CS406 Parallel Computing

Homework 3 - GPU Implementation of Shortness Centrality

ORAN CAN ÖREN

November 2017

1 Task Description

We are provided a variety of graphs in Matrix Market format. Our aim is to compute the shortness centrality of these graphs, on GPU using CUDA technology. Shortness centrality of a connected graph $G = (V, E)$, is defined by the summation of the single source shortest distances originating from all $v \in V$. We will perform this shortest distance computation for all nodes to obtain the shortness centrality of the graph. The graphs we are working with are undirected graphs and therefore the problem can be reduced to computing all-pairs breadth-first distance of the graph.

2 Implementation Details and Results

GPUs utilize the SIMD paradigm of parallelization. Therefore, I parallelized the procedure of computing the aforementioned all pairs shortest path by assigning each thread a source vertex and letting these threads perform BFS starting by their assigned vertex.

2.1 Memory Restrictions

In order to achieve massive work-sharing such that each thread is responsible for performing BFS on the graph; I had to provide the frontier queue and distance array for each thread. Considering that both arrays have a storage index for each vertex, for large graphs the memory requirement grew quadratically. This is because the algorithm keeps two arrays of size n for a graph $G = (V, E)$ with $|V| = n$ for each vertex. Therefore the space complexity of the algorithm becomes $O(n^2)$. As I was implementing the algorithm, I did not consider that this would be an issue; however, I have underestimated the sizes of the graphs. For the "coPapers" data, the algorithm requires approximately 291 GB of memory usage; which is not feasible for modern GPUs (or CPU/RAM).

Algorithm 1: Vectorized All Pairs BFS

```
Input: distance$_i$ :$\forall v_i \in V$,
frontier$_i$ :$\forall v_i \in V$
Output: $s_i$ : sum of breadth-first distances
while $\exists v \in V$ not visited do
    for $f_i \in N$ frontier$_i$ do
        for $n \in neighbors[f_i]$ do
            if $n$ not visited then
                $distance_i[f_i] = distance_i[n] + 1$
                insert n to frontier$_i$
            end
        end
    end
end
```

2.2 Further improvement by utilizing additional streams

An idea to increase the performance of the algorithm would be to create multiple streams and having these streams perform BFS starting from distinct vertices. I anticipate that the performance of the algorithm would benefit from such work-sharing mechanism.

3 Final Notes

Managing 2 dimensional memory on CUDA is not practical, furthermore it was an inefficient use of space as mentioned in section 2.