NC State University

Department of Electrical and Computer Engineering

ECE 463/563: Fall 2022 (Rotenberg)

Project #1: Cache Design, Memory Hierarchy Design

by

Sounder Rajendran

NCSU Honor Pledge: "I have neither given nor received unauthorized aid on this project."

Student's electronic signature: Sounder Rajendran
(sign by typing your name)

Course number: 563
(463 or 563 ?)

Report template for Project #1.

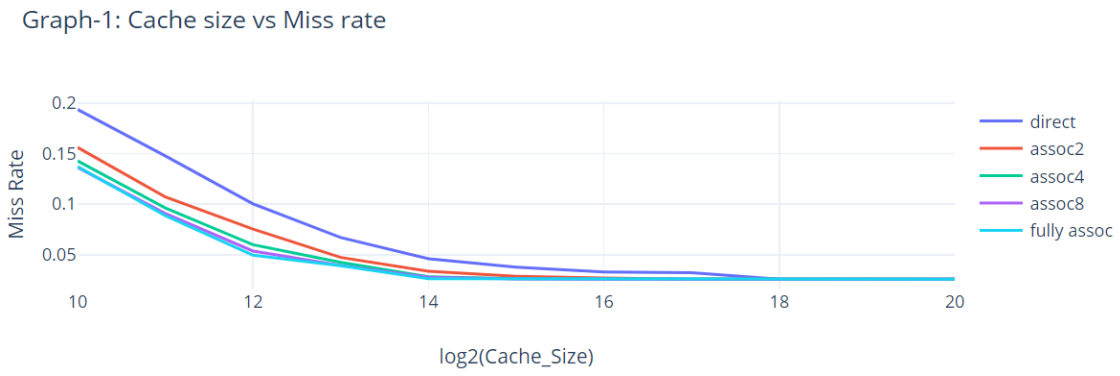
9.1. L1 cache exploration: SIZE and ASSOC

GRAPH #1 (***total number of simulations: 55***)

For this experiment:

- Benchmark trace: gcc_trace.txt
- L1 cache: SIZE is varied, ASSOC is varied, BLOCKSIZE = 32.
- L2 cache: None.
- Prefetching: None.

Plot L1 miss rate on the y-axis versus $\log_2$(SIZE) on the x-axis, for eleven different cache sizes: SIZE = 1KB, 2KB, … , 1MB, in powers-of-two. (That is, $\log_2$(SIZE) = 10, 11, …, 20.) The graph should contain five separate curves (*i.e.*, lines connecting points), one for each of the following associativities: direct-mapped, 2-way set-associative, 4-way set-associative, 8-way set-associative, and fully-associative. All points for direct-mapped caches should be connected with a line, all points for 2-way set-associative caches should be connected with a line, *etc.*



Answer the following questions:

1. For a given associativity, how does increasing cache size affect miss rate?

Increasing cache size with fixed associativity and block size, decreases miss rate upto cache size 262144 bytes (log val = 18). After that, even when cache size increases, miss rate stays almost the same.

2. For a given cache size, how does increasing associativity affect miss rate?

For a given cache size and block size, increasing associativity decreases miss rate until size = 65536 bytes. After that, increasing assoc doesn't have any effect on miss rate.

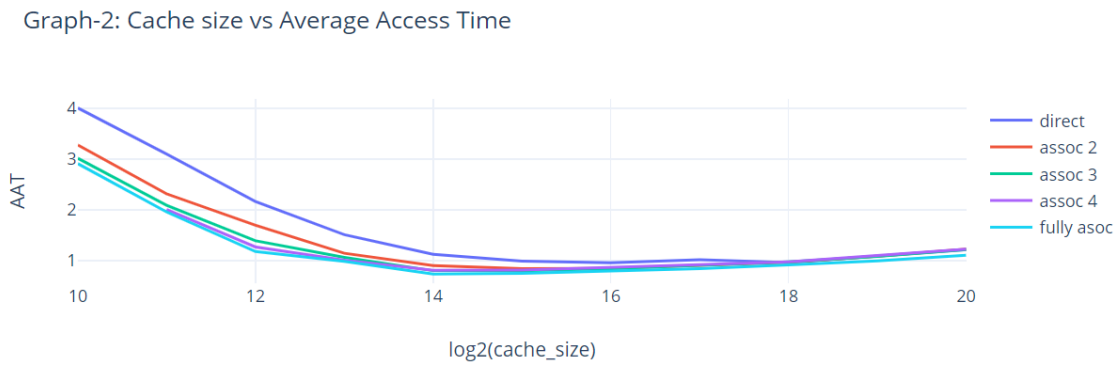
3. Estimate the *compulsory miss rate* from the graph and briefly explain how you arrived at this estimate.

compulsory miss rate = 0.0258 How I arrived at this estimate: *Starting from cache size=131072bytesz cache size becomes large enough to eliminate capacity and conflict misses. This is observed as all*

miss rate values beyond this point, are same value: 0.0258. At this point, we can safely say all conflict and capacity misses have been eliminated. Remaining miss rate is because of compulsory misses. Therefore compulsory miss rate is 0.0258.

