Floyd-Warshall Algorithm for All Pair Shortest Path

OVERVIEW:

- Problem Statement
- Sequential Algorithm
- Rectangular Algorithm
- Blocked Floyd Warshall
- Implementation and Parallel Algorithm
- Cache Optimized Algorithm
- Results
- References

The Floyd Warshall Algorithm

- This algorithm solves the shortest path problem for a directed and weighted graph.

- It tries to find the minimum distance between any pair of vertices in a graph.
  In the adjacency matrix created dist(i,j) represents the distance from source (ith node) to the destination(jth node).

- Since the distance from a vertex to itself is going to be 0 , hence all the diagonals are set to 0 in the matrix.

- In this we consider every vertex as an intermediate vertex 'k' and find if the distance between i,j through k is smaller than the existing distance.
  i.e. dist(i,j) = min(dist(i,j) , dist(i,k) + dist(k,j)) .

SEQUENTIAL ALGORITHM

```
Input: n = number of vertices
       A = adjacency matrix
Output: Transformed A that contains all pair shortest paths

For each vertex k from 1 to n:
  For each pair of vertices i and j from 1 to n:
    If (distance[i][k] != INF && distance[k][j] != INF) {
             If dist[i][j]>dist[i][k]+dist[k][j],then update dist[i][j] to dist[i][k]+dist[k][j]
  end for
end for
```

- Clearly it can be observed that the time complexity of this algorithm is $O(n^3)$.

Rectangular Implementation –

- The Rectangular algorithm efficiently solves the same problem as Floyd-Warshall by skipping iterations where the distance between vertices is infinite, reducing unnecessary computations for faster execution.

- Clearly the time complexity is $O(n^3)$.

Input: num_vertices, dist (adjacency matrix of size num_vertices x num_vertices)
Output: Updated adjacency matrix containing all-pair shortest paths

```
 procedure solve(dist, num_vertices)
   for j from 0 to num_vertices - 1
      for i from 0 to num_vertices - 1
         r ← dist[i][j]
         if r equals INF
            continue
         for k from 0 to num_vertices - 1
            t ← dist[j][k]
            if t equals INF or i equals j or k equals j or i equals k
               continue
            s ← dist[i][k]
            if s is greater than r + t
               s ← r + t
```

PARALLEL BLOCKED ALGORITHM

- Divide the distance matrix into smaller blocked matrices to improve cache locality, accelerating memory operations.

- The original n×n matrix A is divided into smaller b×b sub-matrices (Aij, Aik, Akj, Akk), where each sub-matrix has an optimal blocking factor b. Determining this factor is crucial for cache efficiency, as it ensures that the adjacency matrix fits within cache constraints.

- Our computation has been performed in the OpenMP environment.

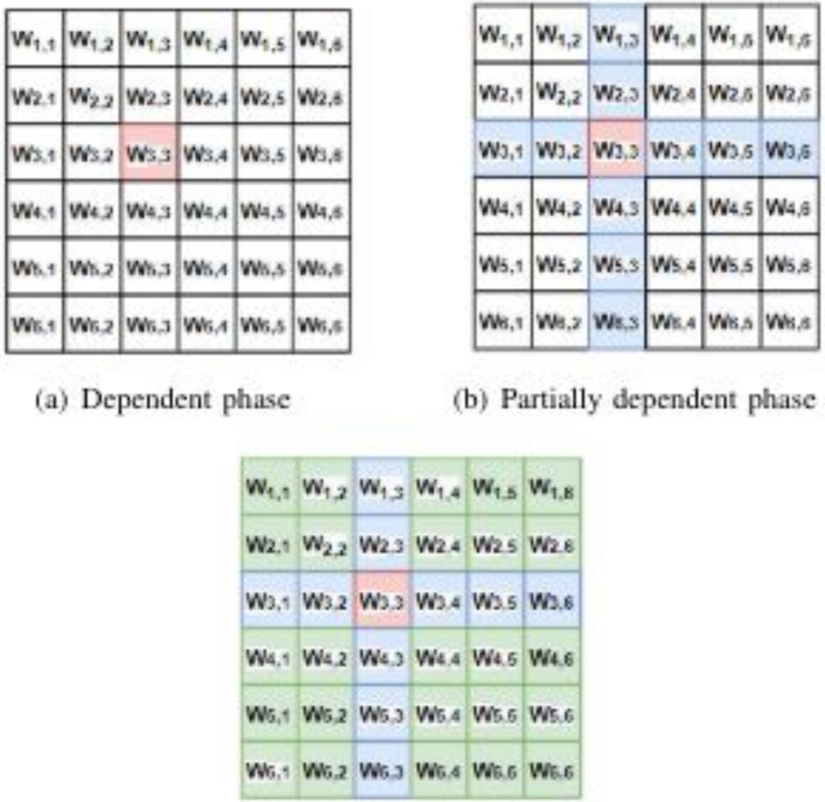
This blocked parallel implementation, proposed must be processed in
three separate phases:

• Dependent phase - processing the kth diagonal block.

