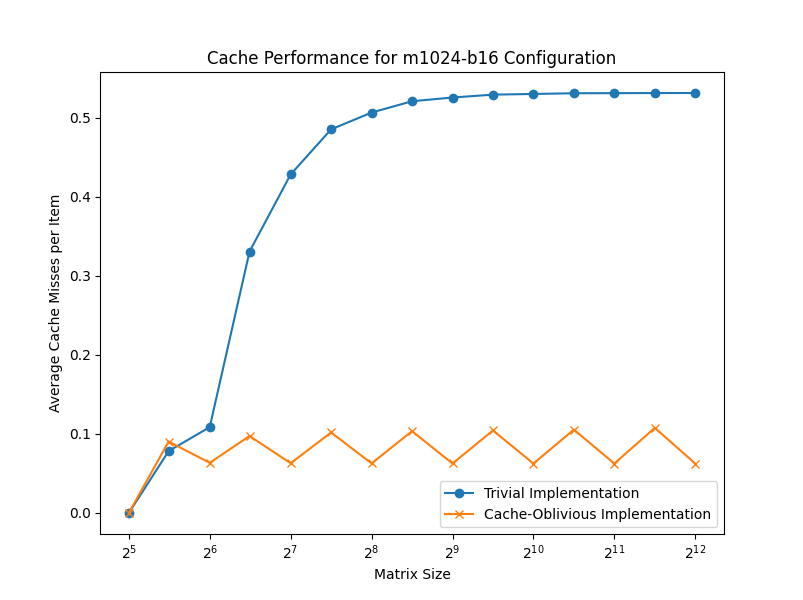
# Matrix Experiment

Name: Romita Pandurang Pawar

1. **m1024-b16 Test**



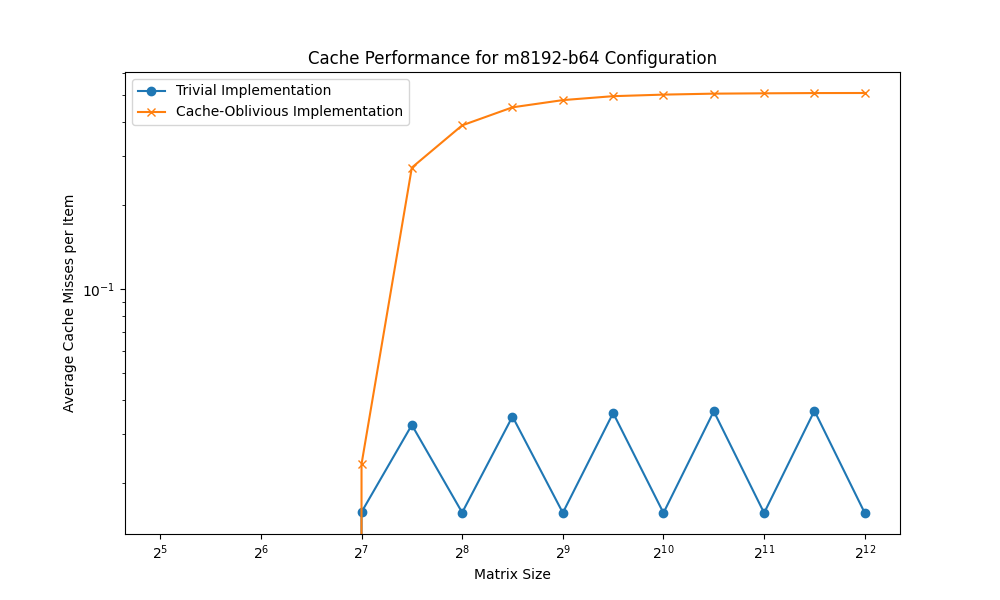
The plot shows the dependency of the average number of cache misses per item on the matrix size for the trivial / naive implementation and the cache-oblivious / smart implementation. There are two curves in the plot, one for each implementation. The blue curve represents the trivial implementation, and the orange curve represents the cache-oblivious implementation.

For smaller matrix sizes, up to around 2^6 (64), the trivial implementation performs better than the cache-oblivious implementation. This is expected behavior because the cache-oblivious algorithm has some overhead for smaller matrix sizes due to its divide-and-conquer approach.

