

Cache Simulation

By- Harsh Gupta (IMT2023121)

Shivek Ranjan (IMT2023042)

Navish Malik (IMT2023060)

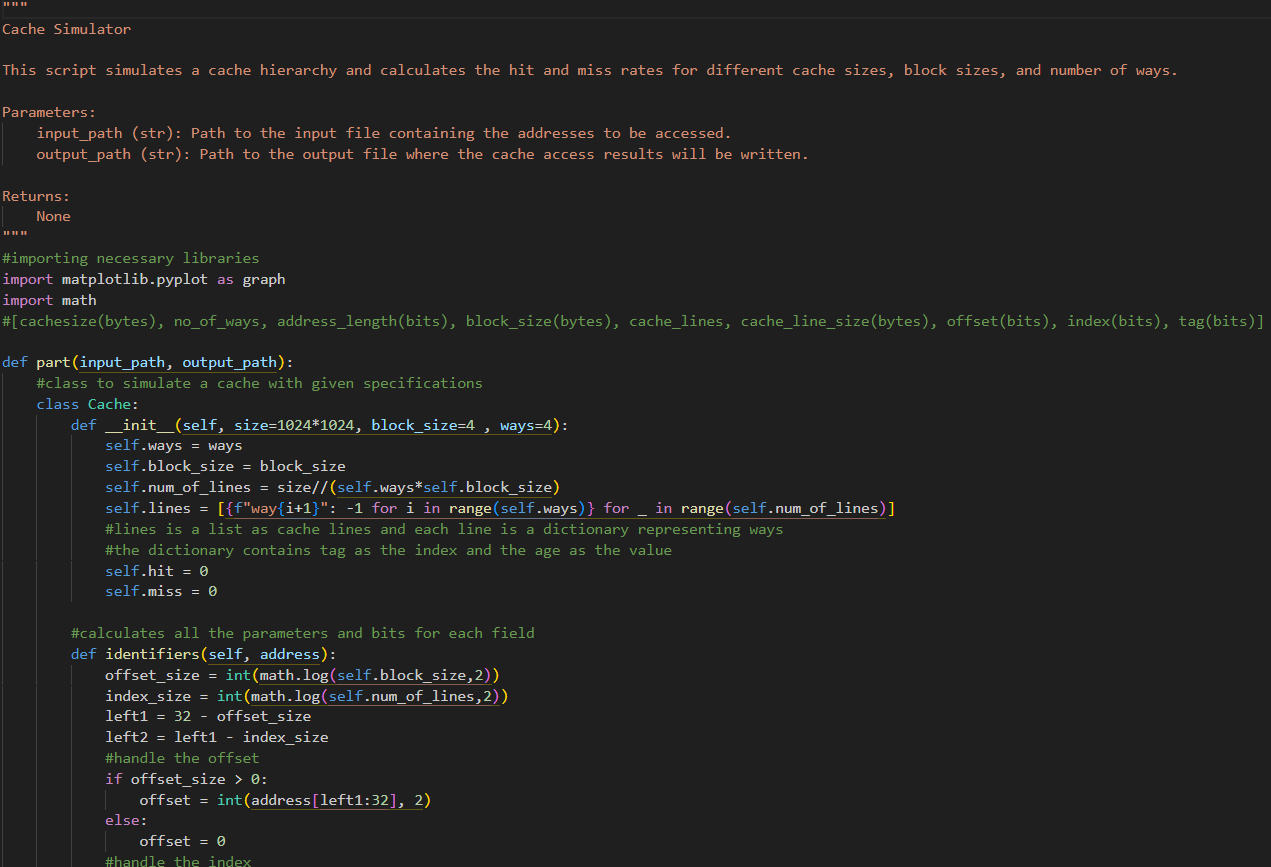
**Caches | Computer Architecture | September 6, 2024**

**Introduction**

This project involves the development of a \*\*Cache Simulator\*\*, which emulates the behavior of a cache memory system in a computing environment. The main objective of the simulator is to study and analyze how various cache parameters—such as cache size, block size, and associativity (number of ways)—affect the performance of the cache, specifically in terms of hit and miss rates. The simulator helps visualize the effectiveness of different cache configurations and access patterns on the overall efficiency of the system. In this project, trace files containing memory addresses accessed by real-world programs (such as `gcc`, `gzip`, `mcf`, `swim`, and `twolf`) are used as input to the simulator. The simulator evaluates cache performance for each trace by simulating cache access and calculating hit and miss rates for various cache parameters. Additionally, the project generates graphical representations of how hit and miss rates vary with cache size, block size, and associativity, which allows for a deeper understanding of how these factors influence cache performance.

This report details the implementation of the cache simulator, explains the experimental setup and testing with different trace files, and presents the results in the form of individual and combined graphs to illustrate the effects of the varying cache parameters.

