Project 1: Cache Simulator

Cory Brynds

cory.brynds@ucf.edu

EEL4768: Computer Architecture
Section 0002

Due: 2 November 2023
Submitted: 2 November 2023

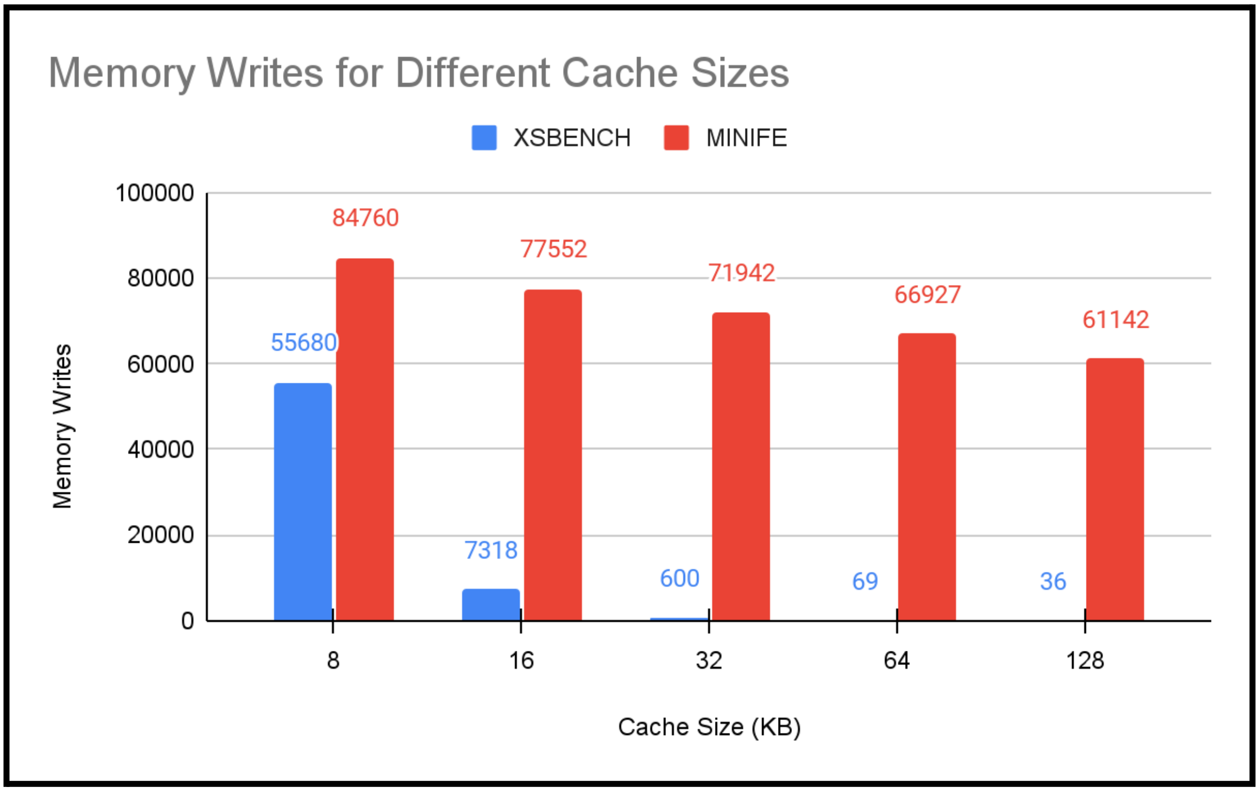
1.0 Introduction

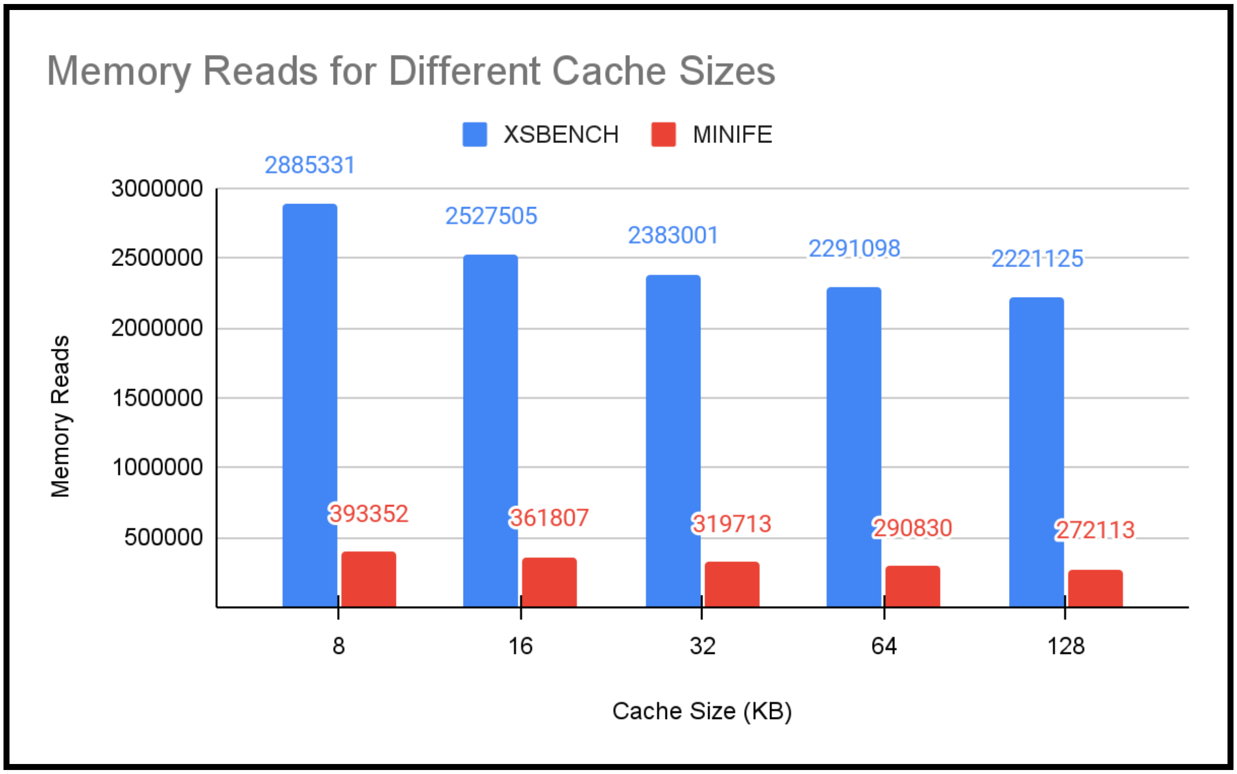
In this project, an L1 cache was implemented using C++ code that could simulate various cache configurations for size, associativity, write-back policy, and replacement policy given a certain command-line input. Taking an additional trace file path as input, the overall number of memory reads and writes, as well as the miss rate, of a given cache configuration could be tested. In the following section, the effects of altering these cache parameters will be observed, and a conclusion will be drawn from these comparisons on the optimal cache configuration.

