
Project 1: Cache Hierarchy

October 28, 2016

CONTENTS

1	Exploring the L1 Cache Design	3
1.1	Plot 1	3
1.2	Plot 2	4
1.3	Discussion based on Plot 1 and Plot 2	5
2	Exploring the Replacement Policy	5
2.1	Plot 3 and Discussion	5
3	Exploring the L2 Cache Design	7
3.1	Plot 4	7
3.2	Plot 5	8
4	Exploring the Inclusion Property choices	8
4.1	Plot 6	8

1 EXPLORING THE L1 CACHE DESIGN

1.1 PLOT 1

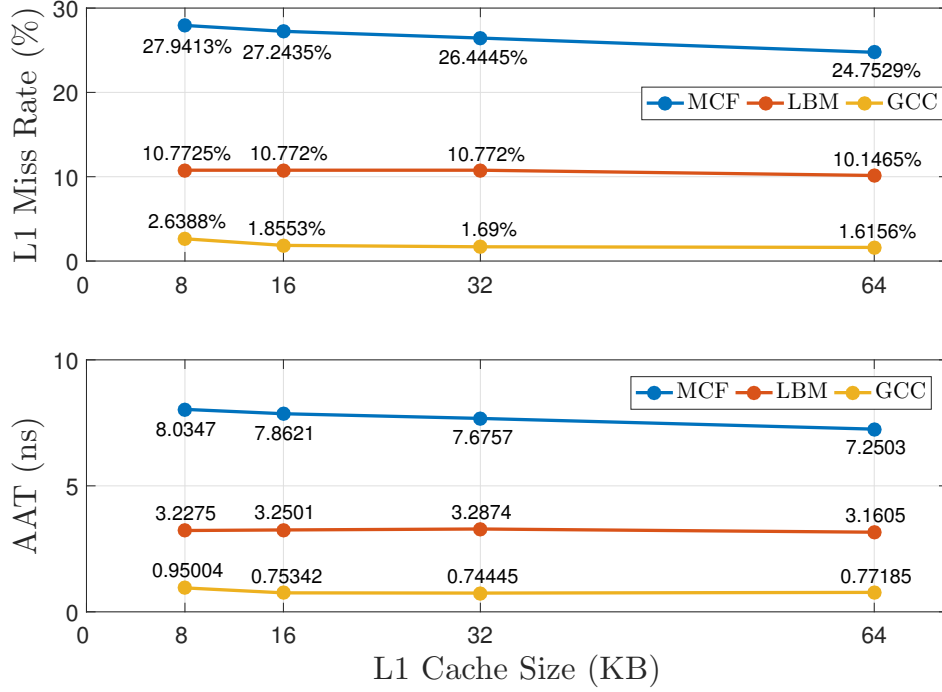


Figure 1.1: L1 cache miss rate, Average Access Time(AAT) and L1 cache size, when L1 associativity is 4

From Figure 1.1, generally, we have observations that L1 cache miss rate and Average Access Time of L1 cache decrease as L1 cache size increases. However, we can see an exception that AAT slightly increases when L1 cache size increases from 32KB to 64KB in benchmark GCC, also when it increases from 8KB to 32KB in LBM.

