Alexandria University

Faculty of Engineering

ssp-program

Computer organization

Part 2

Dr. Layla Abo Hadid

Name: **ID:**

Zeyad shawki 7383

This generated matrices of different sizes, and tests different block sizes for each matrix size. It records the time taken for each combination of matrix size and block size, and plots the results as a line graph, with block size on the x-axis and time on the y-axis, and a separate line for each matrix size.

The results of running this will depend on the specific hardware and software environment used for the tests, but in general, we would expect to see that increasing the block size leads to faster multiplication times, up to a certain point. This is because larger block sizes allow for more efficient use of the CPU cache, which can lead to faster memory access times. However, if the block size is too large, it may lead to more cache misses, which can slow down the computation. The optimal block size will depend on the specific hardware and software environment, as well as the size and structure of the matrices being multiplied.

The code:

```
import numpy as np
import time
import matplotlib.pyplot as plt

def block_multiply(A, B, block_size):
    m, n = A.shape
    n, p = B.shape
    C = np.zeros((m, p))
    for i in range(0, m, block_size):
        for j in range(0, p, block_size):
            for k in range(0, n, block_size):
                # Compute block C[i:i+block_size, j:j+block_size]
                # as a product of blocks of A and B
                C_block = np.dot(A[i:i+block_size, k:k+block_size],
                                 B[k:k+block_size, j:j+block_size])
                # Add block C_block to the corresponding block of C
                C[i:i+block_size, j:j+block_size] += C_block
    return C

# Define matrix sizes and block sizes to test

matrix_sizes = [50, 100, ]
block_sizes = [5, 10, ]

# Measure time taken for each combination of matrix size and block size

for n in matrix_sizes:
    for p in matrix_sizes:
            # Initialize lists for storing times and block sizes
        times = []
        block_sizes_list = []
```

```
        for block_size in block_sizes:
                # Generate random matrices A and B
            A = np.random.rand(n, n)
            B = np.random.rand(n, p)

                # Measure time taken for matrix multiplication using blocking
            start_time = time.time()
            C = block_multiply(A, B, block_size)
            end_time = time.time()

                # Append time taken and block size to the lists
            times.append(end_time - start_time)
            block_sizes_list.append(block_size)

            # Plot the results for this matrix size
            plt.plot(block_sizes_list, times, label=f"Matrix size ( {n}, {p})")

# Set the x-axis and y-axis labels and title
plt.xlabel("Blocking size")
plt.ylabel("Time taken (seconds)")
plt.title("Relationship between blocking size and time taken")

# Display the legend and show the plot
plt.legend()
plt.show()
```

blocking and time:

blocking can actually decrease the time required for matrix multiplication.

This is because blocking reduces the number of cache misses and improves memory locality, which can result in faster memory access and computation. When we perform matrix multiplication in the traditional way, where each element of the output matrix is computed by summing the products of the corresponding row in the first matrix and column in the second matrix, we may need to move data between the CPU cache and main memory frequently. This can lead to a significant slowdown in performance due to the overhead of memory access.

However, by using blocking, we can reduce the number of cache misses and improve memory locality by performing matrix multiplication on smaller blocks of data that can fit entirely in the CPU cache. This can result in faster memory access and computation, which can lead to better performance overall.

That being said, the performance gains from blocking depend on various factors, such as the size of the matrices, the size of the blocks, and the particular hardware being used. So, it is important to experiment with different block sizes and matrix sizes to determine the optimal parameters for a particular hardware configuration.

The output graphs:

In figure 2 the blue line has bigger block size and bigger size than red which has smaller size and block size

