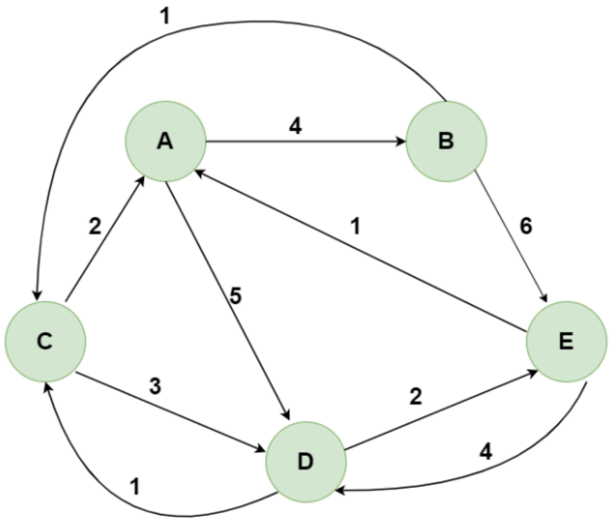
PARALLELIZATION OF FLOYD-WARSHALL ALGORITHM

By Sarath Chandra Reddy Rayapu

Floyd Warshall Algorithm:

- Dynamic programming solution for finding the shortest paths between all pairs of vertices in a weighted graph.
- It can handle positive and negative weight edges, making it versatile for various applications



	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	0	1	∞	6
C	2	∞	0	3	∞
D	∞	∞	1	0	2
E	1	∞	∞	4	0

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Applications of Floyd Warshall Algorithm

- **Network Routing**: Optimizing the path that data packets take across a network.
- **Geographical Mapping and Navigation**: calculating the shortest or fastest routes between locations
- **Social Networks**: Enhances recommendation systems and community discovery features

Sequential algorithm:

1. Start with the adjacency matrix of the graph, where the entry at i, j represents the distance from vertex i to vertex j. If there is no direct path between i and j, the distance is considered infinite.

2. For each vertex k, consider all pairs of vertices i and j. Update the distance from i to j to the minimum of its current value and the sum of the distances from i to k and from k to j.

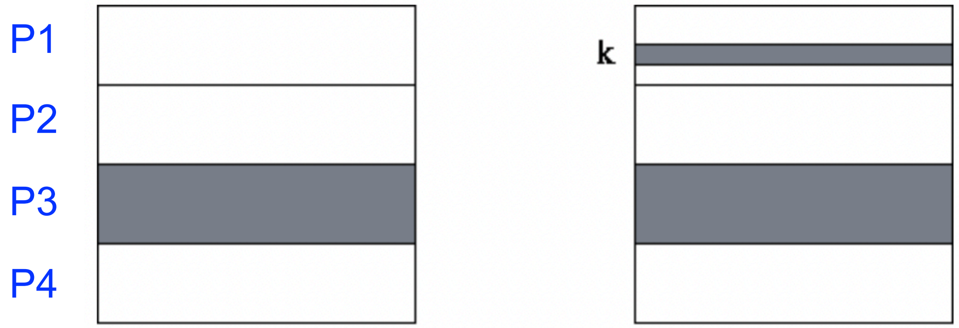
3. After considering all vertices, the matrix contains the lengths of the shortest paths between all pairs of vertices.

4. Time = O(n^3)

```
For k = 0 to n – 1:
    For i = 0 to n – 1:
        For j = 0 to n – 1:
            Distance[i, j] = min(Distance[i, j], Distance[i, k] + Distance[k, j])
```

Parallel approach

- Scatter the adjacency matrix so that each process receives a contiguous block of rows of the matrix (n/p rows)
- Each process executes the algorithm on its portion of the matrix
- The owning process broadcasts the kth row to all other processes.
- Gather the portions of the updated matrix from all processes back to the root processor




```
For k = 0 to n – 1:
    If processor_ID = owner of kth row:
        broadcast( row_k to all processors)
    For i = local_i_start to local_i_end:
        For j = 0 to n – 1:
            Distance[i, j] = min( Distance[i, j], Distance[i, k] + row_k[j] )
```