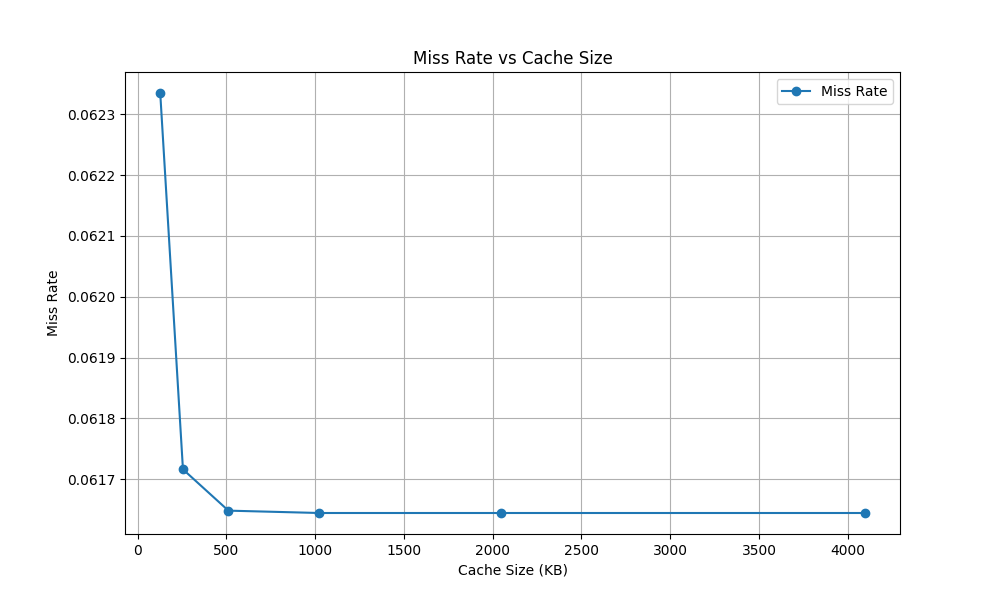
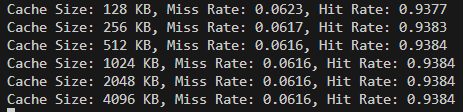
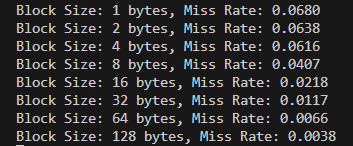
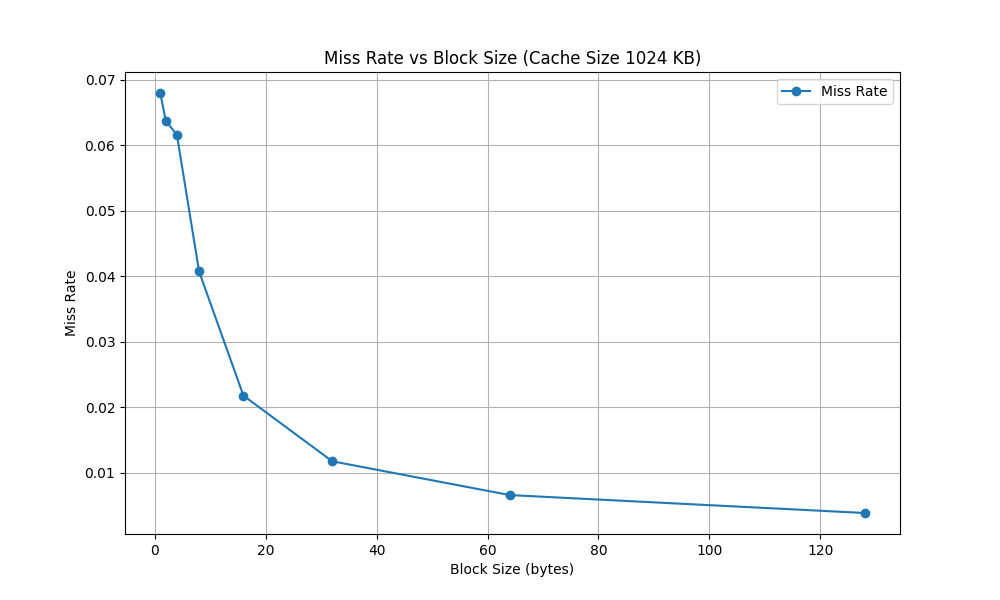
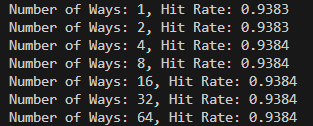
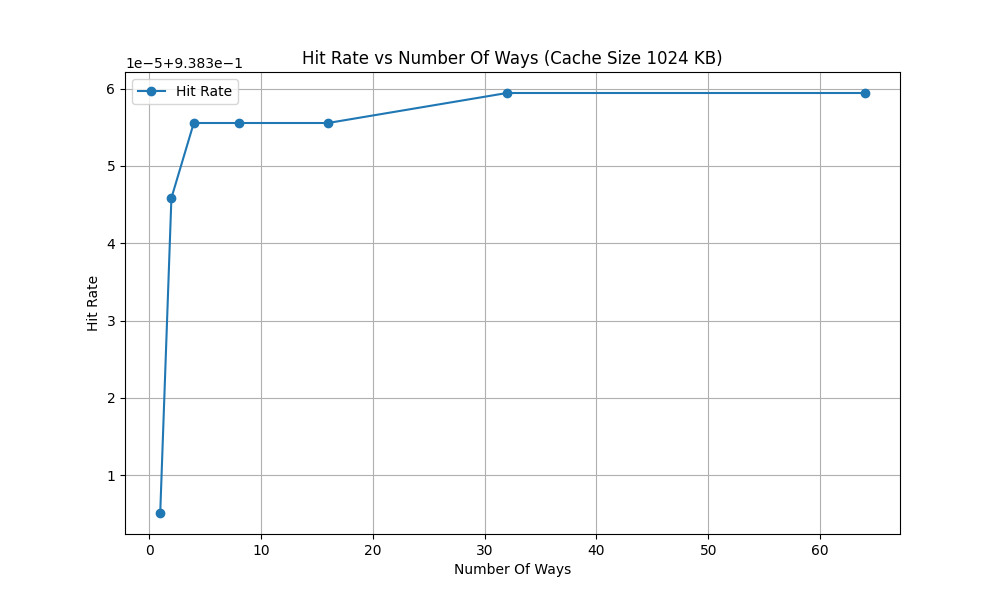
(Snippet of the python code for cache simulation)