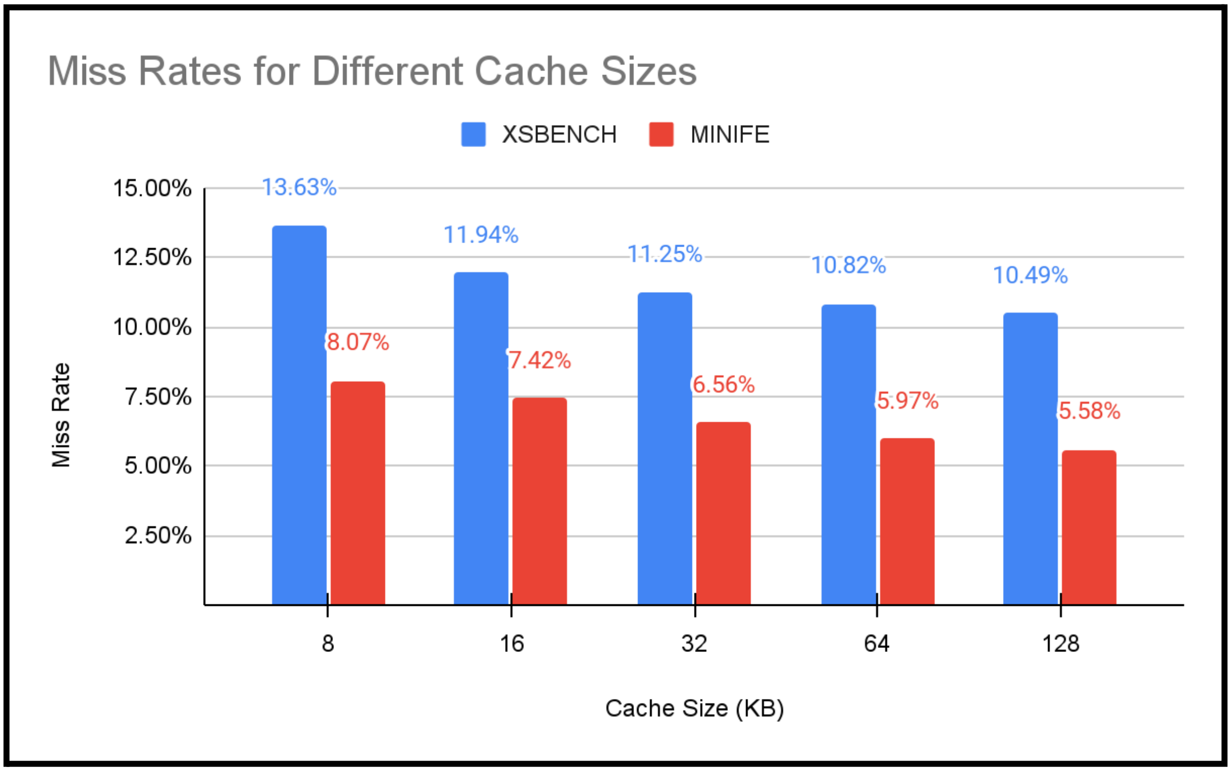
2.0 Performance Evaluation

2.1 Cache Size Comparison

Default cache settings: 4-set associativity, write-back, LRU replacement



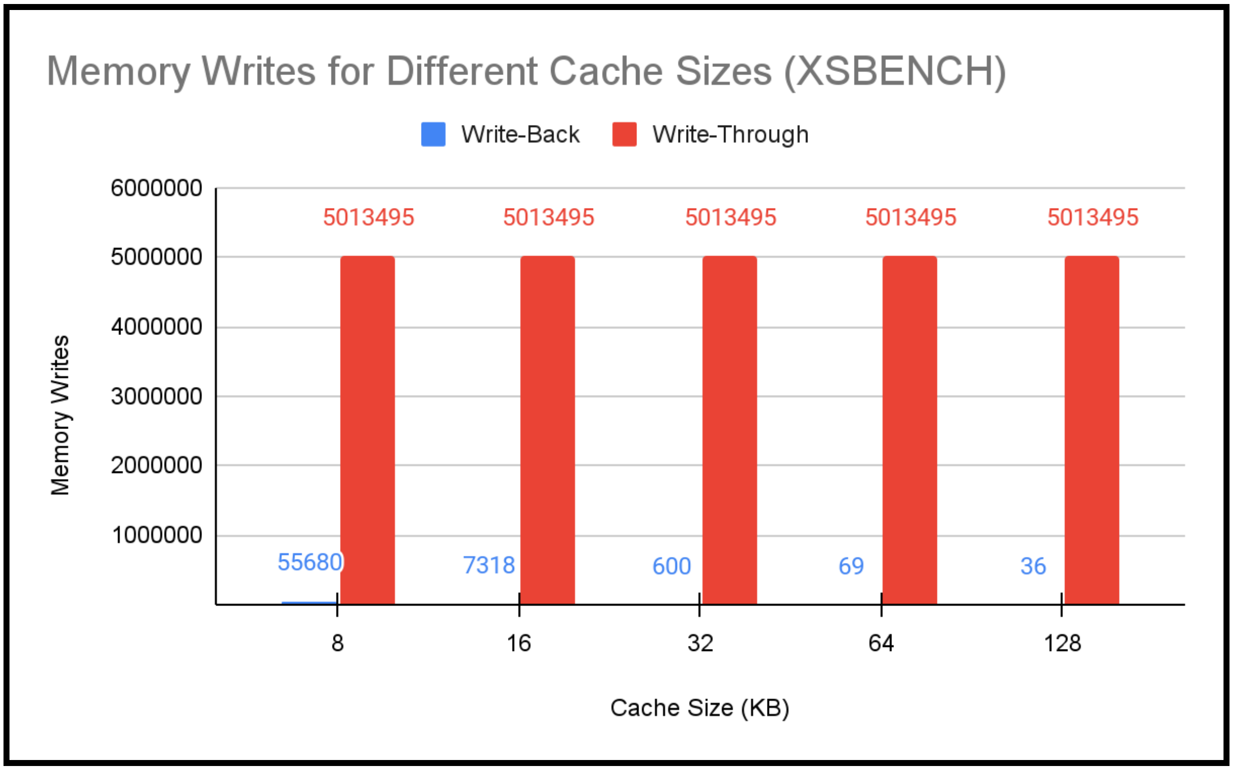


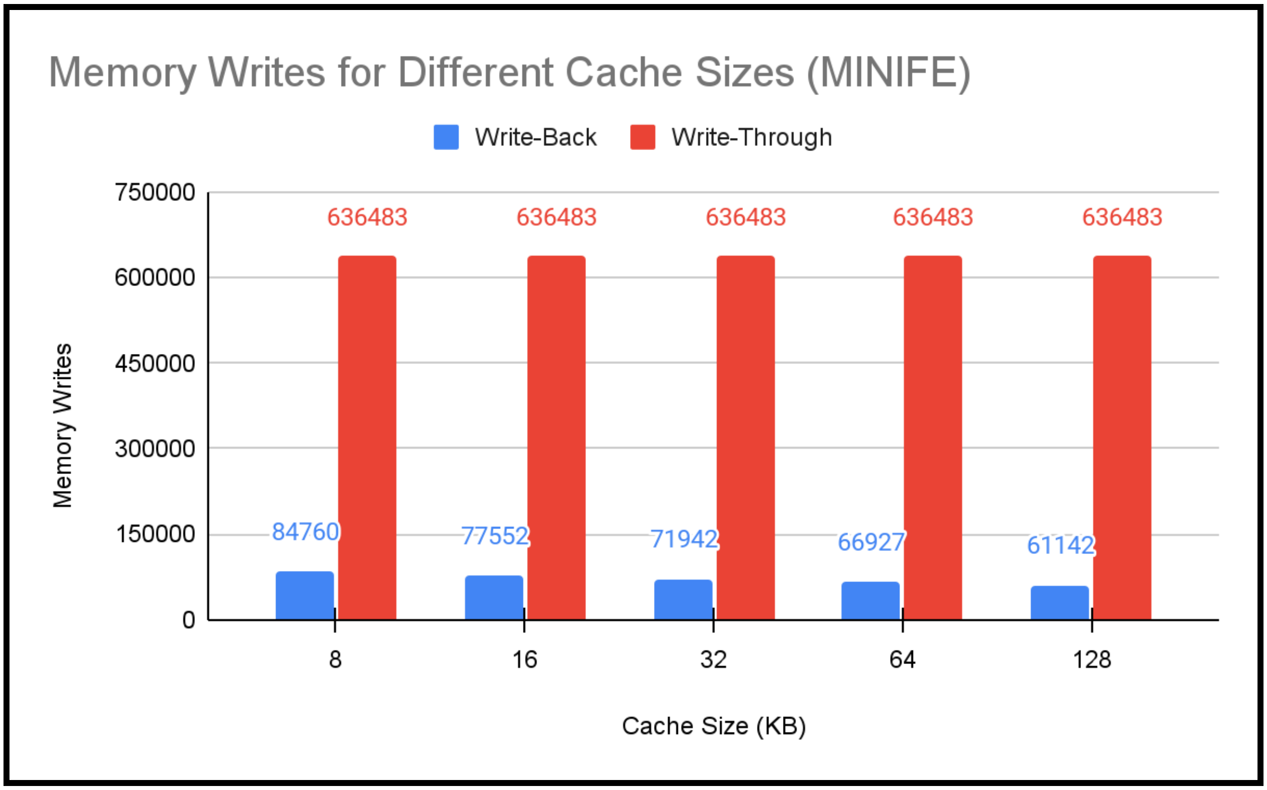


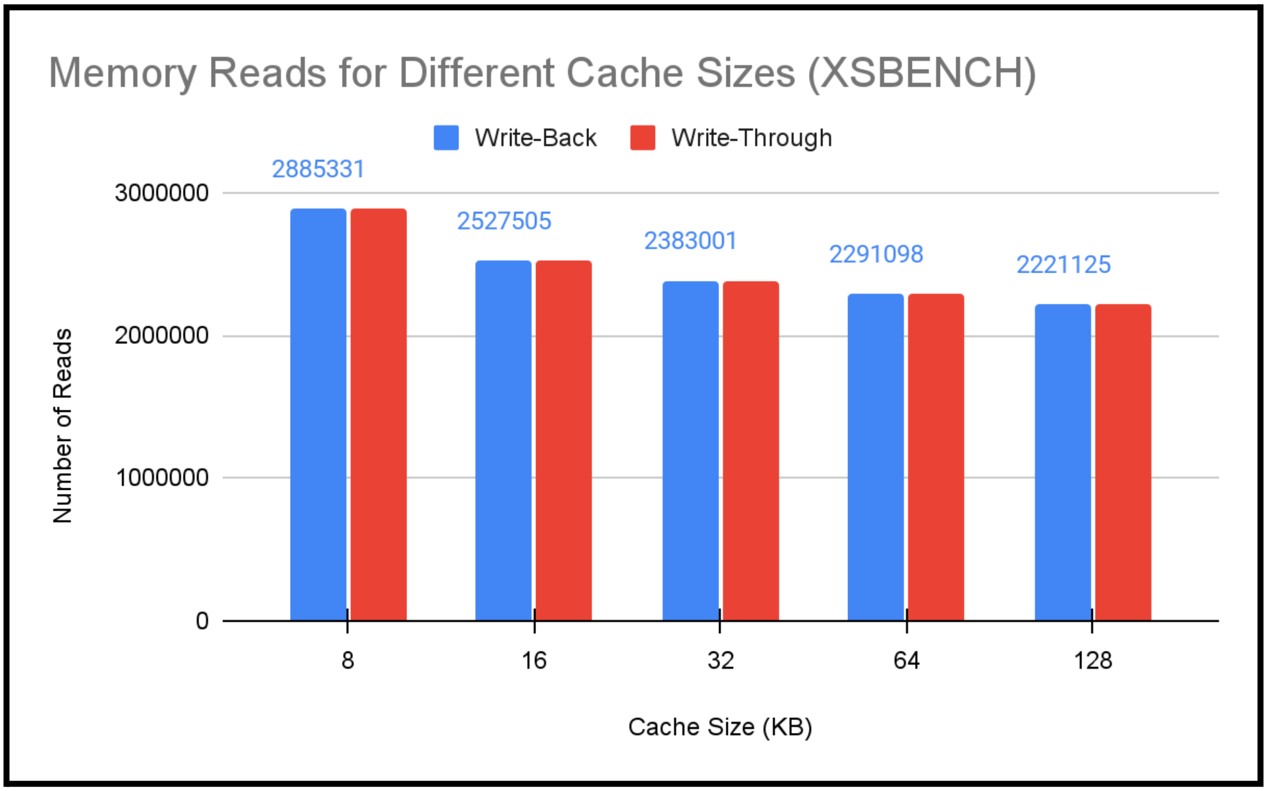
As seen in the above charts, when the cache size is increased, there is a direct decrease in the number of memory writes, reads, and the miss rate. This makes intuitive sense, as a larger cache means blocks will not be replaced as often, reducing the overall number of reads from memory. This trend is the same between the XBENCH and MINIFE test cases; however, a much sharper decline in memory writes is observed in MINIFE.

