RAS-Directed Instruction Prefetching (RDIP)

Aasheesh Kolli*  Ali Saidi†  Thomas F. Wenisch*

* University of Michigan  † ARM

MICRO-46 12/10/2013
• Poor I$ behavior affects modern server workloads
  – [Spracklen ‘05][Ferdman ‘08] [Ferdman ‘11]

• Cache size constraints → Prefetching necessary
Why another prefetcher?

- Next-2-line (N2L)
  + Low overhead
  - Modest benefits, ineffective at discontinuities
- Proactive Instruction Fetch (PIF) [Ferdman ‘11]
  + Best performing academic proposal
  - Storage overhead (> 200kB per core)
  - Design complexity

Our Goal: Low overhead, high accuracy prefetcher
Contributions

• I$ misses - program context correlation
• Program contexts are repetitive, predictable
• RAS succinctly captures program context

RAS-Directed Instruction Prefetching (RDIP)

RDIP achieves 11.5% increase in performance with only 64kB overhead
Outline

• Design overview
• RAS signature generation
• Timely prefetching
• Results
• Conclusions
RDIP design overview

- I$ misses correlate to program context
- Program contexts are predictable
- RAS state represents program contexts

1. Represent program context using a RAS signature
2. Map cache misses to signatures
3. Prefetch upon next occurrence of signature
RDIP design challenges

1. **Hash** RAS contents to generate program context signatures
   - Challenge: Accurately represent program contexts

2. **Record** cache misses associated with signature in *Miss Table*
   - Challenge: Minimize storage

3. **Prefetch** upon signature change based on *Miss Table*
   - Challenge: Ensure timely prefetches
Outline

• Design overview
• RAS signature generation
• Timely prefetching
• Results
• Conclusions
Generating program context signatures

• Use contents of RAS to represent contexts
• Cannot use entire RAS – need compact signatures
• Signatures must differentiate traversals up & down call stack

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

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

Dynamic Instructions | RAS Signature | RAS
A:funcX{          | (A)0          |
    B:funcY{       | (A⊕B)0        |
}                  | (A⊕B)1        |
    ....           |               |
    C:funcY{       | (A⊕C)0        |
}                  | (A⊕C)1        |
Outline

• Design overview
• RAS signature generation
• Timely prefetching
• Results
• Conclusions
Timely prefetching

- *Miss Table* stores signature-misses pairs
- Prefetches issued by looking up *Miss Table*
- If misses tagged with current signature? → Too late!

Mapping misses to previous signature → Timely prefetches
Issuing prefetch requests

Current Signature: $S_1$

Previous Signature

Miss Table:
- $S_2$

Prefetch Queue: $S_1$

To L2$
Updating miss table

Current Signature

$S_2$

Miss Buffer

Misses tagged with previous signature $\rightarrow$ Timely prefetching!

Previous Signature

$S_1$

Signature Change!

Update

From L2$
Outline

• Design overview
• RAS signature generation
• Timely prefetching
• Results
• Conclusions
Experimental setup

- **gem5**
  - **Core**: 2GHz OoO, 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 (twolf)
  - **HD-teraread** – Hadoop: Big data MapReduce job
  - **HD-wdcnt** – Hadoop: Word count
  - **ssj** – Tests Java performance in SPECpower
  - **MC-friendfeed** – Memcached: “Facebook”-like app
  - **MC-microblog** – Memcached: “Twitter”-like app
I$ misses – signature correlation

Strong correlation between misses & signatures
RDIP practical design

• Summary of takeaways from sensitivity studies:
  – RAS size for signature generation ➔ 4 top entries
  – Miss Table ➔ 4K entries (4-way associative)
  – Entry size ➔ 16B (compaction technique [Ferdman ‘11])
    • Max. 27 misses

Total Hardware Overhead = 64kB

Please see the paper for a detailed analysis
Coverage and erroneous prefetches

Coverage: PIF ~ RDIP > N2L
Erroneous prefetches: PIF > N2L > RDIP
RDIP achieves 98% of the performance of PIF, with 3X storage reduction.

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

• I$ misses – program contexts - RAS signatures
• RDIP performs comparably to PIF with 3X storage reduction
Thank You!

Questions?