Parallel Implementation of Flyod Warshall Algorithm

Submitted By: -

AYAYDEEP
170001003

RAHUL ANAND YADAV
170001038

There are many algorithms to find all pair shortest path. Most popular and efficient of them is Floyd Warshall algorithm. In this report the parallel version of the algorithm is presented considering the one-dimensional row wise decomposition of the adjacency matrix. From the results it is observed that parallel algorithm is considerably effective for large graph sizes.

1. Introduction

Finding the shortest path between two objects or all objects in a graph is a common task in solving many day to day and scientific problems. The algorithms for finding shortest path find their application in many fields such as social networks, bioinformatics, aviation, routing protocols, Google maps etc. Shortest path algorithms can be classified into two types: single source shortest paths and all pair shortest paths. There are many different algorithms for finding the all pair shortest paths. Some of them are Floyd-Warshall algorithm and Johnson's algorithm.

All pair shortest path algorithm which is also known as Floyd-Warshall algorithm was developed in 1962 by Robert Floyd [1]. This algorithm follows the methodology of the dynamic programming. The algorithm is used for graph analysis and finds the shortest paths (lengths) between all pair of vertices or nodes in a graph. The graph is a weighted directed graph with negative or positive edges. The algorithm is limited to only returning the shortest path lengths and does not return the actual shortest paths with names of nodes.

Sequential algorithm

```
for k = 0 to N-1
        for i = 0 to N-1
                for j = 0 to N-1

                        Iij (k+1) = min (Iij(k), Iik(k) + Ikj(k))
                endfor
        endfor
endfor
```

Sequential pseudo-code of this algorithm is given above requires N^3 comparisons. For each value of k that is the count of inter mediatory nodes between node i and j the algorithm computes the distances between node i and j and for all k nodes between them and compares it with the distance between i and j with no inter mediatory nodes between them. It then considers the minimum distance among the two distances calculated above. This distance is the shortest distance between node i and j. The time complexity of the above algorithm is $\Theta(N^3)$. The space complexity of the algorithm is $\Theta(N^2)$. This algorithm requires

the adjacency matrix as the input. Algorithm also incrementally improves an estimate on the shortest path between two nodes, until the path length is minimum.

2. Problem and Solution

Parallelizing all pair shortest path algorithm is a challenging task considering the fact that the problem is dynamic in nature. The algorithm can be parallelized by considering the one-dimensional row wise decomposition of the intermediate matrix I. This algorithm will allow the use of at most N processors. Each task will execute the parallel pseudo code stated below.

0	1	999	1	5
9	0	3	2	999
999	999	0	4	999
999	999	2	0	3
3	999	999	999	0

Parallel algorithm

#pragma omp parallel shared(distance_matrix)

```
for (middle = 0; middle < N; middle++)
{
  int * dm=distance_matrix[middle];
  #pragma omp parallel for private(src, dst) schedule(dynamic)
  for (src = 0; src < N; src++)
  {
    int * ds=distance_matrix[src];
    for (dst = 0; dst < N; dst++)
    {
      ds[dst]=min(ds[dst],ds[middle]+dm[dst]);
    }

  }

}
```

3. Implementation

For implementation we have both OpenMP. Sequential and parallel programs are written in c. For creating the adjacency matrix, we have used the random number generator. In terms implementation we have used 28 core CPU provided by lab

4. Results

The graphical results below give insights onto the execution time versus number of threads for different data sizes, speedup versus number of processes for fixed load size.

