

# Introspective computing for Prefetching

(Class project for CS252, Spring 2000)

**Andrew Y. Ng and Eric Xing**

{[fang,epxing](mailto:fang,epxing}@cs.berkeley.edu)}@cs.berkeley.edu

UC Berkeley

# **Introspective computing**

---

---

- Secondary processor ( may be DSP or GP ) to “help” the primary processor.
- Feedback-driven execution to optimize performance. i.e. Prefetching
- Allows far more ambitious control algorithms than hardware prefetchers.

# **Introspective computing**

---

---

## Advantages:

- Gives speed ups without relying on extracting more parallelism. Also Binary compatible.
- Scales up well: Can in principle be applied to multiprocessors, etc.
- Multiple secondary processors also possible.

## Disadvantages:

- Silicon can also be used for other things (e.g. multiprocessors, larger cache)

# The problem

---

---

Prefetching:

- Fetch data into cache to reduce processor stall time.
- Focus on L1 data cache.

Architecture:

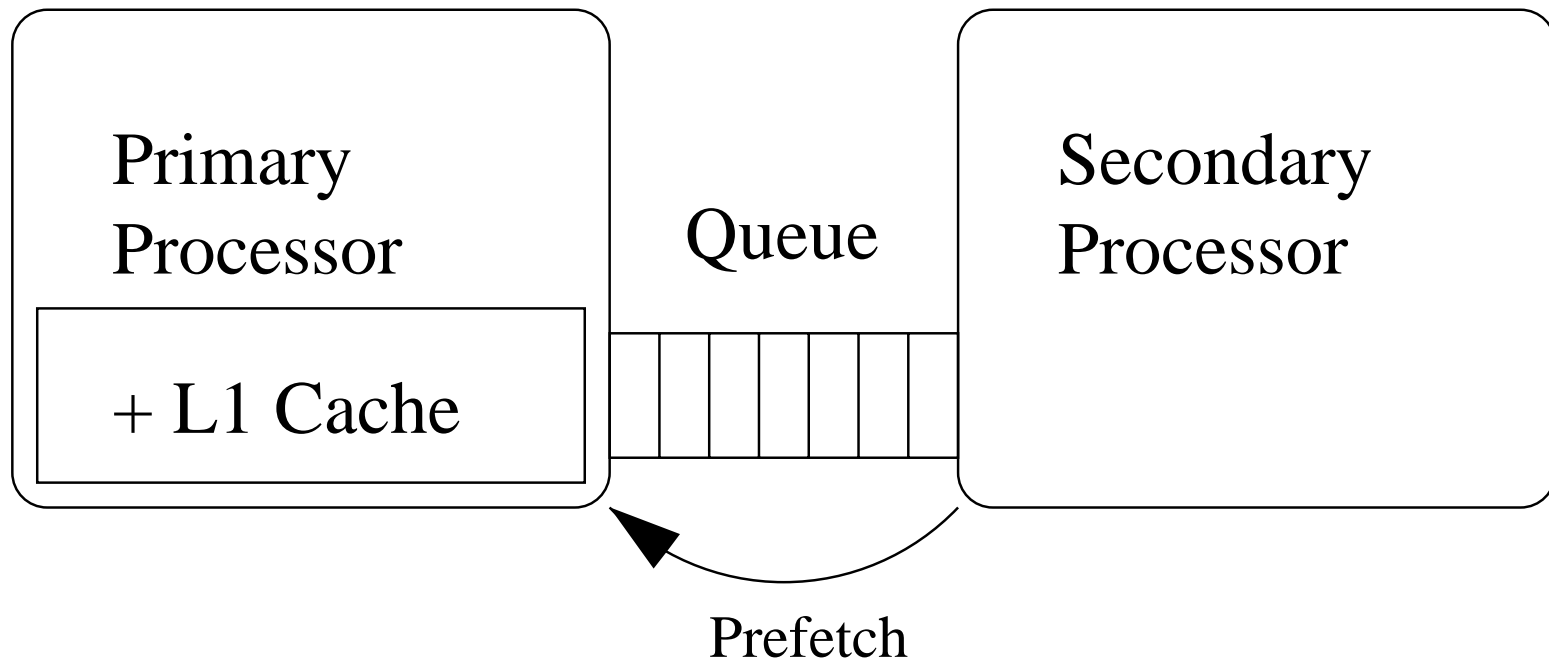
- Primary processor runs program.
- Secondary processor monitors execution, sends prefetch addresses.