****

Note- Some graphs which have some equation written above may seem wrong (like 1e-5+9.383e-1), they are the y-axis (say y) multiplied by this (y\*1e-5+9.383e-1)

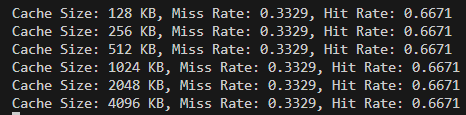
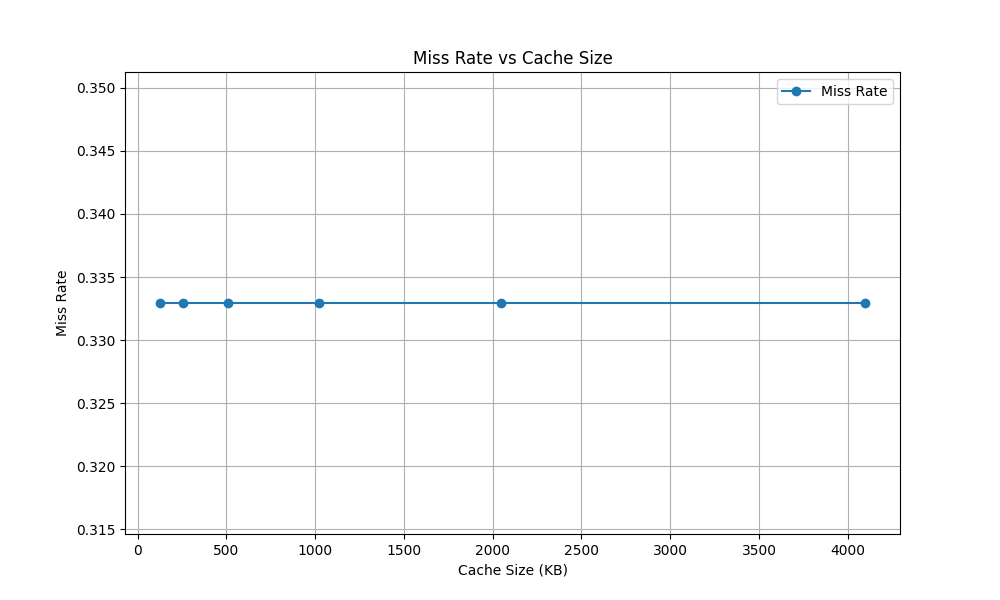
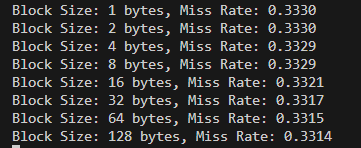
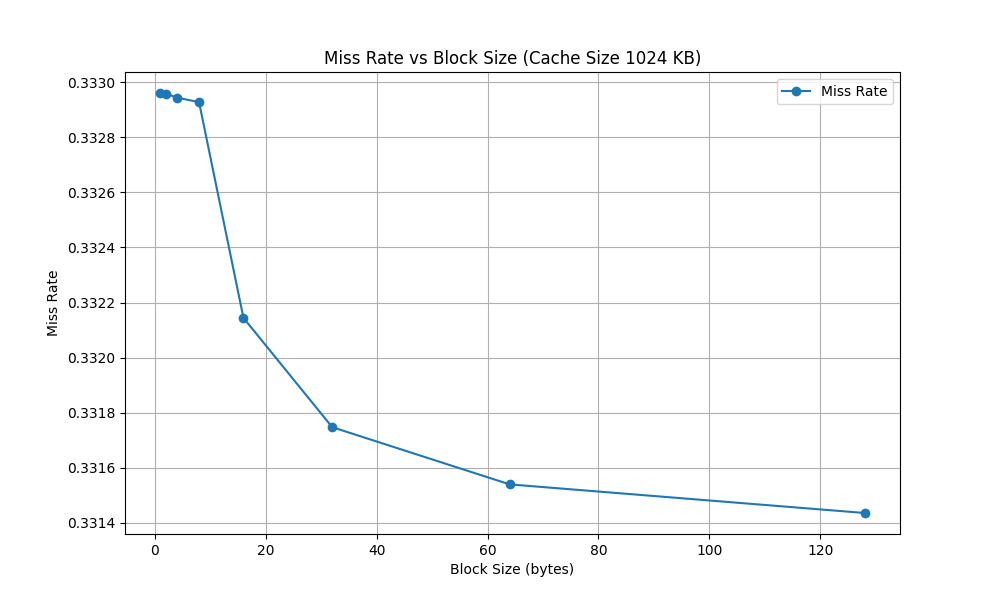
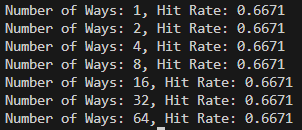
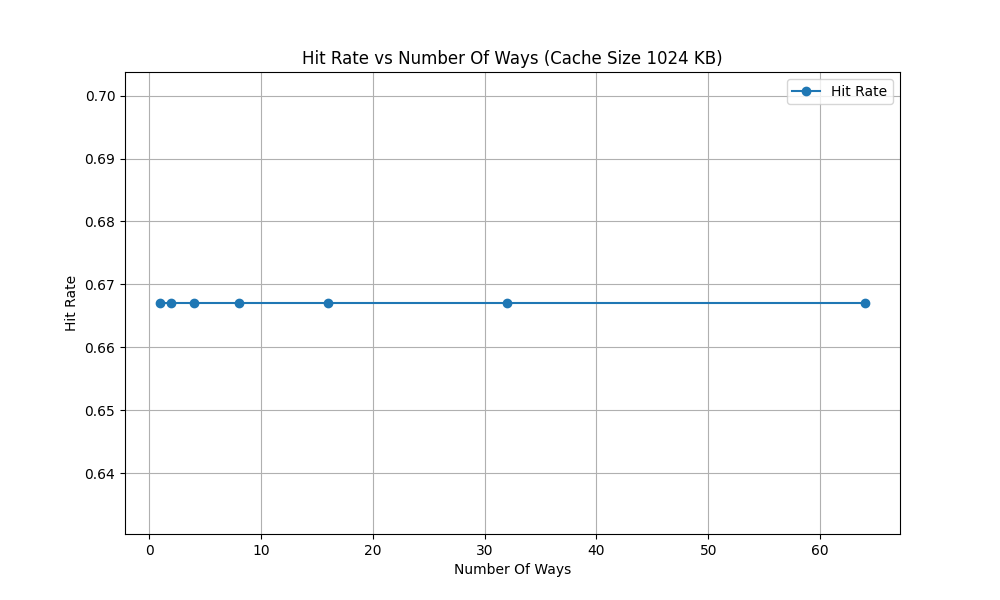
**Trace 1: gcc**

