CS3410 Final Project

Mary Yuan, Cooper Proctor, Max Wang

May 2024

Graph 1: Miss Rate vs Cache Size

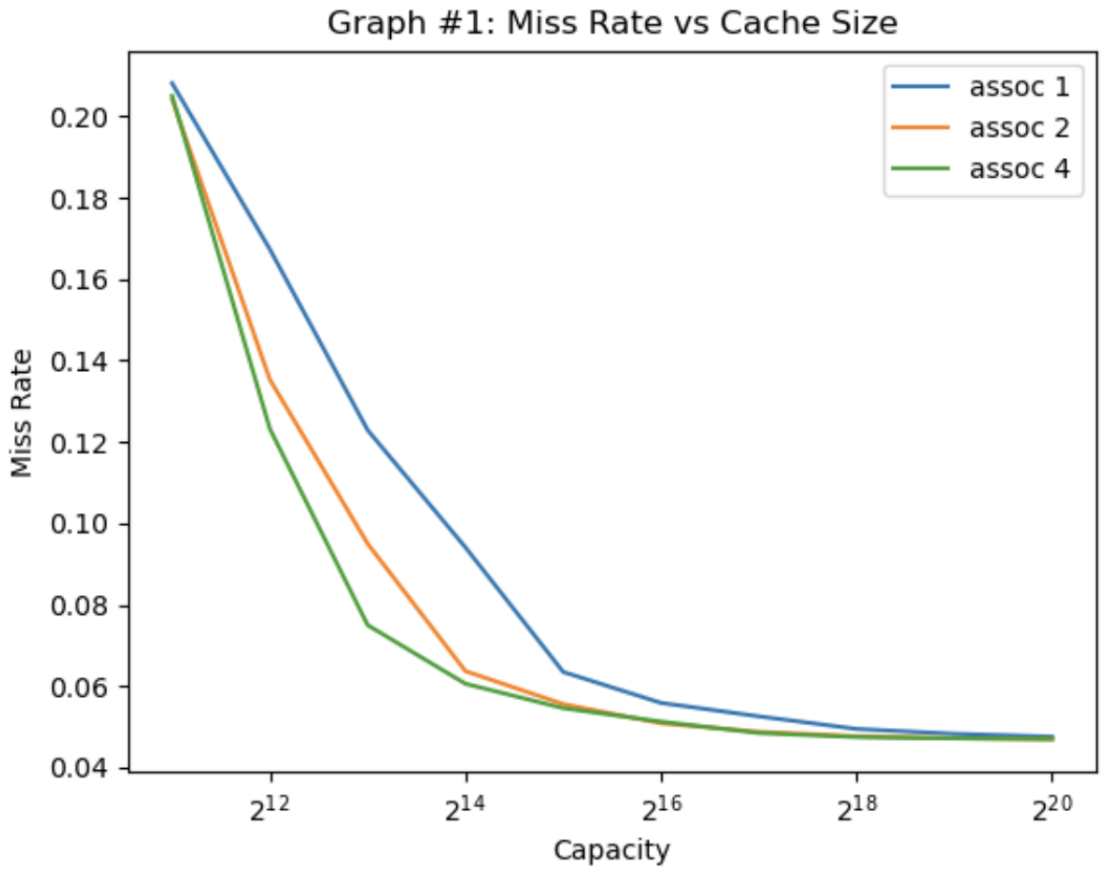

Figure 1: Graph 1

1. What is the smallest capacity that brings the miss rate to less than 10%?

The smallest capacity that brings the miss rate to less than 10 % is 2^{14} bytes, or 16 kilobytes.

2. What is the total memory requirement (i.e. maximum memory needed) for this address space? For this RAM size and your answer to question 1, what is the ratio of cache to memory size?

Our total memory requirement is 2^{32} bits (4 GB) since addresses are 32 bits (2^{32} possible addresses, each storing a byte). Our cache can store 2^{14} bytes, for a 2^{14} to 2^{32} (1: 2^{18}) ratio of cache to memory.

3. Today's processors generally have 32KB to 128KB first-level (L1) data caches. By what ratio does increasing the cache size from 16KB to 32KB reduce the miss rate? (2.0 would correspond to halving the miss rate; 1.0 would correspond to no change in miss rate; less than 1.0 would correspond to an increase in misses).

From simulation, increasing the cache capacity from 16 KB to 32 KB capacity reduces the miss rate from 9.41% to 6.35% for a ratio of 1.48.

