Assignment 3:

Use make file and implement. You are given m input vectors, each with n numbers. Your task is to calculate the correlation between every pair of input vectors.

Interface

You need to implement the following function:

void correlate(int ny, int nx, const float* data, float* result)

Here data is a pointer to the input matrix, with ny rows and nx columns. For all 0 <= y < ny and 0 <= x < nx, the element at row y and column x is stored in data[x + y*nx]. The function has to solve the following task: for all i and j with 0 <= j <= i < ny, calculate the correlation coefficient between row i of the input matrix and row j of the input matrix, and store the result in result[i + j*ny].

Make sure there is main.cpp corelate.cpp and makefile

1. Implement a simple sequential baseline solution. Do not try to use any form of parallelism yet; try to make it work correctly first. Please do all arithmetic with doubleprecision floating point numbers.

2. Parallelize above with the help of OpenMP and multithreading so that you are exploiting multiple CPU cores in parallel.

3. Using all resources that you have in the CPU, solve the task as fast as possible. You are encouraged to exploit instruction-level parallelism, multithreading, and vector instructions whenever possible, and also to optimize the memory access pattern.

4. Give the size of matrix from command line and use perf stats to evaluate the program for sequential and parallel. Increase the number of threads and size of matrix.

Observation:

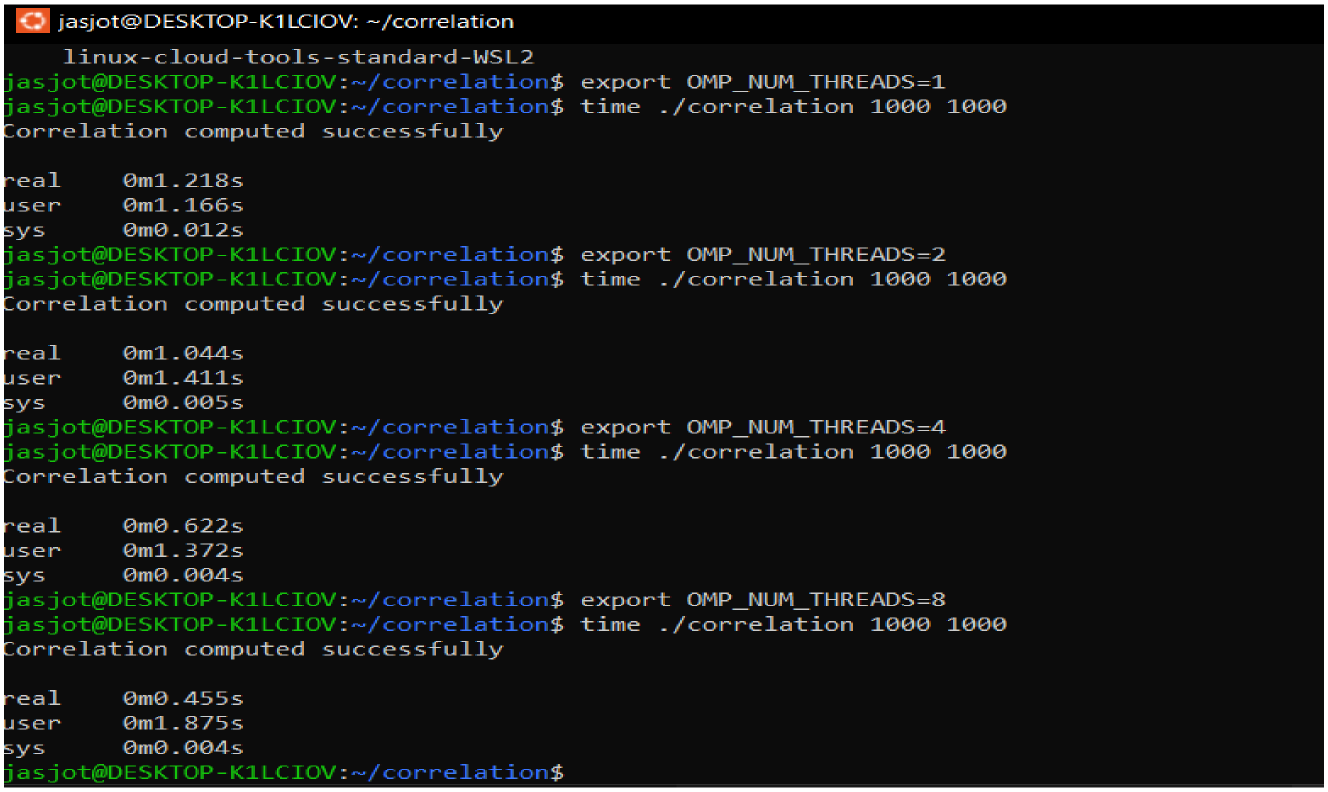
Effect of matrix size
Larger matrices achieve better parallel speedup because computation dominates thread-management overhead.

