INFO 0939: Project 2 – Due on 21 December 2018

Updated 04/12/2018

Intermediate deadline 1 (MPI explicit): 20 November 2018
Intermediate deadline 2 (MPI implicit): 4 December 2018

Goals of the Project

- Solve partial differential equations.
- Experiment stability of explicit and implicit time integration schemes.
- Combine MPI and OpenMP to take advantage of both distributed and shared memory parallelisation strategies.

Statement

Let's imagine an octopus spitting ink into the sea. How will the ink spread? To answer this question we ask you to implement a three-dimensional finite difference solver that solves the following advection-diffusion equation, where x, y, and z are the (cartesian) spatial coordinates, t represents time, $c = c(x, y, z, t)$ is the concentration of ink (g/m^3), D is the diffusion coefficient of the ink into the water (m^2/s), $\mathbf{v} = [v_x, v_y, v_z]^T$ is the water velocity (m/s) and R is a source (or sink) term (g/m^{3}s):

$$\frac{\partial c}{\partial t} = \underbrace{\nabla \cdot (D \nabla c)}_{\text{diffusion}} - \underbrace{\nabla \cdot (\mathbf{v} c)}_{\text{advection}} + \underbrace{R}_{\text{source}}. \tag{1}$$

The left hand side of this equation corresponds to the time variation of the concentration. The first term of the right hand side models the diffusion process, the second term models the transport (advection) process and the last is the source/sink term. Figure 1 shows a comparison between a pure diffusion process and a pure advection process.

The finite difference method

To write the well-posed problem corresponding to the phenomenon that we want to model, let's consider the following hypotheses:

- The diffusion coefficient can be assume constant in order that the first term can be written as: $\nabla \cdot (D \nabla c) = D \Delta c$;
- The velocity of water can be assume constant too: $\nabla \cdot (\mathbf{v} c) = \mathbf{v} \cdot \nabla c$;
- There are no sources or sinks: $R = 0$;

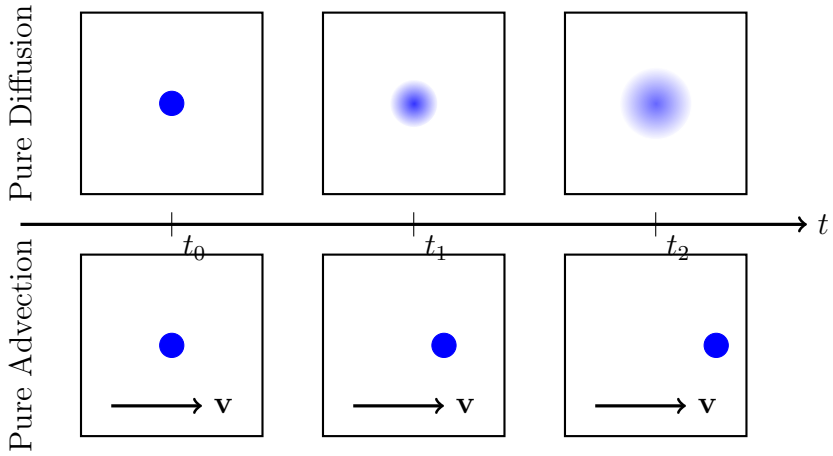

Figure 1: Comparison between a pure diffusion process and a pure advection process. The blue circle corresponds to the ink bubble.

- The space domain of the problem is the cube of size L: $[0, L]^3$;
- The time domain of the problem is: $[0, T_{max}]$;
- The initial concentration is assumed pointwise, with value $c_0 = 1\text{g/m}^3$, at the center of the domain (and zero anywhere else),
- The boundary concentration is zero.