The observation shows that as cache sizes increases, the miss rate slightly decreases (though due to graph scaling it seems large). Miss rates significantly dropped when block sizes were varied. Higher associativity (no of ways) levels increase hit rate, thus increasing associativity improves cache efficiency.

1. 
2. 
3.  
4.  

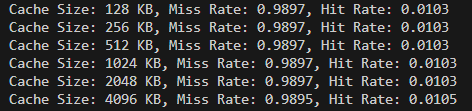
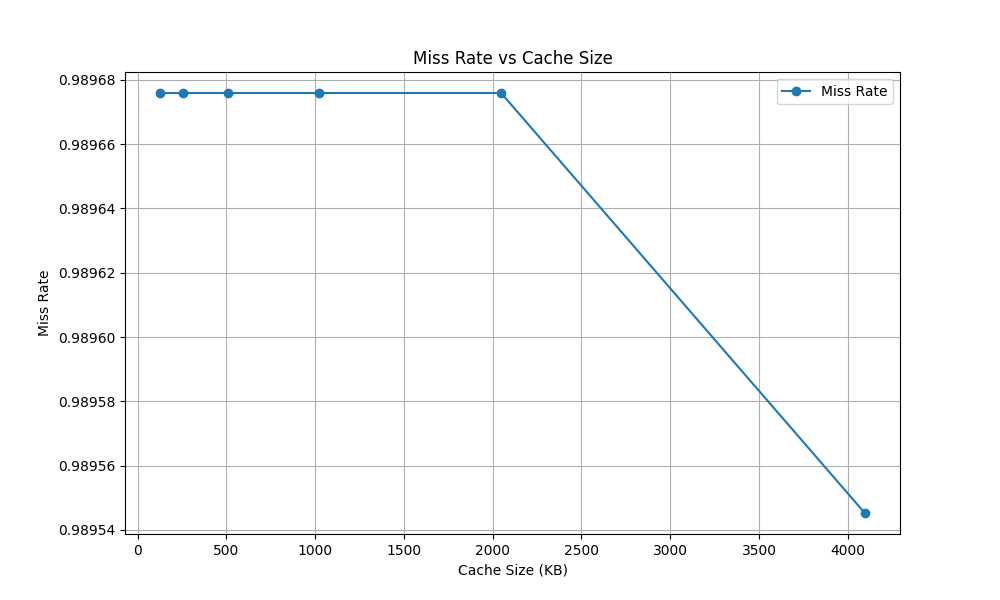
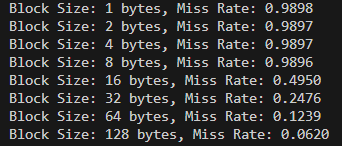
