PARALLEL IMPLEMENTATION OF BELLMAN FORD ALGORITHM

CSE 633 – Parallel Algorithms
Instructor : Dr. Russ Miller
Presented by Shreya Reddy Gouru
Outline

- Problem Statement
- Bellman Ford Algorithm
- Example
- Sequential Algorithm
- Approaches to Parallelize it
- Pseudo code for Course grained approach
- To do task status
- Implementation Results
- Future work
- References
Problem Statement

- Single source shortest path.
- To find shortest path from a given vertex to all other vertices in a weighted directed graph.
- To detect negative cycles in the graph
Bellman Ford Algorithm

• Computes shortest path from a source to all vertices in a weighted graph.
• Capable of handling graphs with negative edge weights.
• Dijkstra vs Bellman Ford.
• Applications in routing.

How it works?

• Relaxes all edges |V-1| times to approximate distances, where |V| is the number of vertices in a graph.
• In case of negative cycle, distances are updated even after last iteration.
Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

```
A  B  C  D  E
0  -1 ∞ ∞ ∞
0  -1 4 ∞ ∞
0  -1 2 ∞ ∞
```

```
A  B  C  D  E
0  -1 4 5 2
0  -1 2 5 3
```
Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>4</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>∞</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-2</td>
<td>1</td>
<td>∞</td>
</tr>
</tbody>
</table>
Sequential Algorithm

```python
function bellmanFord(G, S)
    for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
        distance[S] <- 0
    for each vertex V in G
        for each edge (U,V) in G
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U
        for each edge (U,V) in G
            if distance[U] + edge_weight(U, V) < distance[V]
                Error: Negative Cycle Exists
    return distance[], previous[]
```

TIME COMPLEXITY : $O(V*E)$
Approaches to Parallelize it

COARSE GRAIN

• Each Processor is assigned a subset of edges in the beginning and assignment never changes.
• Iteratively performs computation and communication phases.
• Each processor relaxes its subset of edges and updated local distances.
• At the end of computation, distance vector equal to the minimum of all labels is updated in all processors.

FINE GRAIN

• Each Processor maintains a list of vertices ordered by the labels in the distance vector.
• During communication phase, each processor selects minimum element on its local distance vector and a vertex which has least distance is selected by all processors.
• Edges from that vertex are relaxed by all processors in its subgraph. Computation phase is same in both approaches.
Pseudocode for coarse grained Algorithm

\[ f_g = f_i \quad \text{;; } f_i \text{ is initially FALSE} \]
\[ f_i = \text{FALSE} \]
\[ \text{for each vertex } u \]
\[ \quad \text{do if } d(u) > d_{\text{min}}(u) \]
\[ \quad \quad \text{then } d(u) \leftarrow d_{\text{min}}(u) \]
\[ \quad \quad \pi(v) \leftarrow \infty \]
\[ \quad f_g = \text{TRUE} \]
\[ \quad \text{if outdegree}(u) > 0 \]
\[ \quad \quad \text{then mark } u \]
\[ \text{for each vertex } u \text{ in order} \]
\[ \quad \text{do if } u \text{ is marked} \]
\[ \quad \quad \text{then unmark } u \]
\[ \quad \text{for each edge } (u, v) \]
\[ \quad \quad \text{do if } d(v) < d(u) + w(u, v) \]
\[ \quad \quad \quad \text{then } d(v) \leftarrow d(u) + w(u, v) \]
\[ \quad \quad \pi(v) \leftarrow u \]
\[ \quad f_i = \text{TRUE} \]
\[ \quad \text{if outdegree}(v) > 0 \]
\[ \quad \quad \text{then mark } v \]

\[ \text{if } f_g = \text{FALSE} \]
\[ \quad \text{then terminate} \]
To do

- Run the algorithm on larger input ✓
- Implement couple of heuristics from research paper ✓
- Fine grained approach ✓
- Comparison of Course grained and Fine grained approach
- Negative cycle detection ✓
Implementation Results

GRAPH : 1000 VERTICES

GRAPH : 5000 VERTICES

GRAPH : 10000 VERTICES
Sequential vs Parallel
Speed up

Speed Up = $T_{\text{sequential}} / T_{\text{parallel}}$
Future Work

- Implementation using CUDA
- Fine grained approach on large number of nodes
- Increase input data up to $2^{32}$ vertices
References

• Implementing Parallel Shortest-Paths Algorithms (1994) by Marios Papaefthymiou and Joseph Rodrigue.


• Algorithms Sequential & Parallel: A Unified Approach by Russ Miller and Lawrence Boxer

• https://mpitutorial.com/tutorials/

• https://www.programiz.com/dsa/bellman-ford-algorithm

• https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/
Thank You