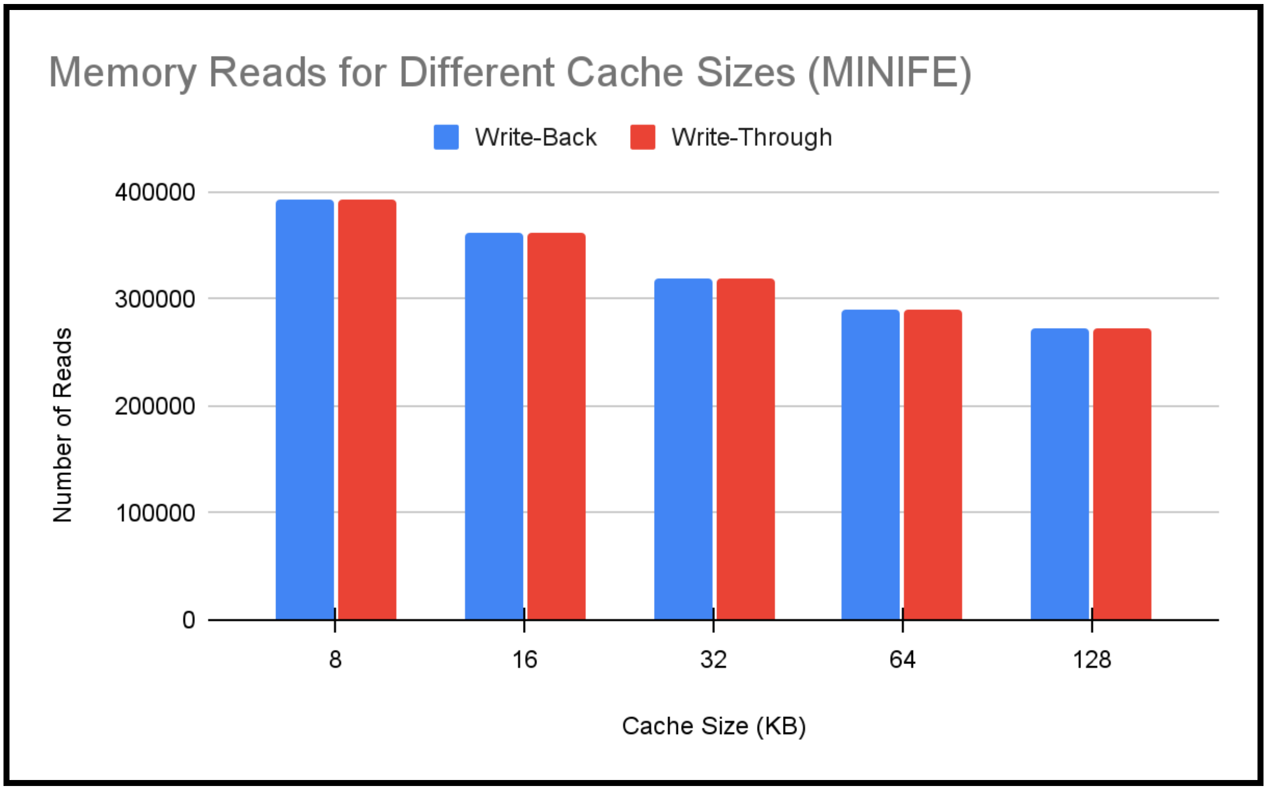
2.2 Write-Back Policy Comparison

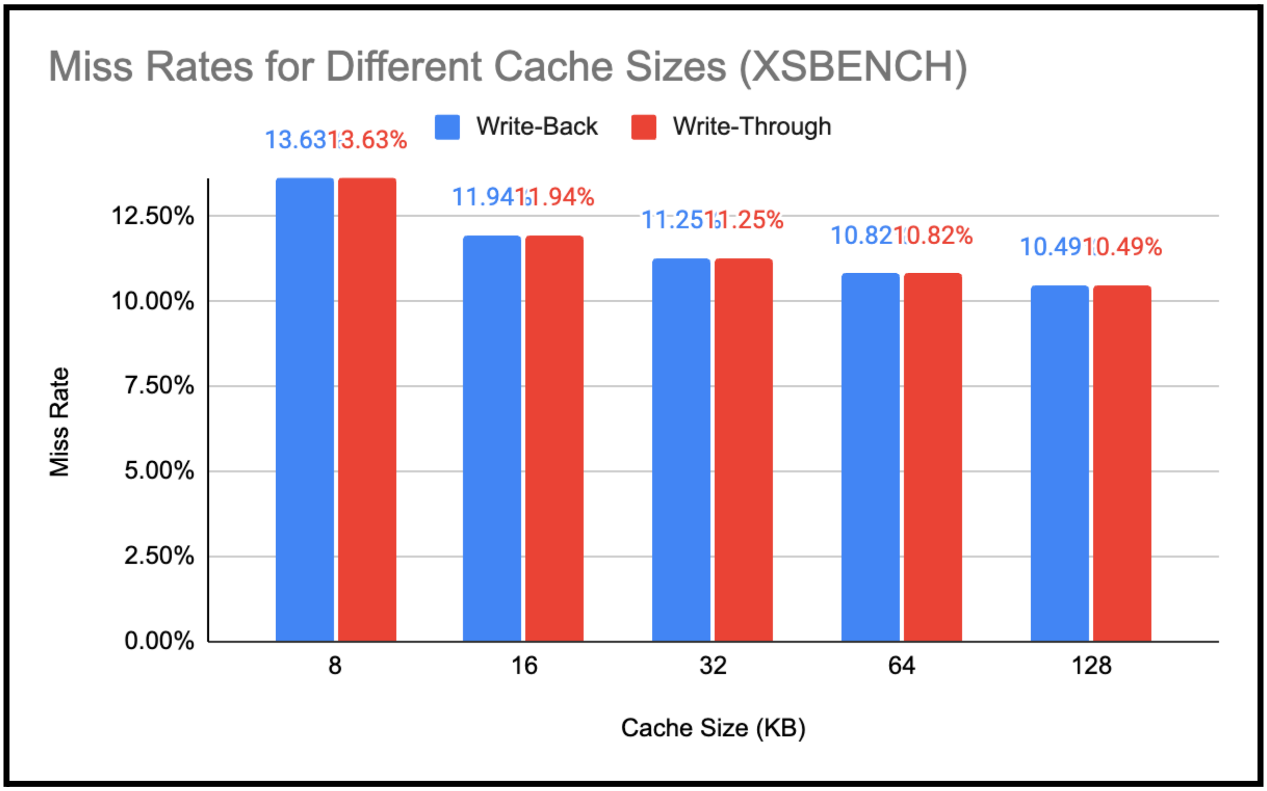
Default cache settings: 4-set associativity, LRU replacement

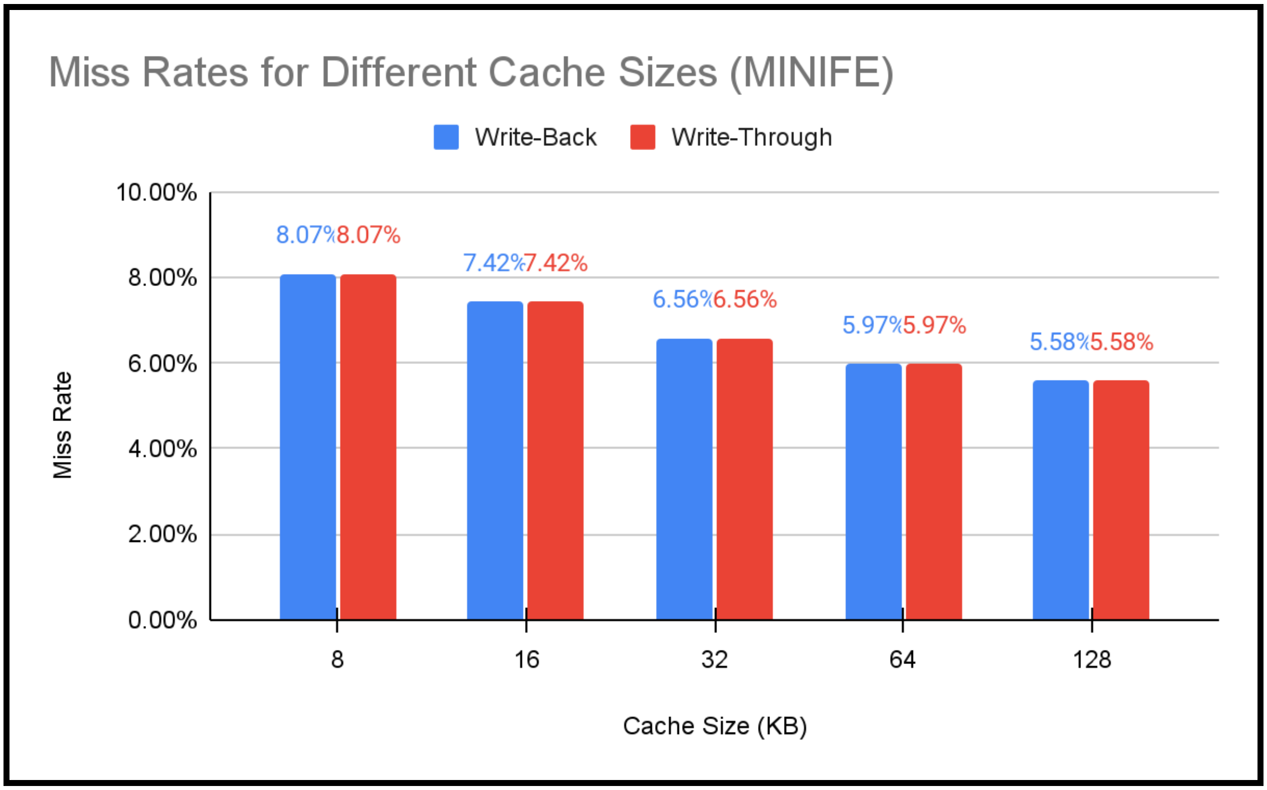



Memory Writes for Different Cache Sizes (MINIFE)
🟦 Write-Back
🟥 Write-Through
Memory Writes
750000
600000
450000
300000
150000
0
636483
636483
636483
636483
636483
84760
77552
71942
66927
61142
8
16
32
64
128
Cache Size (KB)


Memory Reads for Different Cache Sizes (XSBENCH)
🟦 Write-Back
🟥 Write-Through
Number of Reads
3000000
2000000
1000000
0
2885331
2527505
2383001
2291098
2221125
8
16
32
64
128
Cache Size (KB)


Memory Reads for Different Cache Sizes (MINIFE)
🟦 Write-Back
🟥 Write-Through
Number of Reads
400000
300000
200000
100000
0
8
16
32
64
128
Cache Size (KB)

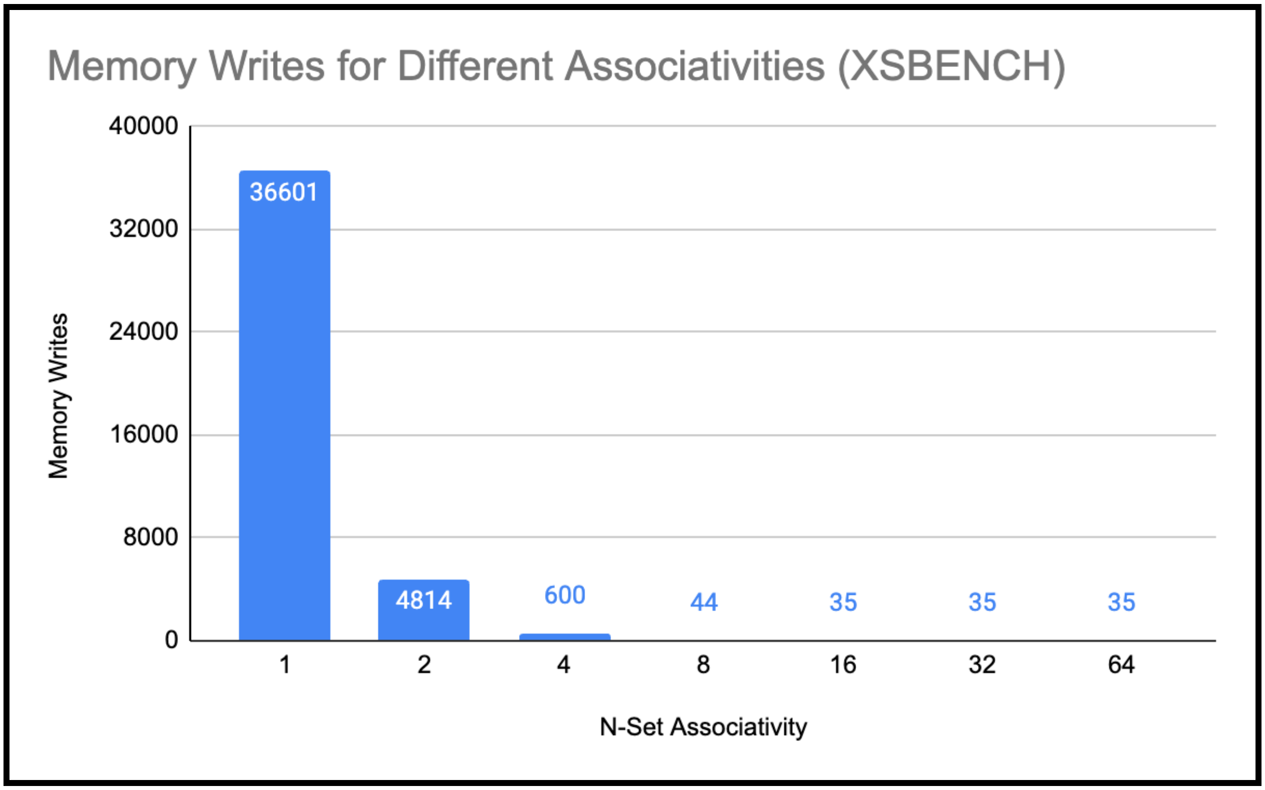

Miss Rates for Different Cache Sizes (XSBENCH)
🟦 Write-Back
🟥 Write-Through
13.63%
13.63%
11.94%
11.94%
11.25%
11.25%
10.82%
10.82%
10.49%
10.49%
Miss Rate
12.50%
10.00%
7.50%
5.00%
2.50%
0.00%
8
16
32
64
128
Cache Size (KB)

