Open Journal of Fluid Dynamics, 2023, 13, 32-46
https://www.scirp.org/journal/ojfd
ISSN Online: 2165-3860
ISSN Print: 2165-3852

Shared Memory Semi-Implicit Solver for Hydrodynamical Instability Processes

Augusto Kielbowicz[1], Diego Fernández[1], Adriana Saal[1], Claudio El Hasi[1,2], Carlos Vigh[1,3,4*]

[1]Instituto de Ciencias, Universidad Nacional de General Sarmiento, Los Polvorines, Argentina
[2]Departamento de Ciencias Básicas, Facultad de Ingeniería, Universidad Argentina de la Empresa, Buenos Aires, Argentina
[3]Departamento de Física, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina
[4]Instituto de Física Interdisciplinaria Aplicada (CONICET-UBA), Buenos Aires, Argentina
Email: *cvigh@campus.ungs.edu.ar

How to cite this paper: Kielbowicz, A., Fernández, D., Saal, A., El Hasi, C. and Vigh, C. (2023) Shared Memory Semi-Implicit Solver for Hydrodynamical Instability Processes. *Open Journal of Fluid Dynamics*, **13**, 32-46.
https://doi.org/10.4236/ojfd.2023.131003

Received: November 20, 2022
Accepted: March 21, 2023
Published: March 24, 2023

Abstract

The Advection-Diffusion Reaction (ADR) equation appears in many problems in nature. This constitutes a general model that is useful in various scenarios, from porous media to atmospheric processes. Particularly, it is used at the interface between two fluids where different types of instabilities due to surface mobility may appear. Together with the ADR equation, the Darcy-Brinkman model describes the phenomena known as fingering that appear in different contexts. The study of this type of system gains in complexity when the number of chemical species dissolved in both fluids increases. With more solutes, the increasing complexity of this phenomenon generally requires much computational power. To face the need for more computational resources, we build a solver tool based on an Alternating Direction Implicit (ADI) scheme that can be run in Central Processing Unit (CPU) and Graphic Processing Unit (GPU) architectures on any notebook. The implementation is done using the MATLAB platform to compare both versions. It is shown that using the GPU version strongly saves both resources and calculation times.

Keywords

Fingering, Fluids, Simulations, Numerical Solver, Hele-Shaw Cell

1. Introduction

The numerical modeling of a natural phenomenon usually demands intensive computational work. The dynamics of the interaction between two fluids is a typical example that requires the use of several specific equations such as the Advection-Diffusion Reaction (ADR) equation or the Darcy-Brinkman (DB) equation.

This constitutes a general model useful in several physical phenomena, from porous media to atmospheric processes, where several variables may be involved and solving strategies are generally highly demanding in terms of computational cost. Although we will restrict ourselves to natural phenomena, there are other disciplines, like economy in the social science, which can be addressed like complex systems where behaviors highly sensitive to perturbations can be modeled mathematically to describe various economic processes [1]. Furthermore, sometimes to describe properly a system or to have a better resolution of the numerical results, the grid refinement increases the computational requirements.

When solving the ADR equation to simulate miscible fluid displacement, a variety of methods have been applied. They include finite differences, pseudo-spectral methods, and modified methods of characteristics, among others. Also, finite element methods have been applied (see [2] and references therein).

In field problems or situations with several species interacting, the increasing complexity of the problem makes it necessary to appeal to optimization criteria that save computational resources and reduce the time of the simulations. Clearly, the optimization strategies are dependent on hardware and software. Usually, sequential codes are used, but they are not adequate for parallelization. However, parallelization and high-performance computing resources have become fundamental strategies to improve numerical performance (see [3] and references therein). Notwithstanding this new paradigm, the procedure is not straightforward. In shared-memory bus-based multiprocessors, when the number of processors grows, the processors spend an increasing amount of time waiting for access to the bus (and shared memory). This contention reduces the performance of processors and imposes a limitation on the number of processors that can be used efficiently in bus-based systems. For example, [4] is made a detailed study concerning to the relationship between the behavior of shared-memory bus-based multiprocessors system and its time performance.

Several examples appear in the literature where the ADR equation was applied not only to solve environmental processes, but also most daily life issues [5]. Meanwhile, the use of Graphics Processing Unit (GPU) has deserved much attention lately. For example, in [6], the processing time could be improved by a factor of 70 using a General-Purpose Graphics Processing Unit (GPGPU) code considering heterogeneous and anisotropic media. In [7], a GPU algorithm was applied to hydrogeologic processes showing different benchmarks. Su *et al.* [8] presented the parallelization of a well-known code (MIN3P-THCm) widely used for the simulation of reactive transport modelling to assess long-term geochemical processes. Li *et al.* [9] proposed parallelization codes to study hydrological problems in Basin Rivers. Other applications of GPU are possible, for example, to speed up Fast Fourier Transformation for imaging processing [10].

Here, we use a high-level environment such as MATLAB [11] which has many friendly tools available for profiling and visualization that also includes a battery of NVIDIA GPU resources that allow us to implement parallelizing criteria. Our main purpose is to have a low-cost code dedicated to modeling fingering phenomena experimentally conducted in a Hele-Shaw cell to analyze several processes. To validate the outputs, we compare them with previous experimental and numerical works. In order to validate the accuracy of our algorithm, we will compare our results with previous experimental and numerical works [12] [13] [14]. To verify its efficiency, we will compare the computing time of several simulations.

In the next section, we present the physical context. In Section 3, we present the ADI method to solve this kind of problem and the essential characteristics of our algorithm and discuss optimization and parallelization strategies. In Section 4, we perform quality tests of rendering and benchmarking. Finally, we present our conclusions.

2. Physical and Mathematical Context

The phenomenon of interfacial deformation between two fluids appears in several branches of science and technology. Its dynamics have been extensively studied experimentally, theoretically, and numerically (see earlier works of Whitaker [15] [16], Turner *et al.* [17] and Homsy [18]). The mentioned deformations appear like fingers that invade the phase of lower mobility, causing the velocity of advance of one fluid onto the other to be strongly increased. The instabilities can be caused by density difference (Rayleigh-Taylor, a denser fluid on top of a less dense one [19]), viscosity difference (Saffman-Taylor, a less viscous fluid displacing a more viscous one), difference in diffusion coefficients, or by concentration difference of the same solute in both fluids and double diffusion [20]. For example, the case of Turner instability is found in oceans when two water masses of different salt concentrations and at different temperatures come into contact [21].

