CACHE POLICIES AND INTERCONNECTS

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

Overview

- Upcoming deadline
  - Feb. 3rd: project group formation
  - Note: email me once you form a group
- This lecture
  - Cache replacement policies
  - Cache partitioning
  - Content aware optimizations
  - Cache interconnect optimizations
  - Encoding based optimizations

Recall: Cache Power Optimization

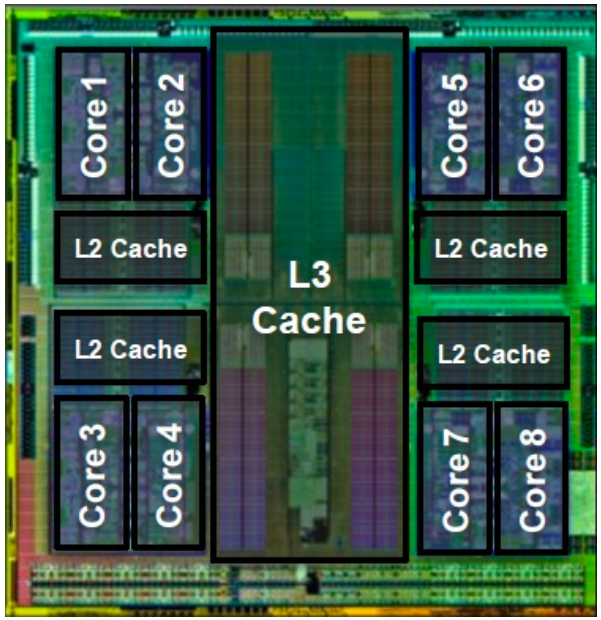
- Caches are power and performance critical components

- Performance
  - Bridging the CPU-Mem gap

- Static power
  - Large number of leaky cells

- Dynamic power
  - Access through long interconnects

Example: FX Processors

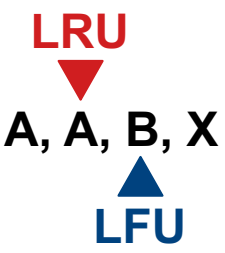


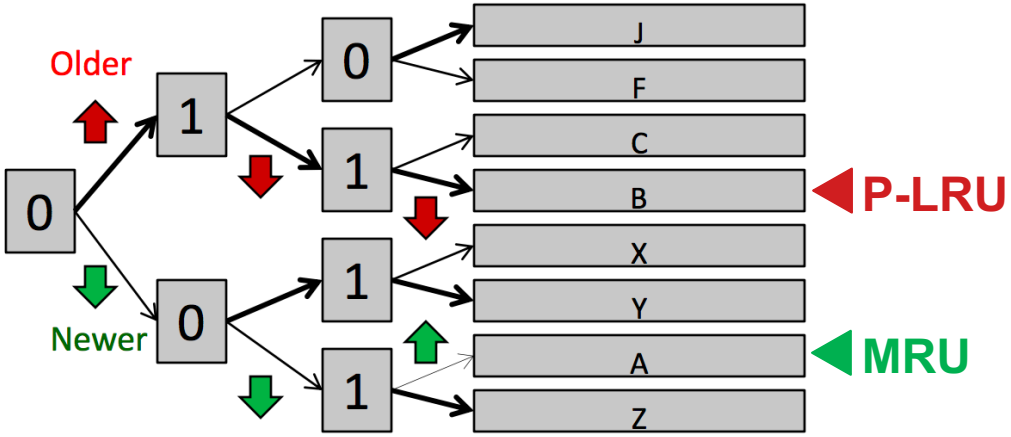
[source: AMD]

Replacement Policies

Basic Replacement Policies

- Least Recently Used (LRU)
- Least Frequently Used (LFU)
- Not Recently Used (NRU)
  - every block has a bit that is reset to 0 upon touch
  - a block with its bit set to 1 is evicted
  - if no block has a 1, make every bit 1
- Practical pseudo-LRU





Common Issues with Basic Policies

- Low hit rate due to cache pollution
  - streaming (no reuse)
    - A-B-C-D-E-F-G-H-I-…

  - thrashing (distant reuse)
    - A-B-C-A-B-C-A-B-C-…

- A large fraction of the cache is useless – blocks that have serviced their last hit and are on the slow walk from MRU to LRU