Parallel Implementation

No. of processers used = N^2

Time Complexity = O(N)

Space Complexity = O(N^2)

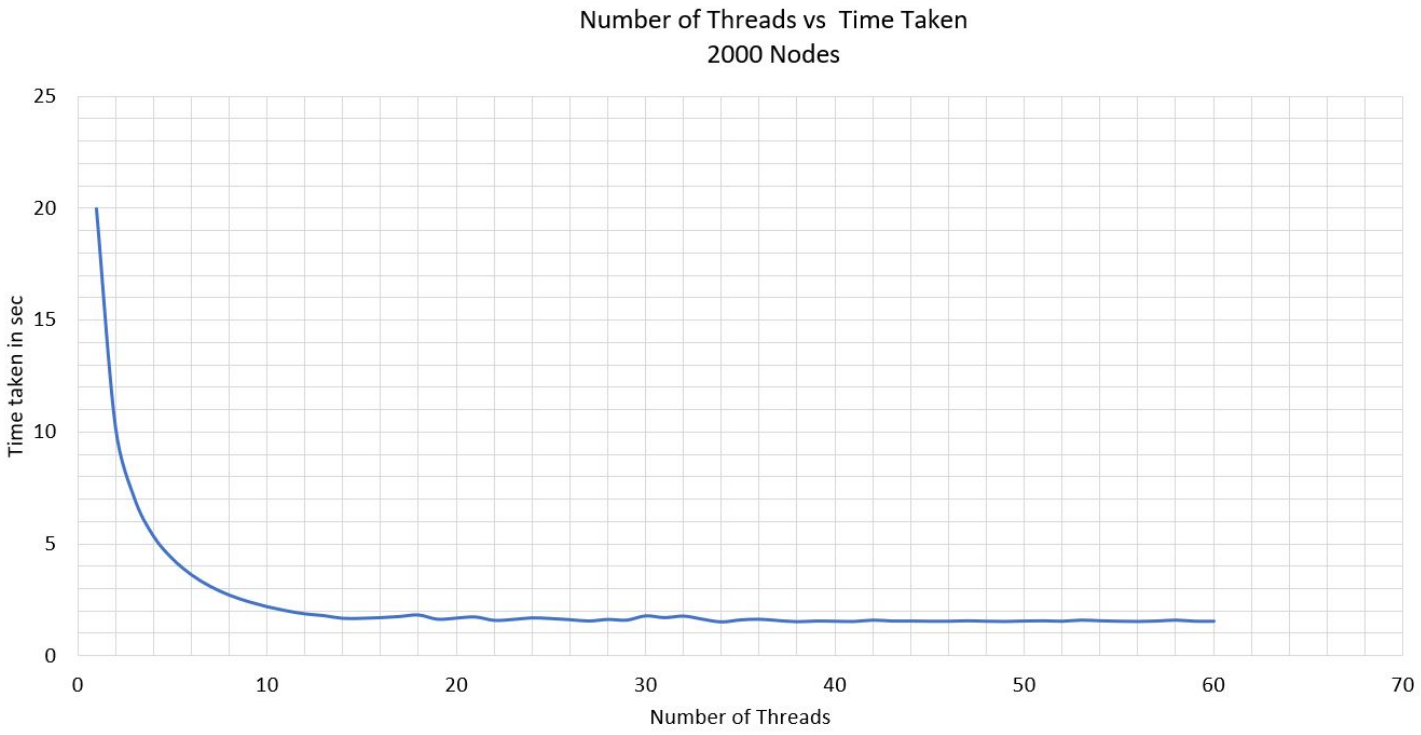
Work = N^2 * O(N) = O(N^3)