Lately, particular attention has been paid to the case of bimolecular reactions of the form $A + B \rightarrow C$ [22] [23] and to the dissolution of CO_2 (g) in aqueous solutions (see also [24] and references therein). Rayleigh-Taylor (RT), Double Diffusion (DD), and/or Dynamic Layer Convection type instabilities appear in all these systems. The case of the liquid-liquid interfaces has deserved detailed studies, both numerical and experimental. It has been shown that generally in, a priori, stable systems; the occurrence of hydrodynamics instabilities is regulated by the existence of chemical reactions [25]. In situations that could give rise to both RT and DD processes, the existence of chemical reactions causes important changes in the fingering pattern [22]. CO_2 dissolution has also been extensively analyzed in the context of its geological sequestration.

Due to the difficulties to make a realistic experiment, to analyze and characterize which mechanisms are at the origin of these instabilities, a typical laboratory experimental array is a Hele-Shaw cell as shown in **Figure 1**, which consists of two plates separated by a thin space. The cell has a width L_y, a highness L_x and thickness e, with $e \ll L_y$, L_x (see more details in [23]).

To study the variety of processes that could be considered, we make an example with a bimolecular reaction in a Hele-Shaw cell. In this way, we consider a bidimensional stationary, laminar, incompressible, and isothermal flow. As the problem is bidimensional, we can express the field velocity in terms of the stream function ψ ($v_x = \partial_y \psi$ and $v_y = -\partial_x \psi$).

For the case of a reactive system like A + B → C, after some mathematical steps; the set of dynamical equations which describes the problem can be written in its dimensionless form as [24]:

$$\partial_t a + (v \cdot \nabla) a = \delta_a \nabla^2 a - ab \tag{1}$$

$$\partial_t b + (v \cdot \nabla) b = \delta_b \nabla^2 b - ab \tag{2}$$

$$\partial_t c + (v \cdot \nabla) c = \delta_b \nabla^2 c + ab \tag{3}$$

$$\omega - \nabla^2 \omega = -\partial_y \Gamma(a,b,c) \tag{4}$$

$$\Gamma(a,b,c) = R_a a + R_b b + R_c c \tag{5}$$

The first three equations correspond to the transport equations for two reactives (a and b) and the product (c), the fourth describes the motion of the fluid (the Brinkmann equation), in terms of the dimensionless curl of the velocity $\omega = \nabla \times V = \nabla^2 \psi$ [24] and the last one is the density (Γ) of the fluid. v is the velocity and δ is the dispersion coefficient, while R is the corresponding Rayleigh number associated with buoyancy driven flows.

3. Numerical Strategy and Implementation

To solve the kind of problems described by the equations stated above, there are different strategies, but usually the most frequent are pseudo-spectral codes [25] using Fast Fourier Transform (FFT) libraries and semi-implicit finite difference schemes [26] [27].

In pseudo-spectral codes, it is necessary to impose periodic boundary conditions requiring to duplicate the computational domain, as shown in **Figure 2(A)**, and

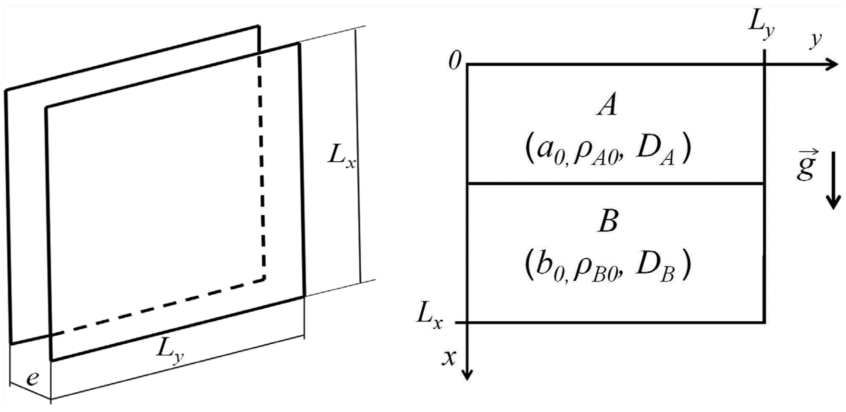

Figure 1. Left: Hele-Shaw cell of dimensions L_x, L_y and thickness e ($e \ll L_y$, L_x). Right: Initial setting with species A and B.

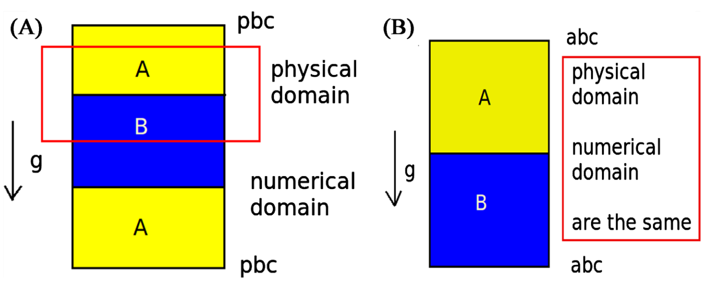

Figure 2. Modelling of two species into a Hele-Shaw cell in presence of gravity. (A) Initial setting for FFT implementation which needs periodic boundary conditions (pbc). (B) Initial setting for ADI implementation with arbitrary boundary conditions (abc).

increasing memory resources. Meanwhile, in semi-implicit finite difference schemes the physical and numerical domains are the same, and the boundary conditions are imposed more straightforwardly (see Figure 2(B)). In what follows, we will use the semi-implicit method and apply optimization techniques.

3.1. Optimization

In general, for specific scientific applications, numerical codes are written to solve a particular problem. This means that probably there is no place for efficiency and speeding up in the first step to fulfill this objective. Parallel computing is the way to follow. However, before implementing a parallel code, it is mandatory to assure an optimal serial version of it. After that, it is necessary to choose the proper parallelization strategy.

3.2. Vectorization

To reduce the number of computations, in MATLAB is easy to optimize through vectorization of the set of variables in the system of Equations (1)-(5). Some advantages of implementing this strategy are concise and simple programming and reduction of processing time.

