mathematics

Article

Nonlinear Multigrid Implementation for the Two-Dimensional Cahn–Hilliard Equation

Chaeyoung Lee [1], Darae Jeong [2], Junxiang Yang [1] and Junseok Kim [1,*]

[1] Department of Mathematics, Korea University, Seoul 02841, Korea; chae1228@korea.ac.kr (C.L.); nexusxiang@outlook.com (J.Y.)

[2] Department of Mathematics, Kangwon National University, Chuncheon-si 200-090, Korea; tinayoyo@kangwon.ac.kr

* Correspondence: cfdkim@korea.ac.kr

Received: 19 December 2019; Accepted: 4 January 2020; Published: 7 January 2020

Abstract: We present a nonlinear multigrid implementation for the two-dimensional Cahn–Hilliard (CH) equation and conduct detailed numerical tests to explore the performance of the multigrid method for the CH equation. The CH equation was originally developed by Cahn and Hilliard to model phase separation phenomena. The CH equation has been used to model many interface-related problems, such as the spinodal decomposition of a binary alloy mixture, inpainting of binary images, microphase separation of diblock copolymers, microstructures with elastic inhomogeneity, two-phase binary fluids, in silico tumor growth simulation and structural topology optimization. The CH equation is discretized by using Eyre's unconditionally gradient stable scheme. The system of discrete equations is solved using an iterative method such as a nonlinear multigrid approach, which is one of the most efficient iterative methods for solving partial differential equations. Characteristic numerical experiments are conducted to demonstrate the efficiency and accuracy of the multigrid method for the CH equation. In the Appendix, we provide *C* code for implementing the nonlinear multigrid method for the two-dimensional CH equation.

Keywords: Cahn–Hilliard equation; multigrid method; unconditionally gradient stable scheme

MSC: 65N06; 65N55

1. Introduction

In this paper, we consider a detailed multigrid [1] implementation of the following two-dimensional Cahn–Hilliard (CH) equation [2] and provide its *C* source code:

$$\begin{aligned}\frac{\partial \phi(x,y,t)}{\partial t} &= M\Delta\mu(x,y,t), \quad (x,y)\in\Omega, \quad t>0,\\ \mu(x,y,t) &= F'(\phi(x,y,t)) - \epsilon^2\Delta\phi(x,y,t),\end{aligned}$$

where ϕ is a conserved scalar field; M is the mobility; $F(\phi) = 0.25(\phi^2-1)^2$ is the free energy function (see Figure 1); ϵ is the gradient interfacial energy coefficient; and $\Omega \subset \mathbb{R}^2$ is a bounded domain.

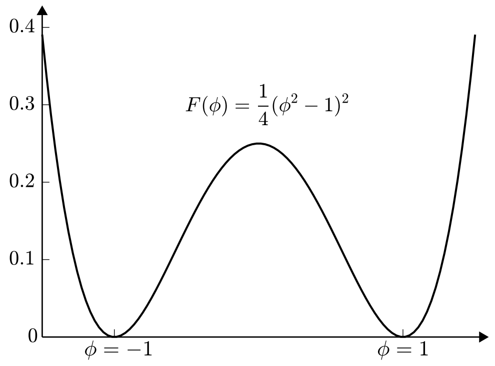


Figure 1. Double-well potential $F(\phi) = 0.25(\phi^2 - 1)^2$.

The homogeneous Neumann boundary conditions are used and are given as follows:

$$\mathbf{n} \cdot \nabla\phi = 0, \tag{1}$$

$$\mathbf{n} \cdot \nabla\mu = 0 \text{ on } \partial\Omega. \tag{2}$$

Here, $\mathbf{n}$ is the unit normal vector on the domain boundary $\partial\Omega$. The first boundary condition (1) implies that the interface contacts the domain boundary at a 90° angle. The second boundary condition (2) implies that the total mass is conserved.

We can derive the CH equation from the following total free energy functional

$$\mathcal{E}(\phi) = \int_\Omega \left[F(\phi) + \frac{\epsilon^2}{2}|\nabla\phi|^2 \right] d\mathbf{x}.$$

Taking the variational derivative of $\mathcal{E}$ with respect to ϕ, we define the chemical potential:

$$\mu = \frac{\delta\mathcal{E}}{\delta\phi} = F'(\phi) - \epsilon^2\Delta\phi.$$

Conservation of mass implies the following CH equation

$$\phi_t = -\nabla \cdot \mathcal{F},$$

where the flux is given by $\mathcal{F} = -M\nabla\mu$. If we differentiate $\mathcal{E}(\phi)$ and $\int_\Omega \phi \, d\mathbf{x}$ with respect to time t, then we have

$$\begin{aligned} \frac{d}{dt}\mathcal{E}(\phi) &= \int_\Omega \left[F'(\phi)\phi_t + \epsilon^2\nabla\phi \cdot \nabla\phi_t \right] \mathrm{d}\mathbf{x} = \int_\Omega \mu\phi_t \, \mathrm{d}\mathbf{x} = M\int_\Omega \mu\Delta\mu \, \mathrm{d}\mathbf{x} \\ &= M\int_{\partial\Omega} \mu\nabla\mu \cdot \mathbf{n} \, \mathrm{d}s - M\int_\Omega \nabla\mu \cdot \nabla\mu \, \mathrm{d}\mathbf{x} = -M\int_\Omega |\nabla\mu|^2 \, \mathrm{d}\mathbf{x} \le 0 \end{aligned} \tag{3}$$

and

$$\frac{d}{dt}\int_\Omega \phi \, \mathrm{d}\mathbf{x} = \int_\Omega \phi_t \, \mathrm{d}\mathbf{x} = M\int_\Omega \Delta\mu \, \mathrm{d}\mathbf{x} = M\int_{\partial\Omega} \nabla\mu \cdot \mathbf{n} \, \mathrm{d}s = 0, \tag{4}$$

which imply that the total energy is decreasing and that the total mass is conserved in time, respectively. The CH equation was originally developed by Cahn and Hilliard to model spinodal decomposition in a binary alloy. The CH equation has been used to address many major problems such as the spinodal decomposition of a binary alloy mixture [3,4], inpainting of binary images [5,6], microphase separation of diblock copolymers [7,8], microstructures with elastic inhomogeneity [9,10], two-phase binary fluids [11,12], tumor growth models [13–15] and structural topology optimization [14,16]. Further details regarding the basic principles and practical applications of the CH equation are available in a recent review [14]. Thus, knowing how to implement a discrete scheme for the CH equation in detail is very useful because this equation is a building-block equation for many applications. The CH equation is discretized by using Eyre's unconditionally gradient stable scheme [17] and is solved by using a nonlinear multigrid technique [1], which is one of the most efficient iterative methods for solving partial differential equations. Several studies have used the nonlinear multigrid method for the CH-type equations [18–23]. However, details regarding the implementation, multigrid performance, and source codes have not been provided.

Therefore, the main purpose of this paper is to describe a detailed multigrid implementation of the two-dimensional CH equation, evaluate its performance and provide its *C* programming language source code.