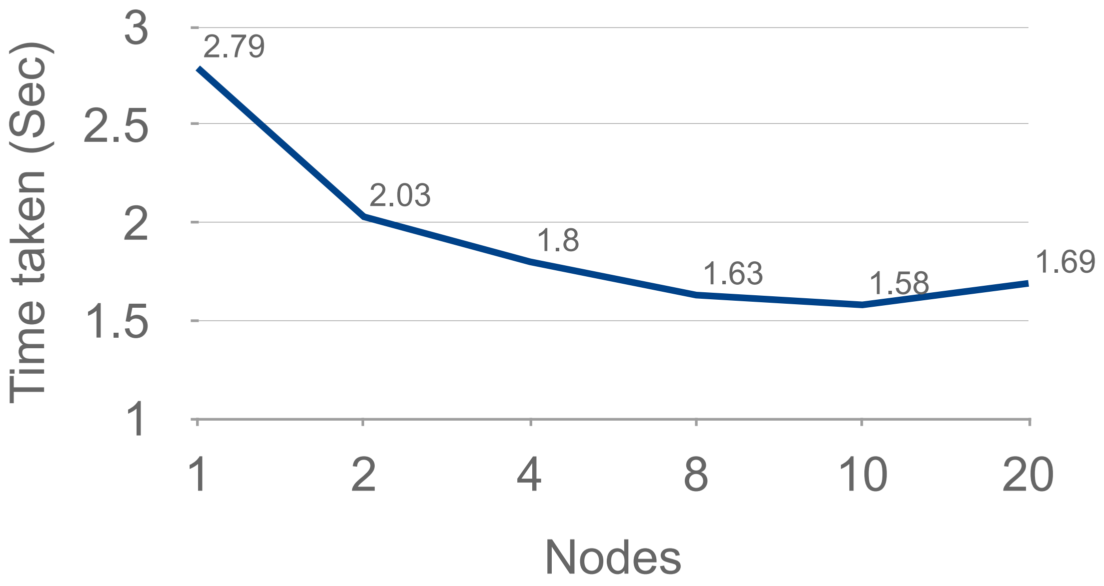
Slurm script

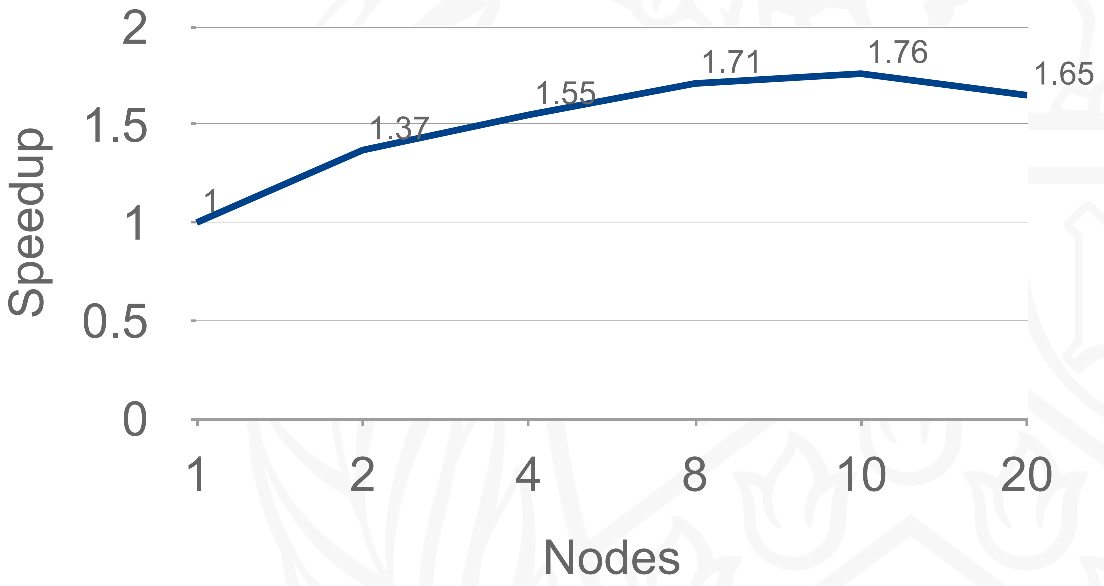
$ slurm.sh

```bash
#!/bin/bash
#SBATCH --nodes=64
#SBATCH --ntasks-per-node=1
#SBATCH --constraint=IB|OPA
#SBATCH --time=00:10:00
#SBATCH --partition=general-compute
#SBATCH --qos=general-compute
#SBATCH --job-name="floyd"
#SBATCH --output=output-floyd.out
#SBATCH --exclusive
module load intel
export I_MPI_PMI_LIBRARY=/opt/software/slurm/lib64/libpmi.so
mpicc -o floyd floyd.c
srun -n 64 floyd input_graph.txt 0
```