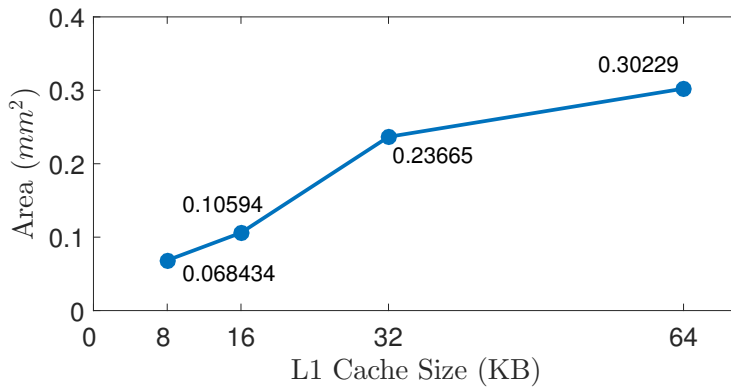


Figure 1.2: L1 cache area and L1 cache size when L1 cache associativity is 4

1.2 PLOT 2

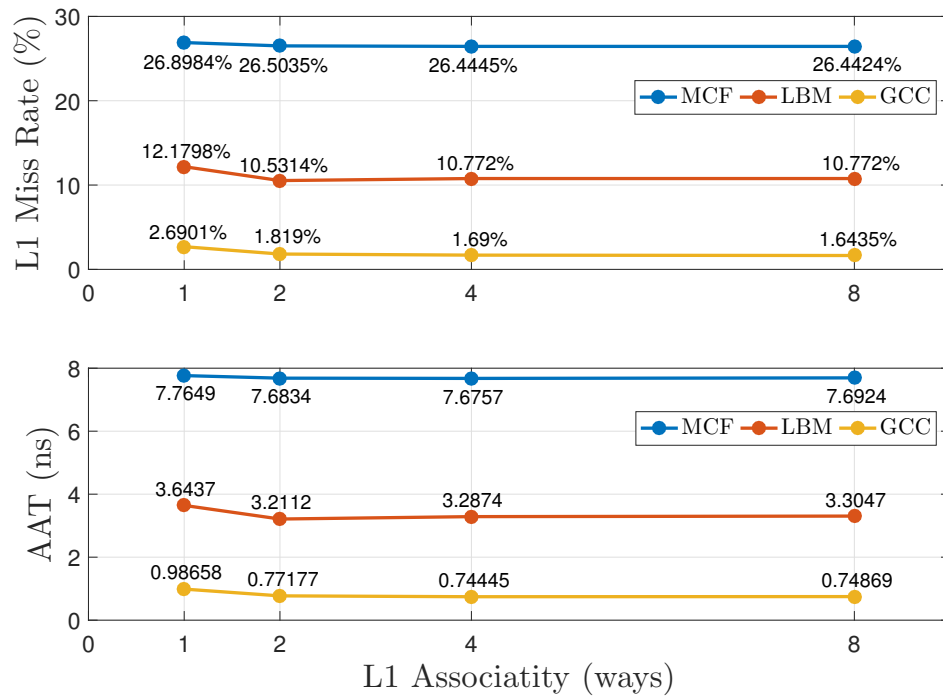


Figure 1.3: L1 cache miss rate, Average Access Time(AAT) and L1 cache associativity, when L1 cache size is 32KB

From Figure 1.3, we can find that L1 cache miss rate and Average Access Time of L1 cache decrease as L1 cache associativity increases from 1 to 4. Moreover, the miss rate and AAT decrease more distinctly when L1 cache associativity changes from 1 way (direct mapped) to 2 ways. The decreasing speed of L1 miss rate and AAT slows down as L1 cache associativity increases. There is also an exception that L1 miss rate and AAT increases when L1 increases from 2 ways to 4 ways in LBM. However, when L1 cache associativity increases from 4 ways to 8 ways, the L1 miss rate decreases but AAT slightly increases.

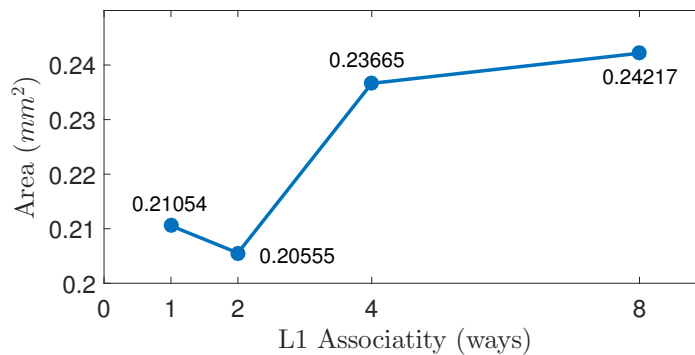


Figure 1.4: L1 cache area and L1 cache associativity when L1 cache size is 32KB

1.3 DISCUSSION BASED ON PLOT 1 AND PLOT 2

We have some embryo observations about the relationship between L1 miss rate, AAT and L1 cache size as well as associativity in section 1.1 and 1.2.

Lower L1 miss rate doesn't mean a lower AAT, just as we have seen from Figure 1.1 and 1.3. For no-L2-cache design, we have,

$$\text{AAT} = \text{L1 hit time} + \text{L1 miss rate} \times \text{Miss Penalty} \quad (1.1)$$

A lower L1 miss rate is achieved by increasing L1 cache size or associativity. However, it might also increase L1 hit time as side-effect. Therefore, whether AAT decreases depends on the competition between L1 hit time change and L1 miss rates change.