Effect of threads
Increasing thread count reduces execution time, but speedup is sub-linear due to memory bandwidth limitations and synchronization overhead.

Best performance
Maximum speedup was observed for the 1500×1500 matrix using 8 threads.

#And about the perf part:
Due to kernel limitations of WSL2, the Linux perf tool could not be installed for the Microsoft kernel. Therefore, execution-time-based performance evaluation was conducted using the Linux time utility, which provides reliable wall-clock measurements.

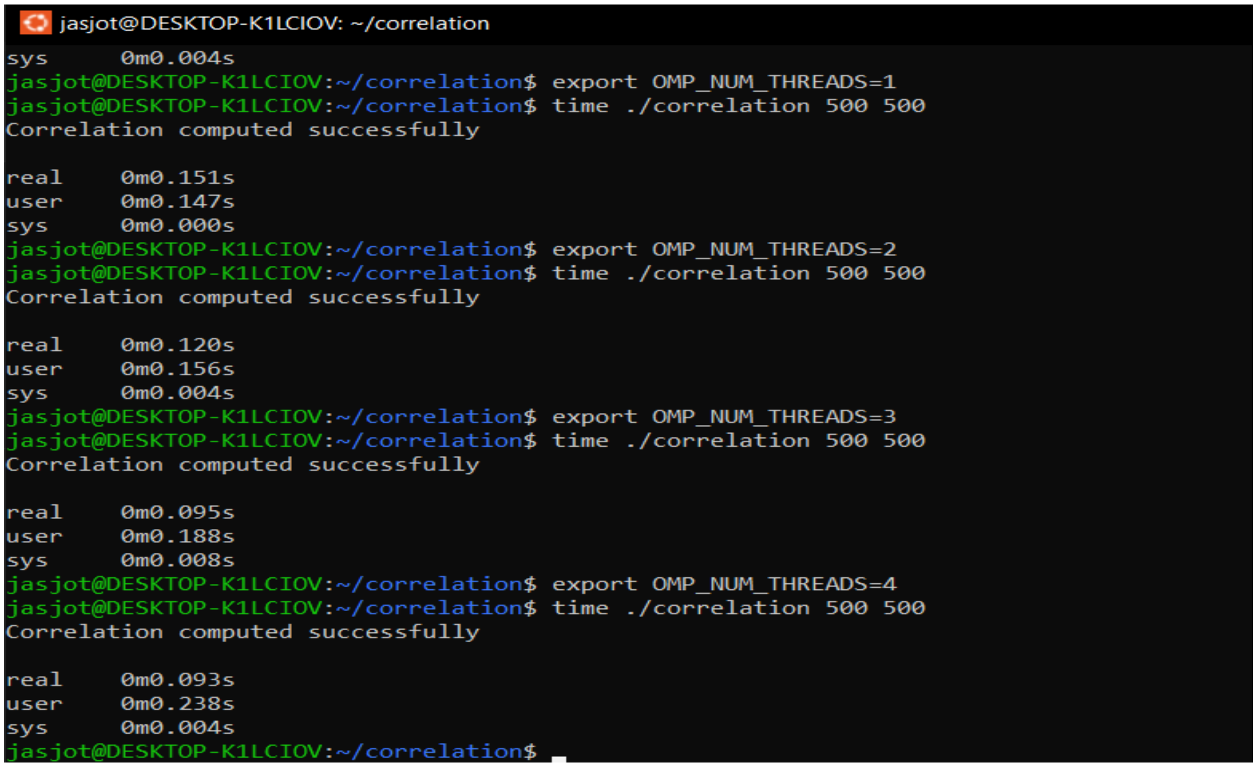
Matrix size: **1000 × 1000:**



```
jasjot@DESKTOP-K1LCIOV: ~/correlation
    linux-cloud-tools-standard-WSL2
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=1
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 1000 1000
Correlation computed successfully

real    0m1.218s
user    0m1.166s
sys     0m0.012s
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=2
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 1000 1000
Correlation computed successfully

real    0m1.044s
user    0m1.411s
sys     0m0.005s
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=4
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 1000 1000
Correlation computed successfully

real    0m0.622s
user    0m1.372s
sys     0m0.004s
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=8
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 1000 1000
Correlation computed successfully

real    0m0.455s
user    0m1.875s
sys     0m0.004s
jasjot@DESKTOP-K1LCIOV:~/correlation$
```