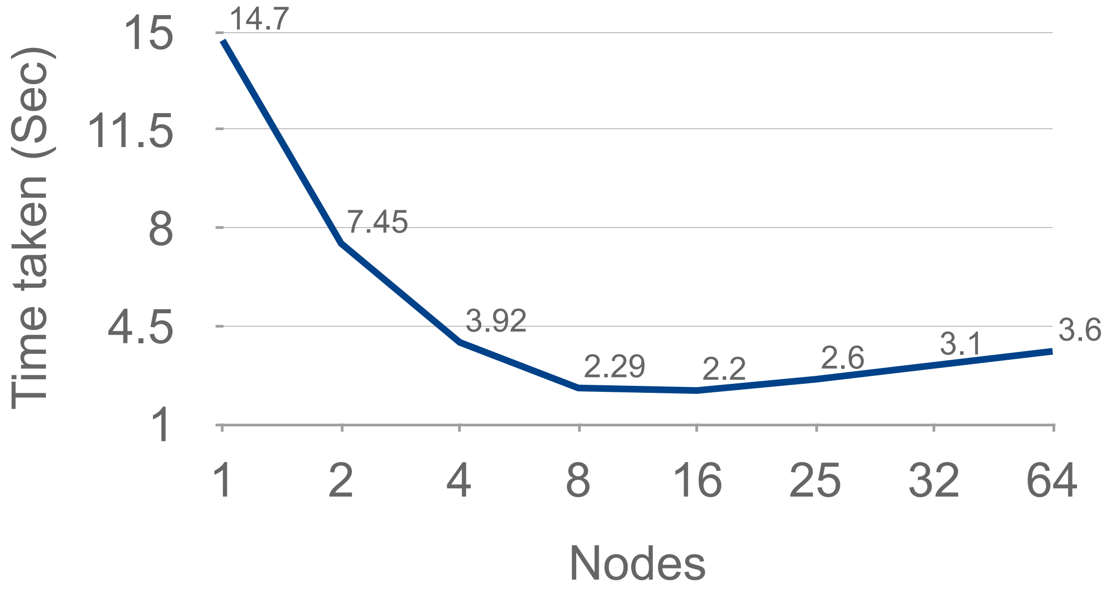
Results

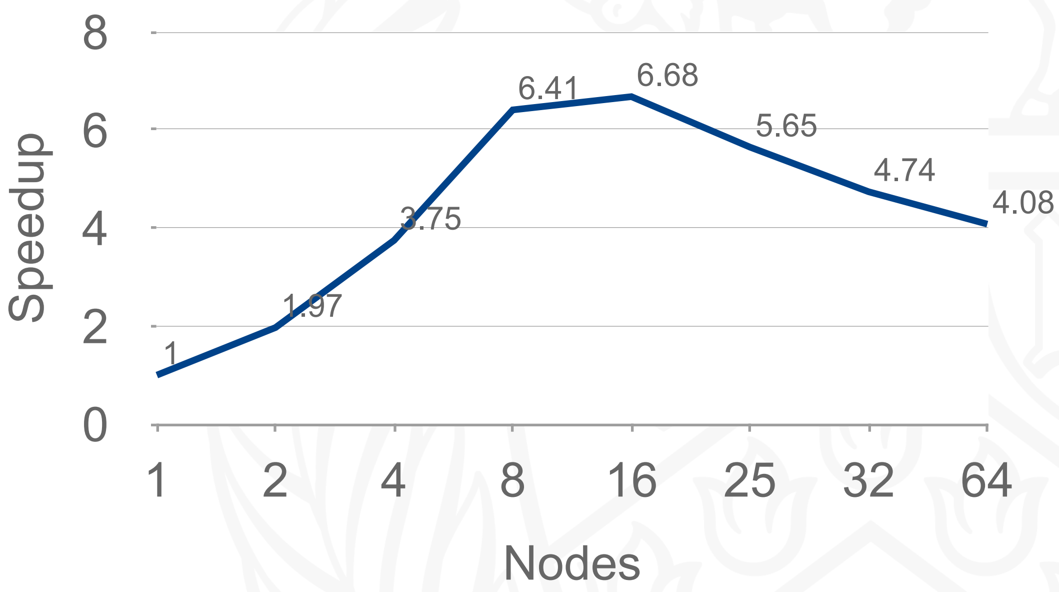
- Input graph: 1000 vertices



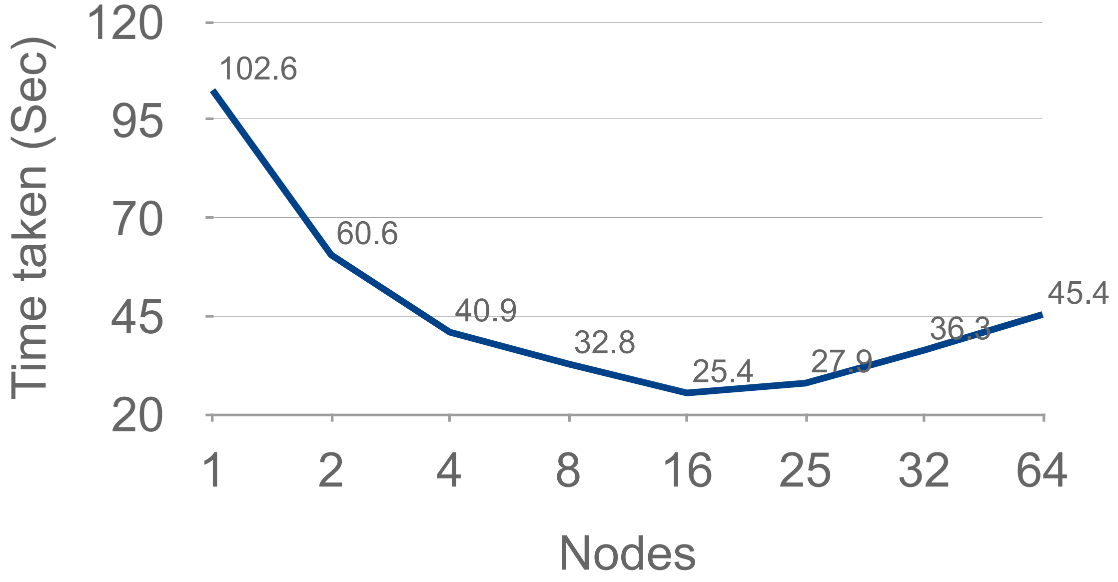


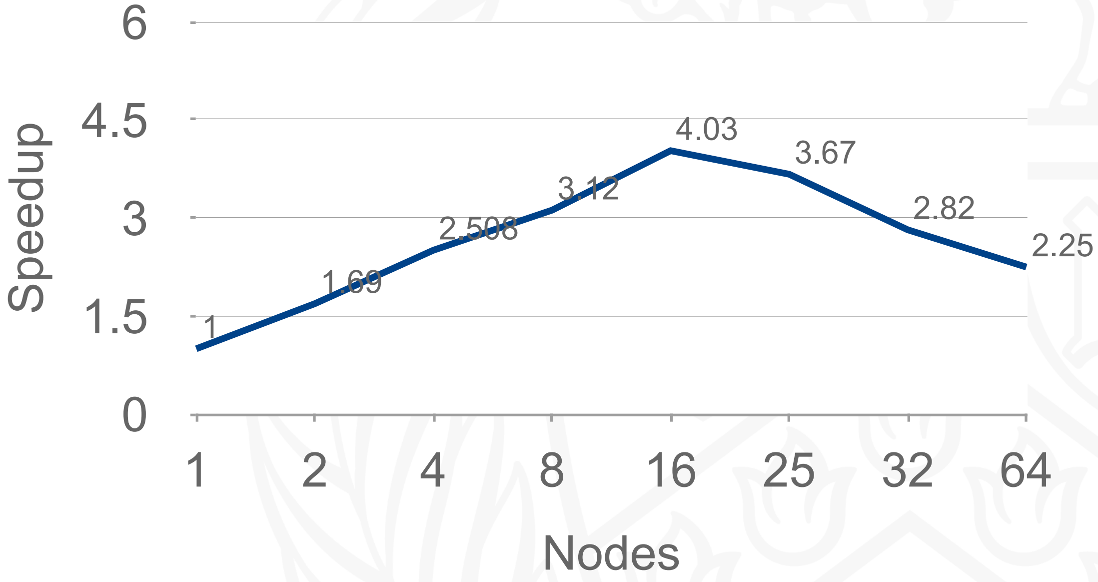
- Input graph: 2500 vertices





