Shape, square

Description automatically generated

ĐẠI HỌC BÁCH KHOA HÀ NỘI

**Trường Công Nghệ Thông Tin & Truyền Thông**

----- □ 🕮 □ -----

A picture containing icon

Description automatically generated

**Middle Project Report**

***Topic:* 2D Heat Equation - Parallel Numerical Solution**

**Instructor: Dr. Vũ Văn Thiệu**

Students:

|  |  |
| --- | --- |
| Lê Hoàng Long | 20232099M |
| Mai Đức Trung | 20232177M |
| Lê Ngọc Lâm | 20232036M |
| Trần Như Thái | 20194835 |

*Ha Noi, May 2024*

[1. Problem Specification 3](#_Toc167194271)

[2. Mathematical formulation/model 4](#_Toc167194272)

[3. (Numerical - Parallel) Algorithm/Method/Approach 5](#_Toc167194273)

[4. Implement/Experiment 6](#_Toc167194274)

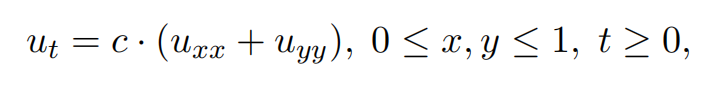
[5. Simulation 13](#_Toc167194275)

## 1. Problem Specification

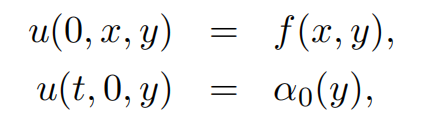
* In this report, we will discuss the numerical solution of the two dimensional Heat Equation. An approximation to the solution function is calculated at discrete spatial mesh points, proceeding in discrete time steps.
* The starting values are given by an initial value condition. We will first explain how to transform the differential equation into a finite difference equation, respectively a set of finite difference equations, that can be used to compute the approximate solution.
* We will then modify this algorithm to parallelize this task on multiple processors using CUDA.

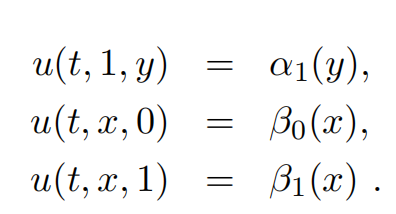
## Mathematical formulation/model

* We will illustrate the procedure with a concrete example, namely the so called “Heat Equation”



* with initial and boundary conditions





* The solution function u(t, x, y) of this differential equation describes the temperature of, for example, a thin metal plate of area 1, at every position (x, y), 0 ≤ x, y ≤ 1, at any time t ≥ 0. At the edge points of the plate, we have constant temperatures α0(y), α1(y), β0(x) and β1(x). At time t = 0, the temperature of every point (x, y) is given by f(x, y). In the simplest case, we may assume that f(x, y) is a constant function with its value somehow in the range defined by the boundary conditions.
* For example, the temperature at position (x = 0.3, y = 0.5) at time t = 10.4 is given by u(10.4, 0.3, 0.5). In this project, we try to approximate the values of the solution function u(t, x, y). This means, we will consider discrete time points t0 = 0, t1, . . . , tk, . . . and discrete positions (xi, yj ), 0 ≤ i, j ≤ (n + 1) and compute the values of u(tk, xi, yj ), 0 ≤ i, j ≤ n + 1, k ≥ 0.
* The differential equation will be replaced by a finite difference equation in the solution process. With this method we can calculate the approximated values u(tk, xi, yj ) by proceeding from time point tk−1 to time point tk, if approximate values at time tk−1 are already known for all positions (xi, yj ).

## 3. (Numerical - Parallel) Algorithm/Method/Approach

* 1. Numerical Algorithm

double[n+2][n+2] u\_old, u\_new;

double c, delta\_t, delta\_s;

Initialize u\_old, u\_new with initial values and boundary

conditions;

while (still time points to compute) {

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

u\_new[i, j] = u\_old[i, j] + c \* delta\_t/delta\_s^2 \*

(u\_old[i+1, j] + u\_old[i-1, j] - 4\*u\_old[i, j] +

u\_old[i, j+1] + u\_old[i, j-1]);

} //end of for

} // end of for

u\_old = u\_new;

