Paper Review Notes

	Static RRIP	Dynamic RRIP
Single Core	4%	10%
Multi Core	7%	9%

uses 2 bit per Cache block

LRU Poor perf

- less Cache size
- Non temporal data replaces the active working set of application

Access Patterns

$[a_1 - - - a_k\ a_k\ a_{k-1} - - a_1]$

Recent friendly (for any k)

$(a_1 - - - a_k)^N$ (Thrashing Access pattern)

$k >$ cache size

$(a_1 - - - a_k)$ (streaming Access pattern)

$k = \infty$

$$[(a_1 \cdots a_k, a_k \ldots a_1)^A P_\varepsilon(a_1 \cdots a_k a_{k+1} \cdots a_m)]^N$$

$$[(a_1 - - - a_k)^A P_\varepsilon(b_1 b_2, b_m)]^N$$

mixed Access pattern

DIP (Dynamic Insertion Policy)

LRU

↓

missed block inserted at the top of LRU chain

LIP

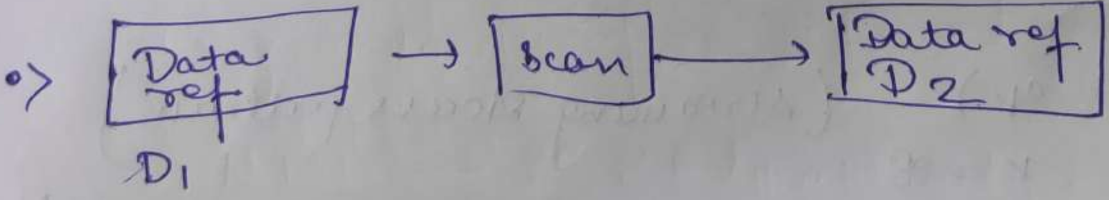
(LRU Insertion Policy)

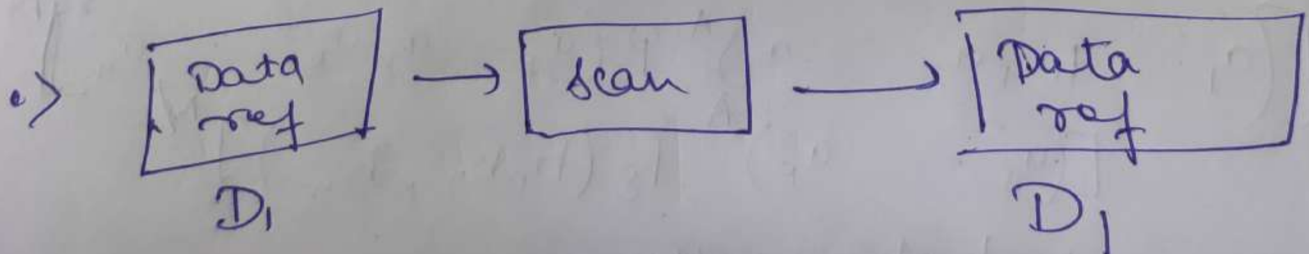
↓

missed block inserted at the tail of LRU chain

- DIP uses LIP when working set is larger than cache, LRU otherwise.
- Both LRU & DIP performs poor in case of mixed access pattern.
- Scan: Burst of references of data whose re-ref interval is in the distant future.
- Real world application frequently suffer from scan. To improve cache performanc we need scan resistant cache replacement policy

We have 2 cases





In case 1 : Decision during Scan did n't matter

But in case 2 it matter.
If we try to apply LIP during Scan & LRU rest. This will give good performance

Related Work

LFU → degrades performance in recency

Self tuning adaptive Policy: Significantly inc. the hardware overhead & complexity

Hybrid Cache Replacement : Uses set duelling to dynamically choose b/w multiple replacement policies. It provide scan resistant but require hardware & verification overhead of 2 caches

Dead block prediction : predict which of the block will be dead from RRIP chain. Victim selection policy select dead block closer to tail. It improve cache performance but require additional hardware overhead for dead block predictor.

Pseudo LIFO :

RRIP : high performing practical scan resistant Cache replacement with no significant hardware overheads or changes to existing cache structure.

