Project B: Two-dimensional simplified heat equation

Morteza Namvar

October 26, 2021

Contents

1 Introduction

The heat equation is a partial differential equation that is mostly studied in mathematics and physics. The theory of the heat equation was first developed by Joseph Fourier in 1822 to study the diffusion of a quantity such as heat diffuses in a given region [1]. As a parabolic partial differential equation, the heat equation is one of the most widely studied equations in pure mathematics, numerical modeling, and computational fluid dynamics.

In this project, the analytical and numerical solutions of this equation are implemented to study different programming approaches. Naive C++ programming and ArrayFire [2] programming style are investigated in this project.

2 Governing equations

A simplified, two-dimensional heat equation is in the form of:

$$u_t - \alpha \left(u_{xx} + u_{yy}\right) = 0 \tag{1}$$

where α is the thermal diffusivity and in this project it is equal to 1.

Like any other PDE, to solve this equation, analytically or numerically, it is required to define a domain and the boundary condition values at that domain. In this project desired domain is a square with length 1, and boundary conditions at each edge of this domain is given as following:

$$\begin{cases} u(x=0,y,t)=0 \\ u(x=1,y,t)=0 \\ u(x,y=0,t)=0 \\ u(x,y=1,t)=0 \end{cases} \tag{2}$$

The initial condition for all positions inside the square domain is:

$$u(x,y,t=0)=1 \tag{3}$$

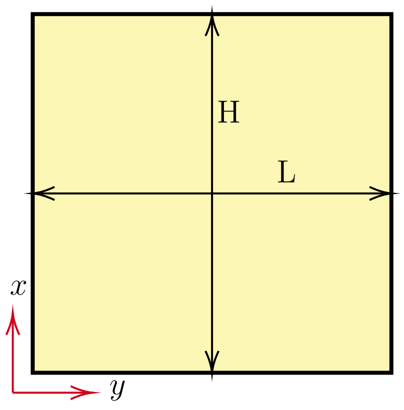


Figure 1: The domain used to solve the 2D heat equation. note that $H = L = 1$.

3 Analytical solution

In engineering applications for some cases, for example in preliminary design phase, analytical solution is used. But, in the numerical modeling this kind of solution is used for verification purposes. In this project the analytical solution of the Eq. 8 is provided as:

$$u_a(x, y, t) = \sum_{n=1}^{N\equiv\infty} \sum_{m=1}^{M\equiv\infty} A_{mn} e^{-\alpha\lambda_{mn}t} sin(n\pi x/L) sin(n\pi y/H) \tag{4}$$

$$A_{mn} = \frac{u(x, y, 0) \int_0^L \int_0^H sin(n\pi x/L) sin(n\pi y/H)\, dx\, dy}{\int_0^L \int_0^H sin(n\pi x/L)^2 sin(n\pi y/H)^2\, dx\, dy} \tag{5}$$

$$\lambda_{mn} = (n\pi/L)^2 + (n\pi/H)^2 \tag{6}$$

where in this project M = 100 and N = 100 are taken as finite numbers instead.

4 Numerical solution

First of all in this project we use the finite-difference method (FDM) for numerical modeling. In FDM domain should be discretized and also the time. So, the following notation is used to show the discretized position and time:

$$\begin{cases} x_i = i\Delta x \\ y_j = j\Delta y \\ t_k = k\Delta t \end{cases} \tag{7}$$

As we can see, i, j, and k are the steps for x, y, and t respectively. Also in this solution we use the assumption that $\Delta x = \Delta y = 1$. What we want is the solution u, which is:

$$u_{x,y,t} = u_{i,j}^{k} \tag{8}$$

Here to discretized the equation, a standard first-order forward Euler's method is used for time discretization. Besides, for spatial discretization, a second-order centered finite difference scheme is used by exploiting the stencil shown in Fig. 2.

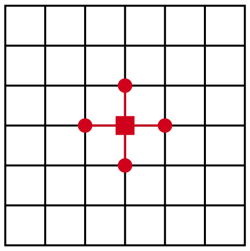


Figure 2: The stencil used for spatial discretization by second order scheme. To solve an equation on the "square" node, all the "circle" node is used.

