

NC State University
Department of Electrical and Computer Engineering
ECE 463/563 (Prof. Rotenberg)
Project #1: Cache Design, Memory Hierarchy Design
REPORT TEMPLATE (Version 2.0)

by

MOHAMMED MUBEEN MIFTHAK

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

Student's electronic signature: **Mohammed Mubeen Mifthak**
(sign by typing your name)

Course number: **563**
(463 or 563 ?)

1. L1 cache exploration: SIZE and ASSOC

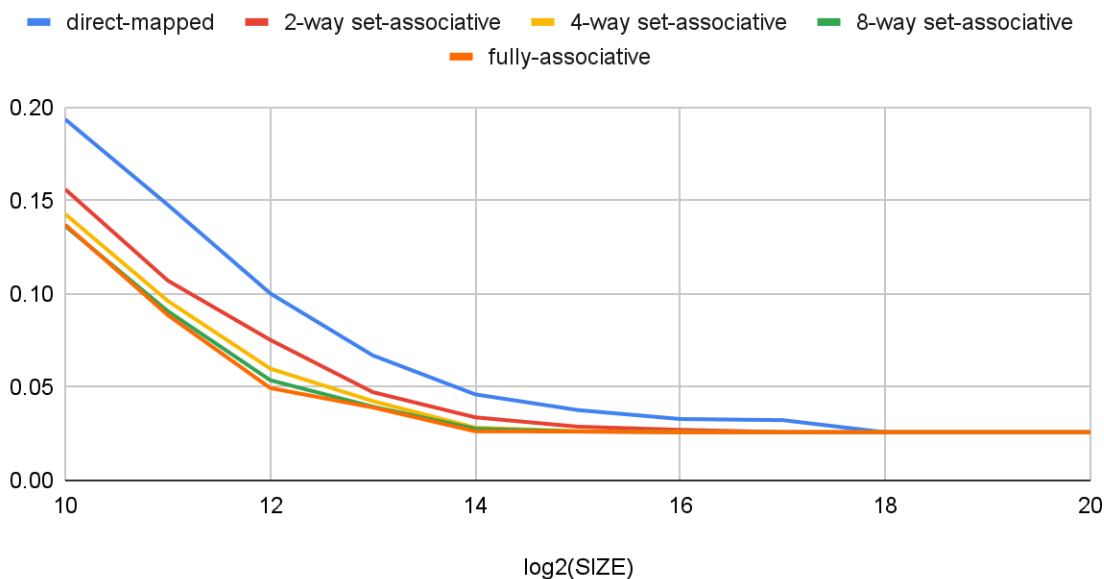
GRAPH #1 (total number of simulations: 55)

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 $\log_2(\text{SIZE})$ on the x-axis, for eleven different cache sizes: SIZE = 1KB, 2KB, ..., 1MB, in powers-of-two. (That is, $\log_2(\text{SIZE}) = 10, 11, \dots, 20$.) 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.*

Graph 1 - Fixed Block Size



Answer the following questions:

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

Miss rate decreases as cache size is increased and settles. This settlement point should be because of the compulsory misses. Beyond this point, further increase in cache size does not improve the miss rate.

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

Increasing associativity for a given cache size decreases the miss rate. But beyond the settlement point mentioned above, the change in associativity has no effect on the miss rate. (Refer to the end of X axis)

