UCS645 – Parallel & Distributed Computing

Assignment 3 – Correlation Matrix Optimization

Name: Divyam Puri **Roll No:** 102483023

Question 1 — Sequential Implementation

Aim

Implement a basic correlation computation to serve as performance baseline.

Problem Description

Each row must be normalized and compared with every other row. Work increases rapidly as dataset grows.

Methodology

Calculated mean, normalized vectors, and computed pairwise dot products without parallelization.

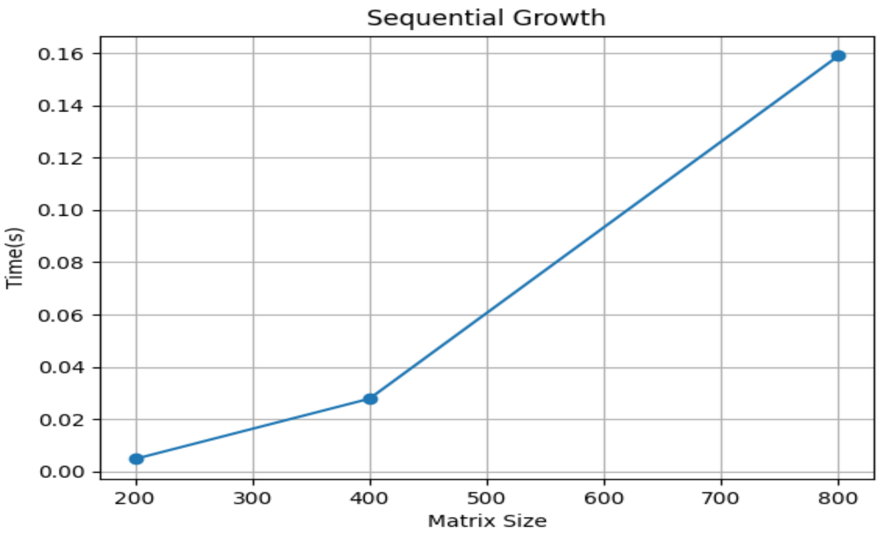
Terminal Snapshot

```
(base) divyam_puri@Divyams-MacBook-Air LAB3 % make
./corr 200 200
./corr 400 400
./corr 800 800

/opt/homebrew/opt/llvm/bin/clang++ -O2 -std=c++17 -c main.cpp
/opt/homebrew/opt/llvm/bin/clang++ -O2 -std=c++17 -c correlate.cpp
/opt/homebrew/opt/llvm/bin/clang++ -O2 -std=c++17 -o corr main.o correlate.o
Time: 0.004787 seconds
Time: 0.027839 seconds
Time: 0.158962 seconds
```

Results Table

Size	Time(s)
200x200	0.004787
400x400	0.027839
800x800	0.158962

Graph



Analysis

Execution time grows rapidly confirming computational explosion of comparisons.

Memory Usage

Requires storage for input, normalized data and result matrix only.

Conclusion

Sequential algorithm correct but inefficient for large inputs.

Question 2 — OpenMP Parallel

Aim

Evaluate effect of multithreading on correlation runtime.

Methodology

Parallelized loops using OpenMP directives.