So, the discretized form of Eq. 8 would be as:

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k}}{\Delta t} - \left(\frac{u_{i+1,j}^{k} - 2u_{i,j}^{k} + u_{i-1,j}^{k}}{\Delta x^2} + \frac{u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k}}{\Delta y^2} \right) = 0 \tag{9}$$

After some algebraic operations and by taking into account that $\Delta x = \Delta y$, this equation would be as:

$$u_{i,j}^{k+1} = u_{i,j}^{k} + \gamma \left(u_{i+1,j}^{k} + u_{i-1,j}^{k} - 4u_{i,j}^{k} + u_{i,j+1}^{k} + u_{i,j-1}^{k} \right) \tag{10}$$

where

$$\gamma = \alpha \frac{\Delta t}{\Delta x^2} \tag{11}$$

As we use explicit method for time discretization, so it will be numerically stable just at this condition:

$$\Delta t < \frac{\Delta x^2}{4\alpha} \tag{12}$$

5 Different implementations for solving heat equation

The algorithm for solving the heat equation is presented in Algorithm 1. Here, it is needed to reserve two arrays to store the solution values at the time "n" and "n+1" according to the Euler scheme. These values are named as U_n and U_{n+1}. To discretize the domain there are two parameter Nx and Ny which represent number of nodes in x and $y$directions. as mentioned in Algorithm 1.

Algorithm 1: Solving 2D heat equation numerically.

```
1  Declare $U_n$ and $U_{n+1}$ arrays for the solution at time "n" and "n+1" respectively
2  Initialize $U_n$ and $U_{n+1}$
3  Applying boundary condition
4  while Time marching condition do
5      Substitute new values to the old values
6      for $i = 1$ to $Nx$ do
7          for $j = 1$ to $Ny$ do
8              Solving equation
9          end
10     end
11     Apply Boundary Condition
12 end
```

In this project, it is desired to implement Algorithm 1. by three different approaches including naive C++, vectorized ArrayFire, and using convolution function of ArrayFire library.

5.1 Naive C++

The naive C++ implementation is a reference to evaluate the other implementations. It should mention that this implementation follows Algorithm 1 step by step.

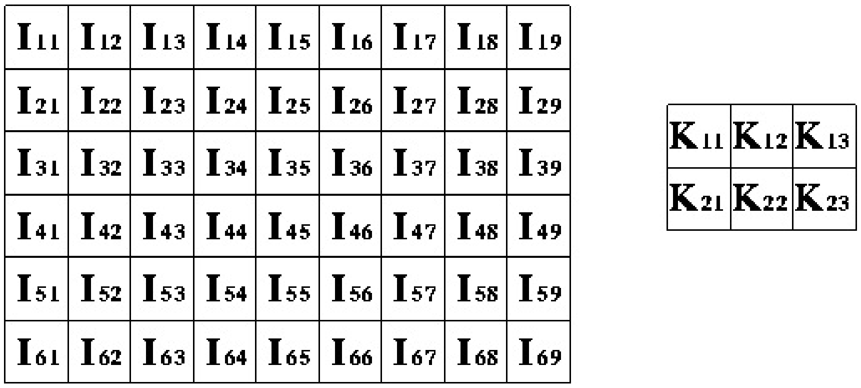
5.2 ArrayFire/vectorization

In this implementation, we are using the vectorization capabilities of the ArrayFire library. So, "loops" should be eliminated in this implementation. Here, we are using the af::seq to traverse the variable of a loop.

5.3 ArrayFire/Convolution

Convolution is a fundamental and simple mathematical operation in many common image processing operators. Convolution provides a way of 'multiplying together' two matrices, to produce a third matrix of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values. Fig.3 shows an example of these two matrices to illustrate convolution. So, in our example, the value of the bottom right pixel in the output image will be given by [3]:

$$Q_{57} = I_{57}K_{11} + I_{58}K_{12} + I_{59}K_{13} + I_{67}K_{21} + I_{68}K_{22} + I_{69}K_{23} \tag{13}$$



I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}	I_{18}	I_{19}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}	I_{28}	I_{29}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}	I_{38}	I_{39}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}	I_{48}	I_{49}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}	I_{58}	I_{59}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}	I_{68}	I_{69}

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

Figure 3: Left Matrix (image) and right Matrix (kernel) to illustrate convolution. The labels within each grid square are used to identify each square [3].

6 Results and discussion

In this section as the first step, the code is verified by comparing the results to the analytical solution. Then, the accuracy of the method of discretization is investigated. After that, the performance of different programming paradigm in studied.

