NC State University

Department of Electrical and Computer Engineering

ECE 463/563 (Prof. Rotenberg)

Project #1: Cache Design, Memory Hierarchy Design

REPORT TEMPLATE (Version 1.0)

by

GAGANA SINDHU SABBAVARAPU

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

Student's electronic signature: GAGANA SINDHU SABBAVARAPU
(sign by typing your name)

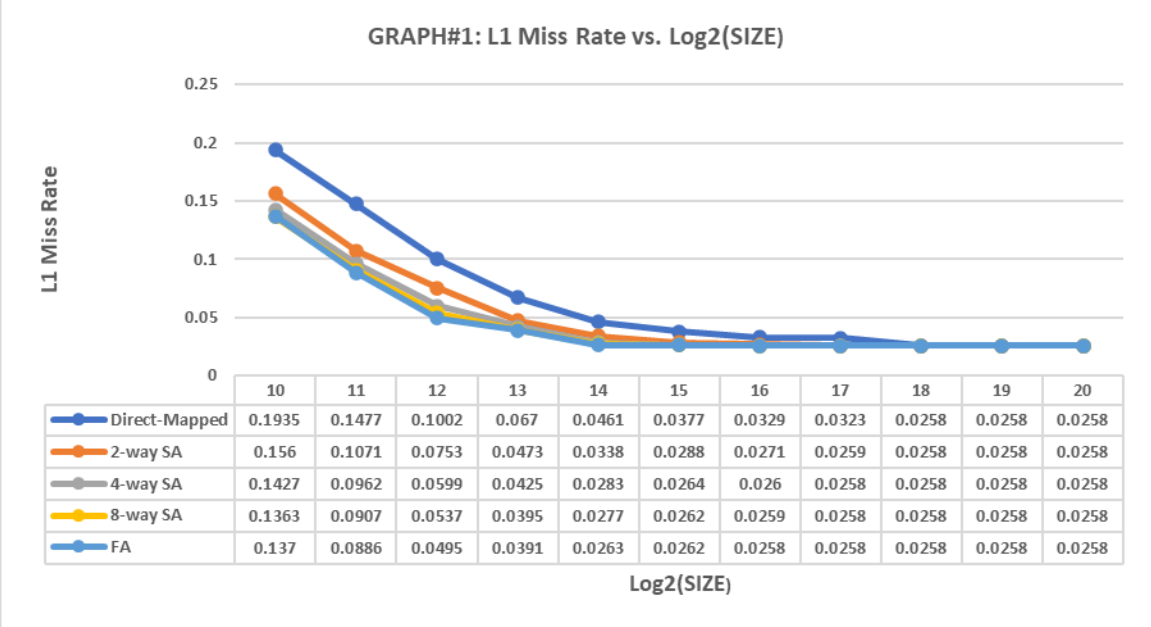
Course number: 563

1. L1 cache exploration:

Cache Configuration:

- Benchmark trace: gcc_trace.txt
- L1 cache:
  SIZE = 1KB, 2KB, 4KB, …, 1MB
  ASSOC = Direct-mapped, 2-way SA, 4-way SA, 8-way SA, Fully Associative
  BLOCKSIZE = 32
- L2 cache: None.
- Prefetching: None.

GRAPH #1: L1 Miss Rate vs. Log2(SIZE)



	10	11	12	13	14	15	16	17	18	19	20
Direct-Mapped	0.1935	0.1477	0.1002	0.067	0.0461	0.0377	0.0329	0.0323	0.0258	0.0258	0.0258
2-way SA	0.156	0.1071	0.0753	0.0473	0.0338	0.0288	0.0271	0.0259	0.0258	0.0258	0.0258
4-way SA	0.1427	0.0962	0.0599	0.0425	0.0283	0.0264	0.026	0.0258	0.0258	0.0258	0.0258
8-way SA	0.1363	0.0907	0.0537	0.0395	0.0277	0.0262	0.0259	0.0258	0.0258	0.0258	0.0258
FA	0.137	0.0886	0.0495	0.0391	0.0263	0.0262	0.0258	0.0258	0.0258	0.0258	0.0258

Answer the following questions:

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

As the cache size increases, we can notice a significant reduction in miss rates, which follows an almost exponential decrease. Specifically, for larger cache sizes, the miss rates for different cache associativities tend to converge to approximately the same value. This convergence occurs because, with a larger cache, issues related to conflict and capacity misses become negligible, leaving only the compulsory miss rate as a significant factor affecting the overall miss rate in the graph.