3.3. Shared and Distributed Memory

In a *shared memory* architecture, a number of independent processors share a single memory and they can directly access any data location. In distributed-memory, messages are used to coordinate data transmission between processors. The time needed to exchange data between processors is in favor of shared memory computers which communicate faster than distributed memory computers. CPU and GPU paradigms intelligently combine the best features of both to achieve even further computational gains. Both are increasingly being seen as indispensable coprocessors, instead of substitutes for each other. We exploit the parallelisms in computation via the NVIDIA CUDA programming model.

3.4. ADI Scheme

Semi-implicit codes are conditional stable, and the stability is easy to manage numerically. We use the *Alternating Direction Implicit* (ADI) method, this method computes the nearest neighbors in alternate directions in one intermediate step (see stencil scheme in **Figure 3**). For more details, see [27] and references therein. The use of this scheme has the following advantages; the boundary conditions are more realistic and allow using around 90% of the integration domain as shown in **Figure 2(B)**.

3.5. Algorithm

We implement our model, Equations (1)-(5), in a MATLAB environment. For the problem, we are facing on the boundary conditions used are:

$$\boldsymbol{v}\cdot\breve{\boldsymbol{n}}=\mathbf{0}\quad \text{(No flux condition)} \tag{6}$$

And Wood's conditions on the vorticity [28]:

$$\omega\big|_x=\frac{-3\psi\big|_{x+\Delta x}}{(\Delta x)^2}-\frac{1}{2}\omega\big|_{x+\Delta x}\quad \text{en}\quad x=0 \tag{7}$$

$$\omega\big|_x=\frac{-3\psi\big|_{x-\Delta x}}{(\Delta x)^2}-\frac{1}{2}\omega\big|_{Lx-\Delta x}\quad \text{en}\quad x=L_x \tag{8}$$

$$\omega\big|_y=\frac{-3\psi\big|_{y+\Delta y}}{(\Delta y)^2}-\frac{1}{2}\omega\big|_{y+\Delta y}\quad \text{en}\quad y=0 \tag{9}$$

$$\omega\big|_y=\frac{-3\psi\big|_{Ly-\Delta y}}{(\Delta y)^2}-\frac{1}{2}\omega\big|_{Ly-\Delta y}\quad \text{en}\quad y=L_y \tag{10}$$

The algorithm follows the diagram flux shown in **Figure 4**, the modules presented are:

- *Initialization*: Initialize all variables and fields.
- *Inputs*: We introduce the input values of the numerical experiment, which includes concentrations values, diffusivities, kind of reactions and Rayleigh numbers, initial velocities.

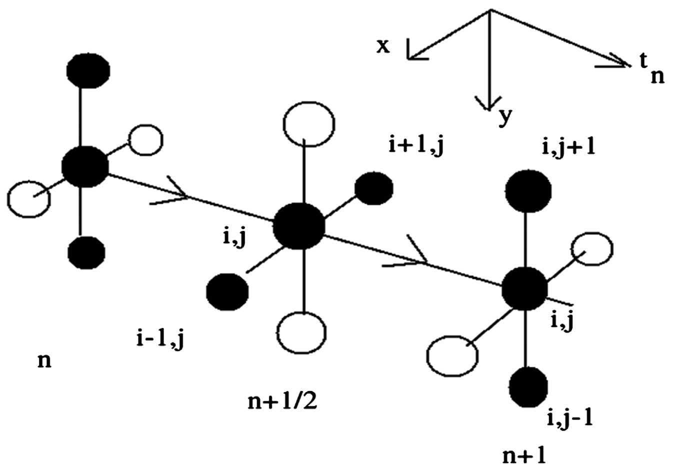

Figure 3. ADI stencil graphic.

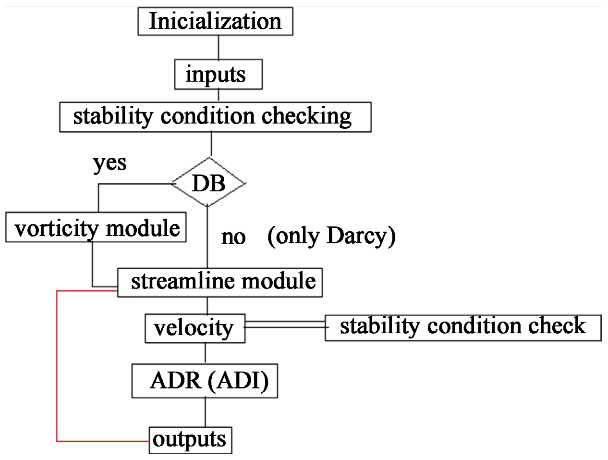

Figure 4. Flux diagram to solve the Advection-Diffusion Reaction (ADR) system.

- *Stability condition checking*: In each iteration, step is necessary to assure that numerical information speed is lower than physical information speed:

$$\left(\frac{v_x \Delta t}{\Delta x}\right)^2 < 2\frac{\delta_i \Delta t}{\Delta x^2} < 1 \tag{11}$$

$$\left(\frac{v_y \Delta t}{\Delta y}\right)^2 < 2\frac{\delta_i \Delta t}{\Delta y^2} < 1 \tag{12}$$

- *DB if statement*: the key point of the algorithm is to decide if this is a Darcy-Brinkman problem or not.

  If *so,* the code activates the:
- V*orticity module*: The code computes the vorticity including the diffusion term, If *not,* it goes directly to the following module:
- *Streamline module.* This module computes the streamline functions and its boundary conditions. After that, begins the loop at the desirable final time.
- *Velocity*: In each step, the velocity field is computed using direct forward time iteration and checking stability again.
- *ADR (ADI) module*: Computes the chemical reactive-diffusion-advection evolution. Here, we implement the ADI scheme and solve the tridiagonal system that appears in this kind of resolution [26].
- *Outputs*: Finally, the outputs are saved.

Essentially, this flux diagram is for the serial version but the parallel one has the same spirit.

4. Numerical Experiments

In the previous sections, we exposed the main ideas to develop two versions of a code used to simulate the onset of instabilities in some physical situations, a CPU sequential vectorized version and a GPU-CUDA parallelized version. From here on, we are going to probe the goodness of them to reproduce the experimental results. On the other hand, we will see that the codes are versatile enough to manage an increasing number of variables. Meanwhile, the GPU version is more efficient to process information.

