r1, r4
C: st r3, 0(r5)
D: add r6, r6, 1
E: bne r6, r7,..

What if?

F: ld r8, 4(r2)
G: mul r3, r8, r4

Exception!

- Compiler can’t schedule loads above branches
- OoO addresses problem with reorder buffer
Decoupled Load to the Rescue

A: ld r1, 0(r2)
B: mul r3, r1, r4
C: st r3, 0(r5)
D: add r6, r6, 1
E: bne r6, r7, ..
F: ld r8, 4(r2)
G: mul r3, r8, r4

Remember our rule?

A: ld r1, 0(r2)
F.0: ld.D$ lt0, 4(r2)
B: mul r3, r1, r4
C: st r3, 0(r5)
D: add r6, r6, 1
E: bne r6, r7, ..
F.1: ld.wb r8, lt0
G: mul r3, r8, r4

Exception?
Not a problem
Not this way?
Wait, Isn’t this?

- Advanced Loads in Itanium
  - No, Itanium is **SPECULATIVE**
  - Require fix up code (software recovery is nasty!)
  - Require large number of registers

- Prefetching
  - No, prefetching is **ORTHOGONAL**
  - Prefetching only helps when data not in L1
Outline

• Motivation
• Decoupled Loads
• **Microarchitecture**
• Compiler
• Evaluation
• Conclusion
Decoupled Loads: Working Example

ld.\text{D}$ lt0, 4(r2)
st r3, 4(r4)
ld.wb r1, lt0

<table>
<thead>
<tr>
<th>LT#</th>
<th>Status</th>
<th>Exn</th>
<th>VA</th>
<th>PA</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Empty</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Decoupled Loads: Working Example

ld.D$  lt0, 4(r2)

st  r3, 4(r4)

ld.wb  r1, lt0

<table>
<thead>
<tr>
<th>LT#</th>
<th>Status</th>
<th>Exn</th>
<th>VA</th>
<th>PA</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Empty</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Decoupled Loads: Working Example

ld.D$ lt0, 4(r2)
st r3, 4(r4)
ld.wb r1, lt0

<table>
<thead>
<tr>
<th>LT#</th>
<th>Status</th>
<th>Exn</th>
<th>VA</th>
<th>PA</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pending</td>
<td></td>
<td>3FCA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Decoupled Loads: Working Example

ld.D$ lt0, 4(r2)
st r3, 4(r4)
ld.wb r1, lt0

<table>
<thead>
<tr>
<th>LT#</th>
<th>Status</th>
<th>Exn</th>
<th>VA</th>
<th>PA</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pending</td>
<td>1</td>
<td>3FCA</td>
<td>8BCA</td>
<td></td>
</tr>
</tbody>
</table>

ld.D$}

D$
Decoupled Loads: Working Example

\[ \text{ld.D$} \text{ lt0, }4(r2) \]
\[ \text{st r3, }4(r4) \]
\[ \text{ld.wb r1, lt0} \]

<table>
<thead>
<tr>
<th>LT#</th>
<th>Status</th>
<th>Exn</th>
<th>VA</th>
<th>PA</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Complete</td>
<td>3FCA</td>
<td>8BCA</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Decoupled Loads: Working Example

\textbf{st} \quad \textbf{ld.D$} \; \text{lt0, 4(r2)} \quad \textbf{st} \; \textbf{r3, 4(r4)} \quad \text{ld.wb r1, lt0}

\begin{array}{|c|c|c|c|c|c|}
\hline
\text{LT\#} & \text{Status} & \text{Exn} & \text{VA} & \text{PA} & \text{Value} \\
\hline
0 & \text{Complete} & 3FCA & 8BCA & 1 \\
\hline
\end{array}

\textbf{Match !}
Decoupled Loads: Working Example

ld.D$ lt0, 4(r2)

st r3, 4(r4)

ld.wb r1, lt0

<table>
<thead>
<tr>
<th>LT#</th>
<th>Status</th>
<th>Exn</th>
<th>VA</th>
<th>PA</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Complete</td>
<td>3FCA</td>
<td>8BCA</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Match!
Decoupled Loads: Working Example

```
ld.D$ lt0, 4(r2)

st r3, 4(r4)

ld.wb r1, lt0
```