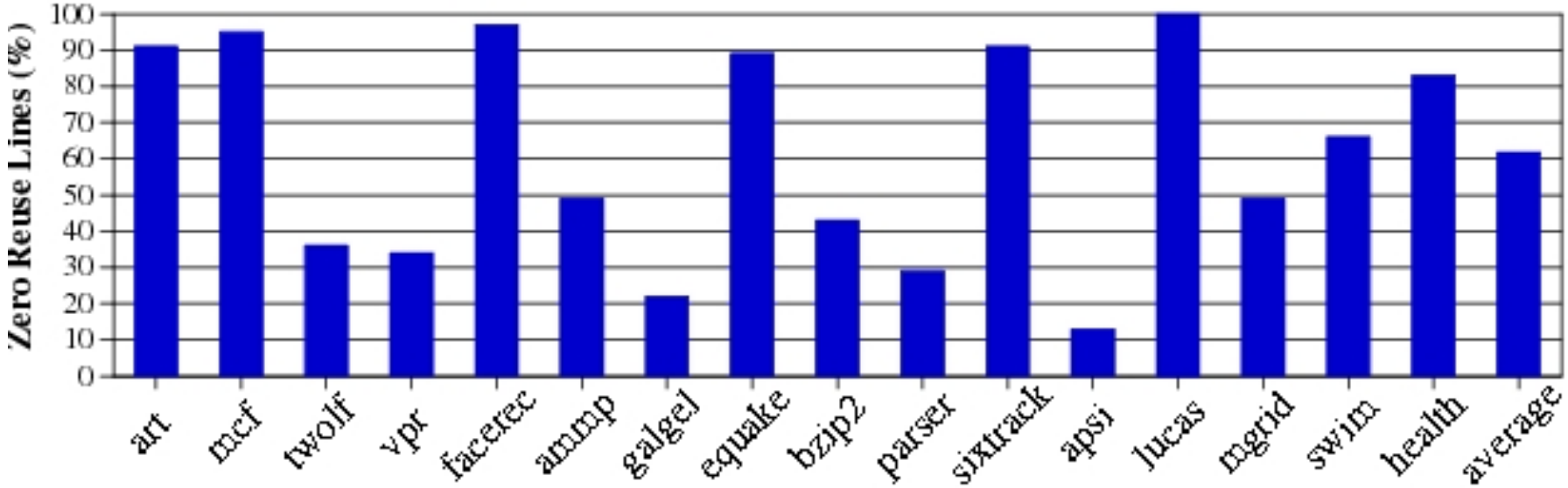
Basic Cache Policies

- Insertion
  - Where is incoming line placed in replacement list?
- Promotion
  - When a block is touched, it can be promoted up the priority list in one of many ways
- Victim selection
  - Which line to replace for incoming line? (not necessarily the tail of the list)

Simple changes to these policies can greatly improve cache performance for memory-intensive workloads

Inefficiency of Basic Policies

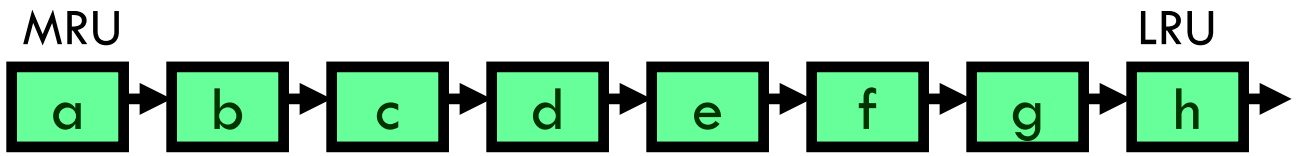
- About 60% of the cache blocks may be dead on arrival (DoA)



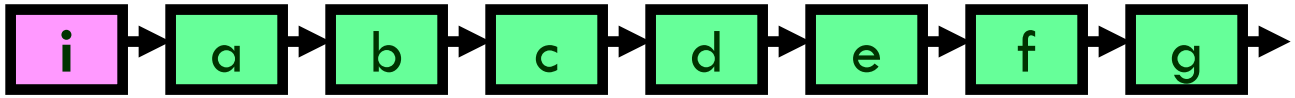
[Qureshi'07]

Adaptive Insertion Policies

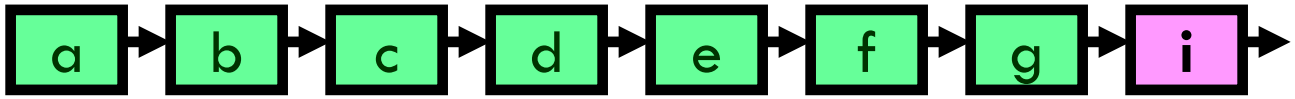
- MIP: MRU insertion policy (baseline)
- LIP: LRU insertion policy



Traditional LRU places 'i' in MRU position.



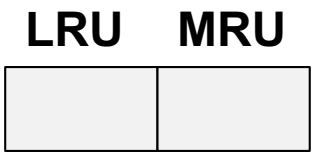
LIP places 'i' in LRU position; with the first touch it becomes MRU.



[Qureshi'07]

Adaptive Insertion Policies

- LIP does not age older blocks
  - A, A, B, C, B, C, B, C, …



- BIP: Bimodal Insertion Policy
  - Let ε = Bimodal throttle parameter

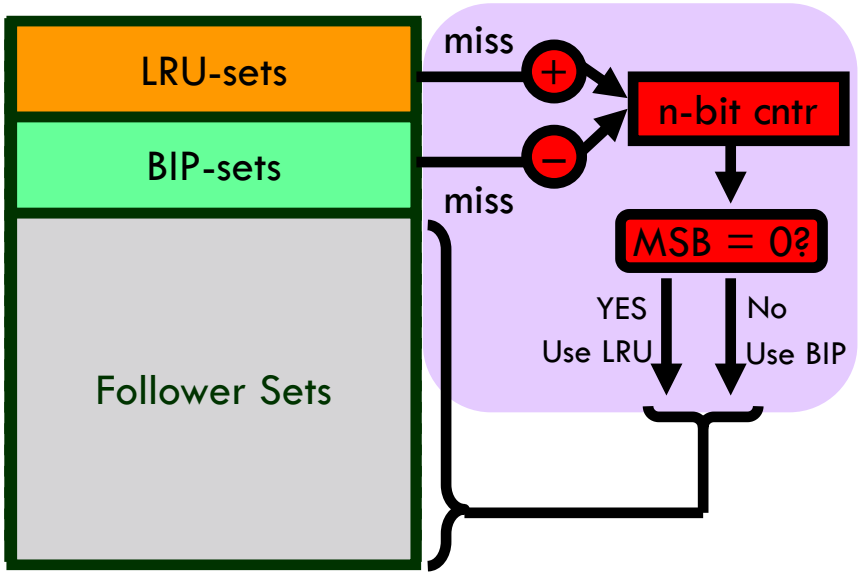
```
if  ( rand() < ε )
      Insert at MRU position;
else
      Insert at LRU position;
```

[Qureshi'07]

Adaptive Insertion Policies

- There are two types of workloads: LRU-friendly or BIP-friendly
- DIP: Dynamic Insertion Policy
  - Set Dueling

Read the paper for more details.

