CS4110: COMPUTER SYSTEM DESIGN LAB
Assignment 2

Aravind Sankar CS11B033
Ganesh P Kumar CS11B015

September 14, 2014

1 Code Structure

We have a base class Cache which implements all basic operations, namely : `read`, `load` and `search`, apart from helper functions like `find_tag`, `find_set` to parse a given memory address. A Cache also has knowledge of its higher level and lower level caches by means of self-referential pointers. In case of $L_1$1 cache and LLC, the upper and lower level pointers respectively are set to `NULL`. The tag array for a cache has been represented in the form of a 2-D matrix, where there are rows equal to number of sets, and columns equal to associativity of the cache. The LRU_Cache, LFU_Cache and RR_Cache inherit from this common base class by overriding the `evict()` function.

A CacheController header offers functionality for handling reads and writes in a cache system/hierarchy. The 2 functions implemented in this class are the ones that will eventually be called from `pinatrace.cpp`. In `pinatrace.cpp`, we have added a function to parse the config file and generate the cache hierarchy. In the functions `RecordMemRead()` and `RecordMemWrite()`, we have added calls to `handle_read()` and `handle_write()` respectively from CacheController. We have also added a new function, `RecordInstructionRead()` that implements a call to `handle_read()`, to simulate instruction reading from the I-Cache.

2 Access Model

Read/Write hit in L_1

Access += 1
Hits += 1
for L_1 only.
If dirty bit is not set, set it.

Read/Write miss at L_1

Search in lower levels till you reach memory or get a hit, say at L_i cache.
Accesses += 1 for all L_x cache, where $1 \leq x \leq i$
Misses += 1 for all L_x cache, where $1 \leq x < i$
Hits += 1 for L_i cache

Now, block has to be loaded into all caches above L_i.
Accesses += 1 for all L_x cache, where $1 \leq x < i$

If you reach memory:
Accesses += 1 for all caches
Misses += 1 for all caches

For loading block in all caches,
Accesses += 1 for all caches

If it is write miss, set dirty bit for corresponding block in L_1 cache.

Eviction from L_i

Invalidate block in L_i.
Accesses += 1 for L_i.

Invalidate corresponding blocks at all higher level caches.
Accesses += 1 for all L_x cache, where $1 \leq x < i$

If original block was dirty, set corresponding block in L_{i+1} cache to dirty.
Accesses += 1 for L_{i+1}.

3 Experimental Results

The readings for LRU, LFU and RR Caches are as shown below.

LRU Cache

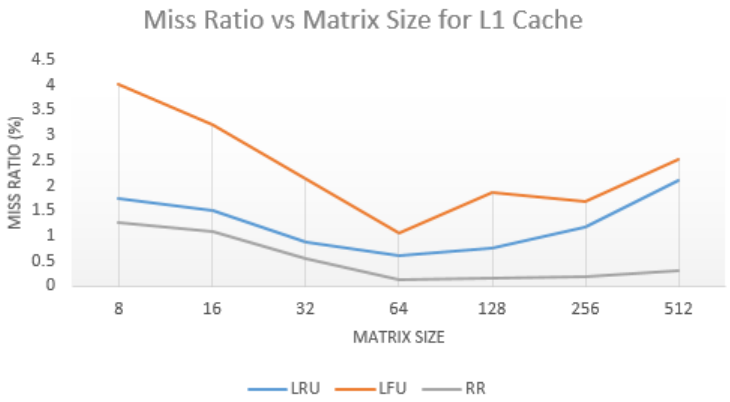
Matrix Size	L1			L2		
	Accesses	Hits	Miss Ratio (%)	Accesses	Hits	Miss Ratio (%)
8	1897277	1816094	1.737	78111	13102	59.198
16	2212402	2129751	1.517	79262	13737	58.134
32	4523235	4426296	0.886	92267	19920	49.658
64	22294974	22009985	0.597	275660	109602	17.102
128	162061776	159462610	0.762	2555182	1069676	12.582
256	1275272243	1243130133	1.176	31778834	12241202	17.262
512	10246533891	9782047462	2.101	457444736	165293075	21.262