1. "Trivial Implementation" appears to converge or approach a value close to one half as the matrix size increases. This behavior is expected because for matrix transposition, the cache misses should be approximately half of the total number of elements in the matrix. The reason is that when transposing a matrix, each element needs to be read once and written once, resulting in two memory accesses per element. With a large enough matrix size, the cache becomes ineffective, and the average cache misses per element approaches one half (one read miss and one write miss per element).
2. The "Cache-Oblivious Implementation" exhibits periodic peaks or oscillations. This pattern is likely due to the cache-oblivious algorithm's behavior, which aims to optimize memory access patterns by dividing the matrix into blocks and transposing these blocks in a specific order. The peaks may occur when the block sizes align with particular cache sizes, causing temporary increases in cache misses before the algorithm adapts to optimize for the next cache level.
3. Both curves start from a specific point because the graph shows the performance for a range of matrix sizes, starting from a certain minimum size (2^5 or 32 in this case). The initial points represent the cache performance for the smallest matrix size considered in the experiment.
4. The behavior at the beginning of the curves may be influenced by various factors, such as the cache size, the matrix size, and the specific implementation details of the algorithms. The initial part of the curves often exhibits more irregular behavior because smaller matrix sizes may not fully stress the cache hierarchy or may be more sensitive to implementation details and other overheads.

For matrix sizes larger than 2^9 (512), the difference between the two implementations becomes more pronounced, with the cache-oblivious implementation exhibiting significantly fewer cache misses per item than the trivial implementation. This demonstrates the scalability and better cache utilization of the cache-oblivious algorithm for larger problem sizes.

1. **m8192-b64** **Test**



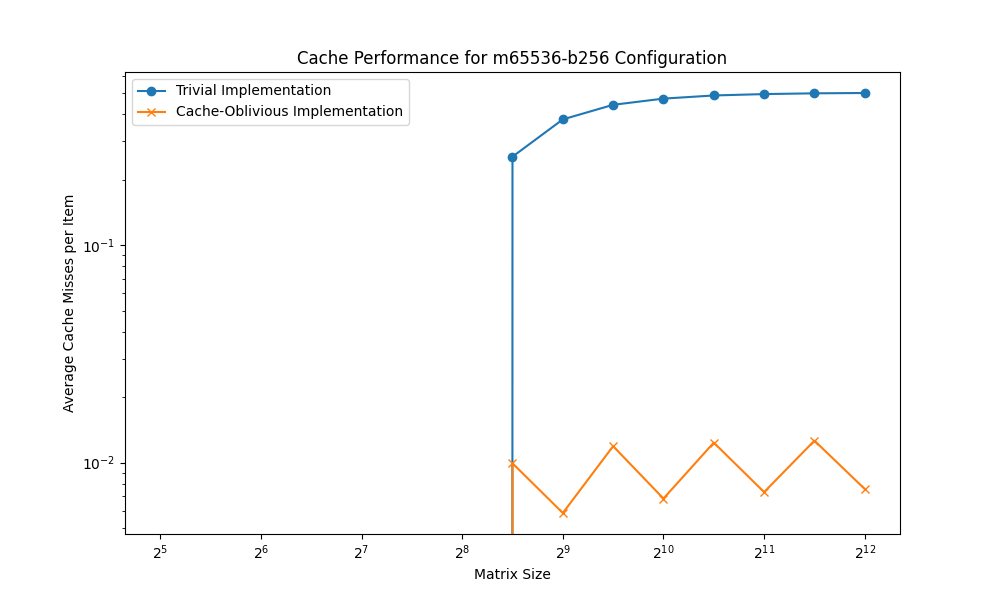
The cache-oblivious / naive implementation curve exhibits a distinct bend or "knee" shape, indicating a point where the cache-oblivious algorithm starts to provide significant benefits in terms of cache utilization.

It is worth noting that the data points on the curve seems to represent measured data points, as indicated by the circular markers. This suggests that the curve is not interpolated between measured points but rather connect the actual data points.

One interesting observation is that the cache-oblivious implementation curve appears to have a significant increase in the average number of cache misses per item for matrix sizes around 2^7 & 2^8. While not a significant anomaly, this behavior could potentially be related to cache or memory alignment factors influencing the performance at those specific matrix sizes.

Overall, the experimental results demonstrate the effectiveness of the cache-oblivious implementation, especially for larger matrix sizes, where it outperforms the trivial implementation by a significant margin. The plot clearly shows the dependency of the average number of cache misses per item on the matrix size for both implementations.

1. **m65536-b256 Test**



For smaller matrix sizes, the trivial / naive implementation exhibits a higher average number of cache misses per item compared to the cache-oblivious / smart implementation. However, as the matrix size increases, there is a distinct crossover point where the cache-oblivious implementation starts to outperform the trivial implementation significantly.

