University of Central Florida

Department of Computer Science

CDA 5106: Fall 2024

Machine Problem 1: Cache Design, Memory Hierarchy Design

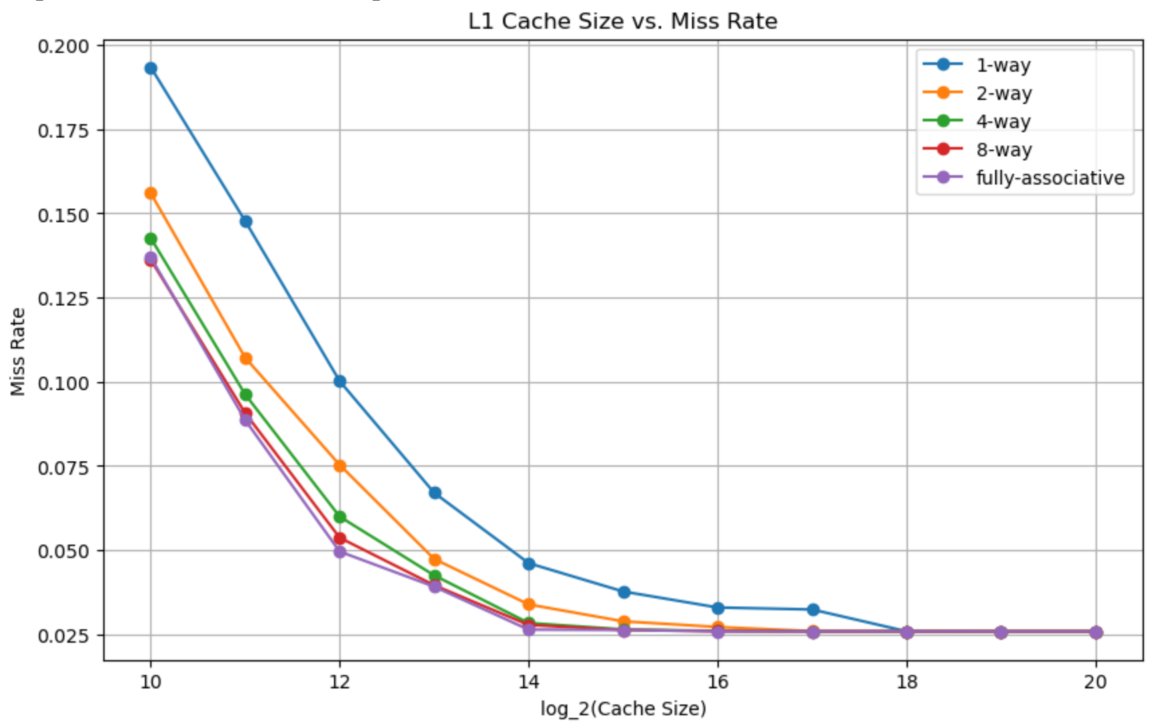
by

Justin Morera

Honor Pledge: "I have neither given nor received unauthorized aid on this test or assignment."

Student's electronic signature: _____Justin Morera___________
(sign by typing your name)

Experiment 1 – L1 Cache Exploration



1. Discuss trends in the graph. For a given associativity, how does increasing cache size affect miss rate? For a given cache size, what is the effect of increasing associativity?

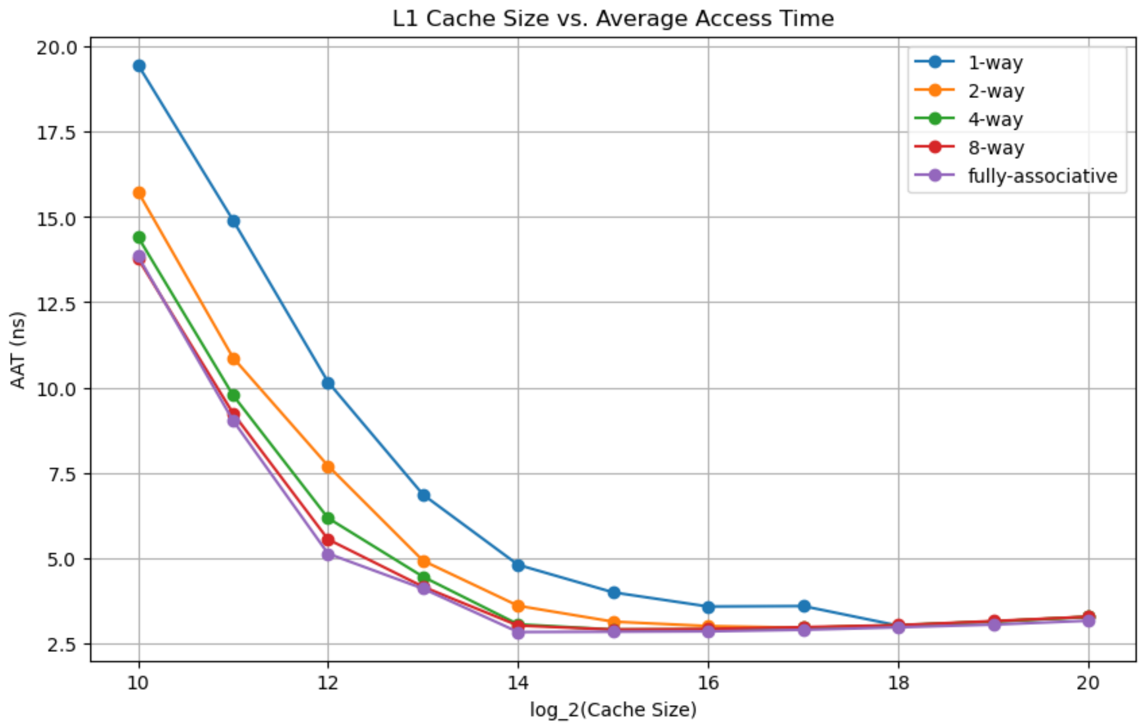
The data clearly show that as cache size is increased, cache miss rate decreases significantly. It is also deducible that increases in associativity also decrease cache miss rate. However, it must be noted that as cache size increases, the reduction in miss rate becomes less significant, and after 16KB (2^{14}), the effect becomes negligible for all cache sizes.

