Efficient GPU Implementation of Graph Algorithms

CS516: Final Project

Course Guide: Dr. Vishwesh Jatala

Team Members:

Moyank Giri, 12310830, M.Tech DSAI

Harshit Kumar, 12310680, M.Tech DSAI

Tanmoy Bhowmick, 12311110, Mtech DSAI

Manish Rai, 12310790, Mtech DSAI

Abstract

- Aim to enhance performance through careful consideration of algorithm design and data structures.

- Targeting efficient solutions for processing large-scale graph data

- Addressing the surge in graph data scale and complexity with optimized implementations.

Problem Statement

- Implement Efficient Graph Algorithms which allows faster processing of graphs utilizing GPU Capabilities

Project Contributions

- The projects contributions includes:
  - CSR to MTX Conversion and vice-versa
  - CSR to CSC Conversion and vice-versa
  - Duplicate Edge Removal
  - Self-loop Removal from CSR Graph
  - Isolated Vertex Removal from CSR Graph
  - Node Degree Computation
  - Histogram of degree of vertex
  - Diameter of the graph
  - Minimum and Maximum Degree with NodeIDs

Implementation Details: Graph Conversions

- CSR to mtx
    - Thread-level parallelism is employed through CUDA kernels,
    - Data parallelism by assigning thread to process an element of input data arrays.
- Mtx to CSR
    - Thread-level parallelism is employed through CUDA kernels
    - Data parallelism by assigning thread to process an element of input data arrays.
    - Atomic operations for concurrent updates in prefix sum algorithm

Implementation Details: Graph Conversions

- CSR to CSC
  - Thread-level parallelism is employed through CUDA kernels,
  - Data parallelism: Each thread processes an elements of data arrays.
  - Atomic operations are used in the count and prefix sum algorithm for column pointers

- CSC to CSR
  - Thread-level parallelism is employed through CUDA kernels,
  - Data parallelism: Each thread processes an elements of data arrays.
  - Atomic operations are utilized in the count and prefix sum algorithm for row pointers

Implementation Details: Graph Operations

- Duplicate Edge Removal (CSR Graph)
    - GPU parallelism for edge list from the CSR graph
    - Thrust library for sorting and removing duplicates using GPU parallelism

- Self-Loop Removal
    - Thread-level parallelism is employed through CUDA kernels

- Isolated Vertices removal
    - Thread-level parallelism is employed through CUDA kernels

Implementation Details: Graph Statistics

- Graph diameter
  - Thread-level parallelism is employed through CUDA kernels
  - Parallel Floyd-Warshall’s Algorithm
  - CUDA Unified Memory
  - Block Thread Synchronization

- Histogram of degree of vertex
  - Thread-level parallelism is employed through CUDA kernels
  - CUDA Unified Memory
  - Atomic Operation for histogram bins update

Implementation Details: Graph Statistics

- Degree of nodes
    - Thread-level parallelism is employed through CUDA kernels

- Minimum and Maximum Degree
    - Thread-level parallelism is employed through CUDA kernels
    - CUDA Unified Memory

Experimental Setup

- All Experiments are done on GPU available on Google Colaboratory
- GPU Specifications: 16GB of Tesla-T4 GPU
- Evaluation done using Manual inputs
  - Due to space limitations of GPU

Demo

- All results and code files are available here: https://github.com/MoyankGiri/Efficient-GPU-Implementation-of-Graph-Algorithms_POPProject/tree/main

Conclusion

- This project demonstrates the implementation of graph algorithms on GPUs using C++. It provides a practical framework for executing various graph operations efficiently

- The methodology outlined in the project offers scalability and flexibility, allowing for the efficient processing