4. By what ratio does increasing the cache size from 32KB to 64KB reduce the miss rate?

Increasing the cache size reduced the miss rate by a ratio of $6.35\%/5.59\% = 1.136$.

5. When deciding on the ideal cache size, engineers typically look for the "knee" of the curve. When considering various cache sizes, we want the point at which increasing to that size yields a great benefit, but increasing beyond that size yields far less benefit. What would you say is the ideal cache size for a direct-mapped cache?

The ideal cache size for a direct-mapped cache appears to be 32 KB; caches at a larger capacity see much less benefit since miss rate decreases at a much lower rate.

6. What is the smallest capacity that brings the miss rate of the 2-way set associative cache to less than 10%?

The smallest capacity that brings the miss rate of the 2-way set associative cache to less than 10% appears to be 2^{13} bytes, or 8 kilobytes.

7. What is the smallest capacity that brings the miss rate of the 4-way set associative cache to less than 10%?

The smallest capacity that brings the miss rate of the 2-way set associative cache to less than 10% appears to be 2^{13} bytes, or 8 kilobytes.

8. How large must the direct-mapped cache be before it equals or exceeds the performance of the 8 KB 4-way associative?

In terms of hit/miss rate, the 8 KB 4-way associative obtains 11.10% miss rate, which the direct-mapped cache outperforms at capacity $2^{15}B = 32KB$.

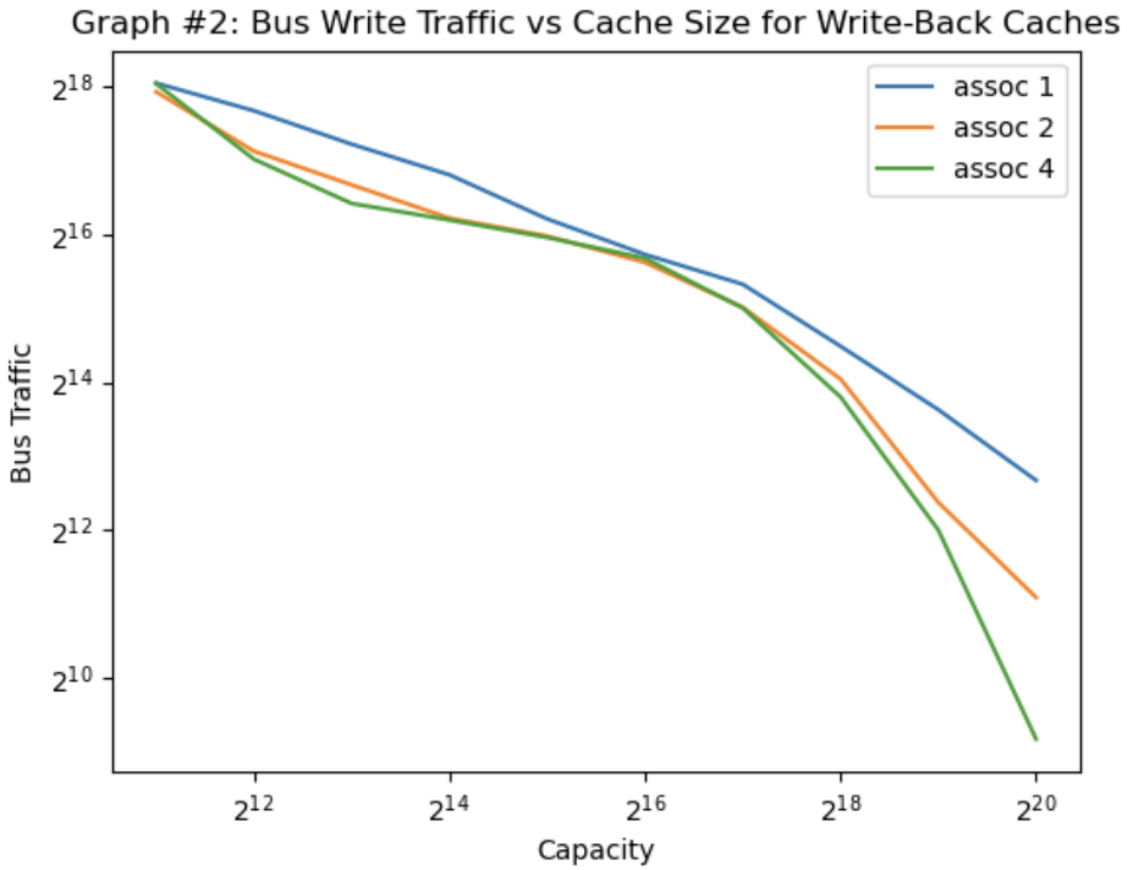

Figure 2: Graph 2

Graph 2: Bus Write Traffic vs Cache Size for Write-Back Caches

9. What happens to traffic as the capacity increases? Is there a pattern? If so, explain what you see.

Memory traffic decreases as capacity increases. This is because as capacity increases, miss rate decreases, meaning that memory needs to be accessed less. We see this since larger capacity reduces conflict misses and has better spatial and temporal locality.