The remainder of this paper is organized as follows. In Section 2, we describe the numerical solution in detail. In Section 3, we describe the characteristic numerical experiments that are conducted to demonstrate the accuracy and efficiency of the multigrid method for the CH equation. In Section 4, we provide a conclusion. In the Appendix A, we provide the *C* code for implementing the nonlinear multigrid technique for the two-dimensional CH equation.

2. Numerical Solution

We consider a finite difference approximation for the CH equation. An unconditionally gradient energy stable scheme, which was proposed by Eyre, is applied to the time discretization. A nonlinear multigrid technique [1] is applied to solve the resulting system at an implicit time level.

2.1. Discretization

We discretize the CH equation in the two-dimensional space $\Omega = (a,b) \times (c,d)$. Let $N_x = 2^p$ and $N_y = 2^q$ be the numbers of mesh points with integers p and q. Let $\Delta x = (b-a)/N_x$ and $\Delta y = (d-c)/N_y$ be the mesh size. Let $\Omega_{ij} = \{(x_i, y_j) : x_i = a + (i-0.5)\Delta x,\ y_j = c + (j-0.5)\Delta y,\ 1 \le i \le N_x,\ 1 \le j \le N_y\}$ be a discrete computational domain. Let ϕ_{ij}^n and μ_{ij}^n be approximations of $\phi(x_i, y_j, t_n)$ and $\mu(x_i, y_j, t_n)$, respectively. Here, $t_n = n\Delta t$ and Δt represent the temporal step. We assume a uniform mesh grid $h = \Delta x = \Delta y$ and a constant mobility $M = 1$. Using the nonlinear stabilized splitting scheme of Eyre's unconditionally gradient stable scheme, the CH equation is discretized as

$$\frac{\phi_{ij}^{n+1} - \phi_{ij}^n}{\Delta t} = \Delta_h \mu_{ij}^{n+1}, \tag{5}$$

$$\mu_{ij}^{n+1} = (\phi_{ij}^{n+1})^3 - \phi_{ij}^n - \epsilon^2 \Delta_h \phi_{ij}^{n+1}, \tag{6}$$

where the discrete Laplace operator is defined by $\Delta_h \psi_{ij} = (\psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} - 4\psi_{ij})/h^2$. The homogeneous Neumann boundary conditions (1) and (2) are discretized as

$$\phi_{0j} = \phi_{1j},\ \phi_{N_x+1,j} = \phi_{N_x,j},\ \mu_{0j} = \mu_{1j},\ \mu_{N_x+1,j} = \mu_{N_x,j},\ j = 1, \ldots, N_y, \tag{7}$$

$$\phi_{i0} = \phi_{i1},\ \phi_{i,N_y+1} = \phi_{i,N_y},\ \mu_{i0} = \mu_{i1},\ \mu_{i,N_y+1} = \mu_{i,N_y},\ i = 1, \ldots, N_x. \tag{8}$$

We define the discrete residual as

$$r_{ij} = \Delta_h \mu_{ij}^{n+1} - \frac{\phi_{ij}^{n+1} - \phi_{ij}^n}{\Delta t}. \tag{9}$$

For each element a_{ij} of size $N_x \times N_y$ in matrix A, we define the Frobenius norm with a scaling and infinite norm as

$$\|A\|_F = \sqrt{\frac{\sum_{i=1}^{N_x} \sum_{j=1}^{N_y} |a_{ij}|^2}{N_x N_y}} \text{ and } \|A\|_\infty = \max_{1 \le i \le N_x, 1 \le j \le N_y} |a_{ij}|, \tag{10}$$

respectively. The discretizations (5) and (6) are conservative, that is,

$$\sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\phi_{ij}^{n+1}=\sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\phi_{ij}^{n}. \tag{11}$$

To show this conservation property, we take the summation of Equation (5)

$$\begin{aligned}\sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\frac{\phi_{ij}^{n+1}-\phi_{ij}^{n}}{\Delta t} &= \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\Delta_h\mu_{ij}^{n+1} \\ &= \sum_{j=1}^{N_y}\left(\frac{\mu_{N_x+1,j}^{n+1}-\mu_{N_xj}^{n+1}}{h^2}-\frac{\mu_{1j}^{n+1}-\mu_{0j}^{n+1}}{h^2}\right) \\ &\quad+\sum_{i=1}^{N_x}\left(\frac{\mu_{i,N_y+1}^{n+1}-\mu_{iN_y}^{n+1}}{h^2}-\frac{\mu_{i1}^{n+1}-\mu_{i0}^{n+1}}{h^2}\right)=0.\end{aligned} \tag{12}$$

Here, we used the homogenous Neumann boundary conditions (7) and (8). Therefore, Equation (11) holds. We define the discrete energy functional as

$$\begin{aligned}\mathcal{E}^h(\phi^n) &= h^2\sum_{i=1}^{N_x}\sum_{j=1}^{N_y}F(\phi_{ij}^n) \\ &\quad+\frac{\epsilon^2}{2}\sum_{j=1}^{N_y}\left(\frac{(\phi_{1j}^n-\phi_{0j}^n)^2}{2}+\sum_{i=1}^{N_x-1}(\phi_{i+1,j}^n-\phi_{ij}^n)^2+\frac{(\phi_{N_x+1,j}^n-\phi_{N_xj}^n)^2}{2}\right) \\ &\quad+\frac{\epsilon^2}{2}\sum_{i=1}^{N_x}\left(\frac{(\phi_{i1}^n-\phi_{i0}^n)^2}{2}+\sum_{j=1}^{N_y-1}(\phi_{i,j+1}^n-\phi_{ij}^n)^2+\frac{(\phi_{i,N_y+1}^n-\phi_{iN_y}^n)^2}{2}\right) \\ &= h^2\sum_{i=1}^{N_x}\sum_{j=1}^{N_y}F(\phi_{ij}^n)+\frac{\epsilon^2}{2}\sum_{j=1}^{N_y}\sum_{i=1}^{N_x-1}(\phi_{i+1,j}^n-\phi_{ij}^n)^2+\frac{\epsilon^2}{2}\sum_{i=1}^{N_x}\sum_{j=1}^{N_y-1}(\phi_{i,j+1}^n-\phi_{ij}^n)^2,\end{aligned} \tag{13}$$

where we used the homogenous Neumann boundary conditions (7) and (8). We also define the discrete total mass as

$$\mathcal{M}^h(\phi^n)=\sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\phi_{ij}^n h^2. \tag{14}$$

Then, the unconditionally gradient stable scheme satisfies the reduction in the discrete total energy [24]:

$$\mathcal{E}^h(\phi^{n+1})\le\mathcal{E}^h(\phi^n), \tag{15}$$

which implies the pointwise boundedness of the numerical solution:

$$\|\phi^n\|_\infty\le\sqrt{1+2\sqrt{\mathcal{E}^h(\boldsymbol{\phi}^0)/h^2}}\quad\text{for all } n. \tag{16}$$