Now, better values for parameters of L1 cache structure to improve AAT is desired.

Table 1.1: L1 Cache Area

Area		Cache Size			
		8	16	32	64
Cache Associativity	1	0.053293238	0.096748995	0.210543576	0.330469394
	2	0.083756164	0.130107044	0.205554649	0.350242085
	4	0.068434156	0.105941693	0.236647681	0.302289370
	8	0.102584886	0.130444675	0.242170635	0.360317611

Take what is discussed in section 1.1 and 1.2 into account, based on table 1.1, I would like to choose 16KB 4-ways or 32KB 2-ways cache.

2 EXPLORING THE REPLACEMENT POLICY

2.1 PLOT 3 AND DISCUSSION

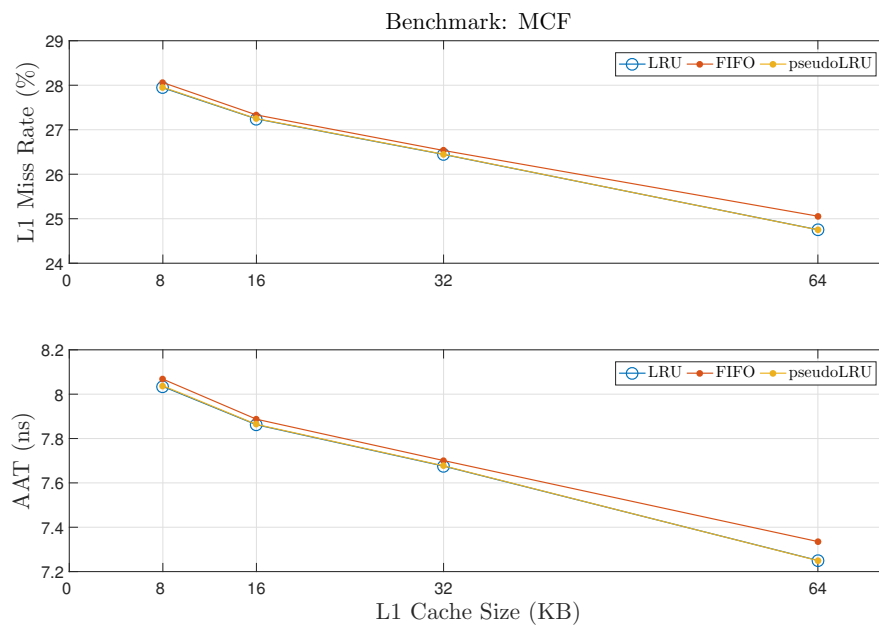


Figure 2.1: L1 cache miss rate, AAT and L1 cache size as well as replacement policy when L1 cache associativity is 4