<table>
<thead>
<tr>
<th>LT#</th>
<th>Status</th>
<th>Exn</th>
<th>VA</th>
<th>PA</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Complete</td>
<td>3FCA</td>
<td>8BCA</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Decoupled Loads: Working Example

ld.wb

ld.D$ lt0, 4(r2)
st r3, 4(r4)
ld.wb r1, lt0
Decoupled Loads: Working Example

ld.D$ lt0, 4(r2)
st r3, 4(r4)
ld.wb r1, lt0

<table>
<thead>
<tr>
<th>LT#</th>
<th>Status</th>
<th>Exn</th>
<th>VA</th>
<th>PA</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Complete</td>
<td>3FCA</td>
<td>8BCA</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Motivation
• Decoupled Loads
• Microarchitecture
• Compiler
• Evaluation
• Conclusion
List Scheduling

```
ld r1, 0(r2)
mul r3, r1, r4
st r3, 0(r5)
ld r6, 4(r2)
mul r3, r6, r4
...```
List Scheduling

ld r1, 0(r2)
mul r3, r1, r4
st r3, 0(r5)
ld r6, 4(r2)
mul r3, r6, r4

Cycle 0: ld
Bubble Cycles
Cycle 4: mul
Bubble Cycles
Cycle 8: st
Cycle 9: ld
Cycle 13: mul

...
List Scheduling with Decoupled Loads

ld r1, 0(r2)
mul r3, r1, r4
st r3, 0(r5)
ld r6, 4(r2)
mul r3, r6, r4
…

Cycle 0:
d
mul
st
ld
Cycle 4:
Cycle 8:
Cycle 9:
Cycle 10:
List Scheduling with Decoupled Loads

```
ld r1, 0(r2)
mul r3, r1, r4
st r3, 0(r5)
ld r6, 4(r2)
mul r3, r6, r4
...
```

Cycle 0: ld
Cycle 1: ld.D$
Cycle 4: mul
Cycle 8: st
Cycle 9: ld.wb
Cycle 10: mul
Hoisting Above Branches

A
...
ld r4, 0(r7)
ld.D$ lt0, 4(r6)

B
ld.D$ lt0, 4(r6)
blt r3, r5, B

99%

100%

1%

C
ld.wb r5, lt0
...

Correctness
Performance
Hoisting Above Branches

A
...  
ld r4, 0(r7)  
ld.D$ lt0, 4(r6)  

B
ld.D$ lt0, 4(r6)  
blt r3, r5, B  

99%

C
ld.wb r5, lt0  
...

A
...  
ld r4, 0(r7)  
ld.D$ lt0, 4(r6)  

B
blt r3, r5, B  
ld.D$ lt0, 4(r6)  

99%

C
ld.wb r5, lt0  
...

Correctness

Performance

100%

1%

100%
Outline

• Motivation
• Decoupled Loads
• Microarchitecture
• Compiler
• Evaluation
• Conclusion
Evaluation: Methodology

- ISA: OpenRISC
- Compiler: LLVM 3.5
- Benchmarks: SPEC2006, 8 INT
- Cycle Level Simulation
Results: Speedup & Instruction Mix

- Average speedup is 8.4%
- Speedup correlates with decoupled loads
Hoisting above Stores versus Branches

- Scheduler must handle both stores, branches
Are We Simply Prefetching?

![Graph showing speedup (%) for different benchmarks with the label 'Perfect L1 + Decoupled Loads.']

- bzip2
- h264ref
- hmmmer
- libquantum
- mcf
- omnetpp
- perlbench
- sjeng
Are We Simply Prefetching?

- Majority of benefits from hiding hit latency
Outline

• Motivation
• Decoupled Loads
• Compiler
• Microarchitecture
• Evaluation

• Conclusion
Towards Promised OoO Performance

- Decoupled loads separate access from writeback and ordering
- Improve static scheduling for IO performance
- Bring IO a step closer to OoO
- Require modest microarchitectural and system support

Decoupled Loads
Branch Vanguard [McFarlin15]

Scheduling Algorithms
iCFP [Hilton09]
Thank you!

Questions?