LMECA2300: Project report

Antoine Quiriny
2761 16 00

Gilles Poncelet
1063 16 00

***Abstract*—A numerical study of the Cahn-Hilliard equation. In this work, we study the Cahn-Hilliard equation with constant and variable mobility. Its spatial discretisation is achieved by means of Fourier spectral methods and we inspect its stability through the eigenvalues of the obtained ODE. This lets us identify the main difficulty of the equation: the ODE is stiff and imposes a $\mathcal{O}(n^4)$ constraint on the time step. To avoid this problem, we had to separate the stiff and linear part of the problem from its non-stiff and non-linear part. This results in implicit-explicit (IMEX) and exponential time differencing (ETDRK4) schemes. These different solutions remove any stability constraint on the Δt and leave only a constraint for the accuracy. Finally we worked on the efficiency of our implementations where the parallel aspect of the problem has been fully exploited thanks to an implementation on GPU using CUDA.**

I. Introduction

Initially developed for the study of the spinodal decomposition of binary alloy, the Cahn-Hilliard equation has subsequently been used in a wide range of fields. For example, it can be used to model Diblock copolymers by adding a long-range non-local energy. It is also used in image painting or even to simulate the evolution of a tumor [1].

It is therefore not surprising that many scientific papers focus on the resolution of this equation. We will exploit this scientific literature to obtain the most stable implementation of this equation that is difficult to solve numerically. The optimization of the code will also be the subject of our research in order to obtain the fastest simulation possible.

Section II quickly summarises the origin of the mathematical model of the Cahn-Hilliard equation. Then, we will discuss the spatial discretisation in Section III and the various time stepping schemes in Section IV. Finally, we will go over the details of our different implementations in Section V and present our results in Section VI.

II. The Cahn-Hilliard equation

The Cahn-Hilliard equation is a mathematical model proposed by John W. Cahn and John E. Hilliard in 1958 [2] to describe the process of phase separation in a binary alloy. In this case, a single field $c(\mathbf{x}, t)$ whose value ranges between ± 1 (both extrema indicating a single-phase domain) is enough to describe the microstructure. Its time evolution is governed by the following diffusion equation:

$$\frac{\partial c}{\partial t} = N_{\mathrm{V}} \nabla \cdot [M(c)\, \nabla \mu] \tag{1}$$

where $N_{\mathbf{V}}$ is the number of atoms per unit volume, $M(c)$ is the mobility and $\mu = \mu_2 - \mu_1$ is the chemical potential difference between the two species. Cahn and Hilliard proposed the following model for the total interfacial free energy:

$$F = \int_{\mathrm{V}} \left[f(c) + \frac{1}{2}\kappa (\nabla c)^2 \right] \mathrm{d}V \tag{2}$$

where $f(c)$ is the local free energy density (chosen here as the double-well potential function $f(c) = (c^2-1)^2/4$) and κ is a positive constant. From this, as the chemical potential is thermodynamically defined as the partial derivative of the total free energy w.r.t. to the quantity of moles in a body at constant temperature and pressure, we find the following constitutive model for μ:

$$N_{\mathrm{V}} \mu = \frac{\delta F}{\delta c} = f'(c) - \kappa \nabla^2 c \tag{3}$$

where $\delta F / \delta c$ is the functional derivative of F. Finally, considering a constant mobility $M(c) = M$, we obtain the Cahn-Hilliard equation in a scaled form (scaled variables are assimilated to non-scaled variables for clarity):

$$\frac{\partial c}{\partial t} = \nabla^2 \left(c^3 - c - \kappa \nabla^2 c \right) \tag{4}$$

We will also consider the Cahn-Hilliard equation with non-constant mobility. Several modelisation choices are possible, but we will use $M(c) = 1 - c^2$, as in [3]:

$$\frac{\partial c}{\partial t} = \nabla \cdot \left[(1-c^2)\, \nabla \left(c^3 - c - \kappa \nabla^2 c \right) \right] \tag{5}$$

III. Spatial discretisation

We shall solve the Cahn-Hilliard equation on a $[0,1] \times [0,1]$ square discretised by a $n \times n$ spatial grid with periodic boundary conditions and initial random state. As in [3], we solve the equation in Fourier space, taking advantage of the exponential convergence of spectral differentiation methods [4]. Taking the Fourier transform of (4) yields the following system of ODEs:

$$\frac{\partial \hat{c}_{\mathbf{k}}}{\partial t} = -k^2 \left\{ c^3 - c \right\}_{\mathbf{k}} - \kappa k^4 \hat{c}_{\mathbf{k}} \tag{6}$$

where $\mathbf{k} = (k_x, k_y)$ is the Fourier wavelength vector and $k = \sqrt{k_x^2 + k_y^2}$ its magnitude. $\hat{c}_{\mathbf{k}}$ and $\left\{ c^3 - c \right\}_{\mathbf{k}}$ denote the Fourier transform of c and $c^3 - c$, respectively.