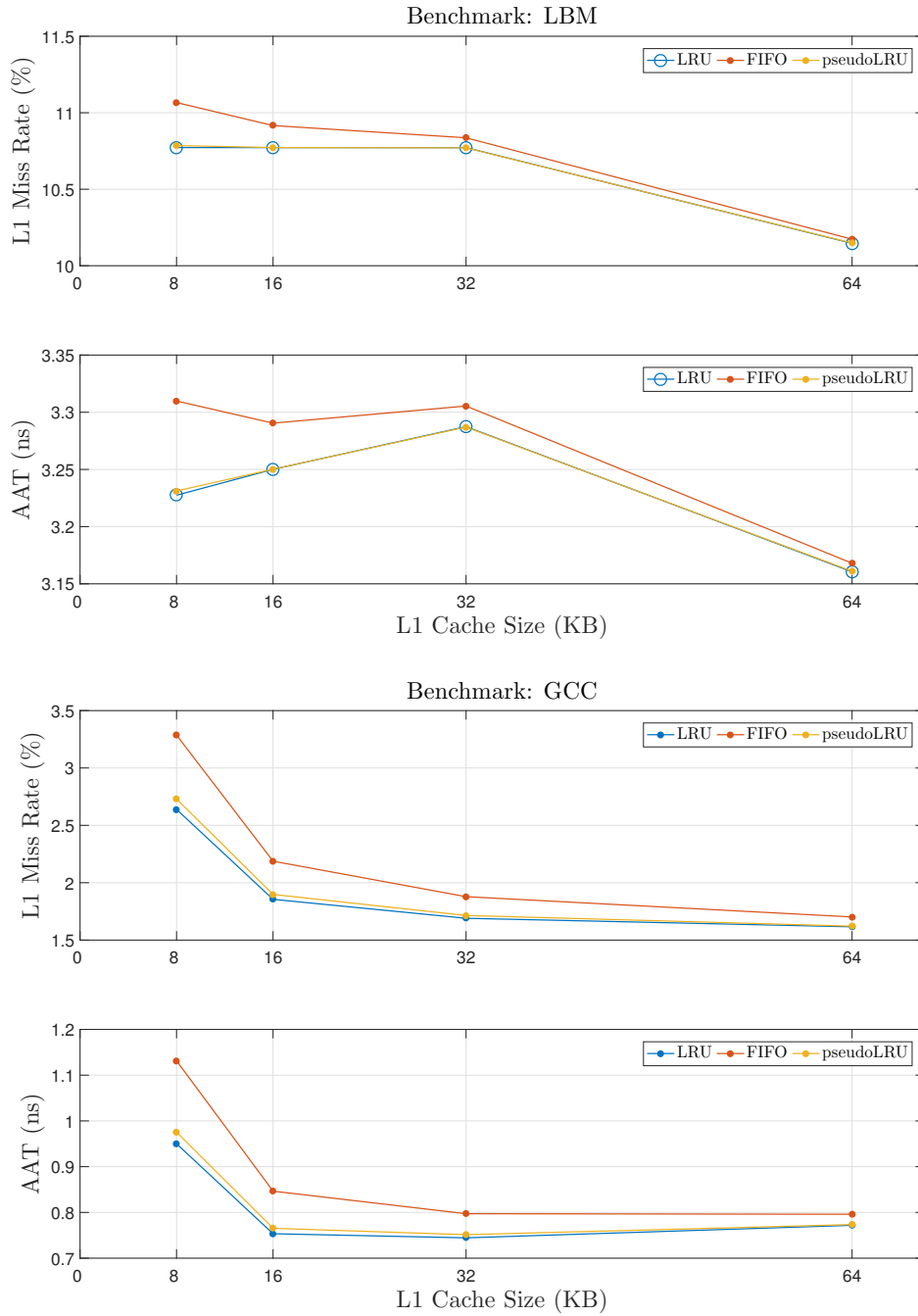


Figure 2.2: L1 cache miss rate, AAT and L1 cache size as well as replacement policy when L1 cache associativity is 4

From the results of 3 benchmarks, we can first find that L1 miss rate and AAT generally decreases as L1 cache size increases, no matter what replacement policy we take, just like what we find in section 1.1 including the exception.

Furthermore, we can find that the blue line is always beneath the other two, which indicates that the one with LRU replacement policy always has best performance on miss rate and AAT among all. The read line is always above the other two, which indicates that the one with FIFO replacement policy always has worst performance. However, the difference of performance,

i.e. miss rate and AAT, among these three replacement policy, becomes smaller and smaller as L1 cache size increases.

Meanwhile, Figure 2.1 and 2.2 show that the blue line and the yellow line nearly stick with each other. This indicates that pseudoLRU policy almost has similar better performance as LRU policy.

In my simulator program, updating LRU and FIFO ranking is $O(1)$ operation, fetching the LRU or FIFO position (the way that will be replaced) is $O(n)$ operation; updating pseudoLRU tree is $O(\log n)$ operation, fetching pseudoLRU leaf (the way that will be replaced) is $O(\log n)$ operation. However, the associativity of cache is no more than 8, which makes them nearly no differences in time complexity. There are other ways to achieve LRU policy, such as using state bits or matrix. State bits method saves space but increase difficulty in operations; matrix method make operations easier but occupies too much space. A pseudoLRU tree needs smaller space than matrix method and operations are easier than state bits method, and in the meantime, it performs as well as LRU policy. These make pseudoLRU a pratical policy to use.

