Boosting Trace Cache Performance with NonHead Miss Speculation

Stevan Vlaović

Architecture and Advanced Development
Sun Microsystems

June 25, 2002      ICS 2002
Motivation

• Performance of new trace cache design (linked list vs. fixed length)
• How to improve on this design in the context of x86 ISA
• Not limited to x86, but is applicable to any design that utilizes a linked list style trace cache
• Especially helpful with longer pipelines
Trace Cache Designs

Academic [Rotenberg96][Patel97]

Issue Width

Pentium 4[Hinton Patent97]

Pentium 4 issue width = 3μops, but TC allows long traces
NonHead Miss

Dynamic Istream

Trace (in trace cache)

- Basic block B is executed, not C
- Only find out in Execute stage!
- Only then look up trace that starts with basic block B
Outline

• Trace Analysis For X86 Interpretation (TAXI)
• Applications
• NonHead Miss Speculation
• Conclusions
Modern X86 Processors

• High performance ==> dynamic translation of x86 instructions into smaller, more RISC-like instructions (µops)
  • Back end can be similar to other, more conventional processors
  • Complexity is moved to the front end
Outline

• Trace Analysis For X86 Interpretation (TAXI)
• Applications
• NonHead Miss Speculation
• Conclusions
Trace Analysis for X86 Interpretation [to appear in ICCD ‘02]

• Simulator
  • functional (state information, correctness)
  • performance (dependencies, timing, resource utilization)

• To do a performance simulator need to do x86 instruction to \( \mu \text{op(s)} \) mapping for each instruction
  • for our applications: 150 instructions, 9 different addressing modes
Example Decomposition

ADD addr, REG

LW Temp1, addr
ADD Temp1, Temp1, REG
SW Temp1, addr
TAXI Attributes

• Trace-based
  • No wrongpath execution
  ± Separate functional/performance simulations

• Bochs
  • limited device models (e.g. monitor)
    + captures OS activity

• sim-outorder (SimpleScalar)
  + easy to modify
Outline

• Trace Analysis For X86 Interpretation (TAXI)
• Applications
• NonHead Miss Speculation
• Conclusions and future work
Applications

- Id’s Doom
- Microsoft Explorer 5.0
- Filemaker Pro 5.0
- Microsoft Visual C++ 5.0
- Netscape Navigator 6.0
- Real System’s RealPlayer 8.0
- Winamp 2.72
- Winzip 8.0
<table>
<thead>
<tr>
<th>Application</th>
<th>Inst</th>
<th>µ/ln</th>
<th>In/St</th>
<th>In/Ld</th>
<th>In/Br</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doom</td>
<td>388</td>
<td>1.52</td>
<td>4.61</td>
<td>2.64</td>
<td>5.40</td>
</tr>
<tr>
<td>Explorer</td>
<td>396</td>
<td>1.50</td>
<td>4.53</td>
<td>2.55</td>
<td>4.77</td>
</tr>
<tr>
<td>FileMaker</td>
<td>447</td>
<td>1.49</td>
<td>4.61</td>
<td>2.85</td>
<td>5.04</td>
</tr>
<tr>
<td>MsDev</td>
<td>469</td>
<td>1.46</td>
<td>4.94</td>
<td>2.63</td>
<td>4.07</td>
</tr>
<tr>
<td>Netscape</td>
<td>377</td>
<td>1.52</td>
<td>4.53</td>
<td>2.65</td>
<td>4.33</td>
</tr>
<tr>
<td>RealPlayer</td>
<td>522</td>
<td>1.44</td>
<td>5.44</td>
<td>2.76</td>
<td>4.63</td>
</tr>
<tr>
<td>Winamp</td>
<td>456</td>
<td>1.47</td>
<td>4.11</td>
<td>2.31</td>
<td>6.88</td>
</tr>
<tr>
<td>Winzip</td>
<td>302</td>
<td>1.44</td>
<td>5.74</td>
<td>3.05</td>
<td>5.63</td>
</tr>
</tbody>
</table>
Application Characteristics
(vs. CPU2000)

