# Assignment #1 – Spatial Locality

Name -Surname : Onur Çetin

ID : 20200808050

Course : Parallel Computing

## Problem and My Solution Strategy

The traditional matrix multiplication method is the process of creating a new matrix by multiplying the elements of matrices in order. In this method, three nested loops are used to multiply and sum each matrix element. The first loop represents each row of the resulting matrix, the second loop represents each column and the third loop calculates the product and sum of each element. This method can increase the processing time as the matrix size increases and can lead to efficiency issues due to the memory access pattern.

The row major matrix multiplication method aims to improve performance by optimizing memory access. In this method, matrices are stored in memory in row major order and this order is used to advantage during multiplication. This is because modern processors are faster at accessing consecutive addresses when reading data from memory. Therefore, with row major ordering, access to matrix elements becomes more efficient and matrix multiplication is faster.

The alternative solution I propose aims to improve performance by optimizing cache usage in matrix multiplication. In this solution, a cache-friendly algorithm is used to access matrix elements more efficiently from memory. This improves the efficiency of cache utilization, reducing the processing time and making matrix multiplication faster.

## Time Comparison

→Row Major Solution Times

```
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 512
Matrix multiplication completed in 0.380208 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 1024
Matrix multiplication completed in 3.156426 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 2048
Matrix multiplication completed in 26.046633 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 4096
Matrix multiplication completed in 223.130027 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 8192
Matrix multiplication completed in 2019.871995 seconds.
```

```
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 16384
Matrix multiplication completed in 18432.311832 seconds.
```

```
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 32768
Matrix multiplication completed in 164.044,862104 seconds.
```

→ Traditional Solution Times

```
onuce@DESKTOP-7S6BRTU:~$ pico traditional_matrix.c
onuce@DESKTOP-7S6BRTU:~$ gcc -Wall -o traditional_matrix traditional_matrix.c
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 512
Traditional matrix multiplication completed in 0.749310 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 1024
Traditional matrix multiplication completed in 8.515207 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 2048
Traditional matrix multiplication completed in 69.860827 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 4096
Traditional matrix multiplication completed in 798.222093 seconds.
```

```
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 8194
Matrix multiplication completed in 7253.110988 seconds.
```

```
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 16384
Matrix multiplication completed in 70453.955387 seconds.
```

I didn't have the time or the computer to calculate a 32768 × 32768 matrix :(.
But considering the amount of increase, I estimate that it will take about 7 days, that is 604800 seconds, and I will add this number to the table for data visualization.

# With Flag?

The -O3 flag is a compilation optimization option provided by GCC (GNU Compiler Collection). This optimization level enables a number of optimizations to ensure a higher performance at runtime of the compiled program, and here we can see that it gives faster results compared to the old compile time.

## Row Major With Flag

```
onuce@DESKTOP-7S6BRTU:~$ gcc -O3 -o new_matrix new_matrix.c
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 512
Matrix multiplication completed in 0.080116 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 1024
Matrix multiplication completed in 0.586546 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 2048
Matrix multiplication completed in 4.430187 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 4096
Matrix multiplication completed in 40.354297 seconds.
onuce@DESKTOP-7S6BRTU:~$
```

## Row Major No Flag

```
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 512
Matrix multiplication completed in 0.380208 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 1024
Matrix multiplication completed in 3.156426 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 2048
Matrix multiplication completed in 26.046633 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./new_matrix 4096
Matrix multiplication completed in 223.130027 seconds.
```

## Traditional With Flag

```
onuce@DESKTOP-7S6BRTU:~$ gcc -O3 -o traditional_matrix traditional_matrix.c
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 512
Traditional matrix multiplication completed in 0.148168 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 1024
Traditional matrix multiplication completed in 1.752231 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 2048
Traditional matrix multiplication completed in 34.343906 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 4096
Traditional matrix multiplication completed in 738.515552 seconds.
onuce@DESKTOP-7S6BRTU:~$
```