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

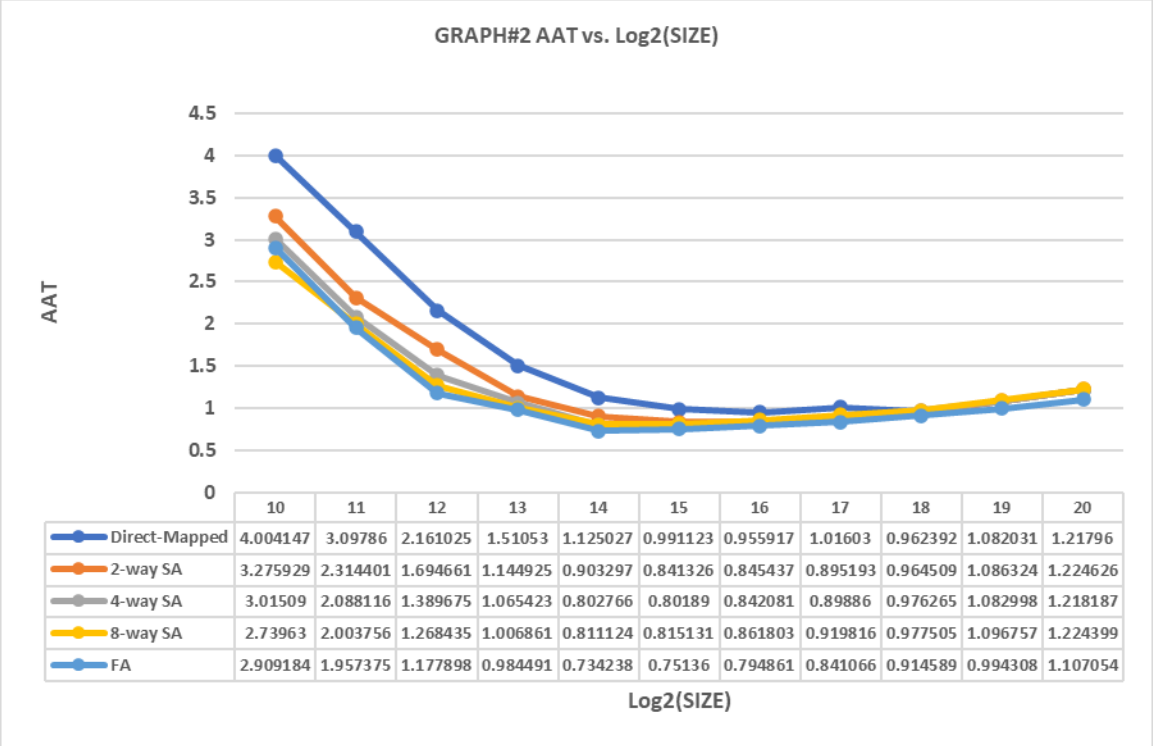
An increase in cache associativity leads to a reduction in miss rates for a given cache size. This improvement can be attributed primarily to the decrease in conflict-misses.

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: Compulsory misses are a consequence of referencing a cache block for the first time. In the context of a significantly large cache with full associativity, the observed miss rate closely approximates the compulsory miss rate. Based on the graph, we can estimate the compulsory miss rate to be approximately 0.0258.

GRAPH #2: AAT vs. Log2(SIZE)



	10	11	12	13	14	15	16	17	18	19	20
Direct-Mapped	4.004147	3.09786	2.161025	1.51053	1.125027	0.991123	0.955917	1.01603	0.962392	1.082031	1.21796
2-way SA	3.275929	2.314401	1.694661	1.144925	0.903297	0.841326	0.845437	0.895193	0.964509	1.086324	1.224626
4-way SA	3.01509	2.088116	1.389675	1.065423	0.802766	0.80189	0.842081	0.89886	0.976265	1.082998	1.218187
8-way SA	2.73963	2.003756	1.268435	1.006861	0.811124	0.815131	0.861803	0.919816	0.977505	1.096757	1.224399
FA	2.909184	1.957375	1.177898	0.984491	0.734238	0.75136	0.794861	0.841066	0.914589	0.994308	1.107054

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?

