Machine Problem 1: Cache Design, Memory Hierarchy Design

Kobee Raveendran

October 28, 2021

Honor Pledge: "I have neither given nor received unauthorized aid on this test or assignment."
Student's electronic signature: Kobee Raveendran

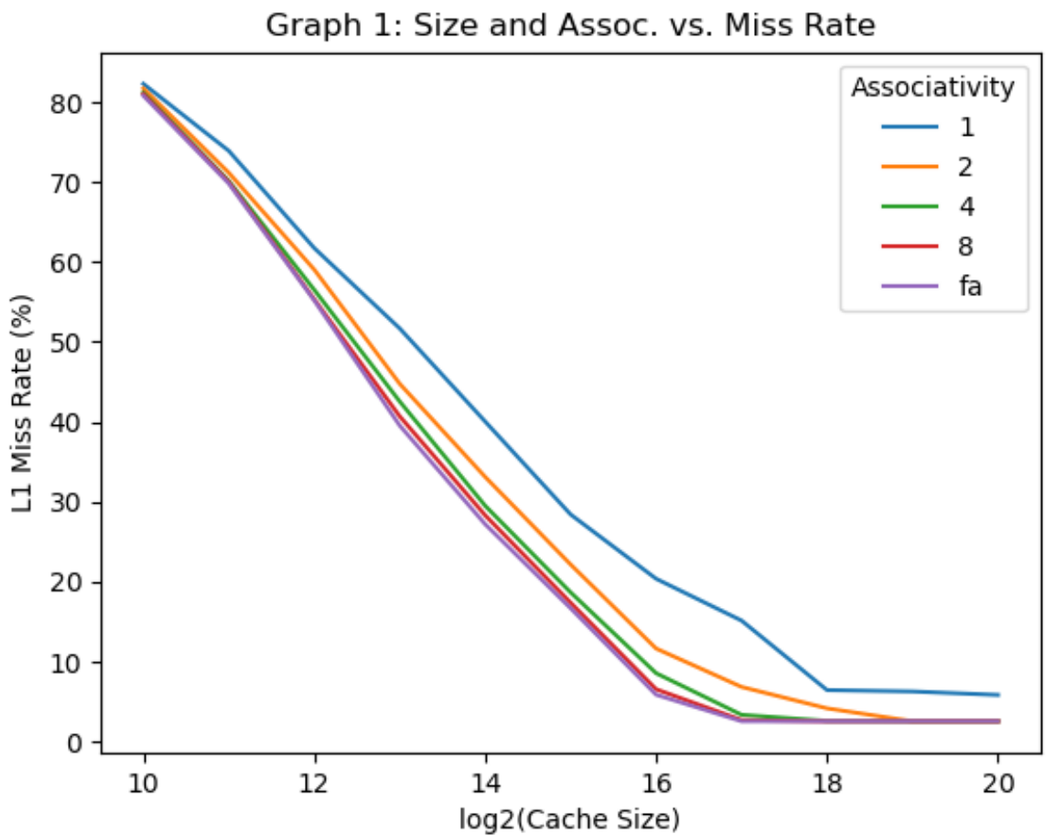
University of Central Florida

Department of Computer Science

CDA5106: Advanced Computer Architecture

Fall 2021

1 L1 Cache Exploration: Size and Associativity



For this experiment, L1 size varies between 2^{10}B and 2^{20}B (1kB and 1MB, respectively), while associativity varies between 1-8, and additionally fully-associative (which is $\frac{size}{blocksize}$).

