RDIP: RAS-Directed Instruction Prefetching
Aasheesh Kolli, Thomas F. Wenisch (University of Michigan) and Ali Saidi (ARM)

1. The instruction fetch bottleneck
Instruction cache performance continues to be a bottleneck for modern server workloads.

2. Why another instruction prefetcher?

<table>
<thead>
<tr>
<th>Next-2-line (N2L)</th>
<th>Proactive Instruction Fetch (PIF) (Ferdman ‘12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industry standard</td>
<td>State-of-the-art academic proposal</td>
</tr>
<tr>
<td>+ Low overhead, implementability</td>
<td>+ Best performance</td>
</tr>
<tr>
<td>+ Modest performance gains</td>
<td>+ High overhead (&gt; 200kB), design complexity</td>
</tr>
</tbody>
</table>

Our goal: Build a low overhead, high accuracy prefetcher

3. Insights
- I$ misses correlate strongly to program context
- Program contexts are predictable
- RAS succinctly captures program context

4. RDIP design and challenges

<table>
<thead>
<tr>
<th>Design steps</th>
<th>Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Use RAS signatures to represent program contexts</td>
<td>Accurate representation</td>
</tr>
<tr>
<td>2. Map I$ misses to signature in Miss Table</td>
<td>Minimize storage</td>
</tr>
<tr>
<td>3. Prefetch on next occurrence of signature</td>
<td>Timely prefetching</td>
</tr>
</tbody>
</table>

5. RAS signature generation

Call: XOR contents of RAS after push onto RAS, append 0
Return: XOR contents of RAS before pop from RAS, append 1

Example:

- Dynamic Instructions
  - A:funcX{
  - B:funcY{
  - C:funcY{

<table>
<thead>
<tr>
<th>RAS contents</th>
<th>RAS signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(A)</td>
</tr>
<tr>
<td>B</td>
<td>(A)</td>
</tr>
<tr>
<td>C</td>
<td>(A)</td>
</tr>
<tr>
<td>A</td>
<td>(A)</td>
</tr>
</tbody>
</table>

- XOR contents of RAS

6. Timely prefetching
Mapping I$ misses with corresponding signature \(\rightarrow\) late prefetches.

7. RDIP hardware summary

8. RDIP practical design
- RAS size for signature generation \(\rightarrow\) 4 top entries
- Miss Table \(\rightarrow\) 4k entries (4-way associative)
- Entry size \(\rightarrow\) 16B (compaction technique [Ferdman ‘11])

Total storage overhead: 64kB

9. Simulation Methodology
- gem5 full system simulator
- Core parameters:
  - Core: 200Hz 64O, 8-wide commit, 16-entry RAS
  - I-Cache: 32kB/2-way/64B, 2 cycles
  - D-Cache: 64kB/2-way/64B, 3 cycles
  - L2: 2MB/8-way/64B, 24 cycles
- Workloads:
  - gem5 – gem5 running a spec benchmark (hotdot)
  - HD-teraread – Hadoop: Big data search MapReduce job
  - HD-workload – Hadoop: Word count
  - ssj – Tests Java performance in SPECpower
  - MC-friendfeed – Memcached: “Facebook”-like app
  - MC-microblog – Memcached: “Twitter”-like app

10. Coverage and erroneous prefetches

An ideal prefetcher tries to maximize coverage (def: prefetchHiTs over prefetchHiTs+misses) and minimize erroneous prefetches.

 Coverage: PIF \(\sim\) RDIP > N2L

11. Performance

Performance increase: N2L 5%, PIF 13%, RDIP 11.5%, Ideal 16%

12. Conclusion
In this work, we show the correlation that exists between I$ misses and program contexts and develop a mechanism to use the RAS state to represent program contexts. Leveraging these insights we develop a prefetching mechanism, RDIP. RDIP performs comparably to the state-of-the-art prefetchers at one third storage costs and simpler design.