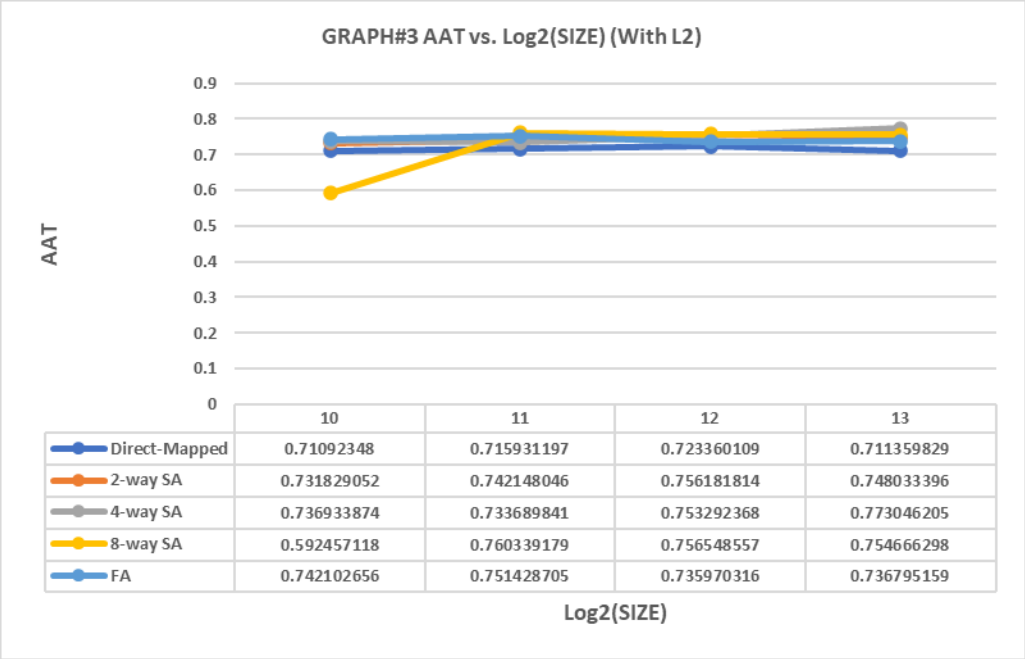
Configuration that yields the lowest AAT: From the Graph the configuration that yielded the lowest AAT is for a 16KB Fully Associative Cache. The Average Access Time (AAT) is influenced by the cache's hit time, miss rate, and a constant miss penalty. While the miss penalty remains consistent across configurations, the hit time and miss rate are contingent on both the cache's associativity and size. Typically, as the cache size increases, hit time also tends to rise, while the miss rate decreases with size expansion. Consequently, a trade-off exists between size and associativity, and it's with a 16KB cache and full associativity that we discover the optimal AAT value, balancing these factors effectively. In caches with full associativity, further increases in cache size result in an elevated AAT due to extended hit times.

Lowest AAT: 0.734238ns

GRAPH #3 AAT vs. Log2(SIZE) With L2 Cache

Cache Configuration:

- Benchmark trace: gcc_trace.txt
- L1 cache:
  SIZE = 1KB, 2KB, 4KB, …, 8KB
  ASSOC = Direct-mapped, 2-way SA, 4-way SA, 8-way SA, Fully Associative
  BLOCKSIZE = 32
- L2 cache:
  SIZE = 16KB
  ASSOC = 8-way SA
- Prefetching: None.



	10	11	12	13
Direct-Mapped	0.71092348	0.715931197	0.723360109	0.711359829
2-way SA	0.731829052	0.742148046	0.756181814	0.748033396
4-way SA	0.736933874	0.733689841	0.753292368	0.773046205
8-way SA	0.592457118	0.760339179	0.756548557	0.754666298
FA	0.742102656	0.751428705	0.735970316	0.736795159

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: 1KB Direct-Mapped
Lowest AAT: 0.7109ns

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 0.0234 ns << less than the lowest AAT without L2 cache. The inclusion of an L2 cache within the hierarchy effectively mitigates the miss penalty, consequently resulting in a reduction of the Average Access Time (AAT).