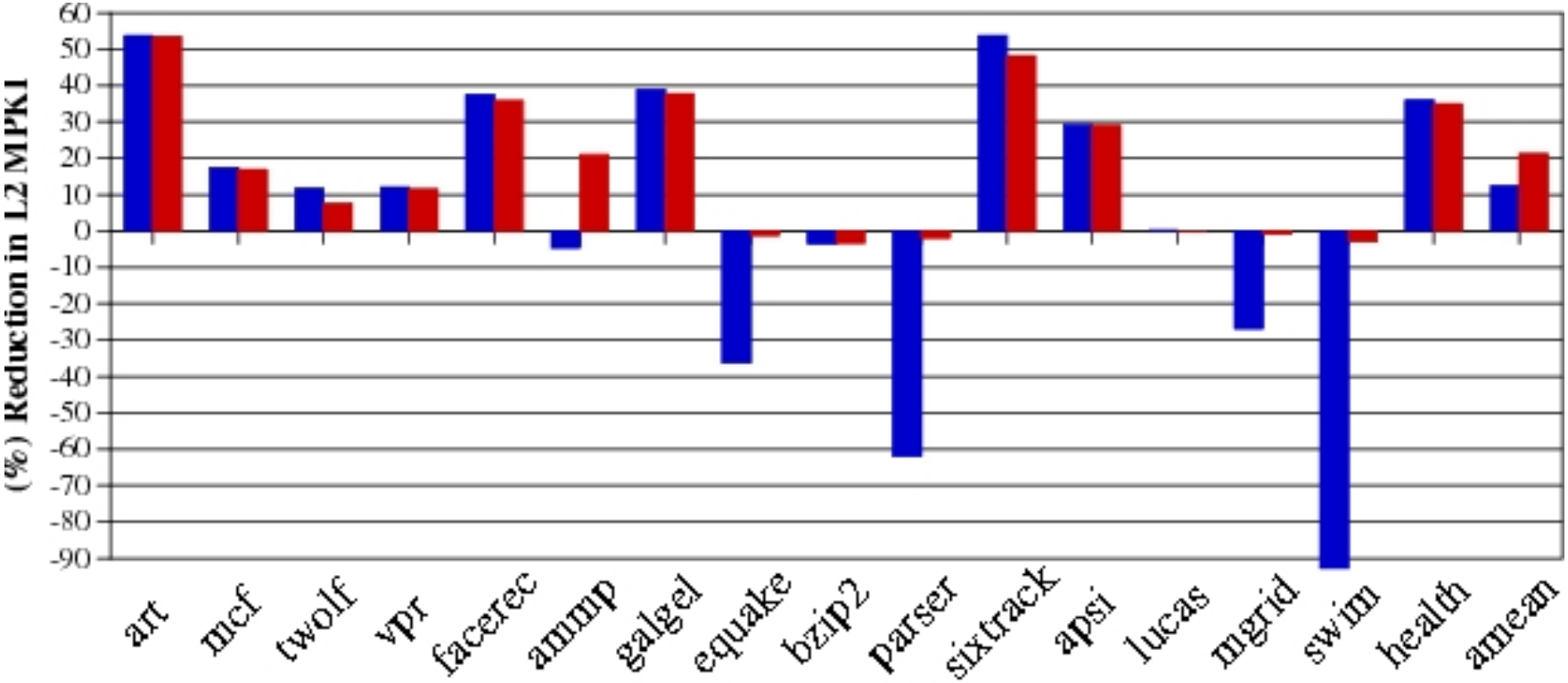


monitor → choose → apply
(using a single counter)

[Qureshi'07]

Adaptive Insertion Policies

- DIP reduces average MPKI by 21% and requires less than two bytes storage overhead



[Qureshi'07]

Re-Reference Interval Prediction

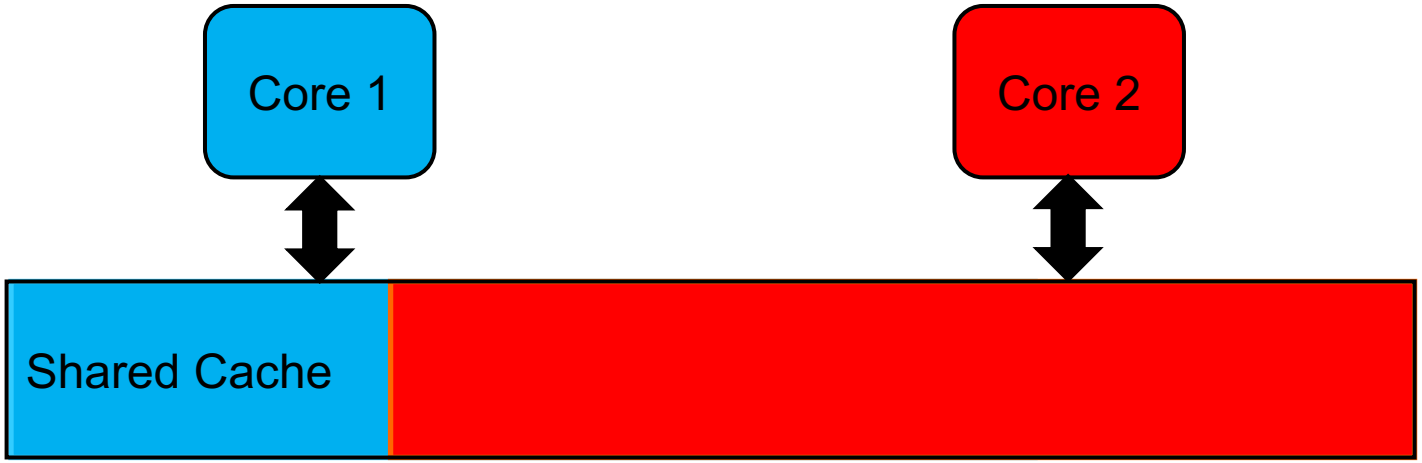
- Goal: high performing scan resistant policy
  - DIP is thrash-resistance
  - LFU is good for recurring scans
- Key idea: insert blocks near the end of the list than at the very end
- Implement with a multi-bit version of NRU
  - zero counter on touch, evict block with max counter, else increment every counter by one

Read the paper for more details.

