INFO 0939: Project 2 – Due on 22 December 2020

Intermediate deadline 1 (MPI explicit code): 17 November 2020
Intermediate deadline 2 (MPI implicit code): 01 December 2020

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

In chemistry, the reaction-diffusion equation naturally appears to model the interaction of different compounds ("components") that diffuse and react together into a solvent. The equation can be written as follows for each component:

$$\frac{\partial c}{\partial t} = \underbrace{\nabla \cdot (D \nabla c)}_{\text{diffusion}} + \underbrace{R}_{\text{reaction}}, \tag{1}$$

where $c = c(t, \mathbf{x})$ is the concentration [mol/ℓ], D is the mass diffusivity [m^2/s] and $R = R(t, \mathbf{x}, c)$ is the reactive term [mol/ℓ.s] that can depend on time, position or the concentration of the component itself. Interesting behavior can be seen when two of more components are involved leading to a reaction-diffusion system. In some cases, the system can reach stable solutions and show pattern structures (see Figure 1). In biology, this behavior is one of the possible causes of the welts and spots observed on animals like zebras and tigers.

In this project, you are asked to study the Gray-Scott model, that involves two components A (with concentration a) and B (with concentration b) that follow the reaction mechanism schematized below in (2). The mechanism considers autocatalytic production of B, which decays to form an inert product P. Moreover, the reactor is crossed by a constant flow at the rate λ that brings reactant A at concentration a_0 and removal of the product P and some of the reactants A and B:

$$\begin{aligned} \emptyset &\xrightarrow{\lambda} A \\ A + 2B &\xrightarrow{k_1} 3B \\ B &\xrightarrow{k_2} P \\ B &\xrightarrow{\lambda} \emptyset \\ A &\xrightarrow{\lambda} \emptyset \end{aligned} \tag{2}$$

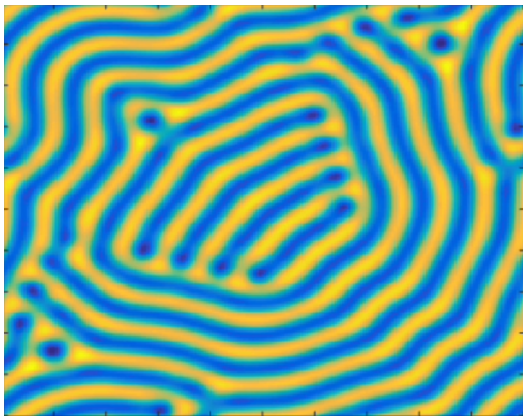
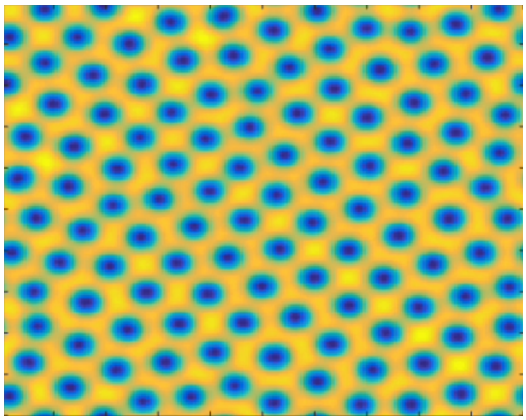

Figure 1: Two stable pattern structures that can be obtained in this project.

After nondimensionalisation of the kinetic equations corresponding to this model, one can find the following system written in term of two nondimensional unknowns $u = a/a_0$ and $v = b/a_0$:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \nabla \cdot (D_u \nabla u) - uv^2 + f(1-u) \\ \dfrac{\partial v}{\partial t} = \nabla \cdot (D_v \nabla v) + uv^2 - (f+k)v \end{cases} \tag{3}$$

where D_u is the mass diffusivity of u, D_v is the mass diffusivity of v, $f = \lambda/k_1 a_0^2$ is the inverse of the mean residence time in nondimensional time units and $k = k_2/k_1 a_0^2$ is the effective rate constant for formation of product P. These are clearly two reaction-diffusion equations, which are particular forms of (1).

The finite difference method

The system of equations (3) is discretized on the nodes of a square grid with spatial step Δx, and with time discretized using an explicit or a semi-implicit Euler scheme, with time step Δt. Periodic boundary conditions are imposed on the boundaries of the domain. (The two discretization methods and the boundary conditions are further explained in the sections below.)