# Architecture

---

---

- What is the mechanism by which secondary processor monitors execution? We chose stream of miss addresses.



# Algorithms

---

---

- Given stream of miss addresses, need to predict what the next miss will be.
- Many standard adaptive methods/machine learning algorithms applicable!
- e.g. Reinforcement learning in MDPs:

$$C(s, a) = T(s) + E \left[ \min_a C(s', a) \right] \quad (1)$$

where  $s$  is the current state of the processor+cache,  $a$  is the action we take (e.g. prefetch or not),  $T(s)$  is the time cost of the current state, and  $s'$  is the successor state.

# Real-time constraints

---

---

- Crucial issue: Secondary processor is receiving a stream of instruction misses in *real time*.
- Must respond to them quite quickly (e.g. at most about 100 cycles of processing)
- We found reinforcement learning and other statistical algorithms to be about an order of magnitude too slow.

# Simple predictive algorithm

---

---

After trying several more alternatives, we settled on the following simple algorithm (and minor variants) –

On each cache miss that the secondary processor gets off the queue, it does two things:

1. Update cache-miss statistics
2. (Maybe) do a prefetch.

# Keeping track of statistics

---

---

- Have a huge hash table hashing from cache tags into a table of the cache tags of the next few “likely” misses.
- On a cache miss on tag  $x$ , keep track of the number of occurrences of misses on a few (5) other cache tags within a short time following the miss on  $x$ .

## Issuing prefetches

---

---

- Suppose that, after a cache miss on tag  $x$ , we had observed at least  $n$  times a cache miss on  $y$  shortly after.
- Then the next time we see a miss on  $x$  again, we also prefetch  $y$ .

# Parameters

---

---

- Queue type: Fifo or Lifo? (Both have advantages.)
- What to do if queue is full?
- Speed of secondary processor.
- Size of hash table for miss statistics.
- What to do in event of collision in hash table.
- What to do in event of collision in table keeping track of next-miss tag.

# Experimental details

---

---

Default parameters:

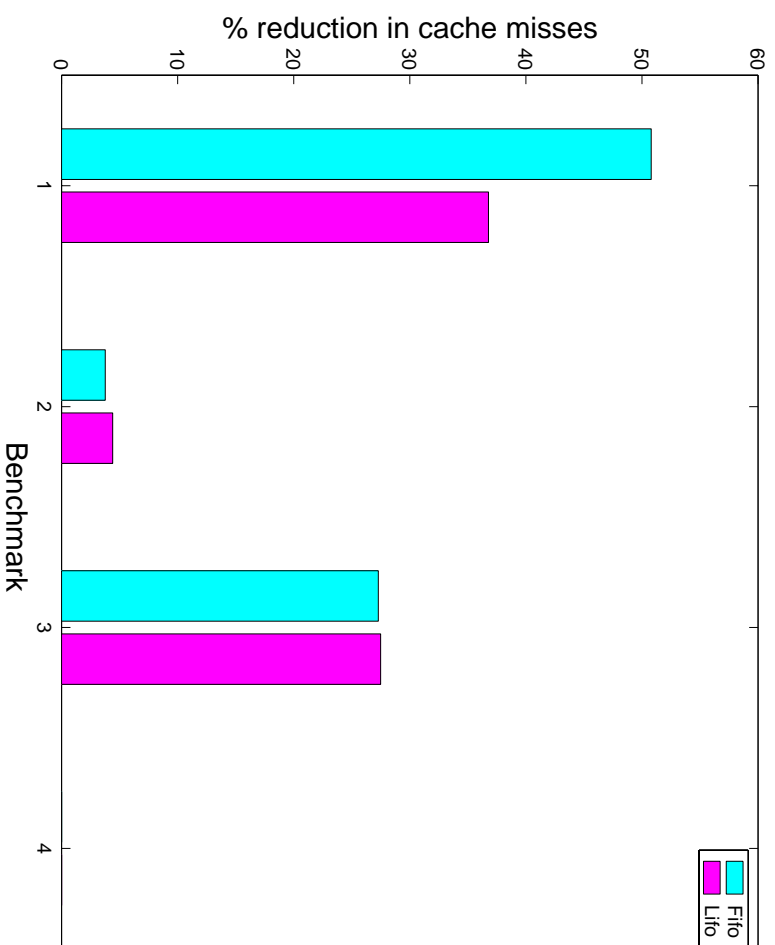
- Used simpleScalar simulator.
- SPEC95 Benchmarks gcc, go, compress, hydro2d.
- Hashsize = 10000.
- Used Queue of size 10.
- Assumed secondary processor as fast as primary processor.
- In event of hash etc. conflicts, information on the new cache tag is thrown away.

