CSE 520 : Assignment 2 Report

Name: Viraj Savaliya

ASU Id: [redacted]

➢ ***RefCount Cache Replacement Policy Summary:***

Based on the recent studies, the authors of the paper "Counter-Based Cache Replacement Algorithms" are proposing a new cache replacement policy for L2 cache which is based on a reference counter. The goal of the policy is to deal with the problem that the performance gap between Least Recently Used (LRU) and other optimal replacement policies is large for highly associative caches. They are suggesting an Access Interval Prediction (AIP) policy based on reference counter. The idea behind the large gap in performance is that as a cache line will stay inside the cache till the point it becomes LRU and will occupy unnecessary space. If we remove at a point/threshold after which we are confident enough that the line will not be accessed again then we can create space for new incoming lines and improve performance by reducing capacity and conflict misses.

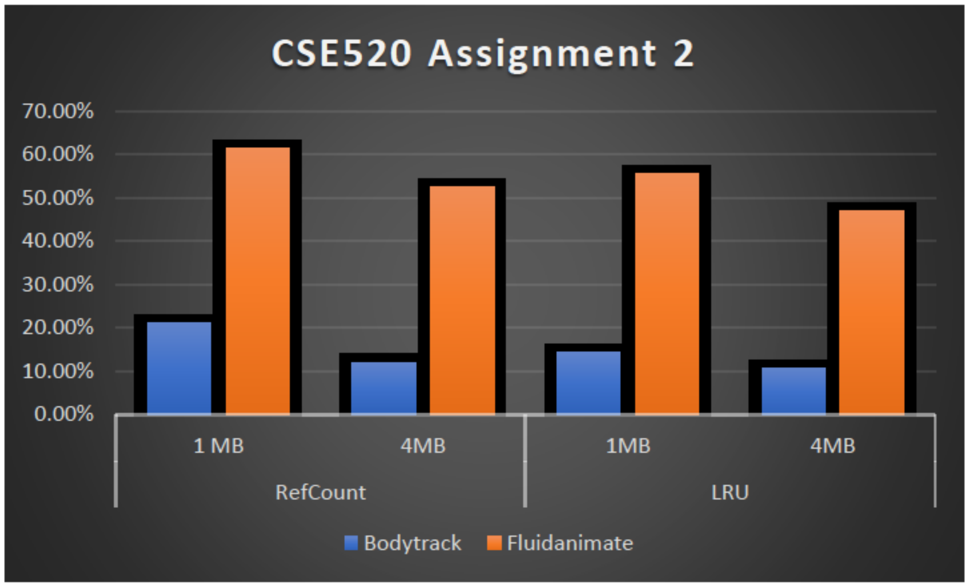
In Reference counter-based AIP replacement policy, each cache line in L2 cache is augmented with 21 bits and a 256x256 prediction table with each value in it of 5-bit size is created for all the cache lines. The expiration threshold value is unique and dynamically learned for all cache lines. In the 21 bits augmented for each cache line, 4 bits are for HashedPC, 4 bits for Counter (C), 4 bits for max event counts of current generation (maxCpresent), 4 bits for max event counts of prior generation (maxCpast) and 1 bit is confidence bit (conf). Inside the Prediction table for each value which is of size 5 bits, 4 bits are for max event counts of prior generation to be stored (maxCstored) and 1 bit is for confidence bit to be stored (confstored). The value inside the prediction table is accessed by using the hashedPC as row number and hashed value of the block address (hashedline) as column number. The 8-bit hashedPC is generated by XORing 8 bits part of the 64-bit PC. The hashedline is generated by XORing 4bit parts of 32 bit Set number (gives first 4 bits) and XORing 4-bit parts of 32-bit way number (gives last 4 bits) and then concatenating both to get 8 bits.

The first step in the algorithm is to increment the counter of all the lines in the set whenever there is an access to the set. Now if the access is hit, maxCpresent is updated with value whichever is greater amongst C and maxCpresent. Also, the counter of that cache line is reset to 0. Now if the access is miss, we need to find which line to replace (victim line to be evicted). For finding the victim we need to check which lines are expired i.e. eligible to be replaced. Except the MRU line, all lines are checked for expiration. A line is expired when all the following 3 conditions are satisfied: (i) C > maxCpresent, (ii) C > maxCpast and (iii) conf =1. After finding the expired lines, if there is no expired line found then the LRU line is the victim to be evicted. If there are more than one expired line then the line closest to LRU line is the victim to be evicted. After the victim is found, we need to update the prediction table by storing maxCstored as maxCpresent of the line which is evicted. The confstored field in the prediction table is set to '1' if maxCpresent = maxCpast of the line which is evicted or else it is reset to '0'. Now, the last step is when the new line is coming in we need to reset the C and maxCpresent fields to '0', maxCpast is taken from the maxCstored field of prediction table and conf bit is taken from the confstored field of prediction table.

➢ ***Simulation Results – Table :***

	RefCount		LRU	
	1 MB	**4MB**	**1MB**	**4MB**
Bodytrack	21.03%	11.98%	14.45%	10.62%
Fluidanimate	61.39%	52.45%	55.64%	46.91%

➢ ***Simulation Results – Graph:***



➢ ***Observations:***

1. Even though the RefCount policy is expected to perform better than the LRU, it is not the case with our experiments for Bodytrack and Fluidanimate benchmarks. The reason for the worse cache miss rate is that the predicted counter thresholds for the benchmarks with given configurations will be large. Hence, less evictions will be predicted and it doesn't improve the performance compared to LRU replacement policy.
2. When cache size is increased the difference between the LRU and RefCount missrates decrease. This is because when we increase the cache size the set associativity increases, and the capacity misses reduces. This means that we can benefit more from the counter-based algorithm when cache size is increased.
3. Also, when we increase the cache size we get a little speedup for the benchmark at times which is because of the more number of hits resulting.
4. When we compare both the benchmarks, we can see that the missrates in fluidanimate is more than in bodytrack for all cases. This might be because the working set in fluidanimate is bigger and is widely spread and not closely related in comparison to working set of bodytrack.
5. The nature and the working set of the benchmarks plays a major role in proving the RefCount replacement policy does not provide a better performance than LRU for L2 cache even when we increase the cache size.