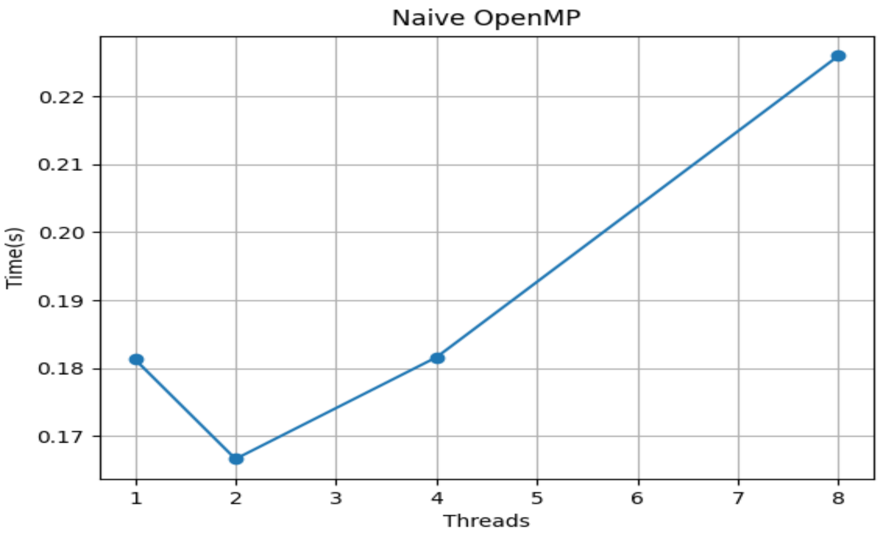
Terminal Snapshot

```
● (base) divyam_puri@Divyams-MacBook-Air LAB3 % make clean
  make

  rm -f *.o corr_omp
  /opt/homebrew/opt/llvm/bin/clang++ -O2 -std=c++17 -fopenmp -c main.cpp
  /opt/homebrew/opt/llvm/bin/clang++ -O2 -std=c++17 -fopenmp -c correlate.cpp
  /opt/homebrew/opt/llvm/bin/clang++ -O2 -std=c++17 -fopenmp -o corr_omp main.o correlate.o -fopenmp
● (base) divyam_puri@Divyams-MacBook-Air LAB3 % OMP_NUM_THREADS=1 ./corr_omp 800 800
  OMP_NUM_THREADS=2 ./corr_omp 800 800
  OMP_NUM_THREADS=4 ./corr_omp 800 800
  OMP_NUM_THREADS=8 ./corr_omp 800 800

  Time: 0.181281 seconds
  Time: 0.16662 seconds
  Time: 0.181591 seconds
  Time: 0.225954 seconds
```

Results Table

Threads	Time(s)
1	0.181281
2	0.166620
4	0.181591
8	0.225954

Graph



Analysis

Parallelization slower due to memory bandwidth contention and cache thrashing.

Conclusion

Not all algorithms benefit from naive multithreading.

Question 3 — Optimized Blocking

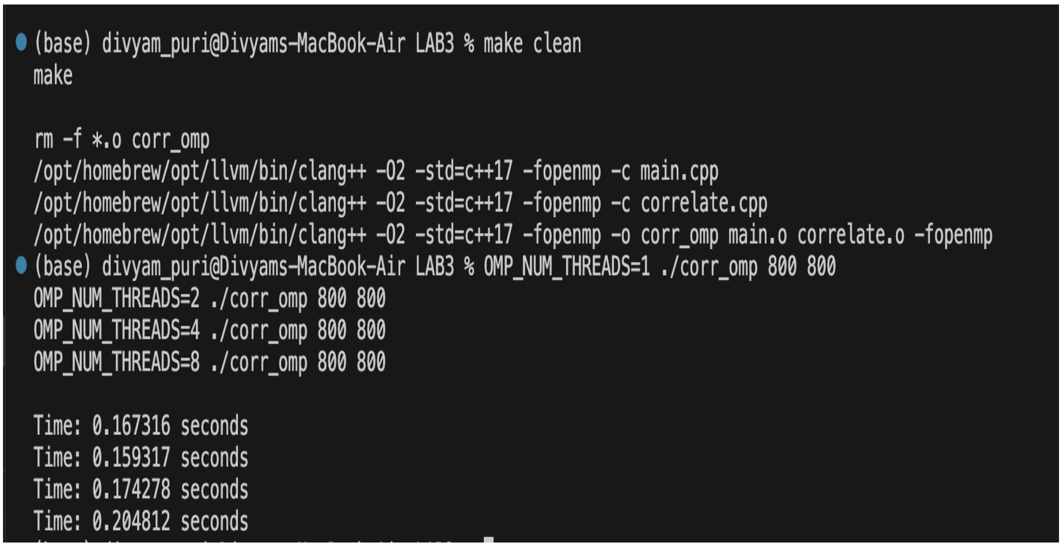
Aim

Improve cache locality using blocking technique.

