COMP5112/MSBD5009 Parallel Programming
Assignment 3: CUDA Programming

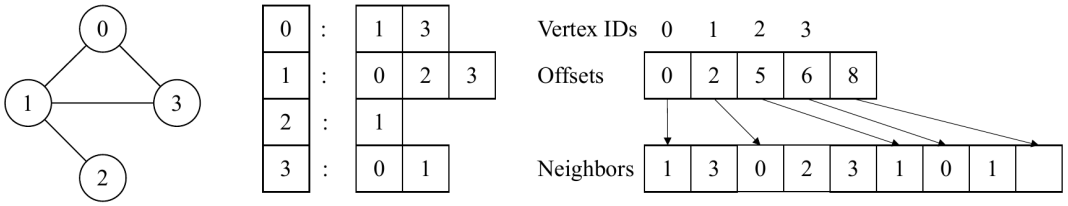
Due on Apr 27 Tue at 5:00 pm

Instructions

- This assignment counts for 15 points.
- This is an individual assignment. You can discuss with others and search online resources, but your submission should be your own code.
- Add your name, student id and email at the first line of comments in your submission.
- Your submission will be compiled and tested on Azure machines through remote terminals.
- Submit your assignment through Canvas before the deadline.
- **No late submission will be accepted!**

Assignment Description

Graph structural clustering is a common data analysis task to cluster vertices by their edge connections in the graph. SCAN (Structural Clustering Algorithm for Networks) [1] is such an algorithm that clusters vertices based on a structural similarity measure. The algorithm is efficient in both computation and memory.



(A) Original Graph. (B) Neighbors. (C) CSR.

Figure 1: Illustration of data structure.

In our implementation, a graph $G = (V, E)$ is stored in a CSR structure as shown in Fig. 1: where one array **Neighbors** stores the neighbors of all vertices, and the other array **Offsets** stores the offset of each vertex into the **Neighbors** array.

Please note that: (1) The length of the array **Offsets** is $|V| + 1$), with the last element of this array points to the end of **Neighbors**. (2) Accordingly, the length of **Neighbors** is the number of edges plus 1 ($|E| + 1$).

The SCAN algorithm, shown in Algorithm 1, can be divided into two stages: (1) find pivot vertices using the structural similarity measure, (2) expand clusters starting from the pivot vertices in the depth-first order.

Algorithm 1: The SCAN Algorithm

Input : a graph $G = (V, E)$, the structural similarity threshold ϵ, and the minimal cluster size μ

Output: the total number of clusters in G, and the cluster ID of each vertex

main()
  read files and initialization
  `// Stage 1: find pivot vertices each of which has at least` μ `neighbors whose similarity with the vertex exceeds` ϵ
  foreach $v \in V$ **do**
    foreach $w \in Neighbors(v)$ **do**
      $\Gamma(v) \leftarrow Neighbors(v) \cup \{v\}$, $\Gamma(w) \leftarrow Neighbors(w) \cup \{w\}$
      if $similarity(v, w) \leftarrow \frac{|\Gamma(v) \cap \Gamma(w)|}{\sqrt{|\Gamma(v)| \times |\Gamma(w)|}} > \epsilon$ **then**
        $Neighbors_\epsilon(v) \leftarrow Neighbors_\epsilon(v) \cup \{w\}$
      end
    end
    if $|Neighbors_\epsilon(v)| > \mu$ **then**
      $pivots[v] \leftarrow$ **true**
    else
  end
  `// Stage 2: expand clusters from pivot vertices`
  foreach $v \in V$ **do**
    if $pivots[v]$ ***and*** $!visited[v]$ **then**
      $visited[v] \leftarrow$ **true**
      $cluster\_result[v] \leftarrow v$
      expansion(v, v)
      $num\_clusters \leftarrow num\_clusters + 1$
    else
  end
  output $num\_clusters$ **and** $cluster\_result$
expansion(v, label)
  foreach $w \in Neighbors_\epsilon(v)$ **do**
    if $pivots[w]$ ***and*** $!visited[w]$ **then**
      $visited[w] \leftarrow$ **true**
      $cluster\_result[w] \leftarrow label$
      expansion$(w, label)$
    end
  end

Input and output

In this assignment, you will implement a **CUDA** version of the SCAN algorithm.

In the assignment folder:

- `main.cpp` contains the main function of the cuda version.
- `clustering.h` declares some utility functions.
- `clustering_impl.cpp` implements these utility functions.
- `clustering_cuda_skeleton.cu` is the code skeleton for your work. Your task is to:
  1. complete the `cuda_scan` function;
  2. set the variable `num_clusters`; and
  3. fill in the array `cluster_result`.

- The `results` folder contains the clustering results.
- The `sequential` folder contains the sequential version for your reference. You can check the clustering results and compare its running time.

The following shows the output of the algorithm on `text1/1.txt`.

Screen output:

```
Elapsed Time: 0.000016516 s
Number of clusters: 2
```

Result file output (sequential.txt):

```
2
-1 -1 -1 -1 -1 -1 6 6 -1 -1 -1 -1 -1 13 -1 13 -1 -1
```

`2` in the first line is the number of clusters. The second line `6 13` is the cluster IDs (`-1` if not in any cluster) of all vertices in order. Please note that in the parallel setting, we set the cluster ID be the lowest vertex ID of the pivots in the cluster. Make sure your file output follows the same format.

Compile and Run

To compile and run the sequential version, execute the following commands in the `./sequential` directory.

```
g++ -lstdc++ -std=c++11 clustering_impl.cpp main.cpp -o sequential
```

```
./sequential ../datasets/test1/1.txt 0.7 3
```

To compile and run the cuda version, execute the following commands in the `./` directory.

```
nvcc -std=c++11 clustering_cuda_skeleton.cu clustering_impl.cpp main.cpp -o cuda
```

```
./cuda ./datasets/test1/1.txt 0.7 3 8 512
```

Utility Functions

The `read_file` and `write_result_to_file` function are provided in `clustering_impl.cpp`.

`GPUErrChk` function in `clustering.h` can report the line number of the occurrence of a GPU runtime error. The usage is

- `GPUErrChk(cudaMalloc(...));`
- `GPUErrChk(cudaFree(...));`
- ...

Submission

1. You only need to complete and submit the `clustering_cuda_skeleton.cpp` to Canvas. Make sure your name information is added as comments in the first line. You can add or adjust any helper functions and variables as you wish in the skeleton file, but keep the other files unchanged.
2. We will use different input data (40M $<$ `file size` $<$ 100M) and specify different numbers of threads ($1 \leq$ `num_blocks_per_grid` ≤ 32 and $32 \leq$ `num_threads_per_block` ≤ 512) to test your program.
3. The correctness, running time and speedup of your program will be considered in grading.

4. We will perform code similarity checks. In case a submission has code similarity issues, we may request clarification and deduct partial marks or full marks on a case-by-case basis.

References

[1] Xu, Xiaowei, et al. "Scan: a structural clustering algorithm for networks." Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. 2007.