Probabilistic Cache Filtering

***Yoav Etsion* and Dror G. Feitelson**

School of Computer Science and Engineering
Hebrew University
Jerusalem, Israel

A Case for More Efficient Caches

- CPUs get more and more cache dependant
  - Growing gap between CPU and memory
  - Growing popularity of multi-core chips
- But larger caches consume more power

- Need to make caches more efficient:
  - Low-latency, low-power, direct-mapped cache for frequently used blocks
  - Bypass buffer for transient blocks
- But how can we classify them at runtime?

Popularity vs. Temporal Locality

- Latter is a combination of two properties:
  1. Some addresses are more popular than others
  2. Accesses to the same block are batched together rather than being random
- Specifically, accesses to seldom referenced blocks are also batched together
- Can we identify the frequently used blocks?

- The skewed popularity can be visualized using ***mass-count disparity*** plots

The Mass-Count Disparity Phenomenon

- Given a finite sample space, we assign a mass function **M(s)** for each sample.
- Mass and Count are two distinct distributions defined over the sample space:

Fc(m) = fraction of samples with mass **M(s)=m**

- The ***count*** distribution

Fm(m) = fraction of total weight composed of samples with mass **M(s)=m**

- The ***mass*** distribution

The Mass-Count Disparity Phenomenon

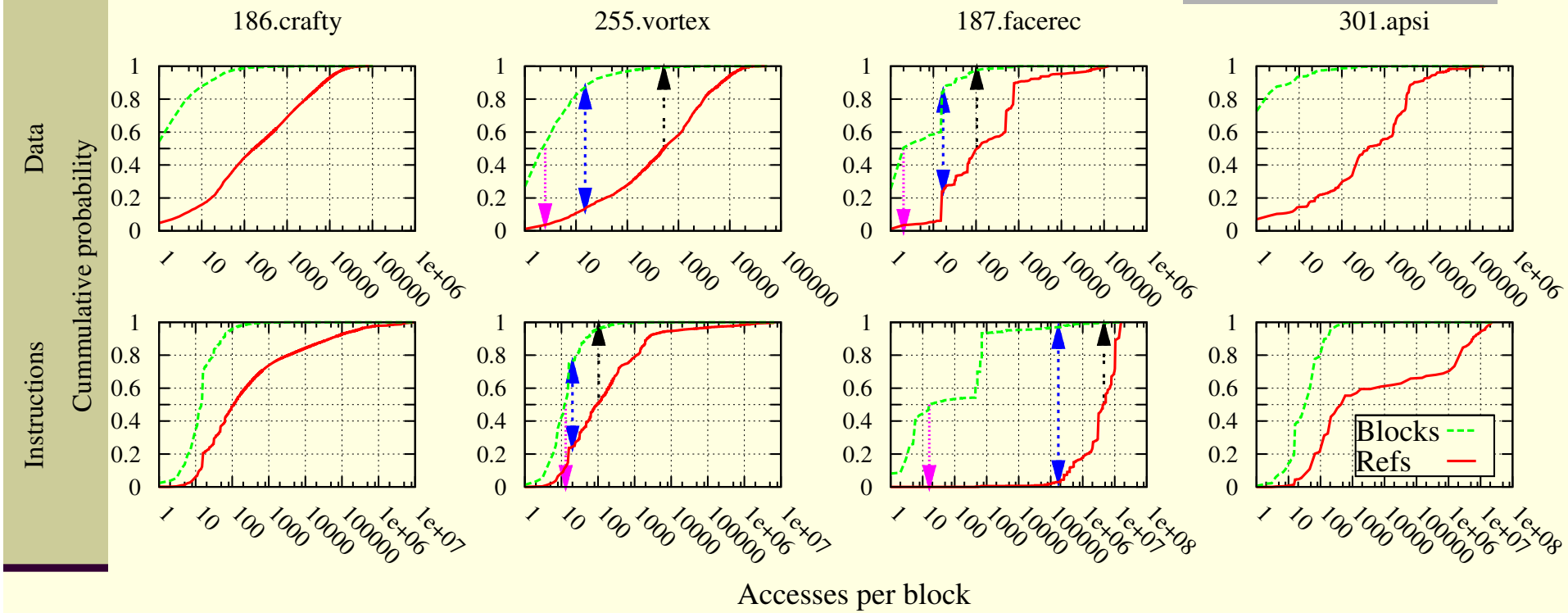
- The disparity:
  - Small masses are much more common , and the different masses span a large range
- The result:

  A small fraction of the "heaviest" samples account for the majority of the mass
- Therefore, randomly selecting a:
  - ***Sample:*** likely get one with a small mass
  - ***Mass unit***: likely belongs to a "heavy" sample

Mass-Count Disparity: Examples

- Economics: Wealth distribution
  - **`M(s)=person's wealth`**
  - Most of the wealth belongs to a tiny fraction of the population
- Computers: Disk space
  - **`M(s)=file's size`**
  - Most files are very small
  - Few large files consume majority of space
- Also apparent in per-block reference counts

Mass-Count Disparity Plots



- Cache residency lengths capture program phases
- Joint-ratio: formalization of the 80/20 rule:
  - Single point where the sum of the two CDFs is 1

Mass-Count Disparity in L1 Caches

- Sample ***space***: mem. blocks in the cache (lines)
- Sample ***mass***: cache residency length
  - number of accesses while in the cache

So what happens if we randomly sample…

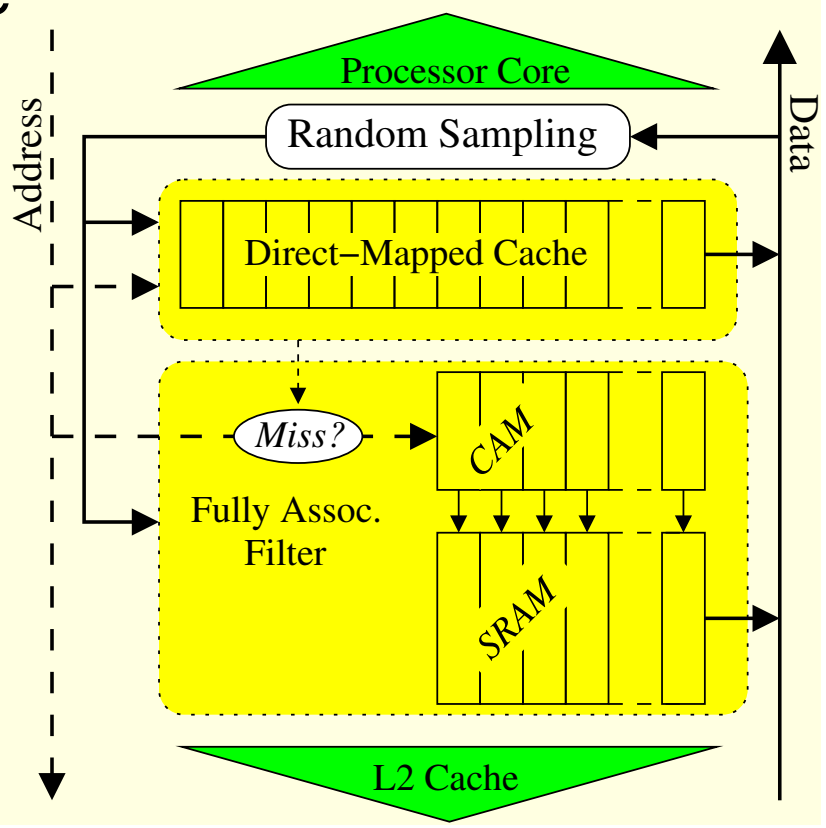
- *Mem.* ***Blocks***: likely find a short residency
  - That is why random eviction works fairly well
- *Mem.* ***References***: likely find a long residency
  - Can be used to identify frequently used blocks

Design of a Random Sampling Cache

- Direct-mapped cache + fully-associative filter
- Policy: flip a biased coin on each memory reference
  - No need to save any state or reuse information!

Results:

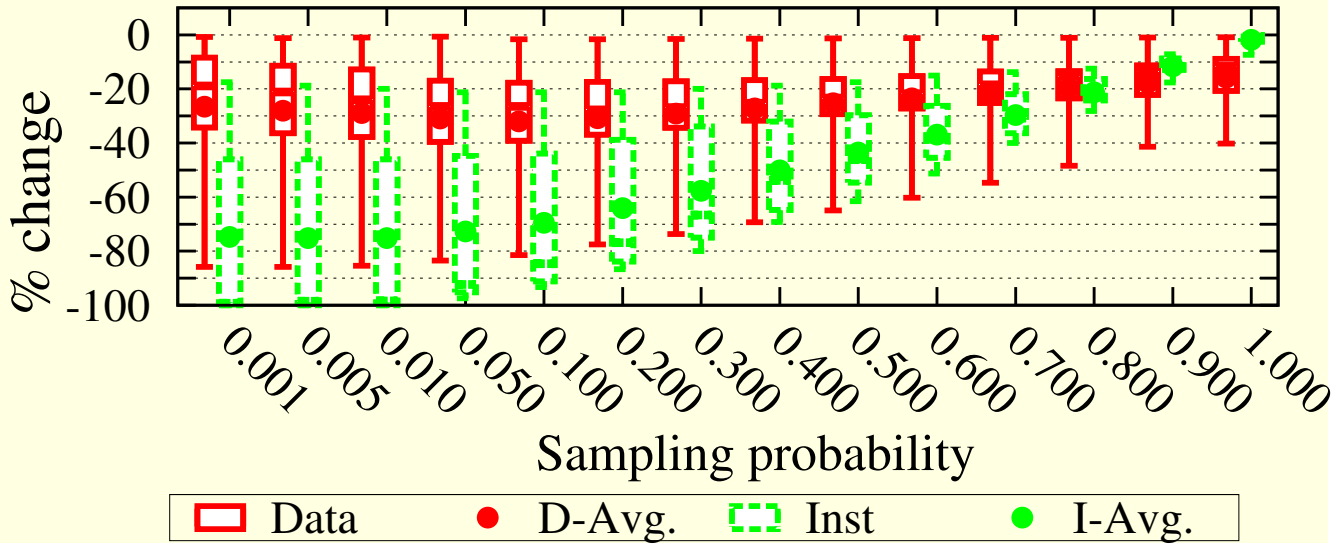
1. long residencies end up in the cache
2. *Short residencies remain in the filter*



Searching for the Right Biased Coin

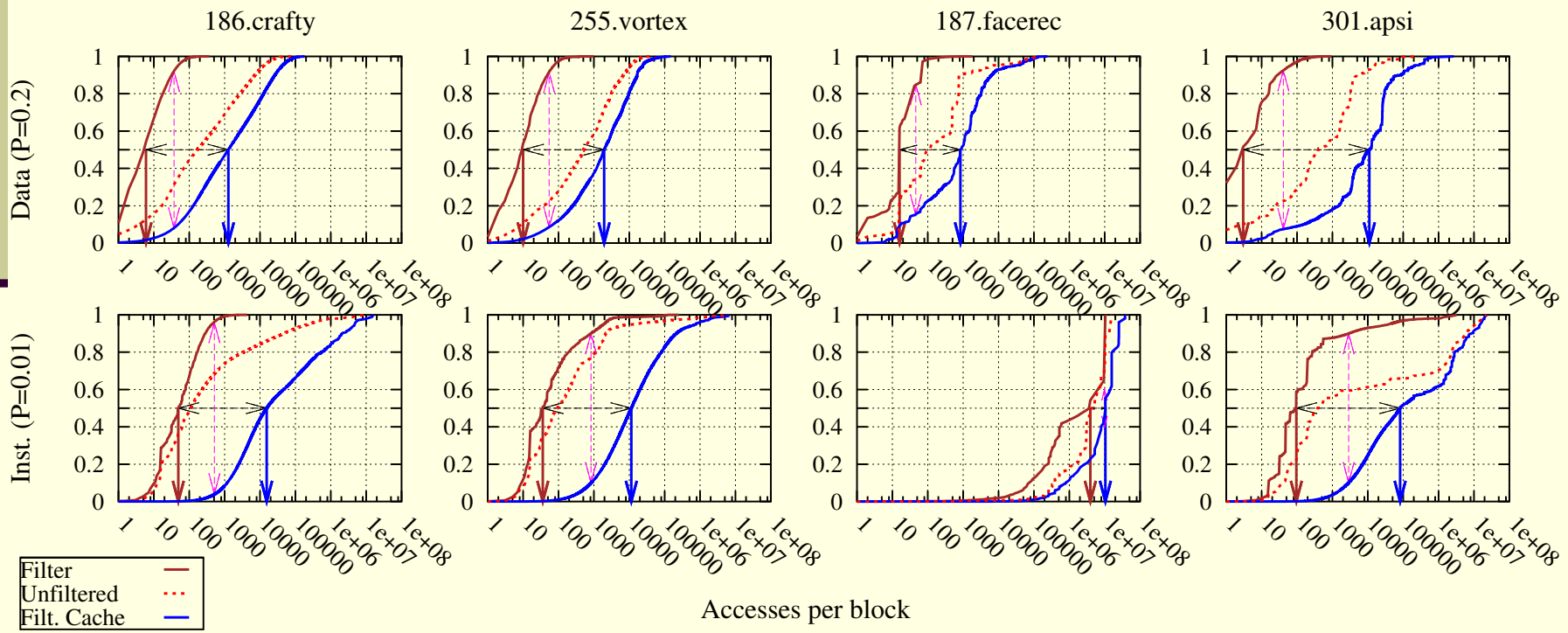
- Find the probability that minimizes the miss-rate
- High probability swamps the cache
  - Low probability swamps the filter
- Still, constant selection probabilities are sufficient:
  - Data miss-rate reduced by 30% for P=0.2
  - Inst. miss-rate reduced by >70% for P=0.01