The proof of Equation (16) can be found in Reference [25]. We provide the proof herein for the sake of completeness. We show that a constant K exists for all n values that satisfy the following inequality:

$$\|\phi^n\|_\infty\le K. \tag{17}$$

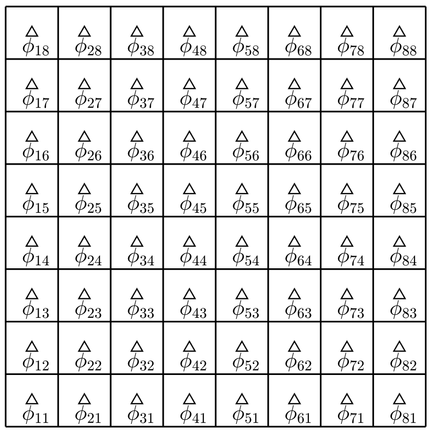
Let us assume that there is an integer n_K that is dependent on K such that $\|\phi^{n_K}\|_\infty > K$ for any K. Then, $\phi_{ij}^{n_K}$ exists such that $|\phi_{ij}^{n_K}| > K$. Let K be the largest solution of $\mathcal{E}^h(\phi^0) = h^2F(K)$, that is, $K = \sqrt{1+2\sqrt{\mathcal{E}^h(\phi^0)/h^2}}$. We then have

$$\mathcal{E}^h(\phi^0) = h^2F(K) < h^2F(|\phi_{ij}^{n_K}|) \le \mathcal{E}^h(\phi^{n_K}) \le \mathcal{E}^h(\phi^0), \tag{18}$$

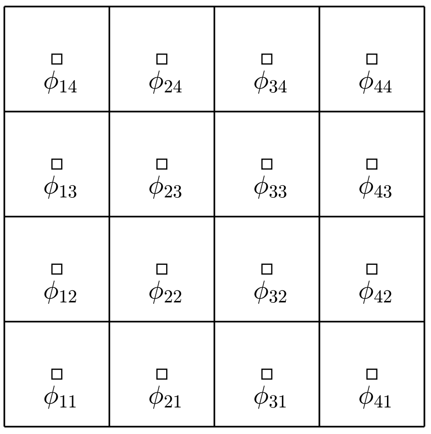
where we utilize the fact that the total energy is decreasing and $F(\phi)$ is a strictly increasing function on (K,∞). Equation (18) leads to a contradiction. Therefore, Equation (17) should be satisfied.

2.2. Multigrid V-Cycle Algorithm

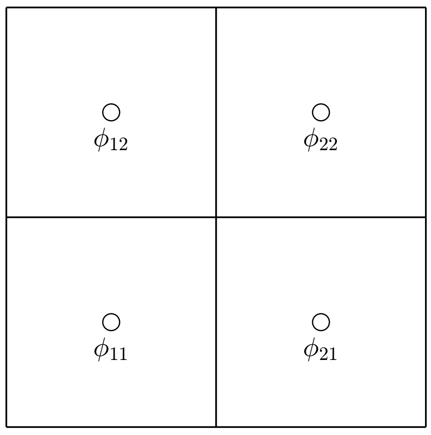
We use the nonlinear full approximation storage (FAS) multigrid method to solve the nonlinear discrete systems (5) and (6). For simplicity, we define the discrete domains, Ω_2, Ω_1, and Ω_0, which represent a hierarchy of meshes (Ω_2, Ω_1, and Ω_0) created by successively coarsening the original mesh Ω_2, as shown in Figure 2.


ϕ_{18} ϕ_{28} ϕ_{38} ϕ_{48} ϕ_{58} ϕ_{68} ϕ_{78} ϕ_{88}
ϕ_{17} ϕ_{27} ϕ_{37} ϕ_{47} ϕ_{57} ϕ_{67} ϕ_{77} ϕ_{87}
ϕ_{16} ϕ_{26} ϕ_{36} ϕ_{46} ϕ_{56} ϕ_{66} ϕ_{76} ϕ_{86}
ϕ_{15} ϕ_{25} ϕ_{35} ϕ_{45} ϕ_{55} ϕ_{65} ϕ_{75} ϕ_{85}
ϕ_{14} ϕ_{24} ϕ_{34} ϕ_{44} ϕ_{54} ϕ_{64} ϕ_{74} ϕ_{84}
ϕ_{13} ϕ_{23} ϕ_{33} ϕ_{43} ϕ_{53} ϕ_{63} ϕ_{73} ϕ_{83}
ϕ_{12} ϕ_{22} ϕ_{32} ϕ_{42} ϕ_{52} ϕ_{62} ϕ_{72} ϕ_{82}
ϕ_{11} ϕ_{21} ϕ_{31} ϕ_{41} ϕ_{51} ϕ_{61} ϕ_{71} ϕ_{81}

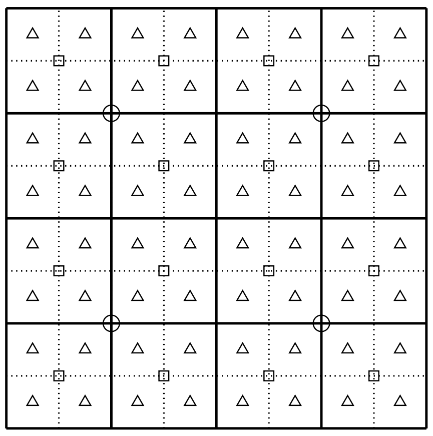
(**a**) Ω_2 $(8\times 8)h$


ϕ_{14} ϕ_{24} ϕ_{34} ϕ_{44}
ϕ_{13} ϕ_{23} ϕ_{33} ϕ_{43}
ϕ_{12} ϕ_{22} ϕ_{32} ϕ_{42}
ϕ_{11} ϕ_{21} ϕ_{31} ϕ_{41}

(**b**) Ω_1 $(4\times 4)h$



(**c**) Ω_0 $(2\times 2)h$



(**d**)

Figure 2. (**a**–**c**) represent a sequence of coarse grids starting with $h = L/N_x$. (**d**) depicts a composition of grids, Ω_2, Ω_1 and Ω_0.

We summarize here the nonlinear multigrid method for solving the discrete CH system as follows: First, let us rewrite Equations (5) and (6) as

$$NSO(\phi^{n+1}, \mu^{n+1}) = (\xi^n, \psi^n),$$

where the linear operator NSO is defined as

$$NSO(\phi^{n+1},\mu^{n+1}) = \left(\phi^{n+1}/\Delta t - \Delta_h\mu^{n+1},\ \mu^{n+1} - (\phi^{n+1})^3 + \epsilon^2\Delta_h\phi^{n+1}\right),$$

and the source term is denoted by

$$(\xi^n,\psi^n) = (\phi^n/\Delta t,\ -\phi^n). \tag{19}$$

Next, we describe the multigrid method, which includes the pre-smoothing, coarse grid correction and post-smoothing steps. We denote a mesh grid Ω_k as the discrete domain for each multigrid level k. Note that a mesh grid Ω_k contains $2^k \times 2^k$ grid points. Let k_{min} be the coarsest multigrid level. We now introduce the SMOOTH and V-cycle functions. Given the number ν_1 of pre-smoothing and ν_2 of post-smoothing relaxation sweeps, the V-cycle is used as an iteration step in the multigrid method.