Methodology

Divided matrix into blocks so reused data remains in cache.

Terminal Snapshot

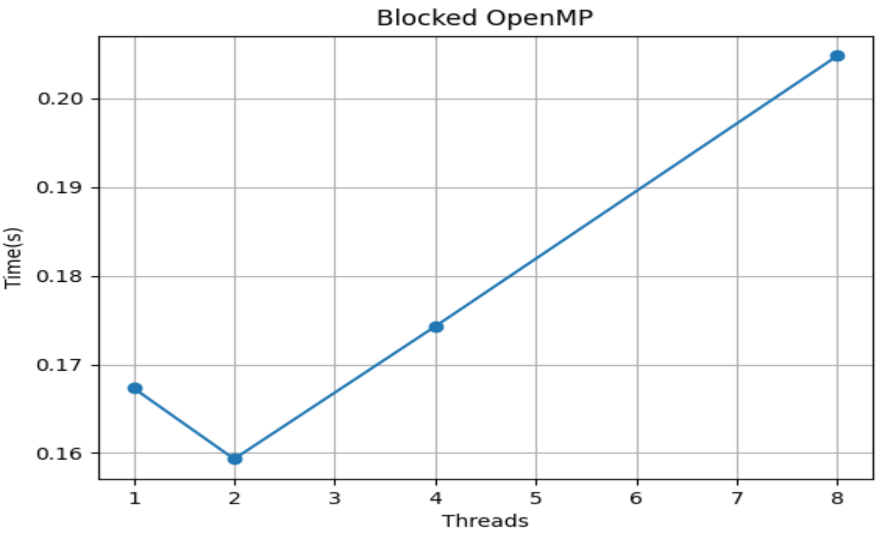


```
(base) divyam_puri@Divyams-MacBook-Air LAB3 % make clean
make

rm -f *.o corr_omp
/opt/homebrew/opt/llvm/bin/clang++ -O2 -std=c++17 -fopenmp -c main.cpp
/opt/homebrew/opt/llvm/bin/clang++ -O2 -std=c++17 -fopenmp -c correlate.cpp
/opt/homebrew/opt/llvm/bin/clang++ -O2 -std=c++17 -fopenmp -o corr_omp main.o correlate.o -fopenmp
(base) divyam_puri@Divyams-MacBook-Air LAB3 % OMP_NUM_THREADS=1 ./corr_omp 800 800
OMP_NUM_THREADS=2 ./corr_omp 800 800
OMP_NUM_THREADS=4 ./corr_omp 800 800
OMP_NUM_THREADS=8 ./corr_omp 800 800

Time: 0.167316 seconds
Time: 0.159317 seconds
Time: 0.174278 seconds
Time: 0.204812 seconds
```