6.1 Verification

Based on the nature of heat equation, it is expected that the amount of temperature (equation variable) be the highest value at the centre of the domain. It is because the heat equation has a diffusive nature. As depicted in Fig. 4 the contour of temperature shows the behaviour of equation correctly.

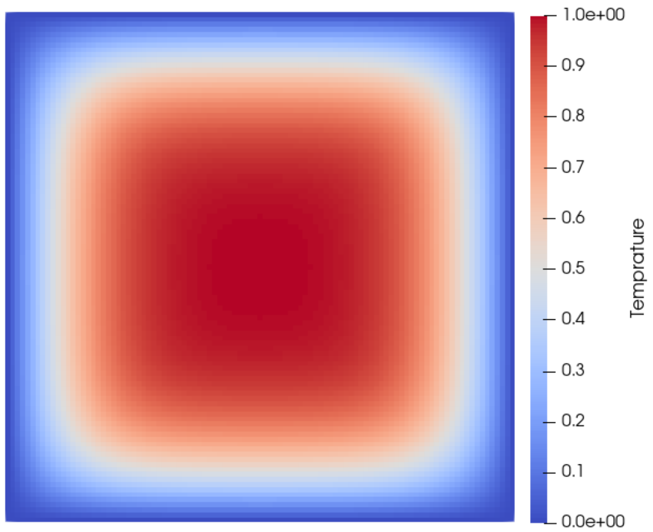


Figure 4: Contour of numerical solution of heat equation at $t = 0.01$.

But, to have a quantitative comparison, the numerical solution should be compared to the analytical solution. Here, the numerical and analytical solutions are obtained at the centerline of the domain. As can be seen in Fig. 5 there is a good agreement to the analytical solution.