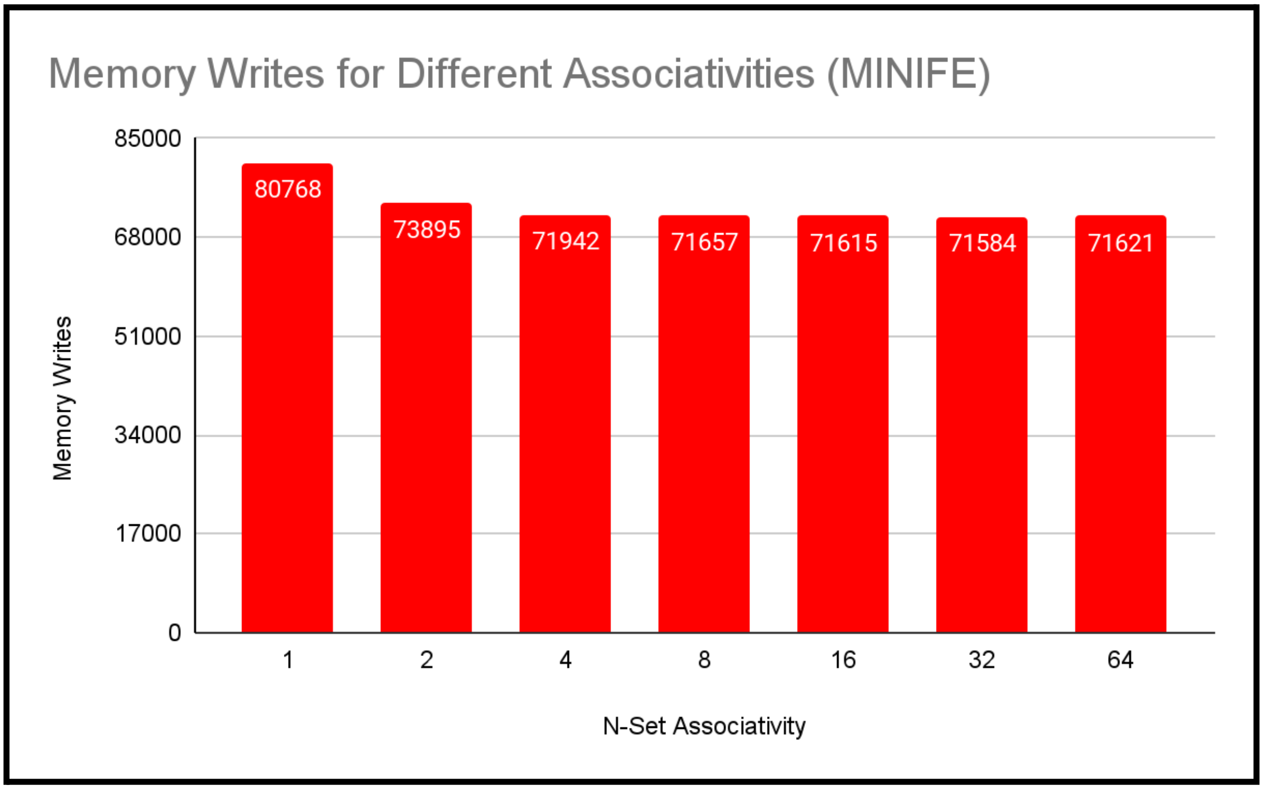


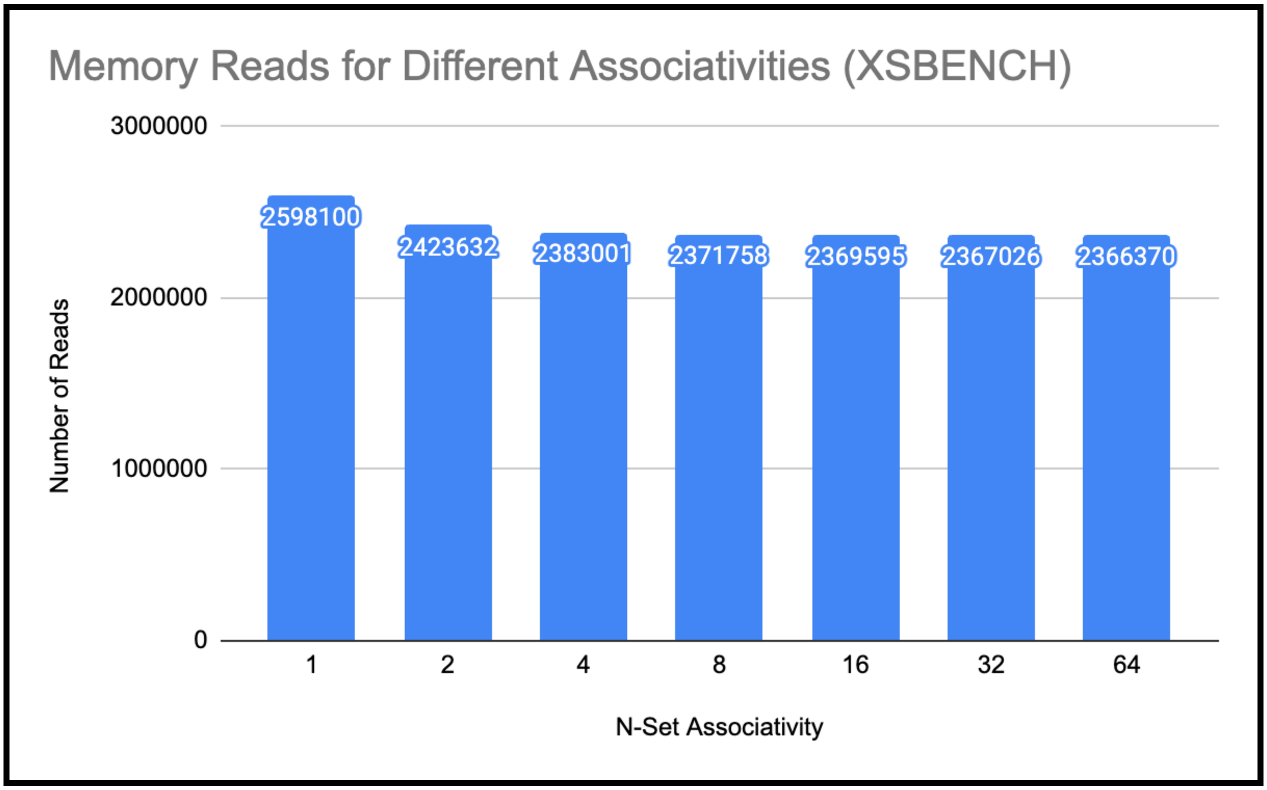
As seen in the above charts, changing the write-back policy from write-back to write-through has a dramatic effect on the overall number of memory writes; however, it leaves the number of memory reads and miss rate unchanged. This is a result of how each policy operates. On write-through, data is written to memory any time that it is updated, meaning that there will be a larger overall number of memory writes with this policy. On the other hand, write-through only writes the updated data to memory when a block in the cache containing the dirty bit is replaced. As a result, there are much less overall writes to memory. This same behavior is observed in both the XSBENCH and MINIFE test cases, but there is a much steeper change in memory writes observed in XSBENCH than in MINIFE.

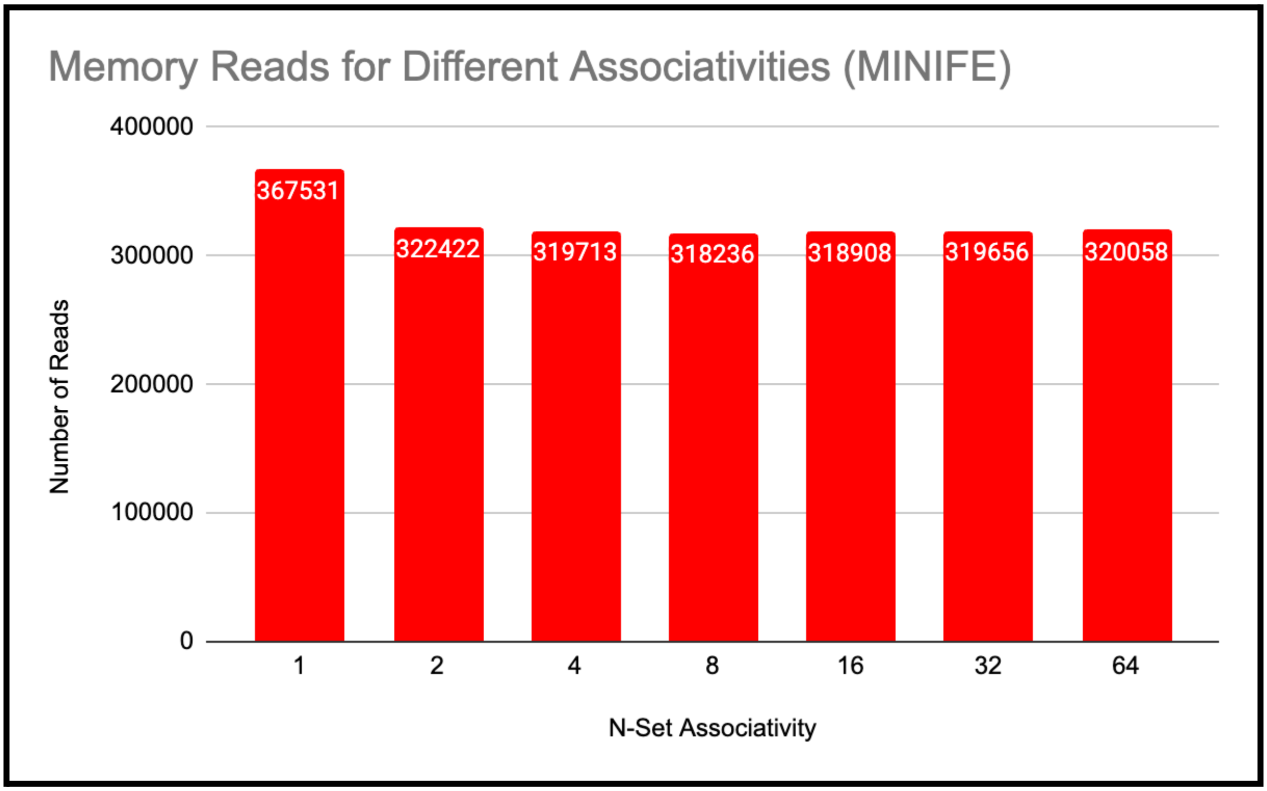
2.3 Associativity Comparison

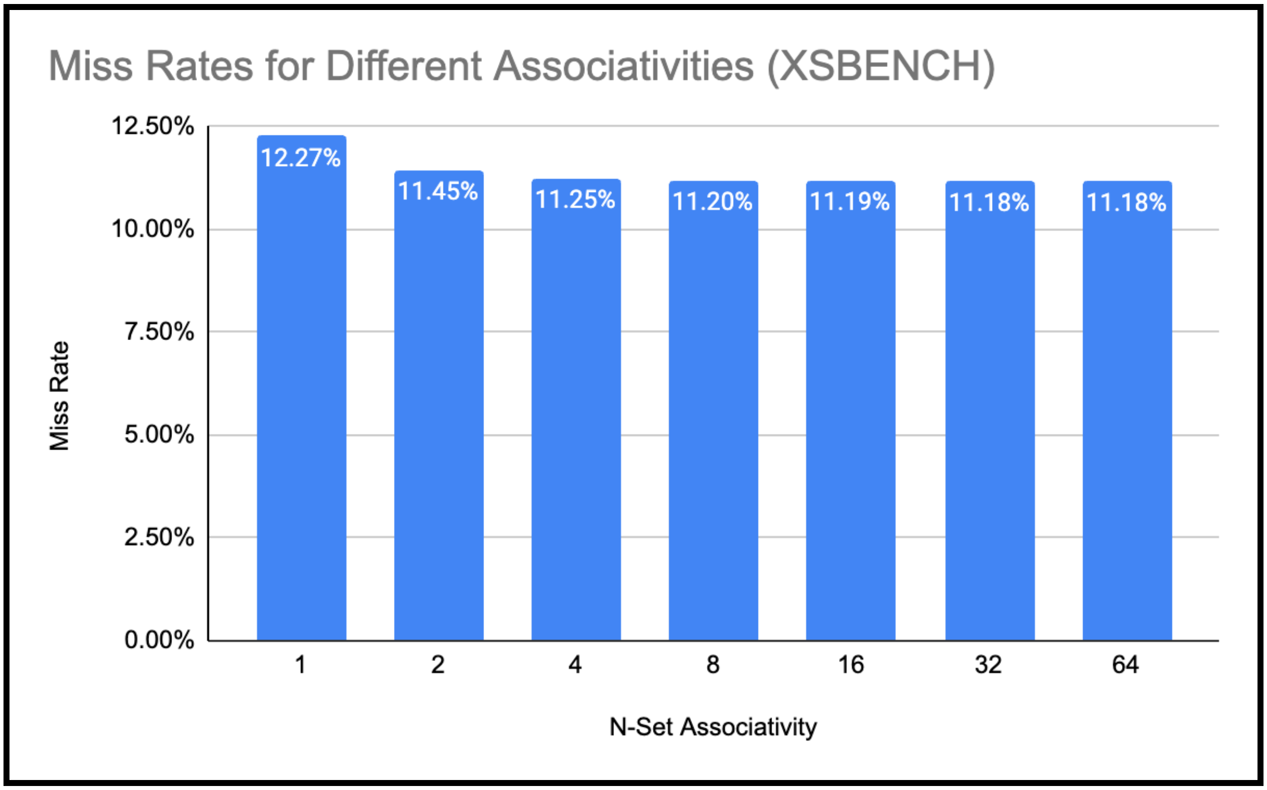
Default cache settings: 32KB in size, write-back, LRU replacement