4.1. Case A → A: Physical Behavior

The first example to study is a numerical experiment considering just one species. That corresponds to gaseous CO_2 absorption in a liquid medium (A → A), as it is the case of CO_2 sequestration. Figure 5 shows the concentration map comparison at the same final time simulation for the CPU and GPU versions, which is consistent with the real and numerical experiments made in [12] and [13]. The similarity between the previous and present results is remarkable. In Figure 6, we compare each point of the grid for the results of both versions of the code, with the same and different initial seeds to see consistency of the outputs of the system. For different initial seeds, the solutions are qualitatively identical into a range of 10% relative to the total concentration. Based on these results, we can

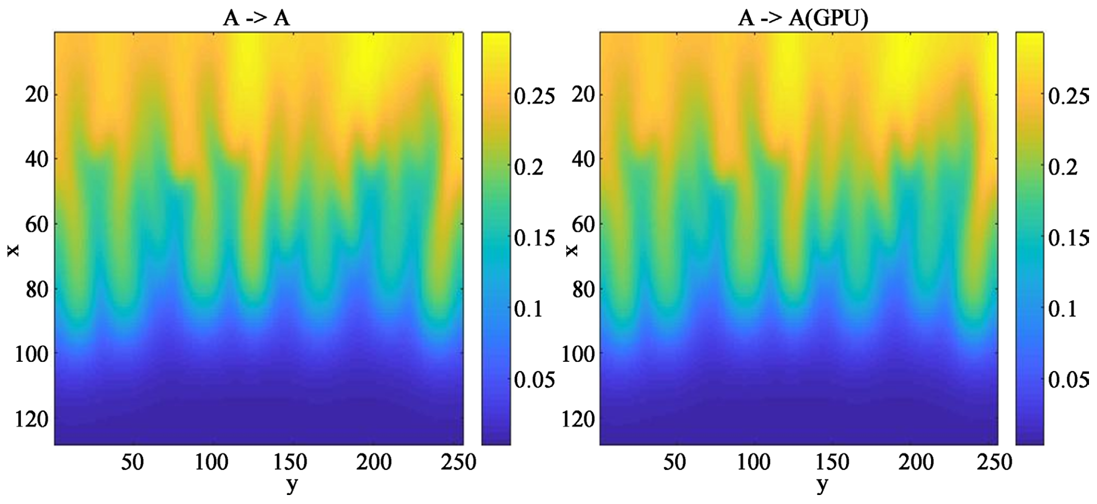

Figure 5. Concentration map of a system A(gaseous) → A(aqueous) at $t = 150$ (a.u), spatial resolution 128 × 256, $\Delta x = 0.5$, $\Delta y = 0.5$, $\Delta t = 0.75$. Left: CPU version. Right: GPU version.

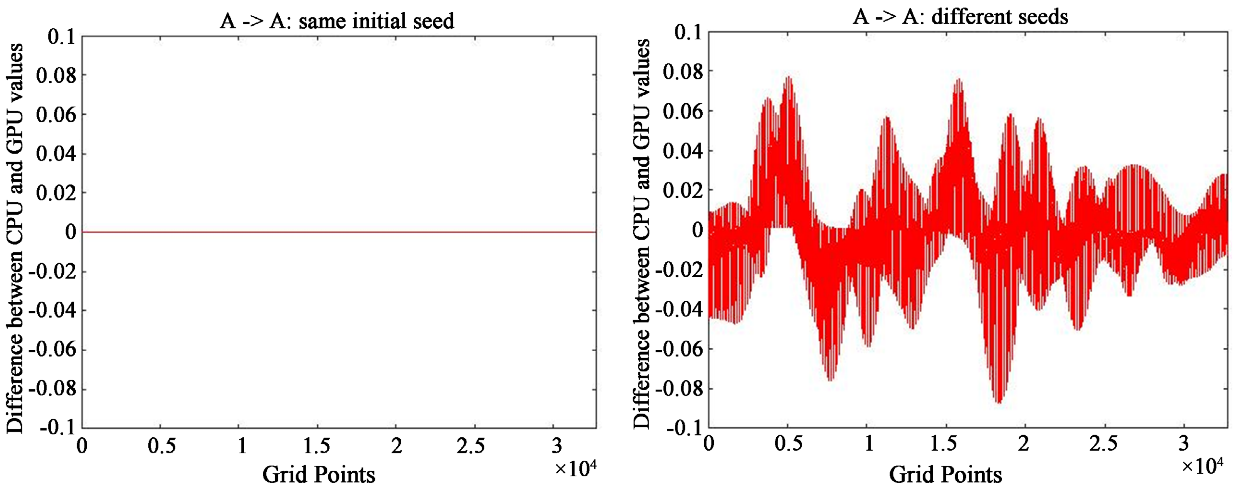

Figure 6. Concentration difference between CPU and GPU outputs for each species. Left: CPU seed = 500, GPU seed = 500. Right: CPU seed = 500, GPU seed = 400. Horizontal scale from grid points 0 to 32,767. Vertical scale is the absolute difference in concentration value between simulations.

conclude that both codes correctly reproduce the experiments.

4.2. Case A → A: Benchmarking

We also consider the speed-up of the GPU version compared to the CPU version. Figure 7 shows that a very low-resolution GPU is not convenient due to the time needed for initialization variables requires around 40 seconds. For higher resolutions, the optimized version is clearly faster than the CPU version.