• Partially dependent phase - processing the kth row and the kth column of blocks.

• Independent phase - processing the remaining blocks.

Image source: [3] Towards performance improvement of a parallel Floyd-Warshall algorithm using OpenMP and Intel TBB



(a) Dependent phase

(b) Partially dependent phase

(c) Independent phase

ARCHITECTURE

The following implementations of the FW algorithm were executed in C++ using OpenMP environments.

Processor Name	Intel Xeon E5-2640 v4
Number of Cores	20
Number of Threads	40
Base Core Clocks[GHz]	2.40
L1 Cache[KB]	32
L1 Cache Line(Byte)	64

Optimal Blocking Factor

According to Pozder, Nudžejma & Ćorović, Dalila & Herenda, Esma & Divjan, Belmin. (2021)[3],
The optimal blocking factor b must be the largest integer such that $2*b^2*4 \leq C$ and b is a dividable of S/4, where C presents L1 cache capacity and S is L1 cache line.

Our system has C = 32 KB and S = 64 bytes
On solving, we obtain that b ≤ 63.24 and b should be divisible by 16.
Taking these two factors into consideration we consider b = 16

THE ALGORITHM

```
floyd(C,A,B) {
C, A and B are b×b matrices
for k from 0 to b do
  for j from 0 to b do
    for i from 0 to b do
      C[i][j] = min(C[i][j], A[i][k] + B[k][j])
    end for
  end for
end for
}
blocked_floyd_warshall(W,n) {
//split W into "blocks" with blocksize "b"
//For simplicity if b divides n, b blocks are created
for k from 0 to B do
    //Dependent phase
    floyd(Wkk, Wkk, Wkk);
    //Partially Dependent Phase
    parallel for j from 0 and j ≠ k to B
    floyd(Wkj , Wkk, Wkj );
    end parallel for
    parallel for i from 0 and i ≠ k to B
    floyd(Wik, Wik, Wkk);
    for j from 0 and j ≠ k to B do
    //Independent Phase
    floyd(Wij , Wik, Wkj );
    end for
    end parallel for
end for
}
```

REFERENCE: [3]
Towards performance improvement of a parallel Floyd-Warshall algorithm using OpenMP and Intel TBB

- Here the 2D distance matrix is converted into a 1D array because in C++ memory is allocated in contiguous blocks in row major fashion also contributing to the creation of a cache efficient algorithm as elements are accessed in a nested loop fashion and parallelization adds on to the overall reduction in time.

CACHE OPTIMIZED ALGORITHM

- This algorithm is similar to the blocked implementation but it has been done without the parallelization to observe how efficient the Floyd Warshall algorithm can be if it is cache optimized.