Cache: 16K
Filter: 4K



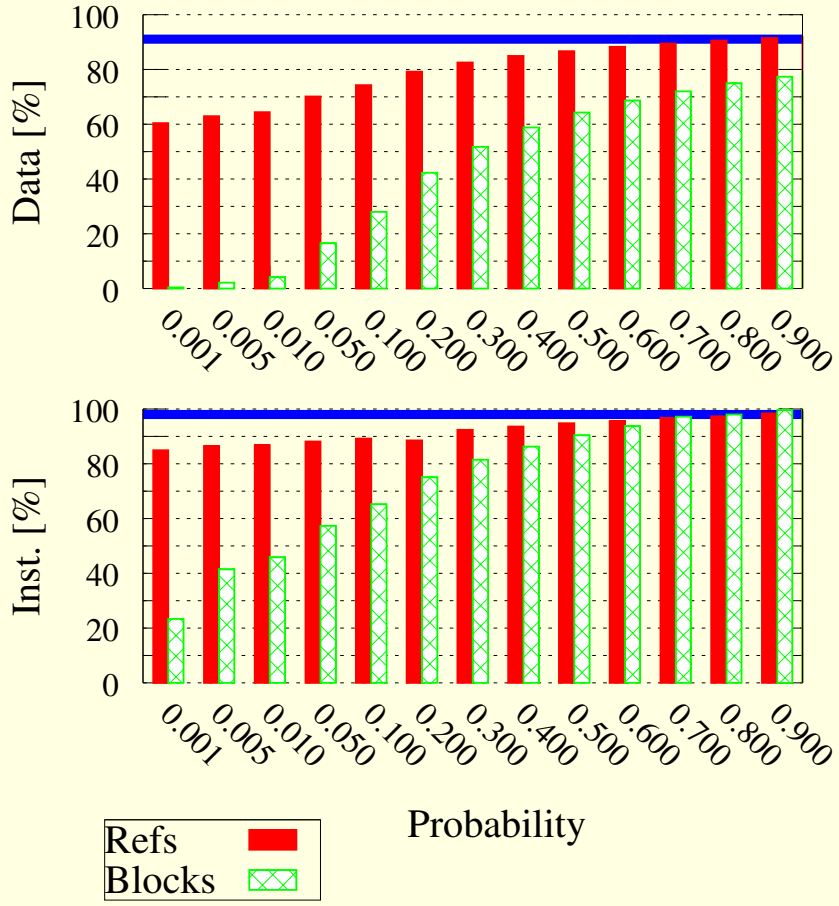
Cache vs. Filter: Reference Distributions

- How was the original distribution split?
  - Medians are 2-3 orders of magnitude apart
  - False-* estimates: data at 14%, inst. at 17%



Cache vs. Filter: Reference/Block Count

- blocks promoted vs. references served
  - (16K + 4K filter)
- Lower probabilities yield longer average residency length
- Chosen probabilities:
  - Data: 45% / 80%
  - Inst.: 45% / 85%