10. What role does associativity play in traffic generation? Is there a pattern? If so, explain what you see.

Memory traffic decreases as associativity increases, because as associativity increases, miss rate decreases, meaning that memory needs to be accessed less for the same reason as above.

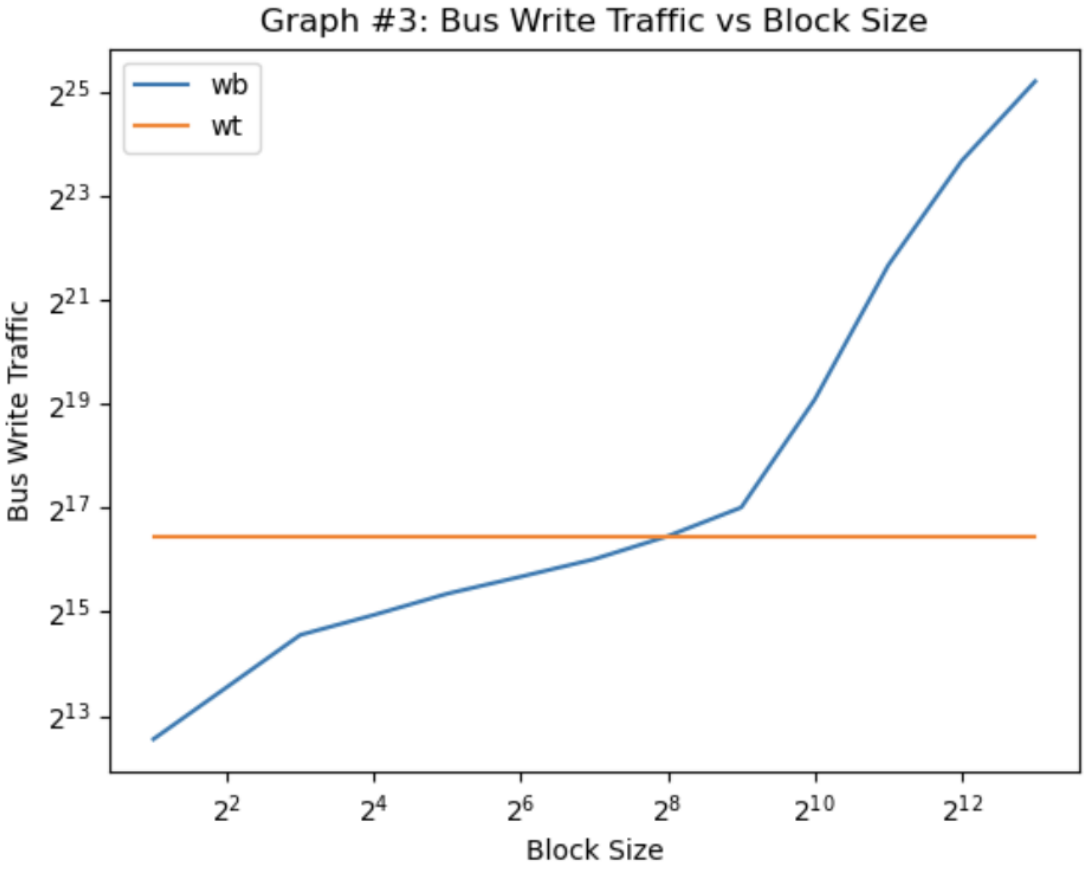

Figure 3: Graph 3

Graph 3: Traffic in Bytes vs Block Size

11. At what block size do the two write policies generate approximately the same number of writes to the bus?

Written_cache_to_bus_wb and written_cache_to_bus_wt generate approximately the same number of writes for a block size of 256B (2^8 bytes) from simulation with 4-way 64 KB capacity cache.

