# CS406-HW4-Report

Muhammed Muvaffak Onus - 17694

December 2017

## 1 Implementation

The program calculates all distances of each point in test dataset sequentially, finds the index of the minimum value and write it to output. First, datasets are loaded to GPU's device memory. Then for each entry in the test dataset, a device function to calculate distances is called. Then array consisting of these distances copied to the host. In the host, a simple for loop is used to find minimum and its index to be written to result array. This whole process goes sequentially for all points in test dataset. We have 19,000 points in train dataset and 1000 in test dataset. The following code is run on GPU with 19,000 blocks and 16 threads.

```
__global__
void calculateDifference(short int *trainLines,
unsigned short int *diffSquare,
short int *testLines,
short int id){

  __shared__ unsigned int s_diffSquare;
  s_diffSquare = 0;
  __syncthreads();
  short int other = testLines[id*DIMENSIONS + threadIdx.x];
  short int self = trainLines[blockIdx.x*DIMENSIONS + threadIdx.x];

  int result = other - self;

  atomicAdd(&s_diffSquare, result * result);
  __syncthreads();
  if (threadIdx.x % DIMENSIONS == 0) {
    diffSquare[blockIdx.x] = s_diffSquare;
  }
}
```

## 2 Execution Times

The following is the distribution of the time spent in GPU over functions in the final version.

| Time(%) | Time | Calls | Avg | Min | Max | Name |
|---|---|---|---|---|---|---|
| 91.38% | 33.768ms | 1000 | 33.768us | 33.602us | 34.627us | calculateDifference |
| 8.48% | 3.1323ms | 1000 | 3.1320us | 3.0720us | 4.0650us | [CUDA memcpy DtoH] |
| 0.15% | 54.852us | 2 | 27.426us | 3.8730us | 50.979us | [CUDA memcpy HtoD] |

The function calculateDifference is called 1000 times sequentially in a for loop for each test point and each time the resulting distance array is copied to the host.

The alternative version is using GPU to find minimum and only copy back that one number. Keeping minimum on device memory with atomicMin function and finding its index in another kernel without copying the whole array back to host has been implemented and tried but it resulted in same or worse runtime. Most probably it is because finding minimum over 1000-sized array is a process where overhead takes longer than the actual process.

## 3 Parallelization

As mentioned in the algorithm section, the program uses __shared__ tag in kernel to have a shared variable between threads in the same block so that they all can add result of their distance calculation to that variable. This is the main reason why there kernel is called with block number same as number of points in train dataset and thread number same as number of dimensions we have. So, at the end of 1 block execution we have the total distance of that 1 point to the corresponding point in the train dataset.

Note that exact Euclidian distance is not used since taking square root of the sum of the differences doesn't help because we use distances only to compare which one is minimum.

## 4 Running The Program

Run the following in the same directory from your terminal:

```
$ make && ./bfs
```

The output will be written to a file named myout.txt so that one can compare it with given example output.

A sequential version of the problem that runs on CPU is written as well to check the results. It is runnable in the same way.