Lecture 9: Large Cache Design II

• Topics: Cache partitioning and replacement policies
Basic Replacement Policies

• More reasonable options when considering the L2

• LRU: least recently used

• LFU: least frequently used (requires small saturating cntrs)

• pseudo-LRU: organize ways as a tree and track which sub-tree was last accessed

• NRU: every block has a bit; the bit is reset to 0 upon touch; when evicting, pick a block with its bit set to 1; if no block has a 1, make every bit 0
Why the Basic Policies Fail

• Access types that pollute the cache without yielding too many hits: streaming (no reuse), thrashing (distant reuse)

• Current hit rates are far short of those with an oracular replacement policy (Belady): evict the block whose next access is most distant

• 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
Insertion, Promotion, Victim Selection

- Instead of viewing the set as a recency stack, simply view it as a priority list; in LRU, priority = recency

- When we fetch a block, it can be inserted in any position in the list

- When a block is touched, it can be promoted up the priority list in one of many ways

- When a block must be victimized, can select any block (not necessarily the tail of the list)
MIP, LIP, BIP, and DIP

Qureshi et al., ISCA’07

- **MIP**: MRU insertion policy (the baseline)

- **LIP**: LRU insertion policy; assumes that blocks are useless and should be kept around only if touched twice in succession

- **BIP**: Bimodal insertion policy; put most blocks at the tail; with a small probability, insert at head; for thrashing workloads, it can retain part of the working set and yield hits on them

- **DIP**: Dynamic insertion policy: pick the better of MIP and BIP; decide with set-dueling
• Re-Reference Interval Prediction: in essence, 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

• RRIP can be easily implemented by setting the initial counter value to max-1 (does not require list management)
• Utility Based Cache Partitioning: partition ways among cores based on estimated marginal utility of each additional way to each core

• Each core maintains a shadow tag structure for the L2 cache that is populated only by requests from this core; the core can now estimate hit rates if it had W ways of L2

• Every epoch, stats are collected and ways re-assigned

• Can reduce shadow tag storage overhead by using set sampling and partial tags
Thread-aware DIP: each thread dynamically decides to use MIP or BIP; threads that use BIP get a smaller partition of cache

Better than UCP because even for a thrashing workload, part of the working set gets to stay in cache

Need lots of set dueling monitors, but no need for extra shadow tags
• Promotion/Insertion pseudo partitioning: incoming blocks are inserted in arbitrary positions in the list and on every touch, they are gradually promoted up the list with a given probability

• Applications with a large partition are inserted near the head of the list and promoted aggressively

• Partition sizes are decided with marginal utility estimates

• In a few sets, a core gets to use N-1 ways and count hits to each way; other threads only get to use the last way
- In an oracle policy, 80% of the evictions belong to a thrashing aggressor thread

- Hence, if the LRU block belongs to an aggressor thread, evict it; else, evict the aggressor thread’s LRU block with a probability of either 99% or 50%

- At the start of each phase change, sample behavior for that thread in one of three modes: non-aggr, aggr-99%, aggr-50%; pick the best performing mode
Set Partitioning

- Can also partition sets among cores by assigning page colors to each core
- Needs little hardware support, but must adapt to dynamic arrival/exit of tasks
Overview

MIP: MRU insertion policy (traditional approach)

DIP: Selects the best

LIP: LRU insertion policy

BIP: Bimodal insertion policy
Few insertions at head, most at tail

TADIP: Selects the best for each thread

RRIP: Probabilistic insertion near tail

PIPP: Inserts each thread at different positions + probabilistic promotion

AGGRESSOR-VT: Victimize the aggressor thread with a high probability

UCP: Partitions ways across threads based on marginal utility

Highest priority ← Priority stack of blocks in a set → Lowest priority
Title

• Bullet