Polytechnique Montréal
Department of Mechanical Engineering
Course Number : **MEC6215**
Course Name : **Méthodes Numériques en Ingénierie**
Professor : **Sébastien Leclaire**
Homework : **Parabolic Equations (20%)**
Date of Delivery : **October 27th 2021 23:55**

Pedagogical Guidelines

- Submit a concise report of 10 pages maximum on Moodle (excluding first page, table of contents, and references), letter format, no narrow fonts, minimum 12pt, and margin 1.9cm;
- Submit your code that can reproduce your numerical results;
- This work must be done in a team of two people;
- All data (report and code) must be submitted in a single zip file "YOURMATRICULE1-YOURMATRICULE2.zip" (less than 100MB);
- Your report must be perfectly understandable when printed in black and white.
- Prepare a maximum of 6 "PowerPoint" slides (required for the final presentation).

Objective

- Learn the basics of programming solvers adapted to parabolic equations.

1 Two-dimensional simplified heat equation

Let's consider the two-dimensional simplified heat equation:

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

where $\alpha = 1$ is the thermal diffusivity. The boundary conditions around the square domain $(x, y) \in [0, L = 1] \times [0, H = 1]$ are:

$$u(x = 0, y, t) = 0 \tag{2}$$
$$u(x = 1, y, t) = 0 \tag{3}$$
$$u(x, y = 0, t) = 0 \tag{4}$$
$$u(x, y = 1, t) = 0 \tag{5}$$

For all positions strictly inside the square domain the initial condition is:

$$u(x, y, t = 0) = 1 \tag{6}$$

Exercises Use the MS Visual Studio solution **"MAIN_HEAT2D.sln"** already provided in the homework statement. Fill in the blanks within the main file **"MAIN_HEAT2D.cpp"** and functions to answer the following questions step-by-step.

Overall, one goal of the assignment is to write functions to solve numerically ($\Delta x = \Delta y$) this simplified heat 2D equation with standard first order forward and second order centered finite difference schemes. The stability condition of the numerical scheme is such that $\alpha \Delta t / \Delta x^2 \leq r$ with $r = 1/4$. Also, another goal is to perform calculations of the error between the numerical solution u_n and the analytical solution u_a:

$$u_a(x, y, t) = \sum_{n=1}^{N \equiv \infty} \sum_{m=1}^{M \equiv \infty} A_{mn} e^{-\alpha \lambda_{mn} t} \sin(n\pi x/L) \sin(m\pi y/H) \tag{7}$$

$$A_{mn} = \frac{\int_0^L \int_0^H u(x, y, 0) \sin(n\pi x/L) \sin(m\pi y/H) dx dy}{\int_0^L \int_0^H \sin(n\pi x/L)^2 \sin(m\pi y/H)^2 dx dy} \tag{8}$$

$$\lambda_{mn} = (n\pi/L)^2 + (m\pi/H)^2 \tag{9}$$

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

In the main program file (.cpp), the function "heat2D_analytical_naive_c" computes and returns the analytical solution at $t = 0.01$ using a naive C/C++ code (i.e. without using ArrayFire capability). This analytical solution is the reference. Also, at the end of the main section, the code provides automatic computation of the error between the analytical and numerical implementations using the L2 norm:

$$||e||_{l_2} = ||u_n - u_a||_{l_2} \equiv \left(\int_0^1 \int_0^1 ||u_n - u_a||^2 dx dy \right)^{\frac{1}{2}} \tag{10}$$

The code also compares the results between various implementations using the L1 norm. **Be careful**, for the L2 errors and L1 comparisons to be correct, you need to fill in the blanks correctly. In the functions that you will modify, always arrange the memory layout of the array return by the function to be the same, i.e. column-based, as the analytical array "uSol_analytical_naive_c". Please proceeds step-by-step as follows:

1. Implement a naive function which will use only C/C++ code with for loop to solve Eq. (1) using a first order forward in time and second order centered in space finite difference method. To do this task **complete the function "heat2D_numerical_naive_c"**. You may note that this function return a type *af::array*, make sure to convert your data (as similarly done in the "heat2D_analytical_naive_c" function) at the end of your function and return an ArrayFire array for post-processing, but do the main computation with naive procedural C/C++.

2. Implement a machine accurate equivalent function to "heat2D_numerical_naive_c" (already completed in step 1) but with vectorization technique using ArrayFire capabilities. To do this task **complete the function "heat2D_numerical_arrayfire"**.

3. Implement a machine accurate equivalent function to "heat2D_analytical_naive_c" with vectorization technique using ArrayFire capabilities in order to reduce significantly the computational time of the analytical solution for the larger grids. **Be careful**, do not overuse vectorization as it can slow down your code. It is recommend to use vectorization only to avoid looping over the computational nodes. To do this task **complete the function "heat2D_analytical_arrayfire"**.

Bonus +20% Implement a machine accurate equivalent function to "heat2D_numerical_naive_c" (already completed in step 1) using a filter image approach. You will need to think and structure your algorithm to use the *af::convolve2* function from ArrayFire. As a side note, convolution functions are very performant and optimized algorithm (if not best!) to compute explicit finite difference (filter) over regular grid (image). To do this task **complete the function "heat2D_numerical_arrayfire_convolution"**. The +20% bonus can be spread over other assignments if it results in a grade above 100% for the current assignment.

Complete the Excel file "heat2D.xls" to obtain the order of accuracy of the scheme. Results should be identical to machine accuracy whatever the implementations. Insert this Excel table into your report document. Compare and discuss your results against the analytical solution. Also, compare and discuss the performance and the accuracy of each of the implementations. In particular, if you have access to an OpenCL or CUDA backend, please use it.