The domain of study is a fixed square of unit dimension. At the beginning of the simulation the domain is filled with reactant u at constant and uniform unit nondimensional concentration, while the reactant v is initialized to one only inside a circle of radius 0.05 at the center of the square domain, and to zero elsewhere. The concentration is computed at the nodes (i, j) of the square grid described above, for every $i \in \{0, 1, 2, \cdots, \lfloor 1/\Delta x \rfloor\}$ and $j \in \{0, 1, 2, \cdots, \lfloor 1/\Delta x \rfloor\}$ and for every time iteration $n \in \{0, 1, 2, \cdots, \lfloor T_{\max}/\Delta t \ \rfloor\}$, such that $u_{i,j}^n$ stands for the nondimensional concentration u at position $(i\Delta x, j\Delta x)$ and at time $n\Delta t$. The same conventions are used for the nondimensional concentration v.

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 explicit finite difference scheme applied to (3) leads to the following system:

$$\begin{cases} \dfrac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = D_u \dfrac{u_{i+1,j}^{n} + u_{i,j+1}^{n} - 4u_{i,j}^{n} + u_{i-1,j}^{n} + u_{i,j-1}^{n}}{\Delta x^2} - u_{i,j}^{n} v_{i,j}^{n} v_{i,j}^{n} + f(1 - u_{i,j}^{n}), \\ \dfrac{v_{i,j}^{n+1} - v_{i,j}^{n}}{\Delta t} = D_v \dfrac{v_{i+1,j}^{n} + v_{i,j+1}^{n} - 4v_{i,j}^{n} + v_{i-1,j}^{n} + v_{i,j-1}^{n}}{\Delta x^2} + u_{i,j}^{n} v_{i,j}^{n} v_{i,j}^{n} - (f+k) v_{i,j}^{n}. \end{cases} \tag{4}$$

Semi-implicit Euler scheme

For the semi-implicit Euler method, the unknown values at the next time step, $n+1$, depend both on the known values at the current time step, n, but also of unknown values at step $n+1$. Therefore, unknown values can no longer be evaluated independently of each other, leading to the necessary solution of a linear system of equations at each time step. The semi-implicit finite difference scheme applied to (3) leads to the following system:

$$\begin{cases} \dfrac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = D_u \dfrac{u_{i+1,j}^{n+1} + u_{i,j+1}^{n+1} - 4u_{i,j}^{n+1} + u_{i-1,j}^{n+1} + u_{i,j-1}^{n+1}}{\Delta x^2} - u_{i,j}^{n} v_{i,j}^{n} v_{i,j}^{n} + f(1 - u_{i,j}^{n}), \\ \dfrac{v_{i,j}^{n+1} - v_{i,j}^{n}}{\Delta t} = D_v \dfrac{v_{i+1,j}^{n+1} + v_{i,j+1}^{n+1} - 4v_{i,j}^{n+1} + v_{i-1,j}^{n+1} + v_{i,j-1}^{n+1}}{\Delta x^2} + u_{i,j}^{n} v_{i,j}^{n} v_{i,j}^{n} - (f+k) v_{i,j}^{n}. \end{cases} \tag{5}$$

At each time step, a sparse linear system of equations must be solved, whose unknowns are the reactant nondimensional concentrations of u^{n+1} and v^{n+1}, stored together in a vector. To store the sparse matrix (named A), an efficient method keeps in memory only the non-zero values. While in this project it is actually *not* necessary to store the matrix, if you wish to do so we suggest that you store non-zero values (named a_k) using 3 arrays (ai, aj and a) of size N_z, equal to the number of non-zero values in A, defined such that:

$$\forall k \in \{0, 1, \cdots, N_z\} \quad (A)_{ai(k),aj(k)} = a(k) = a_k. \tag{6}$$

In order to solve the linear system of equations, we ask you to choose an iterative method (and to justify your choice). A good candidate could be the conjugate gradient method, which is briefly summarized in Algorithm 1.

Periodic boundary condition

The periodic boundary conditions imposed on the four borders of the square domain can be understood as if the flat square domain is folded onto itself so that the top border touches the bottom one leading to a cylinder. This cylinder is then folded again to join

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**
  $\alpha = \frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A}\mathbf{p}_i}$
  $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha \mathbf{p}_i$
  $\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha \mathbf{A}\mathbf{p}_i$
  $\beta = \frac{\mathbf{r}_{i+1}^T \mathbf{r}_{i+1}}{\mathbf{r}_i^T \mathbf{r}_i}$
  $\mathbf{p}_{i+1} = \mathbf{r}_{i+1} + \beta \mathbf{p}_i$
  i = i + 1
end
return $\mathbf{x}_i$.

the two ends, resulting in a torus. Therefore the top border becomes linked to the bottom one and the right border becomes linked to the left one.