These hypotheses allow to express the problem as:

$$\begin{cases} \dfrac{\partial c}{\partial t} = D\Delta c - \mathbf{v}\cdot\nabla c & \text{for } 0 < x, y, z < L \text{ and } t > 0, \\ c = c_0 & \text{for } x = y = z = L/2 \text{ and } t = 0, \\ c = 0 & \text{for } x \neq L/2, y \neq L/2, z \neq L/2 \text{ and } t = 0, \\ c = 0 & \text{for } x = 0, y = 0 \text{ or } z = 0, \text{ and } t > 0, \\ c = 0 & \text{for } x = L, y = L \text{ or } z = L, \text{ and } t > 0. \end{cases} \tag{2}$$

To solve this problem with finite differences, we discretize both space and time as follows:

$$x_i = ih, \qquad y_j = jh, \qquad z_k = kh, \qquad t_n = nm. \tag{3}$$

where h is the spatial step (m), m is the time step (s), i, j, k and n are positive integers defined into $\{0, 1, ..., L/h\}$ (for i, j, and k) or $\{0, 1, ..., T_{max}/m\}$ (for n). **You must chose L, $\mathbf{T_{max}}$, h, m in order that L/h, L/(2h) and $\mathbf{T_{max}}$/m remains an integer.** Because of the discretization of the domain, the unknown field becomes discrete as well: $c^n_{i,j,k} = c(x_i, y_j, z_k, t_n)$.

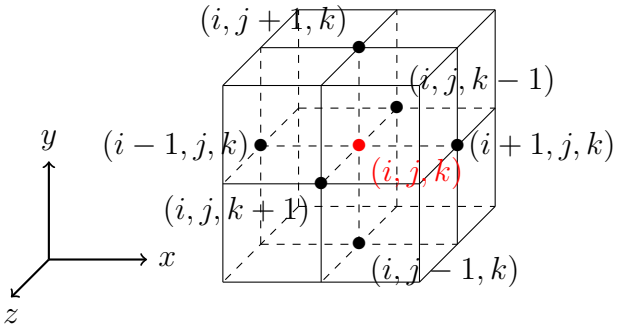

Figure 2: The finite difference grid.

Using Taylor's formula, the continuous equation of Problem 2 can be discretized[1]. This discretization applied to the temporal term leads to the so called Euler method. Depending on the chosen Euler method, the finite difference scheme can be explicit or implicit.

Explicit Euler scheme

The simplest Euler method is called explicit because the unknown values at the next time step $(n+1)$ depend explicitly of the known values at the current time step (n). The general form of the explicit Euler scheme can always be expressed as:

$$\frac{u^{n+1} - u^n}{m} = f(u^n), \tag{4}$$

where u is the unknown field.

Applying this method to Problem 2, the explicit finite difference scheme you should implement is the following:

$$\begin{aligned}\frac{c^{n+1}_{i,j,k} - c^n_{i,j,k}}{m} = D\frac{c^n_{i+1,j,k} + c^n_{i,j+1,k} + c^n_{i,j,k+1} - 6c^n_{i,j,k} + c^n_{i-1,j,k} + c^n_{i,j-1,k} + c^n_{i,j,k-1}}{h^2} \\ - v_x\frac{c^n_{i+1,j,k} - c^n_{i-1,j,k}}{2h} - v_y\frac{c^n_{i,j+1,k} - c^n_{i,j-1,k}}{2h} - v_z\frac{c^n_{i,j,k+1} - c^n_{i,j,k-1}}{2h},\end{aligned} \tag{5}$$

with the following initial condition

$$c^0_{i,j,k} = \begin{cases} 1 & i = j = k = \dfrac{L}{2h}, \\ 0 & \text{otherwise}, \end{cases} \tag{6}$$

and the following boundary condition:

$$c^n_{0,j,k} = c^n_{i,0,k} = c^n_{i,j,0} = c^n_{L/h,j,k} = c^n_{i,L/h,k} = c^n_{i,j,L/h} = 0, \quad n = 0, 1, \ldots, \frac{T_{max}}{m}. \tag{7}$$

[1]See slides `hpc5.pdf` and `hpc6.pdf`, as well as the lecture notes from the numerical analysis course (chapter 2): `http://www.montefiore.ulg.ac.be/~dgerard/fr/lectures/ananum/an2015-2016-en.pdf`

Implicit Euler scheme

For implicit Euler, the unknown values at the next time step $(n+1)$ depend of the known values at the current time step (n) but also of unknown values at $n+1$. This dependency implies that a linear system must be solved to compute values at time step $n+1$. The general form of the implicit Euler scheme can be expressed as:

$$\frac{u^{n+1}-u^n}{m} = f(u^{n+1}), \tag{8}$$

where u is the unknown field.