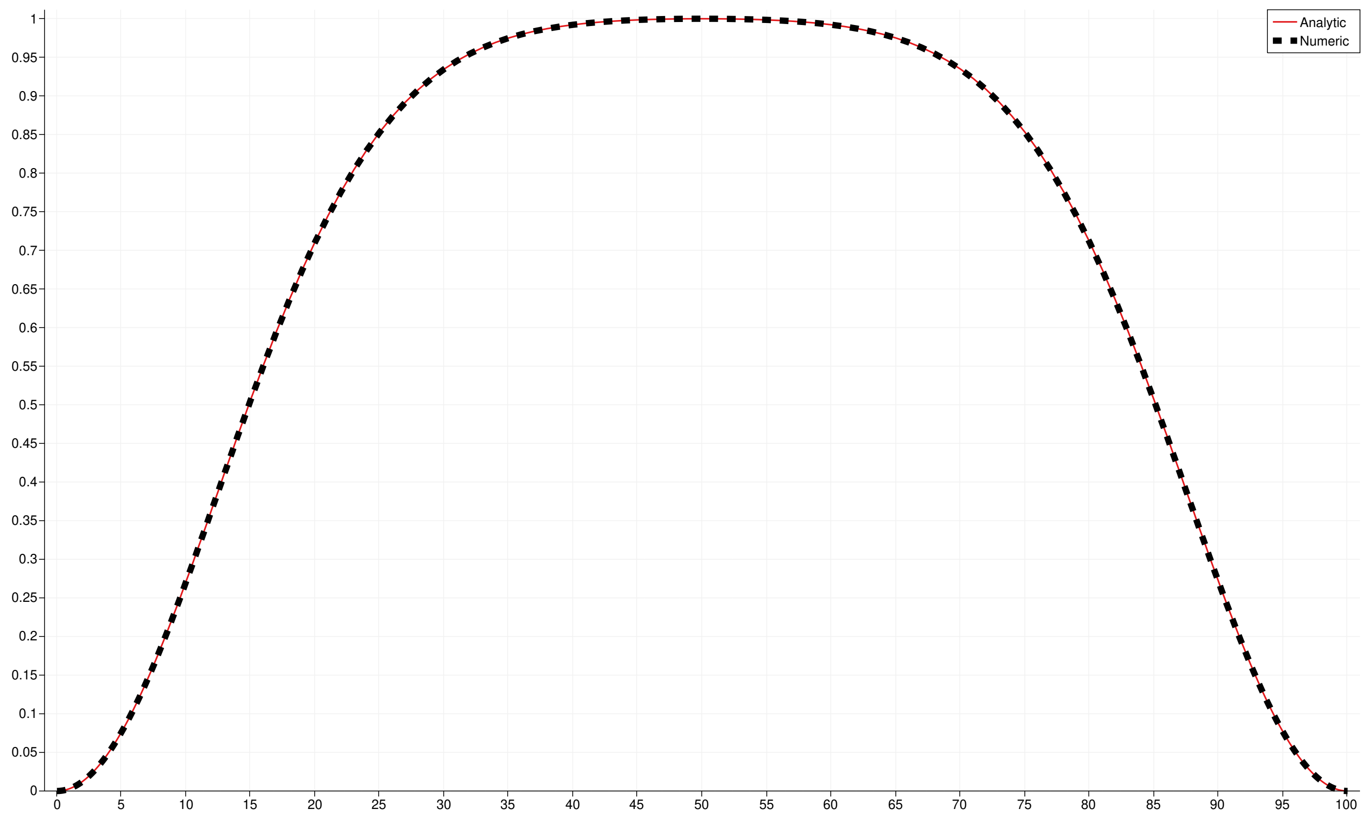


Figure 5: Comparison of analytical and numerical solution profile at the center of the square at $t = 0.01$.

To measure the error between the analytical and numerical implementations, the L2 norm is using according to the Eq. 14:

$$\text{L2error} \equiv ||u^n - u^a||_{L_2} = \sqrt{\sum_{i=0}^{Nx-1} \sum_{j=0}^{Ny-1} \left(u^n_{i,j} - u^a_{i,j}\right)^2} \tag{14}$$

where u^a and u^n represent analytical and numerical solution respectively. Also Nx and Ny are the number of nodes in $x-$ and $y-$direction.

Table 1 shown the *L2error* of different implementations and the analytical solution provided by Eq. 4. As can be seen in this table, all the implementations have the same results as the machine accuracy.

Table 1: L2error for different programming paradigm. The bold digit are the difference between different programming approaches.

Programming approach	**L2error**
Naive C++	0.0002632871862556**24**
ArrayFire/vectorization	0.0002632871862556**01**
ArrayFire/Convolution	0.0002632871862556**26**

Table 2 shows the L1 error between different programming approaches and base code. According to this table, and by considering the amount of the error, it is clear the implementation is correct.

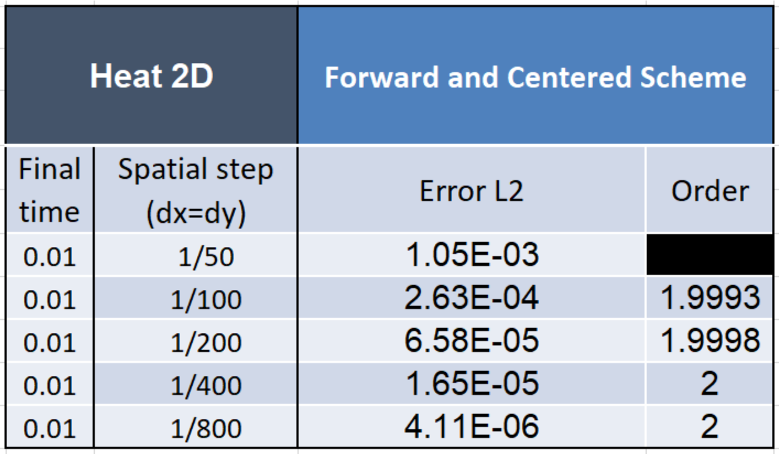
Table 2: L1error for different programming paradigm.

Programming approach / Base code	**L1error**
Analytical arrayfire / Analytical naive C++	3.07484072925579e-13
Numerical arrayfire-vectorization / Numerical naive C++	6.062355478730908e-13
Numerical arrayfire-convolution / Numerical naive C++	7.079549620148118e-13

In summary, by comparing the results and measuring the error it is concluded the implementation is correct.

6.2 Order of accuracy

After verification, it is required to verify the order of accuracy of the method of discretization. As in this project, a ”centered” approach has been used for spatial discretization, it is expected that the implantation is second-order accuracy. Fig. 6 shows the trend of error as a function of grid resolution. Based on this table it is clear the order of accuracy is 2 as expected. For visualization, the order of accuracy is plotted in Fig. 7. In this figure, a nominal second order line is plotted for a better understanding of the order of accuracy.