2. Estimate the compulsory miss rate from the graph.

The trends reveals that associativity plays a major role in compulsory miss reduction. All initial accesses are compulsory misses until a cache set is full; this means that with higher associativity, there is a higher likelihood that blocks will be mapped to the same set, and thus, once the set is full, will no longer contribute to compulsory misses. As such, it can be noted that higher associative caches plateau at the bottom of the graph earlier than lower associative caches, especially direct-mapped caches. At this plateau, caches are so large that the benchmark has a comparatively higher number of tags to map to each set, thereby decreasing the number of conflict misses to almost zero, and thus the miss rate consist almost entirely of compulsory misses with sufficiently high cache size. It seems the compulsory miss rate on this graph is roughly 0.025.

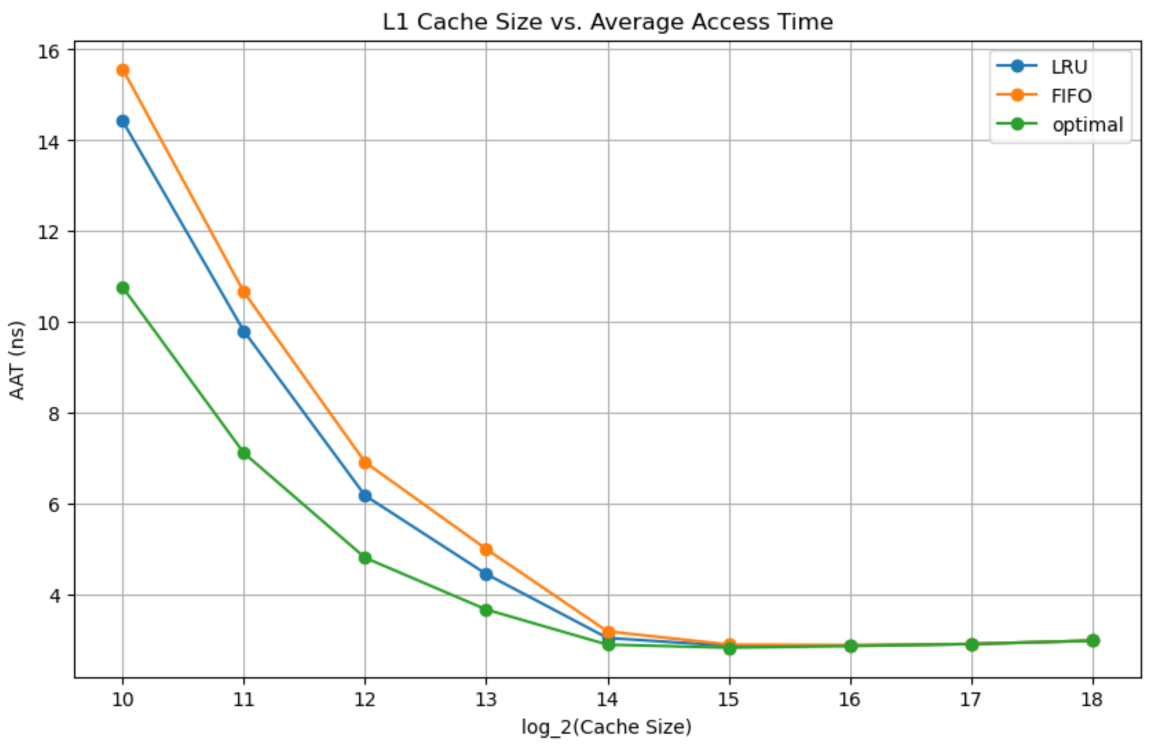
3. For each associativity, estimate the conflict miss rate from the graph.