[Jaleel'10]

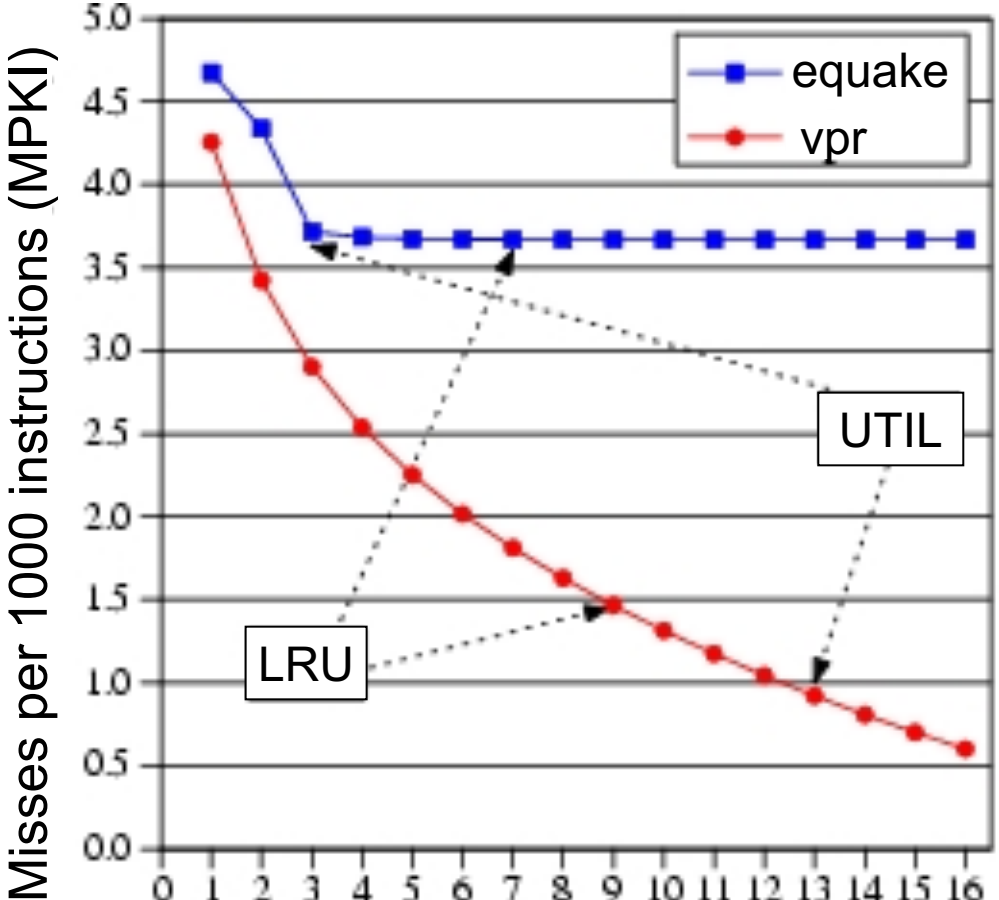
Shared Cache Problems

- A thread's performance may be significantly reduced due to an unfair cache sharing
- Question: how to control cache sharing?
  - Fair cache partitioning [Kim'04]
  - Utility based cache partitioning [Qureshi'06]



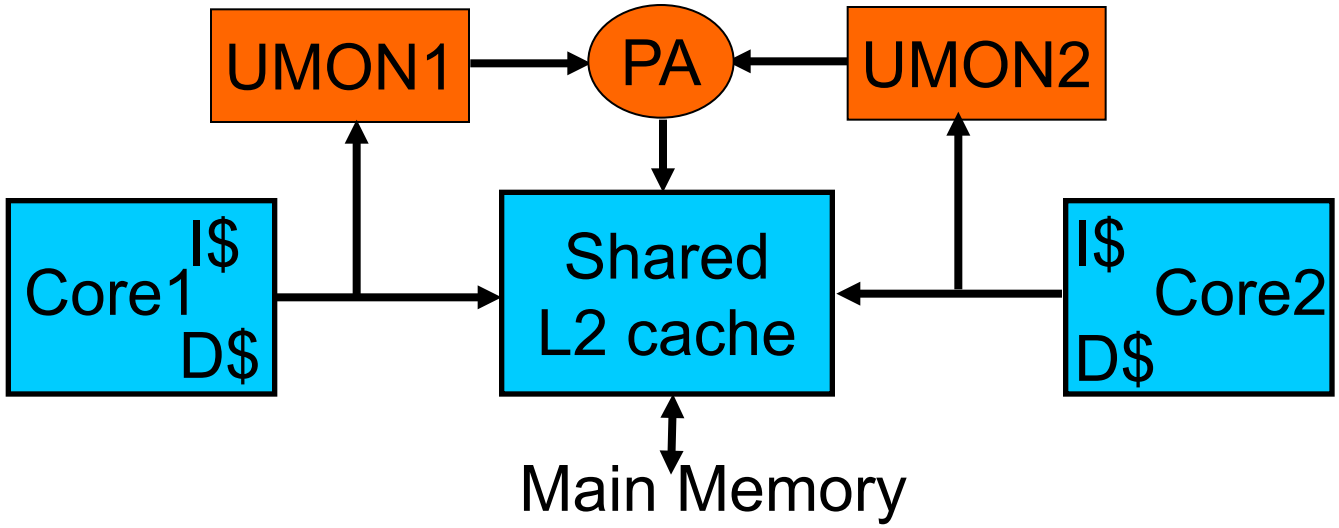
Utility Based Cache Partitioning

- Key idea: give more cache to the application that benefits more from cache



[Qureshi'06]