FAS multigrid cycle

Now, we define the FAScycle:

$$\{\phi_k^{m+1},\mu_k^{m+1}\} = FAScycle(k,\phi_k^m,\mu_k^m,NSO_k,\xi_k^n,\psi_k^n,\beta).$$

In other words, $\{\phi_k^m,\mu_k^m\}$ and $\{\phi_k^{m+1},\mu_k^{m+1}\}$ are the approximations of $\phi^{n+1}(x_i,y_j)$ and $\mu^{n+1}(x_i,y_j)$ before and after an FAScycle, respectively. Here, $\phi_k^0 = \phi_k^n$ and $\mu_k^0 = \mu_k^n$.

(1) Pre-smoothing

$$\{\bar{\phi}_k^m,\ \bar{\mu}_k^m\} = SMOOTH^{\nu_1}(\phi_k^m,\mu_k^m,NSO_k,\xi_k^n,\psi_k^n),$$

which represents ν_1 smoothing steps with the initial approximations ϕ_k^m,μ_k^m, source terms ξ_k^n,ψ_k^n and the $SMOOTH$ relaxation operator to obtain the approximations $\bar{\phi}_k^m,\ \bar{\mu}_k^m$. One $SMOOTH$ relaxation operator step consists of solving the systems (22) and (23), given as follows by 2×2 matrix inversion for each i and j. Here, we derive the smoothing operator in two dimensions. Rewriting Equation (5), we obtain:

$$\frac{\phi_{ij}^{n+1}}{\Delta t} + \frac{4\mu_{ij}^{n+1}}{h^2} = \xi_{ij}^n + \frac{\mu_{i+1,j}^{n+1} + \mu_{i-1,j}^{n+1} + \mu_{i,j+1}^{n+1} + \mu_{i,j-1}^{n+1}}{h^2}. \tag{20}$$

Because $(\phi_{ij}^{n+1})^3$ is nonlinear with respect to ϕ_{ij}^{n+1}, we linearize $(\phi_{ij}^{n+1})^3$ at ϕ_{ij}^m, that is,

$$(\phi_{ij}^{n+1})^3 \approx (\phi_{ij}^m)^3 + 3(\phi_{ij}^m)^2(\phi_{ij}^{n+1} - \phi_{ij}^m).$$

After substituting of this into (6), we obtain

$$-\left(\frac{4\epsilon^2}{h^2} + 3(\phi_{ij}^m)^2\right)\phi_{ij}^{n+1} + \mu_{ij}^{n+1} = \psi_{ij}^n - 2(\phi_{ij}^m)^3 - \frac{\epsilon^2}{h^2}(\phi_{i+1,j}^{n+1} + \phi_{i-1,j}^{n+1} + \phi_{i,j+1}^{n+1} + \phi_{i,j-1}^{n+1}). \tag{21}$$

Next, we replace $\phi_{\alpha,\beta}^{n+1}$ and $\mu_{\alpha,\beta}^{n+1}$ in Equations (20) and (21) with $\bar{\phi}_{\alpha,\beta}^m$ and $\bar{\mu}_{\alpha,\beta}^m$ for $\alpha \le i$ and $\beta \le j$, otherwise with $\phi_{\alpha,\beta}^m$ and $\mu_{\alpha,\beta}^m$, that is,

$$\frac{\bar{\phi}_{ij}^m}{\Delta t} + \frac{4\bar{\mu}_{ij}^m}{h^2} = \xi_{ij}^n + \frac{\mu_{i+1,j}^m + \bar{\mu}_{i-1,j}^m + \mu_{i,j+1}^m + \bar{\mu}_{i,j-1}^m}{h^2}, \tag{22}$$

$$-\left(\frac{4\epsilon^2}{h^2} + 3(\phi_{ij}^m)^2\right)\bar{\phi}_{ij}^m + \bar{\mu}_{ij}^m = \psi_{ij}^n - 2(\phi_{ij}^m)^3 - \frac{\epsilon^2}{h^2}(\phi_{i+1,j}^m + \bar{\phi}_{i-1,j}^m + \phi_{i,j+1}^m + \bar{\phi}_{i,j-1}^m). \tag{23}$$

(2) Compute the defect

$$(\bar{d_1}^m{}_k, \bar{d_2}^m{}_k) = (\xi_k^n, \psi_k^n) - NSO_k(\bar{\phi}_k^m, \bar{\mu}_k^m).$$

(3) Restrict the defect and $\{\bar{\phi}_k^m,\ \bar{\mu}_k^m\}$

$$(\bar{d}_{1k-1}^{\ m}, \bar{d}_{2k-1}^{\ m}) = I_k^{k-1}(\bar{d}_{1k}^{\ m}, \bar{d}_{2k}^{\ m}).$$

The restriction operator I_k^{k-1} maps k-level functions to $(k-1)$-level functions.

$$\begin{aligned} d_{k-1}(x_i, y_j) = I_k^{k-1} d_k(x_i, y_j) \ &= \ \frac{1}{4}[d_k(x_{i-\frac{1}{2}}, y_{j-\frac{1}{2}}) + d_k(x_{i-\frac{1}{2}}, y_{j+\frac{1}{2}}) \\ &\quad + d_k(x_{i+\frac{1}{2}}, y_{j-\frac{1}{2}}) + d_k(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}})]. \end{aligned}$$

(4) Compute the right-hand side

$$(\xi_{k-1}^n, \psi_{k-1}^n) = (\bar{d_1}^m{}_{k-1}, \bar{d_2}^m{}_{k-1}) + NSO_{k-1}(\bar{\phi}_{k-1}^m, \bar{\mu}_{k-1}^m).$$

(5) Compute an approximate solution $\{\hat{\phi}_{k-1}^m,\ \hat{\mu}_{k-1}^m\}$ *of the coarse grid equation on* Ω_{k-1}*, that is,*

$$NSO_{k-1}(\phi_{k-1}^m, \mu_{k-1}^m) = (\xi_{k-1}^n, \psi_{k-1}^n). \tag{24}$$

If $k = 1$, we explicitly invert the 2×2 matrix to obtain the solution. If $k > 1$, we solve Equation (24) by performing a FAS k-grid cycle using $\{\bar{\phi}_{k-1}^m,\ \bar{\mu}_{k-1}^m\}$ as the initial approximation:

$$\{\hat{\phi}_{k-1}^m, \hat{\mu}_{k-1}^m\} = \text{FAScycle}(k-1, \bar{\phi}_{k-1}^m, \bar{\mu}_{k-1}^m, NSO_{k-1}, \xi_{k-1}^n, \psi_{k-1}^n, \beta).$$

(6) Compute the coarse grid correction (CGC):

$$\hat{v}_{1k-1}^m = \hat{\phi}_{k-1}^m - \bar{\phi}_{k-1}^m, \quad \hat{v}_{2k-1}^m = \hat{\mu}_{k-1}^m - \bar{\mu}_{k-1}^m.$$

(7) Interpolate the correction:

$$\hat{v}_{1k}^m = I_{k-1}^k \hat{v}_{1k-1}^m, \quad \hat{v}_{2k}^m = I_{k-1}^k \hat{v}_{2k-1}^m.$$

Here, the coarse values are simply transferred to the four nearby fine grid points, that is, $v_k(x_i, y_j) = I_{k-1}^k v_{k-1}(x_i, y_j) = v_{k-1}(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}})$ for i and j odd-numbered integers.

