CSC505 - Final Project Report

Andy Sauerbrei (amsauerb@ncsu.edu)
AJ Bulthuis (ajbulthu@ncsu.edu)
Tyrone Wu (tkwu@ncsu.edu)

April 28, 2023

1 Introduction

In recent years, the field of High Performance Computing (HPC) has seen significant advancements in processor technology, resulting in increased performance capabilities. To fully leverage the potential of these systems, it is essential to develop sophisticated algorithms that can efficiently utilize the underlying hardware. In this paper, we aim to address this challenge by exploring the impact of a multi-thread model on runtime performance for identifying connected components. With connected components being a fundamental topic in graph algorithms, this study aims to experiment with the application of multi-threading on Breath-first Search and Label Propagation, and identify any potential benefits and/or limitations.

2 Experimental Design

2.1 Algorithms

As mentioned earlier, our algorithms will utilize multi-threading to facilitate the concurrency model. Although these algorithms are not primarily known parallel algorithms, the following can be modified to allow to support concurrency:

- Breath First Search (BFS)
- Label Propagation

2.1.1 Frontier-based BFS

For identifying connected components, breath-first search is a classic algorithm used to traverse a connected graph. In a traditional BFS, vertices are processed sequentially, with unexplored neighbors being added to a queue. The order in which vertices are marked is determined by the queue, and this functions as a "checklist" of vertices that have not been explored.

The concept of neighbor exploration in BFS can be extended to support concurrency by partitioning the exploration among multiple threads. Rather than sequentially exploring each neighbor in the queue, a list is used to maintain the next "frontier/layer" of unexplored vertices.

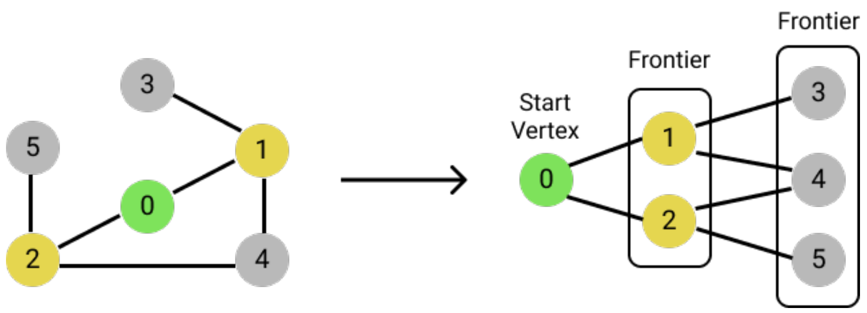

Figure 1: Visualization of frontiers

The difference is that instead of immediately processing and adding the neighbors one after another, the parallel operation waits to load/fill the unexplored frontier first before traversing it. In a way, exploration is executed in batches.

Algorithm 1 BFSConnectedComponents

```
    Input: vertices - List[Vertex]                          ▷ Adjacency List of vertices
    Return: List of vertices, one from each component
1: connectedComponents ← {}                                 ▷ List of vertices; starts empty
2: levels ← { $-1_1, \cdots, -1_n$ }      ▷ List of integers with same length as vertices; all initialized to -1
3: return
```

Algorithm 2 ParallelBFS

```
    Input: vertices - List[Vertex]                          ▷ Adjacency List of vertices
           startVertexIndex - Integer                       ▷ Index of starting vertex
           levels - List[Integer]                           ▷ List of integers
    Return: Element with the max value/degree
1: return maxNode
```

2.1.2 Label Propagation

Description
visualization
pseudocode

2.2 Hypothesis

2.3 Methodology

3 Implementation Description

3.1 Graph Object

The following graph data structure
adj list

3.2 Experimental Environment

3.2.1 Compilation Specifications

Our entire implementation of the program was done in `C++` with the following compilation specification:

- `C++` Standard: `C++ 20`
- Compiler: `clang++`
  - Version: `Debian clang version 11.0.1-2`
  - Target: `x86_64-pc-linux-gnu`

3.2.2 Hardware:

The experiment was conducted inside a Linux (Debian) Docker container on a Windows machine with the image:
`mcr.microsoft.com/devcontainers/cpp:latest`:

Hardware specification (including the container):

- System Type: 64-bit OS, x64-based processor
- Operating System:
  - Linux (Debian) Container: Linux version 5.4.72-microsoft-standard-WSL2 (gcc version 8.2.0 (GCC))
  - PC: Windows 10 Education Version 21H2 (OS Build 19044.2604)
- CPU:
  - Model Name: AMD Ryzen 5 2600X Six-Core Processor
  - Base Speed (MHz): 3.60 MHz
  - Cores: 6 cores

- Logical Processors: 12 processors
  - Processor Speed (MHz): 3.60 MHz
  - Cache Size (KB): 512 KB

- Memory:
  - Total Physical Memory (GB): 16 GB
  - Total Virtual Memory (GB): 26 GB
  - RAM Speed (MHz): 2667 MHz
  - RAM Type: DDR4

4 Experimental Results

4.1 Comparing The Three Algorithms

5 Conclusions and Future Work

5.1 Future Work