Results Table

Threads	Time(s)
1	0.167316
2	0.159317
4	0.174278
8	0.204812

Graph



Analysis

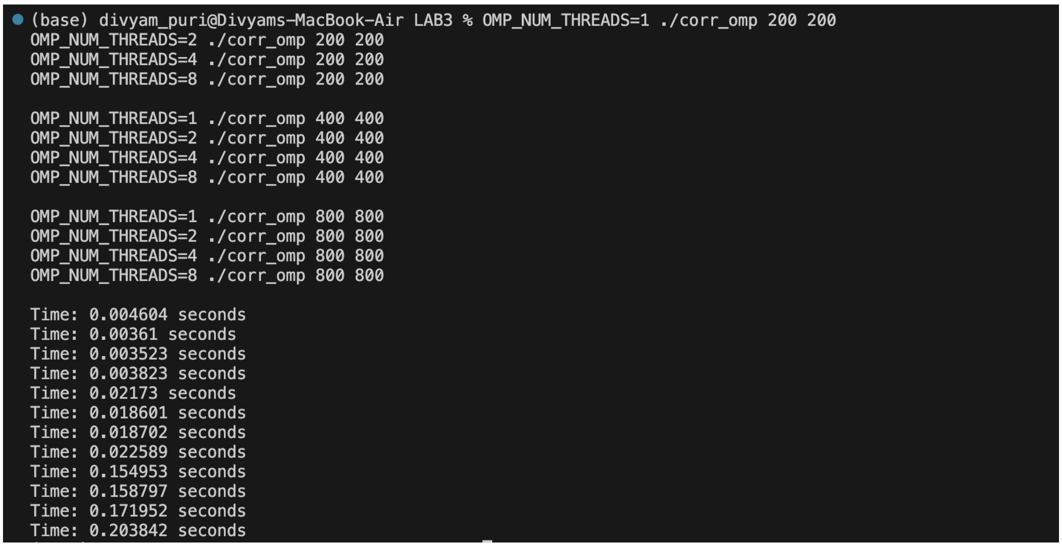
Blocking stabilizes performance by reducing RAM fetches.

Conclusion

Cache-friendly design improves efficiency but still memory-limited.

Question 4 — Performance Study

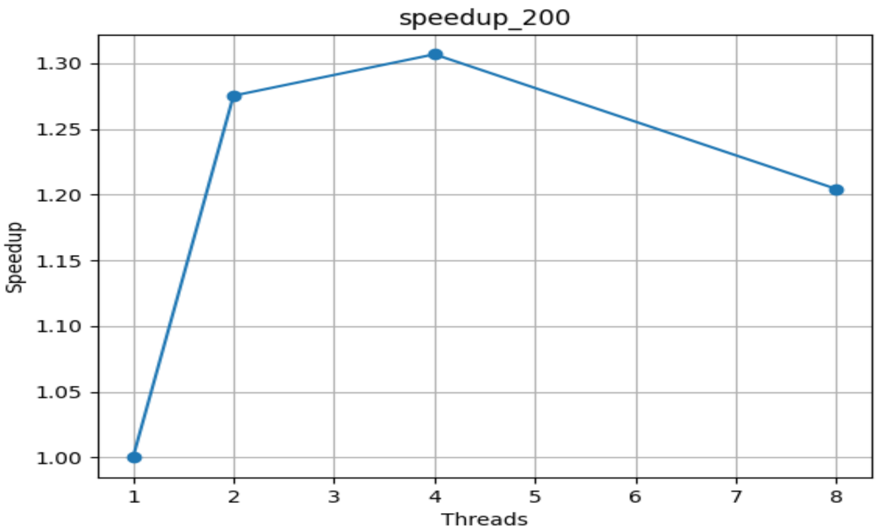
Terminal Snapshot



```
(base) divyam_puri@Divyams-MacBook-Air LAB3 % OMP_NUM_THREADS=1 ./corr_omp 200 200
OMP_NUM_THREADS=2 ./corr_omp 200 200
OMP_NUM_THREADS=4 ./corr_omp 200 200
OMP_NUM_THREADS=8 ./corr_omp 200 200

OMP_NUM_THREADS=1 ./corr_omp 400 400
OMP_NUM_THREADS=2 ./corr_omp 400 400
OMP_NUM_THREADS=4 ./corr_omp 400 400
OMP_NUM_THREADS=8 ./corr_omp 400 400

OMP_NUM_THREADS=1 ./corr_omp 800 800
OMP_NUM_THREADS=2 ./corr_omp 800 800
OMP_NUM_THREADS=4 ./corr_omp 800 800
OMP_NUM_THREADS=8 ./corr_omp 800 800

Time: 0.004604 seconds
Time: 0.00361 seconds
Time: 0.003523 seconds
Time: 0.003823 seconds
Time: 0.02173 seconds
Time: 0.018601 seconds
Time: 0.018702 seconds
Time: 0.022589 seconds
Time: 0.154953 seconds
Time: 0.158797 seconds
Time: 0.171952 seconds
Time: 0.203842 seconds
```

