CMP SCI 635: Modern Computer Architecture

Rethinking Belady's Algorithm to Accommodate Prefetching

Name: Kunjal Panchal Date: 3rd Oct, 2019

Student ID: 32126469 Paper: 9 – Cache [Jain18]

Strengths:

1. Harmony not only discards inaccurate prefetches to decrease the cache miss rates, but also evicts the lines which will be inaccurately prefetched in the future. PACMan does not consider the tradeoff between hit rate and traffic but instead uniformly deprioritizes all prefetch-friendly lines, resulting in large traffic overheads.
2. Harmony considers the global impact of its decisions by solving Flex-MIN collectively for all applications instead of solving it for each core in isolation, so it successfully leverages the cache space freed by one application to improve hit rates for other applications. As bandwidth becomes more constrained, Flex-MIN's ability to reason about the tradeoff between hit rate and traffic becomes more important.

Weaknesses:

1. Too much overhead for streaming workloads. When we know the application has high instruction/ data locality and won't need a cache line prefetched beyond a certain time interval, there is no point in keeping that even in low priority. (Lot of *dead lines*)
2. Improvement in demand hits did not increase overall memory traffic, it is possible to trade multiple prefetch hits for a single demand hit, which can lead to extra prefetch traffic.

Questions/Assertions:

1. Are prefetches worth considering? Do they actually take up enough amount out of total cycles for one process that we must include their impact on cache misses?
2. What does "prefetcher remaining fixed" means?
3. Why not focus on building a robust prefetching schedule from the PC behaviour of the past rather than learning the behaviour to detect inaccurate prefetches. I am not entirely sure, but is Harmony trying to do exactly this?
4. "*Evict the line that will be prefetched furthest in the future, and if no such line exists, evict the line that will see a demand request furthest in the future.*" Doesn't checking for the future behaviour incurs high cost and what will be the optimal window size to look into future?
5. In a multicore processor, what happens to the contents of a core's cache (say L1) when a context switch occurs on that cache? Is the behaviour dependent on the architecture or is it a general behaviour followed by all chip manufacturers?
6. Direct map cache uses a valid bit to effectively know if any data is present to a specific cache line. I wonder why similar optimization has not been done for associative (full or k-way) cache. Only reason I could imagine is that since for every read/write, a cache-slot will be filled, the cache will be filled pretty soon; and the probability of a slot being empty is zero in no time.