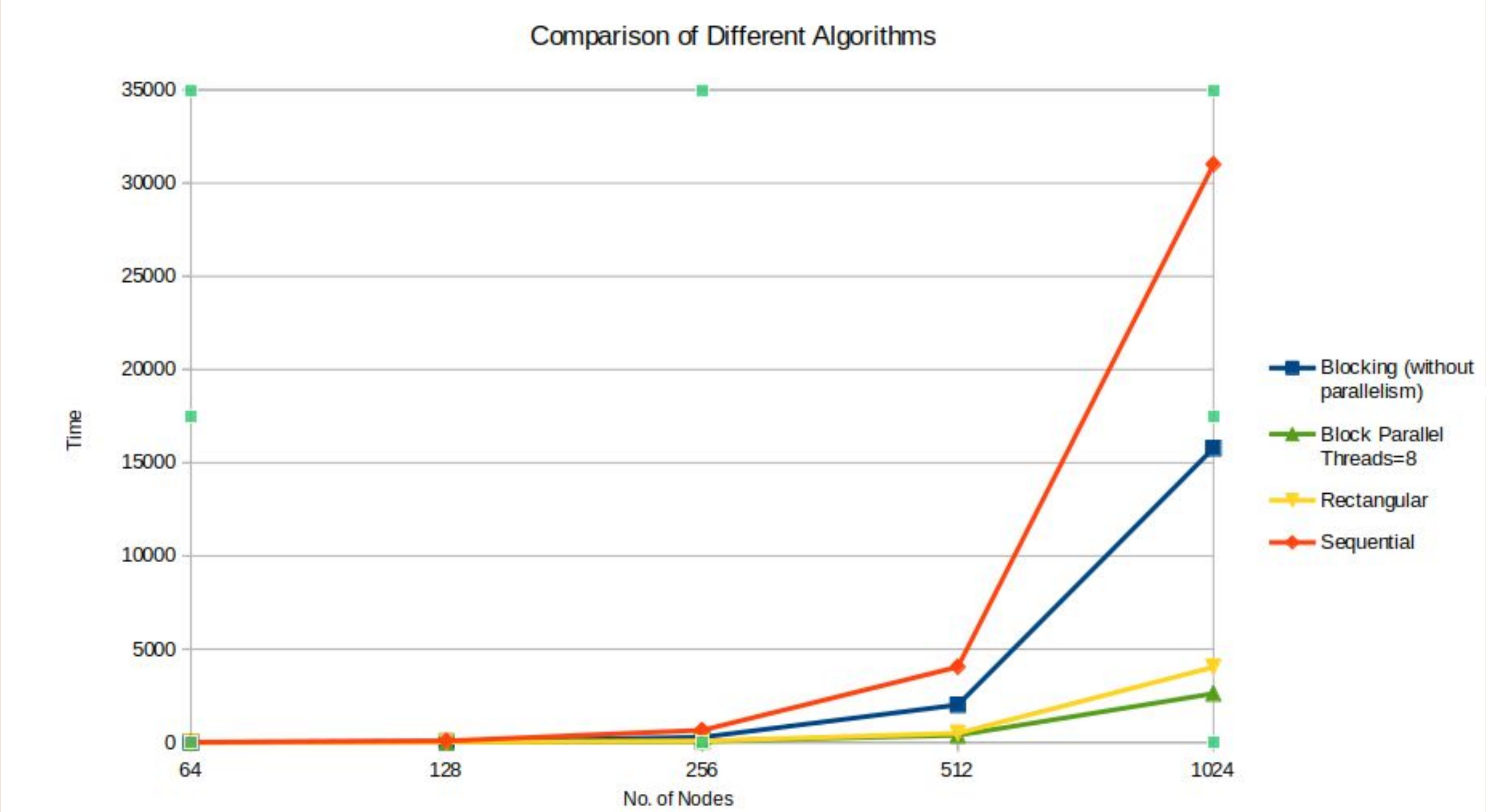
- In C++, converting a 2D distance matrix into a 1D array is done to leverage memory allocation in contiguous blocks, following the row major fashion. This enhances cache efficiency, especially when elements are accessed in nested loop fashion.

The codes for all the algorithms can be viewed here:
https://github.com/b22cs005/Optimization-of-Floyd-Warshall-Algorithm

RESULTS

COMPARING ALL THE IMPLEMENTED ALGORITHMS:

Nodes	Sequential	Rectangular	Cache Optimized	Blocked Parallel
32	1	0	1	1
64	10	1	6	3
128	79	10	45	9
256	654	77	277	56
512	4049	491	2015	381
1024	31000	4040	15780	2620
2048	248196	33247	127250	19548

