Name : Youssef Mohamed Mahmoud El-raggal

ID : 7806

Group : 2 **section**: 2

Matrix Multiplication

The code is a Python script that performs simple matrix multiplication without blocking and matrix multiplication using blocking and measures the time taken to perform the operation for different matrix sizes and block sizes. The purpose of using blocking is to optimize matrix multiplication performance by reducing the number of cache misses.

We will run the code three times in each run the output will be two figures a table and a graph.

1st run : Matrix sizes (64,128,256,512)

Block sizes (4,8,16,32)

2nd run : Matrix sizes (64,128,256,512)

Block sizes (1,4,8,16,32)

3rd run : without blocking matrix multiplication

1st run outputs :

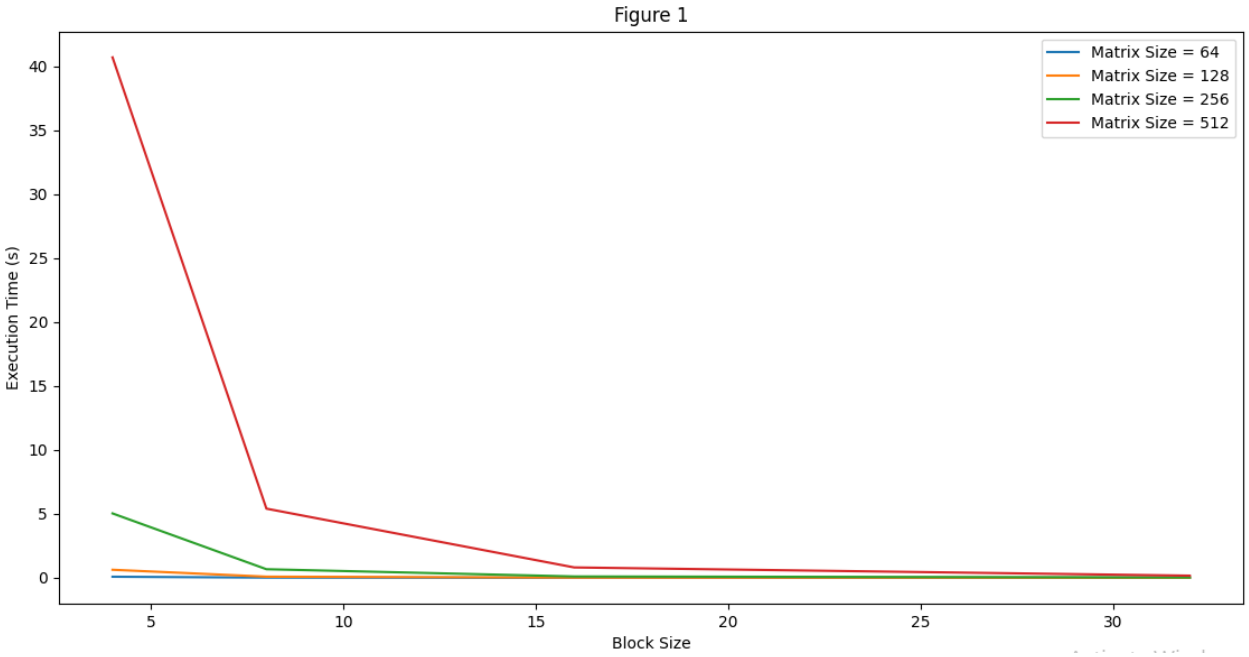

Figure 2
Matrix Multiplication with Blocking Results

	4	8	16	32
64	0.0780	0.0110	0.0010	0.0010
128	0.6236	0.0839	0.0120	0.0030
256	5.0339	0.6666	0.0999	0.0220
512	40.7219	5.4067	0.8065	0.1619

Observation

Based on the results of run on different matrix sizes and block sizes, we can analyze the performance of the matrix multiplication algorithm using blocking.