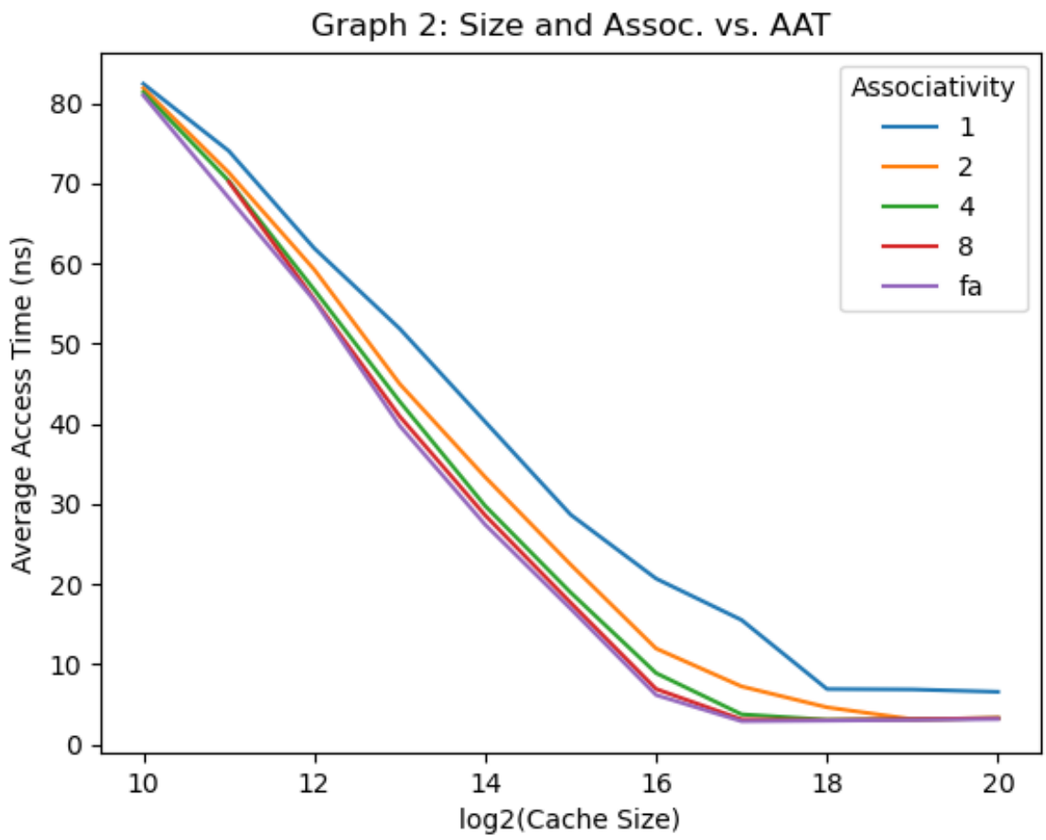
Several trends appear in the graph, exposing the relationship between cache size, associativity, and miss rate:

1. As cache size increases, L1 miss rate decreases universally (regardless of associativity).
2. Additionally, increasing cache associativity also decreases L1 miss rate, though not as drastically as a cache size increase would (note that, at any fixed cache size, the variance in miss rate between the different associativity values is noticeable, but still very slight at times).
3. Increasing associativity yields diminishing returns; note that, as cache size increases, the difference in associativity between assoc-8 and assoc-FA will increase. However, on the right half of the graph, it's easy to see that the difference in miss rate between an 8-way set-associative cache and a fully-associative one is minimal.

Miss rate estimations:

- **compulsory misses:** These are inevitable misses, resulting from having to fetch blocks from memory the first time they're needed in the cache. I estimate an average of between 0-3% of all misses are compulsory, based on the performance of a fully-associative cache (which has the lowest conflict miss rate), since these are misses that would occur even in a "perfect" and infinitely-sized cache with full associativity and LRU replacement. I arrived at this range because it seems to be the miss rate that the cache would converge to as cache size and associativity tended toward infinity, indicating that a miss rate of <3% is a lower bound (which makes sense because these misses are inevitable).
- **conflict misses:** These are caused by collisions or conflicts in which there are more blocks to be placed in a set than the set can hold (thus bottlenecked by cache associativity). Based on this, it makes sense that a fully-associative cache with LRU replacement would have a low (zero) conflict miss rate, while a 1-way set-associative cache would have the highest conflict miss rate. Below are some rough guesses based on the graph and intuition (such as difference in miss rate between an assoc. A_i and it's next largest assoc. A_{i+1}, inherent correlation between associativity and conflict miss rate, etc.), but keep in mind that cache size also has an effect on conflict miss rate in tandem with associativity (so these miss rates will vary):
  - **1-way:** $\sim 5 - 7\%$
  - **2-way:** $\sim 3\%$
  - **4-way:** $\sim 2\%$
  - **8-way:** $\leq 1\%$
  - **fully-associative:** 0%