4.3. Case A + B → C: Physical Behavior

Here, we apply both codes to a more complex situation a bimolecular reaction, *i.e.*, two species A and B that react and give a product C. In Figure 8, we show

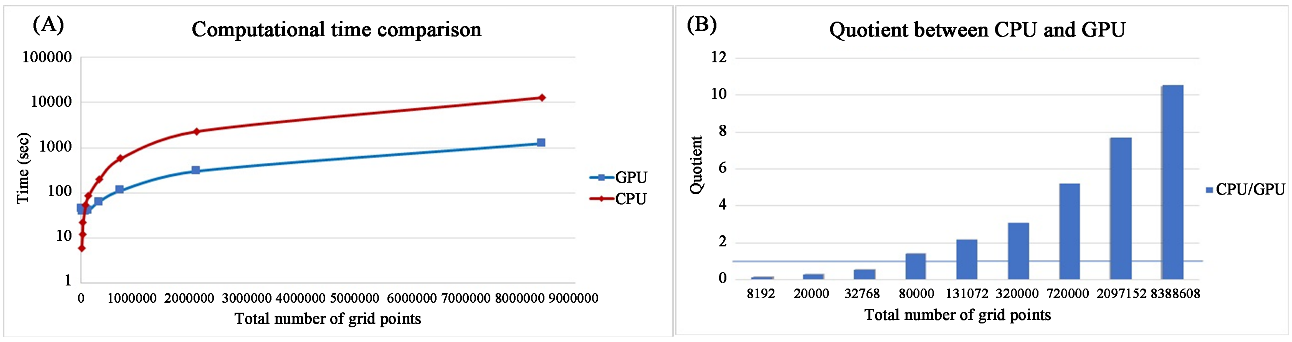

Figure 7. (A) Comparison time between GPU and GPU versions as a function of the number of grid points; (B) Relative time quotient between GPU and CPU versions.

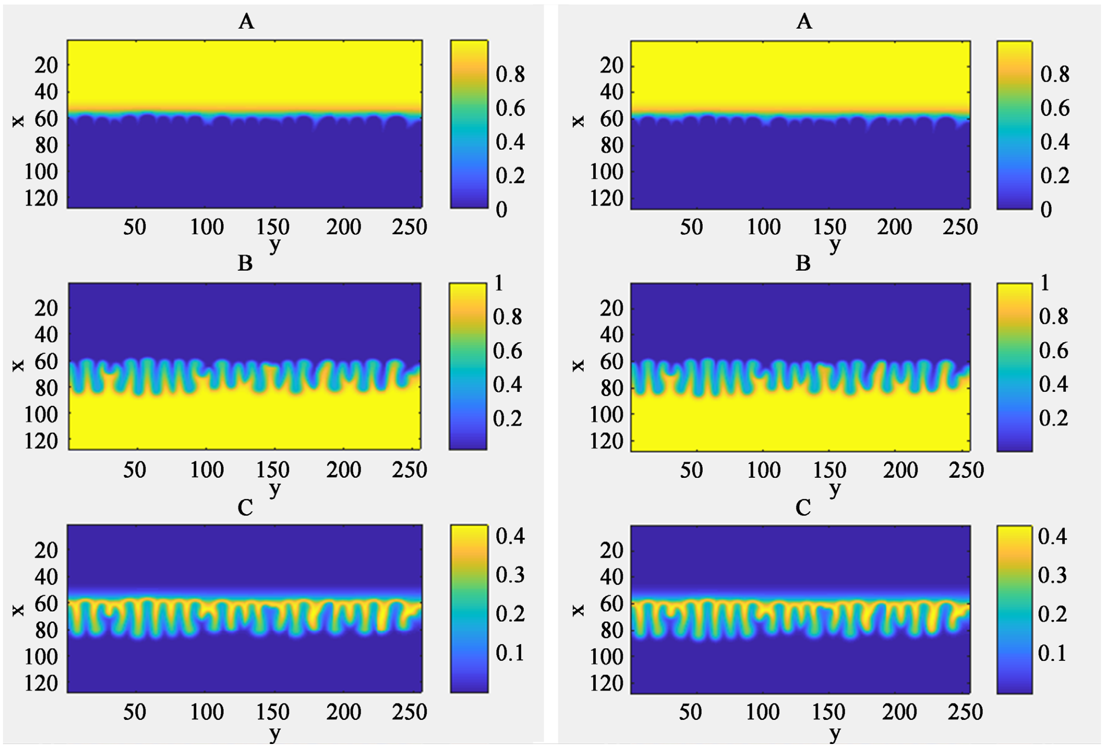

Figure 8. Concentrations A, B and C for t =75 (a.u.), Δt = 0.5 and resolution 128 × 256. CPU version (left) and GPU version (right) using the same noise seed.

the fingering evolution at a given time of the CPU (left) and GPU (right) versions. Both numerical results are quite like the real experiments [29].

To verify the accuracy between versions, we computed, for the simulations corresponding to **Figure 8**, the difference for each grid point comparing the concentrations A, B and C in both versions. In **Figure 9(A)**, we showed that if we use the same initial noise seed to generate the evolution, we obtain the same output. In **Figures 9(B)-(D)**, we used different initial noise seeds in CPU and GPU versions. As seen in the figures the concentration difference for each case is of the order of 1% - 5% with respect to the absolute concentration values for each reactive species. The difference for concentration C in this scale is not appreciable.

4.4. Case A + B→ C: Benchmarking

To get some idea of the comparative performance of both versions of the code, we make several experiments ranging from 50 × 50 to 900 × 900 grid points for the same final time, the results are shown in **Figure 10(A)**. It is clearly seen, on