For smaller matrix sizes, you might observe that the time taken to perform the multiplication process is relatively consistent across different block sizes, and there is no significant benefit to using blocking. However, as the **matrix size increases**, you may notice that the time taken to perform the multiplication process increases significantly with larger matrix sizes. This is expected since the number of operations required to perform matrix multiplication is proportional to the square of the matrix size.

At the same time, you may observe that using blocking can lead to significant performance improvements for larger matrix sizes. This is because blocking can reduce the number of cache misses and improve memory locality, which can improve performance by reducing the time spent accessing memory.

Conclusion

The results of run can be used to determine the optimal block size for a given matrix size to achieve the best performance. This can be useful in applications that require efficient matrix multiplication, such as scientific computing, machine learning, and data analysis.

2nd run

Now we will rerun the code and make matrix multiplication with Block size =1 ,from the conclusion of the previous run you expect that the block size 1 will take the most time

Outputs:

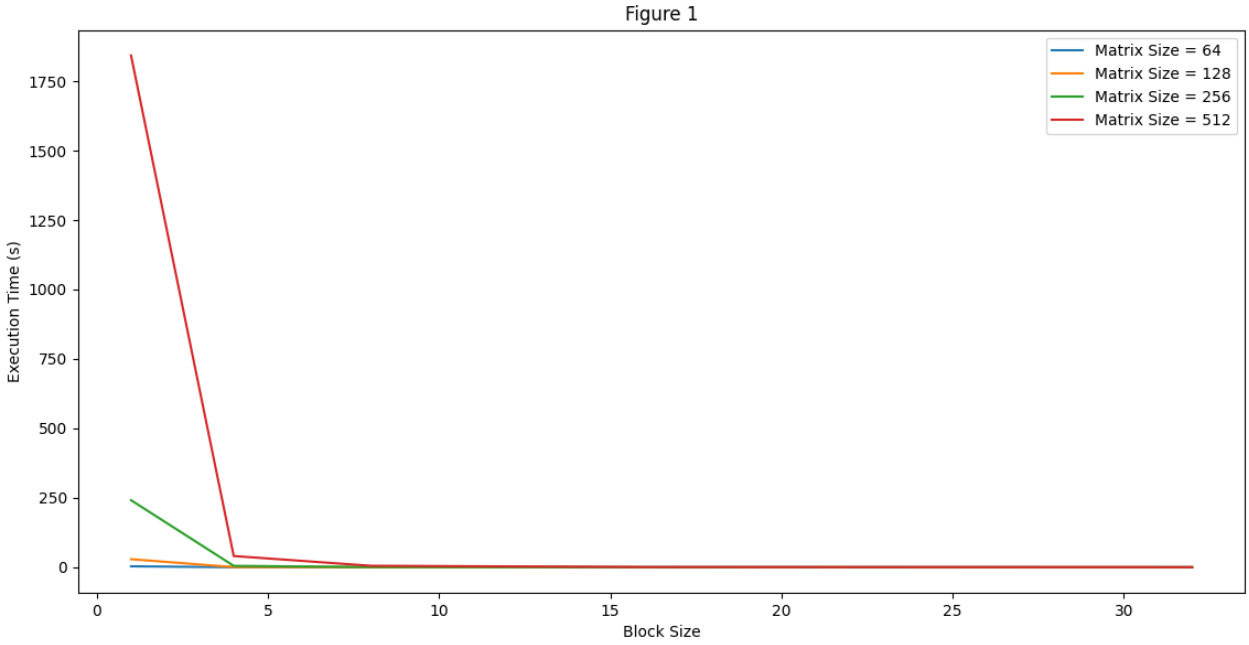

Figure 2
Matrix Multiplication with Blocking Results

	1	4	8	16	32
64	3.5048	0.0780	0.0100	0.0020	0.0010
128	29.0859	0.6286	0.0830	0.0131	0.0029
256	241.5302	5.0439	0.6746	0.1029	0.0230
512	1842.4612	40.4690	5.3787	0.8175	0.1609

3rd run outputs:

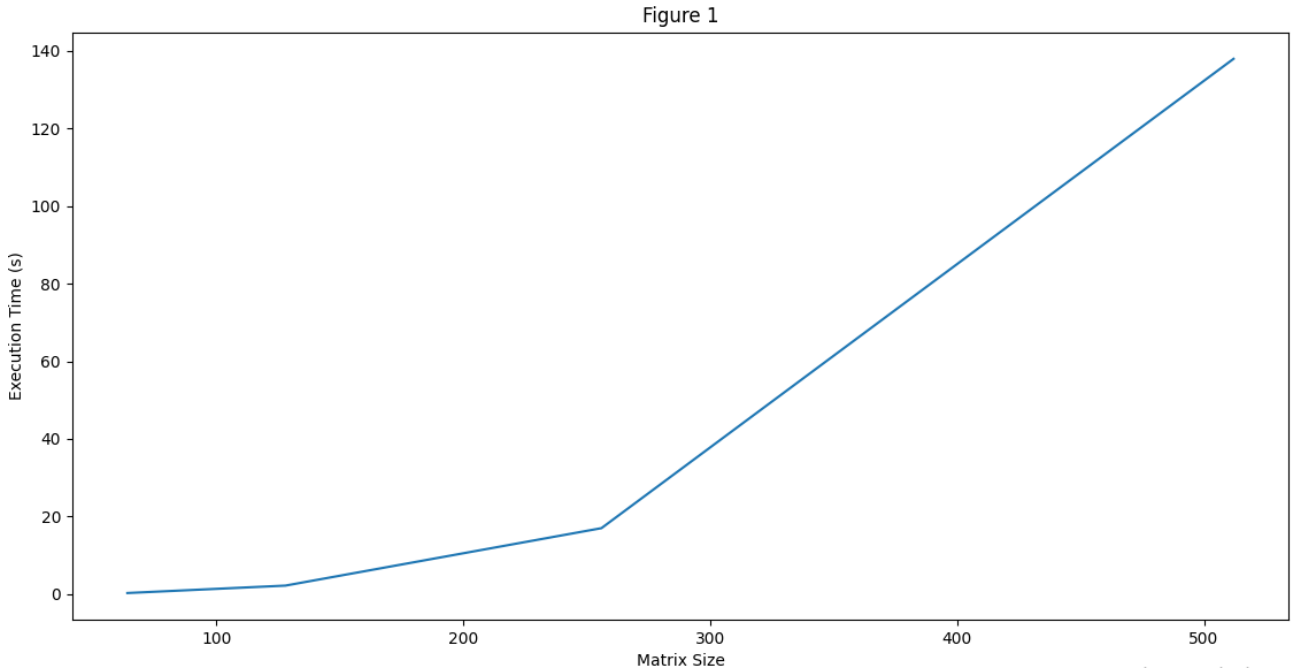

Figure 2

Matrix Size	Time Taken
64.0000	0.2618
128.0000	2.1727
256.0000	16.9845
512.0000	137.9824

Comparison between the matrix multiplication with blocking and matrix multiplication without blocking (2nd & 3rd runs)

For matrix multiplication performance, we implemented two strategies (Blocking and performing the multiplication without blocking). Blocking involves dividing the matrices into smaller blocks and performing the multiplication on these blocks, while performing the multiplication without blocking involves iterating over each element of the matrices and computing the corresponding element of the resulting matrix in a single pass through the nested loops.

Observation

When the block size is set to 1, the matrix multiplication without blocking is better in performance than multiplication with blocking, as each block contains only a single element. In this case, there is no advantage to using blocking, as the overhead of dividing the matrices into smaller blocks will actually slow down the computation.

Conclusion

- It's generally better to perform matrix multiplication without blocking when the block size is set to 1 (or very small block sizes) due to the overhead of the blocking implementation.
- For larger matrices, blocking can be more efficient as it can take advantage of caching and reduce the amount of memory access required.
- For small matrices, the overhead of dividing the matrices into smaller blocks can actually slow down the computation, making the standard algorithm more efficient.
- Therefore, the choice between using blocking and performing the multiplication without blocking ultimately depends on the specific use case, the size of the matrices, and the block size being used.