Introsp ective computing for Prefetching

UC Berkeley

Andrew Y. Ng and Eric Xing

UC Berkeley

Class project for CS252, Spring 2000

Introsp ective computing for Prefetching
Introsp ective computing

Prefetchers.

Allows far more ambitious control algorithms than hardware prefetchers.

Prefetching Feedback-driven execution to optimize performance. i.e.

Prefetching. Primary processor.

Secondary processor (may be DSP or GP) to “help” the

Introsp ective computing
Advantages:

- Gives speed ups without relying on extracting more parallelism.
- Also Binary compatible.
- Gives speed ups with well: Can in principle be applied to multiprocessors, also Binary compatible.

Disadvantages:

- Silicon can also be used for other things (e.g., multiprocessors, etc.)
- Multiple secondary processors also possible.
- Scales up well: Can in principle be applied to multiprocessors.

Introsp ective computing
The problem

Prefetching:
- Fetch data into cache to reduce processor stall time.
- Focus on L1 data cache.
- Primary processor runs program.
- Secondary processor monitors execution, sends prefetch addresses.

Architecture:
- Focus on L1 data cache.
- Primary processor runs program.
- Secondary processor monitors execution, sends prefetch addresses.

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

Diagram:

- Primary Processor
  - + L1 Cache
- Queue
- Secondary Processor
- Prefetch
Given a 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:

\[(v, s) \in \min C \mathbb{E} + (s, t) = (v, s') \mathcal{C}\]

Algorithms
Real-time constraints

- Algorithms to be about an order of magnitude too slow.
- 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.
After trying several more alternatives, we settled on the following simple predictive algorithm:

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

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

- Update cache miss statistics (and minor variants)
- Maybe do a prefetch.

Simple Predictive Algorithm
Keeping track of statistics

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.

On a cache miss on tag x, keep track of the number of occurrences of misses on the next few "likely" misses.

Have a huge hash table hashing from cache tags into a table of

Keeping track of statistics
Then the next time we see a miss on $x$ again, we also prefetch $y$. If the next time we see a cache miss on $y$ shortly after, we prefetch it as well.

Suppose there is another cache miss on the tag $x$, we had observed at least $n$ times a cache miss on $y$ shortly after.

---

Issuing Prefetches

What to do in event of collision in table keeping track of next-miss tag.

What to do in event of collision in hash table.

What to do if queue is full?

Queue type: FIFO or LIFO? (Both have advantages).

Parameters

Speed of secondary processor.

Size of hash table for miss statistics.
Experimental details

- In event of hash etc., conflicts, information on the new cache tag is thrown away.
- Assumed secondary processor as fast as primary processor.
- Used Queue of size 10.
- Hashsize = 10000.
- SPEC95 Benchmarks 6c, 6g, compress, hydror2.
- Used SimpleScalar Simulator.

Default parameters:
Queue type: FIFO or LIFO?

FIFO processes data in order. LIFO more urgently gets to recent misses first.
What to do when queue full?

Policy on push when queue is full

Benchmark

% reduction in cache misses

Drop push
Delete oldest
Queue size

% reduction in cache misses
Hashtable size

% reduction in cache misses
# misses observed before we prefetch

---

% reduction in cache misses vs prefetch threshold.
Similar results over fairly large range of hash table sizes.

Conflicts in hash table
address replacement chance
% reduction in cache misses
Speed of secondary processor

% reduction in cache misses

Secondary processor speed

Speed of secondary processor
Specialized instructions for secondary processors

Parallelizes easily: Can have multiple secondary processors.

Branch prediction (but hard)

Introspective computing can similarly be used to improved prefetching instructions.

Prefetching instructions.

Updating of the statistics, to focus on issuing appropriate sometimes advantageous to (stochastically skip some of the)

If the queue is full (i.e. the secondary processor is busy), it is

Up to about about 60% reduction in cache misses possible.

_____________________________________

Summary and Related issues