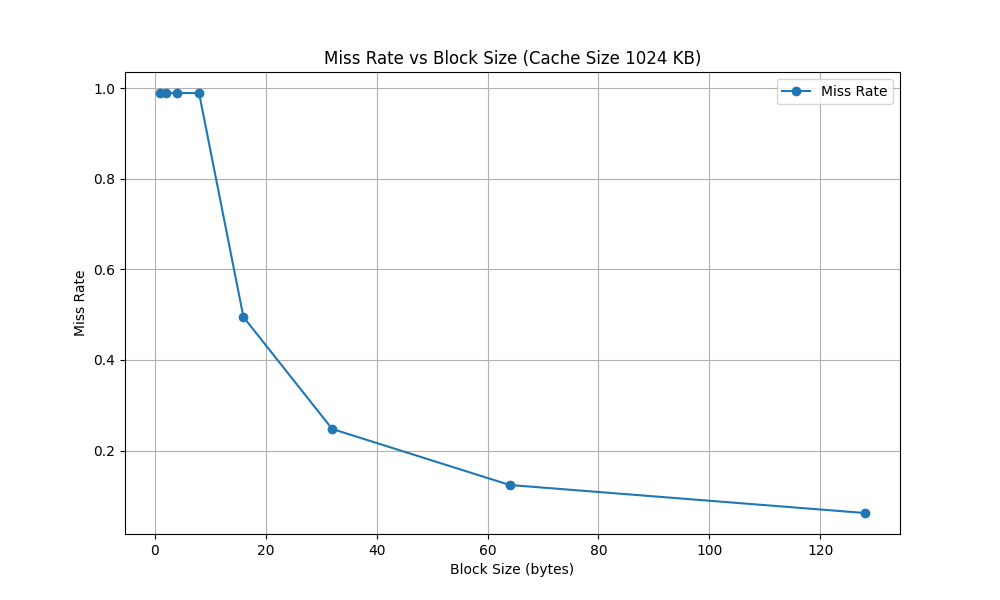
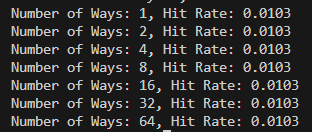
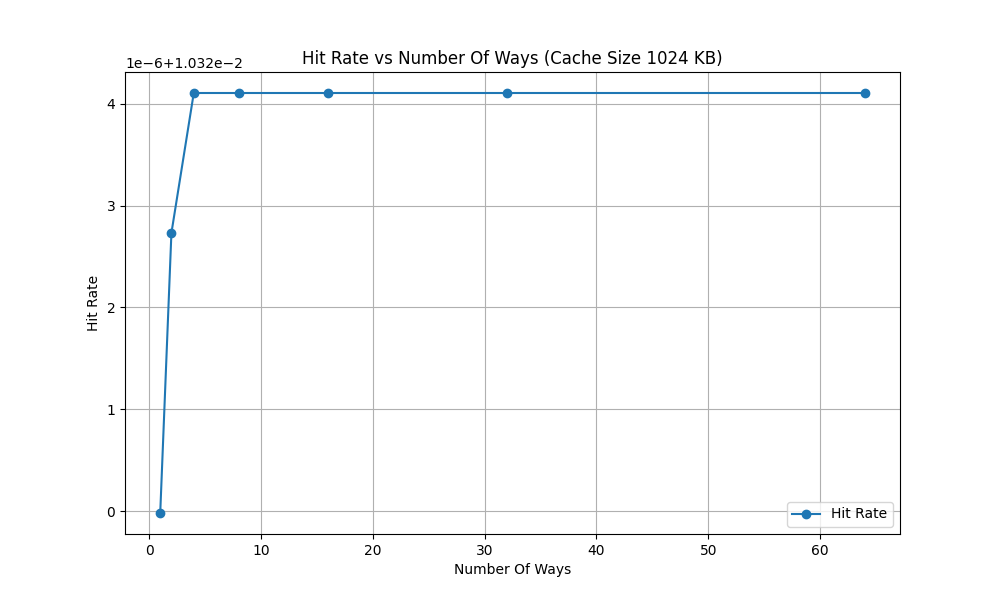
**Trace 2: gzip**

The miss rate at 0.3329 across all cache sizes. Block size variations have minimal impact on miss rates (though due to graph scaling it seems large), especially for very small or large block sizes. Similarly, varying associativity shows no change in performance, keeping hit rates stable.

1. 
2.  
3.  
4.  