# Queue type: Fifo or Lifo?

---

---

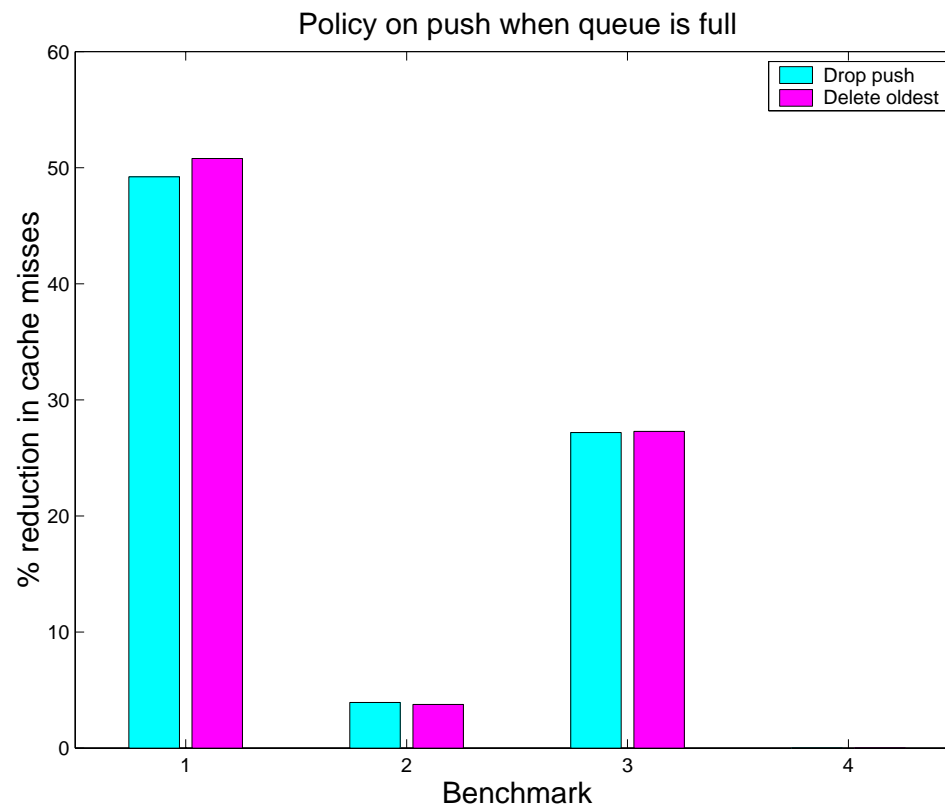
Fifo processes data in order. Lifo more urgently gets to recent misses first.



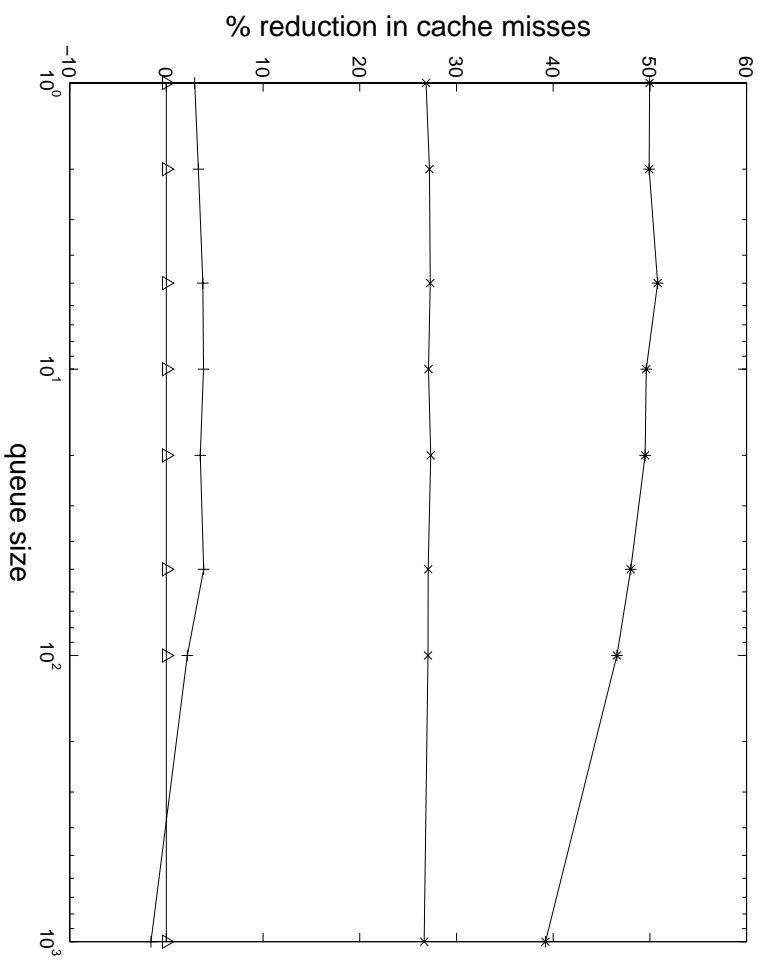
# What to do when queue full?