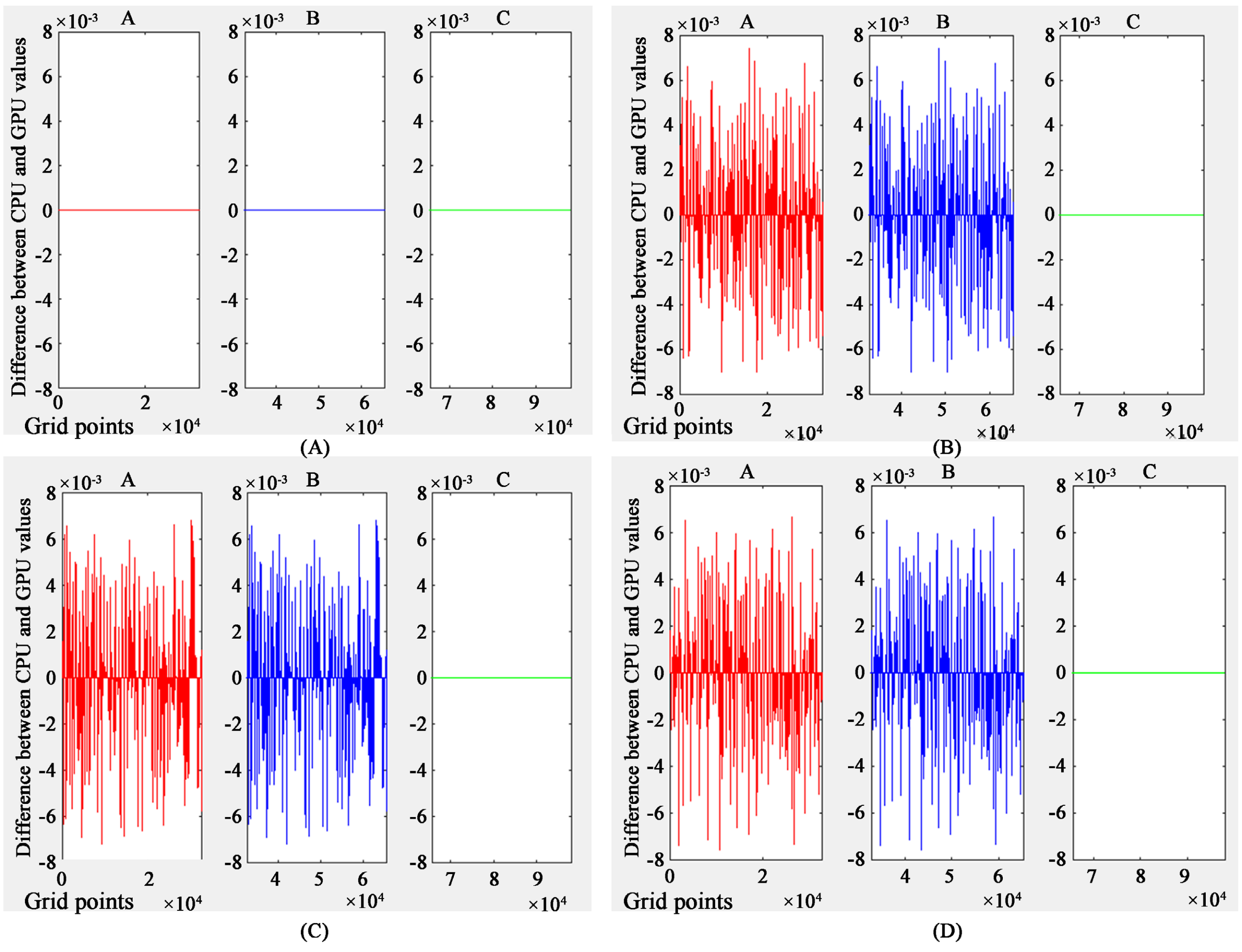

Figure 9. Concentration difference between CPU and GPU outputs for each species. Concentration A: red, Concentration B: blue and Concentration C: green. Horizontal scale from 0 to 32,767 (A) grid points, 32,768 to 65,535 (B) and 65,536 to 98,303 (C). Vertical scale is the absolute difference value between simulations in the range −0.008 - 0.008. (A) cpu_seed = 100, gpu_seed = 100; (B) cpu_seed = 4, gpu_seed = 100; (C) cpu_seed = 99, gpu_seed = 100; (D) cpu_seed = 999, gpu_seed = 100.

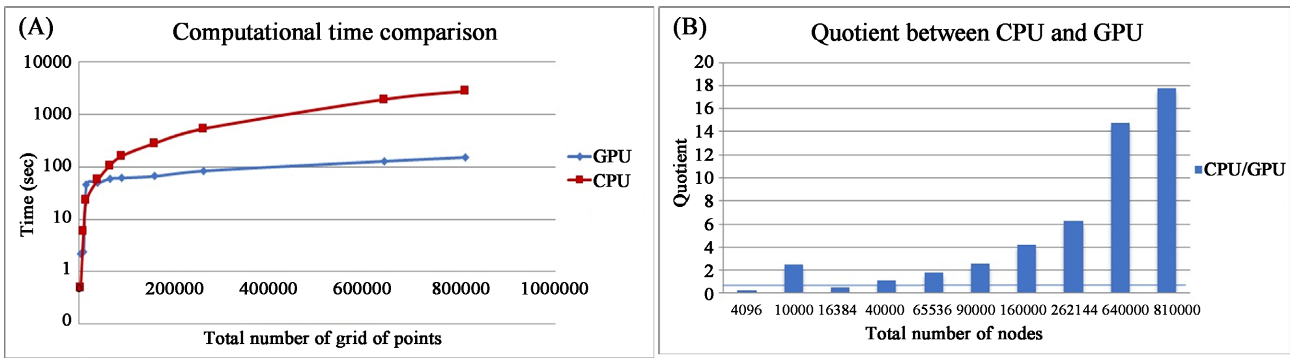

Figure 10. (A) Comparison time between GPU and GPU versions as a function of the number of grid; (B) Relative time quotient between GPU and CPU versions.

a logarithmic scale, that the GPU strategy runs more than an order of magnitude faster than the CPU one. In Figure 10(B), we show the computational time ratio of CPU over GPU. As seen the GPU version is faster than CPU version after around 40,000 grid points. At 900,000 grid points, the ratio of computational time is around 18.

Both figures show that the setting and initialization of the code require extra time for the GPU version. This time is relevant for smaller grids, but for numerical domains of the order of 100 × 100 points the computational time cost is similar. Then, for greater grids, the time consumption for the simulations strongly grows in the case of the CPU version with respect to the GPU version. It means that for low resolution experiments, it is not strictly necessary to use parallelization, but is mandatory when the resolution requirements imply many grid points.

4.5. Case A + B → C + D

The codes presented here can easily be adapted to problems with an increasing number of variables. For instance, the analysis of a bimolecular reaction is sometimes performed using color indicators to visualize how the reaction proceeds. Recently, it was discovered that the color indicators are not passive species, but they play a role in the development of the instabilities. In this way, the number of involved species must be increased. To compare the performance of both codes when the number of variables becomes greater, we study a bimolecular reaction including a color indicator (A + B → C + D) [29]. Also, in this case, the results of the numerical simulations using CPU and GPU versions of the code are like the real experiments. In Figure 11, we present the map of the density and total concentration. Allow us to observe that numerically, we can study each species separately to get a better understanding of the physical subjacent processes. Figure 12 shows how each reactant and product contributes to the behavior of the whole system.