To evaluate the stability of (6), we can compute the jacobian of the right-hand term by differentiating w.r.t. $\hat{c}_{\mathbf{k}}$:

$$\frac{\partial}{\partial \hat{c}_{\mathbf{k}}} \left(\frac{\partial \hat{c}_{\mathbf{k}}}{\partial t} \right) = -k^2 \left(\frac{\partial}{\partial \hat{c}_{\mathbf{k}}} \left\{ c^3 \right\}_{\mathbf{k}} - 1 \right) - \kappa k^4 \tag{7}$$

The evaluation of the derivative of the non-linear term is tricky and involves convolutions, which is beyond the scope of this work. However, it is possible to compute the eigenvalues of the linear term :

$$\lambda_{\mathrm{L}} = -\kappa k^4 \tag{8}$$

Supposing that the eigenvalues of the linear term largely dominate those of the non-linear term (which is verified in practice) and knowing that $k_{\max} = n\pi\sqrt{2}$, we obtain the biggest eigenvalue that will rule the stability of the ODE:

$$\lambda_{\mathrm{L,max}} = -4\kappa(n\pi)^4 \tag{9}$$

As the eigenvalues $\lambda\Delta t$ need to be enclosed within the stability region of any time stepping scheme, we have a stiff ODE with a $\mathcal{O}(n^4)$ constraint on the time step Δt.

A. Variable mobility

Taking the Fourier transform of the variable mobility Cahn-Hilliard equation (5) yields the following system of ODEs:

$$\frac{\partial \hat{c}_{\mathbf{k}}}{\partial t} = i\mathbf{k} \cdot \left\{ (1-c^2) \left[i\mathbf{k}' \left(\left\{ c^3 - c \right\}_{\mathbf{k}} + \kappa k^2 \hat{c}_{\mathbf{k}} \right) \right]_{\mathbf{r}} \right\}_{\mathbf{k}} \tag{10}$$

where $[.]_{\mathbf{r}}$ denotes the inverse Fourier transform. While its jacobian is more tricky to compute, it still has an $\mathcal{O}(n^4)$ constraint on the timestep.

B. Convergence of spectral methods

A well-known advantage of spectral methods is that they exhibit the fastest convergence possible, the so-called exponential convergence. However, it the discretised function needs to be "smooth enough" for it to work. In the ideal settings, few points are required to obtain accurate results, thus potentially enabling one to use few points and greatly alleviating the constraint on the time step Δt.

We attempted to confirm this smoothness condition was satisfied for the Cahn-Hilliard equation. To do so, we considered a centered square initialisation at 1 and the rest of the domain at -1 and solved it up to $t = 10^{-3}$ for different values of n. The error was computed by interpolating the values onto the bigger reference grid at $n = 4096$ using zero-padding. The resulting relative error $||c - c_{\mathrm{ref}}||_2 / ||c_{\mathrm{ref}}||_2$ is shown on Fig. 1. Clearly, we do not observe exponential convergence, but we do have some sort of super linear convergence.

Those results do not allow us to conclude anything as the initialisation considered here is quite different from the random initialisation. However, seeing that the interface between two phases becomes sharper as the simulation moves forward in time, it is not unreasonable to suppose that the solution of the Cahn-Hilliard equation is not "smooth enough" to exhibit exponential convergence.

IV. Time stepping

The Cahn-Hilliard equation belongs to the class of PDEs that can be expressed as the sum of a high-order (stiff) linear term and a low-order (non- or midly stiff) non-linear term:

$$\frac{\partial u}{\partial t} = \mathcal{L}u + \mathcal{N}(u,t) \tag{11}$$

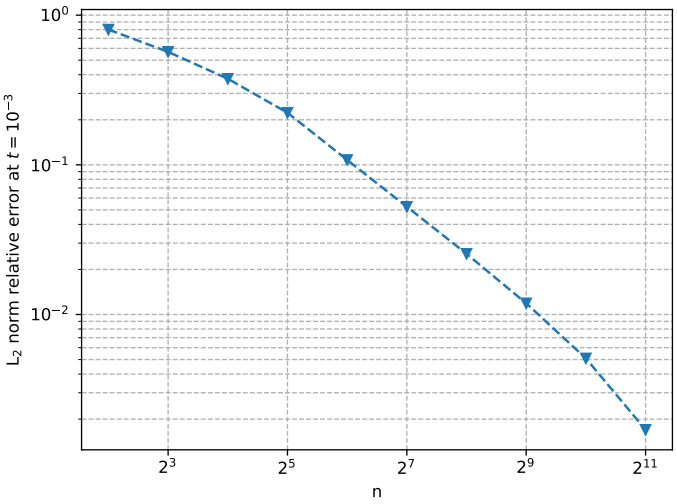


Fig. 1: Convergence results of the spatial discretisation for ETDRK4 with $\Delta t = 10^{-7}$. Error is estimated using a reference solution computed for $n = 4096$.

TABLE I: Coefficients for the IMEX-BDF schemes [6], [7].