---

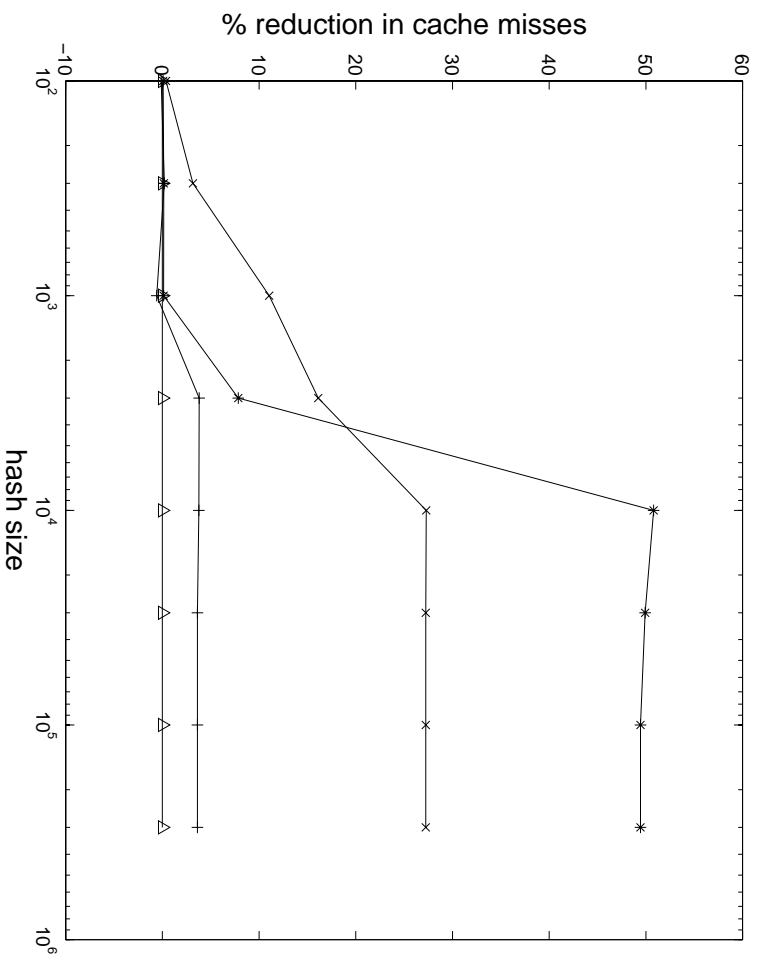
---



# Queue size



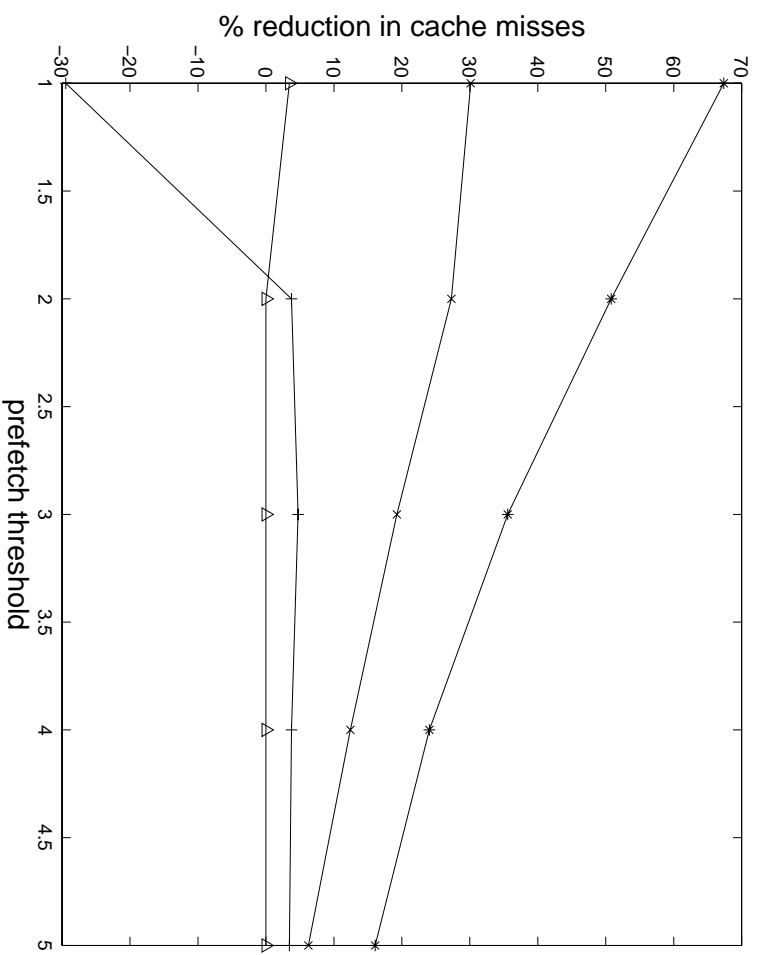
# Hashable size



# #misses observed before we prefetch

---

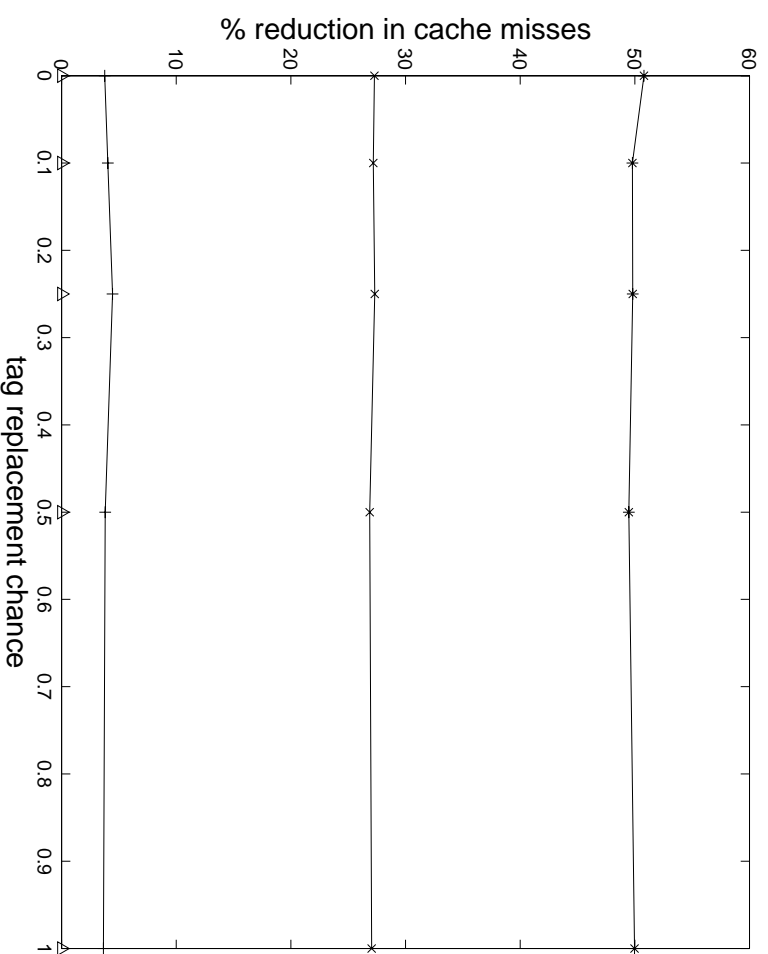
---



# Conflicts in hash table

---

---

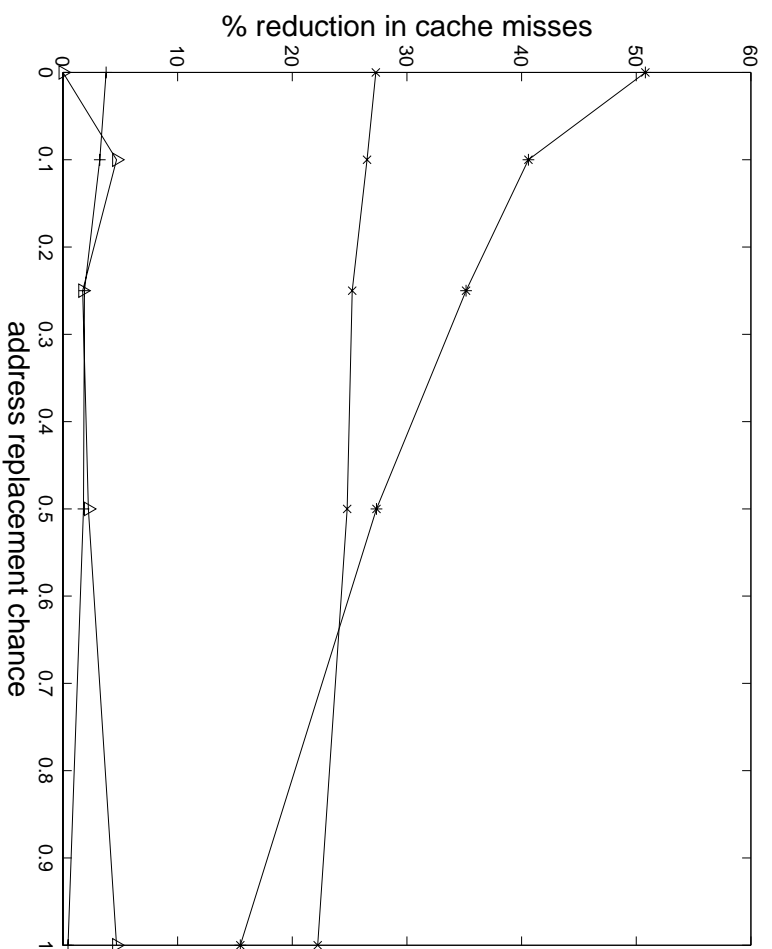


(Similar results over fairly large range of hash table sizes.)