Applying this method on Problem 2, the implicit finite difference scheme you should implement is the following:

$$\begin{aligned}\frac{c_{i,j,k}^{n+1}-c_{i,j,k}^n}{m} = D\frac{c_{i+1,j,k}^{n+1}+c_{i,j+1,k}^{n+1}+c_{i,j,k+1}^{n+1}-6c_{i,j,k}^{n+1}+c_{i-1,j,k}^{n+1}+c_{i,j-1,k}^{n+1}+c_{i,j,k-1}^{n+1}}{h^2} \\ - v_x\frac{c_{i+1,j,k}^{n+1}-c_{i-1,j,k}^{n+1}}{2h} - v_y\frac{c_{i,j+1,k}^{n+1}-c_{i,j-1,k}^{n+1}}{2h} - v_z\frac{c_{i,j,k+1}^{n+1}-c_{i,j,k-1}^{n+1}}{2h},\end{aligned} \tag{9}$$

with the same initial and boundary condition as Equations 6 and 7.

Using this numerical scheme leads to the following linear system of equations:

$$Ac^{n+1} = c^n, \tag{10}$$

where A is a symmetric positive definite square matrix of size $\left(\frac{L}{h}\right)^3$, c^{n+1} and c^n are vectors that contain all respectively all unknowns and all previous unknowns. A is a sparse matrix. That means that most of the elements are zero. To store such matrices, a efficient method keeps in memory only the non-zero values. In this project, we suggest to store non-zero values using 3 arrays (i, j and v) of size N_z equal to the number of non-zero values in A, defined such as:

$$\forall k \quad (A)_{i(k),j(k)} = v(k). \tag{11}$$

The conjugate gradient method

In order to solve the linear system of equation, a iterative method must be used. The method you have to implement is called the conjugate gradient method, which is briefly summarized in Algorithm 1.

Instructions

By group of **two students** (this is mandatory) you are asked to implement a code that:

Algorithm 1: Conjugate gradient method.

$\mathbf{x}_0 = \mathbf{0}$
$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{p}_0 = \mathbf{r}_0$
i $= 0$
while $\frac{\|\mathbf{r}_i\|_2}{\|\mathbf{r}_0\|_2} \geq r_{threshold}$ **do**
  $\alpha = \frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{p}_i^T A \mathbf{p}_i}$,
  $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha \mathbf{p}_i$
  $\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha \mathbf{A}\mathbf{p}_i$
  $\beta = \frac{\mathbf{r}_{i+1}^T \mathbf{r}_{i+1}}{\mathbf{r}_i^T \mathbf{r}_i}$
  $\mathbf{p}_{i+1} = \mathbf{r}_{i+1} + \beta \mathbf{p}_i$
  i $=$ i $+ 1$
end
return $\mathbf{x}_i$.

