#include <stdio.h>  
#include <stdlib.h>  
#include <math.h>  
#include <iostream>  
#include <sys/time.h>  
#include "time.h"  
using namespace std;  
\_\_global\_\_ void parMap(float \*pD, float \*netD, int grid)  
{  
unsigned int rID= blockDim.x\*blockIdx. x+threadIdx. x;  
int left, right, top, bottom;  
float x, y, fL, fR, fB, fT;  
x = pD[rID\*2];  
y = pD[rID\*2+1];  
left = (int) floorf(x);  
right = left + 1;  
bottom = (int) floorf(y);  
top = bottom +1;  
if (left>= grid|| right>= grid|| top>= grid|| bottom>= grid){  
left=0;  
right=1;  
top=1;  
bottom = 0;  
x=0.500000;  
y=0.500000;  
}  
fL = x - left;  
fR = 1 - fL;  
fB = y - bottom;  
fT = 1 - fB;  
netD[grid\*left + bottom] = netD[grid\*left + bottom] +(fT\*fR);  
netD[grid\*right + bottom] = netD[grid\*right + bottom]+(fT\*fL);  
netD[grid\*left+ top] = netD[grid\*left + top] +(fB\*fR);  
netD[grid\*right+ top] = netD[grid\*right + top] +(fB\*fL);  
}  
int main(int argc, char \*argv[])  
{  
//--------------------Declaring Variables-------------------------  
int grid = 1024, i, j , lp=1, max = grid, sizeGrid= grid\*grid;  
unsigned int par = 160000, loop=2000, sizePar = 2\*par;  
float t\_i=0.0, t\_mc\_h2d=0.0, t\_mc\_d2h=0.0, t\_pl=0.0, ti=0.0, tmc\_h2d=0.0, tpl=0.0;  
cudaEvent\_t s\_i, e\_i, s\_mc\_h2d, e\_mc\_h2d, s\_mc\_d2h, e\_mc\_d2h, s\_pl, e\_pl;  
float \*netH, \*pH, \*netD, \*pD;  
//\_\_\_Time flags\_\_\_  
cudaEventCreate(&s\_i);  
cudaEventCreate(&e\_i);  
cudaEventCreate(&s\_mc\_h2d);  
cudaEventCreate(&e\_mc\_h2d);  
cudaEventCreate(&s\_mc\_d2h);  
cudaEventCreate(&e\_mc\_d2h);  
cudaEventCreate(&s\_pl);  
cudaEventCreate(&e\_pl);  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
//-----------------------------------------------------------------  
//--------------------Declaring Variables--------------------------  
//\_\_\_start clock\_\_\_.  
cudaEventRecord(s\_i, 0);  
//\_\_\_CPU Memory allocation\_\_\_  
netH = (float\*) malloc(sizeof(float)\*sizeGrid);  
pH = (float\*) malloc(sizeof(float)\*sizePar);  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
//\_\_\_initializing grid\_\_\_  
for(i=0; i< grid; i++)  
for(j =0; j < grid; j ++)  
netH[grid\*i+j ]=0.0;  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
//\_\_\_Random particle position\_\_\_  
for( i = 0; i < sizePar; i++)  
pH[i]= ((float) rand()/(float)(RAND\_MAX) \* (float)(max-1));  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
cudaEventRecord( e\_i, 0 );  
cudaEventSynchronize( e\_i );  
cudaEventElapsedTime( &ti, s\_i, e\_i);  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
//-----------------------------------------------------------------  
//--------------------GPU memory allocation for grid--------------------------  
//\_\_\_start clock\_\_\_.  
cudaEventRecord(s\_mc\_h2d, 0);  
//\_\_\_GPU memory allocation\_\_\_  
cudaMalloc( (void \*\*)&netD, sizeof(float)\*sizeGrid);  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
//\_\_\_Data Transfer\_\_\_  
cudaMemcpy(netD, netH, sizeGrid\*(sizeof(float)), cudaMemcpyHostToDevice);  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
cudaEventRecord( e\_mc\_h2d, 0 );  
cudaEventSynchronize( e\_mc\_h2d );  
cudaEventElapsedTime( &tmc\_h2d, s\_mc\_h2d, e\_mc\_h2d);  
t\_mc\_h2d+=tmc\_h2d; //calculating time  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
//----------------------------------------------------------------------------  
for(lp=1; lp<loop; lp++)  
{  
cudaEventRecord(s\_mc\_h2d, 0);  
// Allocating GPU memory And transferring data to GPU  
cudaMalloc( (void \*\*)&pD, sizeof(float)\*sizePar);  
cudaMemcpy( pD, pH, sizePar\*(sizeof(float)), cudaMemcpyHostToDevice);  
cudaEventRecord( e\_mc\_h2d, 0 );  
cudaEventSynchronize( e\_mc\_h2d );  
cudaEventElapsedTime( &tmc\_h2d, s\_mc\_h2d, e\_mc\_h2d);  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
//\_\_\_Launching threads\_\_\_  
cudaEventRecord( s\_pl, 0 );  
//\_\_\_thread dimentions\_\_\_  
dim3 dimBlock(192);  
dim3 dimGrid((par/192));  
parMap<<<dimGrid, dimBlock>>>(pD, netD, grid);  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
cudaEventRecord( e\_pl, 0 );  
cudaEventSynchronize( e\_pl );  
cudaEventElapsedTime( &tpl, s\_pl, e\_pl);  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
//\_\_\_Time keeing\_\_\_  
t\_i+=ti;  
t\_mc\_h2d+=tmc\_h2d;  
t\_pl+=tpl;  
}  
cudaEventRecord( s\_mc\_d2h, 0 );  
// Copy the results in GPU memory back to the CPU  
cudaMemcpy(netH, netD, sizeof(float)\*sizeGrid, cudaMemcpyDeviceToHost);  
cudaEventRecord( e\_mc\_d2h, 0 );  
cudaEventSynchronize( e\_mc\_d2h );  
cudaEventElapsedTime( &t\_mc\_d2h, s\_mc\_d2h, e\_mc\_d2h);  
FILE \*f = fopen("file.txt", "w");  
par\*=loop;  
if (f == NULL){  
printf("Error opening file!\n");  
exit(1);  
}  
float avg= par/(max\*max);  
for ( i = 0; i < sizeGrid; ++i){  
fprintf (f, "%f ",((netH[i])/avg)) ;  
if (i%grid==(grid-1))  
fprintf (f, " \n" );  
}  
fclose(f);  
printf("\n\nGrid size: \t\t%d \n particle:\t %d\n", grid, par);  
printf("\nInitialisation time:\t%f \n", t\_i);  
printf("\nMemory Copy H 2 D:\t%f \n", t\_mc\_h2d);  
printf("\nMemory Copy D 2 H:\t%f \n", t\_mc\_d2h);  
printf("\nProcessing time:\t%f \n\n", t\_pl);  
//event destroy  
cudaEventDestroy(s\_i);  
cudaEventDestroy(e\_i);  
cudaEventDestroy(s\_mc\_h2d);  
cudaEventDestroy(e\_mc\_h2d);  
cudaEventDestroy(s\_pl);  
cudaEventDestroy(e\_pl);  
cudaEventDestroy(s\_mc\_d2h);  
cudaEventDestroy(e\_mc\_d2h);  
// Free the memory  
cudaFree(netD);  
cudaFree(pD);  
free(netH);  
free(pH);  
return 0;  
}