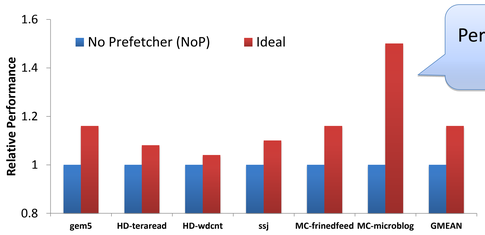
RDIP: RAS-Directed Instruction Prefetching

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.



Performance gains of up to 50% and 16% on average possible with an ideal I$.

Constraints on I$ hit-latency and cycle time do not allow for increasing cache capacity → Instruction prefetching

2. Why another instruction prefetcher?

Next-2-line (N2L)	Proactive Instruction Fetch (PIF) [Ferdman '11]
Industry standard	State-of-the-art academic proposal
+ Low overhead, implementability	+ Best performance
- Modest performance gains	- High overhead (> 200kB), design complexity

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

Design steps	Challenges
1. Use RAS signatures to represent program contexts	Accurate representation
2. Map I$ misses to signature in *Miss Table*	Minimize storage
3. Prefetch on next occurrence of signature	Timely prefetching

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	RAS contents	RAS signature
A:funcX{	A	(A)0
B:funcY{	B A	(A⊕B)0
	~~B~~ A	(A⊕B)1
}		
....	C A	(A⊕C)0
C:funcY{	~~C~~ A	(A⊕C)1

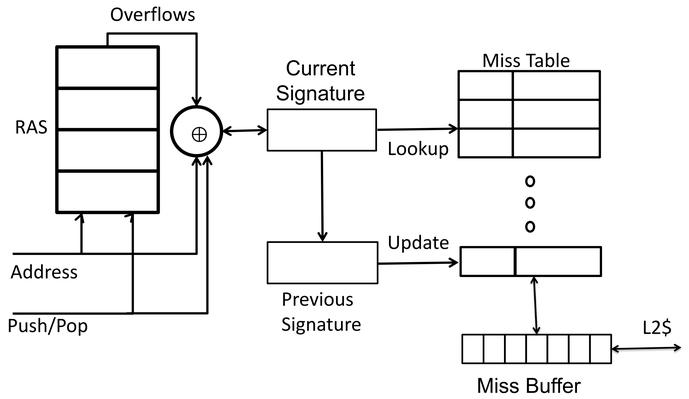
- Same function called from different locations has different signatures
- Large functions can be broken down into multiple contexts

6. Timely prefetching

Mapping I$ misses with corresponding signature → late prefetches.

Mapping I$ misses with preceding signature → timely prefetches

7. RDIP hardware summary



8. RDIP practical design

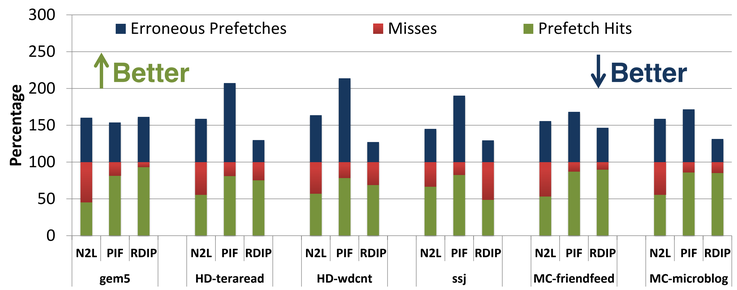
- RAS size for signature generation → 4 top entries
- Miss Table → 4k entries (4-way associative)
- Entry size → 16B (compaction technique [Ferdman '11])

Total storage overhead: 64kB

9. Simulation Methodology

- **gem5** full system simulator
- Core parameters:
  - **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 search 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

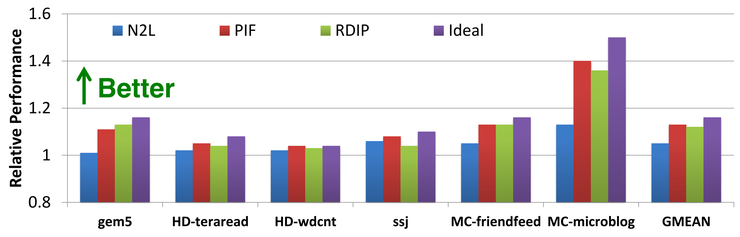
10. Coverage and erroneous prefetches



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

Coverage: PIF ~ RDIP > N2L
Erroneous prefetches: PIF > N2L > RDIP

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.

This work is partially supported by the National Science Foundation under grant CCF-0815457.