GRAPH #2 **(*no additional simulations with respect to GRAPH #1*)**

Same as GRAPH #1, but the y-axis should be AAT instead of L1 miss rate.



Answer the following question:

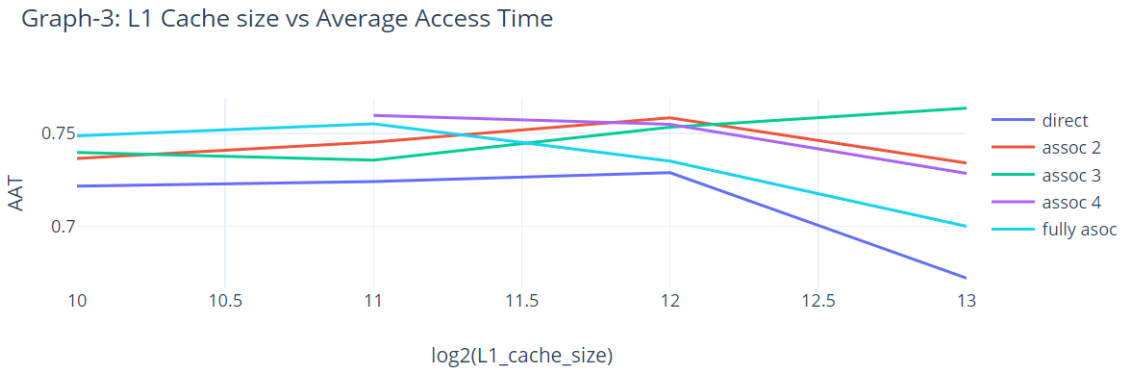
1. For a memory hierarchy with only an L1 cache and BLOCKSIZE = 32, which configuration yields the best (*i.e.*, lowest) AAT and what is that AAT?

Configuration that yields the lowest AAT: Assoc: 512; Cache Size: 16384; Block size: 32 Lowest AAT: 0.734238 ns

GRAPH #3 **(*total number of simulations: 20*)**

Same as GRAPH #2, except make the following changes:

- Add the following L2 cache to the memory hierarchy: 16KB, 8-way set-associative, same block size as L1 cache.
- Vary the L1 cache size only between 1KB and 8KB (since L2 cache is 16KB).



Answer the following questions:

1. With the L2 cache added to the system, which L1 cache configuration yields the best (*i.e.*, lowest) AAT and what is that AAT?

L1 configuration that yields the lowest AAT with 16KB 8-way L2 added: Assoc: 1; cache size: 8192 bytes
Lowest AAT: 0.672282548

2. How does the lowest AAT with L2 cache (GRAPH #3) compare with the lowest AAT without L2 cache (GRAPH #2)?

The lowest AAT with L2 cache is 0.061955452 ns less than the lowest AAT without L2 cache.

3. Compare the *total area* required for the lowest-AAT configurations with L2 cache (GRAPH #3) versus without L2 cache (GRAPH #2).

Total area for lowest-AAT configuration with L2 cache =
0.053293238 mm^2 (L1 area) + 0.130444675 mm^2 (L2 area) = 0.183737913 mm^2 (total area)

Total area for lowest-AAT configuration without L2 cache = 0.063446019 mm^2 (L1 area)

The total area of the lowest-AAT configuration with L2 cache is 189.59722% more than the total area of the lowest-AAT configuration without L2 cache.
FYI: How to calculate % difference of x with respect to y:
If x > y: x is ((x-y)/y * 100%) more than y.
If x < y: x is ((y-x)/y * 100%) less than y.

Report template for Project #1.