Order	a_0	a_1	a_2	a_3	$\hat{b}_0$	$\hat{b}_1$	$\hat{b}_2$	$\hat{b}_3$	b_0
2	$\frac{4}{3}$	$-\frac{1}{3}$	0	0	$\frac{4}{3}$	$-\frac{2}{3}$	0	0	$\frac{2}{3}$
4	$\frac{48}{25}$	$-\frac{36}{25}$	$\frac{16}{25}$	$-\frac{3}{25}$	$\frac{48}{25}$	$-\frac{72}{25}$	$\frac{48}{25}$	$-\frac{12}{25}$	$\frac{12}{25}$

where $\mathcal{L}$ and $\mathcal{N}$ are linear and non-linear operators, respectively. Once the spatial part of the PDE is discretised, we obtain the following system of ODEs:

$$\frac{\partial u}{\partial t} = \mathbf{L}u + \mathbf{N}(u,t) \tag{12}$$

Specific methods [5] can be used to solve such system to overcome the stiff constraint on the timestep due to the fourth order linear term in the Cahn-Hilliard equation while maintaining high-order accuracy.

A. Implicit-explicit "IMEX" schemes

The main idea behind implicit-explicit schemes is to solve the stiff linear part of the ODE using an implicit scheme and solve the non-linear term using an explicit scheme. The general formula for linear multistep IMEX schemes is given by:

$$u_{n+1} = \sum_{j=0}^{k-1} a_j u_{n-j} + \sum_{j=0}^{k-1} \hat{b}_j \Delta t\, \mathbf{N}_{n-j} + \sum_{j=-1}^{k-1} b_j \Delta t\, \mathbf{L} u_{n-j} \tag{13}$$

where the coefficients a_j, $\hat{b}_j$ and b_j can be found in [6] for any order up to 5. In this work, we use the second and fourth order BDF-based IMEX schemes whose coefficients are given in Table I.

B. Exponential time differencing

Exact integration of (12) from t_n to t_{n+1} can be obtained as:

$$u_{n+1} = e^{\mathbf{L}\Delta t} u_n + e^{\mathbf{L}\Delta t} \int_0^{\Delta t} e^{-\mathbf{L}\tau} \mathbf{N}\left[u(t_n + \tau), t_n + \tau\right] \mathrm{d}\tau \tag{14}$$

where the linear term is perfectly integrated, greatly reducing the timestep constraint. Cox and Matthews [8] derived a set of methods to approximate the integral of the non-linear term, one of them using a fourth order Runge-Kutta scheme, called ETDRK4:

$$\begin{aligned}
a_n &= e^{\mathbf{L}\Delta t/2} u_n + \delta\, \mathbf{N}(u_n, t_n) \\
b_n &= e^{\mathbf{L}\Delta t/2} u_n + \delta\, \mathbf{N}(a_n, t_n + \Delta t/2) \\
c_n &= e^{\mathbf{L}\Delta t/2} a_n + \delta\left[2\mathbf{N}(b_n, t_n + \Delta t/2) - \mathbf{N}(u_n, t_n)\right] \\
u_{n+1} &= e^{\mathbf{L}\Delta t} u_n + \alpha\, \mathbf{N}(u_n, t_n) + \beta\big\{\mathbf{N}(a_n, t_n + \Delta t/2) \\
&\quad + \mathbf{N}(b_n, t_n + \Delta t/2)\big\} + \gamma\, \mathbf{N}(c_n, t_n + \Delta t)
\end{aligned} \tag{15}$$

where:

$$\begin{aligned}
\alpha &= \Delta t^{-2}\mathbf{L}^{-3}\left[-4 - \mathbf{L}\Delta t + e^{\mathbf{L}\Delta t}\left(4 - 3\mathbf{L}\Delta t + (\mathbf{L}\Delta t)^2\right)\right] \\
\beta &= \Delta t^{-2}\mathbf{L}^{-3}\left[2 + \mathbf{L}\Delta t + e^{\mathbf{L}\Delta t}\left(-2 + \mathbf{L}\Delta t\right)\right] \\
\gamma &= \Delta t^{-2}\mathbf{L}^{-3}\left[-4 - 3\mathbf{L}\Delta t - (\mathbf{L}\Delta t)^2 + e^{\mathbf{L}\Delta t}(4 - \mathbf{L}\Delta t)\right] \\
\delta &= \quad \mathbf{L}^{-1}\left[e^{\mathbf{L}\Delta t/2} - \mathbf{I}\right]
\end{aligned} \tag{16}$$

However, a naive implementation of the ETDRK4 scheme suffers from numerical instabilities due to floating point round-off errors when evaluating (16), as shown by Kassam and Trefethen in [5]. They also proposed to evaluate those expressions using a complex integral of a contour Γ containing the eigenvalues of $\mathbf{L}$:

$$f(\mathbf{L}) = \frac{1}{2\pi i} \oint_\Gamma f(t)\, \left[t\mathbf{I} - \mathbf{L}\right]^{-1} \mathrm{d}t \tag{17}$$

When $\mathbf{L}$ is diagonal, the evaluation of (17) to full accuracy reduces to taking the 32-points mean of $f(t)$ over a circle centered on each element of $\mathbf{L}$. This also simplifies the computations for the ETDRK4 scheme (15), such as the matrix exponential $e^{\mathbf{L}\Delta t}$.

C. Time step constraint

Thourough stability analysis of both the IMEX and ETDRK4 schemes is carried in [7], [9], [10]. The conclusion of interest here is that the stability region extends indefinitely along the real axis. The constraint (9) vanishes and only the constraint due to the eigenvalues of the non-linear term remains.