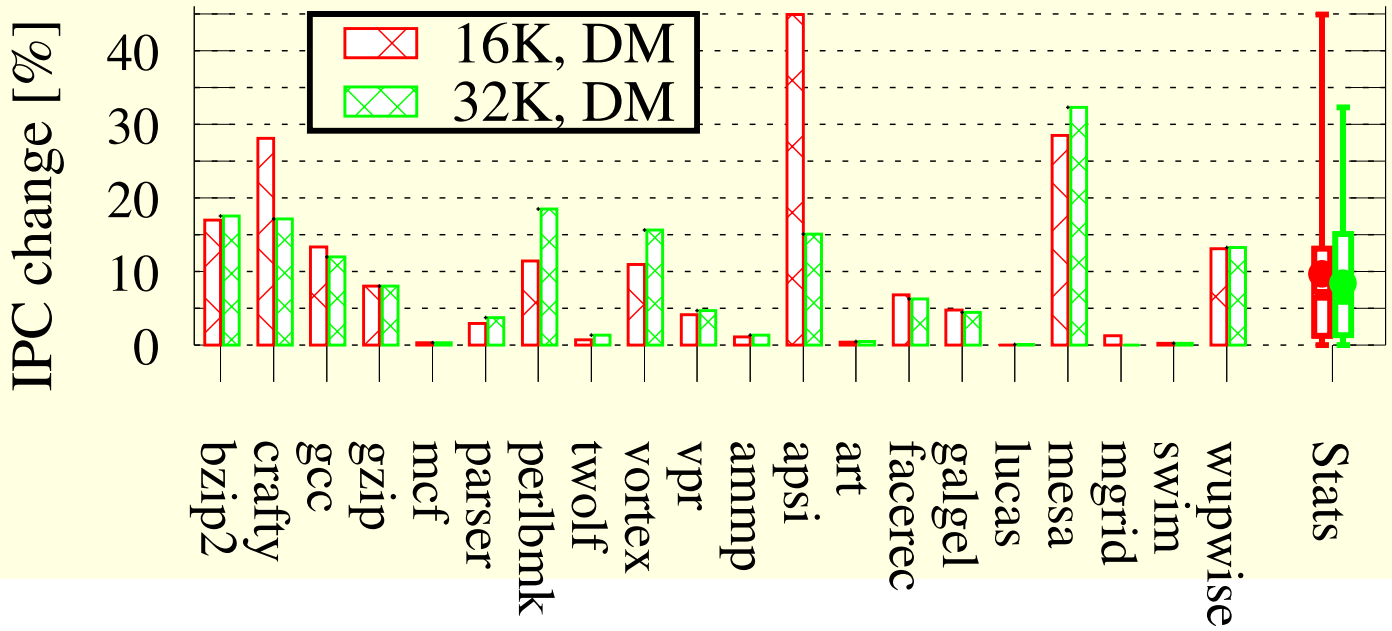
Random Sampling: Where are we?

- Reduced overall miss-rate
  - 30% for data, >70% for instructions
- Majority of references serviced by the cache, majority of blocks remain in the filter
  - Cache services 80% - 85% of references…
  - …with only 45% of the blocks
- Result: most references enjoy direct-mapped cache's speed and low-power, less conflicts

- But what is the final power/performance score?

Effect on Performance: Benchmarks

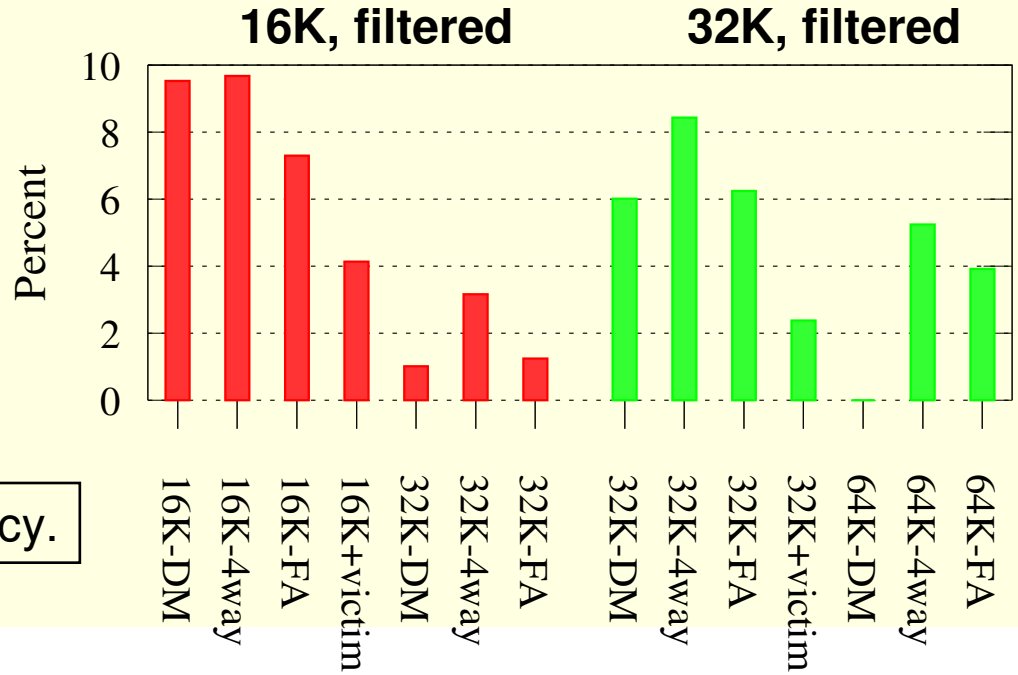
- SimpleScalar, 512K L2, 8 wide processor
  - Balance between sensitivity to data and inst. latencies.
- DM Cache latency: 1 cycle; Filter: extra 3 cycles
- Our base case is a common ***4-way*** cache
- IPC Improvement: ~9% on average for both 16K/32K



