Dynamic Speculation-Synchronization of Data Dependences

Guri Sohi

work with:

Andreas Moshovos, Scott Breach, T. N. Vijaykumar

Computer Sciences Department
University of Wisconsin — Madison
URL: http://www.cs.wisc.edu/~sohi
Dynamic Speculation-Synchronization of Data Dependences...

Overview

Program Order

STORE  ???  \( R_x \)

LOAD  100  \( R_y \)

Dependence

YES  LOAD has to wait

NO  LOAD may execute immediately

Opportunity for higher ILP

Unfortunately don’t know in advance!

Solution: Speculate whether a dependence exists
Overview

So far two policies:

Speculate Never
   Safe but loss of opportunity
Speculate Always (blind)
   Penalty when wrong, but it pays (today)

Argue:

   As the window size increases
   Net penalty of mis-speculation becomes significant
   Room for significant improvement over both policies

Our Solution:

   (1). Predict Dependences
   (2). Force Synchronization
Roadmap

• **Overview**

• **The Problem and Our Solution** (14 slides)
  • Dependence Speculation and Performance
  • Impact of Window Size
  • Ideal Solution - Alternatives
  • Our Solution

• Evaluation (7 slides)

• Other uses - Ongoing work
Dependence Speculation

Program Order

<table>
<thead>
<tr>
<th>store</th>
<th>load</th>
<th>inst A</th>
<th>inst B</th>
<th>inst C</th>
<th>inst D</th>
</tr>
</thead>
</table>

Execution Order

<table>
<thead>
<tr>
<th>No Speculation</th>
<th>Speculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst A</td>
<td>load</td>
</tr>
<tr>
<td>inst C</td>
<td>inst B</td>
</tr>
<tr>
<td>load</td>
<td>inst D</td>
</tr>
</tbody>
</table>

No Dependence

Dependence

Speculation may affect performance either way
Dependence Speculation and Performance

**Performance**

\[ \text{Gain} \times (100\% - \text{Mis-speculation}\%) \]
\[ - \text{Penalty} \times \text{Mis-speculation}\% \]

- **ASAP**
- **SPECULATE**
- **Never**

<table>
<thead>
<tr>
<th></th>
<th>Gain</th>
<th>Penalty</th>
<th>Mis-speculation%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gain</strong></td>
<td>↑</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Penalty</strong></td>
<td>↑↓</td>
<td>↓↓↓</td>
<td></td>
</tr>
<tr>
<td><strong>Mis-speculation%</strong></td>
<td>↑↓</td>
<td>↓↓↓</td>
<td></td>
</tr>
</tbody>
</table>

- **Balance** between Gain and Penalty
Dependences vs. Window Size

Frequency of loads with Dependences within the Window
Small Instruction Windows and Speculation

Small Instruction Window:

- Loads are speculated past few instructions
- Dependences are infrequent

Blind Speculation a good choice:

- Mis-speculations are infrequent
- Low probability of other, independent work
- Low mis-speculation penalty

Not Speculating at times is acceptable.
Wider Instruction Windows

As the Window size increases:

- Loads are speculated past many more instructions
- Dependences become more frequent

Overall:

- Mis-speculations are more frequent
- Higher probability of other, independent work
- Higher mis-speculation penalty

Blind Speculation is still a viable approach

Not Speculating is not

HOWEVER! Net penalty of mis-speculation becomes significant

Potential for performance improvement
Reducing the Net Mis-speculation Penalty

**Ideally:**

- Dependent load/store pairs are synchronized
- Other loads execute as early as possible

Code

Blind Speculation

Ideal Speculation
To mimic the ideal we need:

(1). Identify the loads that have dependences
(2). Identify the relevant stores
(3). Enforce synchronization

Can we do without synchronization?