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

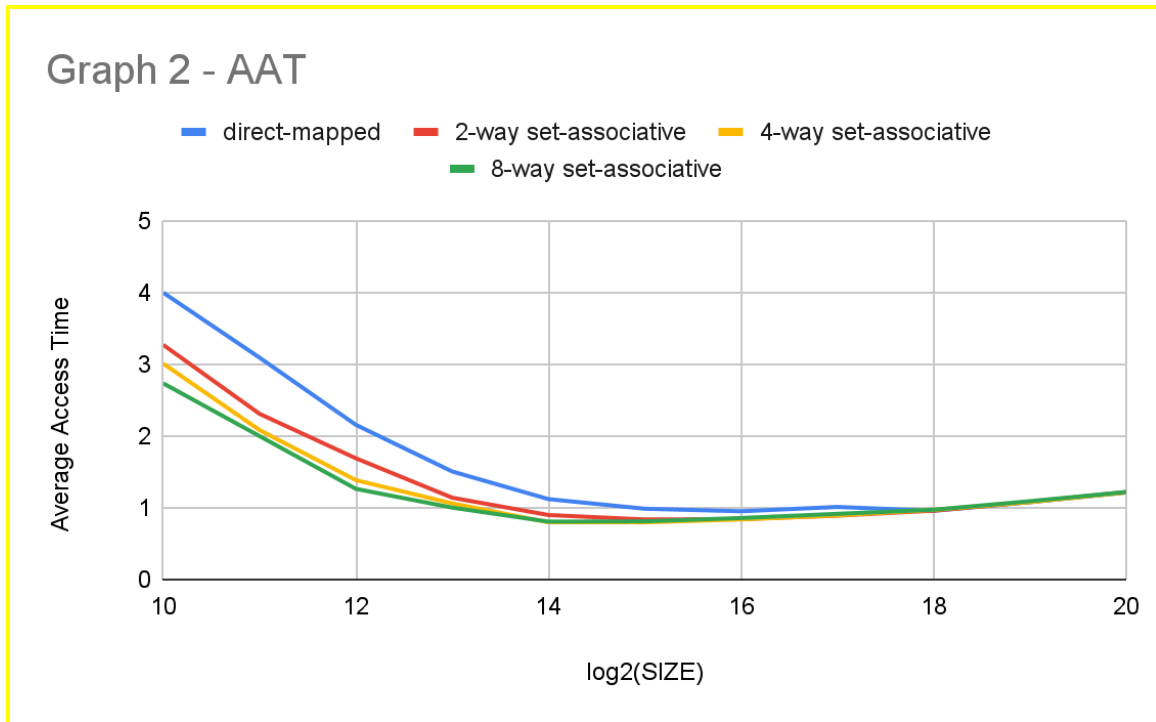
compulsory miss rate = 0.0258

How I arrived at this estimate: In the graph above, this is the value of the settlement point in the y-axis. This signifies compulsory misses, as with an extremely large cache size, our cache space becomes close to the memory address space. In this situation, the cache misses are the compulsory misses.

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

Same as GRAPH #1, except make the following changes:

- The y-axis should be AAT instead of L1 miss rate.
- Do NOT include fully-associative cache configurations (power-hungry, impractical cost for implementing LRU). Thus, GRAPH #2 should contain four (not five) separate curves: direct-mapped, 2-way set-associative, 4-way set-associative, 8-way set-associative.



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?

Direct-mapped or set-associative cache configuration that yields the lowest AAT: **32 KB,**

