EECS 452 – Lecture 6
Memory Dependencies

Instructor: Gokhan Memik
EECS Dept., Northwestern University
OOO and Load/Store Instructions

- Support for Speculative Stores
  - Stores write into temporary store buffer
    - Entries tagged with address and contain data
    - Entries allocated at dispatch in program order

- Loads access first the store buffer
  - Scan backward with address to find matching store
  - Get data if available
  - Otherwise access memory

- Implements memory renaming
Store Buffer Renaming Example

- Associative searches
- Ordering enforcing

<table>
<thead>
<tr>
<th>type</th>
<th>addr</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>store</td>
<td>0x100</td>
<td>30</td>
</tr>
<tr>
<td>store</td>
<td>0x200</td>
<td>40</td>
</tr>
<tr>
<td>load</td>
<td>0x100</td>
<td>NA</td>
</tr>
<tr>
<td>store</td>
<td>0x100</td>
<td>50</td>
</tr>
<tr>
<td>load</td>
<td>0x100</td>
<td>NA</td>
</tr>
</tbody>
</table>
Store buffer complications

- Data types: complications
  - Store byte 0x100, 10
  - Load word 0x100 → r1
    - Access cache and combine
Scheduling Loads

- Requires knowledge of memory dependences
- This can be done only if we know all addresses of all loads/stores
- Unfortunately, these are produced out-of-order

Solution #1:
- Loads wait for all preceding stores
  - To calculate their addresses
  - To have their data ready
- OK for small windows, BAD for large windows
Scheduling Loads

- Wait until you can determine all memory dependences
- Wait until all preceding stores calculate their address
  - Wait only for those that you have a conflict with
- Mechanism is fairly elaborate
Scheduling Loads

- Use the store buffer as a load scheduler
- Loads and stores post their addresses and data
- Load can execute when:
  - All preceding store address are known
  - Either there is no conflict with preceding store
  - There is a conflict with a store that has its data ready
Scheduler Actions

- **At dispatch:**
  - Allocate next sequential entry
  - Mark addr and data as unavailable

- **Store address**
  - Scan forward (in time)
  - Stop if store with unknown address
  - Stop if store with same address
  - Wakeup non-matching loads if no preceding store with unknown address
  - Wakeup matching loads if have data

- **Store data after address**
  - Repeat above
Scheduler Actions

- Load address calculated
  - Scan backward (in time)
  - Stop if matching store with data → can execute
  - Stop if store with unknown address → can’t execute
  - Stop if matching store without data → can’t execute

- Performs better than no scheduling at all
- Not good for larger instruction windows
Memory Dependence Speculation

- Allow a load to execute **before** a store with which a memory dependence **may** exist.

**Naïve Speculation:**
- Load with address known executes immediately
- If a preceding store later detects a dependence
  - Load and everything after it are squashed
    - Very much like branch miss-prediction
  - If not, performance improves since load got to execute early

- This works fairly well: few loads have dependences
- But, dependence frequency rises for larger windows
Memory Dependence Speculation Revisited

- For large instruction windows
  - More loads have dependences
  - Naïve speculation miss-predicts often enough so that there is room for improvement

- Need more intelligent speculation methods
Store Barrier

- Start with naïve speculation

- On miss-speculation record PC of offending store
  - In a prediction table

- Next time the same store is fetched
  - Make all loads after it wait until it calculates its address

- Delays many loads unnecessarily

- Good for small windows

- Was planned for PPC 620
Selective Speculation

- Start with naïve
- On miss-prediction record PC of load
- Next time the load is fetched make it wait for all preceding stores

- Tricky tradeoff
- Sometimes it’s faster to speculate, squash and re-execute than wait
  - When store and load are far apart
  - This is the common case in large windows
Speculation/Synchronization

- Start with naïve
- On miss-prediction, record the PCs of the offering store and load
- Next time synchronize
- Attempts to approximate the ideal
  - A load waits only for the store it depends on if any
Speculation/Synchronization

- Can use regular register renaming hardware to do the synchronization
- A dependent load and store are associated with the same name (synonym) on miss-prediction
- Upon decoding the store, the synonym is predicted and renamed into a dummy register
- Upon decoding the load, the synonym is predicted, it is used to locate the dummy register and the load is forced to wait for the store
Memory Dependence Speculation

Bibliography


- Also the Memory Conflict Buffer is a software/hardware solution primarily for VLIW processors, but in principle applicable to any processor
- Transmeta’s Crusoe implements the MCB and so does Intel’s EPIC