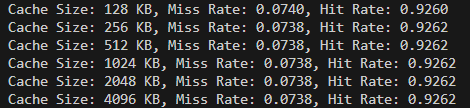
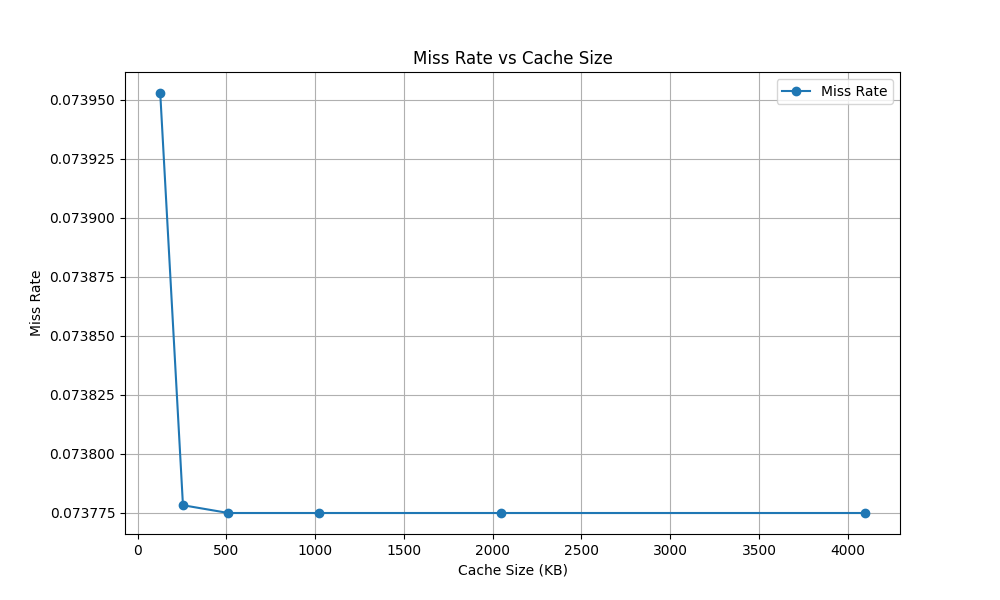
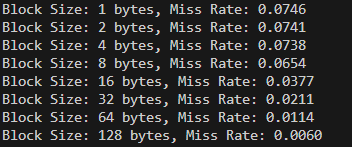
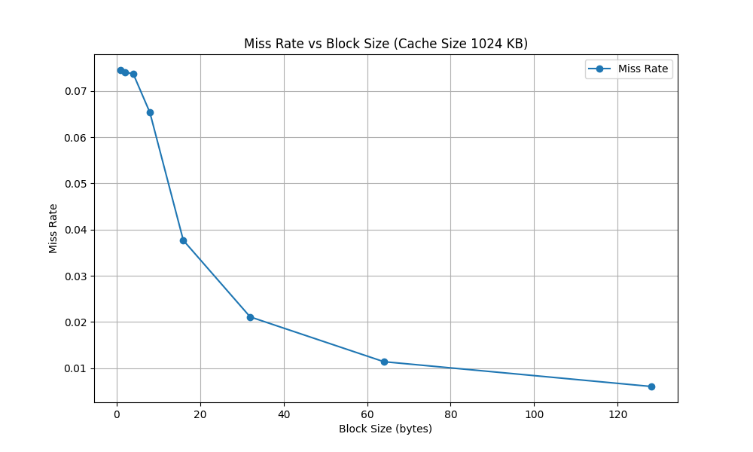
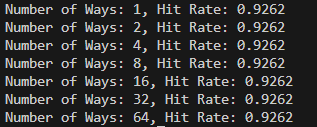
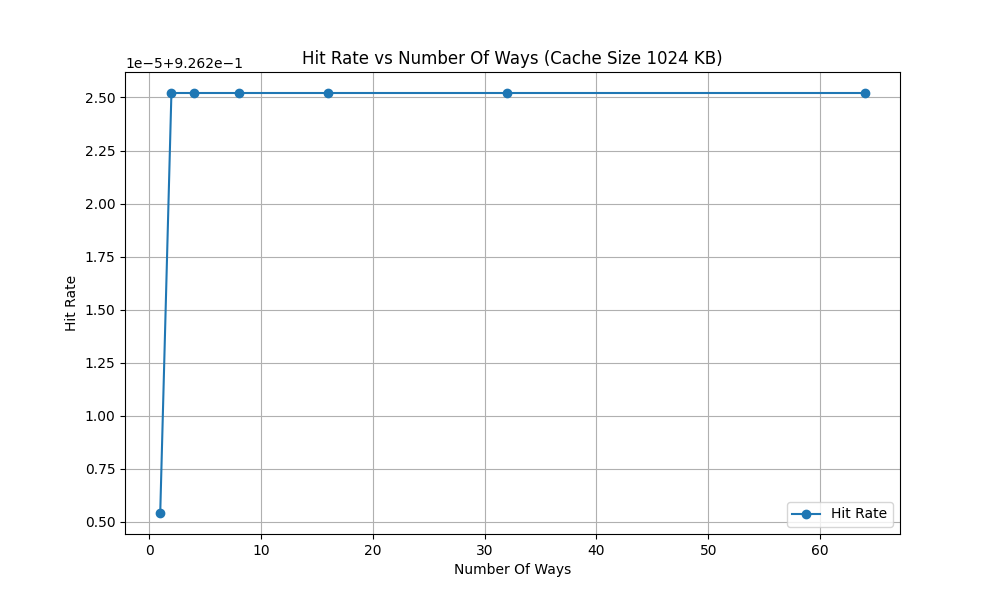
**Trace 3: mcf**

