COMP5112/MSBD5009 Parallel Programming

Assignment 3: CUDA Programming

Due on 5pm on May. 9, 2019

Instructions

- This assignment counts for 20 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 as the first line of comments.
- Submit your assignment through Canvas before the deadline.
- Your submission will be compiled and tested on CS lab2 (room 4214) machines.
- **No late submissions will be accepted!**

Assignment Description

The push–relabel algorithm (alternatively, preflow–push algorithm) is an algorithm for finding maximum flows from a single source vertex to a sink vertex in a directed weighted graph. The weight on each edge represents the capacity of the edge. The algorithm starts from the source and pushes a number (the flow value) that is no more than the edge weight to each neighbor. This process goes on until all vertices have no excess flows (the incoming flow is equal to the outgoing flow). The resulting flows in the graph are the maximum flows.

The Maximum Flow Problem. Given source and sink vertices `(src, sink)` and a capacity matrix `C` of a directed graph `G=(V, E)` where the in-degree of `src` is zero, and the out-degree of `sink` is zero; find the maximum flows F from the source vertex `src` to the sink vertex `sink` with two constraints.

- Flow on an edge doesn't exceed the given capacity of the edge. (`F[v][w] ≤ C[v][w]`).
- Incoming flow is equal to outgoing flow for every vertex except `src` and `sink`.

The pseudo code of a push-relabel algorithm is given in Algorithm 1. The input parameters are the capacity matrix `C`, source and sink vertices `src` and `sink`. Four vertex properties are introduced for the push-relabel operations, namely: (1) distance labels `dist`, (2) stashed distance labels `stash_dist`, (3) excess flows `excess` and (4) stashed excess flows `stash_excess`.

The algorithm starts with an initialization of the distance labels `dist`, excess flows `excess` and the flow matrix `F` (Lines 1-4). After the preflow operation, the algorithm conducts iterative computations on each active node whose excess flow is greater than zero. The set of active nodes is denoted by `Q`. In each iteration, there are four stages: (1)

excess flow pushing, (2) distance relabeling, (3) stashed distance change applying and (4) stashed excess flow change applying.

```
Algorithm 1: Push-Relabel
Input: a directed graph $G = (V, E)$, source and sink vertices $src$ , $sink \in V$ with
       $d_{in}[src] = 0$ and $d_{out}[sink] = 0$ ($d_{in}$ and $d_{out}$ denote in-degree and out-degree
       respectively), and capacity constraints $C[(v, w)]$ on each edge $(v, w) \in E$
Output: the maximum valid flows $F[(v, w)]$ of each edge from the $src$ to the $sink$
foreach $v \in [0, |V|)$ do
    $dist[v] \leftarrow 0$, $excess[v] \leftarrow 0$
    foreach $w \in [0, |V|)$ do  $F[(v, w)] \leftarrow 0$
$Preflow(C, F, src, dist, excess)$, $Q \leftarrow \{v | v \in [0, |V|) \wedge (v \neq src) \wedge (v \neq sink)\}$
while $|Q| > 0$ do
    /* Stage 1: Push excess flows                                  */
    foreach $v \in Q$ do
        foreach $w \in [0, |V|)$ do
            if $C[(v, w)] - F[(v, w)] > 0$ and $dist[v] > dist[w]$ and $excess[v] > 0$ then
                $send \leftarrow Min(excess[u], C[(v, w)] - F[(v, w)])$
                $F[(v, w)] \leftarrow F[(v, w)] + send$, $F[(w, v)] \leftarrow F[(w, v)] - send$
                $excess[v] \leftarrow excess[v] - send$
                $stash\_excess[w] \leftarrow stash\_excess[w] + send$
    /* Stage 2: Relabel distances                                  */
    $stash\_dist \leftarrow Copy(dist)$
    foreach $v \in Q$ and $excess[v] > 0$ do
        $min\_dist \leftarrow \infty$
        foreach $w \in [0, |V|)$ do
            if $C[(v, w)] - F[(v, w)] > 0$ then
                $min\_dist \leftarrow Min(min\_dist, dist[w])$, $stash\_dist[v] \leftarrow min\_dist + 1$
    /* Stage 3: Apply stashed distance changes                     */
    $dist \leftarrow stash\_dist$
    /* Stage 4: Apply stashed excess flow changes                  */
    foreach $w \in [0, |V|)$ do
        if $stash\_excess[w] > 0$ then
            $excess[w] \leftarrow excess[w] + stash\_excess[w]$, $stash\_excess[w] \leftarrow 0$
    /* Update active nodes                                         */
    $Q \leftarrow \{v | v \in [0, |V|) \wedge (v \neq src) \wedge (v \neq sink) \wedge excess[v] > 0\}$
Procedure $Preflow(C, F, src, dist, excess)$
    $dist[src] \leftarrow |V|$
    foreach $v \in [0, |V|)$ do
        $F[(src, v)] \leftarrow C[(src, v)]$, $F[(v, src)] \leftarrow -C[(src, v)]$, $excess[v] \leftarrow C[(src, v)]$
```

