Parallel Floyd-Warshall Paths (PFP)

Pelin Cetin, pc2807

1. Introduction

Finding the shortest path (SP) between any two nodes in a large-scale network analysis is a difficult but crucial endeavor. The SP can assist the developer in analyzing the performance of information dissemination and researching latent relationships in weighted social networks, among other things.

The all-pair-shortest-paths (APSP) issue is defined as the challenge of finding the shortest path between every pair of nodes. Parallelization has proven to be advantageous in this domain because sequential solutions for this issue typically result in lengthy runtimes.

My project will use a parallel functional Floyd-Warshall algorithm. I have seen many non-functional parallel versions of this algorithm, but not a functional one.

2. Project Idea

2.1 Objective

The goal of this project is to generate the shortest paths for all pairs in a directed graph. I will prove that the parallel version will run faster than the sequential version.

2.2 Algorithms

For directed graphs, the Floyd-Warshall algorithm finds the shortest path for all pairs. The algorithm generates shorter pathways repeatedly using the adjacency matrix of a graph as input. The distance-matrix contains all of the shortest pathways after |V| iterations.

The main principle behind parallelizing this algorithm is to divide the matrix and distribute the computation amongst the processes. Each process has its own section of the matrix to which it is assigned. The matrix is divided into squares of equal size, each of which is given to a process.

Here is an example of a non-functional parallel pseudocode of Floyd-Warshall with 2-D block mapping (taken from https://moorejs.github.io/APSP-in-parallel/)

```
floyd(C, A, B) {
  C, A, B are bxb matrices
  for (int k = 0; k < b; k++) {
    for (int j = 0; j < b; j++) {
      for (int i = 0; i < b; i++) {
        C[i][j] = min(C[i][j], A[i][k] + B[k][j])
      }
    }
  }
}

blocked_floyd_warshall(W, n) {
  // split W into "blocks" with blocksize "b"
  // For simplicity if b divides n we will have B blocks
  for (int k = 0; k < B; k++) {
    // Dependent Phase
    floyd(W_kk, W_kk, W_kk);

    // Partially Dependent Phase
    parallel for (int j = 0; j < B && j != k; j++)
      floyd(W_kj, W_kk, W_kj)

    parallel for (int i = 0; i < B && i != k; i++) {
      floyd(W_ik, W_ik, W_kk);
      for (int j = 0; j < B && j != k; j++) {
        // Independent Phase
        floyd(W_ij, W_ik, W_kj);
      }
    }
  }
```

2.3 Deliverables

I will provide serial and parallelized implementations of all pairs shortest paths for the completion of this project. Additionally, I will provide test results showing comparisons of the two versions of the algorithm.

3. Resources

https://moorejs.github.io/APSP-in-parallel/
https://en.wikipedia.org/wiki/Parallel_all-pairs_shortest_path_algorithm
https://ieeexplore.ieee.org/document/4447721?arnumber=4447721