Utility Based Cache Partitioning



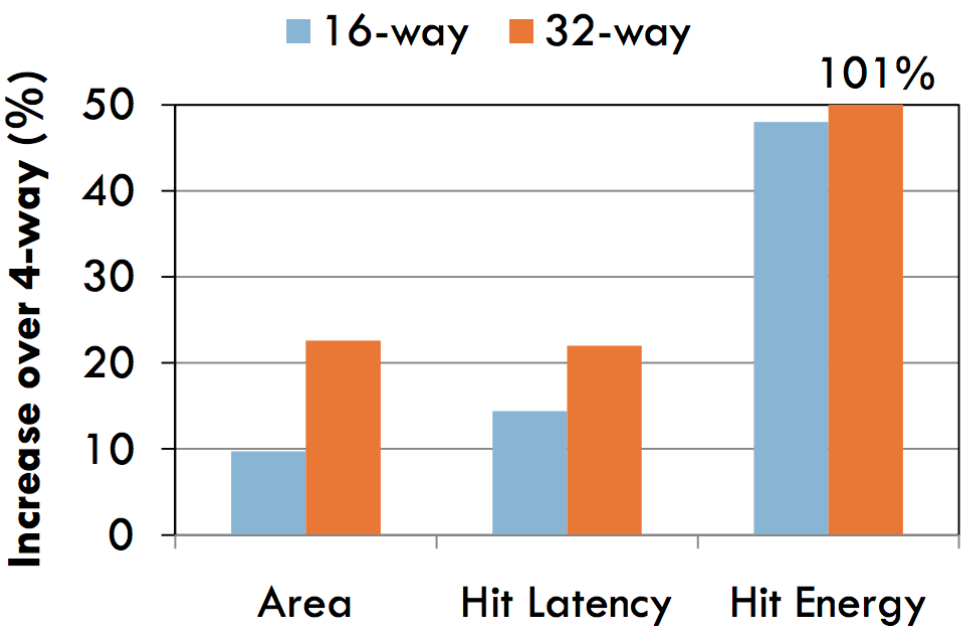
Three components:

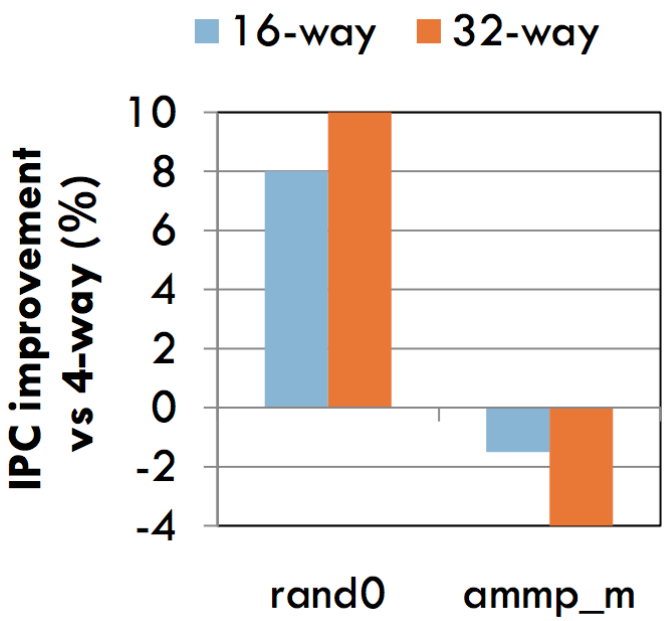
- Utility Monitors (UMON) per core
- Partitioning Algorithm (PA)
- Replacement support to enforce partitions

[Qureshi'06]

Highly Associative Caches

- Last level caches have ~32 ways in multicores
  - Increased energy, latency, and area overheads



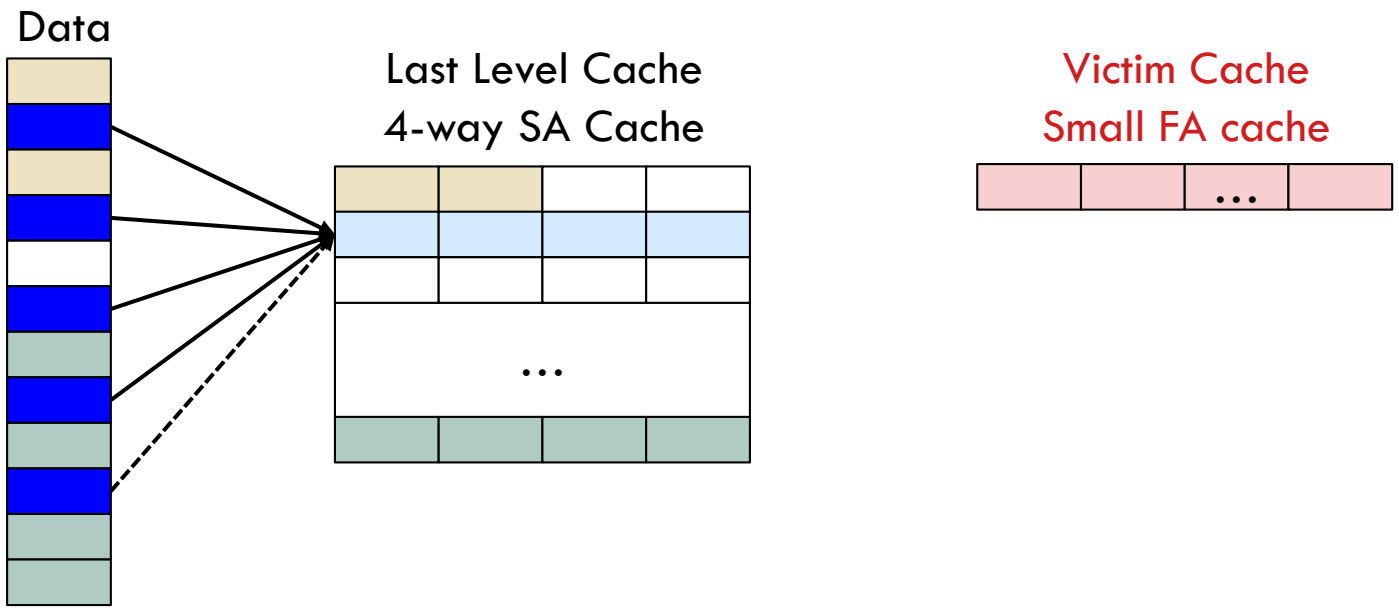


[Sanchez'10]

Recall: Victim Caches

- Goal: to decrease conflict misses using a small FA cache

Can we reduce the hardware overheads?