(8) Compute the corrected approximation on Ω_k

$$\phi_k^{m,\text{ after } CGC} = \bar{\phi}_k^m + \hat{v}_{1k}^{\ m}, \quad \mu_k^{m,\text{ after } CGC} = \bar{\mu}_k^m + \hat{v}_{2k}^{\ m}.$$

(9) Post-smoothing

$$\{\phi_k^{m+1}, \mu_k^m\} = SMOOTH^{\nu_2}(\phi_k^{m,\text{ after } CGC}, \mu_k^{m,\text{ after } CGC}, NSO_k, \xi_k^n, \psi_k^n).$$

This completes the description of the nonlinear FAScycle. One FAScycle step stops if the consequent error $\|\phi^{n+1,m+1} - \phi^{n+1,m}\|_\infty$ is less than a given tolerance *tol*. The two-grid V-cycle is illustrated in Figure 3.

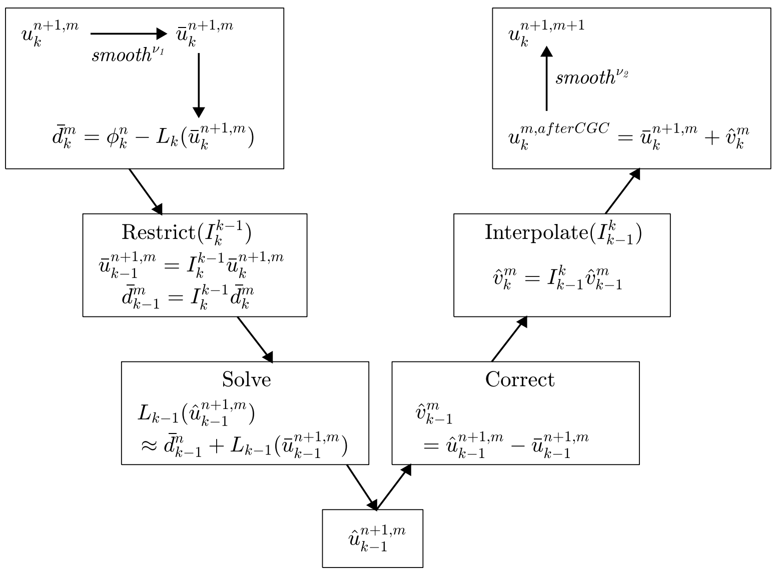

$u_k^{n+1,m}$
$smooth^{\nu_1}$
$\bar{u}_k^{n+1,m}$
$\bar{d}_k^m = \phi_k^n - L_k(\bar{u}_k^{n+1,m})$
Restrict(I_k^{k-1})
$\bar{u}_{k-1}^{n+1,m} = I_k^{k-1}\bar{u}_k^{n+1,m}$
$\bar{d}_{k-1}^m = I_k^{k-1}\bar{d}_k^m$
Solve
$L_{k-1}(\hat{u}_{k-1}^{n+1,m})$
$\approx \bar{d}_{k-1}^n + L_{k-1}(\bar{u}_{k-1}^{n+1,m})$
$\hat{u}_{k-1}^{n+1,m}$
Correct
$\hat{v}_{k-1}^m$
$= \hat{u}_{k-1}^{n+1,m} - \bar{u}_{k-1}^{n+1,m}$
Interpolate(I_{k-1}^k)
$\hat{v}_k^m = I_{k-1}^k \hat{v}_{k-1}^m$
$u_k^{m,afterCGC} = \bar{u}_k^{n+1,m} + \hat{v}_k^m$
$smooth^{\nu_2}$
$u_k^{n+1,m+1}$

Figure 3. Multigrid two-grid V-cycle method.

Further Numerical Schemes for the CH Equation

Previous studies have described the numerical solution of the CH equation with a variable mobility [19], the adaptive mesh refinement technique [26,27], the Neumann boundary condition in complex domains [20], the Dirichlet boundary conditions in complex domains [28], contact angle boundary [29], parallel multigrid method [30] and fourth-order compact scheme [31].

3. Numerical Experiments

In numerical experiments, we consider an equilibrium solution $\phi(x,\infty) = \tanh(x/\sqrt{2}\epsilon)$ for the CH equation on the one-dimensional infinite domain $\Omega = (-\infty,\infty)$. In other words, $\phi(x,\infty)$ satisfies $\mu(\phi(x,\infty)) = F'(\phi(x,\infty)) - \epsilon^2\phi_{xx}(x,\infty) = 0$ and is an equilibrium solution. Then, across the interfacial regions, ϕ varies from -0.9 to 0.9 over a distance of approximately $\xi = 2\sqrt{2}\epsilon\tanh^{-1}(0.9)$ (see Figure 4). Therefore, if we want this value to be approximately mh, the ϵ value can be taken as $\epsilon = \epsilon_m = mh/[2\sqrt{2}\tanh^{-1}(0.9)]$ [32].

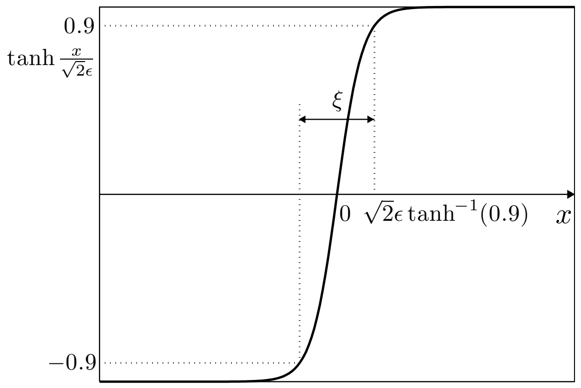
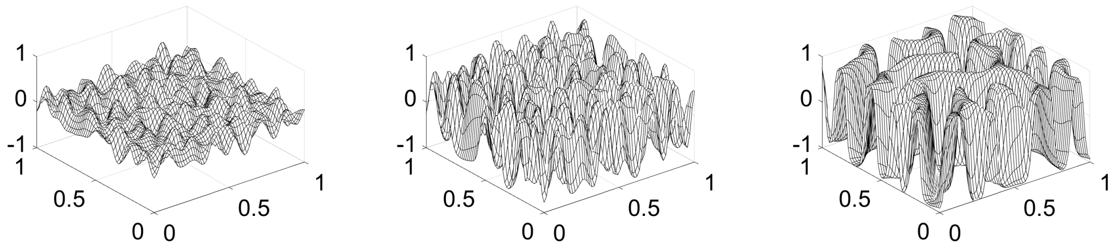


Figure 4. Concentration field varying from -0.9 to 0.9 over a distance of approximately $\xi = 2\sqrt{2}\epsilon\tanh^{-1}(0.9)$.

All computational simulations described in this section are performed on an Intel Core i5-6400 CPU @ 2.70 GHz with 4 GB of RAM.

