Parallelization Strategy

Team Members:

Mesam E Tamaar Khan	22i-1304	CS-D
Aamna Saeed	22i-1179	CS-D
Syed Tashfeen Hassan	22i-0860	CS-D

Parallelization Strategies for Dynamic Single-Source Shortest Paths (SSSP)

This guide explores practical strategies to parallelize the task of updating Single-Source Shortest Paths (SSSP) in large, dynamic networks. These methods are tailored to take advantage of different computing environments:

- **MPI** for distributing workloads across multiple nodes.
- **OpenMP** or **OpenCL** for shared-memory and accelerator-based parallelism.
- **METIS** for smart graph partitioning.

Each approach is explained separately, followed by a combined strategy that leverages all three.

---

1. MPI-Based Strategy:

Objective:

Use MPI to break the graph across several computing nodes. Each node processes a portion of the graph and communicates with others when needed.

How It Works:

1. **Graph Partitioning**:
    - Divide the graph across `N` MPI processes.
    - Each process handles a portion of vertices and edges.
2. **Edge Change Detection**:
    - When an edge is inserted or removed, determine if it's relevant locally or shared across partitions.
    - Notify other MPI processes if needed.
3. **Local SSSP Update**:
    - Each process updates its part of the SSSP tree independently.
4. **Communication**:
    - Share updated boundary values (distance, parent) with neighbors.
    - Use collective or point-to-point MPI calls for efficiency.

Pseudocode:

```
// Process p_i
```

```
Initialize local subgraph G_i
Build initial SSSP tree T_i

while (true) {
    Receive changed edges DE
    for each edge e(u, v) in DE:
        if e belongs to local partition:
            mark affected nodes
            update local tree
            communicate boundary updates

    do {
        for each affected node:
            relax neighbor distances
        exchange boundary data
    } while (updates exist)
}
```

---

2. OpenMP / OpenCL Strategy:

a. OpenMP (For CPU Multithreading)

Approach:

- Run parallel loops over affected graph elements.
- Allow multiple threads to update distances concurrently.
- Use minimal synchronization by converging iteratively.

Pseudocode:

```
#pragma omp parallel
{
    #pragma omp for schedule(dynamic)
    for each edge in DE:
        process insert/delete logic

    do {
        #pragma omp for
        for each affected vertex:
            relax distances to neighbors
    } while (any Affected[i] is true)
}
```

b. OpenCL (For GPU Acceleration)

Approach:

- Write GPU kernels to process edge changes and perform updates.
- Keep track of which vertices are affected and launch kernels in loops until convergence.

Pseudocode:

```
__kernel void ProcessChanges(...) {
    // Detect changes and mark affected nodes
}

__kernel void UpdateDistances(...) {
    // Attempt to relax distances to neighbors
}
```

---

3. METIS-Based Strategy: Smart Partitioning for Load Balancing

Objective:

Use METIS to split the graph into well-balanced, minimally-connected parts. This reduces communication between partitions.

How It Works:

1. Run METIS to partition the graph into `k` parts.
2. Assign each part to a process or thread.
3. Update the SSSP tree locally in each partition.
4. If the graph structure changes significantly, re-partition using ParMETIS.

Pseudocode:

```
partitions = METIS_Partition(G, k)
for part in partitions:
    assign to worker
    run SSSP update locally

if graph changed heavily:
    partitions = METIS_Repartition(G)
    reassign workloads
```

These techniques offer scalable ways to handle SSSP updates in large graphs, enabling efficient processing across CPUs, GPUs, and distributed systems.