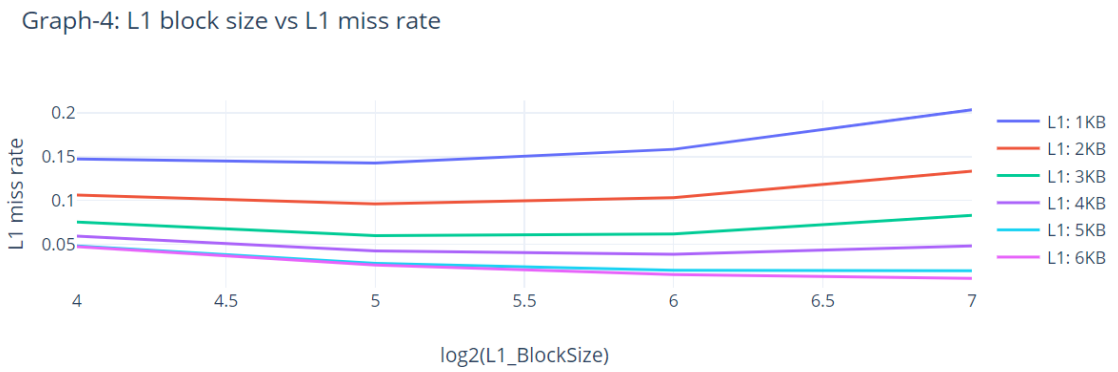
9.2. L1 cache exploration: SIZE and BLOCKSIZE

GRAPH #4 **(*total number of simulations: 24*)**

For this experiment:

- Benchmark trace: gcc_trace.txt
- L1 cache: SIZE is varied, BLOCKSIZE is varied, ASSOC = 4.
- L2 cache: None.
- Prefetching: None

Plot L1 miss rate on the y-axis versus $\log_2$(BLOCKSIZE) on the x-axis, for four different block sizes: BLOCKSIZE = 16, 32, 64, and 128. (That is, $\log_2$(BLOCKSIZE) = 4, 5, 6, and 7.) The graph should contain six separate curves (*i.e.*, lines connecting points), one for each of the following L1 cache sizes: SIZE = 1KB, 2KB, …, 32KB, in powers-of-two. All points for SIZE = 1KB should be connected with a line, all points for SIZE = 2KB should be connected with a line, *etc*.



Answer the following questions:

1. Do smaller caches prefer smaller or larger block sizes?

Smaller caches prefer smaller block sizes. For example, the smallest cache considered in Graph #4 (1KB) achieves its lowest miss rate with a block size of 32 B.

2. Do larger caches prefer smaller or larger block sizes?

Larger caches prefer larger block sizes. For example, the largest cache considered in Graph #4 (32KB) achieves its lowest miss rate with a block size of 128 B.

3. As block size is increased from 16 to 128, is the tension between *exploiting more spatial locality* and *cache pollution* evident in the graph? Explain.

Yes, the tension between *exploiting more spatial locality* and *cache pollution* is evident in the graph.

For example, consider the smallest (1KB) cache in Graph #4. Increasing block size from 16 B to 32 B is helpful (reduces miss rate) due to exploiting more spatial locality. But then increasing block size further, from 32 B to 128 B, is not helpful (increases miss rate) due to cache pollution having greater effect.

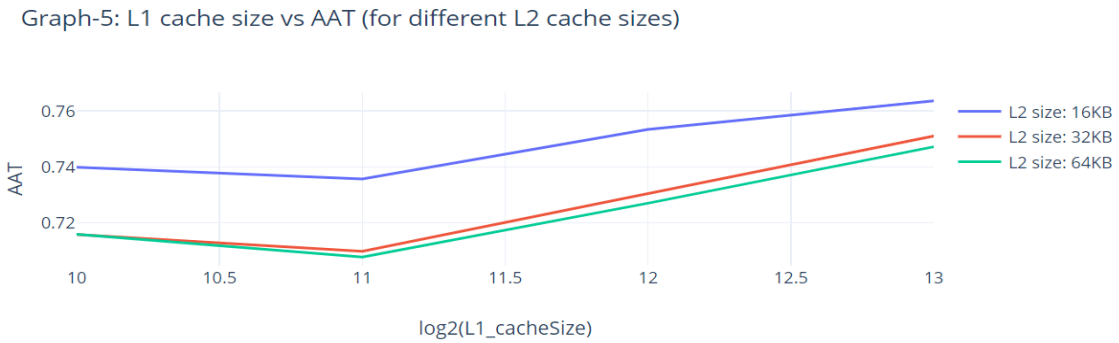
9.3. L1 + L2 co-exploration

GRAPH #5 **(*total number of simulations: 12*)**

For this experiment:

- Benchmark trace: gcc_trace.txt
- L1 cache: SIZE is varied, BLOCKSIZE = 32, ASSOC = 4.
- L2 cache: SIZE is varied, BLOCKSIZE = 32, ASSOC = 8.
- Prefetching: None.