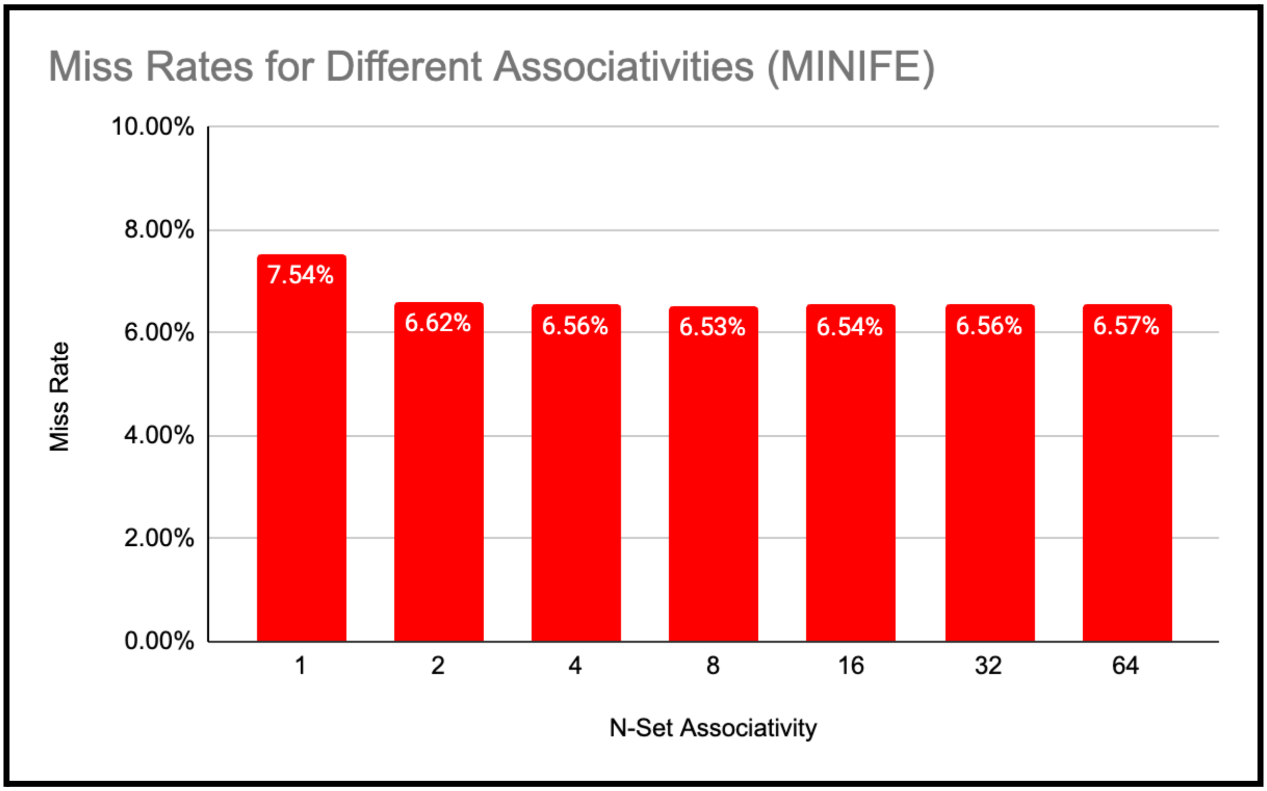

Memory Writes for Different Associativities (XSBENCH)
Memory Writes
40000
32000
24000
16000
8000
0
36601
4814
600
44
35
35
35
1
2
4
8
16
32
64
N-Set Associativity


Memory Writes for Different Associativities (MINIFE)
Memory Writes
85000
68000
51000
34000
17000
0
80768
73895
71942
71657
71615
71584
71621
1
2
4
8
16
32
64
N-Set Associativity


Memory Reads for Different Associativities (XSBENCH)
3000000
2000000
1000000
0
Number of Reads
2598100
2423632
2383001
2371758
2369595
2367026
2366370
1
2
4
8
16
32
64
N-Set Associativity


Memory Reads for Different Associativities (MINIFE)
400000
300000
200000
100000
0
Number of Reads
367531
322422
319713
318236
318908
319656
320058
1
2
4
8
16
32
64
N-Set Associativity



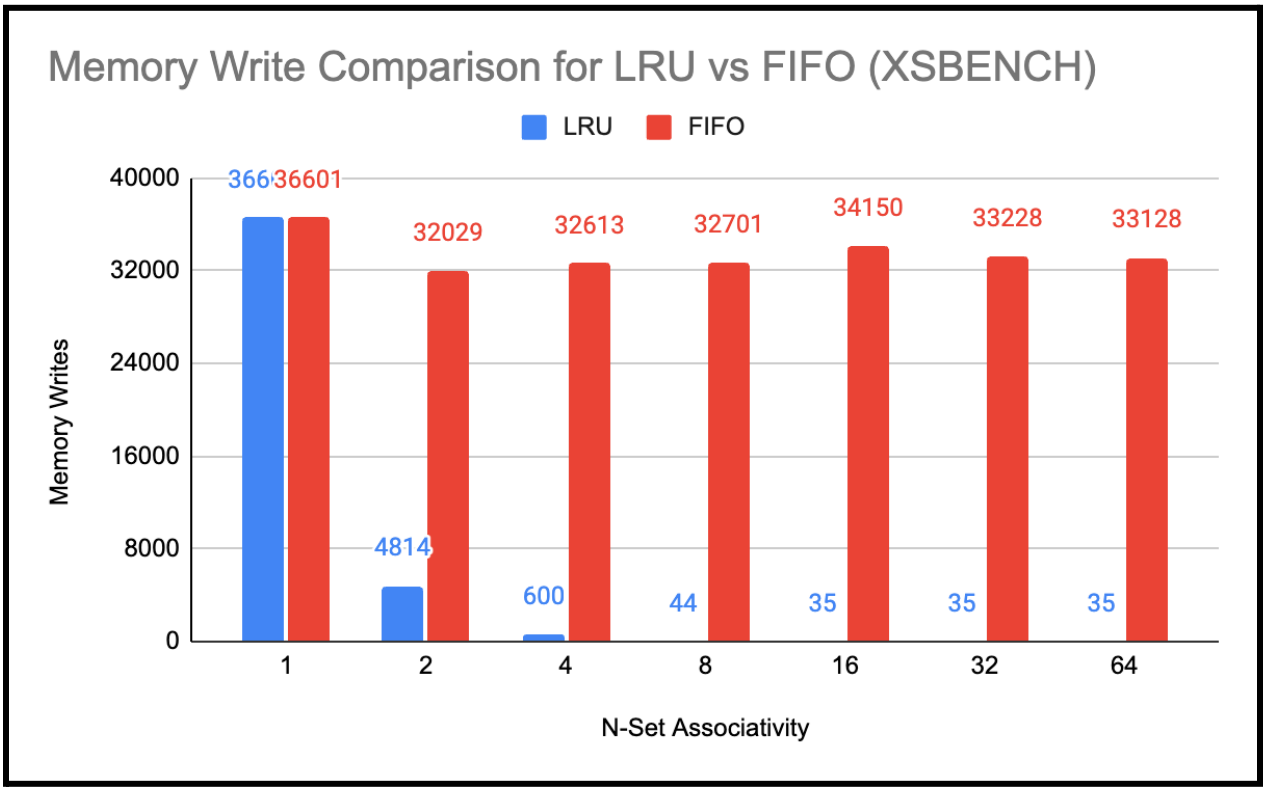


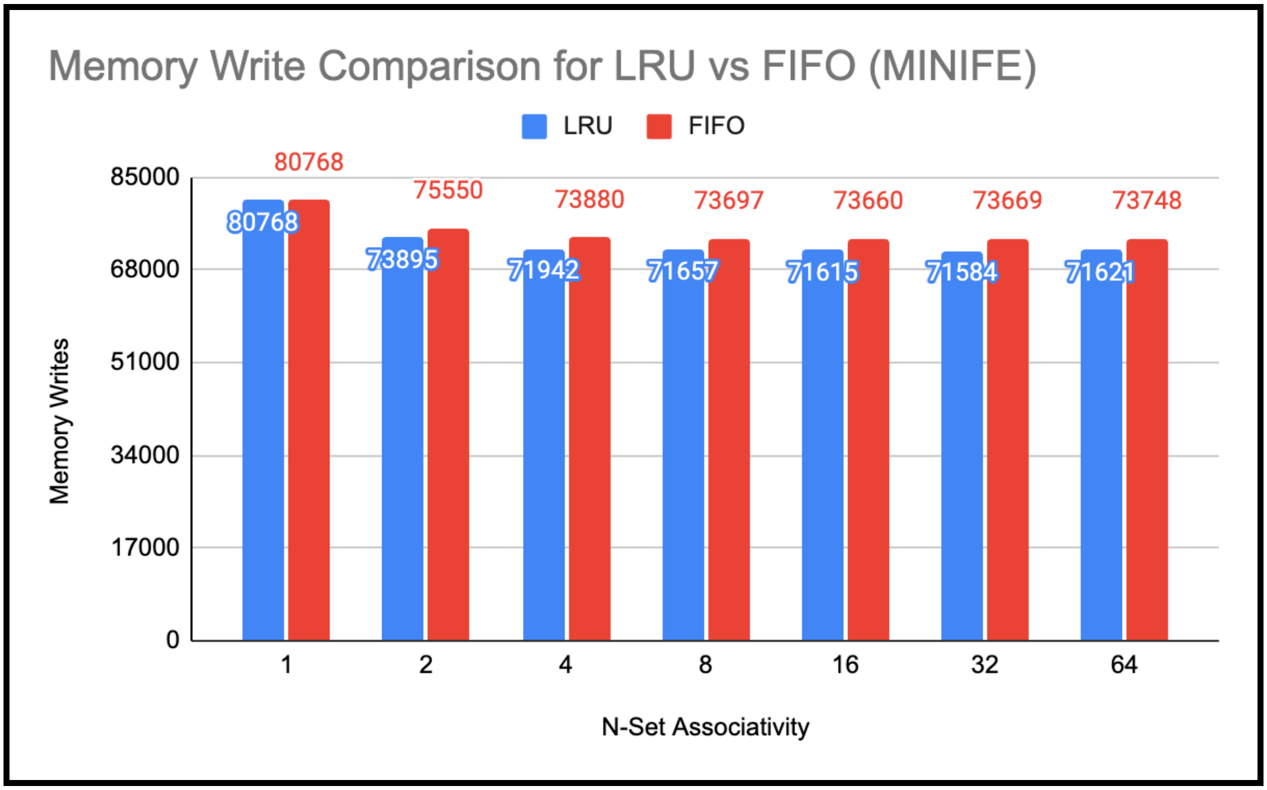
As seen in the charts above, increasing the associativity of a cache will marginally decrease the miss rate and number of memory writes and reads. However, this has diminishing returns after