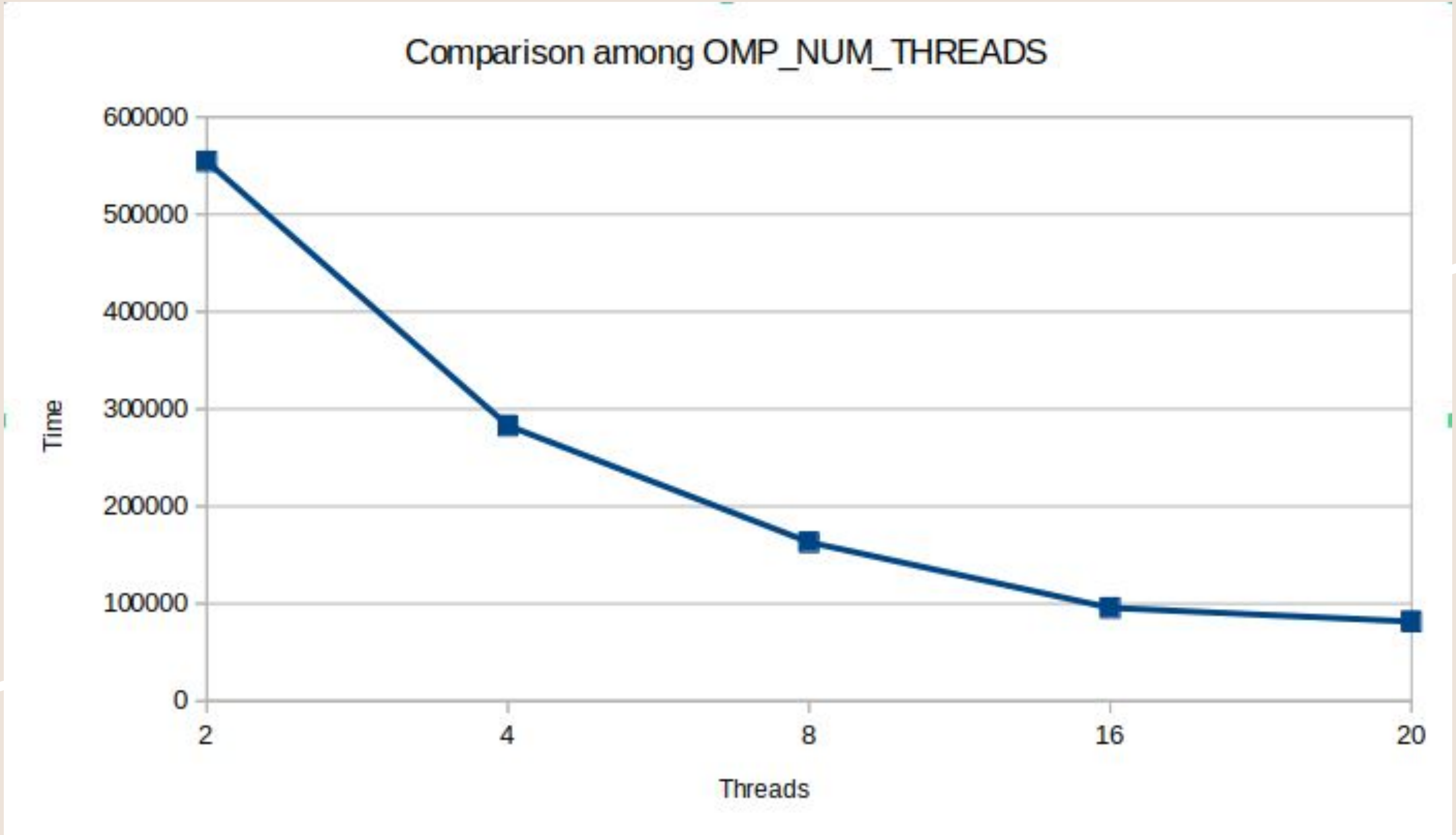


Clearly the Blocked Parallel version takes the lowest time as observed from the graph causing it to be the most optimized algorithm.

Comparing with fixed matrix size(4096x4096) and different number of nodes for the blocked parallel algorithm

Number of Threads	Time (ms)
2	554371
4	282853
8	163085
16	95673
20	81582

Comparing with different number of threads - (fixed matrix size = 4096X4096)



REFERENCES

[1] Asghar Aini, Amir Salehipour, ”Speeding up the Floyd–Warshall algorithm for the cycled shortest path problem”, Applied Mathematics Letters,Volume 25, Issue 1,2012,Pages 1-5.
Link:Speeding up the Floyd–Warshall algorithm for the cycled shortest path problem - ScienceDirect

[2] Jared Moore and Josh Kalapos,”Floyd-Warshall vs Johnson:Solving All Pairs Shortest Paths in Parallel”
Link: https://moorejs.github.io/APSP-in-parallel/

[3] Pozder, Nudžejma & Ćorović, Dalila & Herenda, Esma & Divjan, Belmin. (2021). “Towards performance improvement of a parallel Floyd-Warshall algorithm using OpenMP and Intel TBB.”
Link: Towards performance improvement of a parallel Floyd-Warshall algorithm using OpenMP and Intel TBB