In practice, we observed that constraint to be of $\mathcal{O}(1)$! Indeed, we managed to obtain a stable simulation for $\Delta t = 10^{-4}$ for values of n up to 4096 using the ETDRK4 scheme. This means that the constraint on the time step Δt can be determined solely based on a given precision requirement instead of any stability concern.

In practice, the presented schemes are all stable for $\Delta t = 10^{-6}$ which remains acceptable in terms of accuracy. The ETDRK4 scheme can even go as high as $\Delta t = 10^{-4}$, though at the cost of accuracy. For reference, an explicit fourth order Runge-Kutta scheme would require $\Delta t \sim 2 \times 10^{-13}$ for $n = 4096$.

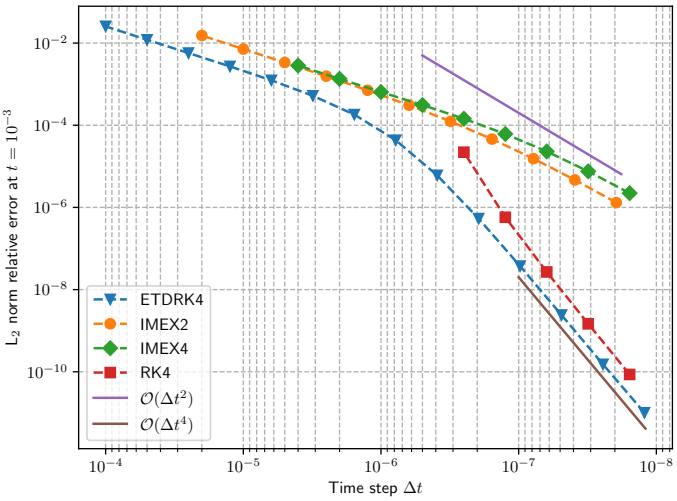


Fig. 2: Convergence results of the time stepping schemes for $n = 128$. Error is estimated using a reference solution computed using RK4 with $\Delta t = 10^{-9}$.

D. Results

Fig. 2 shows the convergence results of the various schemes for $n = 128$. The best scheme clearly is ETDRK4. It even outperforms the classical fourth order explicit Runge-Kutta scheme. While the IMEX schemes use four times less calls to the FFT than ETDRK4, they clearly underperform, especially the IMEX4 scheme. Unexpectedly, it displays only a slightly worse second order convergence behavior than IMEX2. The exact cause of those results is unclear to us, but [10] suggests the IMEX schemes are subject to dispersive errors. Moreover, we observe similar effects in our results as [10] in their analysis of the 2D Boussinesq equation (amongst others).

V. Implementation

Section IV shows how the choice of the integration scheme influences the size of the time step and thus the simulation speed. The way a solver is implemented is equally important and we tried to optimise that using different methods. Fig. 3 summarises the performance of the various implementations discussed in the following sections.

A. Fast Fourier Transform optimisations

Most of the execution of the algorithm is spent inside FFT routines. Thus, most of the optimisation efforts are focused on improving the efficiency of the FFT. First, every FFT call is made using real-valued Fourier Transform routines. Indeed, as the input data $c(\mathbf{x}, t)$ is purely real, it is possible to take advantage of the complex conjugate symmetry of $\hat{c}_\mathbf{k}$ and only compute half the values, thus speeding up the computations by a factor two.

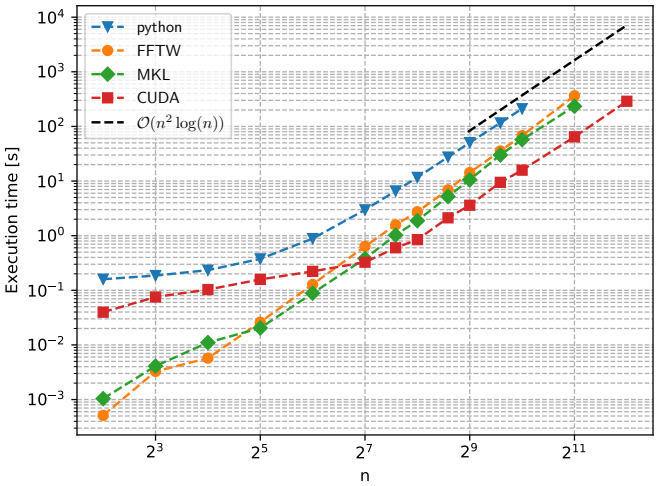


Fig. 3: Performance of the different implementations for 1000 iterations of the ETDRK4 solver

Next, we took a look at different libraries and FFT implementations:

- PYTHON implementation - SCIPY's FFT module: under the hood, SCIPY uses Fortran's FFTPACK and is reasonably fast, though definitely not the fastest. Its main interest is the incredible ease of use: indeed, very few lines of codes are required to have a functioning Cahn-Hilliard integrator.
- C implementation - FFTW3 and Intel's MKL: one of the most popular FFT libraries out there is the so-called Fastest Fourier Transform in the West, or FFTW3. It implements various transforms, including the real-valued ones. During our tests, it performed up to $18\times$ faster than SCIPY. However, we also found Intel's MKL to be slightly faster on our Intel-based processors, though it may not be the case on other platforms.
  We also tried to parallelise our code using OPENMP, but to no great success. Results obtained using 6 threads (one per processor) were slower than a single-threaded implementation. This is likely due to overheads during threads creation or a non-optimal usage of OPENMP's capabilities on our side.
- CUDA implementation: lastly, we implemented a parallel version of our code on GPU using CUDA and its CUFFT library. Using Nvidia's profiler, *Nsight Compute*, we analysed the performance of our code and transferred all relevant data arrays in the GPU's global memory to minimise memory access bottlenecks. Thus, all comuptations are performed on the GPU and data is only transferred back to the CPU when required for visualisation. More optimisations could be done using the GPU's faster shared memory, but it was not practical in our case as the CUFFT API requires to be called from the CPU and thus a single iterations has to be subdivided in several kernels.
  The end result is only slightly faster for $n = 128$ due to the various added overheads, but the scaling to bigger grids yields far better results with up to $4\times$ faster code execution for $n = 4096$.

B. Visualisation

Since the execution time for a single iteration is quite short ($\sim 0.5\,\mathrm{ms}$ for $n = 128$ on GPU), a fast visualization method is important to avoid any plot-related bottlenecks and maintain a high framerate. During developpment, we tried three different methods:

- MATPLOTLIB-CPP, a C++ wrapper for the well-known MATPLOTLIB python library. It is fairly easy to use, but not really efficient as it takes around $150\,\mathrm{ms}$ to update a single frame.
- BOV, a thin wrapper around OPENGL developped by the UCLouvain with the goal of fast and easy mesh drawing. However, as our spatial discretistation remains constant throughout the simulation, it proved quite inefficient with around $50\,\mathrm{ms}$ to update a single frame.
- OPENGL, which is a well-known graphical library. Our custom-made implementation allowed to update only the relevant information at each frame and achieve the fastest frame update time of $0.15\,\mathrm{ms}$.

Overall, our custom OPENGL implementation yields the fastest results by a wide margin (more than $300\times$ faster than BOV !) and allows us to run $60\,\mathrm{Hz}$ simulations with less than $1\,\%$ of the execution time dedicated to the plot update.

VI. SIMULATION RESULTS

Fig. 4 shows the results of a simulation for $n = 2048$ and $\Delta t = 10^{-6}$. We can observe that both phase tend to regroup and form domains of homogeneous composition and grow over time.

To further validate our results, we reproduced Zhu *et al.*'s results [3]. They studied the growth law of a spinodal decomposition, both in the case of a constant and variable mobility. To do so, they picked the following parameters: $\Delta t = 1$, $\kappa = 1$ and a $[0, 1023] \times [0, 1023]$ physical domain with 1024^2 discretisation points. They studied the domain size growth law using the structure function $S(k, t)$, which we compute as:

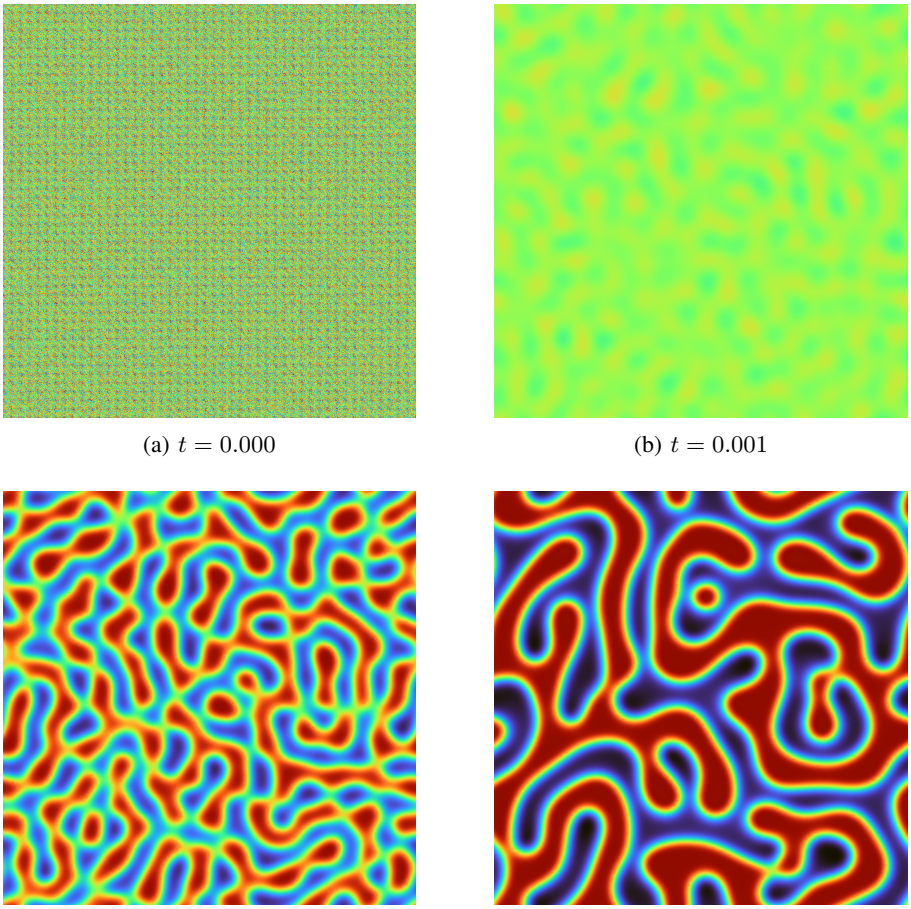
$$S(\mathbf{k}, t) = \frac{\hat{c}_{\mathbf{k}} \hat{c}^*_{\mathbf{k}}}{\alpha} \tag{18}$$

From this, the correct normalisation factor α can be found using Parseval's identity and we obtain $S(k, t)$ by computing the circular average of $S(\mathbf{k}, t)$.