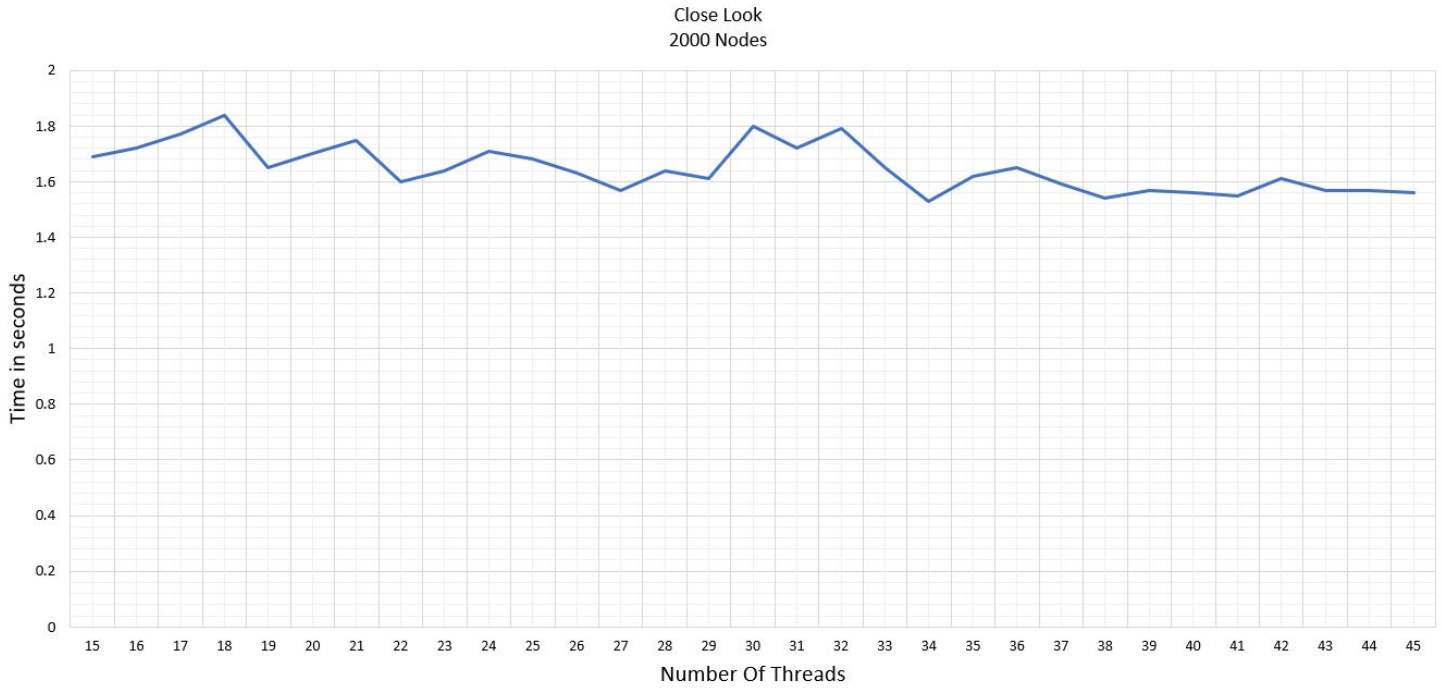
Cost = N^2 * O(N) = O(N^3)