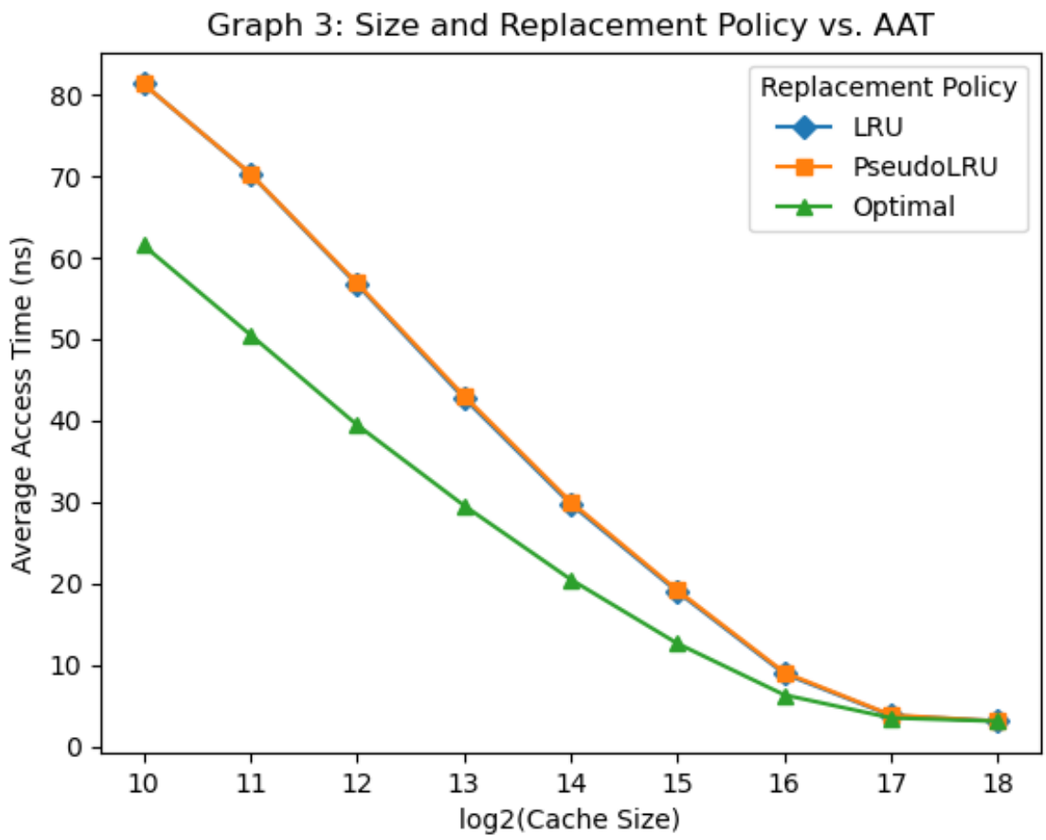
2 L1 Cache Exploration: Size and Associativity vs. AAT



In this experiment, L1 cache size and associativity vary in the same ranges as described in experiment 1, but here average access time (AAT) is measured instead of L1 miss rate (some points excluded, like $x = 10$ with assoc. $= 8$, as such points where not included in the CACTI table). Note the similarity in pattern/trend in this graph compared to the graph in experiment 1.

Based on the graph's trends, the best possible cache configuration to choose given the parameters plotted (assuming others, like block size, staying fixed), would be the fully-associative cache with a size between 2^{17} and 2^{20} bytes (depending on whether the AAT starts increasing again, which is hard to tell from the plot alone). However, an 8-way set associative cache (with a size lying in that same interval) is almost as good and would serve as a decent alternative candidate given the weaknesses of a fully-associative cache.