5. Conclusions

We present an optimized solver for problems related to the advection-diffusion

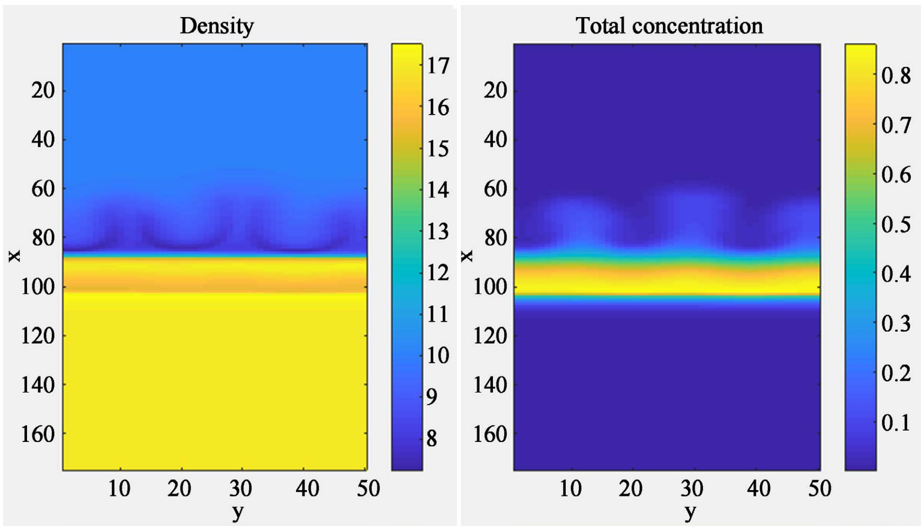

Figure 11. System A + B → C + D at time $t = 210$ in arbitrary time units, $\Delta t = 0.001$ for a resolution grid 175 × 50. Initial contrast $a/b = 12$ [14]. Left: Density map. Right: Total concentration field.

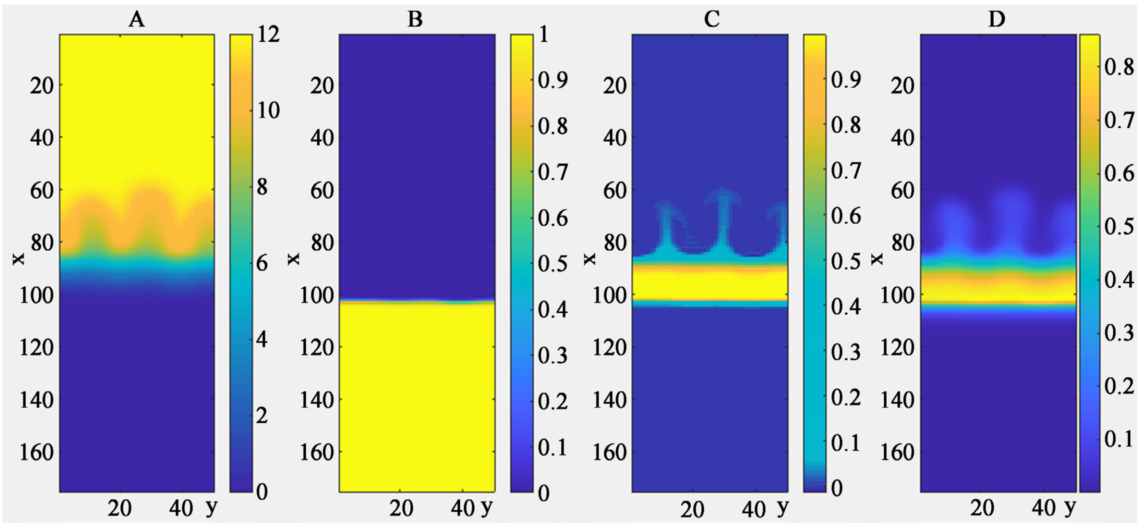

Figure 12. Concentration fields for each specie for the same experiment.

reaction equation. We developed two versions using a semi-implicit scheme. The codes are implemented in MATLAB using vectorized architectures for CPU and GPU-CUDA. With these tools, we can analyze the study of how to solve fingering phenomena using ADR models. To stress the main ideas presented here, we can say that:

1) The fingering problem is explicitly solved using both sequential vectorize and GPU versions of the code.

2) The numerical results of the presented cases agree with experimental results and other numerical results shown in previous works.

3) For low-resolution simulations, the performance of both codes is similar, but for grids of a larger number of points, the use of GPU is more suitable. It is seen that as the number of grid points increases, the GPU version becomes more

efficient.

4) This solver code could be implemented in an arbitrary number of variables; this will allow us to study phenomena including temperature, for example.

In summary, we get a code that can be useful even in an environment where not much hardware power is available.

Acknowledgements

The authors are grateful for the financial support of UNGS.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Perevoznikov, E. and Lomteva, E. (2019) Modeling of Economic Processes, Instability and Chaos. *Journal of Applied Mathematics and Physics*, **7**, 356-363. https://doi.org/10.4236/jamp.2019.72027

[2] Sesini, P.A., de Souza, D.A. and Coutinho, A.L. (2010) Finite Element Simulation of Viscous Fingering in Miscible Displacements at High Mobility-Ratios. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, **32**, 292-299. https://doi.org/10.1590/S1678-58782010000300013

[3] Narang, H., Wu, F. and Mohammed, A.R. (2019) An Efficient Acceleration of Solving Heat and Mass Transfer Equations with the First Kind Boundary Conditions in Capillary Porous Radially Composite Cylinder Using Programmable Graphics Hardware. *Journal of Computer and Communications*, **7**, 267-281. https://doi.org/10.4236/jcc.2019.77022

[4] Zuberek, W.M. (2018) Timed Petri Net Models of Shared-Memory Bus-Based Multiprocessors. *Journal of Computer and Communications*, **6**, 1-14. https://doi.org/10.4236/jcc.2018.610001

[5] Egidi, N., Giacomini, J., Maponi, P., Perticarini, A., Cognigni, L. and Fioretti, L. (2022) An Advection-Diffusion-Reaction Model for Coffee Percolation. *Computational and Applied Mathematics*, **41**, Article No. 229. https://doi.org/10.1007/s40314-022-01929-9

[6] Carlotto, T., da Silva, R.V. and Grzybowski, J.M.V. (2019) GPGPU-Accelerated Environmental Modelling Based on the 2D Advection-Reaction-Diffusion Equation. *Environmental Modelling & Software*, **116**, 87-99. https://doi.org/10.1016/j.envsoft.2019.02.001