The data indicates that increasing cache size has little impact on hit and miss rates (though due to graph scaling it seems large), whereas block size plays a more significant role in performance. Larger block sizes, in particular, lead to a noticeable reduction in miss rates. For associativity, there is minimal effect on performance, with hit rates stabilizing 0.0103(minimal change in further decimal places).

1. 
2.  
3.  
4.  

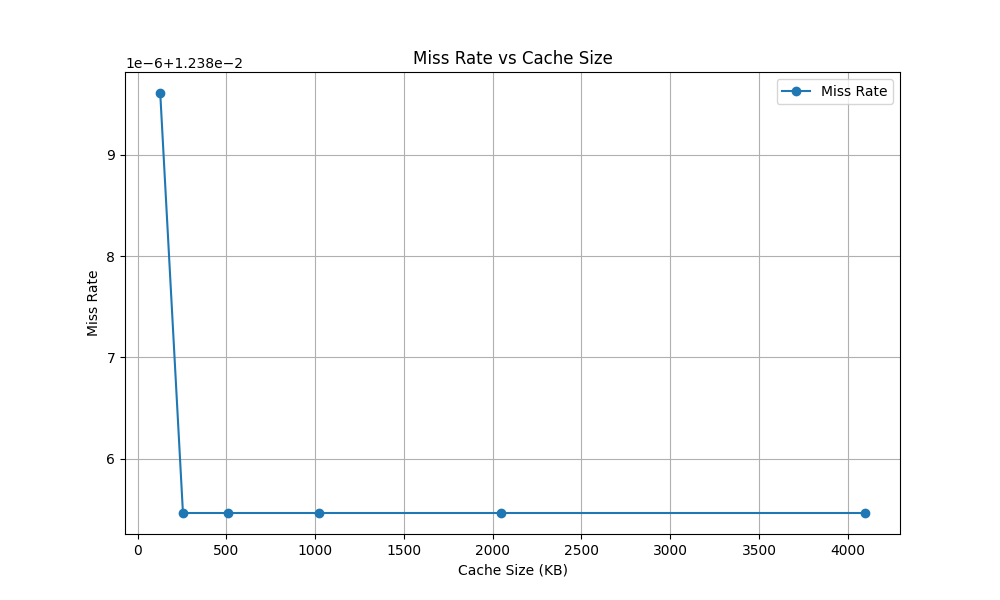
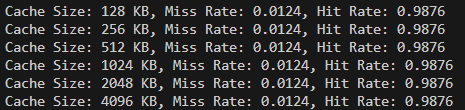
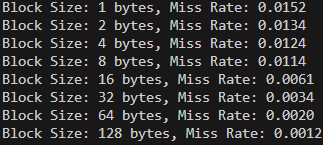
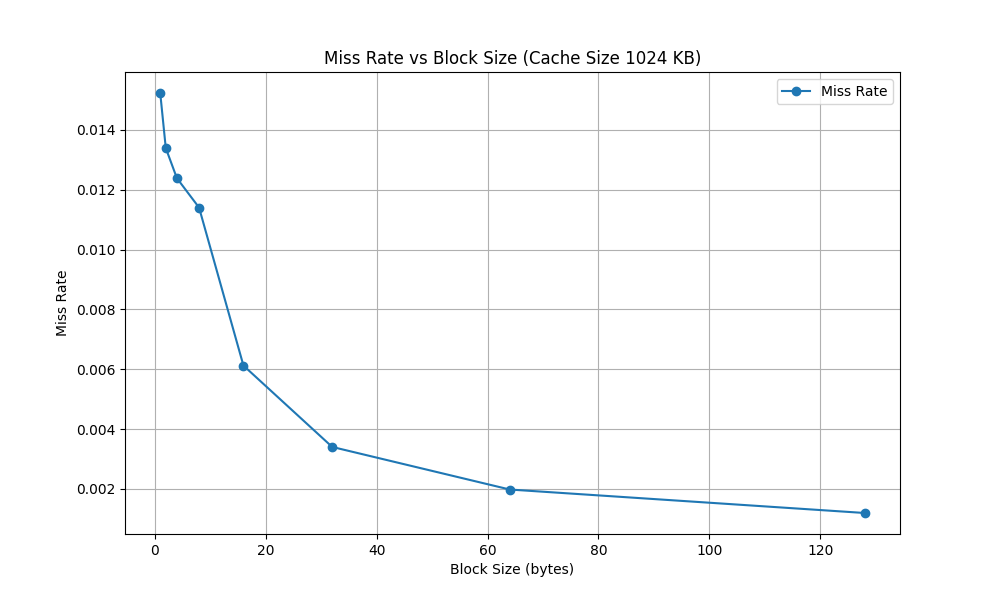
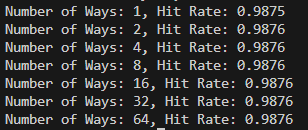
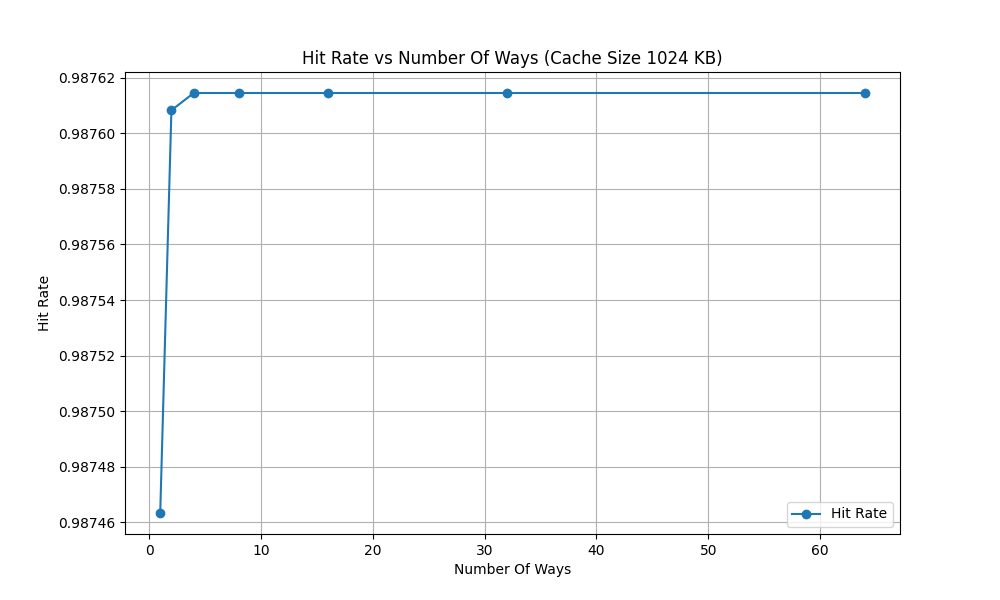
**Trace 4: swim**