3 EXPLORING THE L2 CACHE DESIGN

3.1 PLOT 4

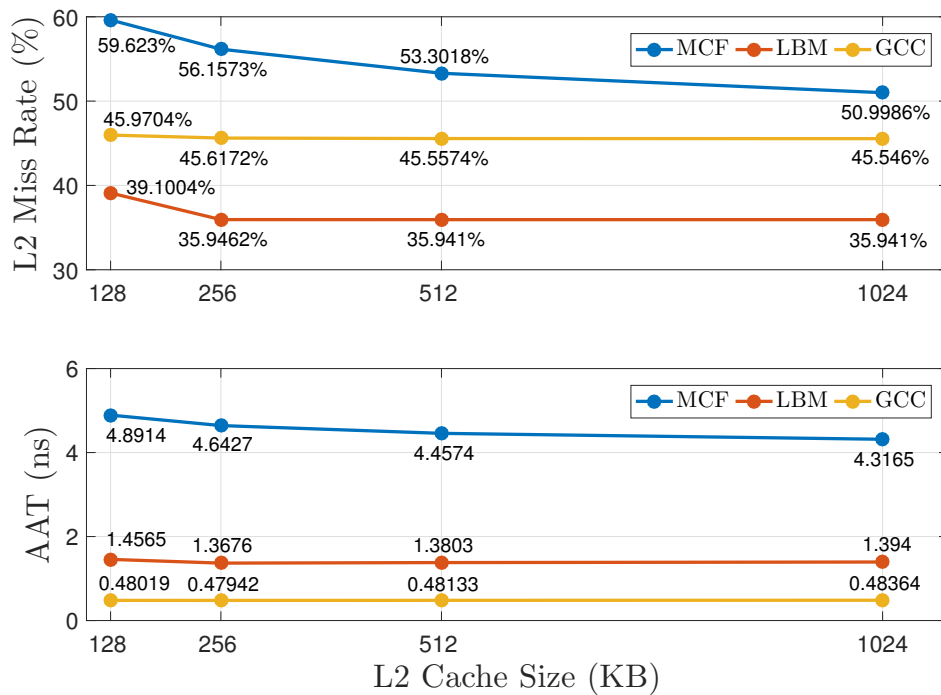


Figure 3.1: L2 cache miss rate, Average Access Time(AAT) and L2 cache size, when L1 cache size is 16KB, L1 associativity is 4, L2 associativity is 8

3.2 PLOT 5

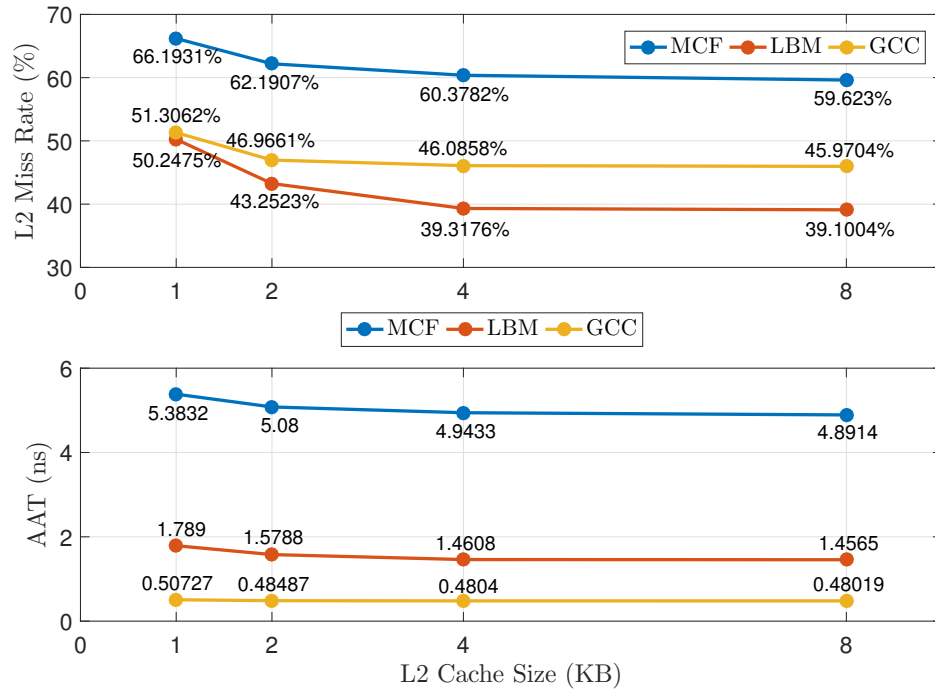


Figure 3.2: L2 cache miss rate, Average Access Time(AAT) and L2 cache associativity, when L1 cache size is 16KB, L1 associativity is 4, L2 cache size is 128KB

4 EXPLORING THE INCLUSION PROPERTY CHOICES

4.1 PLOT 6