} // end of while

* 1. Parallel Algorithm

Allocate memory space on host and device

Copy u\_old from host to device

Initialize number of block and number of threads per block

Loop for T times

Run parallel block

Calculate value of the cell using data from u\_old

Save the new value to the right cell in u\_new

End parallel block

Wait until all thread are finished

Run parallel block

Copy a cell from u\_new to u\_old

End parallel block

End loop

Print the result

## 4. Implement/Experiment

#include <stdio.h>

#include <malloc.h>

#include <cuda.h>

#define N 10

#define c 0.002

#define delta\_t 0.05

#define delta\_s 0.04

#define epoch 125

\_\_global\_\_ void updateGPU(float \*u\_old, float \*u\_new)

{

// [][N + 2]

int ix = blockIdx.x \* blockDim.x + threadIdx.x;

int iy = blockIdx.y \* blockDim.y + threadIdx.y;

//unsigned int idx = iy \* N + ix;

int i, j;

i = iy + 1;

j = ix + 1;

/\*printf("thread\_id (%d,%d) block\_id (%d,%d) coordinate (%d,%d) "

"global index %2d i: %d j: %d \n",

threadIdx.x, threadIdx.y, blockIdx.x,

blockIdx.y, ix, iy, idx, i, j);

\*/

int x = N + 2;

float \*uij = u\_old + (i \* x + j);

float \*uip1j = u\_old + ((i + 1) \* x + j);

float \*uis1j = u\_old + ((i - 1) \* x + j);

float \*uijp1 = u\_old + (i \* x + j + 1);

float \*uijs1 = u\_old + (i \* x + j - 1);

float \*un = u\_new + (i \* x + j);

\*un = (\*uij) + c \* delta\_t /

(delta\_s \* delta\_s) \*

((\*uip1j) + (\*uis1j) - 4 \* (\*uij) + (\*uijp1) + (\*uijs1));

}

\_\_global\_\_ void copyGPU(float \*u\_old, float \*u\_new)

{

// [][N + 2]

int ix = blockIdx.x \* blockDim.x + threadIdx.x;

int iy = blockIdx.y \* blockDim.y + threadIdx.y;

// unsigned int idx = iy \* N + ix;

int i, j;

i = iy + 1;

j = ix + 1;

/\*printf("thread\_id (%d,%d) block\_id (%d,%d) coordinate (%d,%d) "

"global index %2d i: %d j: %d \n",

threadIdx.x, threadIdx.y, blockIdx.x,

blockIdx.y, ix, iy, idx, i, j);\*/

int x = N + 2;

float \*uij = u\_old + (i \* x + j);

float \*un = u\_new + (i \* x + j);

\*uij = \*un;

}

void init1DArray(float \*u\_old)

{

// [][N + 2]

int x = N + 2;

size\_t n = N;

float \*p;

for (size\_t i = 0; i <= n + 1; i++)

{

for (size\_t j = 0; j <= n + 1; j++)

{

p = u\_old + (i \* x + j);

\*p = 25;

}

}

for (size\_t j = 0; j <= n + 1; j++)

{

p = u\_old + (j \* x + N + 1);

\*p = 100;

// u\_old[j][6] = 100;

}

}

void print1DArray(float \*u\_old)

{

int n = N + 2;

float \*cell;

for (size\_t i = 0; i < n; i++)

{

for (size\_t j = 0; j < n; j++)

{

cell = u\_old + (i \* n + j);

printf("%8.1f", \*cell);

}

printf("\n");

}

}

void parallelCode()

{

int nx = N;

int ny = N;

dim3 block(N, 1);

dim3 grid((nx + block.x - 1) / block.x, (ny + block.y - 1) / block.y);

int size = (nx + 2) \* (ny + 2) \* sizeof(float);

float \*h\_old = (float \*)malloc(size);

float \*d\_old;

cudaMalloc((void \*\*)&d\_old, size);

float \*d\_new;

cudaMalloc((void \*\*)&d\_new, size);

init1DArray(h\_old);

cudaMemcpy(d\_old, h\_old, size, cudaMemcpyHostToDevice);

for (size\_t i = 0; i < epoch; i++)

{

updateGPU<<<grid, block>>>(d\_old, d\_new);

cudaDeviceSynchronize();

copyGPU<<<grid, block>>>(d\_old, d\_new);

cudaDeviceSynchronize();

}

cudaMemcpy(h\_old, d\_old, size, cudaMemcpyDeviceToHost);

print1DArray(h\_old);

free(h\_old);

cudaFree(d\_old);

cudaFree(d\_new);