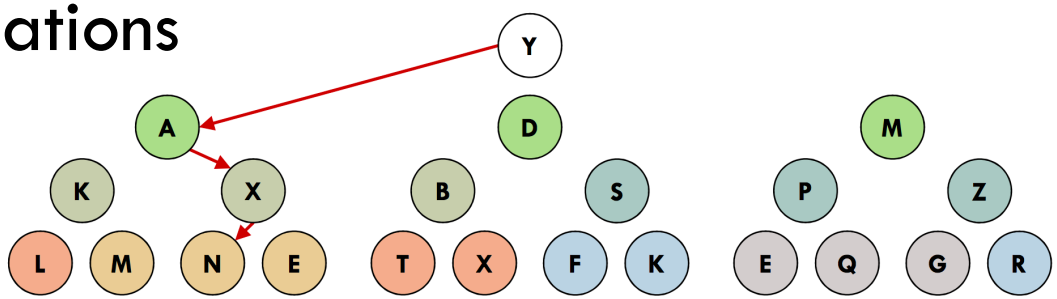
The ZCache

- Goal: design a highly associative cache with a low number of ways
- Improves associativity by increasing number of replacement candidates
- Retains low energy/hit, latency and area of caches with few ways
- Skewed associative cache: each way has a different indexing function (in essence, W direct-mapped caches)

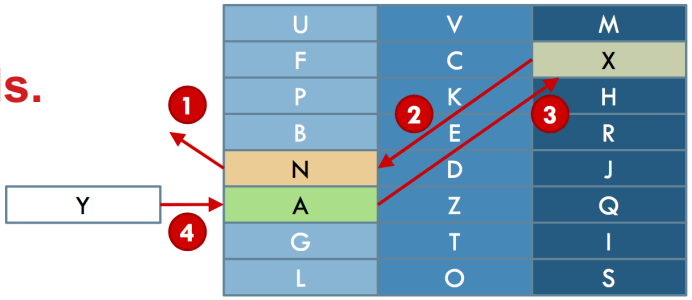
[Sanchez'10]

The ZCache

- When block A is brought in, it could replace one of four (say) blocks B, C, D, E; but B could be made to reside in one of three other locations (currently occupied by F, G, H); and F could be moved to one of three other locations



Read the paper for more details.

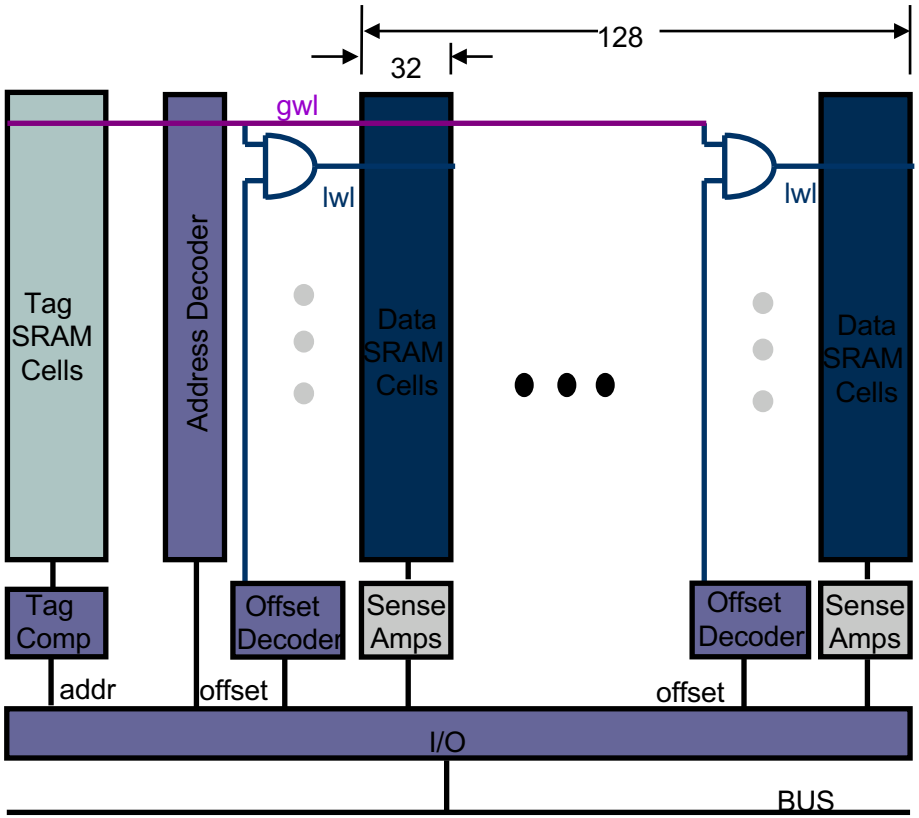


[Sanchez'10]