LFU Cache

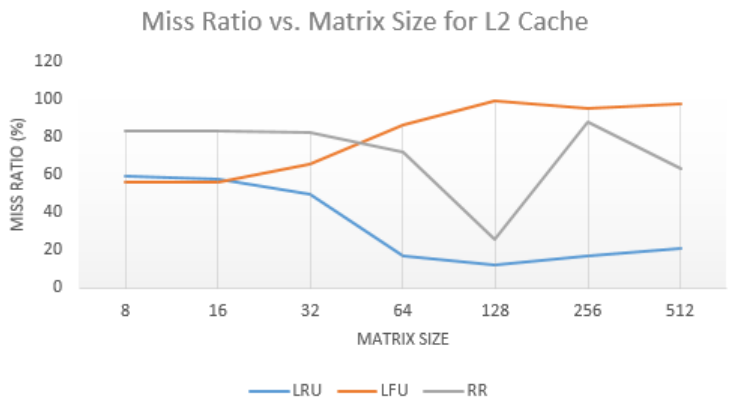
Matrix Size	L1			L2		
	Accesses	Hits	Miss Ratio (%)	Accesses	Hits	Miss Ratio (%)
8	1961877	1774015	4.014	176831	32660	55.978
16	2268645	2093187	3.208	168784	30622	55.861
32	4622777	4370408	2.142	240478	32554	65.948
64	22580321	21905970	1.067	651113	32285	86.333
128	166680154	157676530	1.873	8866453	23458	99.221
256	1298990246	1236861714	1.674	60700777	1060037	94.967
512	10491076846	9739682858	2.525	741905476	5479688	97.828

RR Cache						
Matrix Size	L1			L2		
	Accesses	Hits	Miss Ratio (%)	Accesses	Hits	Miss Ratio (%)
8	1889329	1824611	1.277	61897	4016	82.979
16	2204122	2138721	1.102	62459	4077	82.9
32	4508443	4441747	0.545	63552	4210	82.688
64	22192873	22111513	0.139	76732	8647	71.82
128	161035198	160407063	0.174	617976	207348	25.729
256	1262640331	1255419064	0.199	7184658	295252	88.219
512	10040890330	9962078499	0.299	78626000	10874922	63.624

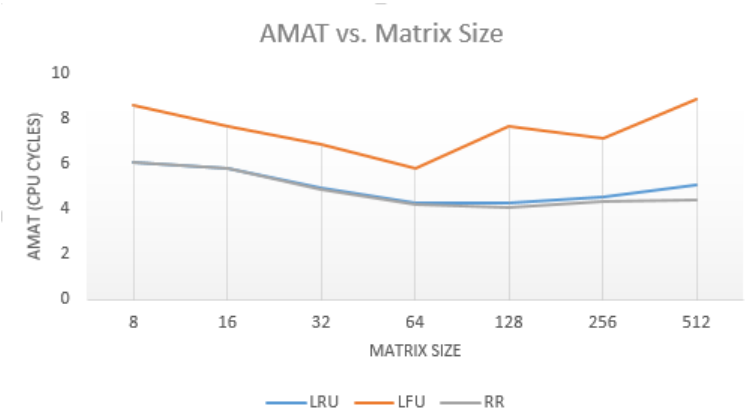
4 Observations and Inference



As you can see, RR has the best performance with respect to L1 Miss ratio, followed by LRU, and LFU has the worst performance. Also, note that for all three policies, the Miss Ratio decreases till Matrix size 64, after which it increases. The right hand side of this V is intuitive: as the matrix size becomes larger, it becomes tougher to accomodate the whole matrix into the cache, resulting in more misses during the process. The reason for the initial decrease in miss ratio is described as follows.
Initially, matrix sizes ≤ 64 will fit completely into the caches. So the only misses that will be encountered will be during the initial filling stage. Note that each element in the matrix gets accessed n times during the algorithm, where n is the size of the matrix. So the number of hits increases with increase in matrix size, leading to a decrease in miss ratio.



The behaviour of L2 Miss ratio is more erratic. For small matrices (< 16), LFU seems to have the best performance, followed by LRU and finally RR. However, as matrix size increases, LRU miss rate decreases significantly, while LFU miss rate increses, making LFU the worst for matrix sizes > 32. RR has an understandably random behaviour with respect to L2 Miss ratio.



RR and LRU have very similar AMAT pattern with respect to increasing matrix sizes, with RR being marginally better than LRU for matrix sizes > 64. However, LFU has the worst performance, and takes at least 25% more time for similar matrix sizes.