3.1. Phase Separation

For the first numerical test, we consider spinodal decomposition in binary alloys. This decomposition is a process by which a mixture of binary materials separates into distinct regions with different material concentrations [2]. Figure 5a–c show snapshots of the phase-field ϕ at $t = 100\Delta t$, $200\Delta t$ and $1000\Delta t$, respectively. The initial condition is $\phi(x, y, 0) = 0.1(1 - 2\text{rand}(x, y))$ on $\Omega = (0, 1) \times (0, 1)$, where $\text{rand}(x, y)$ is a random value between 0 and 1. The parameters $\epsilon = \epsilon_4$, $h = 1/64$, $\Delta t = 0.1h^2$ and a tolerance of *tol* = 1.0×10^{-10} are used.



(**a**) $t = 100\Delta t$ (**b**) $t = 200\Delta t$ (**c**) $t = 1000\Delta t$

Figure 5. Snapshots of the phase-field ϕ at (**a**) $t = 100\Delta t$, (**b**) $t = 200\Delta t$ and (**c**) $t = 1000\Delta t$. Here, $\epsilon = \epsilon_4$, $h = 1/64$ and $\Delta t = 0.1h^2$ are used.

3.2. Non-Increase in Discrete Energy and Mass Conservation

Figure 6 shows the time evolution of the normalized discrete total energy $\mathcal{E}^h(\phi^n)/\mathcal{E}^h(\phi^0)$ (solid line) and the average mass $\mathcal{M}^h(\phi^n)/(h^2 N_x N_y)$ (diamond) of the numerical solutions with the initial state (25) on $\Omega = (0, 1) \times (0, 1)$.

$$\phi(x, y, 0) = 0.1(1 - 2\text{rand}(x, y)), \tag{25}$$

where $\text{rand}(x, y)$ is a random value between 0 and 1.

We use the simulation parameters, $\epsilon = \epsilon_4$, $h = 1/64$, $\Delta t = 0.1h^2$ and *tol* = 1.0×10^{-10}. The energy is non-increasing and the average concentration is conserved. These numerical results agree well with the total energy dissipation property (3) and the conservation property (4). The inscribed small figures are the concentration fields at the indicated times.

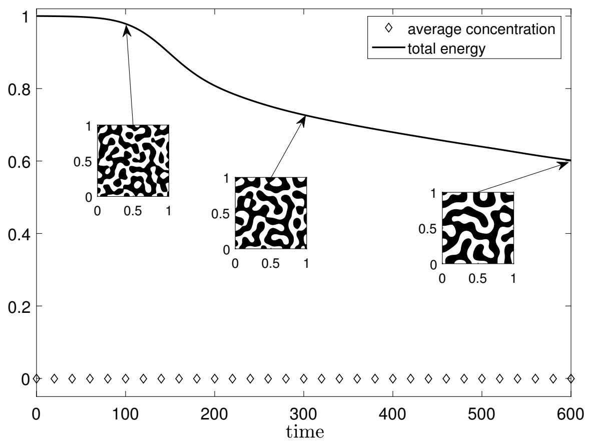


Figure 6. Normalized discrete total energy $\mathcal{E}^h(\phi^n)/\mathcal{E}^h(\phi^0)$ (solid line) and average concentration $\mathcal{M}^h(\phi^n)/(h^2 N_x N_y)$ (diamond line) of the numerical solutions with the initial state (25).

3.3. Convergence Test

We consider the convergence of the Frobenius norm with a scaling of the residual error with respect to the grid size. The initial condition on the domain $\Omega = (0,1) \times (0,1)$ is given as

$$\phi(x,y,0) = 0.1\cos(\pi x)\cos(\pi y). \tag{26}$$

We fix $\epsilon = 0.06$, $\Delta t = 0.01$ and $tol = 1.0 \times 10^{-15}$. Here, we use the $V(2,2)$ scheme with a Gauss–Seidel relaxation, where $(2,2)$ indicates 2 pre- and 2 post-correction relaxation sweeps. We define the residual after m V-cycles as

$$r_{ij}^{m} = \Delta_h \mu_{ij}^{n+1,m} - \frac{\phi_{ij}^{n+1,m} - \phi_{ij}^{n}}{\Delta t}. \tag{27}$$

Table 1 shows the residual norm $\|r^m\|_F$ after each V-cycle. Because no closed-form analytical solution exists for this problem, we define the Frobenius norm with a scaling of the residual error $\|r^m\|_F = \left\|\Delta_h \mu^{n+1,m} - (\phi^{n+1,m} - \phi^n)/\Delta t\right\|_F$. The grid sizes are set as 32×32, 64×64 and 128×128. The error norms and ratios of residual between successive V-cycle are shown in Table 1. As we have expected, the residual error decreases successively along with the V-cycle. The sharp increase in the residual norm ratio during the last few cycles reflects the fact that the numerical approximation is already accurate to near machine precision.

Table 1. Error and convergence results for various grid spaces.

Mesh Size	**32 × 32**		**64 × 64**		**128 × 128**	
V-Cycle	$\|r\|_F$	**Ratio**	$\|r\|_F$	**Ratio**	$\|r\|_F$	**Ratio**
1	5.28×10^{-2}		6.83×10^{-2}		9.10×10^{-2}	
2	2.41×10^{-3}	0.05	3.42×10^{-3}	0.05	4.30×10^{-3}	0.05
3	1.15×10^{-4}	0.05	1.56×10^{-4}	0.05	2.09×10^{-4}	0.05
4	6.54×10^{-6}	0.06	7.25×10^{-6}	0.05	8.69×10^{-6}	0.04
5	4.23×10^{-7}	0.06	4.23×10^{-7}	0.06	4.66×10^{-7}	0.05
6	2.80×10^{-8}	0.07	2.65×10^{-8}	0.06	2.90×10^{-8}	0.06
7	1.83×10^{-9}	0.07	1.58×10^{-9}	0.06	1.63×10^{-9}	0.06
8	1.23×10^{-10}	0.07	1.01×10^{-10}	0.06	1.01×10^{-10}	0.06
9	8.33×10^{-12}	0.07	6.75×10^{-12}	0.07	6.83×10^{-12}	0.07
10	5.46×10^{-13}	0.07	4.24×10^{-13}	0.06	4.42×10^{-13}	0.06
11	3.70×10^{-14}	0.07	4.90×10^{-14}	0.12	1.69×10^{-13}	0.38
12	1.10×10^{-14}	$\boxed{0.30}$	4.10×10^{-14}	$\boxed{0.84}$	1.66×10^{-13}	$\boxed{0.98}$

3.4. Effect of Tolerance

The effect of multigrid tolerance is related to the average mass convergence. We set the initial condition $\phi(x,y,0) = 0.1\cos(\pi x)\cos(\pi y)$ on $\Omega = (0,1) \times (0,1)$ with tolerance, $tol = 1.0 \times 10^{-1}$, 1.0×10^{-2}, 1.0×10^{-3}, and 1.0×10^{-10} to investigate the relationship between the mass convergence and tol. We use the simulation parameters $\epsilon = \epsilon_4$, SMOOTH relaxation = 2, $h = 1/32$, $\Delta t = 1/32$ and mesh size 32×32. To compare the theoretical value (solid line) with the computational value $tol = 1.0 \times 10^{-1}$ (dotted line), $tol = 1.0 \times 10^{-2}$ (dash-dot line), $tol = 1.0 \times 10^{-3}$ (dashed line) and $tol = 1.0 \times 10^{-10}$ (square), we set the interval of average mass from -0.0019 to 0.0023. In Figure 7, the average mass $\mathcal{M}^h(\phi^n)/(h^2 N_x N_y)$ gradually converges to a theoretical value with the decrease in tolerance. In addition, comparing the results of $tol = 1.0 \times 10^{-1}$, 1.0×10^{-2}, 1.0×10^{-3} and 1.0×10^{-10}, we observe that the average mass become nearly convergent for $tol = 1.0 \times 10^{-10}$.