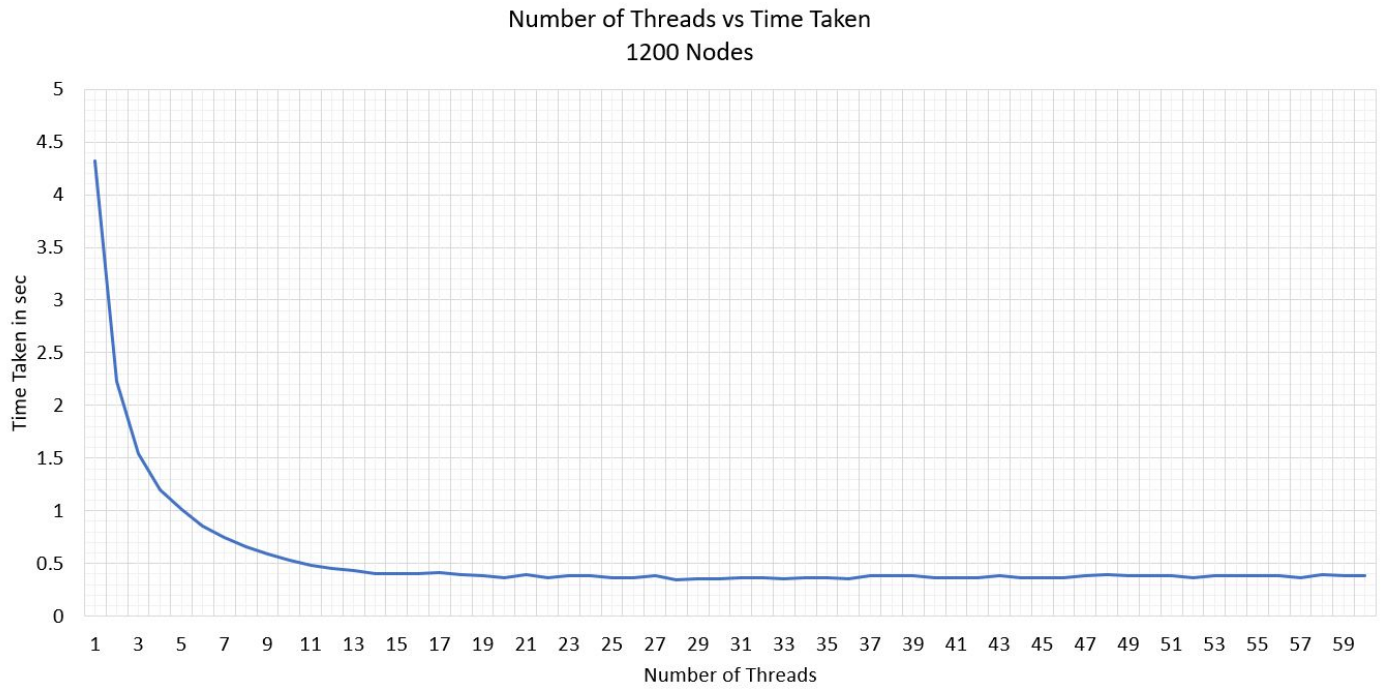
Sequential Algorithm

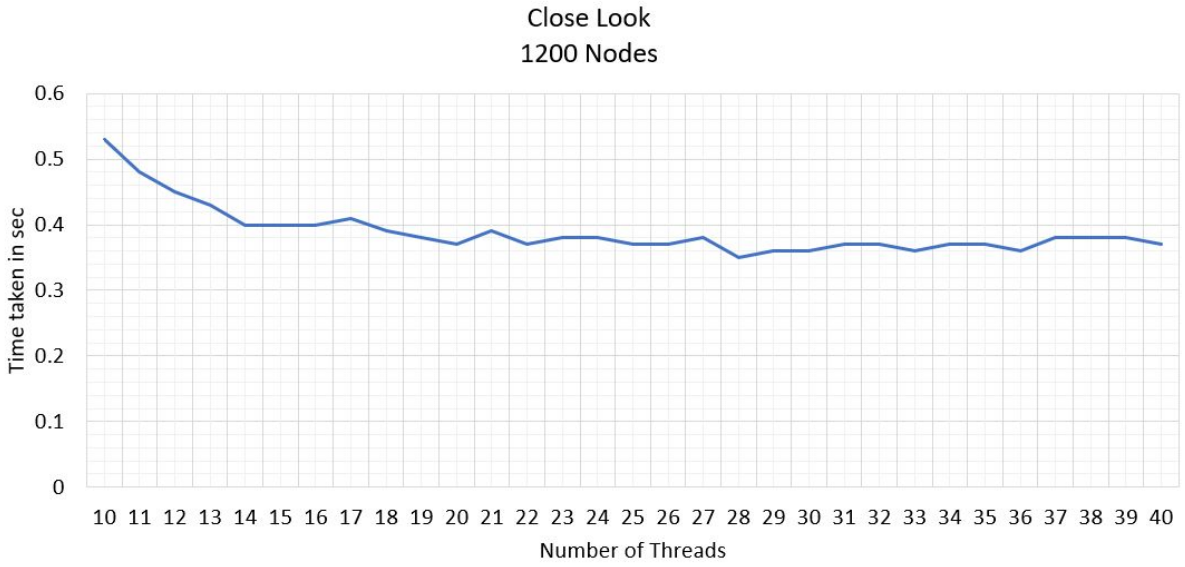
Time Complexity = O(N^3)