Heat 2D		Forward and Centered Scheme	
Final time	Spatial step (dx=dy)	Error L2	Order
0.01	1/50	1.05E-03	■
0.01	1/100	2.63E-04	1.9993
0.01	1/200	6.58E-05	1.9998
0.01	1/400	1.65E-05	2
0.01	1/800	4.11E-06	2

Figure 6: Comparison of analytical and numerical solution profile at the centre of square at $t = 0.01$.

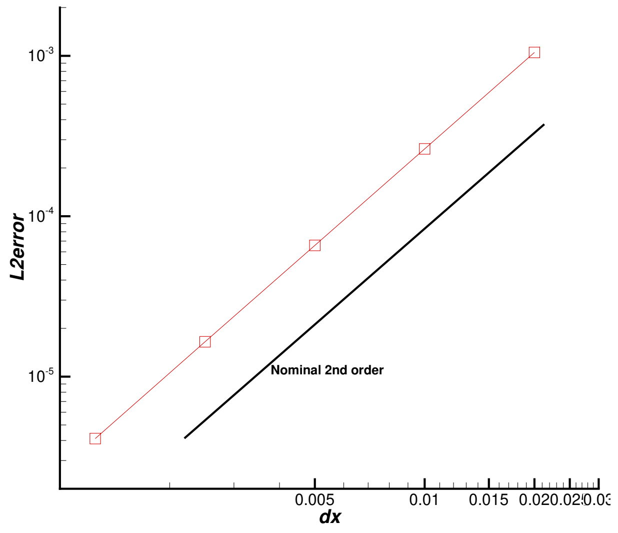


Figure 7: Plot of L2error for different grid size at $t = 0.01$.

6.3 Performance analysis

In this section, the execution time of different implementations is investigated to evaluate them. To perform these tests, the CPU and OpenCL backend have been used. The grid size is chosen as: $Nx = 101$ and $Ny = 101$.

At first, the execution time of the analytical solution is illustrated in Table. 3. As could be seen in Table. 3, the execution time decreased by 30 percent for CPU backend. Using the openCL backend for using the capabilities of GPU will results in more speed up. So, as shown in the table, a speedup of around 4 is obtained by using the GPU.

Table 3: Execution time of analytical solution. Time is in second.

Programming approach	**CPU**	**OpenCL**
Naive C++	3.4427	3.42741
ArrayFire	1.2402	0.859464

Table 4 shows the elapsed time to do the computation of numerical simulation. Here, we compare three different programming styles: Naive C++, vectorization by ArrayFire, and using the convolution function of ArrayFire. Also, the time used for numerical simulation is $t = 10s$. It could be concluded from the execution time for the openCL backend that the vectorization programming by ArrayFire, decreases the execution time. However, the speedup for the CPU backend is better than the OpenCL. The strange behavior of the convolution function of ArrayFire is the difference between the execution time of the CPU and OpenCL. While using OpenCL backend results in considerable speed up, for CPU backend it increases the execution time.

Table 4: Execution time of numerical solution. Time is in second.

Programming approach	**CPU**	**OpenCL**
Naive C++	295.058	295.087
ArrayFire/vectorization	34.3029	105.231
ArrayFire/Convolution	336.441	68.9458

6.4 Memory usage analysis

After investigating the execution time, the memory usage should be studied. To do that, the memory usage of different implementations of numerical simulation is compared. In this simulation, $Nx = Ny = 801$ has been used. As could be seen in Table 5 the memory consumption of the vectorization paradigm is about 4 times of naive C++. As expected the convolution is using less memory as it is an optimized implementation in the ArrayFire.

Table 5: Comparison of memory usage for CPU backend.

Programming approach	**Memory usage in MB**
Naive C++	5
ArrayFire/vectorization	19
ArrayFire/Convolution	10

7 Conclusion

In this project different implementations of the heat equation, both analytical and numerical, have been studied. The analytical solution is used for the verification of the numerical scheme. The vectorization approach by using ArrayFire compared to the naive C++ programming and also the convolution function of ArrayFire has been used.

References

[1] wikipedia.com, https://en.wikipedia.org/wiki/heat_equation.

[2] P. Yalamanchili, U. Arshad, Z. Mohammed, P. Garigipati, P. Entschev, B. Kloppenborg, J. Malcolm, J. Melonakos, ArrayFire - A high performance software library for parallel computing with an easy-to-use API (2015).
URL https://github.com/arrayfire/arrayfire

[3] Hipr2, https://homepages.inf.ed.ac.uk/rbf/hipr2/convolve.htm.