- Worse instruction and data cache performance
  - larger applications
  - use much more OS support
  - 32 KB or 64 KB caches generally sufficient

- Generally worse branch prediction performance
  - more indirects
  - more frequent branches, except for streaming
Outline

• Trace Analysis For X86 Interpretation (TAXI)
• Applications
• NonHead Miss Speculation
• Conclusions
Model Trace Cache (TC)

- Holds $\mu$ops, not x86 instructions
- 12 K $\mu$ops
- Provides $\mu$ops that are physically noncontiguous
- Traces composed of segments (TC lines)
  - Each segment holds 6 $\mu$ops [at most 2 branches/segment]
  - Possibly multiple segments per trace
  - First segment called head, last segment called tail
- Trace ends on the following conditions:
  - Indirect branch
  - Call
  - Return
  - Allows up to 64 segments/trace
Functional Diagram

L2 Cache → Streaming Buffer → Trace Cache → DIQ → Back end

Bpred → BTB
Model Parameters

• **Front end**
  - 12 K-\(\mu\)ops, 4-way trace cache (512 entry BTB)
  - 1 cache line (64 bytes) Streaming Buffer
  - 4 K-entry GAg predictor (4 K-entry BTB)
  - 1 complex (>5 \(\mu\)ops) decoders

• **Back end**
  - 8 entry Decoded Instruction Queue
  - 4 \(\mu\)op issue and commit width
  - 8 \(\mu\)op decode width
  - 126 ROB (RUU) entries
  - 8 KB, 4-way L1 data cache
Trace Cache Access Classification (8 cases)
Trace Cache Access Classification

- BTB hit with correct address followed by a trace cache head hit
- BTB hit with correct address followed by a trace cache head miss
- BTB hit with wrong address followed by a trace cache head hit
- BTB hit with wrong address followed by a trace cache head miss
- Nonhead miss, followed by a trace cache head hit
- Nonhead miss, followed by a trace cache head miss
- BTB miss followed by a trace cache head hit
74% of all TC accesses are BTB hits (w. correct addr.) followed by a Head hit.

15% of all accesses are Nonhead misses

BUT 14% of all accesses are Nonhead misses that then hit in the trace cache as the Head of a new Trace.

BUT Nonhead misses discovered at Execute (avg. 17 cycles after TC)
BTB Access

Hit

Correct Address

TC Head Lookup

Hit

More Blocks

no

Exit

yes

TC Body Pointer

Correct Address

Wrong Address

Miss

Wrong Address

Get Address of next block from execution pipeline

BTB Miss

Head Miss

Nonhead Miss

Trace Cache Access Classification (8 cases)
Trace Cache Access Breakdown

- 74% of all TC accesses are BTB hits (w. correct addr.) followed by a Head hit
- 15% of all accesses are Nonhead misses
- **BUT** 14% of all accesses are Nonhead misses that then hit in the trace cache as the Head of a new Trace
- **BUT** Nonhead misses discovered at Execute (avg. 17 cycles after TC
Trace Cache Access Classification (8 cases)
74% of all TC accesses are BTB hits (w. correct addr.) followed by a Head hit

15% of all accesses are Nonhead misses

BUT 14% of all accesses are Nonhead misses that then hit in the trace cache as the Head of a new Trace

BUT Nonhead misses discovered at Execute (avg. 17 cycles after TC access)!
BTB Access