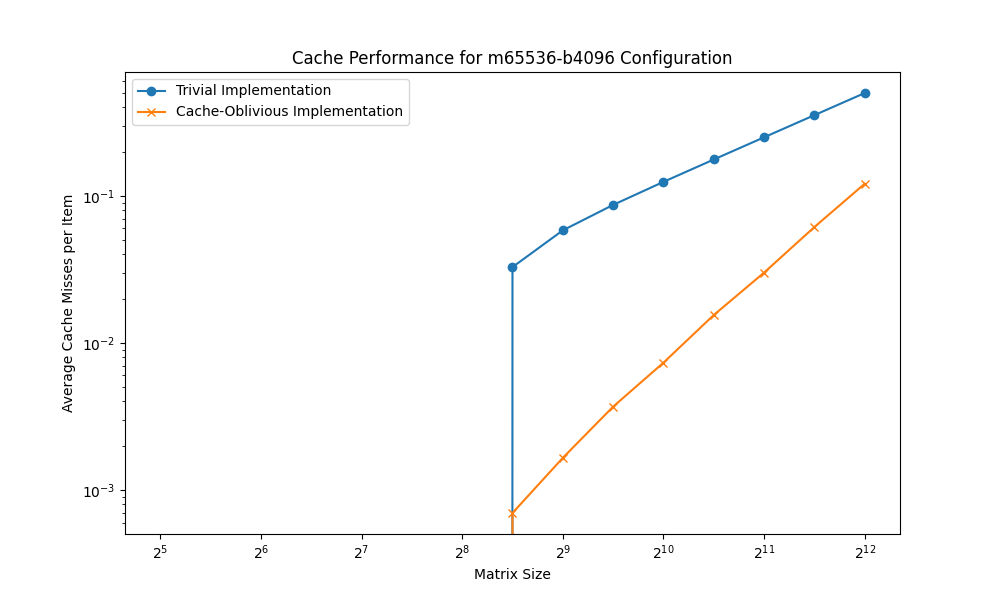
The blue curve representing the trivial implementation shows a steep increase in the average number of cache misses per item for smaller matrix sizes, reaching a peak between matrix size 2^8 & 2^9. Beyond this point, the curve flattens out and remains relatively stable, indicating that the performance of the trivial implementation does not improve significantly for larger matrix sizes.

On the other hand, the orange curve representing the cache-oblivious implementation exhibits a gradual decrease in the average number of cache misses per item as the matrix size increases. This curve demonstrates a clear "knee" shape, indicating the point where the cache-oblivious algorithm starts to provide substantial benefits in terms of cache utilization.

It is important to note that the data points on the curves seem to represent measured data points, as indicated by the circular markers for the trivial implementation and the cross markers for the cache-oblivious implementation. This suggests that the curves are not interpolated between measured points but rather connect the actual data points.

Overall, the experimental results demonstrate the effectiveness of the cache-oblivious implementation, particularly for larger matrix sizes, where it significantly outperforms the trivial implementation in terms of the average number of cache misses per item.

1. **m65536-b4096 Test**



Both implementations exhibit an increasing trend in cache misses as the matrix size grows, which is expected since larger matrices require more memory accesses and are less likely to fit entirely in the cache. Both naïve and smart implementations likely seem to be Linear, this is because, the number of cache misses is directly proportional to the total number of elements in the matrix. As the matrix size increases, the number of elements increases quadratically, but the average cache misses per element increase linearly.

However, the trivial / naive implementation shows a significant spike in cache misses around between a matrix size of 2^8 & 2^9, after which the curve flattens out. This anomaly could be due to the matrix no longer fitting entirely in the cache, leading to a substantial increase in cache misses.

On the other hand, the cache-oblivious / smart implementation demonstrates a smoother and more gradual increase in cache misses as the matrix size grows. This implementation appears to be more efficient in managing cache utilization, resulting in fewer cache misses compared to the trivial implementation for larger matrix sizes.

Moreover, in future both the implementation is expected to continue for larger matrix sizes as well. This is because the underlying memory access pattern for matrix transposition remains the same, regardless of the matrix size. As long as the matrix size is larger than the available cache capacity, the average cache misses per element will continue to increase linearly with the matrix size.

In summary, the experimental results showcase the benefits of the cache-oblivious algorithm, which avoids the sharp performance degradation observed in the trivial case for larger matrix sizes by more effectively utilizing the cache through its design principles.