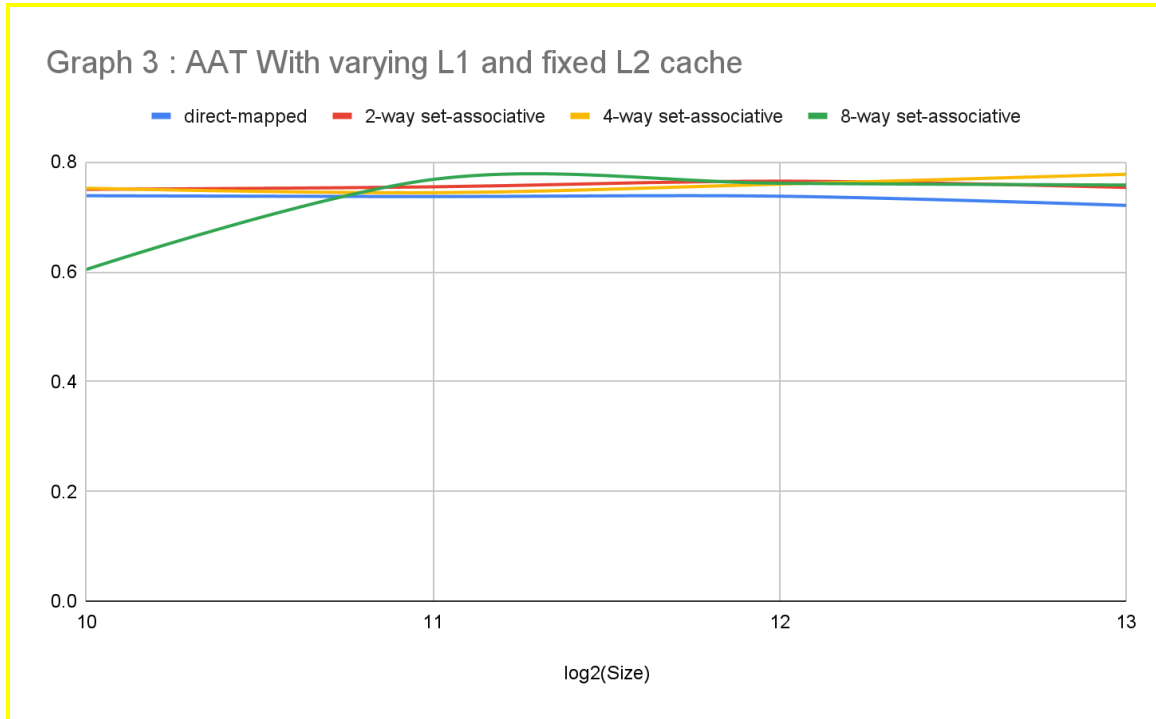
4-way set-associative

AAT for this configuration: **0.80189**

GRAPH #3 (total number of simulations: 16)

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). (And as with GRAPH #2, GRAPH #3 should contain four (not five) separate curves for L1 associativity: direct-mapped, 2-way set-associative, 4-way set-associative, 8-way set-associative.)



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:

1 KB, 8-way set-associative

AAT for this configuration: **0.604**

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 **less than** the lowest AAT without L2 cache.

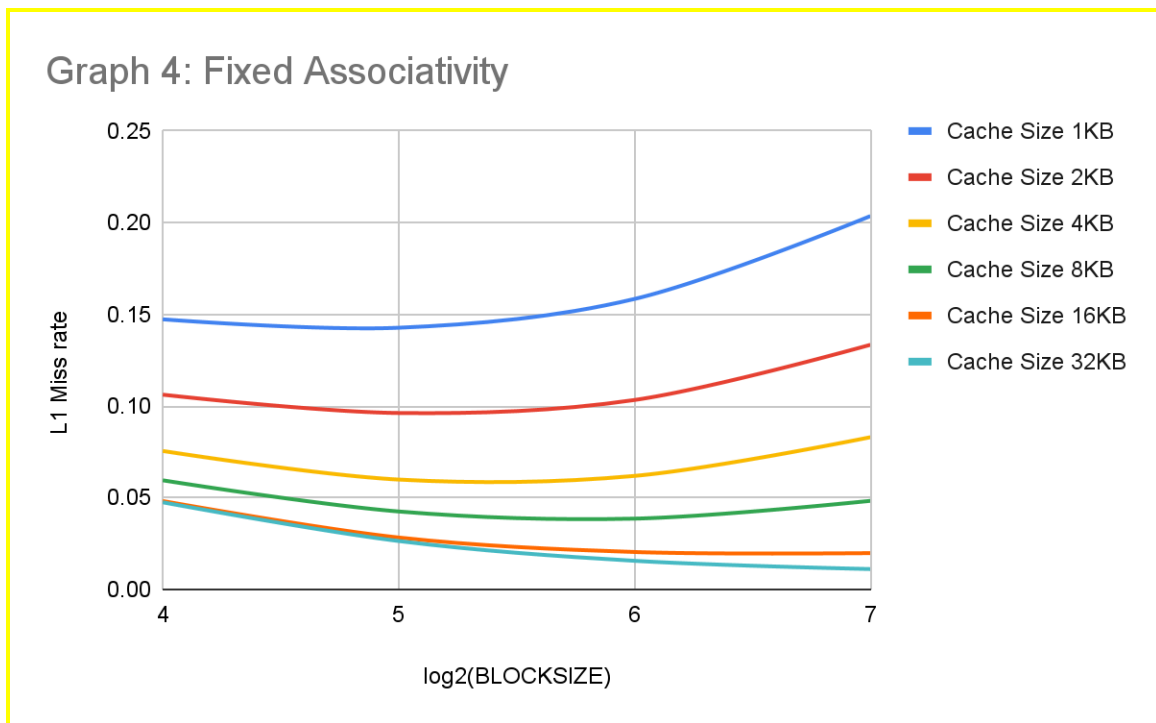
2. L1 cache exploration: SIZE and BLOCKSIZE