1. Models the octopus problem using **double precision** representation;
2. Stops the simulation when the ink touch the boundary of the domain (when the concentration at any point of the boundary is greater or equal than 5.10^{-8} g/m^3);
3. Uses **both** the explicit and implicit scheme;
4. Solves the linear system (for the implicit scheme) with the conjugate gradient method;
5. Uses both MPI and OpenMP;
6. Takes from the command line a parameter file and an integer to chose the scheme used to solve the problem (0 for the explicit scheme and 1 for the implicit scheme). A call to your program could thus look like `./octopus param.dat 0` or `./octopus param.dat 1`. The parameter file (written in ASCII) has the following structure:

h(double)
m(double)
L(double)
T_{max}(double)
v_x(double)
v_y(double)
v_z(double)
D(double)
S(unsigned int)
$r_{threshold}$(double)

Your program should interpret the given values with the type specified between the parentheses. The parameters h, m, L, T_{max}, v_x, v_y, v_z, D are defined above; S is the sampling rate at which the results should be saved to disk ($S = 1$ corresponds to saving all time steps, $S = 2$ corresponds to saving on time step every two, ...) and $r_{threshold}$ is the residual threshold of the conjugate gradient method;

7. Saves the concentration field using the following **binary** output file format (one file per time step) where where N is the number of elements in the x-, y- and z-directions, written as an unsigned integer, and the $c_{i,j,k}$ are the concentrations written as double precision numbers:

N			
$c_{0,0,0}$	$c_{1,0,0}$	$\cdots$	$c_{L/h,0,0}$
$c_{0,1,0}$	$c_{1,1,0}$	$\cdots$	$c_{L/h,1,0}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$c_{0,L/h,0}$	$c_{1,L/h,0}$	$\cdots$	$c_{L/h,L/h,0}$
$c_{0,0,1}$	$c_{1,0,1}$	$\cdots$	$c_{L/h,0,1}$
$c_{0,1,1}$	$c_{1,1,1}$	$\cdots$	$c_{L/h,1,1}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$c_{0,L/h,L/h}$	$c_{1,L/h,L/h}$	$\cdots$	$c_{L/h,L/h,L/h}$

Submit your C code on the Montefiore submission platform `https://submit.montefiore.ulg.ac.be/`. Note that no error have to be reported during the automatic tests performed on this platform. If your last submission generates error(s), a default grade of 0/20 will be attributed.

Write a report of maximum 20 pages where you:

1. Describe your implementation;
2. Study the stability of the explicit numerical scheme for various combinations of parameters;
3. Perform a thorough scalability analysis (weak and strong) for both the explicit and implicit version of the code;
4. Compare the performance of the explicit and implicit methods for various combinations of parameters, in particular depending on the relative importance of diffusion versus advection.
5. Bonus (Advanced! Only do it if all the previous points have been fully treated!): derive the numerical scheme for a spatially- and time-dependent velocity field $\mathbf{v} = \mathbf{v}(x, y, z, t)$ and apply it to study the advection-diffusion of the ink bubble in rip currents or other physically more complex velocity fields (rotational, sinusoidally damped, etc.)

When your code passes on the submission platform, send your report by email to `anthony.royer@uliege.be` and in PDF format together with your C code and SLURM submission script. The files (report, C code, SLURM script) should be named

```
project2_Lastname1_Lastname2.pdf
project2_Lastname1_Lastname2*.c
project2_Lastname1_Lastname2*.h
project2_Lastname1_Lastname2.sh
```

Annexes

Post-processing

Matlab functions for post-processing data can be downloaded on Nic4:

- `/home/ulg/ace/aroyer/INFO0939/hw2/postpro/getSliceZ.m`
- `/home/ulg/ace/aroyer/INFO0939/hw2/postpro/movieSliceZ.m`
- `/home/ulg/ace/aroyer/INFO0939/hw2/postpro/viewSliceZ.m`

Set of working parameters

Parameters files can also be downloaded on Nic4. These files can help you to find parameters where the explicit Euler scheme is stable. They are sorted by execution time:

- `/home/ulg/ace/aroyer/INFO0939/hw2/parameters/param_tiny.dat`
- `/home/ulg/ace/aroyer/INFO0939/hw2/parameters/param_small.dat`
- `/home/ulg/ace/aroyer/INFO0939/hw2/parameters/param_medium.dat`
- `/home/ulg/ace/aroyer/INFO0939/hw2/parameters/param_big.dat`