// reset device

cudaDeviceReset();

}

void serialCode()

{

const int n = N;

double u\_old[n + 2][n + 2];

double u\_new[n + 2][n + 2];

for (size\_t i = 0; i <= n + 1; i++)

{

for (size\_t j = 0; j <= n + 1; j++)

{

u\_old[i][j] = 25;

}

}

for (size\_t j = 0; j <= n + 1; j++)

{

u\_old[j][N+1] = 100;

}

int time = 0;

while (time < epoch)

{

for (size\_t i = 1; i <= n; i++)

{

for (size\_t j = 1; j <= n; j++)

{

u\_new[i][j] = u\_old[i][j] + c \* delta\_t /

(delta\_s \* delta\_s) \*

(u\_old[i + 1][j] + u\_old[i - 1][j] - 4 \* u\_old[i][j] +

u\_old[i][j + 1] + u\_old[i][j - 1]);

}

}

for (size\_t i = 1; i < n + 1; i++)

{

for (size\_t j = 1; j < n + 1; j++)

{

u\_old[i][j] = u\_new[i][j];

}

}

time++;

}

for (size\_t i = 0; i <= n + 1; i++)

{

for (size\_t j = 0; j <= n + 1; j++)

{

printf("%8.1f", u\_old[i][j]);

}

printf("\n");

}

}

int main(int argc, char \*\*argv)

{

printf("serial code\n");

serialCode();

printf("\n");

printf("parallel code\n");

parallelCode();

return 0;

}

## 5. Simulation

* Collab link: https://colab.research.google.com/drive/1JAaWb5e0336NoreJK7zhGLWaa2lWtTzQ?usp=sharing
* Result:

serial code  
 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 100.0  
 25.0 25.2 25.4 25.8 26.4 27.5 29.3 32.1 36.7 44.8 61.0 100.0  
 25.0 25.3 25.7 26.5 27.7 29.7 32.9 37.8 45.4 56.8 74.3 100.0  
 25.0 25.4 26.0 27.0 28.7 31.4 35.6 41.8 50.8 63.3 79.9 100.0  
 25.0 25.5 26.2 27.4 29.4 32.5 37.3 44.3 53.9 66.6 82.3 100.0  
 25.0 25.5 26.3 27.6 29.7 33.0 38.1 45.4 55.3 68.0 83.2 100.0  
 25.0 25.5 26.3 27.6 29.7 33.0 38.1 45.4 55.3 68.0 83.2 100.0  
 25.0 25.5 26.2 27.4 29.4 32.5 37.3 44.3 53.9 66.6 82.3 100.0  
 25.0 25.4 26.0 27.0 28.7 31.4 35.6 41.8 50.8 63.3 79.9 100.0  
 25.0 25.3 25.7 26.5 27.7 29.7 32.9 37.8 45.4 56.8 74.3 100.0  
 25.0 25.2 25.4 25.8 26.4 27.5 29.3 32.1 36.7 44.8 61.0 100.0  
 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 100.0  
  
parallel code

25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 100.0  
 25.0 25.2 25.4 25.8 26.4 27.5 29.3 32.1 36.7 44.8 61.0 100.0  
 25.0 25.3 25.7 26.5 27.7 29.7 32.9 37.8 45.4 56.8 74.3 100.0  
 25.0 25.4 26.0 27.0 28.7 31.4 35.6 41.8 50.8 63.3 79.9 100.0  
 25.0 25.5 26.2 27.4 29.4 32.5 37.3 44.3 53.9 66.6 82.3 100.0  
 25.0 25.5 26.3 27.6 29.7 33.0 38.1 45.4 55.3 68.0 83.2 100.0  
 25.0 25.5 26.3 27.6 29.7 33.0 38.1 45.4 55.3 68.0 83.2 100.0  
 25.0 25.5 26.2 27.4 29.4 32.5 37.3 44.3 53.9 66.6 82.3 100.0  
 25.0 25.4 26.0 27.0 28.7 31.4 35.6 41.8 50.8 63.3 79.9 100.0  
 25.0 25.3 25.7 26.5 27.7 29.7 32.9 37.8 45.4 56.8 74.3 100.0  
 25.0 25.2 25.4 25.8 26.4 27.5 29.3 32.1 36.7 44.8 61.0 100.0  
 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 25.0 100.0