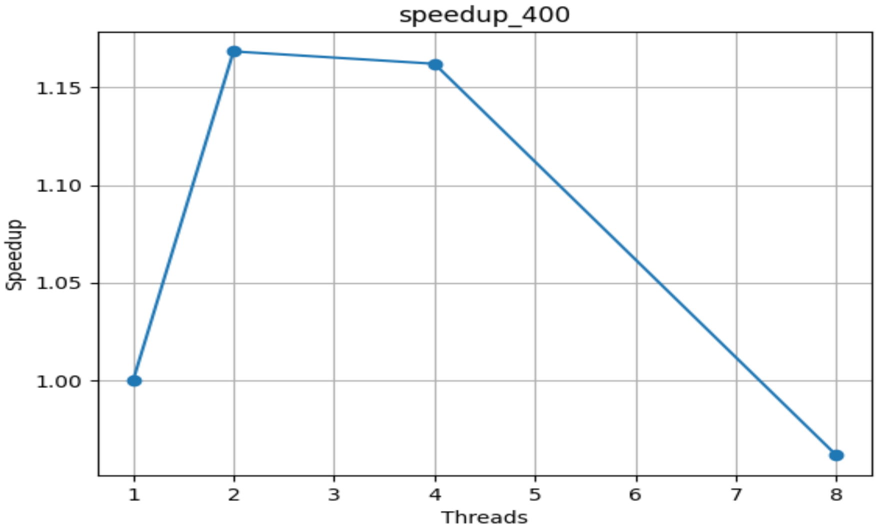
200x200

Threads	Time(s)
1	0.004604
2	0.003610
4	0.003523
8	0.003823



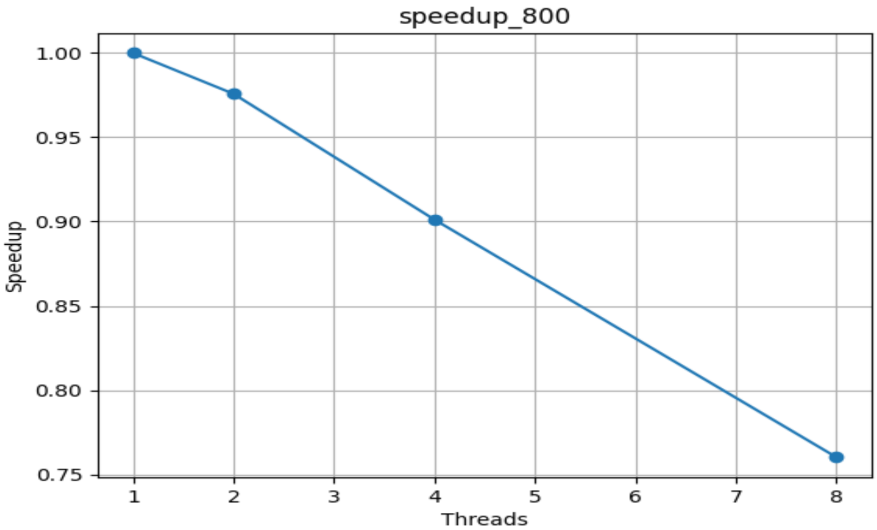
400x400

Threads	Time(s)
1	0.021730
2	0.018601
4	0.018702
8	0.022589



800x800

Threads	Time(s)
1	0.154953
2	0.158797
4	0.171952
8	0.203842



Conclusion

Optimal threads around 2–4; larger counts cause contention due to memory bandwidth limits.