- Hit
  - Correct Address
  - Wrong Address
  - TC Head Lookup

- Miss
  - BTB Miss
  - Get Address of next block from execution pipeline

TC Head Lookup

- Hit
- Miss

More Blocks

- no
  - Exit
- yes
  - TC Body Pointer
  - Correct Address
  - Wrong Address

Trace Cache Access Classification
(8 cases)
Trace Cache Access Breakdown

- 74% of all TC accesses are BTB hits (w. correct addr.) followed by a Head hit
- 15% of all accesses are Nonhead misses
- BUT 14% of all accesses are Nonhead misses that then hit in the trace cache as the Head of a new Trace
- BUT Nonhead misses discovered at Execute (avg. **17 cycles** after TC)
Optimal nonhead miss handling
(discover wrong address immediately, 17 -> 0 cycles)
NonHead Miss Speculation (NHMS)
Miss Predictor Detail

16-bit Nonhead History Register

Table of 3-bit Counters
64 K-entry Miss Predictor

- 72.5% of Nonhead misses correctly predicted
- 27.5% of Nonhead misses not predicted to miss
- # of Nonhead hits predicted to miss
  - 0.22% of all accesses to the predictor (once per 455 x86 instructions)
  - equals only 4.2% of all nonhead misses (net effect is as if 68.3% of nonhead misses discovered early)
## 4 K NonHead Miss Target Buffer (Hit Rate)

<table>
<thead>
<tr>
<th>Application</th>
<th>Hit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doom</td>
<td>97.7</td>
</tr>
<tr>
<td>Explorer</td>
<td>98.0</td>
</tr>
<tr>
<td>FileMaker</td>
<td>98.3</td>
</tr>
<tr>
<td>MsDev</td>
<td>99.2</td>
</tr>
<tr>
<td>Netscape</td>
<td>96.3</td>
</tr>
<tr>
<td>RealVideo</td>
<td>99.4</td>
</tr>
<tr>
<td>Winamp</td>
<td>99.3</td>
</tr>
<tr>
<td>Winzip</td>
<td>98.7</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>98.4</strong></td>
</tr>
</tbody>
</table>
Impact of Adding NHMS
(removes 80% of NonHead Miss Penalty on average)
NHMS Results

- BTB hit then head hit increases from 74% to **84%**
- Nonhead misses reduced from 15% to **4%**
- Average \( \text{CP} \) improvement \( \sim 10\% \)
  - only 2% for Netscape
  - 20% for MsDev
  - 15% for RealVideo
  - 14% for Winamp
Conclusions

• 8 categories of Model trace cache accesses help identify the majority of misses as nonhead misses.
• NHMS removes most of this penalty, yielding almost 10% performance improvement.
NHMS vs. Other Implementations

![Bar chart comparing NHMS vs. Other Implementations for various applications. The chart includes applications such as doom, explorer, filemaker, msdev, netscape, realvideo, winamp, winzip, and hmean. The y-axis represents CPU utilization (CP), and the x-axis represents different applications and benchmarks. The chart uses different colors to represent different implementations and benchmarks.]
% of NonHead Miss Penalty Removed

<table>
<thead>
<tr>
<th>Software</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doom</td>
<td>70%</td>
</tr>
<tr>
<td>Explorer</td>
<td>80%</td>
</tr>
<tr>
<td>FileMaker</td>
<td>85%</td>
</tr>
<tr>
<td>MsDev</td>
<td>90%</td>
</tr>
<tr>
<td>Netscape</td>
<td>80%</td>
</tr>
<tr>
<td>RealVideo</td>
<td>90%</td>
</tr>
<tr>
<td>Winamp</td>
<td>90%</td>
</tr>
<tr>
<td>Winzip</td>
<td>80%</td>
</tr>
<tr>
<td>Average</td>
<td>80%</td>
</tr>
</tbody>
</table>
Related Work (Trace Cache)

• Decoded Instruction Cache [Patt85]
  • Fill unit: larger piece of atomic work that could be given to the dynamic scheduler [Melvin88]
  • VAX instructions decomposed into microoperations [Melvin89]
    • CPI 6-> 2 for current VAX implementations
  • Pentium Pro could use a fill unit [Franklin95]

• Trace Cache
  • Branch Address Cache [Yeh93] [Dutta95]
  • Patent [Peleg94]