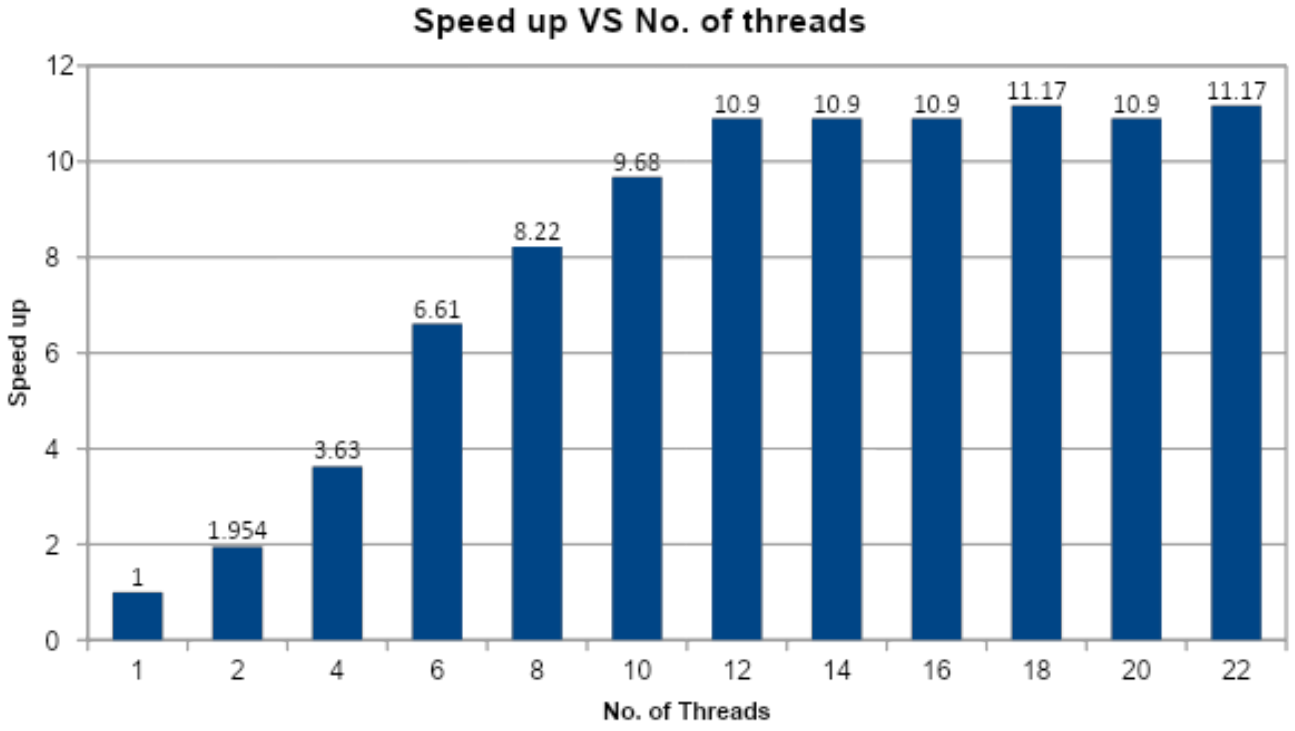
Space Complexity = O(N^2)




Close Look
2000 Nodes
Time in seconds
Number Of Threads


Number of Threads vs Time Taken
1200 Nodes
Time Taken in sec
Number of Threads





5. Execution and Analysis

As can be seen from the graph, that there is a sharp decline in execution time for **node size** of **1200** for increasing number of processes. The execution time almost reaches a constant value of **0.40 sec** for increasing number of threads after **14 threads** .
And for **node size** of **2000** the execution time almost reaches a constant value of **1.60 sec** for increasing number of threads after **25 threads**.

From graph it is observed that the **maximum speedup** is **11.47** when there are **18 threads** for fixed **1200** number of **nodes**. Speed up increases in beginning but becomes stable at the end.

5. Conclusion and Applications

From the observations I believe that the best use of parallel implementation of all pair shortest path algorithm is for large datasets or graphs. The speedups can only be observed with large adjacency matrices.

It can be used on real life dataset such as that of social network that are publically available.

6. References

https://gkaracha.github.io/papers/floyd-warshall.pdf