Effects on Performance: Caches

- We compare the random sampling cache to common cache structures
- Outperforms a ***4-way cache double its size!***
  - Interesting: DM's low-latency compensates for conflict misses

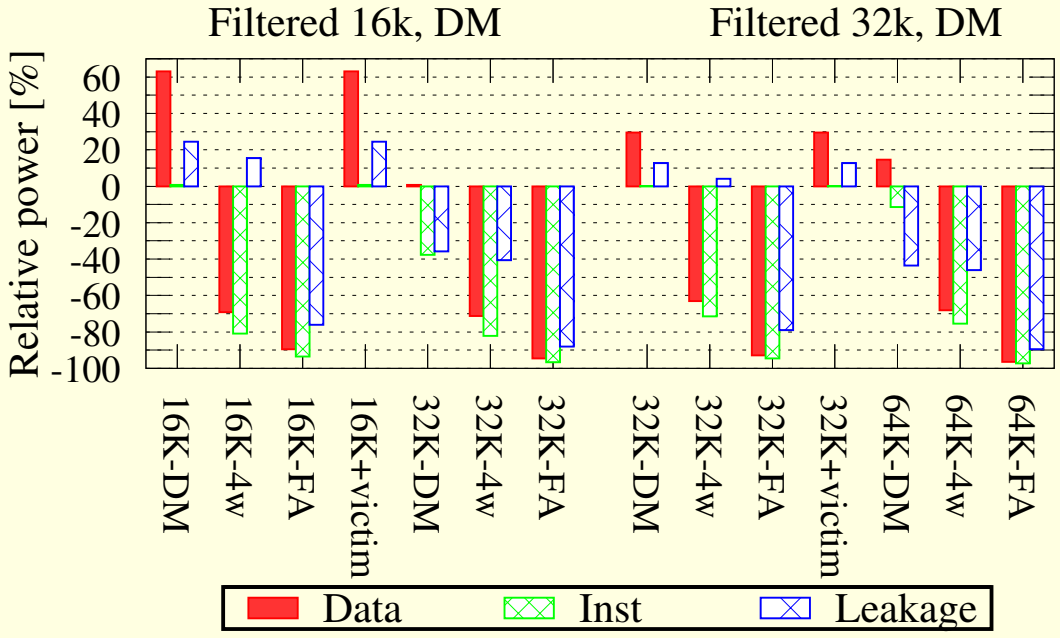
Filtered 16k,DM Filtered 32k,DM



Associative Latency: 2cy.

Effect on Power Consumption

- Direct mapped cache reduces dynamic power, but the filter adds ~25% more leakage
- Over same size: ***60%-80% less dyn. power***
- Over double size: ***40% less leakage power***



Related Work

- Use auxiliary buffer for other locality types:
  - Victim buffer *[Jouppi'90]*
  - Don't cache problematic refs. *[Tyson+'95]*
  - Spatial vs. Temporal caches *[Gonzalez+'95]*
  - Only cache blocks that won't be evicted before the next access *[Johnson+'97][Karlsson+'01][Jalminger+'03]*
  - *All require maintaining per-block information*
- Use sampling: generate every 10th trace when using a trace-cache *[Behar+'05]*
  - Mass-Count disparity is also apparent in traces…

Conclusions

- The Mass-Count disparity phenomenon can be leveraged for caching policies
  - Both insertion and eviction
- Using random sampling can effectively identify frequently used blocks
  - Eliminates the need to maintain state information
- The proposed cache design offers:
  - ~9% IPC improvement over 4-way cache
  - 60%-80% less dynamic power
- Overall, outperforms caches double its size both in terms of IPC and power consumption

Thank You