It is known that fully associative caches have the lowest conflict miss rate, which we can treat as 0. The total miss rate for fully associative caches is virtually entirely due to compulsory misses; as such, for any given cache size, we can use the total miss rate from a fully associative cache of the same size to subtract out the compulsory miss rate (as it will be the same for any cache of the same size) from the miss rate of a cache with a lower associativity. For direct-mapped caches, the conflict miss rate is roughly $0.18 - 0.1374 = 0.0425$, for 2-way caches, it is $0.075 - 0.05 = 0.025$, for 4-way, it is $0.0625 - 0.05 = 0.0125$, for 8-way, $0.14375 - 0.1375 = 0.00625$, and for fully associative, it is 0.



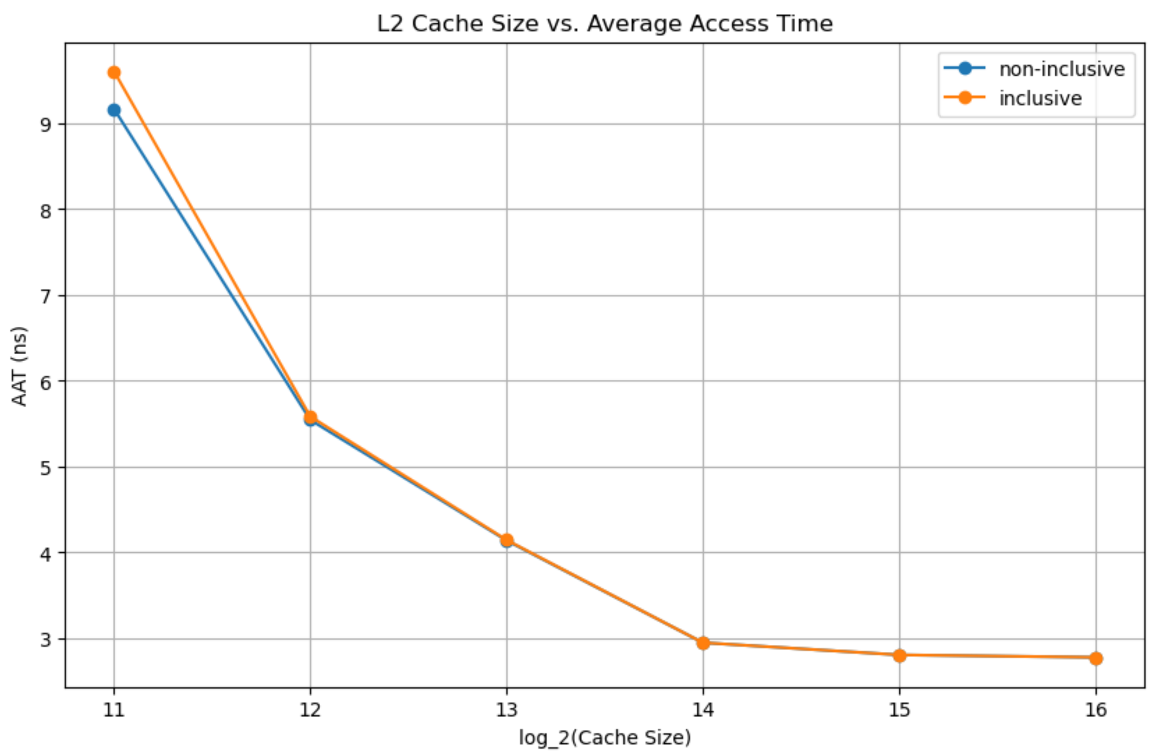
1. For a memory hierarchy with only an L1 cache and BLOCKSIZE = 32, which configuration yields the best (i.e., lowest) AAT?

Given the data, a cache size of 16KB (2^{14}) with full associativity is the optimal configuration to minimize average access time.



1. Discuss trends in the graph. Which replacement policy yields the best (i.e., lowest) AAT?

It goes without saying that optimal replacement policy is, of course, optimal; however, since this policy is not implementable in practice, the data also show that LRU replacement is marginally better than FIFO in terms of AAT. The difference is relatively constant until 16KB (2^{14}), where the access times begin to converge at just under 3ns.



1. Discuss trends in the graph. Which inclusion property yields a better (i.e., lower) AAT?

Both inclusion policies gave nearly identical results except for when the cache size was at its lowest. Naturally, a non-inclusive property should yield a lower average access time due to the absence of invalidations between caches. This is, however, clearly a negligible operation as it does not take much time to flip a single bit, and a write back to main memory is not always necessary. It should be noted that there is a known bug in the code used to produce this data where, when testing on the validation files, there was exactly one additional writeback and three sets where two blocks were inverted for L2 cache only. While this bug could not be identified and fixed, the differences from the expected output were minute and did not affect AAT calculations.