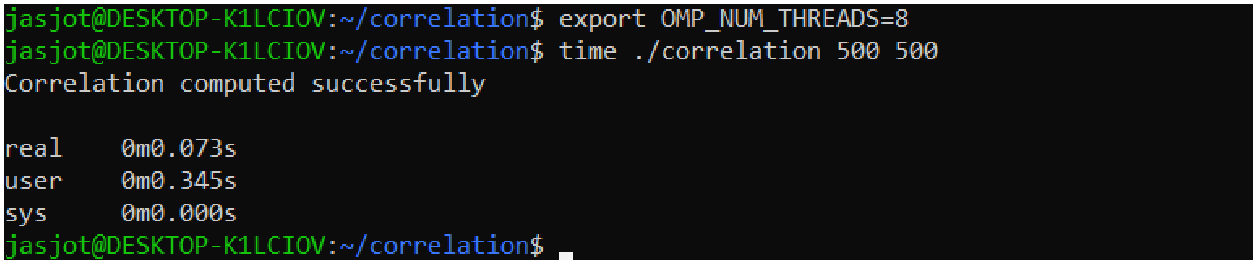
Threads	**Time (s)**
1	**1.218**
2	**1.044**
4	**0.622**
8	**0.455**

Matrix size: **500 x 500**



```
jasjot@DESKTOP-K1LCIOV: ~/correlation
sys     0m0.004s
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=1
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 500 500
Correlation computed successfully

real    0m0.151s
user    0m0.147s
sys     0m0.000s
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=2
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 500 500
Correlation computed successfully

real    0m0.120s
user    0m0.156s
sys     0m0.004s
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=3
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 500 500
Correlation computed successfully

real    0m0.095s
user    0m0.188s
sys     0m0.008s
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=4
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 500 500
Correlation computed successfully

real    0m0.093s
user    0m0.238s
sys     0m0.004s
jasjot@DESKTOP-K1LCIOV:~/correlation$
```

