Parallel Implementation of Dijkstra’s and Bellman Ford Algorithm

**Team 26 B**
- Priyanka Ramanuja Kulkarni
- Meenakshi Muthuraman
- Aslesha Pansare
- Nikhil Kataria
Single Source Shortest Path Problem

- The Problem of finding the shortest path from a source vertex $S$ to all vertices in the graph
- Weighted graph $G = (V,E)$
- Distance from $S$ to all the vertices
Dijkstra’s Algorithm

- Solution to single source shortest path algorithm in graph theory
  - Both directed and undirected graphs
  - All edges must have non-negative weights
  - Graph must be connected
  - Dijkstra’s algorithm runs in $O(E \cdot \log|V|)$
Pseudocode of Dijkstra’s Algorithm

\[
\begin{align*}
dist[s] & \leftarrow 0 \\
& \text{(distance to source vertex is zero)} \\
\text{for all } v \in V-\{s\} & \text{ do } dist[v] \leftarrow \infty \\
& \text{(set all other distances to infinity)} \\
S & \leftarrow \emptyset \\
& \text{(S, the set of visited vertices is initially empty)} \\
Q & \leftarrow V \\
& \text{(Q, the queue initially contains all vertices)} \\
\text{while } Q \neq \emptyset & \text{ do } u \leftarrow \text{mindistance}(Q, \text{dist}) \\
& \text{(select the element of Q with the min. distance)} \\
S & \leftarrow S \cup \{u\} \\
& \text{(add u to list of visited vertices)} \\
& \text{for all } v \in \text{neighbors}[u] \\
& \text{ do if } \text{dist}[v] > \text{dist}[u] + w(u, v) \\
& \text{ then } d[v] \leftarrow d[u] + w(u, v) \\
& \text{(if new shortest path found)} \\
& \text{(set new value of shortest path)} \\
& \text{(if desired, add traceback code)} \\
\text{return dist}
\end{align*}
\]
Negative Cycles

A negative cycle is a cycle in a weighted graph whose total weight is negative.
Bellman Ford’s Algorithm

BELLMAN-FORD(G,w,s)
1. INITIALIZE-SINGLE-SOURCE(G,s)
2. for i = 1 to |G.V|-1
3. for each edge (u,v) ∈ G.E
4. RELAX(u,v,w)
5. for each edge (u,v) ∈ G.E
6. if v.d > u.d + w(u,v)
7. return FALSE
8. return TRUE

● Worst case performance - 0 (|V||E|)
● Space Complexity - O(|V|)
Comparing Sequential performance

<table>
<thead>
<tr>
<th>No of Nodes</th>
<th>Dijkstra's</th>
<th>Bellman Ford's</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.02354</td>
<td>0.02355</td>
</tr>
<tr>
<td>3500</td>
<td>0.27148</td>
<td>0.27145</td>
</tr>
<tr>
<td>5000</td>
<td>0.5337</td>
<td>0.5539</td>
</tr>
<tr>
<td>7500</td>
<td>1.203</td>
<td>1.703</td>
</tr>
<tr>
<td>10000</td>
<td>2.1728</td>
<td>2.3027</td>
</tr>
<tr>
<td>13000</td>
<td>3.7544</td>
<td>4.1034</td>
</tr>
<tr>
<td>15000</td>
<td>5.04</td>
<td>5.71</td>
</tr>
<tr>
<td>17000</td>
<td>6.5479</td>
<td>6.994</td>
</tr>
<tr>
<td>20000</td>
<td>9.1839</td>
<td>9.5033</td>
</tr>
<tr>
<td>25000</td>
<td>14.4462</td>
<td>16.8432</td>
</tr>
<tr>
<td>30000</td>
<td>20.73</td>
<td>23.2003</td>
</tr>
<tr>
<td>35000</td>
<td>27.9595</td>
<td>30.9635</td>
</tr>
<tr>
<td>40000</td>
<td>36.1225</td>
<td>45.8344</td>
</tr>
<tr>
<td>50000</td>
<td>55.3941</td>
<td>75.4112</td>
</tr>
</tbody>
</table>

Comparison of Sequential Dijkstra and Bellman Ford Algorithm

[Graph showing running time for different numbers of nodes for Dijkstra's and Bellman Ford's algorithms]
Assumptions

- All the edges are undirected
- We consider a sparse matrix for the initial distance vector
- Each vertex is connected to at least 1 other vertex
- The weights of the edges are assigned randomly with a restriction on the x-axis
  rand()
Parallel Dijkstra’s Algorithm

- On each cluster identify vertices closest to the source vertex
- Use parallel prefix to select the globally closest vertex
- Broadcast the results to all cores
- On each cluster update the distance vectors
- Running time = $o(V^2/P + V\times\log(P))$
Parallel Approach

1. Allocate graph nodes to different processors.
2. Shortest distance is computed using parallel prefix.
3. Broadcast the Result to all nodes after closest path.

- MPI_Reduce(MPI_IN_PLACE,nextAN,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
- MPI_Bcast( G->node[0], G->N*G->N, MPI_CHAR, 0, MPI_COMM_WORLD);
Dijkstra’s sequential Vs Parallel Implementation

![Graph showing Dijkstra's Algorithm: Sequential vs Parallel]

- **Running Time** vs **Size of Graph**
  - Sequential (blue line)
  - Parallel (orange line)

As the size of the graph increases, the running time for the parallel implementation grows significantly slower compared to the sequential implementation.
Dijkstra’s: #nodes v/s Running time
Dijkstra’s: Speed up

Speedup is defined by the following formula:

$$S_p = \frac{T_1}{T_p}$$

where:

- $p$ is the number of processors
- $T_1$ is the execution time of the sequential algorithm
- $T_p$ is the execution time of the parallel algorithm with $p$ processors

![Dijkstra's Algorithm Speedup Chart](chart.png)
Dijkstra's: Cost Analysis

Cost Vs No of Nodes

No of Nodes | 1000   | 2000   | 3000   | 4000   | 5000   |
------------|--------|--------|--------|--------|--------|
2           | 0.232  | 0.072004| 0.1522 | 0.2462 | 0.3496 |
4           | 0.05044| 0.13244| 0.2404 | 0.376  | 0.5308 |
8           | 0.132  | 0.268  | 0.5256 | 0.8552 | 1.1584 |
12          | 0.2208 | 0.6972 | 0.9216 | 1.314  | 1.9332 |
14          | 0.2884 | 0.5628 | 1.14032| 1.6464 | 2.2694 |
16          | 0.3056 | 0.5648 | 1.0224 | 1.4464 | 2.1232 |
32          | 0.784  | 1.4816 | 2.4064 | 3.6384 | 5.3952 |
Bellman Ford: sequential vs parallel
Bellman Ford: Running Time v/s # of Nodes

![Graph showing Bellman Ford's running time vs number of nodes]
Bellman Ford: Speedup
Bellman Ford: Cost Analysis
Comparing parallel performance
References

- Implementing Parallel Shortest-Paths Algorithms (1994) by Marios Papaefthymiou and Joseph Rodrigue
- Algorithms Sequential & Parallel: A Unified Approach / Edition 2 by Russ Miller and Lawrence Boxer
- An Experimental Study of a Parallel Shortest Path Algorithm for Solving Large-Scale Graph Instances by Kamesh Madduri and David A. Bader, Georgia Institute of Technology and Jonathan W. Berry Sandia National Laboratories and Joseph R. Crobak, Rutgers University
- Parallel Algorithms by Guy E. Blelloch and Bruce M. Maggs, School of Computer Science, Carnegie Mellon University