There are several ways to characterise the characteristic length $L(t)$ of the domains. Here, we use the first moment $k_1(t)$ of $S(k, t)$, defined by:

$$k_1(t) = \frac{\sum_k k\, S(k, t)}{\sum_k S(k, t)} \tag{19}$$

Then, the domain size $L(t)$ is given by $1/k_1(t)$. We find that $L(t) \sim t^{1/3}$ for a constant mobility, while the scaling is much slower for a variable mobility with $L(t) \sim t^{1/4}$. Our results are presented in Appendix A and match those found by Zhu *et al.*



(a) $t = 0.000$ (b) $t = 0.001$

(c) $t = 0.003$ (d) $t = 0.012$

Fig. 4: Numerical simulation of the Cahn-Hilliard equation with $\kappa = 10^{-4}$, $n = 2048$ and $\Delta t = 10^{-6}$. Blue and red levels indicate a homogeneous composition of either phase while green shows an heterogeneous composition.

VII. Conclusion

In this work, we implemented fast and efficient solvers for the Cahn-Hilliard equation using Python, C and CUDA. The parallel version using CUDA is by far the fastest and allows to use large simulation domain without a too high of a time cost. We ran simulations up to $n = 4096$, which were solved in a reasonable time frame. Considering we don't have the latest GPUs, it is reasonnable to expect even bigger domains could be simulated using more recent hardware.

We also studied different semi-implicit time stepping schemes which allow to remove any stability constraint on the time step Δt without having to bear with the inherent high cost of fully implicit schemes. Amongst those, the ETDRK4 was determined to be the most efficient, both in terms of accuracy and efficiency.

To validate our code, we also reproduced Zhu *et al.*'s result both for the constant and variable mobility Cahn-Hilliard equations. We obtained very similar results, thus validating the correctness of our implementation.

References

[1] Q. Yuming and J. Kim, "Basic principles and practical applications of the Cahn–Hilliard equation," *Mathematical Problems in Engineering*, vol. 2016, 2016.

[2] J. W. Cahn and J. E. Hilliard, "Free energy of a nonuniform system. I. Interfacial free energy," *The Journal of chemical physics*, vol. 28, no. 2, pp. 258–267, 1958.

[3] J. Zhu, L.-Q. Chen, J. Shen, and V. Tikare, "Coarsening kinetics from a variable-mobility Cahn-Hilliard equation: Application of a semi-implicit fourier spectral method," *Phys. Rev. E*, vol. 60, pp. 3564–3572, Oct 1999. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.60.3564

[4] L. N. Trefethen, *Spectral methods in MATLAB*. SIAM, 2000.

[5] A.-K. Kassam and L. N. Trefethen, "Fourth-order time-stepping for stiff PDEs," *SIAM Journal on Scientific Computing*, vol. 26, no. 4, pp. 1214–1233, 2005. [Online]. Available: https://doi.org/10.1137/S1064827502410633

[6] W. Hundsdorfer and S. J. Ruuth, "IMEX extensions of linear multistep methods with general monotonicity and boundedness properties," *Journal of Computational Physics*, vol. 225, no. 2, pp. 2016–2042, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999107001003

[7] U. M. Ascher, S. J. Ruuth, and B. T. Wetton, "Implicit-explicit methods for time-dependent partial differential equations," *SIAM Journal on Numerical Analysis*, vol. 32, no. 3, pp. 797–823, 1995.

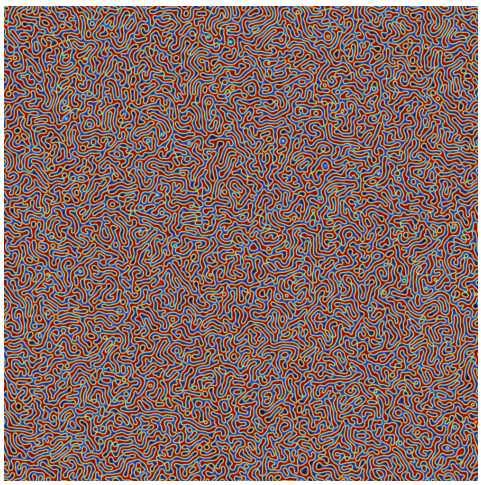
[8] S. Cox and P. Matthews, "Exponential time differencing for stiff systems," *Journal of Computational Physics*, vol. 176, no. 2, pp. 430–455, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999102969950

[9] Q. Du and W. Zhu, "Analysis and applications of the exponential time differencing schemes and their contour integration modifications," *BIT Numerical Mathematics*, vol. 45, no. 2, pp. 307–328, 2005.