Content Aware Optimizations

Dynamic Zero Compression

- More than 70% of the bits in data cache accesses are 0s



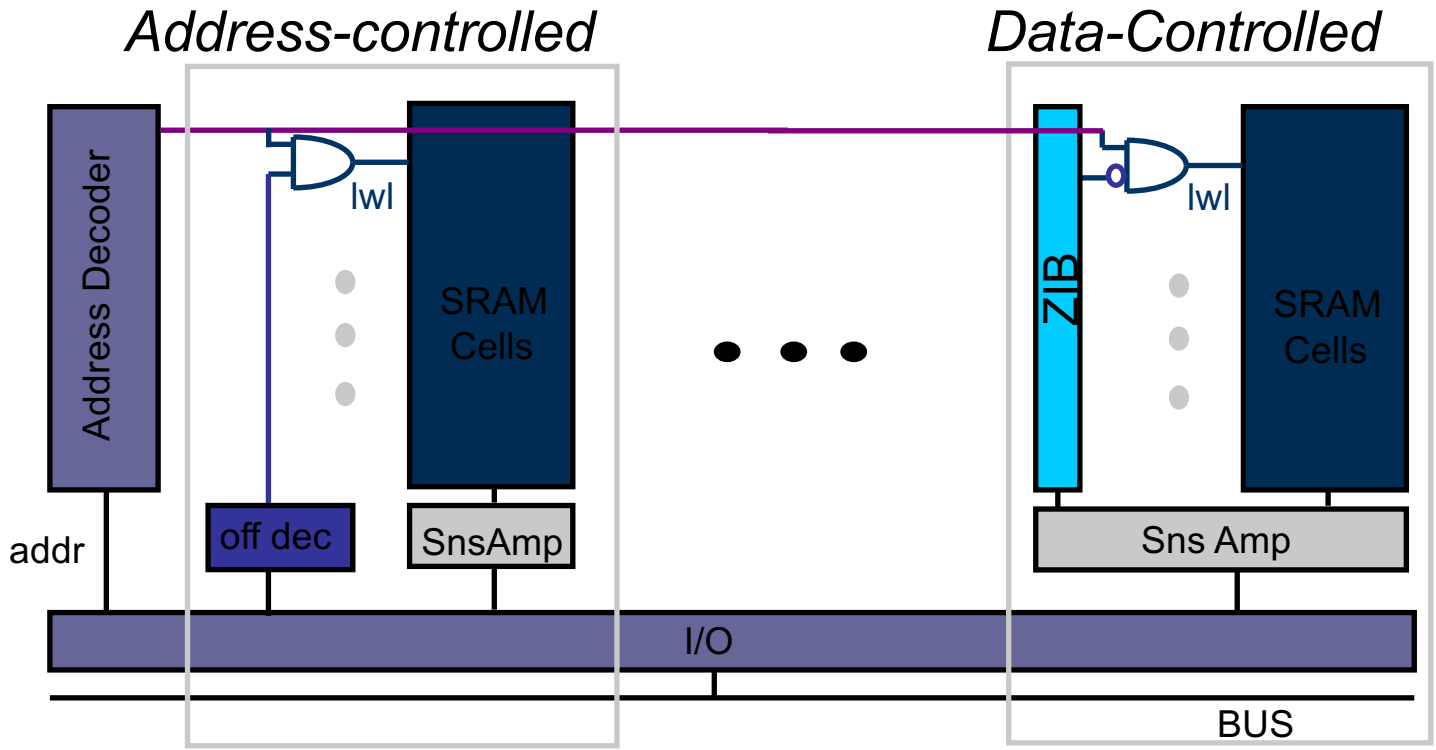
Example of a small cache

	Read		Write	
	(pJ)	(%)	(pJ)	(%)
Total	44.4	100.0	99.1	100.0
Decoder	5.5	12.4	5.5	5.5
word lines	1.1	2.5	1.1	1.1
Tag bitlines and sense-amp	3.0	6.2	3.0	3.0
Data bitlines and sense-amp	14.5	32.7	69.2	69.9
I/O buses	12.1	27.3	12.1	12.2
Other	8.4	18.9	8.4	8.5

[Villa'00]

Dynamic Zero Compression

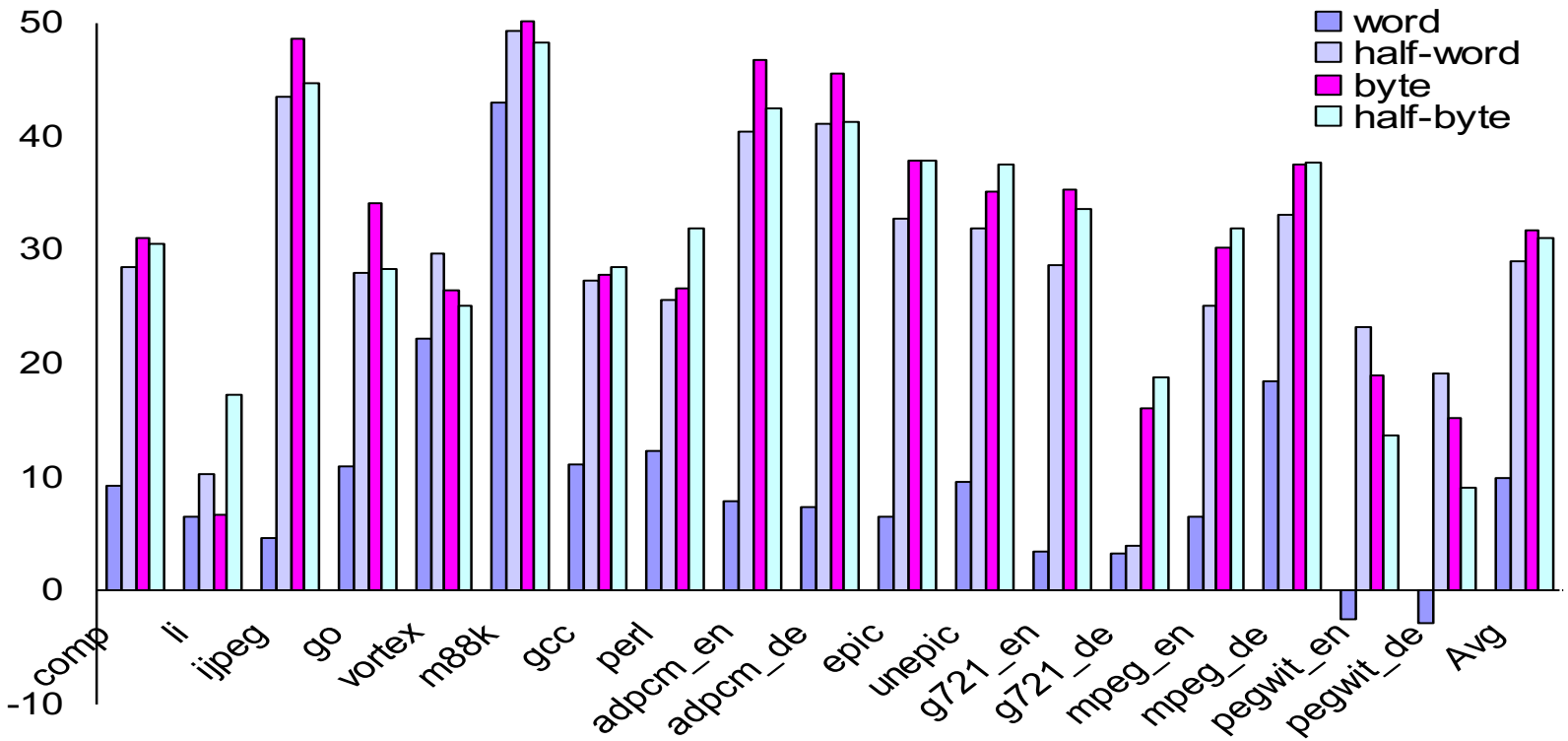
- Zero Indicator Bit; one bit per grouping of bits; set if bits are zeros; controls wordline gating