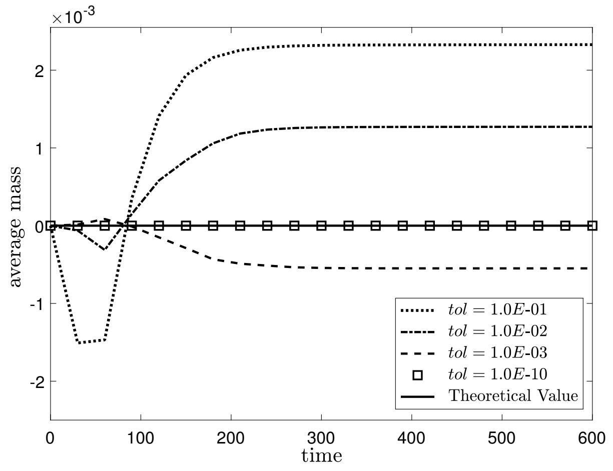


Figure 7. Average mass $\mathcal{M}^h(\phi^n)/(h^2 N_x N_y)$ of the numerical solutions in various values of tolerance. Here, the theoretical value (solid line), *tol* = 1.0×10^{-1} (dotted line), *tol* = 1.0×10^{-2} (dash-dot line), *tol* = 1.0×10^{-3} (dashed line) and *tol* = 1.0×10^{-10} (square).

3.5. Effects of the Smooth Relaxation Numbers ν_1 and ν_2

We investigate the effects of the SMOOTH relaxation numbers ν_1 (pre-relaxation) and ν_2 (post-relaxation) on the CPU time. In this test, we perform a numerical simulation with the initial condition $\phi(x,y,0) = 0.1\cos(\pi x)\cos(\pi y)$ on $\Omega = (0,1) \times (0,1)$, $h = 1/128$, ϵ_4, $\Delta t = 0.1h^2$ and *tol* = 1.0×10^{-10}. Table 2 lists the average CPU times and average numbers of V-cycles for various pre- and post-relaxation numbers after 100 time steps. The relaxation numbers are rounded off to the nearest integer. Figure 8 shows the average CPU times for different pre- and post-relaxation numbers. We observe that the average CPU time is the lowest when the numbers of pre- and post-relaxation iterations are $\nu_1 = 2$ and $\nu_2 = 4$, respectively.

Table 2. Average CPU times and average numbers of V-cycles (given in parentheses) for various pre- and post-relaxation numbers after 100 time steps.

ν_2 \ ν_1	**1**	**2**	**3**	**4**	**5**
1	0.075(9)	0.075(7)	0.081(6)	0.077(5)	0.089(5)
2	0.075(7)	0.079(6)	0.090(5)	0.089(5)	0.081(4)
3	0.079(6)	0.082(5)	0.089(5)	0.081(4)	0.090(4)
4	0.078(5)	0.075(4)	0.081(4)	0.090(4)	0.075(3)
5	0.072(4)	0.081(4)	0.090(4)	0.075(3)	0.082(3)

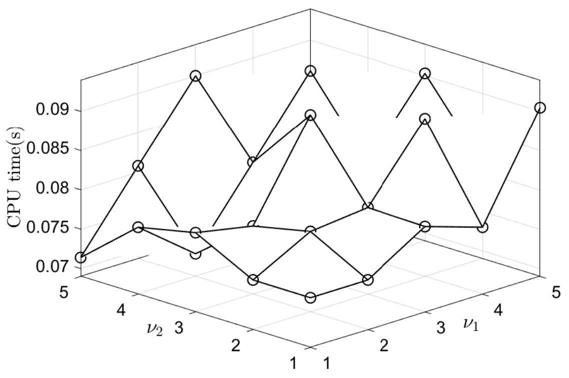


Figure 8. Average CPU time for different pre- and post-relaxation numbers after 100 time steps.

Next, we investigate the effect of SMOOTH relaxation numbers on the finest multigrid level. We perform a numerical simulation with $\epsilon = 0.0038$ and $\Delta t = 1.0 \times 10^{-7}$. The SMOOTH relaxation numbers ν_0^1 and ν_0^2 (on the finest multigrid level) are taken to be 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10. In addition, other multigrid levels $\nu_k^1 = \nu_k^2$ are 1, 2, and 3 in the 128×128 and 512×512 mesh sizes. The other parameters are the same as those used previously. Table 3 shows the variations in average CPU time for different relaxation numbers with a 128×128 mesh size. Figure 9 illustrates the results in Table 3.

Table 3. Average CPU times for various relaxation numbers on finest multigrid level (ν_0^1, ν_0^2) after 10 time steps. In other grids, $\nu_k^1 = \nu_k^2, (1 \le k)$ relaxation number is fixed at 1, 2 and 3.

128 × 128		$\nu_k^1 = \nu_k^2$	
$\nu_0^1 = \nu_0^2$	**1**	**2**	**3**
1	0.043	0.049	0.054
2	0.040	0.045	0.048
3	0.027	0.029	0.031
4	0.032	0.034	0.037
5	0.038	0.041	0.042
6	0.044	0.047	0.048
7	0.050	0.053	0.054
8	0.055	0.058	0.060
9	0.062	0.065	0.065
10	0.068	0.070	0.072

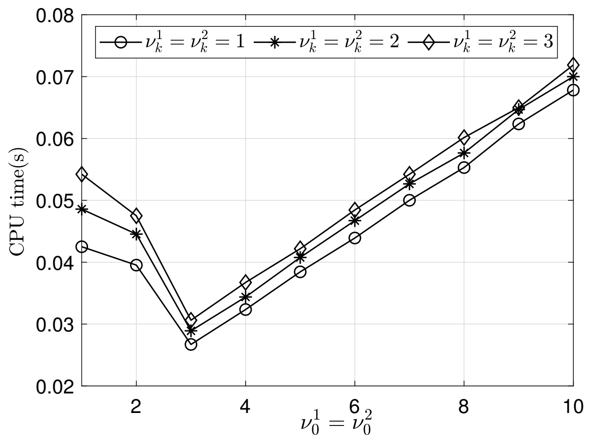


Figure 9. Average CPU times for various relaxation numbers on finest multigrid level (ν_0^1, ν_0^2) and fixed relaxation number 1, 2 and 3 on other grids with a 128×128 mesh size.

Table 4 lists the variations in average CPU time for different relaxation numbers with a 512×512 mesh size. Figure 10 illustrates the results in Table 4.