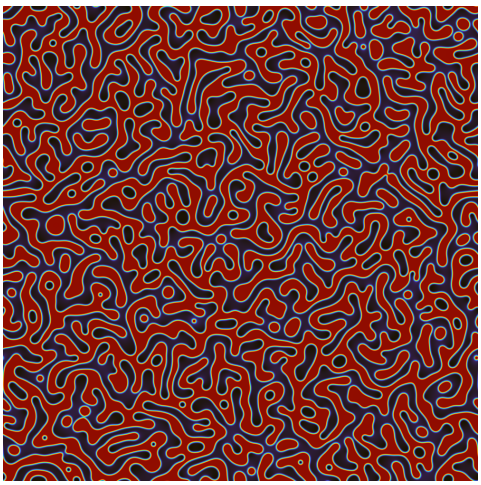
[10] I. Grooms and K. Julien, "Linearly implicit methods for nonlinear PDEs with linear dispersion and dissipation," *Journal of Computational Physics*, vol. 230, no. 9, pp. 3630–3650, 2011.

APPENDIX

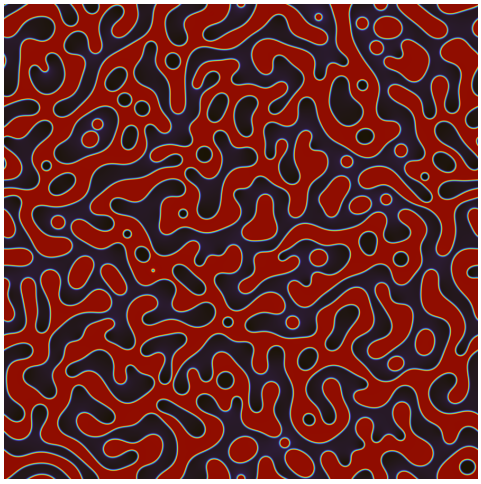
A. Simulation results



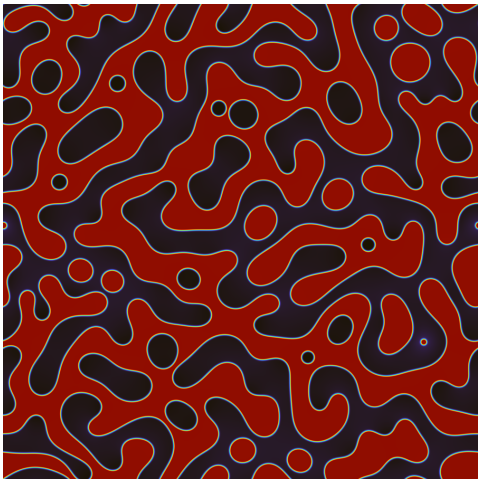
(a) $t = 100$



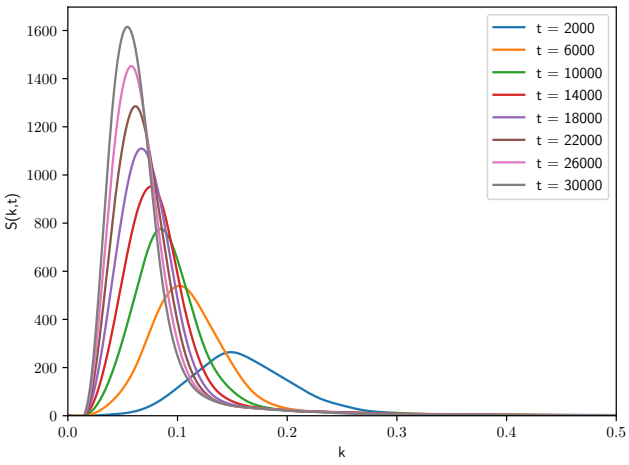
(b) $t = 2000$



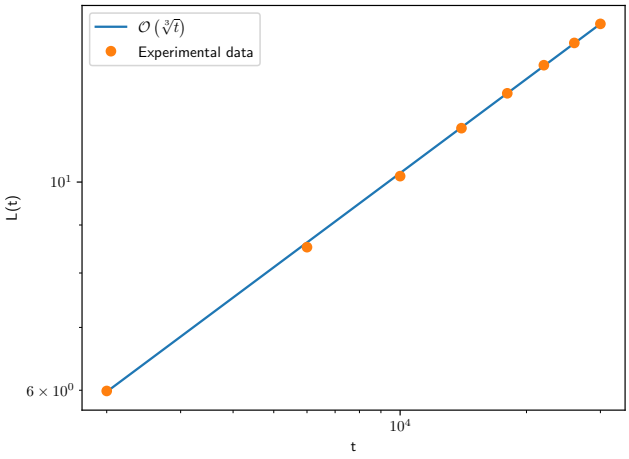
(c) $t = 10000$



(d) $t = 30000$

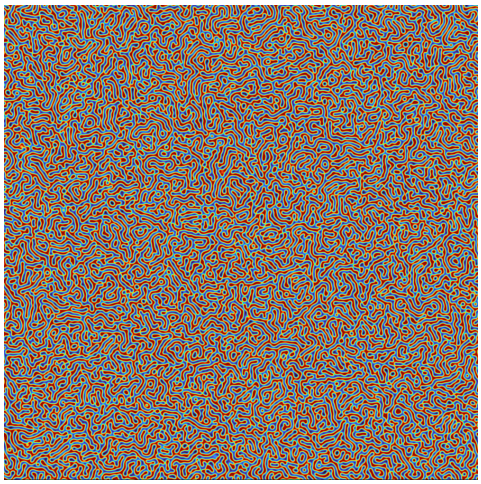