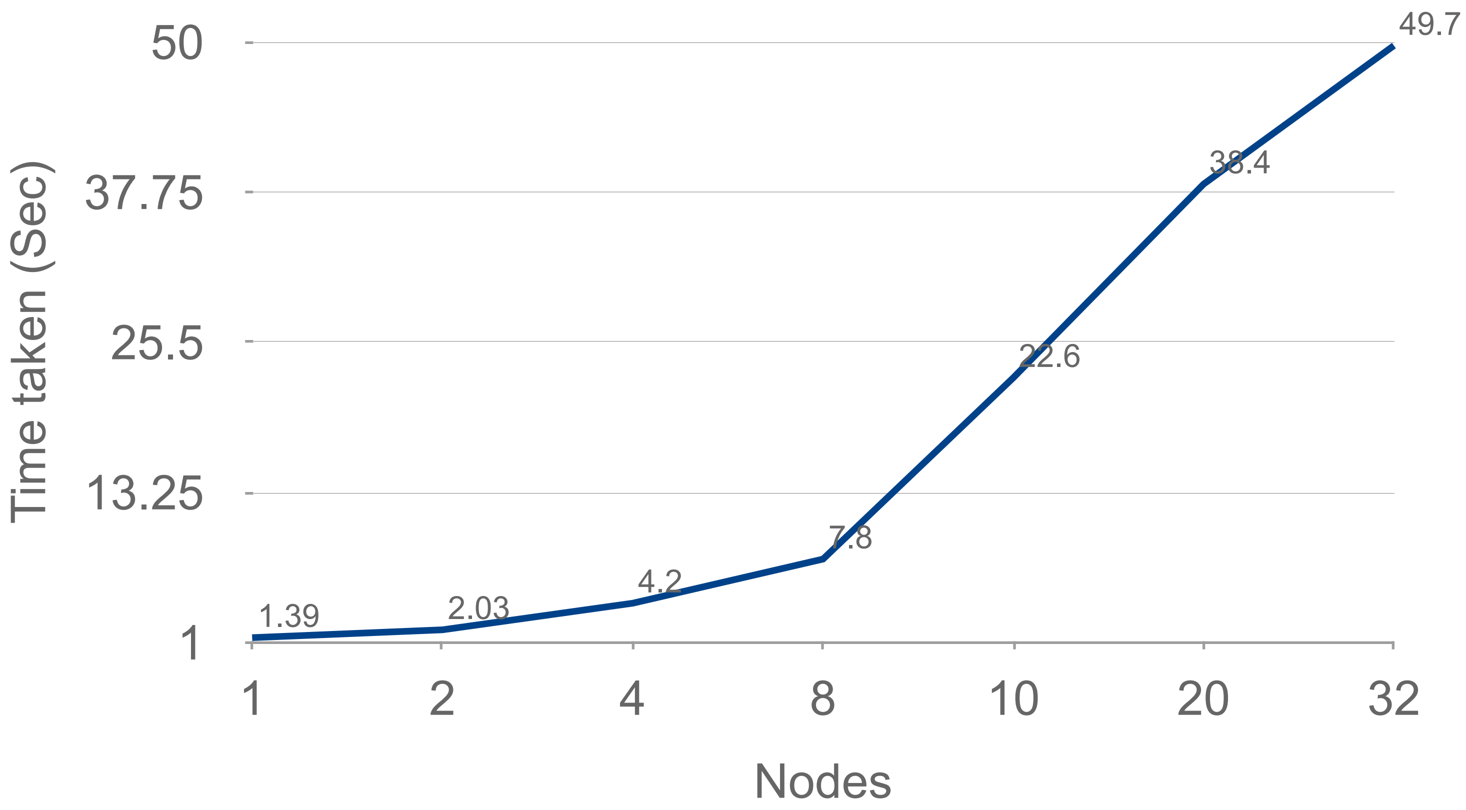
- Input graph: 5000 vertices





Weak scaling

- 500 vertices per node



Observations

- As per the results, we can see that the parallelism can be efficient only up-to a certain number of processors.
- If nodes are further added, it would increase the communication overhead.

References

- Case Study on Shortest-Path Algorithms. (n.d.). Retrieved March 20, 2023, from https://www.mcs.anl.gov/~itf/dbpp/text/node35.html

THANK YOU

University at Buffalo The State University of New York