The data reveals that cache size has minimal influence, with the miss rate fixed at 0.0738 (approx.). However, block size variations have a more pronounced effect, with larger block sizes leading to a significant reduction in the miss rate. Increasing associativity beyond 1-way shows no improvement in performance, as both hit and miss rates remain consistent across all levels of associativity. The analysis of the cache simulation across various trace files (`gcc`, `gzip`, `mcf`, `swim`, and `twolf`) reveals key insights into how different cache parameters—cache size, block size, and associativity—affect performance, as indicated by hit and miss rates. Each trace file represents a unique workload, and the impact of changing parameters varied accordingly.

1. 
2.  
3.  
4.  

**Trace 5: twolf**

The data indicates that increasing cache size has no notable effect on performance, with the hit rate consistently around 0.9876 and the miss rate around 0.0124. Reducing the block size results in higher miss rates, particularly for sizes below 16 bytes, while larger block sizes significantly decrease misses. Increasing associativity beyond 1-way offers no performance gains, as hit and miss rates remain steady regardless of the associativity level.

1. 
2. 
3.  
4.  

**Conclusion**

The analysis of the cache simulation across various trace files (`gcc`, `gzip`, `mcf`, `swim`, and `twolf`) reveals key insights into how different cache parameters—cache size, block size, and associativity—affect performance, as indicated by hit and miss rates. Each trace file represents a unique workload, and the impact of changing parameters varied accordingly.

1. Across most traces, increasing cache size had minimal impact miss rates. For example, in the `gzip` and `twolf` traces, miss rates remained almost constant regardless of the cache size, highlighting that the workload either fits comfortably in the cache or the memory access patterns are not highly sensitive to cache size changes. However, the `gcc` trace showed a slight improvement in miss rate with increased cache size, likely due to the program's tendency to reuse a larger working set of data, making it more sensitive to cache size changes.
2. Block size variations had a more pronounced effect on cache performance. In general, larger block sizes led to a noticeable reduction in miss rates across most traces, particularly in `mcf` and `swim`, which likely have spatial locality (accessing data in nearby memory locations). However, reducing the block size below a certain threshold (such as 16 bytes) in the `twolf` trace resulted in higher miss rates, emphasizing that smaller blocks can lead to increased cache misses due to more frequent reloads of data from memory.
3. The impact of associativity varied across traces, but in most cases, increasing associativity beyond 1-way provided limited performance improvement. In the `gcc` trace, higher associativity levels led to an increase in hit rate, suggesting that the trace benefits from more flexible placement of data within the cache. On the other hand, traces like `gzip`, `swim`, and `twolf` showed no significant performance change beyond 1-way associativity, implying that these workloads did not suffer from high levels of conflict misses.