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.
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 = $O(n^3)$
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 $n - 1$:

1. If `processor_ID` = owner of kth row:
   - broadcast(row_k to all processors)
2. 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

1  #!/bin/bash
2  #SBATCH --nodes=64
3  #SBATCH --ntasks-per-node=1
4  #SBATCH --constraint=IB|OPA
5  #SBATCH --time=00:10:00
6  #SBATCH --partition=general-compute
7  #SBATCH --qos=general-compute
8  #SBATCH --job-name="floyd"
9  #SBATCH --output=output-floyd.out
10 #SBATCH --exclusive
11 module load intel
12 export I_MPI_PMI_LIBRARY=/opt/software/slurm/lib64/libpmi.so
13 mpicc -o floyd floyd.c
14 srun -n 64 floyd input_graph.txt 0
Results

• Input graph: 1000 vertices
- Input graph: 2500 vertices

![Graph showing time taken vs nodes](image1)

![Graph showing speedup vs nodes](image2)
• 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

THANK YOU