12. Explain the difference in traffic observed for smaller block sizes.

With a smaller block (¡256 B), the chances of a block having a write is lower since it contains less data; thus it is less likely that we need to write to memory. Furthermore, a smaller block size means that recently written data may be accessed or modified again before it is evicted from the cache. As a result, the data may not need to be written back to memory immediately after each modification, leading to fewer write-back operations. Therefore, write-back traffic is less than write-through traffic.

13. Explain the difference in traffic observed for larger block sizes.

With larger block sizes (¿256 B), data is more likely to be written to the block; so when a block is evicted a large block is written to memory. This results in an exponential increase in writes to bus as block size increases since just one write occurring means that a large block must be written to memory. Thus, write-back traffic is higher than write-through traffic for larger block sizes.

Graph 4A: Miss Rate vs Block Size (VI)

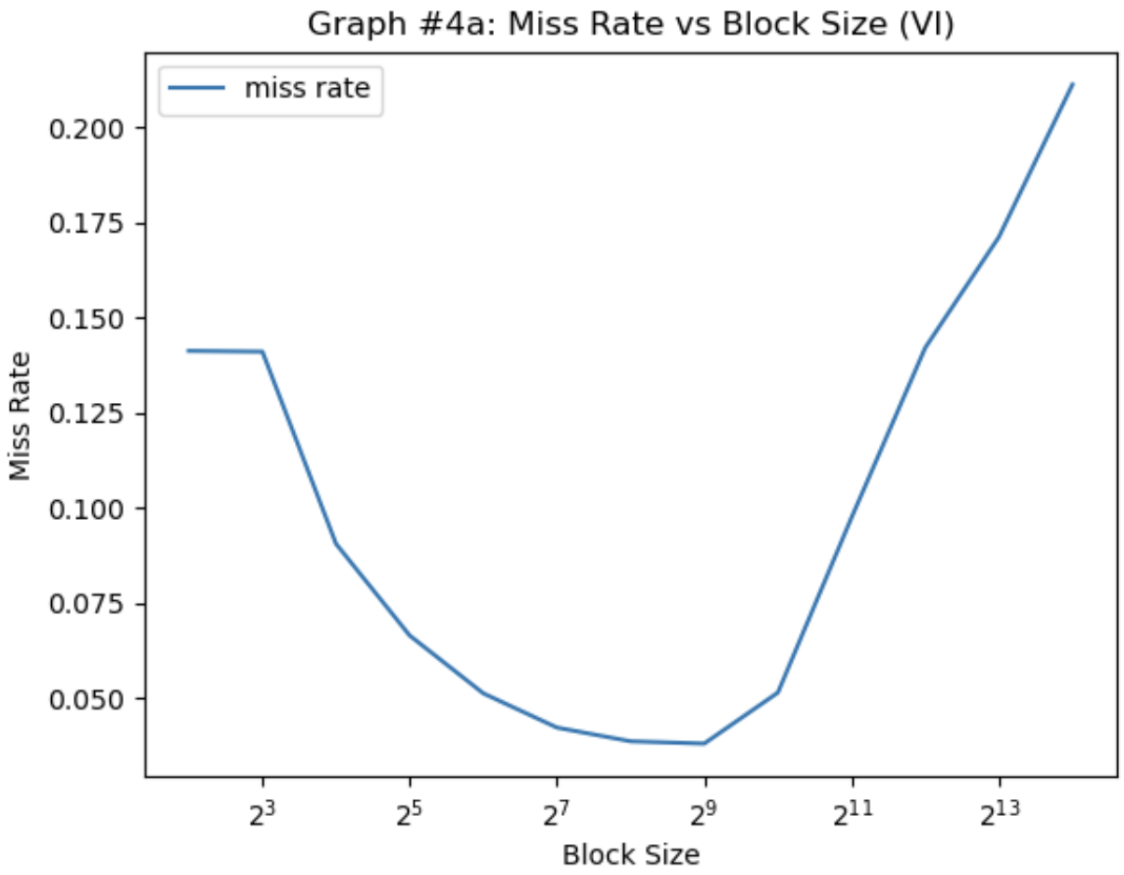

Figure 4: Graph 4A

14. Explain the observed miss rate associated with a small block size.

Miss rate starts high, then decreases as block size increases from its smallest (4B). This is because of temporal locality as more data is loaded in with larger block sizes.

15. Explain the observed miss rate associated with a large block size.

Miss rate starts to increase again as block size increases past 2^9 bytes, as now the larger block size increases conflict misses. Evicting a block can lead to more misses due to reduced spatial locality, so more data is fetched than necessary and replaces common use cases.

16. What is the block size with the lowest miss rate?

From the graph, the block size with the lowest miss rate appears to be 2^9 bytes.