[Villa'00]

Dynamic Zero Compression

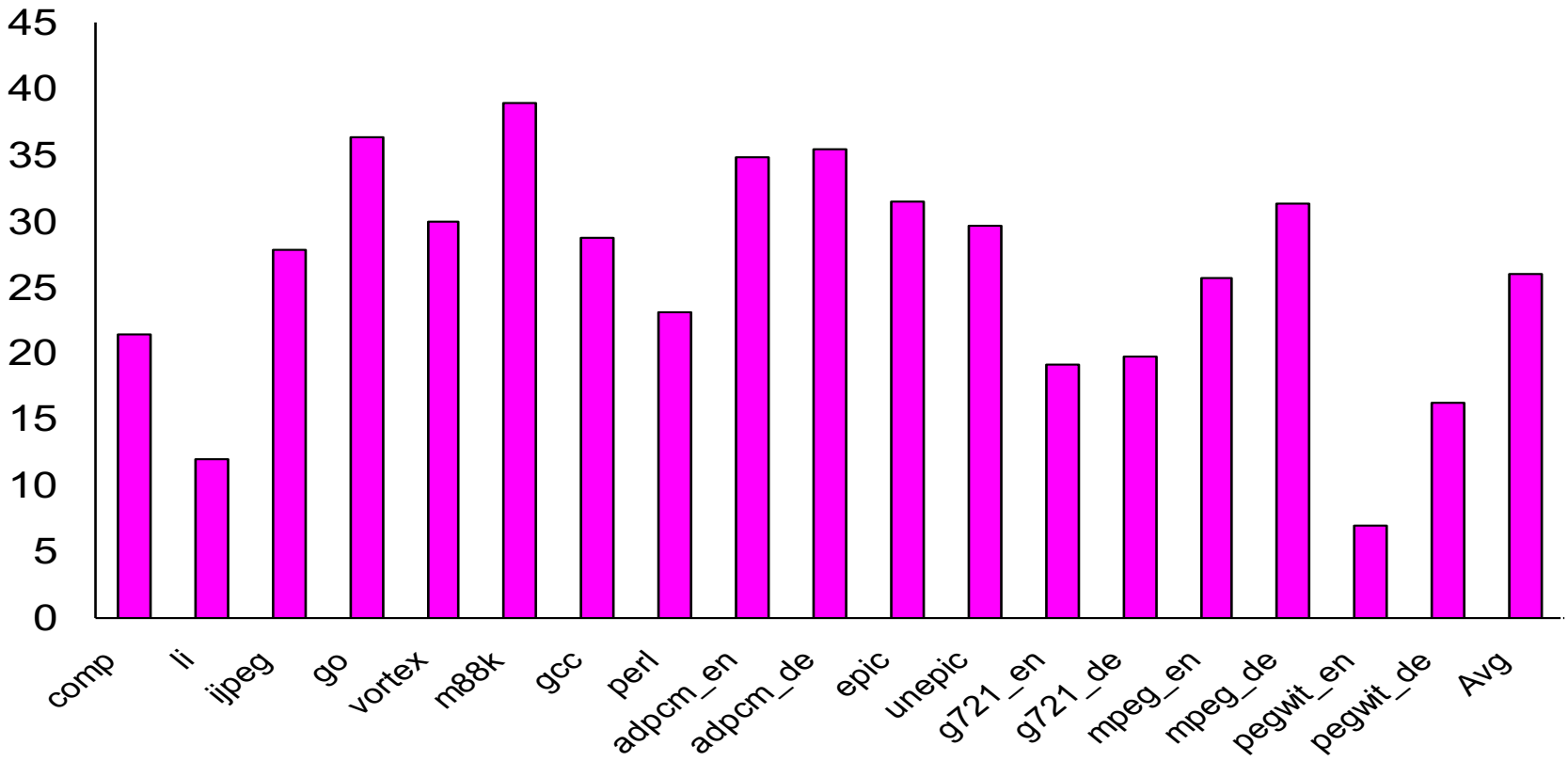
- Data cache bitline swing reduction



[Villa'00]

Dynamic Zero Compression

- Data cache energy savings



[Villa'00]