```c
/*
 * ME 2054 Parallel Scientific Computing
 * Project 1 - Finite Difference Solution of a Vibrating 2D Membrane on a GPU
 * Due: November 6,2018
 *
 * Author: Dustin (Ting-Hsuan) Ma
 *
 * Compile: gcc cpuHeat.c -O3 -lm -o CPU.exe
 * Clang: clang-format -i cpuHeat.c
 */

#include "timer.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/resource.h>

// Spacial
#define LX (REAL) 20.0f
#define LY (REAL) LX
#define NX (INT) 10
#define NY (INT) NX
#define DX LX / ((REAL) NX - 1.0f)
#define DY (REAL) DX

// Temperature
#define TMAX (REAL) 100.0f
#define TMIN (REAL) 0.0f

// Time
#define DT (REAL) 0.25f * DX *DX

// Calculation index
#define IC i + j *NX
#define IP1 (i + 1) + j *NX
#define IM1 (i - 1) + j *NX
#define JP1 i + (j + 1) * NX
#define JM1 i + (j - 1) * NX

typedef double       REAL;
typedef const double C_REAL;
typedef int          INT;

void SolveHeatEQ(C_REAL *now, REAL *out)
{
        for (INT j = 1; j < NY - 1; j++) {
                for (INT i = 1; i < NX - 1; i++) {
                        out[IC] = (((now[IP1] - 2.0f * now[IC] + now[IM1]) / (DX * DX))
                                   + ((now[JP1] - 2.0f * now[IC] + now[JM1])) / (DY * DY))
                                  * DT
                                  + now[IC];
                }
        }
}

void initializeM(REAL *in)
{
        for (INT j = 0; j < NY; j++) {
                for (INT i = 0; i < NX; i++) {
                        if (j == 0) {
                                in[IC] = TMIN;
                        }
                        if (j == NY - 1) {
                                in[IC] = TMAX;
                        }
                        if (i == 0) {
                                in[IC] = TMIN;
                        }
                        if (i == NX - 1) {
                                in[IC] = TMIN;
                        }
                }
        }
}

void analyticalSolution(REAL *out, C_REAL *x, C_REAL *y)
{
        REAL part = 0.0f;
        for (INT i = 1; i < NX - 1; i++) {
                for (INT j = 1; j < NY - 1; j++) {
                        for (REAL n = 1.0f; n <= 101.0f; n += 2.0f) {
                                REAL a = 4 * TMAX / (n * M_PI); // shows good values
                                REAL b = sin(n * M_PI * x[IC] / LX);
                                REAL c = sinh(n * M_PI * y[IC] / LY);
                                REAL d = sinh(n * M_PI);
                                part += (a * b * c / d);
                        }
                        out[IC] = part;
                        part    = 0.0f;
                }
        }
}

void meshGrid(REAL *xGrid, REAL *yGrid)
{
        for (INT j = 0; j < NY; j++) {
                for (INT i = 0; i < NX; i++) {
                        xGrid[IC] = i * DX;
                        yGrid[IC] = j * DY;
                }
        }
}

void outputMatrix(C_REAL *in)
{
        for (INT j = 0; j < NY; j++) {
                for (INT i = 0; i < NX; i++) {
                        printf("%8.4f ", in[IC]);
                }
                printf("\n");
        }
        printf("\n");
}

INT main(int argc, char *argv[])
{
        if (argc < 2) {
                perror("Command-line usage: executableName <Number of Iteration>");
                exit(1);
        }
        int ENDTIME = atoi(argv[1]);

        // Allocating memory
        REAL *xGrid      = (REAL *) calloc(NX * NY, sizeof(*xGrid));      // X Grid
        REAL *yGrid      = (REAL *) calloc(NX * NY, sizeof(*yGrid));      // Y Grid
        REAL *theta      = (REAL *) calloc(NX * NY, sizeof(*theta));      // Theta now
        REAL *theta_new  = (REAL *) calloc(NX * NY, sizeof(*theta_new));  // Next Theta
        REAL *analytical = (REAL *) calloc(NX * NY, sizeof(*analytical)); // Analytical

        // Analytical Solution
        meshGrid(xGrid, yGrid);
        initializeM(analytical);
        analyticalSolution(analytical, xGrid, yGrid);
        printf("===== Analytical ===== \n");
        outputMatrix(analytical);

        // Initializing matrix boundaries
        initializeM(theta);
        initializeM(theta_new);

        // Timing
        double start, finish;
        REAL * tmp, time = 0.0f;

        GET_TIME(start);
        while (time < ENDTIME) {
                SolveHeatEQ(theta, theta_new);
                tmp       = theta;
                theta     = theta_new;
                theta_new = tmp;

                time += DT;
        }
        GET_TIME(finish);

        printf("===== Numerical ===== \n");
        outputMatrix(theta_new);

        // Outputing CPU solution
        printf("elapsed wall time = %3.5f (ms)\n", (finish - start) * 1e3);
        printf("\n");

        // Deallocating Memory
        free(theta);
        free(theta_new);
        free(xGrid);
        free(yGrid);
        free(analytical);
        theta      = NULL;
        theta_new  = NULL;
        xGrid      = NULL;
        yGrid      = NULL;
```

```
        analytical = NULL;

        return EXIT_SUCCESS;
}
```