In this assignment, you will implement a **CUDA version** of the push-relabel algorithm. The input file will be in the following format:

1. The first line is an integer `N`, the number of vertices in the input graph.
2. The second line is a pair of integers `(src, sink)`, representing the source and sink vertices.
3. The following lines are triplets `(v,w,c)` for the non-zero capacities of edges `(v,w)`, with `C[v][w]=c`.

The vertex labels are non-negative, consecutive integers: in an input graph with `N` vertices, the vertices will be labeled by `0, 1, 2, …, N-1`.

The output of your program consists of three lines: (1) input meta information `N`, `src` and `sink` vertices, (2) elapsed time for an execution, and (3) the maximum flow.

Here is an example of input and output:

```
Input:
6
0 5
0 1 16
0 2 13
1 2 10
2 1 4
1 3 12
2 4 14
3 2 9
3 5 20
```

```
Output:
N: 6, src: 0, sink: 5
Elapsed Time: 0.000029892 s
max flow:23
```

The code skeleton `cuda_push_relabel_skeleton.cu` is provided. You task is to complete the following function in the code:

*int push_relabel(int blocks_per_grid, int threads_per_block, int N, int src, int sink, int *cap, int *flow);*

The description of the parameters is as follows:

Parameter	Description
int blocks_per_grid	Number of blocks per grid.
int threads_per_lock	Number of threads per block.
int N	Number of vertices.
int src	The source vertex.
int sink	The sink vertex.
*int *cap*	The capacity matrix (stored in **one dimension**), N * N elements.
*int *flow*	The result flow matrix (stored in **one dimension**), N * N elements.

The element `cap[v * N + w]` stores the capacity from vertex `v` to vertex `w`.
The element `flow[v * N + w]` stores the flow from vertex `v` to vertex `w`.

Note 1: You will be given three files `cuda_push_relabel_skeleton.cu`, `main.cu` and `cuda_push_relabel.h`. You only need to complete and submit the `cuda_push_relabel_skeleton.cu` to the Canvas.

Note 2: The sequential push-relabel algorithm is provided for your reference. Your parallel version can follow the same logic flow of the sequential version, but you will need to parallelize it in CUDA.

Note 3: You can add helper functions and variables as you wish in the cu file `cuda_push_relabel_skeleton.cu`, but keep the other two files `main.cu` and `cuda_push_relabel.h` unchanged.

Note 4: We will use different input files and specify different CUDA kernel launch parameters in (*`./cuda_push_relabel <input file> <num of blocks per grid> <number of thread per block>`*) to test your program.

Note 5: The correctness, running time and speedup of your program will be considered in grading.

Note 6: We will perform code similarity checks. In case a submission has code similarity issues, we will deduct partial marks or full marks on a case-by-case basis.

References

- Goldberg A V, Tarjan R E. A new approach to the maximum-flow problem[J]. Journal of the ACM (JACM), 1988, 35(4): 921-940.
- https://www.geeksforgeeks.org/push-relabel-algorithm-set-1-introduction-and-illustration/