### Traditional No Flag

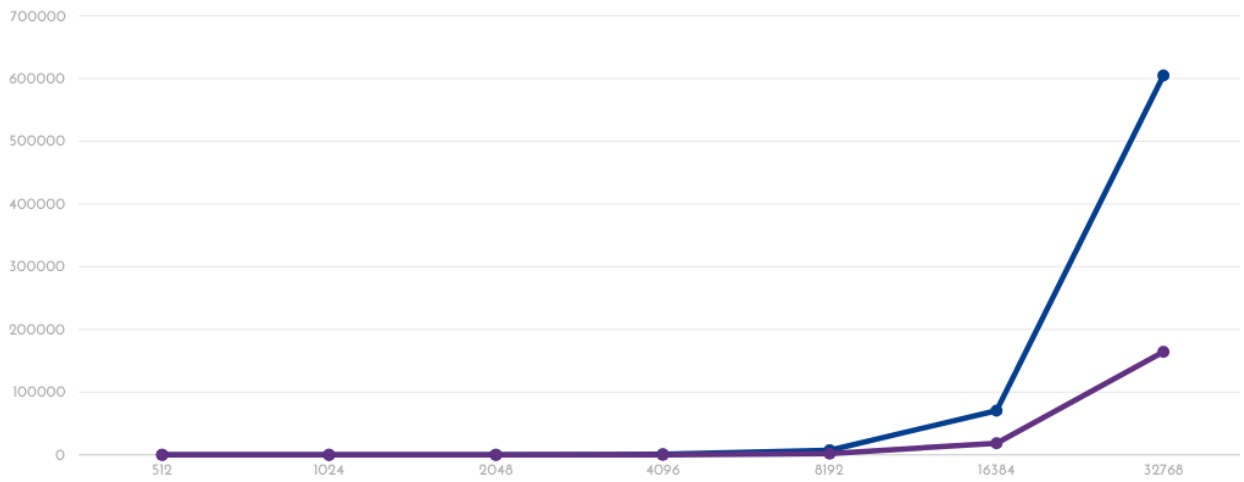
```
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 512
Traditional matrix multiplication completed in 0.749310 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 1024
Traditional matrix multiplication completed in 8.515207 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 2048
Traditional matrix multiplication completed in 69.860827 seconds.
onuce@DESKTOP-7S6BRTU:~$ ./traditional_matrix 4096
Traditional matrix multiplication completed in 798.222093 seconds.
```

| Matrix Size | TRADITIONAL | ROW MAJOR |
|---|---|---|
| 512 × 512 | 0.749310 seconds | 0.380208 seconds |
| 1024 × 1024 | 8.515207 seconds | 3.156426 seconds |
| 2048 × 2048 | 69.860827 seconds | 26.046633 seconds |
| 4096 × 4096 | 798.222093 seconds | 223.130027 seconds |
| 8192 × 8192 | 7253.110988 seconds | 2019.871995 seconds |
| 16384 × 16384 | 70453.955387 seconds | 18432.311832 seconds |
| 32768 × 32768 | 604800 seconds ~ | 164044.862104 seconds |

# Data Visualizations

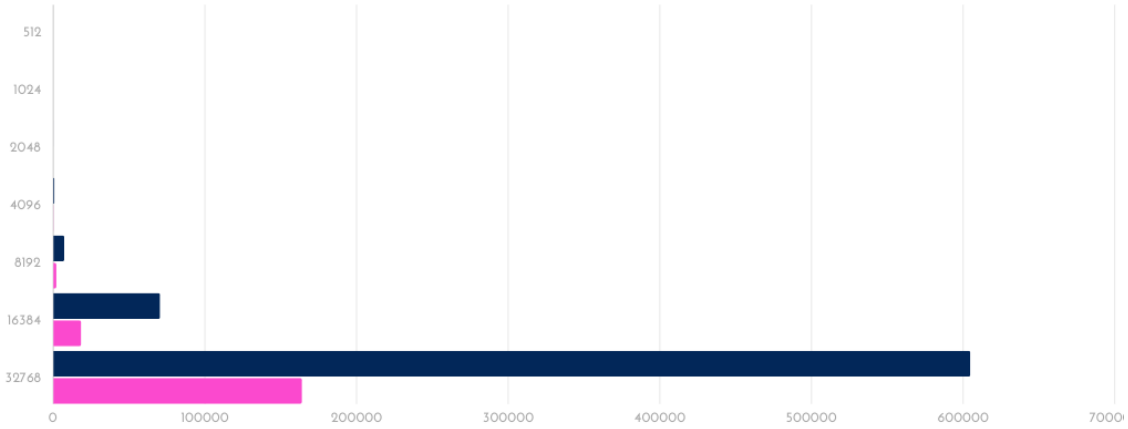
# TIME COMPARISON

The blue line represents the traditional solution and the purple one represents the row major solution. At first, there doesn't seem to be much difference, but as the matrix size increases, the fold time reaches significant differences.



## TIME COMPARISON

The blue line represents the traditional solution and the purple one represents the row major solution. At first, there doesn't seem to be much difference, but as the matrix size increases, the fold time reaches significant differences.




# Conclusion

Comparative experiments show that the cache-friendly algorithm is generally faster for large data sets. In particular, as the matrix size increases, the cache-friendly algorithm is observed to optimize data access by using the memory cache more efficiently.

In particular, row-major ordering optimization can give effective results when memory access is based on sequential ordering. In general, these optimizations in matrix multiplication not only optimize memory usage for more efficient operation, but also significantly reduce processing times. Therefore, using cache-friendly and row-major ordering optimizations in computationally intensive operations such as matrix multiplication can provide a significant performance advantage.