Plot AAT on the y-axis versus $\log_2$(L1 SIZE) on the x-axis, for four different L1 cache sizes: L1 SIZE = 1KB, 2KB, 4KB, 8KB. (That is, $\log_2$(L1 SIZE) = 10, 11, 12, 13.) The graph should contain three separate curves (*i.e.*, lines connecting points), one for each of the following L2 cache sizes: 16KB, 32KB, 64KB. All points for the 16KB L2 cache should be connected with a line, all points for the 32KB L2 cache should be connected with a line, *etc*.



Answer the following question:

1. Which memory hierarchy configuration in Graph #5 yields the best (*i.e.*, lowest) AAT and what is that AAT?

Configuration that yields the lowest AAT: L1: cache size: 2KB, Assoc: 4; L2: cache size: 64KB, Assoc:8; Lowest AAT: 0.707657832 ns

9.4. Stream buffers study (ECE 563 students only)

TABLE #1 **(*total number of simulations: 5*)**

For this experiment:

- Microbenchmark: stream_trace.txt
- L1 cache: SIZE = 1KB, ASSOC = 1, BLOCKSIZE = 16.
- L2 cache: None.
- PREF_N (number of stream buffers): 0 (pref. disabled), 1, 2, 3, 4
- PREF_M (number of blocks in each stream buffer): 4

The trace "stream_trace.txt" was generated from the loads and stores in the loop of interest of the following microbenchmark:

```
#define SIZE 1000

uint32_t a[SIZE];
uint32_t b[SIZE];
uint32_t c[SIZE];

int main(int argc, char *argv[]) {
…
   // LOOP OF INTEREST
   for (int i = 0; i < SIZE; i++)
      c[i] = a[i] + b[i];   // per iteration: 2 loads (a[i], b[i]) and 1 store (c[i] = …)
…
}
```

Fill in the following table and answer the following questions:

PREF_N, PREF_M	L1 miss rate
0,0 (pref. disabled)	0.25
1,4	0.25
2,4	0.25
3,4	0.001
4,4	0.001

1. For this streaming microbenchmark, with prefetching disabled, do L1 cache size and/or associativity affect the L1 miss rate (feel free to simulate L1 configurations besides the one used for the table)? Why or why not?

With prefetching disabled, L1 cache size and/or associativity do not affect L1 miss rate (for this streaming microbenchmark).

The reason: For 1000 values the for loop will run. So the CPU will request for 3 new address values every iteration. First if we miss all three addresses, we will fetch from memory and store in cache. Since for every new iteration we request new value, we will get compulsory misses for every 1 in 4 requests. Since increasing assoc and/or cache size decrease only capacity and conflict misses, we surely will get compulsory misses no matter what. Also cache would've gotten more hits had the code asked for reusable address blocks.

2. For this streaming microbenchmark, what is the L1 miss rate with prefetching disabled? Why is it that value, *i.e.*, what is causing it to be that value? Hint: each element of arrays a, b, and c, is 4 bytes (uint32_t).

The L1 miss rate with prefetching disabled is 0.25 because for all a[i], b[i], c[i] if we miss one block, we fetch 16 bytes of memory block to cache. Since it is continuous (array), it must be reusable. Thus after using our 4 bytes which was requested by CPU, we will hit on 3 next blocks requested by CPU for all a[i], b[i], c[i]. This means we have 1 in 4 requests as misses. That is 0.25 miss rate.

3. For this streaming microbenchmark, with prefetching disabled, what would the L1 miss rate be if you doubled the block size from 16B to 32B? (hypothesize what it will be and then check your hypothesis with a simulation)

The L1 miss rate with prefetching disabled and a block size of 32B is 0.125 because in this case we will hit twice as frequent as 16B block size as we would've fetched 7 extra address bytes that would be requested by CPU in the next iterations. This is called exploiting spatial locality.

4. With prefetching enabled, what is the minimum number of stream buffers required to have any effect on L1 miss rate? What is the effect on L1 miss rate when this many stream buffers are used: specifically, is it a modest effect or huge effect? Why are this many stream buffers required? Why is using fewer stream buffers futile? Why is using more stream buffers wasteful?

Minimum number of stream buffers needed to have any effect on L1 miss rate: 3

With this many stream buffers, the effect on L1 miss rate is huge. Specifically, the L1 miss rate is nearly 0. We only miss on the first elements of a, b and c (hence a total of 3 misses).

This many stream buffers are required because there are three variables that requests three separate array address (sequential) blocks.

Using fewer stream buffers is futile because CPU will stall while the remaining address blocks are fetched as the code runs sequentially.

Using more stream buffers is wasteful because there is only 3 sequential address are being used in the given code. Only three streams will be stored in their respective stream buffers.