CS622 : Advanced Computer Architecture

Assignment 1

1^{st} September 2019

Siddharth Jayashankar (170699) | Abir Mukherjee(19111005)

Group Id: 5

Problem 1:

Table 1.1: L2 Cache Miss Rates

Trace File	*L2 Accesses (Count)*	*L2 Misses (Count)*			*L2 Miss Rate (PKL2A)*		
		Inclusive	*NINE*	*Exclusive*	*Inclusive*	*NINE*	*Exclusive*
hmmer	3,509,766	1,743,421	1,735,322	1,735,322	496.734	494.427	494.427
bzip2	10,657,628	5,398,166	5,397,576	5,397,576	506.507	506.452	506.452
gcc	14,610,812	3,036,461	3,029,809	3,029,809	207.823	207.368	207.368
sphinx3	10,753,448	8,820,349	8,815,130	8,815,130	820.234	819.749	819.749
gromacs	3,431,512	336,851	336,724	336,724	98.164	98.127	98.127
h264ref	2,348,574	969,678	965,624	965,624	412.879	411.153	411.153

PKL2A := Per 1K L2 Accesses

Observation 1.1: The L2 cache miss rate is marginally higher in case of an inclusive L3 cache.

Explanation: This marginal increase in the miss rate can be attributed to the inclusion victims. In an inclusive cache, a block that is frequently hit in L2 has its hit time updated only in L2 and not in L3. Thus it runs the risk of becoming the LRU block in L3. In case the LRU block in L3 is a recently used block in L2, it will upon eviction from L3, get invalidated in L2 to preserve inclusion. This causes an additional L2 miss on the next access to this block which must go all the way to main memory; thereby significantly increasing the miss penalty. Such blocks are termed as inclusion victims. Inclusion victims are a major drawback of an inclusive cache. As NINE and exclusive caches do not have any inclusion victims, their L2 miss rates are marginally lower.

Table 1.2: L3 Cache Miss Rates

Trace File	*L2 Accesses (Count)*	*L3 Misses (Count)*			*L3 Miss Rate (PKL2A)*		
		Inclusive	*NINE*	*Exclusive*	*Inclusive*	*NINE*	*Exclusive*
hmmer	3,509,766	391,226	376,344	300,046	111.468	107.228	85.489
bzip2	10,657,628	1,446,388	1,445,846	889,222	135.714	135.663	83.435
gcc	14,610,812	1,373,402	1,366,248	1,242,826	93.999	93.509	85.062
sphinx3	10,753,448	8,207,362	8,205,144	7,220,777	763.231	763.024	671.485
gromacs	3,431,512	170,531	170,459	159,306	49.696	49.675	46.424
h264ref	2,348,574	342,146	333,583	143,685	145.682	142.036	61.180

Observation 1.2: The miss rate of a NINE L3 cache is marginally lower than the miss rate of an inclusive L3 cache.

Observation 1.3: The miss rate of an exclusive L3 cache is significantly lower than the miss rates of both inclusive and NINE caches.

Explanation: This observation can be explained by examining the utilisation of the storage space in the L3 cache in the three cases. In an inclusive cache, all the blocks in L2 are also stored in L3. Thus the number of distinct cache blocks stored in both caches is equal to the size of the L3 cache. A miss in the L2 cache has the possibility of being found in one of the remaining ($\#_{block}$(L3) - $\#_{block}$(L2)) blocks. In case of a NINE cache, the inclusion policy is relaxed slightly and the number of distinct blocks goes up slightly as a block evicted from L3 need not be invalidated in L2. The marginal improvement in the miss rate of the NINE cache is attributed to this. In case of an exclusive cache, an L2 cache miss has the possibility of being found in the remaining $\#_{block}$(L3). Which is significantly greater than the ($\#_{block}$(L3) - $\#_{block}$(L2)) of the inclusive and NINE caches. This explains the significant decrease in the miss rate of the exclusive L3 cache.

Problem 2:

Table 2.1: L3 Cache Miss Rates

Trace File	*L2 Accesses (Count)*	*L3 Misses (Count)*			*L3 Miss Rate (PKL2A)*		
		SA LRU	*FA LRU*	*FA Belady*	*SA LRU*	*FA LRU*	*FA Belady*
hmmer	3,509,766	391,226	377,024	153,447	111.468	107.421	43.720
bzip2	10,657,628	1,446,388	1,361,401	536,836	135.714	127.740	50.371
gcc	14,610,812	1,373,402	1,369,924	939,289	93.999	93.761	64.287
sphinx3	10,753,448	8,207,362	8,387,248	3,068,580	763.231	779.959	285.358
gromacs	3,431,512	170,531	169,368	143,254	49.696	49.357	41.747
h264ref	2,348,574	342,146	335,880	111,605	145.682	143.014	47.520

Observation 2.1: The LRU replacement policy performs far worse than Belady's optimal replacement policy.

Explanation : The LRU replacement policy operates on the assumption that a block that has not been accessed recently is unlikely to be accessed in the near future. While this assumption is reasonable in some situations, it need not hold in all situations. When this assumption fails, the LRU policy is ineffective at optimising which blocks must be present in the cache. It is quite likely that the LRU assumption was incorrect for a large number of evictions from the L3 cache for the above set of trace files. This explains the vast gap in performance between the optimal policy and LRU.

Table 2.2: L3 Cache Miss Classification : FA LRU

Trace File	*L3 Misses (Count)*				
	SA LRU	*FA LRU*	*Cold*	*Capacity*	*Conflict*
hmmer	391,226	377,024	75,884	301,140	14,202
bzip2	1,446,388	1,361,401	119,753	12,41,648	84,987
gcc	1,373,402	1,369,924	773,053	596,871	3,478
sphinx3	8,207,362	8,387,248	122,069	8,265,179	-179,886
gromacs	170,531	169,368	107,962	61,406	1,163
h264ref	342,146	335,880	63,703	272,177	6,266

Table 2.3: L3 Cache Miss Classification : FA Belady

Trace File	*L3 Misses (Count)*				
	SA LRU	*FA Belady*	*Cold*	*Capacity*	*Conflict*
hmmer	391,226	153,447	75,884	77,563	237,779
bzip2	1,446,388	536,836	119,753	417,083	909,522
gcc	1,373,402	939,289	773,053	166,236	434,113
sphinx3	8,207,362	3,068,580	122,069	2,946,511	5,138,782
gromacs	170,531	143,254	107,962	35,292	27,277
h264ref	342,146	111,605	63,703	47,902	230,541

Observation 2.2: A fully associative cache does not always perform better than a set associative cache.

Explanation : The LRU replacement policy operates on the assumption that a block that has not been accessed recently will not be accessed in the near future. In a set associative LRU cache, the block that will be evicted will be the LRU block in that set; whereas, in the fully associative LRU cache, the block that will be evicted is the LRU block in the whole cache. These two blocks need not be the same. If the LRU assumption holds for the block evicted by the set associative cache but does not hold for the block evicted by the fully associative cache, the fully associative cache will have more misses. This phenomenon is showcased by trace file sphinx3 in the above table.