From a practical point of view, that means that when you implement (4) and (5) you should pay attention when you evaluate an unknown on the borders, when some i or j indices go out of the possible range (-1 or $\lfloor 1/\Delta x \rfloor + 1$). When this happens, you should replace indices equal to -1 by $\lfloor 1/\Delta x \rfloor$, and indices equal to $\lfloor 1/\Delta x \rfloor + 1$ by 0, as shown in Figure 2.

Instructions

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

1. Models the Gray-Scott reaction diffusion system using **double precision** representation;
2. Uses the explicit or the semi-implicit scheme;
3. Solves the linear system (for the semi-implicit scheme);
4. Uses both MPI and OpenMP;
5. Takes from the command line a parameter file and a flag used to chose the numerical scheme (0 for explicit and 1 for semi-implicit). A call to your program could thus look like `./prog param.txt 0` or `./prog param.txt 1`.

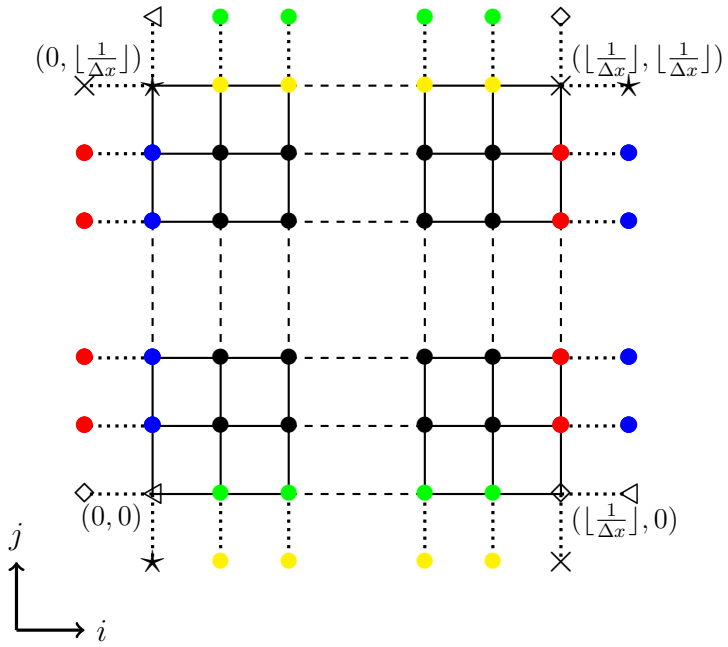

Figure 2: The computation grid with the periodic boundary condition.

The parameter file is written in ASCII and has the following structure:

Δt(double)
Δx(double)
T_{max}(double)
D_u(double)
D_v(double)
f(double)
k(double)
$r_{\text{threshold}}$(double)
S(unsigned int)

Your program should interpret the given values with the type specified between parentheses. The parameters Δt, Δx, T_{max}, D_u, D_v, f and k are defined above; $r_{\text{threshold}}$ is the residual threshold for the Conjugate Gradient algorithm; S is the sampling rate at which the results should be saved to disk ($S = 0$ stands for never saving the results, $S = 1$ corresponds to saving all time steps, $S = 2$ corresponds to saving one time step out of two, etc.).

6. Saves the solutions using the following **binary** output file format (two files per time step, `u.dat` and `v.dat`) where N is the number of values in the x and y directions, respectively, written as a 32 bit unsigned integer, and the $\alpha_{i,j}$ are either the u concentration or the v concentration written as double precision numbers. Note that

because the file is written in binary mode, no newline or space characters should be used in the file.

N			
$\alpha_{0,0}$	$\alpha_{1,0}$	$\cdots$	$\alpha_{N-1,0}$
$\alpha_{0,1}$	$\alpha_{1,1}$	$\cdots$	$\alpha_{N-1,1}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\alpha_{0,N-1}$	$\alpha_{1,N-1}$	$\cdots$	$\alpha_{N-1,N-1}$

Submit your C code on the Montefiore submission platform `https://submit.montefiore.ulg.ac.be/`. Note that no errors 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 30 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 semi-implicit version of the code;
4. Compare the performance of the explicit and semi-implicit methods for various combinations of parameters.
5. Present some parameters (D_u, D_v, f and k) that give interesting or funny pattern structures.

When your code passes on the submission platform, send your report by email to `anthony.royer@uliege.be` 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
```

("*" means that you can have multiple header or source files.)

Appendix

- Matlab scripts are provided to help with the visualization of your results (see `/home/ulg/ace/aroyer/info0939/hw2` on the Lemaitre3 cluster).

- Parameter files to generate the patterns of Figure 1 can be found at `/home/ulg/ace/aroyer/info0939/hw2` on the Lemaitre3 cluster as well.
- The diffusion coefficients have to be different to observe patterns. A good ratio could be $\frac{D_u}{D_v} = 2$.