(e) Structure function

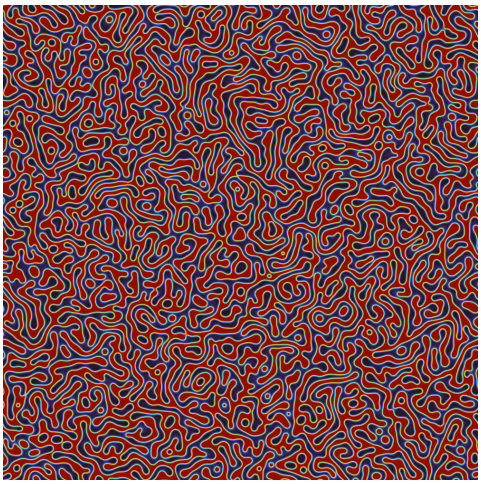


(f) Characteristic length growth

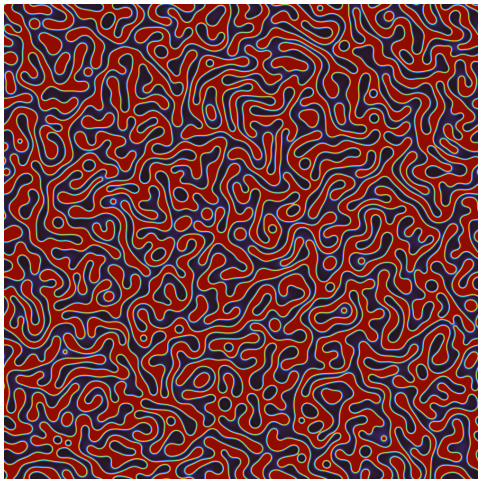
Fig. 5: Simulation of a spinodal decomposition with constant mobility.



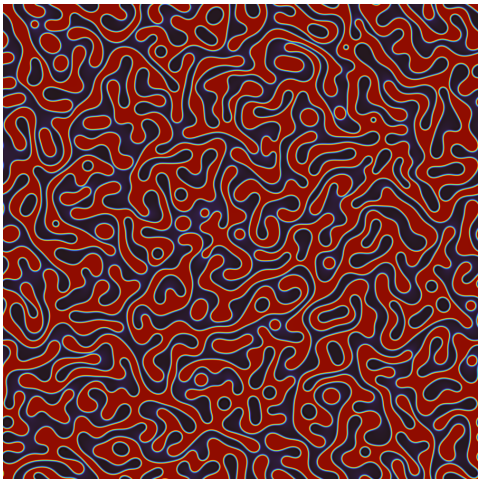
(a) $t = 100$



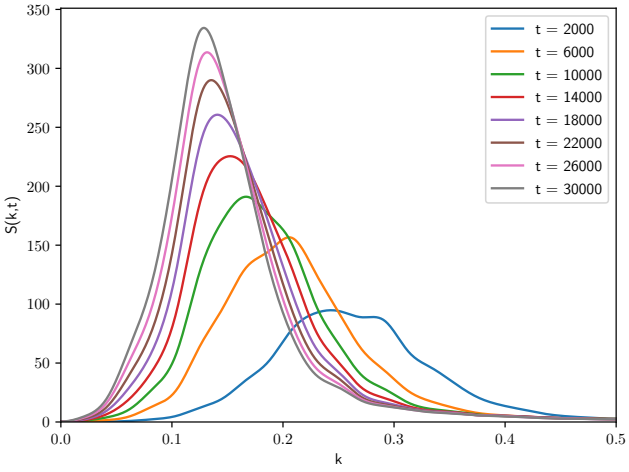
(b) $t = 2000$



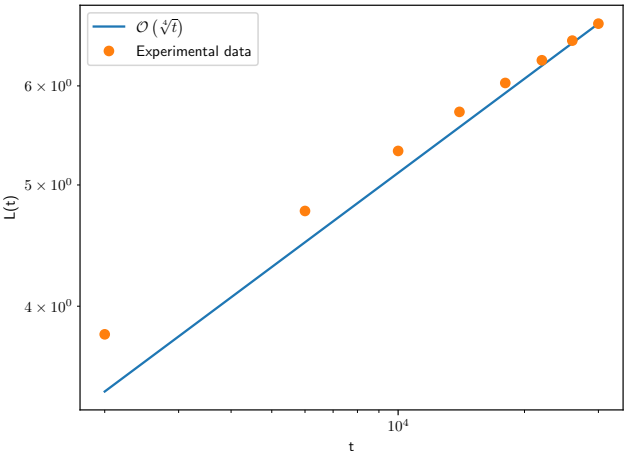
(c) $t = 10000$



(d) $t = 30000$



(e) Structure function



(f) Characteristic length growth

Fig. 6: Simulation of a spinodal decomposition with variable mobility.