Graph 4B: Miss Rate vs Block Size for 2 and 4 Cores (VI)

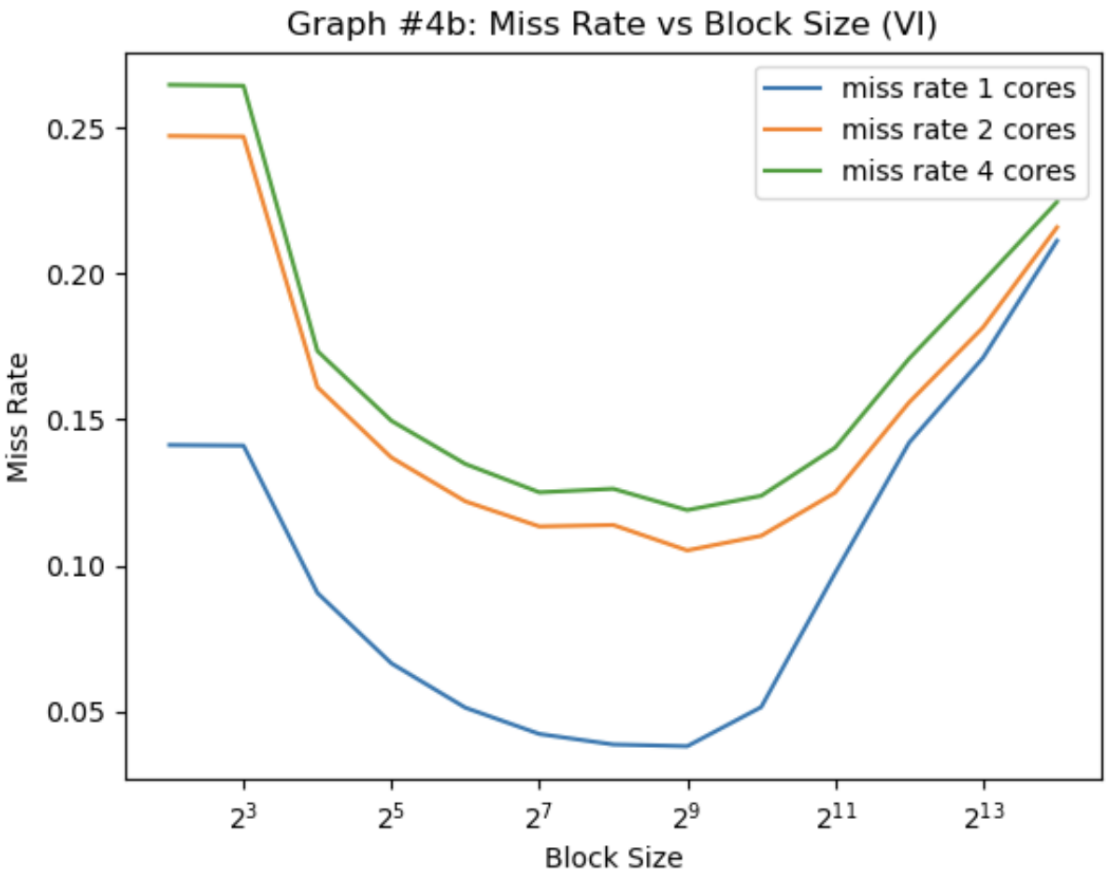

Figure 5: Graph 4B

17. Why does the miss rate get worse with more cores?

Since there is only 1 cached copy allowed, when multiple cores want to access the same cache line, the VI protocol would cause a load/store miss: not only does a cache line need to be loaded in for a hit, but now a core must have a valid copy–else, a load/store miss would occur and it must transition to the valid state to obtain a hit. Furthermore, the VI protocol incurs overhead, which can contribute to cache misses. As the number of cores increases, the overhead associated with maintaining cache coherence also increases, potentially leading to a higher miss rate.

18. If the miss rate is so bad, why would one use the VI protocol over no protocol?

With no protocol, we would have issues with cache coherence; updates (writes) in one core would not be reflected in other cores, which would cause random inconsistencies in code execution depending on which core receives which instructions first; instructions would execute based on outdated values.