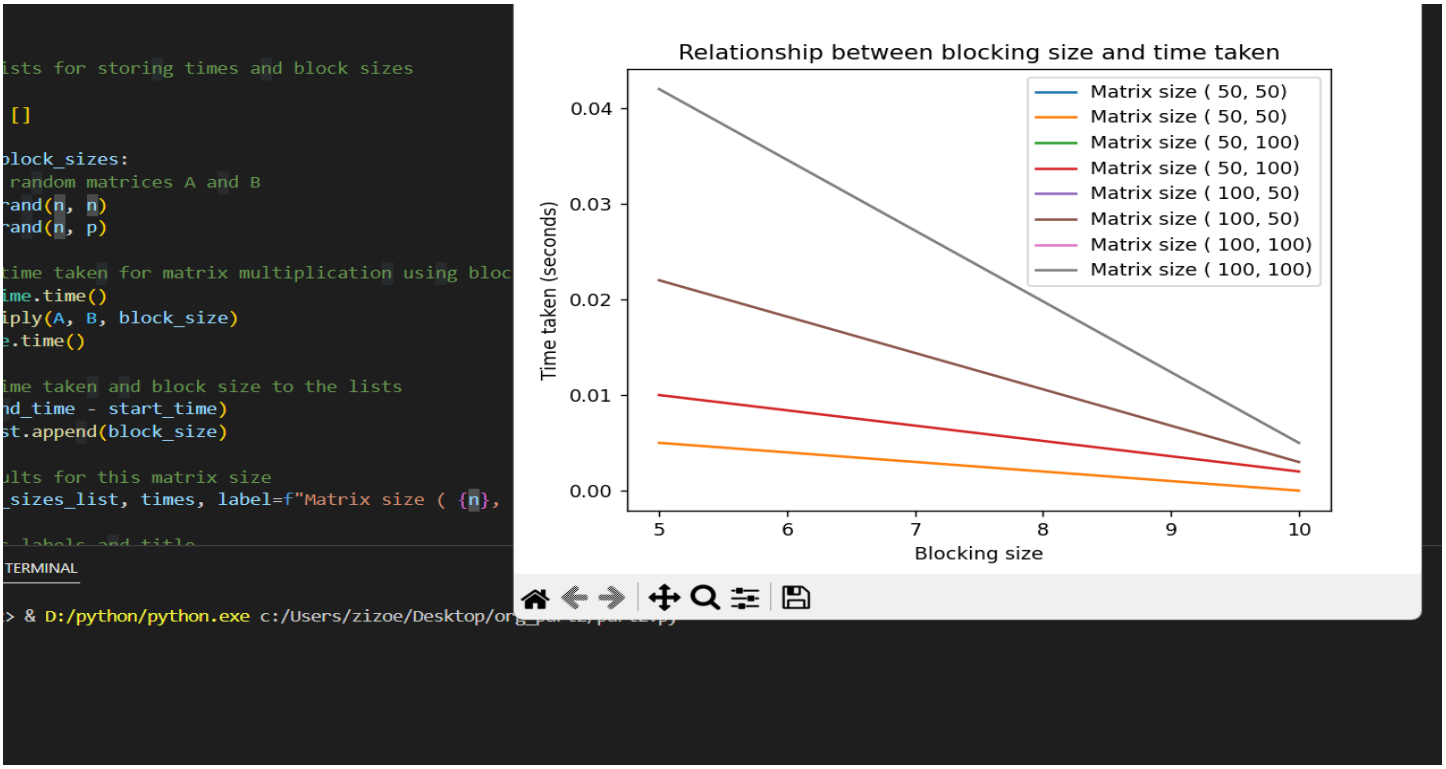

Fig 1

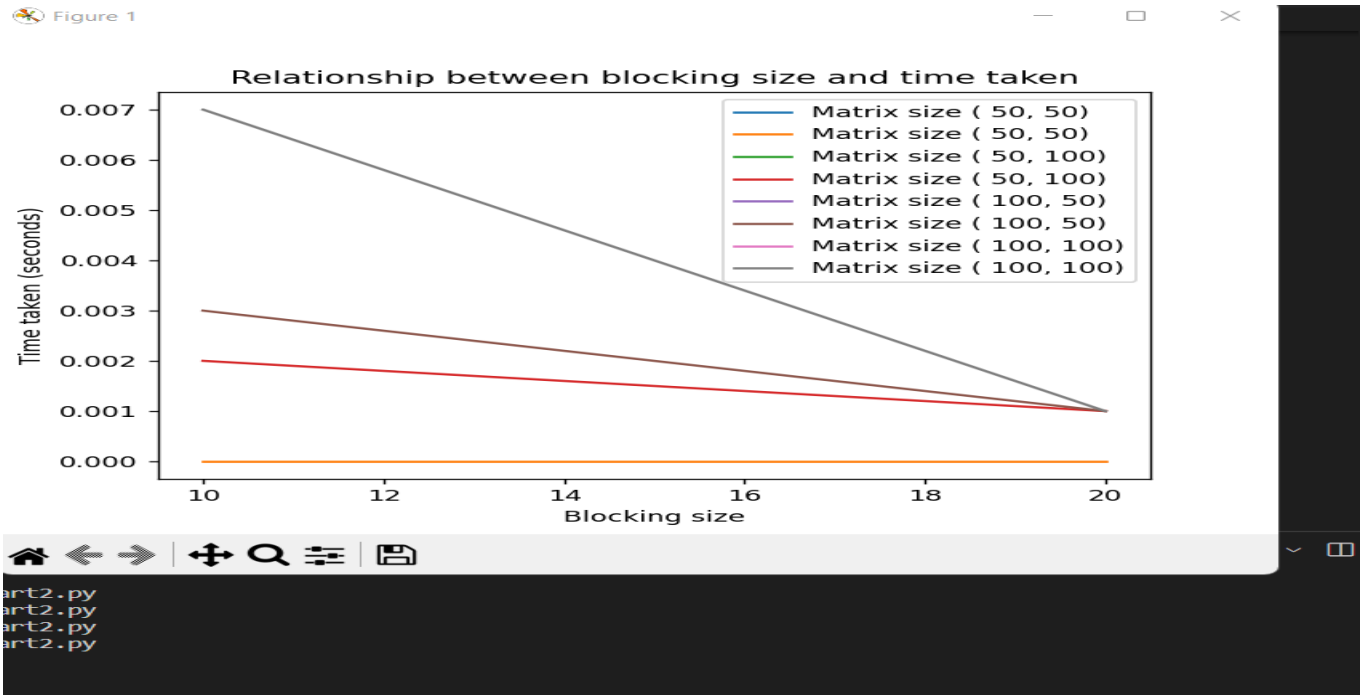

Fig 2

```
            for k in range(0, n, block_size):
                # Compute block C[i:i+block_size,
                # as a product of blocks of A and
                C_block = np.dot(A[i:i+block_size,
                                   B[k:k+block_size,
                # Add block C_block to the corresp
                C[i:i+block_size, j:j+block_size]
    return C

# Define matrix sizes and block sizes to test

matrix_sizes = [50,  1000]
block_sizes = [10, 100]

# Measure time taken for each combination of matri
for m in matrix_sizes:
    for n in matrix_sizes:
        for p in matrix_sizes:
            # Initialize lists for storing times
            times = []
            block_sizes_list = []

            for block_size in block_sizes:
                # Generate random matrices A and
                A = np.random.rand(m, n)
                B = np.random.rand(n, p)

                # Measure time taken for matrix mu
                start_time = time.time()
                C = block_multiply(A, B, block_siz
                end_time = time.time()
```

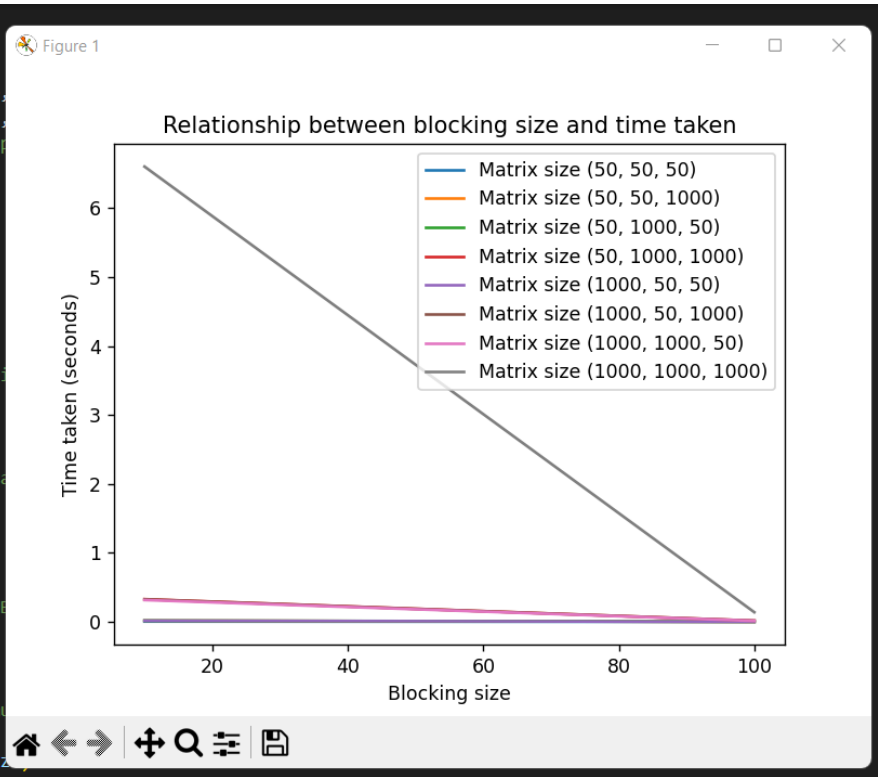

Fig 3