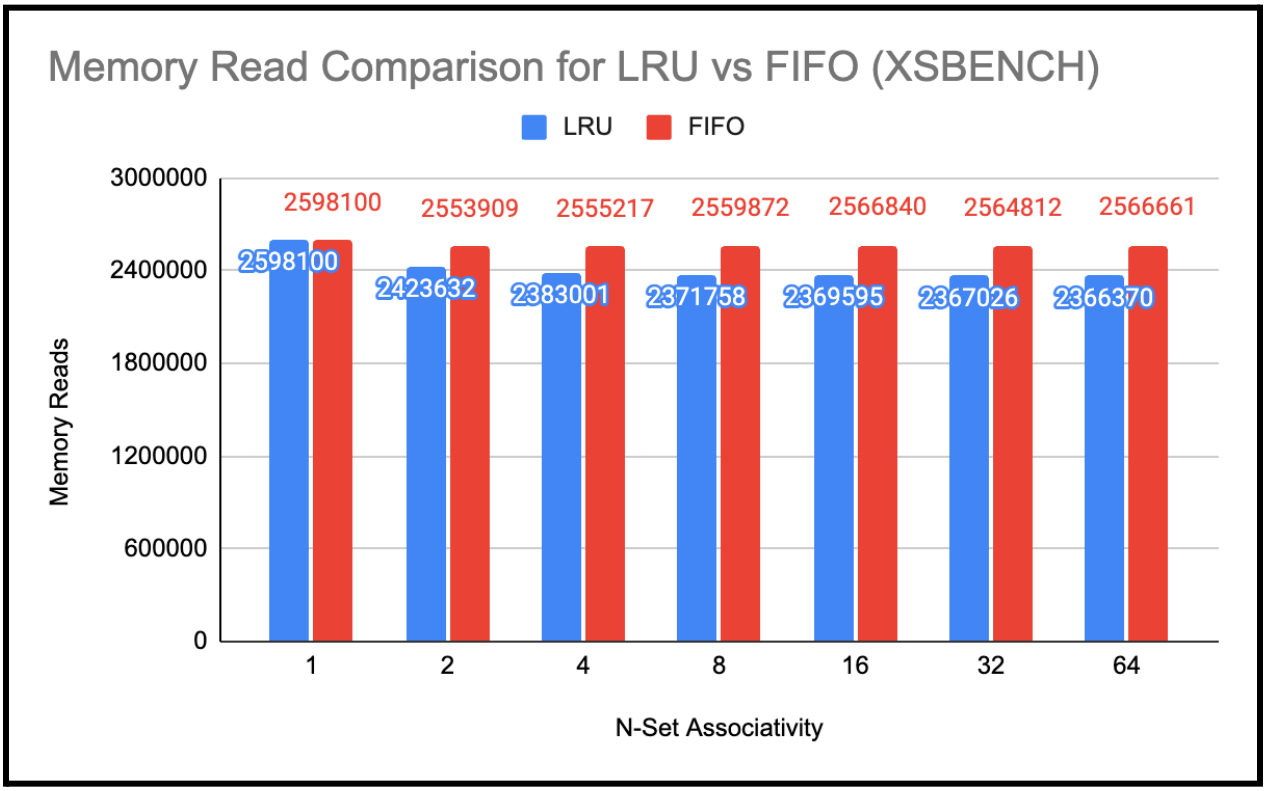
N=8, and it even begins to increase across some metrics. The only outlier to this trend is the chart for memory writes in XSBENCH, where a steep drop in memory writes is observed as associativity increases. Despite this, between both test cases (XSBENCH and MINIFE), one can conclude that increasing the associativity beyond 8 is not beneficial and may even be counterproductive.

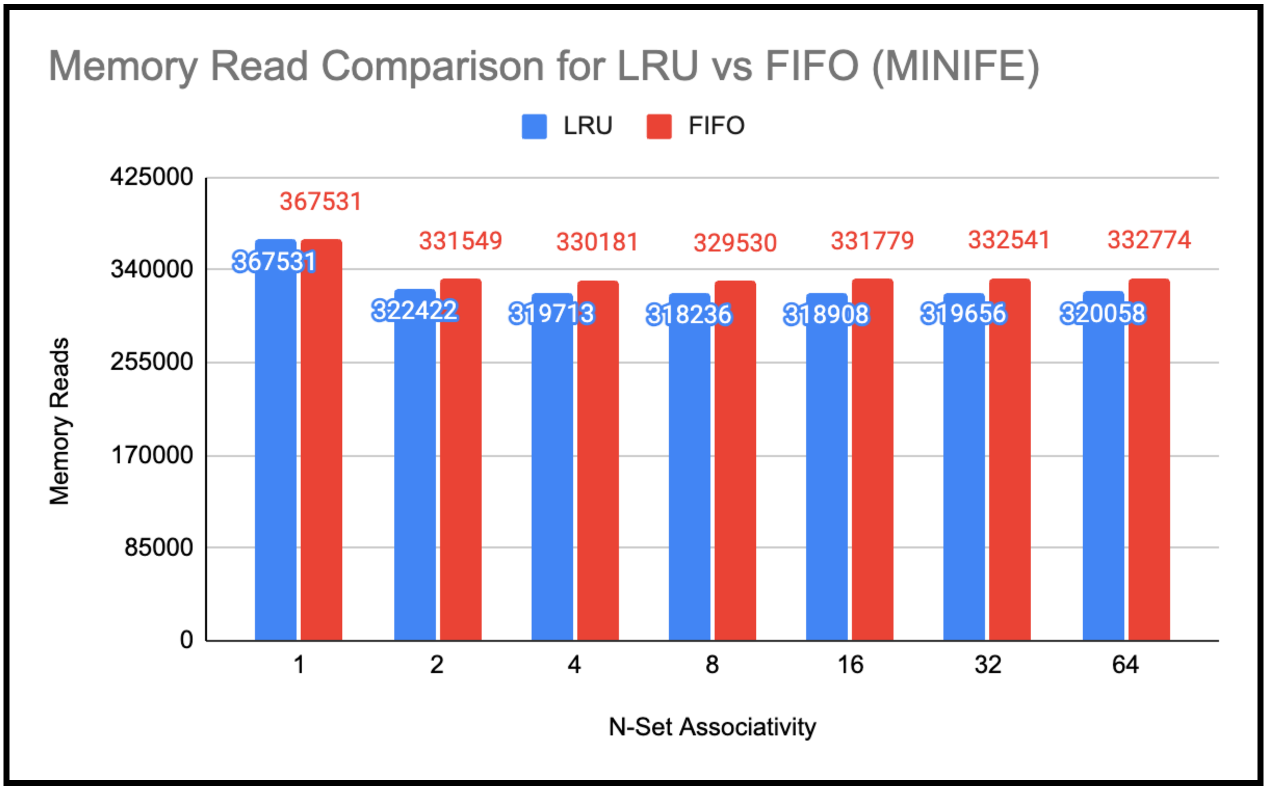
2.4 Replacement Policy Comparison

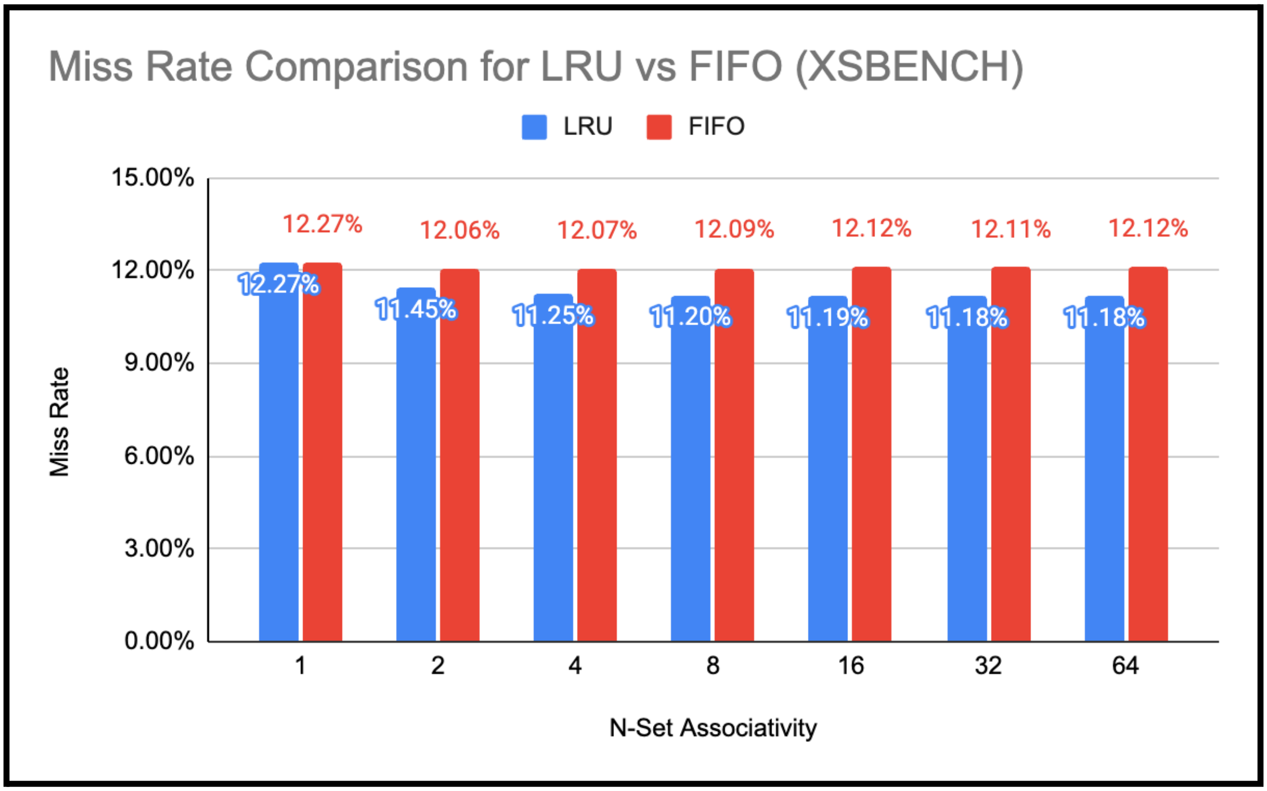
Default cache settings: 32KB in size, write-back

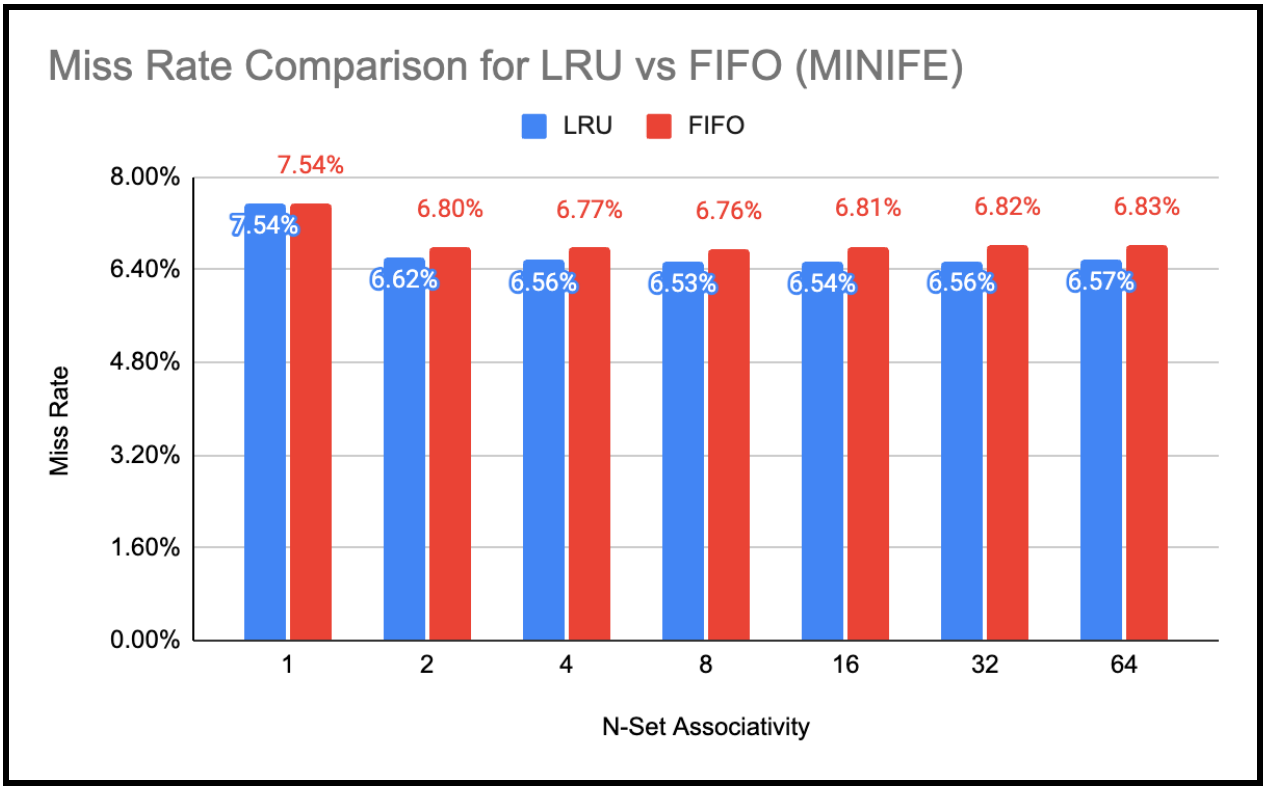



Memory Write Comparison for LRU vs FIFO (MINIFE)
🟦 LRU
🟥 FIFO
Memory Writes
85000
68000
51000
34000
17000
0
80768
80768
75550
73895
73880
71942
73697
71657
73660
71615
73669
71584
73748
71621
1
2
4
8
16
32
64
N-Set Associativity