Graph 5: Miss Rate vs Block Size (MSI)

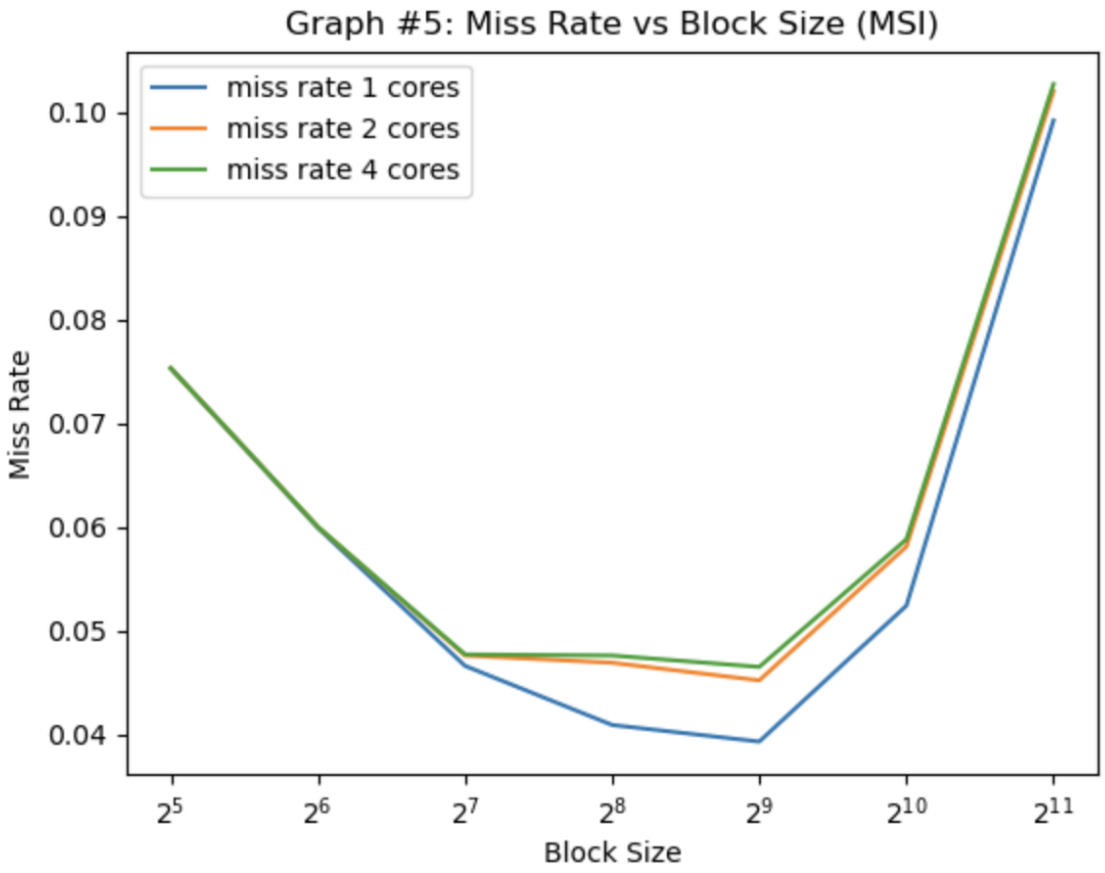

Figure 6: Graph 5

19. For the 1 core trace, which has a lower miss rate: the VI protocol or the MSI protocol? Explain why.

For 1 core, both protocols will have around the same miss rate given that other parameters are kept constant. This is because neither protocol is necessary when there is a single core reading and writing because there will be no cache coherency issues.

20. For the 4 core trace, which has a lower miss rate: the VI protocol or the MSI protocol? Explain why.

The miss rate of the MSI is much less with a higher number of cores. This is because this type of protocol is better for avoiding issues with coherency by using an extra state that can look at modified and shared data. Furthermore, the VI protocol only allows one cached copy, while MSI allows multiple. Since the MSI protocol allows multiple read-only copies, it is less likely to miss.

21. For a 2 core trace, with a block size of 64B, what fraction of bus snoops by Core 0 are "hits" (i.e., the LD_MISS or ST_MISS on the bus is for a cache line that is currently valid in Core 0's cache).

This was run on "trace.2t.long.txt" with 64 B block size, 4-way associativity, and 2 cores. We had 179 snoop hits and 5743 bus snoops, for a ratio of $179/5743 = 0.031 = 3.1\%$.

22. For a 4 core trace, with a block size of 64B, what fraction of bus snoops by Core 0 are "hits"?

This was run on "trace.4t.long.txt" with 64 B block size, 4-way associativity, and 4 cores. We had 521 snoop hits and 17533 bus snoops, for a ratio of $521/17533 = 0.0297 = 2.97\%$.