NRU : 1 bit per cache block.

bit = 0 — means reference in near future

bit = 1 — reference in distant future

If all bit are 0, change all bits to 1.

Problem :

↳ Alway predict re-ref. in near future for incoming block

↳ If predict distant then do poor on near-immediate reref. Interval

Static RRIP

↳ granularity of re-ref pred is inc.

M bit → 2^M bit granularity

0 → near immediate

2^M-1 → distant future

↳ Always insert at long re-reference interval

↳ Author used $2^M - 2$ as l r-ref I.

[Prevent scan from polluting the cache]

[~~2 pood~~ giving initial $2^M - 2$ instead of $2^M - 1$ gives cache to learn some time

Victim Selection Policy

↳ Select with RRPV = $2^M - 1$

↳ Ties broken by starting victim search ~~by~~ from fixed location

↳ If no block found increment RRPV of all block by 1.

Hit promotion Policy

Hit Priority

↳ on hit change RRPV = 0

(Prioritize cache block that receive hit over not receiving hits)

↳ degrade when CB only hit onces

Freq. Priority

↳ on hit decrease RRPV by 1

↳ Goal is to prioritize replacement of infrequently re-ref cach.

SRRIP Scan resistant length

~~Sen~~ $S_{len} = (2^M - 1)*(A - W)$

Can be increased by increasing M.

This can also cause inefficient Cache Utilization when a cache block receive its last hit and RRPV became Zero

Dynamic RRIP

↳ SRRIP is vulnerable to thrashing

↳ To Avoid thrashing we can use

Bimodal RRIP

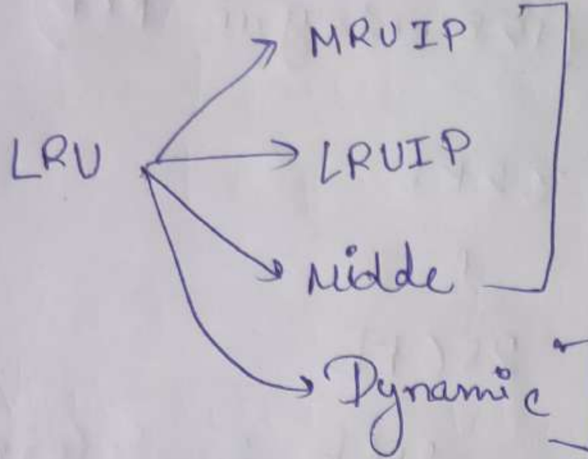
Insert Majority CB with distant reref interval

few with long re-ref interval

(for non thrashing it can lead to performance degrade)

We choose dynamically between SRRIP & BRRIP using set dueling

SRRIP Vs LRU



Cannot provide good scan resistane

Can provide good scan resistance : Can be implemented using SDM. But for higher associative SDM required can be more than the ~~set~~ Total set in cache

Extending RRIP to Shared Caches.

↳ ~~Same~~ We can see access as mixed pattern

↳ Extend DRRIP to shared caches is same as extending DIP to shared Caches.

↳ TADRRIP → uses 2SDM per Application to determines its optimal policy.

Experimental

4way out of Order (128 entry in ROB)

L1 I = 4way 32Kb
L1 D = " "

L2 → 8 way 256 Kb

LLC → 16 way 2Mb Single Core
8Mb Multi Core

64B → Cache line

only demand ref change LRU state

Load latencies →

L1 → 1
L2 = 10
L3 = 24
Main Mem = 250

Support 32 outstanding misses to memory

Benchmarks used —

↳ Vulnerable to replacement decision

Results and Analysis

↳ ~~Bi~~ taking m=2,3 performs best.

↳ Best Insertion position for every M is 2^M-2

↳ SRRIP-HP performs better than SRRIP-FP which seems little counter intuitive

↳ SRRIP performs good on cache size (LLC) range from 512Kb to 8MB with optimal m=2,3

↳ Can Easily be extended to shared caches due to its good performance on mixed pattern.

↳ Perform good even changing cache level to 2 (L_1 & LLC)

↳ performs 4% & 10% better than LRU on single Core

→ perform 7% & 9% better on multi-core than LRU.