Cesar Marin Margaillan

CSE 420

Assignment 2

Section I. Replacement Policies

In this lab, we used 4 different cache replacement policies:

Random Replacement

Random replacement policy is exactly as it sounds like. For accessing a block, there are no updates done, the block remains in its position. Whenever a victim is to be found, it is selected at random.

LRU

Least-recently-used algorithm works by creating a recency stack of the memory blocks. The stack is position based, going from the most recently used (MRU) position, to the least recently used (LRU) position. Whenever a block is accessed, it is moved to the MRU position, and all the other blocks in between the previous position of the block and the MRU position are pushed back to a more LRU position to create space. Whenever a victim has to be found for replacement, we simply select the one at the LRU position, once the block has been replaced, we move it to MRU.

PLRU

Pseudo LRU works the same but using a different technique. PLRU is binary tree based. The binary tree of bits (1 and 0) is of the necessary size so that all of the leaf nodes represent the blocks of memory. Then, every node for the root to the lead has a bit. Whenever searching for a victim, the algorithm will start at the root, go left when a 0 is found, and go right when a 1 is found until it reaches a leaf node, and that node will be the one replaced. Whenever a block is accessed then, it reverse-traverses the tree to the root, changing each plru bit so that it points away from the node.

IPV LRU

Insertion and promotion vector (IPV) LRU works using a recency stack as well like LRU, but using a different technique for insertion and access. For this algorithm we have an integer vector of size k+1 (k is the size of the each set) of predetermined integers from 0-(k-1). The last number in the vector specifies in what position should a new block be inserted. Each position i in the vector represents a block in the same position on the set. The number at V[i] represent where a block in the ith position will move to when accessed, pushing the other blocks forward or backward for space, just like in LRU. That's how insertion and accessing works in IPV. Finding a victim then, is done the same was as in LRU, the last element of the recency stack is selected for eviction.

Section II. Data

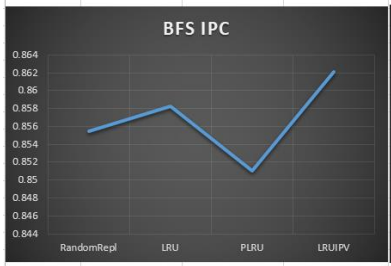
In this assignment, then, we used gem5 to test each of these 4 different replacement policies of 4 different workloads each: BFS, MST, SHA, and QUEENS. From each of these workloads, we compare the IPC (instructions per cycle) and L2 cache miss rate. NOTE: For a closer look at tables and graphs look at excel file.

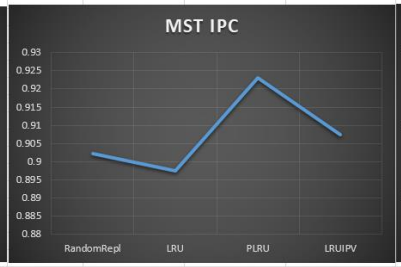
BFS	IPC	L2 Miss rate
RandomRepl	0.855493	0.63455
LRU	0.858282	0.636594
PLRU	0.851007	0.644151
LRUIPV	0.862139	0.628123

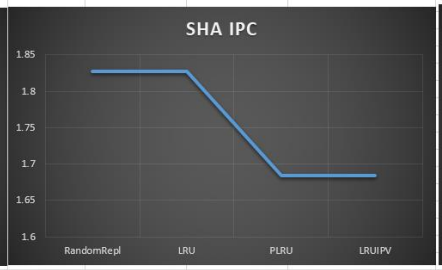
MST	IPC	L2 Miss rate
RandomRepl	0.902189	0.606594
LRU	0.897443	0.620035
PLRU	0.92318	0.535858
LRUIPV	0.907544	0.578146

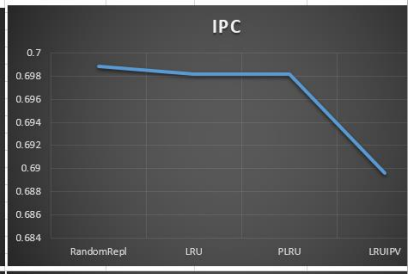
SHA	IPC	L2 Miss rate
RandomRepl	1.827777	0.83848
LRU	1.827864	0.838703
PLRU	1.683919	0.839698
LRUIPV	1.683877	0.84629

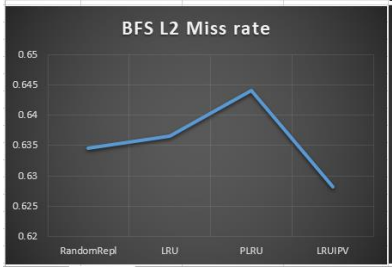
QUEENS	IPC	L2 Miss rate
RandomRepl	0.698875	0.833145
LRU	0.698177	0.833333
PLRU	0.698171	0.833786
LRUIPV	0.689601	0.842355

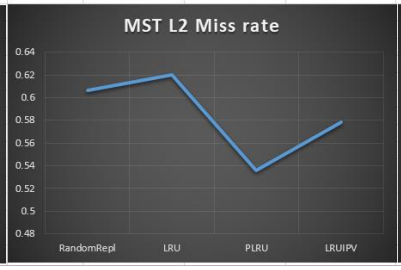


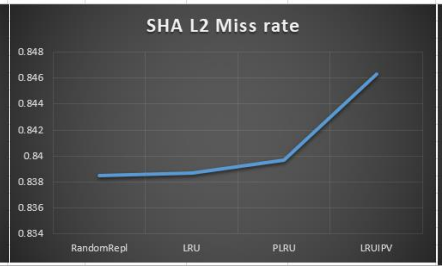


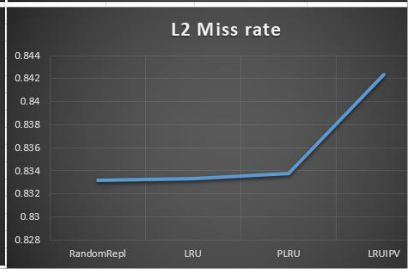












Section III. Observations

BFS. For BFS, we can see that IPV had the best IPC, and at the same time the lowest miss rate. I think the promotion vector in this case helped the way BFS works. The vector does not directly work like lru in this case, and sometimes moves blocks towards lru, and sometimes it keeps in the same position, especially at the end. I think this helps the miss rate, because when we access BFS, we keep going back to the sister nodes at the parent level, which are kept 'at the back' for later use without being moved up.

MST. For MST PLRU seemed to take the win on both stats. Although the IPC difference was small, we did notice a big difference in miss rate from LRU. Although these are supposed to give similar stats, PLRU has a 9% drop in the miss rate. Maybe this works better for MST just based in the way the blocks were placed in the leaf nodes, the bit change might be better for every access.

SHA. For SHA, the best method seems to be random replacement. Given the random nature of hashing, maybe giving no structure to cache at all might be the best policy for SHA. However, LRU and PLRU do not fall behind, and are really close to the random replacement performance.

QUEENS. For queens random replacement and LRU/PLRU seem to be the best the same way as in SHA. Although queens is not random in nature as SHA, is it a brute force algorithm, searching for many positions sequentially. Therefore, I am guessing that the given vector (which keeps some elements in the same position after access) was not helpful with this, as new blocks needed to constantly come in.