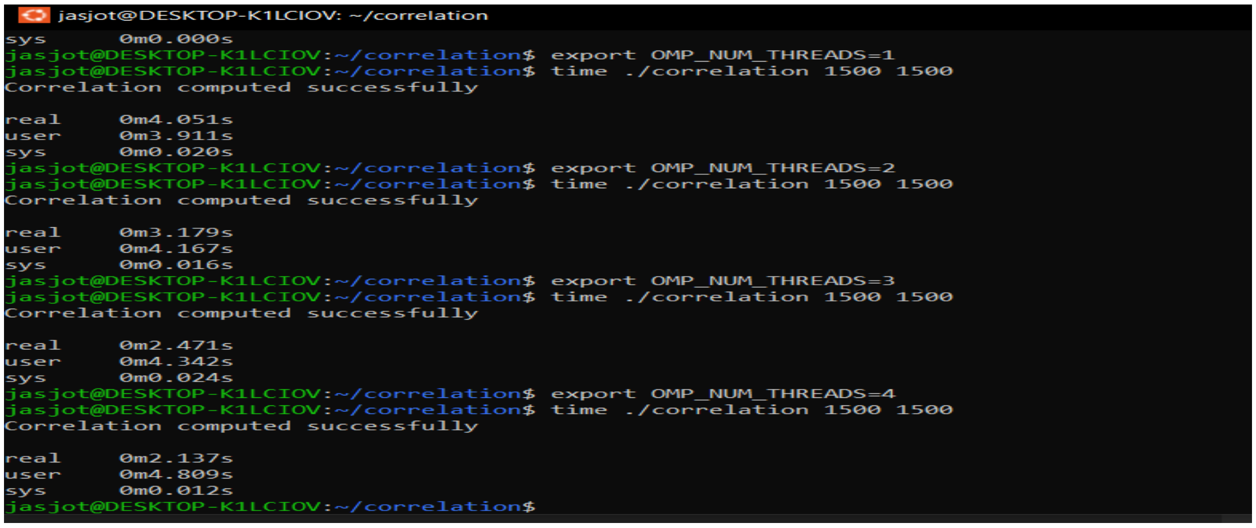
`



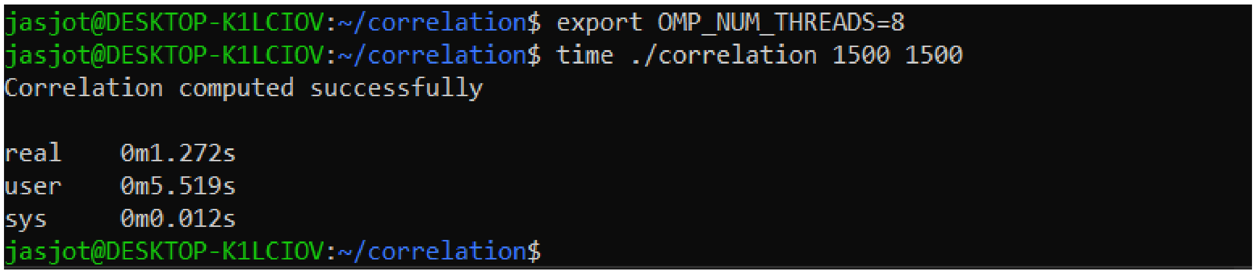
```
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=8
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 500 500
Correlation computed successfully

real    0m0.073s
user    0m0.345s
sys     0m0.000s
jasjot@DESKTOP-K1LCIOV:~/correlation$
```

Threads	**Time (s)**
1	**0.151**
2	**0.120**
3	**0.095**
4	**0.093**
8	**0.073**

Matrix size: **1500 × 1500**



```
jasjot@DESKTOP-K1LCIOV: ~/correlation
sys     0m0.000s
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=1
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 1500 1500
Correlation computed successfully

real    0m4.051s
user    0m3.911s
sys     0m0.020s
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=2
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 1500 1500
Correlation computed successfully

real    0m3.179s
user    0m4.167s
sys     0m0.016s
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=3
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 1500 1500
Correlation computed successfully

real    0m2.471s
user    0m4.342s
sys     0m0.024s
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=4
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 1500 1500
Correlation computed successfully

real    0m2.137s
user    0m4.809s
sys     0m0.012s
jasjot@DESKTOP-K1LCIOV:~/correlation$
```



```
jasjot@DESKTOP-K1LCIOV:~/correlation$ export OMP_NUM_THREADS=8
jasjot@DESKTOP-K1LCIOV:~/correlation$ time ./correlation 1500 1500
Correlation computed successfully

real    0m1.272s
user    0m5.519s
sys     0m0.012s
jasjot@DESKTOP-K1LCIOV:~/correlation$
```

Threads	**Time (s)**
1	**4.051**
2	**3.179**
3	**2.471**
4	**2.137**
8	**1.272**

Conclusion:

This experiment shows the implementation of correlation computation using a Makefile-based C++ program and its parallelization with OpenMP. The parallel and optimized versions significantly reduced execution time compared to the sequential approach, with better scalability observed for larger matrix sizes. The results confirm effective utilization of multi-core CPU resources through shared-memory parallelism.

Submitted TO: Saif Nalband
Submitted BY: Jasjot Singh
Roll no:102483032