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 = 0.010298465744 mm^2 (L1 area) + 0.130444674885 mm^2 (L2 area) = 0.140743140629 mm^2 (total area)

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

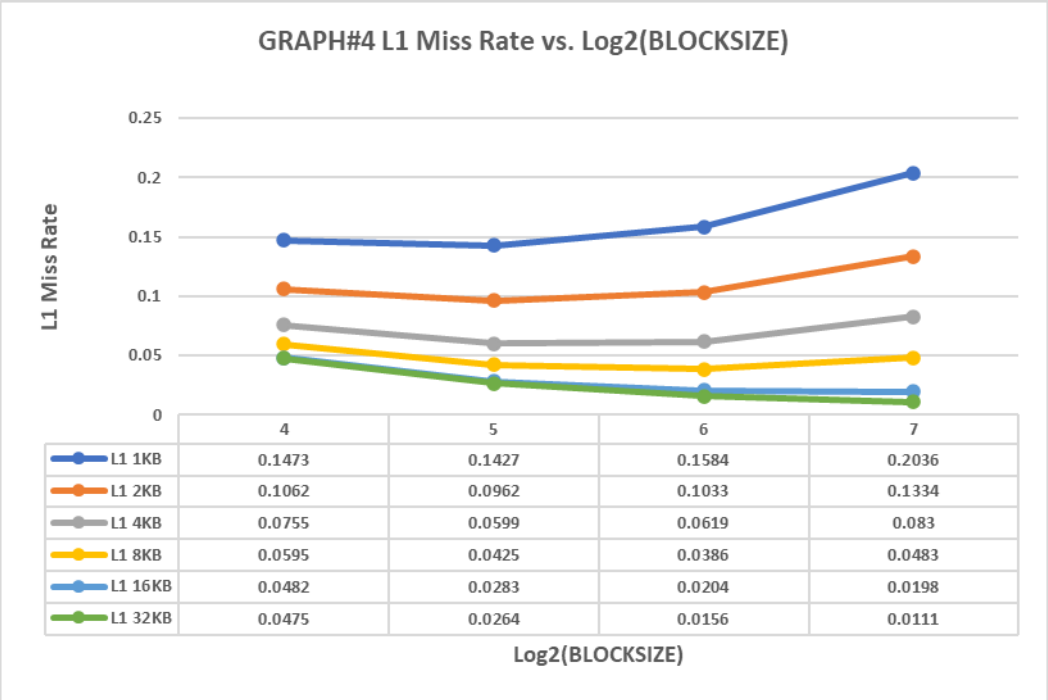
The total area of the lowest-AAT configuration with L2 cache is 121.97% more than the total area of the lowest-AAT configuration without L2 cache.

2. L1 cache exploration: SIZE and BLOCKSIZE

GRAPH #4: L1 Miss Rate vs. log2(BLOCKSIZE)

Cache Configuration:

- Benchmark trace: gcc_trace.txt
- L1 cache:
  SIZE = 1KB, 2KB, 4KB, 8KB, 16KB, 32KB
  BLOCKSIZE = 16, 32, 64, 128
  ASSOC = 4.
- L2 cache: None.
- Prefetching: None



	4	5	6	7
L1 1KB	0.1473	0.1427	0.1584	0.2036
L1 2KB	0.1062	0.0962	0.1033	0.1334
L1 4KB	0.0755	0.0599	0.0619	0.083
L1 8KB	0.0595	0.0425	0.0386	0.0483
L1 16KB	0.0482	0.0283	0.0204	0.0198
L1 32KB	0.0475	0.0264	0.0156	0.0111

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 32B.

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 128B.

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.