How about selective speculation:

• Identify the loads that have dependences
• Do not speculate them
Selective Dependence Speculation

- Selective may perform worse than blind
- Can also perform as well as the ideal
- In practice:
  performance behavior varies
Our approach

Attempt to mimic the Ideal:

• To identify the dependent load/store pairs:
  
  **Predict!**
  
  Based on the history of mis-speculations

• To synchronize:
  
  Use *dynamically assigned* synchronization variables
Predicting Dependences

- Dependence: (Load PC, Store PC)
- Temporal locality - Small Working Set.
- Use a small table to:
  1. track recent mis-speculations
  2. Predict dependences

**Memory Dependence Prediction Table**

<table>
<thead>
<tr>
<th>LDPC</th>
<th>STPC</th>
<th>PRED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

① Misspeculation
② Allocate entry

① Execute?
② No! Synchronize

① Synchronize?
② Synchronize
Synchronization - Load Waits

- Provide a small pool of full/empty bits
- Use \((LD\ PC,\ ST\ PC)\) to associate entries w/ dependences

Memory Dependence Synchronization Table

<table>
<thead>
<tr>
<th>MDPT</th>
<th>MDST</th>
<th>F/E V</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDPC</td>
<td>STPC</td>
<td>PRED</td>
</tr>
</tbody>
</table>

① May I Execute? (LDPC)
② No Wait
③
Synchronization - Load Resumes

Memory Dependence Synchronization Table

1. Do I need to synchronize?
2. Probably
3. Anyone waiting?
4. Resume Execution

<table>
<thead>
<tr>
<th>MDPT</th>
<th>LDPC</th>
<th>STPC</th>
<th>PRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>F/E V</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MDST</th>
<th>LDPC</th>
<th>STPC</th>
<th>F/E V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multiple Instances of the Same Dependence

- Can’t just use (Load PC, Store PC) for synchronization

In addition to (Load PC, Store PC) use:

- The data address accessed, OR

- Attempt to determine the dependence distance

*Analogous to simple static linear recurrence analysis*

- In Multiscalar we use the distance in stages
- In a superscalar we may count the # of stores
Other alternatives exist for both prediction and synchronization.

Simplifications may be possible.

For example:
- Use PC to identify only loads
- Use the data address to indirectly identify the stores and to synchronize
Roadmap

• Overview
• The Problem and Our Solution
• Evaluation (7 slides)
  • Comparison of speculation policies
  • Accuracy of prediction
  • Reduction in Mis-speculation rate
  • Speedup
• Other uses - Ongoing work
Evaluation - Methodology

- Use the Multiscalar architecture
  - is able to establish wide windows
  - uses memory data speculation aggressively
  - penalty of mis-speculation is significant

- Parameters
  - 8 stages, 2-way OoO
  - 32k 2-way Icache, 128k direct mapped data cache
  - 512 entry ARB
  - Non-blocking loads/stores

- Benchmarks
  - SPEC’95 INT and FP (compiled w/ gcc 2.7.2)
  - train/test inputs up to 2 Billion instructions
  - SPEC’92 for some experiments

- Instruction driven timing simulation
Evaluation - Parameters

- Centralized scheme
- 64 entry combined MDPT/MDST
- Fully associative
- Multiple dependences: wait for all
- Multiple instances: use dependence distance
- Predictor:
  - 3-bit counter based (threshold of 3)
  - Also maintains minimal control path information:
  - Records the PC of the task that issued the store
Comparison of Speculation Policies

![Bar chart showing speedups for different programs and speculation policies.]

- Speedups are relative to no speculation (IPC along X axis)
- Perfect dependence prediction is used
Dependence Prediction Accuracy

[Bar chart showing predicted vs. actual for various programs with accuracy rates]
Dynamic Speculation-Synchronization of Data Dependencies...

Mis-speculation Rates

8 Stages

Mis-speculation Rates

Ours

Blind
Speedup - SPECint95

- Speedups are relative to blind speculation
- IPC w/ our mechanism
Speedup - SPECfp95

SPECfp95 - 8 Stages

- Speedups are relative to blind speculation
- IPC w/ our mechanism
Roadmap

• Overview
• The Problem and Our Solution
• Evaluation
• Other uses - Ongoing work
  • Speeding up the communication of data values
  • Other
Other Uses of Dependence Prediction

- Speedup the communication of memory data values
  1. When store is **decoded**, predict the dependent load
  2. Associate the **value** or **creator** with the **PC** of the load

---

Slide 28
Other Uses of Dependence Speculation

- When load is **decoded**, use its PC to determine the value or creator
- Dependent Instructions may use the speculative value
- Load is also executed, to verify the value speculation

![Diagram showing the process of dependence speculation]

1. Dependence?
2. Yes!
3. Value?
4. Yes!
5. Use Ry

**Load Ry**
Ongoing Work

- Alternative prediction/synchronization methods
- Superscalar environment
- Distributed organization
- Integration with the memory disambiguation mechanism
- Sensitivity analysis
- Exposing to the compiler