Lecture-16 (Cache Replacement Policies)
CS422-Spring 2018

Biswa@CSE-IITK
From SPEC92 – Miss rate: Still Applicable Today

Cache Size (KB)

0
0.02
0.04
0.06
0.08
0.1
0.12
0.14

1-way
2-way
4-way
8-way

Capacity

Compulsory
2:1 Cache Rule (Why? Piazza +1, Before Next Lecture)

miss rate 1-way associative cache size \( X \) = miss rate 2-way associative cache size \( X/2 \)

![Graph showing the relationship between miss rate and cache size for different associativity levels.](image-url)
Cache Replacement-101

- Think of each block in a set having a “priority”
  - Indicating how important it is to keep the block in the cache
- Key issue: How do you determine/adjust block priorities?
- There are three key decisions in a set:
  - Insertion, promotion, eviction (replacement)

- Insertion: What happens to priorities on a cache fill?
  - Where to insert the incoming block, whether or not to insert the block
- Promotion: What happens to priorities on a cache hit?
  - Whether and how to change block priority
- Eviction/replacement: What happens to priorities on a cache miss?
  - Which block to evict and how to adjust priorities
Eviction (Replacement) Policy?

- Which block in the set to replace on a cache miss?
  - Any invalid block first
  - If all are valid, consult the replacement policy
    - Random
    - FIFO
    - Least recently used (how to implement?)
    - Not most recently used
    - Least frequently used?
    - Least costly to re-fetch?
      - Why would memory accesses have different cost?
- Hybrid replacement policies
- Optimal replacement policy?
Belady

• Belady’s OPT
  • Replace the block that is going to be referenced furthest in the future by the program
  • How do we implement this? Simulate?

• Is this optimal for minimizing miss rate?

• Is this optimal for minimizing execution time?
  • No. Cache miss latency/cost varies from block to block!
  • Two reasons: Remote vs. local caches and miss overlapping
Our Goal:
To minimize off-chip DRAM accesses
LRU - 101

Cache Eviction Policy: On a miss (block $i$), which block to evict (replace) ?

Cache Insertion Policy: New block $i$ inserted into MRU.

Cache Promotion Policy: On a future hit (block $i$), promote to MRU

LRU causes thrashing when working set > cache size
Access Patterns

Recency friendly \((a_1, a_2, \ldots, a_k, a_{k-1}, \ldots, a_2, a_1)^N\)

Thrashing \((a_1, a_2, \ldots, a_k)^N\) \[k > \text{cache size}\]

Streaming \((a_1, a_2, \ldots, a_{\infty})^N\)

Combination of above three
Types of Workloads – 4MB cache

- (a) Cache “Friendly” Workloads
- (b) Cache “Fitting” Workloads
- (c) Cache “Thrashing” Workloads
- (d) Streaming Workloads
Limitations of LRU

LRU exploits **temporal locality**

**Streaming data** \((a_1, a_2, a_3, \ldots, a_\infty)\):
No temporal locality,
No temporal reuse

**Thrashing data** \((a_1, a_2, a_3, \ldots, a_n) [n>c]\)
Temporal locality exists. However, LRU fails to capture.
Bimodal Insertion Policy

if ( rand() < ε ) ε=1/16, 1/32, 1/64
   Insert at MRU position;
else
   Insert at LRU position;

For small ε: BIP retains thrashing protection of LRU insertion policy.

Infrequently insert lines in MRU position
Dynamic Insertion Policy – Multicore:+2 in Piazza

SDM – Set Dueling monitors

PSEL – n-bit saturating counters for deciding a policy

SDM (LRU-sets)

SDM (BIP-sets)

Follower Sets

n-bit PSEL

MSB = 0?

YES

Use LRU

No

Use BIP
Still Miles to Go

References to non-temporal data (*scans*) discards frequently referenced working set

Working set larger than the cache causes thrashing
Still Miles to Go

Working set larger than the cache \(\Rightarrow\) Preserve some of working set in the cache

Recurring scans (bursts of non-temporal data) \(\Rightarrow\) Preserve frequently referenced working set in the cache
RRP: Re-reference prediction

LRU chain position stored with each cache block

RRP head
MRU position

RRP tail
LRU position
RRIP

Re-reference Prediction Value
RRPV (n=2):

Qualitative Prediction:

Intuition: New cache block will not be re-referenced soon. Replaces block with distant RRPV.

Insert with RRPV=2, Evict with RRPV=3
promote blocks with RRPV=0.

Static RRIP (Single core) and Thread-Aware Dynamic RRIP
(SRRIP+BRRIP, multi-core, based on SDMs).
DRRIP

0 Imme-
diate

1 Inter-
mediate

re-reference

re-reference

re-reference

2 far

No Victim

3 distant

No Victim

No Victim

(SRRIP) Scan-Resistant insertion

(BRRIP) Thrash-Resistant insertion

eviction