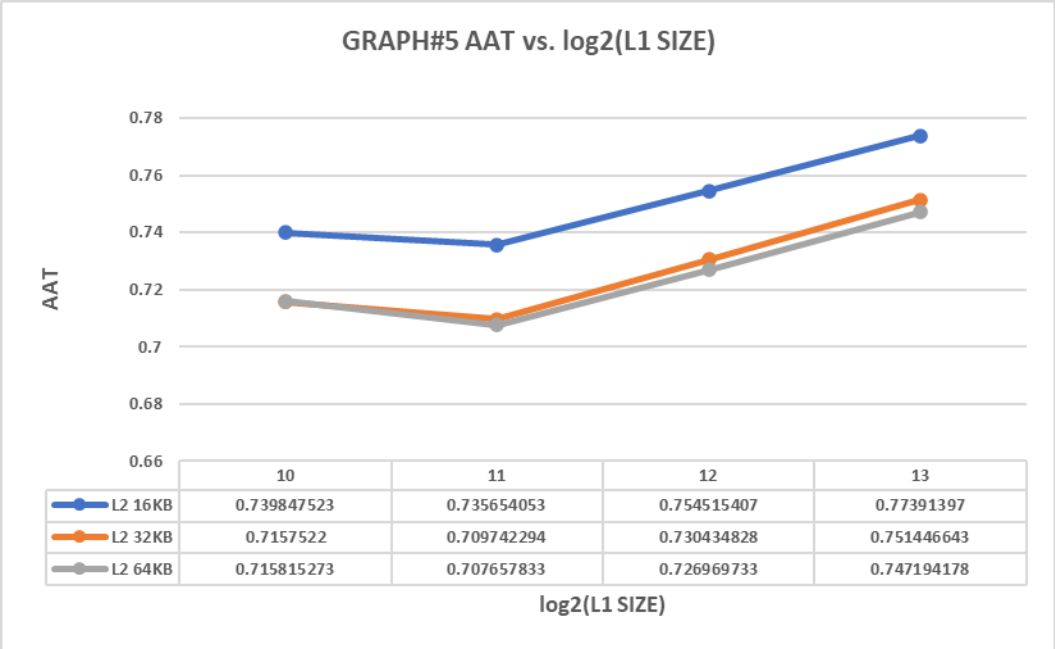
For example, consider the smallest (1KB) cache in Graph #4. Increasing block size from 16B to 32B is helpful (reduces miss rate) due to exploiting more spatial locality But then increasing block size further, from 32B to 64B, is not helpful (increases miss rate) due to cache pollution having greater effect.

3. L1 + L2 co-exploration

GRAPH #5: AAT vs. log2(L1 SIZE)

Cache Configuration:

- Benchmark trace: gcc_trace.txt
- L1 cache:
  SIZE = 1KB, 2KB, 4KB, 8KB
  BLOCKSIZE = 32, ASSOC = 4.
- L2 cache: SIZE = 16KB, 32KB, 64KB
  BLOCKSIZE = 32, ASSOC = 8.
- Prefetching: None.



	10	11	12	13
L2 16KB	0.739847523	0.735654053	0.754515407	0.77391397
L2 32KB	0.7157522	0.709742294	0.730434828	0.751446643
L2 64KB	0.715815273	0.707657833	0.726969733	0.747194178

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 4-way set associative L1 cache with a 64KB 8-way set associative L2 cache both with a Block Size of 32B.
Lowest AAT: 0.7077ns

4. Stream buffers study (ECE 563 students only)

TABLE #1

Cache Configuration:

- Microbenchmark: streams_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)	0.2500
1,4	0.2500
2,4	0.2500
3,4	0.0010
4,4	0.0010

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: The L1 miss rate in this streaming microbenchmark is primarily attributed to compulsory misses, which arise when blocks are accessed for the first time. These types of misses cannot be mitigated by adjusting cache size or associativity alone. The sole effective solution to address compulsory misses is the implementation of prefetching mechanisms, either through hardware or software prefetchers. Consequently, irrespective of cache size or associativity configurations, this specific benchmark will consistently experience the same volume of misses in the absence of a prefetcher.

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. This is attributed to the fact that, with a block size of 16 bytes, the cache fetches 16 consecutive bytes upon the initial reference to a memory location, ensuring no misses for the following three blocks. Hence, for every sequence of four consecutive references, a single miss occurs. Given that this streaming microbenchmark encompasses references to a total of 3000 consecutive memory locations, it accumulates a total of 750 misses.

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. This decrease is attributed to the larger block size, which allows the cache to fetch 32 consecutive bytes upon the initial memory location reference. As a result, it avoids misses for the following 7 blocks, ultimately reducing the total number of misses when compared to the 16B block size.

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 Zero (0.0010). We only miss on the first elements of the arrays a, b, c from the program (hence a total of 3 misses).

This many stream buffers are required because different streams, such as a[], b[], and c[], need to be effectively managed without interfering with each other. Having separate stream buffers for each stream ensures that data from one stream does not displace or evict data from another stream, preventing unnecessary demand misses and improving overall cache performance. Hence, we need minimum of 3 stream buffers to maintain 3 different streams.

Using fewer stream buffers is futile because the distinct streams consistently displace each other in a sequential manner, resulting in a situation where the count of demand misses is on par with the scenario where no stream buffer is employed.

Using more stream buffers is wasteful because this particular streaming microbenchmark doesn't make use of more than 3 stream buffers. The optimal miss rate has already been achieved with the existing 3 stream buffers.