Memory Read Comparison for LRU vs FIFO (XSBENCH)
🟦 LRU
🟥 FIFO
Memory Reads
3000000
2400000
1800000
1200000
600000
0
2598100
2598100
2553909
2423632
2555217
2383001
2559872
2371758
2566840
2369595
2564812
2367026
2566661
2366370
1
2
4
8
16
32
64
N-Set Associativity


Memory Read Comparison for LRU vs FIFO (MINIFE)
LRU
FIFO
Memory Reads
425000
340000
255000
170000
85000
0
367531
367531
322422
331549
319713
330181
318236
329530
318908
331779
319656
332541
320058
332774
1
2
4
8
16
32
64
N-Set Associativity


Miss Rate Comparison for LRU vs FIFO (XSBENCH)
LRU
FIFO
Miss Rate
15.00%
12.00%
9.00%
6.00%
3.00%
0.00%
12.27%
12.27%
11.45%
12.06%
11.25%
12.07%
11.20%
12.09%
11.19%
12.12%
11.18%
12.11%
11.18%
12.12%
1
2
4
8
16
32
64
N-Set Associativity



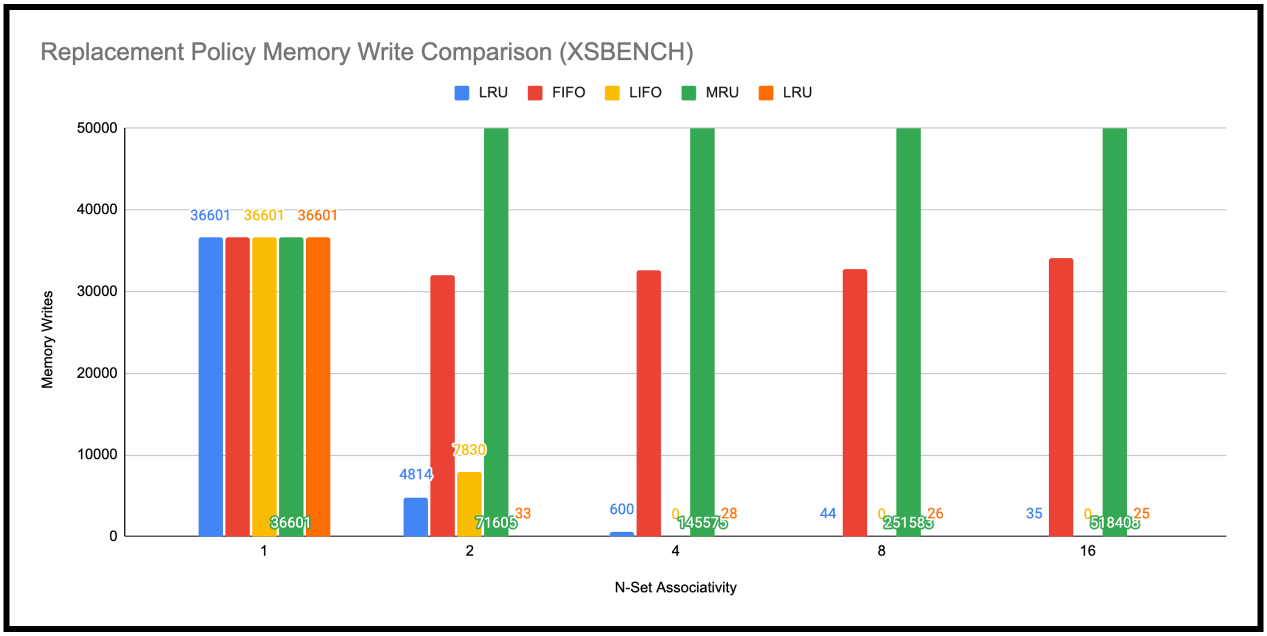
From the charts above, it can be concluded that the LRU and FIFO replacement policies have very comparable performance metrics. Using LRU leads to slightly fewer memory reads and a slightly lower miss rate than FIFO. It also leads to fewer memory writes than FIFO in the MINIFE test case, and significantly fewer memory writes than FIFO in the XSBENCH test case. Thus, it can be concluded that it is more advantageous to use least-recently used replacement when designing a cache than using first-in, first-out.

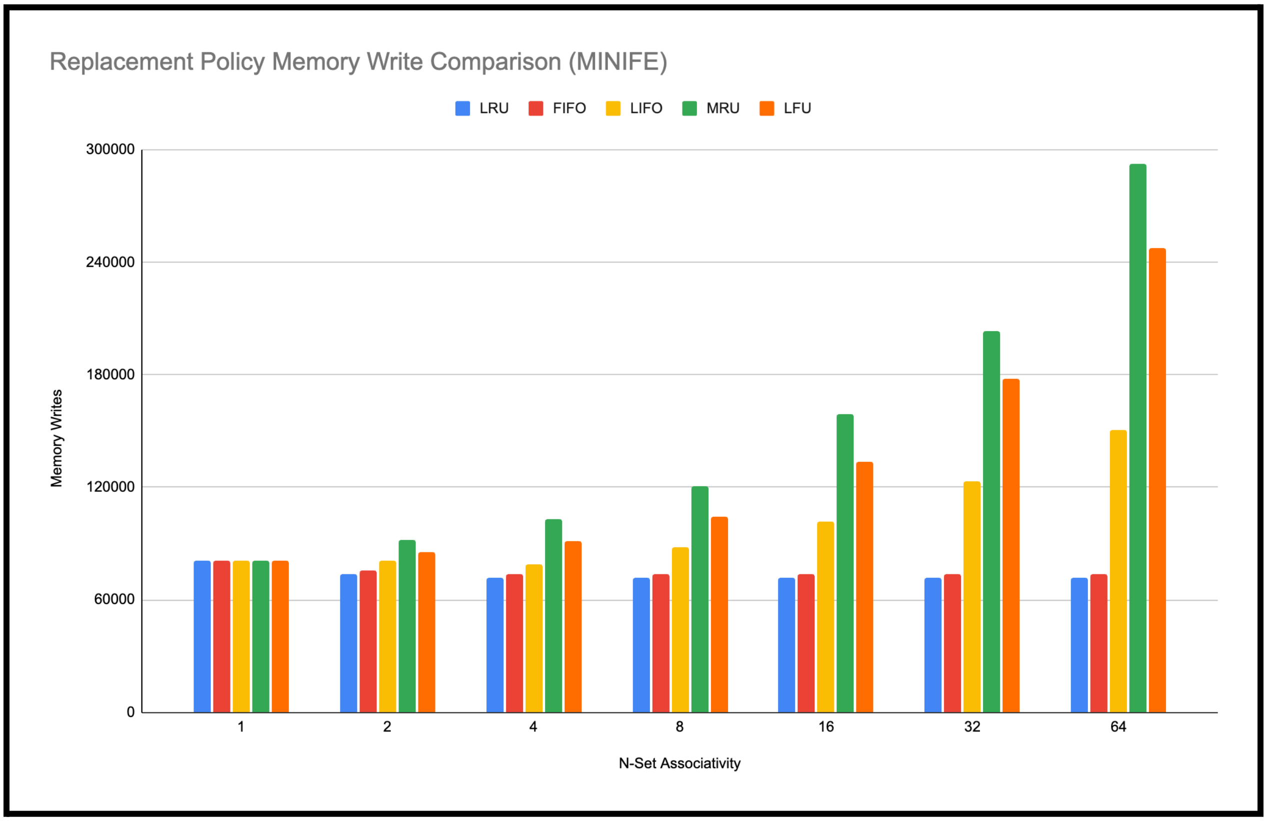
3.0 Bonus Replacement Policy Implementation

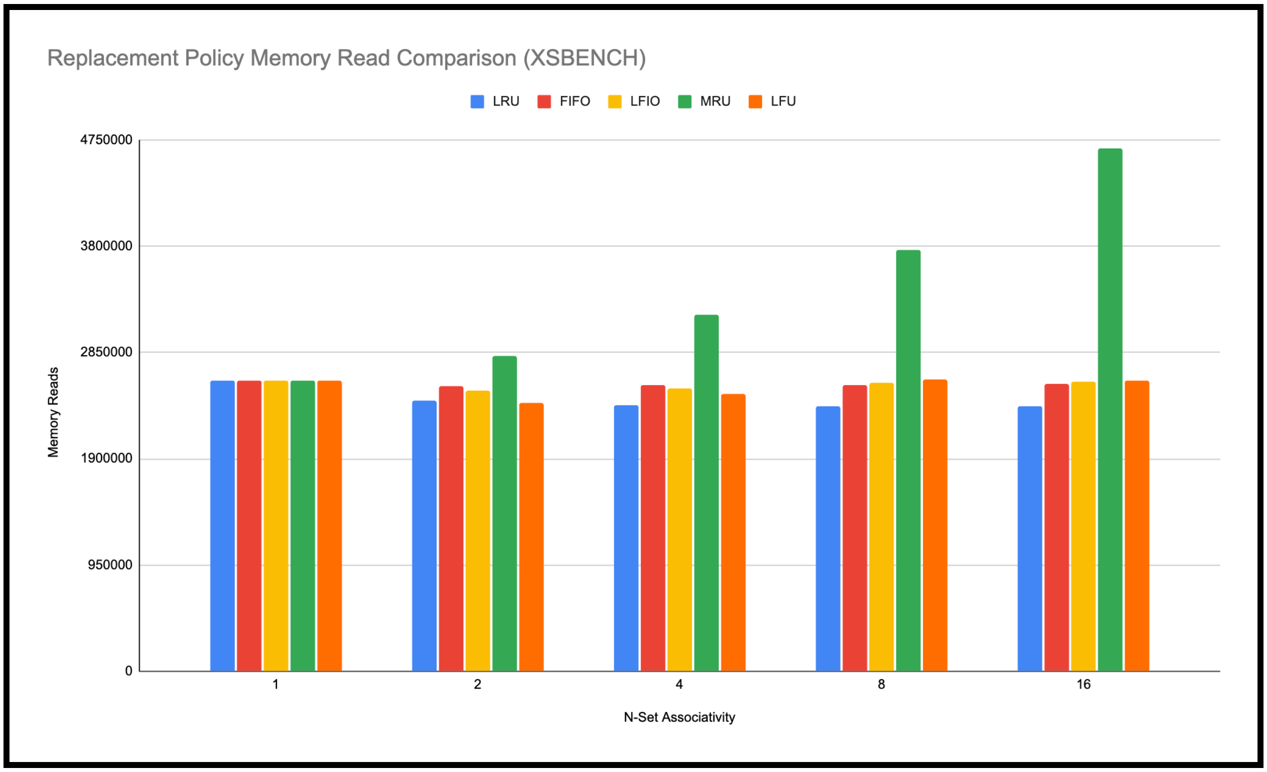
As a bonus section of this project, the following replacement policies were also implemented and tested in C++ code:

- Last-in, first-out (LIFO)
- Most-recently used (MRU)
- Least-frequently used (LFU)

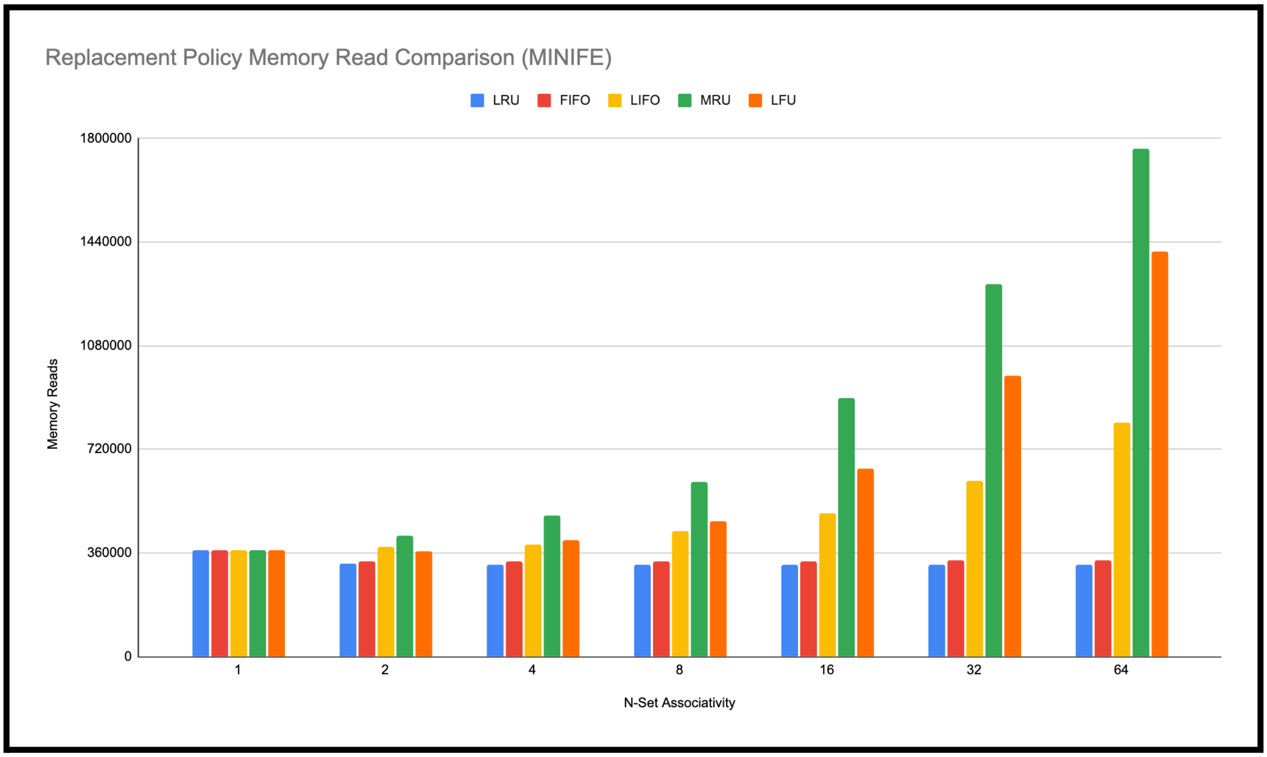
The charts below compare each of the replacement policies using a 32KB cache and write-back policies for associativities ranging from 1 to 64 in powers of 2. For XSBENCH, the chart range from 1 to 16 in powers of 2.


Replacement Policy Memory Write Comparison (XSBENCH)
LRU
FIFO
LIFO
MRU
LRU
Memory Writes
50000
40000
30000
20000
10000
0
36601
36601
36601
36601
4814
7830
71605
33
600
0
145575
28
44
0
251583
26
35
0
518408
25
1
2
4
8
16
N-Set Associativity

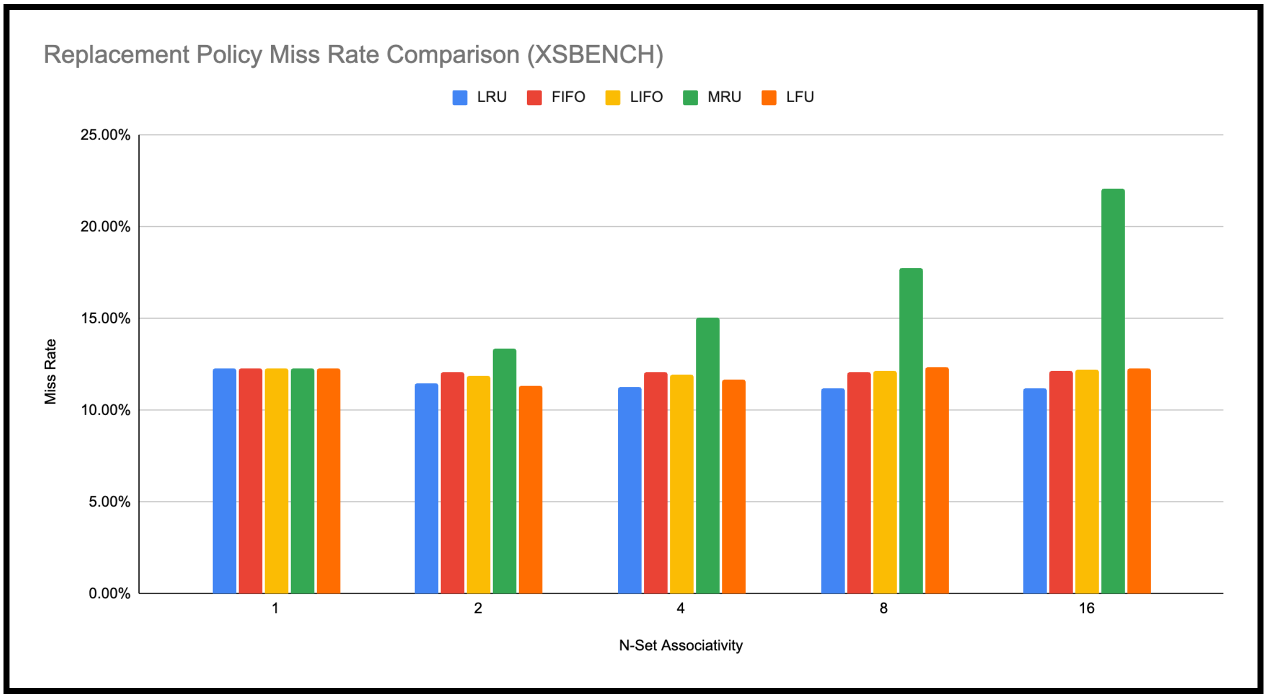

Replacement Policy Memory Write Comparison (MINIFE)
LRU
FIFO
LIFO
MRU
LFU
Memory Writes
300000
240000
180000
120000
60000
0
1
2
4
8
16
32
64
N-Set Associativity

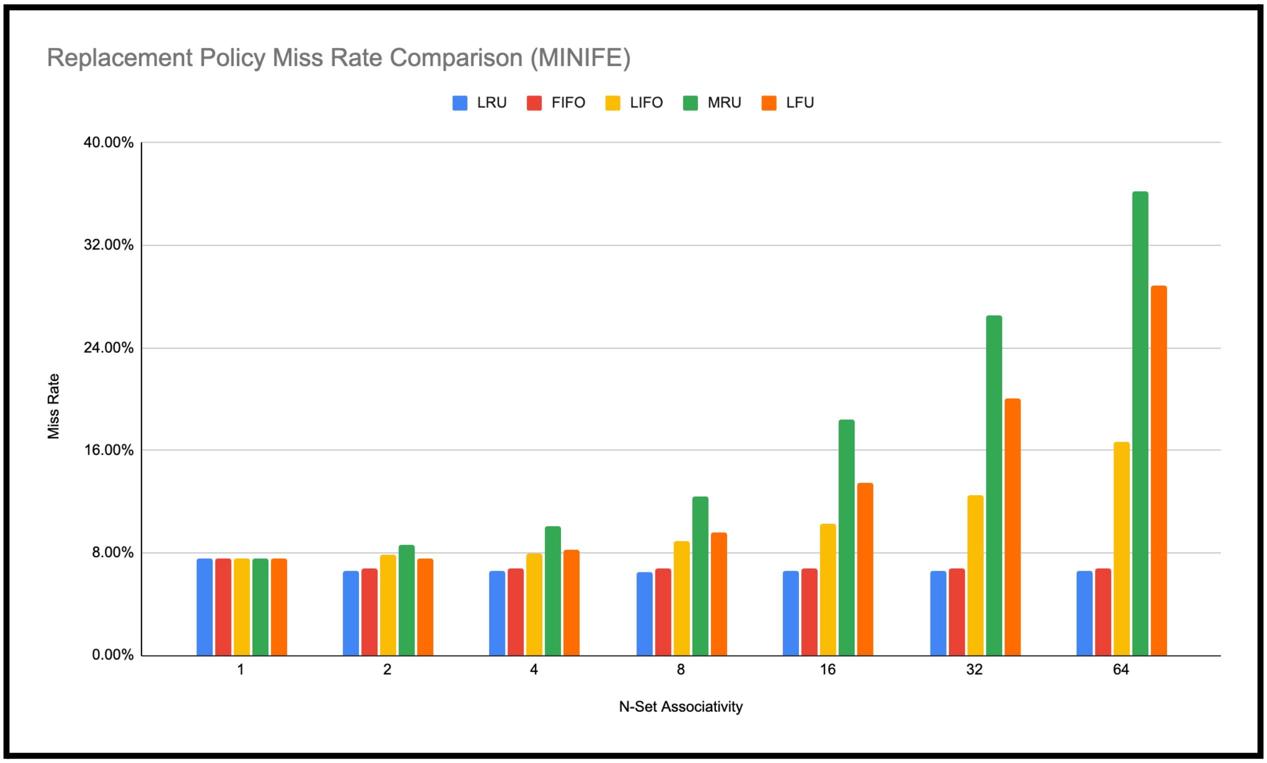


Replacement Policy Memory Read Comparison (XSBENCH)



Replacement Policy Memory Read Comparison (MINIFE)


Replacement Policy Miss Rate Comparison (XSBENCH)


Replacement Policy Miss Rate Comparison (MINIFE)

As seen from the charts above, the worst-performing policy across all metrics is most-recently used (MRU). This policy is counterintuitive to the idea of temporal locality, which states that data items accessed recently will be accessed again in the near future. Least-recently used was the best-performing policy, followed closely by first-in, first-out. Last-in, first-out performed surprisingly well, between FIFO and least-frequently used (LFU). Overall, each policy brings different trade-offs, but LRU is still the best overall option for this cache implementation.

4.0 Conclusion

After comparing performance metrics for cache size, associativity, write-back policy, and replacement policy, with both large and small test cases, some conclusions can be drawn about designing the optimal memory cache. It is beneficial to have a larger cache, use a write-back policy over write-through, select an associativity no greater than 8, and use an LRU replacement policy over FIFO. These conclusions were drawn from solely looking at the miss rate and the number of memory writes and reads, and they do not factor in cost or latency. In reality, having a larger cache would be more expensive and slower, so further testing for these metrics would be necessary to devise the ideal cache scheme.