Still Speculating After All These Years

Guri Sohi
University of Wisconsin-Madison
Outline

• Speculation infancy
  – performance

• Speculation adolescence
  – energy

• Speculation maturity
  – security
Speculation Infancy

• Performance was primary design objective
• Simple hardware was key to performance
• RISC: relegate interesting stuff to compiler
• VLIW/EPIC was the future
  – Speculation OK in software but not hardware
Speculation: Infancy

• Stand alone machines were king
  – Mainframes, workstations, PCs

• Network connectivity was for the elite
  – First Web browser: 1990-91

• As were problems due to connectivity
  – Morris worm: Nov 1988
Speculation Infancy

• Yet hardware speculation became the norm
• Increasing comfort level leads to more forms of speculation
  – Golden age of microarchitecture
• Yet the naysayers continue
Speculation Adolescence

• Energy/Power become important design goal
• Attacks on speculation continue using new argument
Speculation Adolescence

• Move to multicore
  – Energy efficiency argument
• Single thread performance is dead
• Programming means parallel programming
Speculation Adolescence

• Use of speculation continues to increase
  – E.g., data dependence speculation

• Speculation may actually be an energy win overall
  – E.g., run to halt
Speculation Maturity

• Speculative execution used in processors across the spectrum
  – From servers to low-end mobile devices

• Security is of increasing importance
Speculation Maturity

• Speculative execution is now a security problem

• Flaw in design

Question: someone crossing the road is hit by a car. What comes to mind?
Speculation Maturity

• Speculation is a security problem
• No more speculation
• Architecture is obsolete; we need Architecture 2.0

Relax, stop pontificating and think!
Speculation and Security

• Speculation impacts micro-architectural state
  – E.g., speculative (and later squashed) load brings data into cache

• Adversary can observe such state

Control speculation of loads
Controlling load speculation

• All loads

• Selective loads
  – E.g., those using result of another load for address calculation

• Other (additional) policies for selection
Percent Pointer Chase Loads

Number of pointer chased loads for the different applications

<table>
<thead>
<tr>
<th>Application names</th>
<th>Number of loads</th>
</tr>
</thead>
<tbody>
<tr>
<td>leela</td>
<td>0</td>
</tr>
<tr>
<td>xz</td>
<td>0</td>
</tr>
<tr>
<td>mcf</td>
<td>20</td>
</tr>
<tr>
<td>deepsjeng</td>
<td>10</td>
</tr>
<tr>
<td>x264</td>
<td>20</td>
</tr>
<tr>
<td>average</td>
<td>10</td>
</tr>
</tbody>
</table>
Stalling Loads

![Graph showing slowdown with stalling of pointer chased load instructions]

- **Title**: Slowdown with stalling of pointer chased load instructions
- **X-axis**: Application names (leela, xz, mcf, deepsjeng, x264, average)
- **Y-axis**: Percentage Slowdown
- **Legend**:
  - Pointer chased loads stalled
  - All loads stalled

The graph illustrates the percentage slowdown for different applications due to stalling loads.
Random Selective Stalling

Slowdown with pointer chased loads picked at random

- Unconditional stall
- Stall with prob. 0.5
- Stall with prob 0.25
- Stall with prob 0.125

Application names:
- leela
- xz
- mcf
- deepsjeng
- x264
- average

Percentage Slowdown:
- leela: 3%
- xz: 2%
- mcf: 5%
- deepsjeng: 2%
- x264: 3%
- average: 3%
Reminiscing about the Old Days

• Virtual caches
  – the original caches before things became complicated

• Complications thwarted their adoption
  – Software solution not feasible
  – Proposed hardware designs problematic
Virtual Caches

• Performance win
  – No address translation latency

• Energy win
  – No address translation energy for L1 hit

Now security is an important parameter
Virtual Cache VC-DSR

• Recently proposed virtual cache design
• Data cached with only one (primary) virtual address
• Dynamic Synonym Remapping
  – Synonym virtual address mapped to primary virtual address with which data cached
  – Many interesting issues need to be dealt with
Virtual Cache VC-DSR

Phases

1. Virtual Address Generation
   - CPU
   - $V_i \rightarrow <\text{ASID, Vaddr}>

2. Active Synonym Remapping
   - Synonym Signature (SS)
   - ART
   - Address Remapping Table

3. L1 Virtual Cache
   - L1 Virtual $\rightarrow$ Hits
   - Misses

4. Active Synonym Detection
   - TLB
   - [V → P translations]
   - ASDT
   - [P → Leading_V translators]

5. Lower-Level Physical Caches
   - L2 Physical $\rightarrow$ Physical Address
   - Coherence

Translations:
- $V \rightarrow P$ translations
- $P \rightarrow$ Leading_V translations
Virtual Cache VC-DSR

• Virtual, or any other address, can be used for L1 cache
  – Conventional physical address in L2 and beyond
• Address scheme different for different pages, and same page at different times
  – VC-DSR tracks page-level info to ensure correct operation
Virtual Cache VC-DSR

• Very frequent address mapping changes in L1
  – Every time block from new page into L1
  – In resource with highest rate of info leakage
  – At fine (e.g., at page-level) granularity
  – Without perturbing other parts of system

Modern version of VC may also security win
Summary

• We will still be speculating after many more years
  – despite new criteria arguing against them
• Security is a new design criteria
• Successful solutions may involve adapting old ideas to new goals
  – Back to the future