COMP5112 Parallel Programming

Assignment 4: CUDA Programming

Due: 5pm on 5th May 2017, Friday

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 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

Dijkstra's algorithm is a well-known solution to "the single-source shortest path(SSSP)" problem. The input graph *G(V, E)* for this assignment is connected, directed and has non-negative weights for each edge. The algorithm finds a shortest path from a specified vertex (the '*source vertex*') to every other vertex in the graph.

In this assignment, you will implement an **CUDA version** of Dijkstra's algorithm.

The input will be in following format:

1. The first line is an integer `N`, the number of vertices in the input graph.
2. The following lines are an `N*N` adjacency matrix `mat`, one line per row. The entry in row `v` and column `w`, `mat[v][w],` is the distance (weight) from vertex `v` to vertex `w`. All distances are non-negative integers. If there is no edge joining vertex `v` and `w`, `mat[v][w]` will be `1000000` to represent infinity.

The vertex labels are non-negative, consecutive integers, for an input graph with `N` vertices, the vertices will be labeled by `0, 1, 2, …, N-1`. We always use vertex `0` as the source vertex.

The output consists of the following:

1. A list of the lengths of the shortest paths from vertex `0` to each vertex `v`.
2. The shortest path `0->v`, for each vertex `v`.

Here is a sample input and output for your reference:

Input:

```
4
0 1 1000000 3    /* from vertex 0 to other vertices */
1000000 0 5 1    /* from vertex 1 to other vertices */
1 1000000 0 1    /* from vertex 2 to other vertices */
1000000 6 6 0    /* from vertex 3 to other vertices */
```

Output:

```
0 1 6 2
0
0->1
0->1->2
0->1->3
```

The code skeleton `cuda_dijkstra_skeleton.cu` is provided. Your task is to complete the following three CUDA kernel functions in the code:

*__global__ void FindLocalMin(int N, int *d_visit, int *d_all_dist, int *d_local_min, int *d_local_min_index)*

*__global__ void UpdateGlobalMin(int *global_min, int *global_min_index, int *d_local_min, int *d_local_min_index, int *d_visit)*

*__global__ void UpdatePath(int N, int *mat, int *d_visit, int *d_all_dist, int *d_all_pred, int *global_min, int *global_min_index)*

The description of the parameters is as follows:

Parameter	Description
int N	Number of vertices.
int p	Number of threads for each CUDA thread block.
*int *mat*	Adjacency matrix (stored in one dimension), N * N elements
*int *d_visit*	An array to record the status of vertices (visited or un-visited).
*int *d_all_dist*	The result array storing the final distance from the source for each vertex, N elements

*int *d_all_pred*	The result array storing the predecessor of each vertex on the shortest path from the source, N elements
*int *d_local_min*	Array to store the local minimum values for each CUDA thread block.
*int *d_local_min_index*	Array to store the local minimum indexes for each CUDA thread block.
*int *global_min*	A device memory address to store the global minimum value.
*int *global_min_index*	A device memory address to store the global minimum index.

The element `mat[v * N + w]` stores distance(weight) from vertex `v` to vertex `w`.

Note 1: You can add helper functions and variables as you wish, but keep the existing code skeleton unchanged.

Note 2: We will use different input files and specify different numbers of threads `p` (`p>=32` & `p<=1024 & p is power of 2`) to test your program.