

```
from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun

```
with open('/content/drive/My Drive/data.txt', 'r') as f:
    print("file opened")
```

file opened

```
!pip install git+git://github.com/andreinechaev/nvcc4jupyter.git
```

```
Collecting git+git://github.com/andreinechaev/nvcc4jupyter.git
  Cloning git://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-uqk41vmz
  Running command git clone -q git://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-cp36-none-any.whl size=4307 st
  Stored in directory: /tmp/pip-ephem-wheel-cache-2y7qi78d/wheels/10/c2/05/ca241da37bff;
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
```

```
%load_ext nvcc_plugin
```

created output directory at /content/src
Out bin /content/result.out

```
%%cu
#include<bits/stdc++.h>
using namespace std;
int main() {
    cout<<"Hello This is Amar Kumar - CED17I029"<<endl;
    return 0;
}
```

Hello This is Amar Kumar - CED17I029

```
%%cu
#include<bits/stdc++.h>
#include<chrono>
using namespace std::chrono;
using namespace std;

const int num_data_points = 1000;
const int num_data_center = 10;
```

```

const int num_pizza_center = 10;
const int num_iteration = 1000;

__global__ void calculateDistance(float *x1, float *y1, float x2, float y2, float *res, int t)
{
    int id = threadIdx.x;
    for(int i=id ; i<num_pizza_center ; i+=th){
        res[i] = (x2-x1[i])*(x2-x1[i]) + (y2-y1[i])*(y2-y1[i]);
    }
}

int main( int argc, char* argv[] ){
    float *x,*y, *kx, *ky;
    float *dist;
    float *d_x,*d_y, *d_kx, *d_ky;
    float *d_dist;

    ///# defining size for various data
    size_t dpsize = num_data_points*sizeof(float);
    size_t pcsize = num_pizza_center*sizeof(float);

    ///# allocating host memory
    x = (float*)malloc(dpsize);
    y = (float*)malloc(dpsize);
    kx = (float*)malloc(pcsize);
    ky = (float*)malloc(pcsize);
    dist = (float*)malloc(pcsize);

    ///# allocating cuda(device) memory
    cudaMalloc(&d_x, dpsize);
    cudaMalloc(&d_y, dpsize);
    cudaMalloc(&d_kx, pcsize);
    cudaMalloc(&d_ky, pcsize);
    cudaMalloc(&d_dist, pcsize);

    // #read input from file
    freopen("/content/drive/My Drive/data.txt", "r", stdin);
    for(int i=0 ; i<num_data_points ; ++i){
        cin>>x[i]>>y[i];
    }
    for(int i=0 ; i<num_pizza_center ; ++i){
        cin>>kx[i]>>ky[i];
    }

    //cout<<"1. fine"<<endl;
    ///# Copy host vectors to device
    cudaMemcpy( d_x, x, dpsize, cudaMemcpyHostToDevice);
    cudaMemcpy( d_y, y, dpsize, cudaMemcpyHostToDevice);
    cudaMemcpy( d_kx, kx, pcsize, cudaMemcpyHostToDevice);
    cudaMemcpy( d_ky, ky, pcsize, cudaMemcpyHostToDevice);

    int tt[10] = {1,2,4,8,16,32,64,128,256,500};
    for(int t=0 ; t<10 ; ++t){
        auto start = high_resolution_clock::now();

```

```

for(int i=0 ; i<num_iteration ; ++i){
    vector<pair<float,float>> points[num_pizza_center];
    /// for each data point, serve it under that pizza center which is nearest to it
    for(int j=0 ; j<num_data_points ; ++j){
        int temp1 = x[j], temp2 = y[j];
        //cout<<temp1<<" , "<<temp2<<endl;
        calculateDistance<<<1,2>>>(d_kx, d_ky, temp1, temp2,d_dist,2);
        cudaMemcpy( dist, d_dist, pcsize, cudaMemcpyDeviceToHost );
        int index = distance(dist,min_element(dist,dist+num_pizza_center));
        points[index].push_back({x[j],y[j]});
    }
    // # updated mean position(pizza center location)
    for(int it1=0 ; it1<num_pizza_center ; ++it1){
        float xavg = 0, yavg = 0;
        for(auto x:points[it1]){
            xavg+=x.first;
            yavg+=x.second;
        }
        kx[it1] = xavg/points[it1].size();
        ky[it1] = yavg/points[it1].size();
    }
    cudaMemcpy( d_kx, kx, pcsize, cudaMemcpyHostToDevice);
    cudaMemcpy( d_ky, ky, pcsize, cudaMemcpyHostToDevice);
    /*if(i==num_iteration -1){
        cout<<"final updted mean position"<<endl;
        for(int it2=0 ; it2<num_pizza_center ; ++it2){
            cout<<"("<<kx[it2]<<" , "<<ky[it2]<<")"<<endl;
        }
    }*/

}

auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);
cout << "Time taken by function: "<< duration.count() << " microseconds" << endl;
}

///# Release device memory
cudaFree(d_x); cudaFree(d_y); cudaFree(d_kx); cudaFree(d_ky); cudaFree(d_dist);
///# Release host memory
free(x); free(y); free(kx); free(ky); free(dist);

return 0;
}

```

```

↳ Time taken by function: 22680218 microseconds
Time taken by function: 22688623 microseconds
Time taken by function: 22646035 microseconds
Time taken by function: 22656562 microseconds
Time taken by function: 22609342 microseconds
Time taken by function: 22737826 microseconds
Time taken by function: 22742092 microseconds
Time taken by function: 22593556 microseconds

```

Time taken by function: 22556165 microseconds

Time taken by function: 22628016 microseconds