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 $\log_2(\text{BLOCKSIZE})$ on the x-axis, for four different block sizes: BLOCKSIZE = 16, 32, 64, and 128. (That is, $\log_2(\text{BLOCKSIZE}) = 4, 5, 6,$ 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.

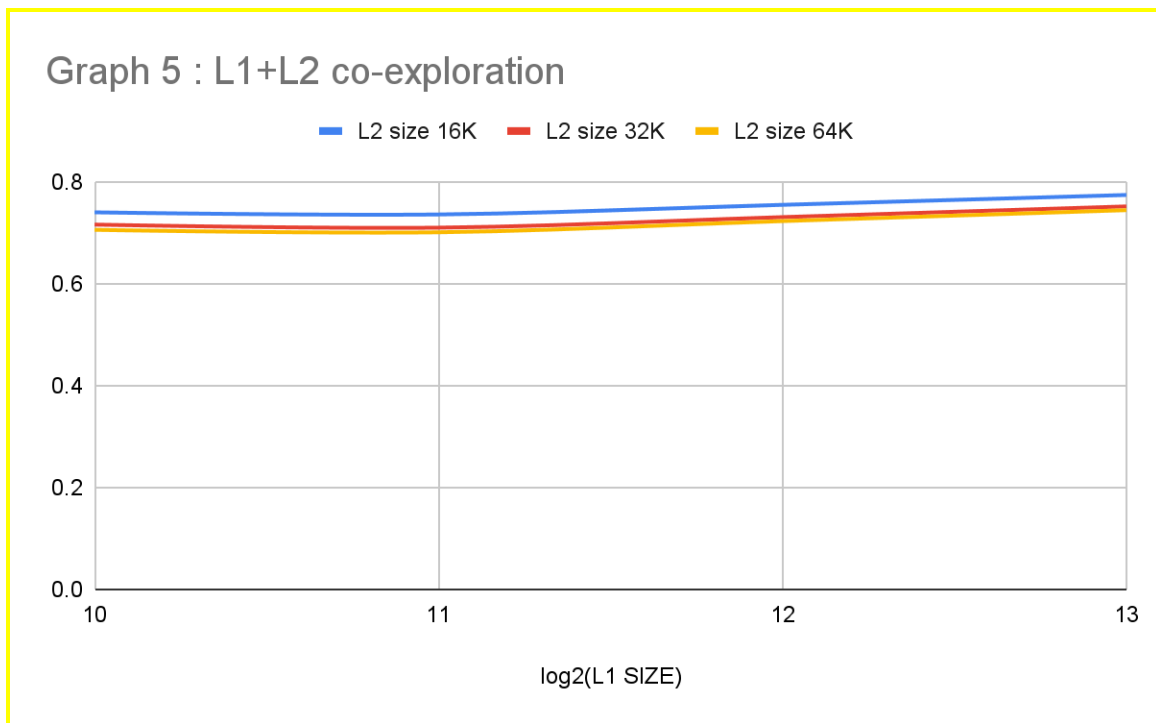
3. L1 + L2 co-exploration

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 $\log_2(\text{L1 SIZE})$ on the x-axis, for four different L1 cache sizes: L1 SIZE = 1KB, 2KB, 4KB, 8KB. (That is, $\log_2(\text{L1 SIZE}) = 10, 11, 12, 13$.) 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:

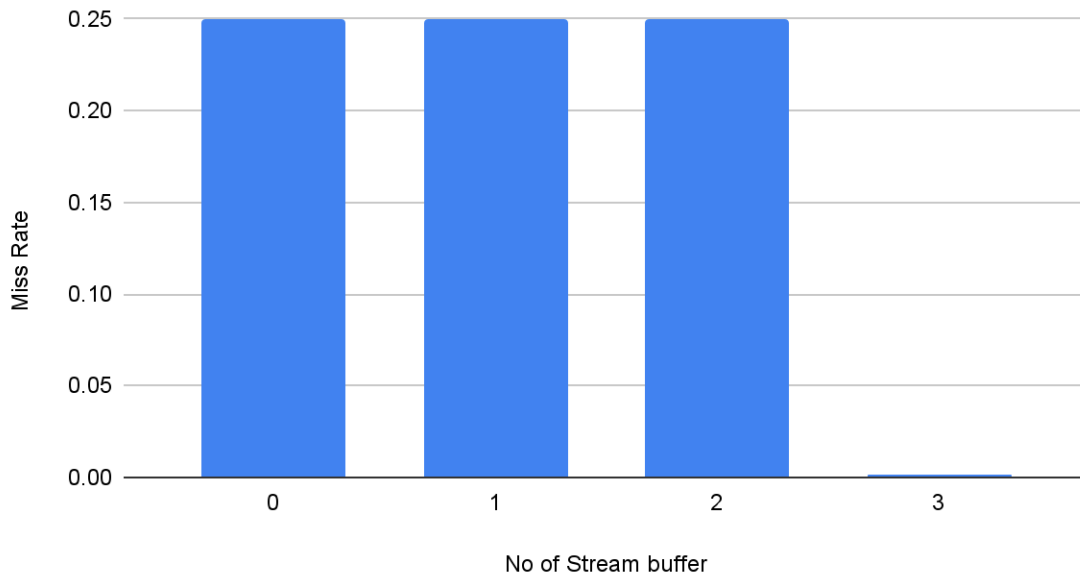
1. Which memory hierarchy configuration in Graph #5 yields the best (*i.e.*, lowest) AAT and what is that AAT?

Configuration that yields the lowest AAT: 2KB L1 cache + 64KB L2 cache
Lowest AAT: 0.7007

4. Stream buffers study (ECE 563 students only)

TABLE #1 (total number of simulations: 5)

Miss Rate vs. No of Stream buffer



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] = ...)
    ...
}
```

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: This is a sequential request of memory since the data is maintained in an array. These constitute **compulsory misses** as each of these blocks are fetched for the first time and they cannot be avoided in any cache. This is also the case with the ideal cache of fully-associative cache.

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 each block will fetch 4 elements of an array as the block size is 16 bytes and int is 4 bytes in this system under test. This block will take the free block available in the set. The 5th element will lead to another cache miss as the block is not available in the cache irrespective of the size or associativity of the cache. So for every 4th element access, we would suffer a cache miss. $\frac{1}{4} = 0.25$.

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 0.126 (Simulated value), because compared to before, now we have 8 elements in a block. So for every 8th element, we would suffer a compulsory miss, thus Miss rate = $\frac{1}{8} = 0.125$. The actual simulated result is close to this number because of the number of reads and writes not being a proper multiple of 8.

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 (99.6% improvement)**. Specifically, the L1 miss rate is nearly 0.001. We only miss on the first elements of each array, *ie*, a[0], b[0], c[0] (hence a total of 3 misses).

This many stream buffers are required because we have three arrays that are accessed in the loop, thus one stream buffer for a, b and c.

Using fewer stream buffers is futile because the stream would get flushed out once the next array is accessed. For example, in case of 2 buffer, buffer 1 and 2 gets filled with a and b. Once c is accessed, it replaces buffer 1. Now when a is accessed in the next iteration, it results in a miss. This would produce the same result as when without a stream buffer.

Using more stream buffers is wasteful because the other buffers beyond 3 will remain empty. This is because for each access of the elements we would always receive a buffer hit with buffer 1, 2 or 3, thus they would always keep getting resynchronized with the buffer stream. As a result the rest of the buffers would never be used.