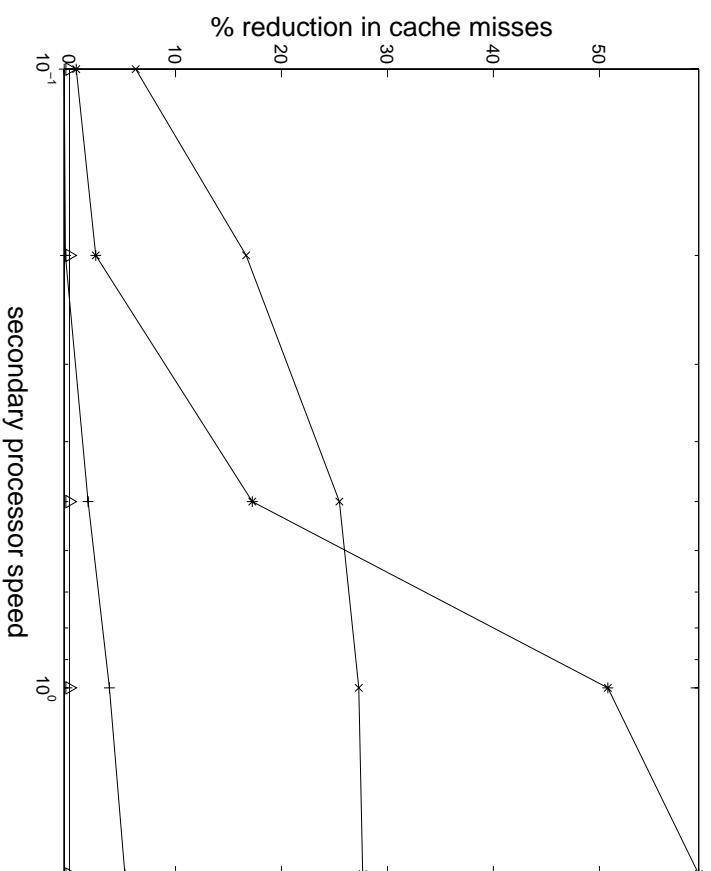
# Conflicts in table storing next-miss address

---

---



# Speed of secondary processor



## Summary and Related issues

---

---

- Up to about about 60% reduction in cache misses possible.
- If the queue is full (i.e. the secondary processor is busy), it is sometimes advantageous to (stochastically) skip some of the updating of the statistics, to focus on issuing appropriate prefetching instructions.
- Introspective computing can similarly be used to improved branch prediction (but hard!)
- *Parallelizes* easily: Can have multiple secondary processors.
- Specialized instructions for secondary processor?