[7] Le, P.V., Kumar, P., Valocchi, A.J. and Dang, H.V. (2015) GPU-Based High- Performance Computing for Integrated Surface-Sub-Surface Flow Modeling. *Environmental Modelling & Software*, **73**, 1-13. https://doi.org/10.1016/j.envsoft.2015.07.015

[8] Su, D., Mayer, K.U. and MacQuarrie, K.T.B. (2017) Parallelization of MIN3P-THCm: A High Performance Computational Framework for Subsurface Flow and Reactive Transport Simulation. *Environmental Modelling & Software*, **95**, 271-289. https://doi.org/10.1016/j.envsoft.2017.06.008

[9] Li, T., Wang, G., Chen, J. and Wang, H. (2011) Dynamic Parallelization of Hydrological Model Simulations. *Environmental Modelling & Software*, **26**, 1736-1746.

https://doi.org/10.1016/j.envsoft.2011.07.015

[10] Haque, M.N. and Uddin, M.S. (2011) Accelerating Fast Fourier Transformation for Image Processing Using Graphics Processing Unit. *International Conference on Signal Processing, Image Processing, and Pattern Recognition*, Jeju Island, 8-10 December 2011, 300-309.https://doi.org/10.1007/978-3-642-27183-0_32

[11] MATLAB, 9.2.0.538062. (R2017a) The MathWorks Inc., Natick, MA.

[12] Fernandez, D., Binda, L., Zalts, A., El Hasi, C. and D'Onofrio, A. (2018) Lateral Movements in Rayleigh-Taylor Instabilities Due to Frontiers. Numerical Analysis. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, **28**, Article ID: 013108. https://doi.org/10.1063/1.4995396

[13] Binda, L., Fernandez, D., El Hasi, C., Zalts, A. and D'Onofrio, A. (2018) Lateral Movements in Rayleigh-Taylor Instabilities Due to Frontiers. Experimental Study. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, **28**, Article ID: 013107. https://doi.org/10.1063/1.4995395

[14] Kuster, S., Riolfo, L.A., Zalts, A., El Hasi, C., Almarcha, C., Trevelyan, P.M.J. and D'Onofrio, A. (2011) Differential Diffusion Effects on Buoyancy-Driven Instabilities of Acid-Base Fronts: The Case of a Color Indicator. *Physical Chemistry Chemical Physics*, **13**, 17295-17303. https://doi.org/10.1039/c1cp21185d

[15] Whitaker, N. (1990) Numerical Solution of the Hele-Shaw Equations. *Journal of Computational Physics*, **90**, 176-199. https://doi.org/10.1016/0021-9991(90)90202-C http://www.sciencedirect.com/science/article/pii/002199919090202C

[16] Whitaker, N. (1994) Some Numerical Methods for the Hele-Shaw Equations. *Journal of Computational Physics*, **111**, 81-88. https://doi.org/10.1006/jcph.1994.1046

[17] Turner, J. (1974) Double-Diffusive Phenomena. *Annual Review of Fluid Mechanics*, **6**, 37-54. https://doi.org/10.1146/annurev.fl.06.010174.000345

[18] Homsy, G.M. (1987) Viscous Fingering in Porous Media. *Annual Review of Fluid Mechanics*, **19**, 271-311. https://doi.org/10.1146/annurev.fl.19.010187.001415

[19] Sharp, D.H. (1984) An Overview of Rayleigh-Taylor Instability. *Physica D: Nonlinear Phenomena*, **12**, 3-18. https://doi.org/10.1016/0167-2789(84)90510-4

[20] Lemaigre, L., Budroni, M.A., Riolfo, L.A., Grosfils, P. and De Wit, A. (2013) Asymmetric Rayleigh-Taylor and Double-Diffusive Fingers in Reactive Systems. *Physics of Fluids*, **25**, Article ID: 014103. https://doi.org/10.1063/1.4774321

[21] You, Y. (2002) A Global Ocean Climatological Atlas of the Turner Angle: Implications for Double-Diffusion and Water-Mass Structure. *Deep Sea Research Part I: Oceanographic Research Papers*, **49**, 2075-2093. https://doi.org/10.1016/S0967-0637(02)00099-7

[22] Trevelyan, P.M.J., Almarcha, C. and De Wit, A. (2015) Buoyancy-Driven Instabilities Around Miscible $A + B \rightarrow C$ Reaction Fronts: A General Classification. *Physical Review E*, **91**, Article ID: 023001. https://doi.org/10.1103/PhysRevE.91.023001

[23] Zalts, A., El Hasi, C., Rubio, D., Urena, A. and D'Onofrio, A. (2008) Pattern Formation Driven by an Acid-Base Neutralization Reaction in Aqueous Media in a Gravitational Field. *Physical Review E*, **77**, Article ID: 015304. https://doi.org/10.1103/PhysRevE.77.015304

[24] Tan, C.T. and Homsy, G.M. (1986) Stability of Miscible Displacements in Porous Media: Rectilinear Flow. *The Physics of Fluids*, **29**, Article No. 3548. https://doi.org/10.1063/1.865832

[25] Mangiavacchi, N., Coutinho, A.L.G.A. and Ebecken, N.F.F. (1997) Parallel Pseudo-Spectral Simulations of Nonlinear Viscous Fingering in Mis-Cible Displacements.

WIT Transactions on the Built Environment, **32**, 498-506.

[26] Tan, C.T. and Homsy, G.M. (1988) Simulation of Nonlinear Viscous Fingering in Miscible Displacement. *The Physics of Fluids*, **31**, 1330-1338. https://doi.org/10.1063/1.866726

[27] Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. (1993) Numerical Recipes in Fortran 77: The Art of Scientific Computing. 2nd Edition, Cambridge University Press, Cambridge, MA.

[28] Woods, L.C. (1954) A Note on the Numerical Solution of Fourth Order Differential Equations. *Aeronautical Quarterly*, **5**, 176-184. https://doi.org/10.1017/S0001925900001177

[29] Kim, M.C. and Cardoso, S.S. (2019) Diffusivity Ratio Effect on the Onset of the Buoyancy-Driven Instability of an A + B → C Chemical Reaction System in a Hele-Shaw Cell: Numerical Simulations and Comparison with Experiments. *Physics of Fluids*, **31**, Article ID: 084101. https://doi.org/10.1063/1.5094913