3 Replacement Policy Study

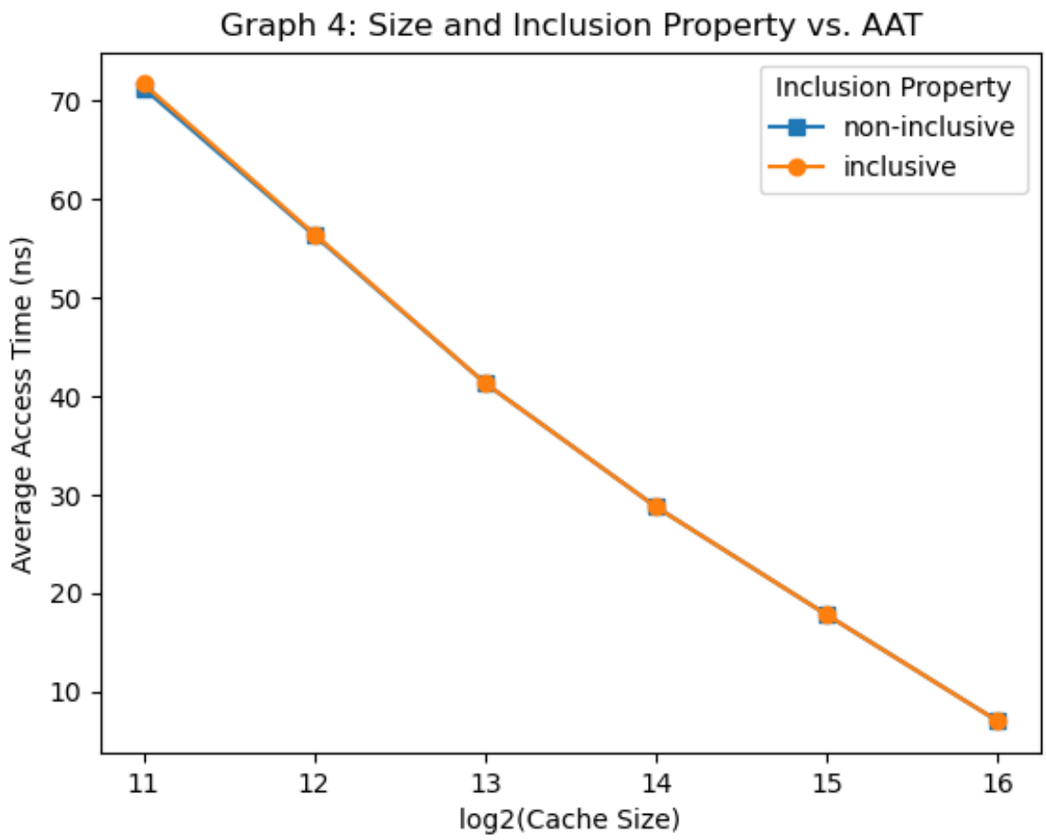


In this experiment, L1 cache size varies between 2^{10}B and 2^{18}B (1kB and 256kB, respectively), while the replacement policy varies between LRU, PseudoLRU, and optimal.

Of course, the best-performing cache is one with the largest size (256kB) that uses the optimal replacement policy. This makes sense because this policy guarantees the replacement of the block that will not be needed for the longest period of time (or never again), so less time is spent re-fetching blocks from memory in case an evicted block needed to be in the cache again soon after its eviction. However, since optimal replacement is impossible to implement in practice, I'd also like to compare LRU and PseudoLRU, which are both practical.

While they appear to yield exactly equal performance, when viewing my simulation stats, I did notice a slightly-worse performance in PseudoLRU (to confirm, feel free to click here to view my simulation logs once the repository is public). However, considering the fact that LRU is more memory-efficient in practice, and can still approximate LRU almost unnoticeably-closely, it could be a better candidate between the two.

4 Inclusion Property Study



In this experiment, cache size varies between 2^{11}B and 2^{16}B (2kB and 64kB, respectively), while the inclusion property is either non-inclusive or inclusive.

Interestingly enough, the inclusivity of the caches does not seem to matter much in this benchmark (between a cache that is inclusive and one that is non-inclusive). Only in the smallest cache size does the non-inclusive cache yield a lower AAT, but afterwards, they both seem to exactly match (unnoticeably small differences, if any).

Perhaps the performance between inclusive/non-inclusive caches are close due to lucky access order in the gcc benchmark, but I do not imagine this same trend would hold if they were both compared to an exclusive outer cache (not tested here). That is because there will be less cache coherency (since each cache level has unique items), but better memory utilization (no redundancy) and *potentially* worse (higher) access time. This last point will be due to the fact that inclusive caches can eliminate the need to search entire cache levels (i.e. if a block is not found in the outermost cache, we do not need to search in any of the inner caches), while exclusive and non-inclusive caches do not necessarily have this luxury.