Table 4. Average CPU times for various relaxation numbers on finest multigrid level (ν_0^1, ν_0^2) after 10 time steps. In other grids, the $\nu_k^1 = \nu_k^2, (1 \le k)$ relaxation number is fixed at 1, 2 and 3.

512 × 512		$\nu_k^1 = \nu_k^2$	
$\nu_0^1 = \nu_0^2$	**1**	**2**	**3**
1	2.182	2.506	2.750
2	2.030	2.208	2.397
3	1.726	1.850	1.982
4	2.096	2.227	2.351
5	1.872	1.943	2.047
6	2.151	2.236	2.330
7	2.426	2.514	2.617
8	1.817	1.877	1.944
9	2.002	2.066	2.118
10	2.197	2.252	2.295

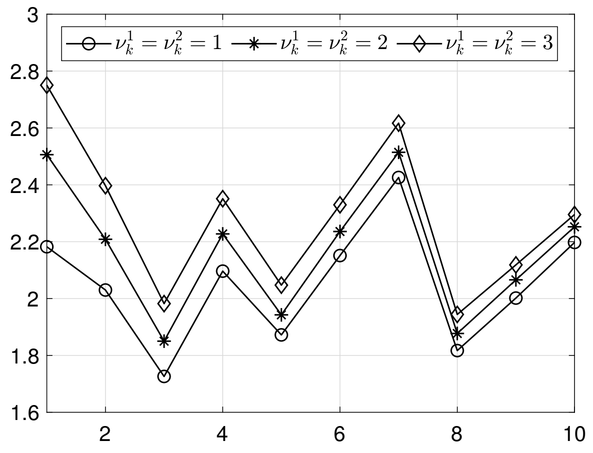


Figure 10. Average CPU times for various relaxation numbers on finest multigrid level (ν_0^1, ν_0^2) and fixed relaxation numbers 1, 2 and 3 on other grids with a 512×512 mesh size.

3.6. Effect Of V-Cycle

Next, we investigate the effect of V-cycle by changing the multigrid levels. In this test, we use the parameters $\Delta t = 0.01$, $\epsilon = 0.06$, SMOOTH relaxation $= 2$, $tol = 1.0 \times 10^{-10}$ and $h = 1/128$ on $\Omega = (0,1) \times (0,1)$ with a initial condition $\phi(x,y,0) = 0.1\cos(\pi x)\cos(\pi y)$. The highest number of V-cycles is taken to be 10000. We use $level_2$, $level_3$, $level_4$, $level_5$, $level_6$ and $level_7$ in a single time step as examples to illustrate the effect of the V-cycle. We calculate the CPU time for each level after $100\Delta t$, as listed in Table 5.

Table 5. Numbers of multigrid levels and CPU times required until tolerance $\leq 1.0 \times 10^{-10}$. Here, different levels are used.

Level	**2**	**3**	**4**	**5**	**6**	**7**
CPU time(s)	1271.437	342.906	103.032	66.422	66.172	65.218

The number of the multigrid level and CPU time shown in Figure 11 indicate that a greater number of the multigrid level leads to a obvious decrease in CPU time. It is important to select an appropriate multigrid level for a specific mesh size.

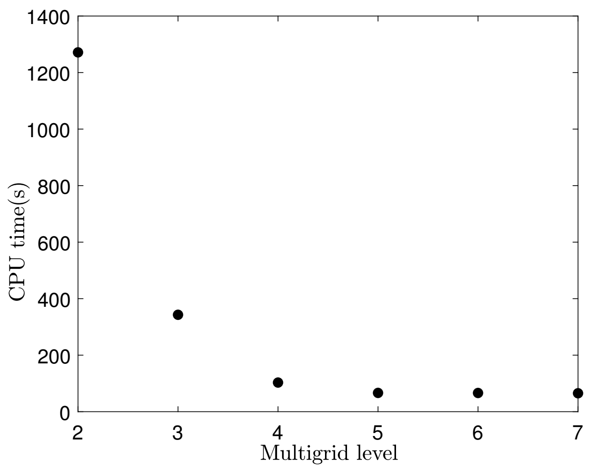


Figure 11. Required CPU times in various multigrid levels after 100 time steps.

3.7. Comparison between Gauss–Seidel and Multigrid Algorithms

We compare the average CPU times required to perform the Gauss–Seidel algorithm and multigrid algorithm. In this test, the initial condition is taken to be $\phi(x,y,0) = \cos(\pi x)\cos(\pi y)$. The following parameters are used—$\Delta t = 0.01$, $T = 10\Delta t$, the SMOOTH relaxation $= 2$ and $\epsilon = 0.06$. The highest number of V-cycle is taken to be 10000. The mesh sizes are 32×32, 64×64 and 128×128. The tolerances are 1.0×10^{-3}, 1.0×10^{-4} and 1.0×10^{-5}. Table 6 shows the average CPU times for these two methods after 10 time steps. We observe that the multigrid method require less CPU time than the Gauss–Seidel method does.

Table 6. Average CPU times for Gauss–Seidel and multigrid algorithms with different tolerances after 10 time steps.

Mesh Size	***tol***	**Gauss–Seidel**	**Multigrid**
32×32	1.0×10^{-3}	0.468	0.046
	1.0×10^{-4}	0.610	0.063
	1.0×10^{-5}	0.735	0.062
64×64	1.0×10^{-3}	7.046	0.203
	1.0×10^{-4}	8.984	0.234
	1.0×10^{-5}	11.360	0.266
128×128	1.0×10^{-3}	109.093	0.844
	1.0×10^{-4}	140.219	0.906
	1.0×10^{-5}	174.922	1.078

3.8. *Effects of tol and* Δt *on the V-Cycle*

In this test, we study the effects of *tol* and Δt on the V-cycle with the initial condition being $\phi(x, y, 0) = 0.1\cos(\pi x)\cos(\pi y)$, $h = 1/128$, $\epsilon = 0.06$. The highest number of the V-cycle is taken to be 10,000. Table 7 shows the number of V-cycle for various *tol* and Δt after a single time step. We can find that lower values of *tol* lead to an increase in the V-cycle. For different values of *tol*, it is essential to choose an appropriate Δt to reduce the number of V-cycle.

Table 7. Numbers of V-cycles for various *tol* and Δt after a single time step.

Δt / *tol*	$10^{-7}h^2$	$10^{-6}h^2$	$10^{-5}h^2$	$10^{-4}h^2$	$10^{-3}h^2$	$10^{-2}h^2$	$10^{-1}h^2$
1.0×10^{-5}	2	2	3	4	6	8	8
1.0×10^{-6}	10,000	2	3	4	7	9	9
1.0×10^{-7}	10,000	10,000	3	5	8	10	10
1.0×10^{-8}	10,000	10,000	10,000	5	9	11	12
1.0×10^{-9}	10,000	10,000	10,000	10,000	10	13	13
1.0×10^{-10}	10,000	10,000	10,000	10,000	10,000	14	15
Δt / *tol*	h^2	$10h^2$	10^2h^2	10^3h^2	10^4h^2	10^5h^2	10^6h^2
1.0×10^{-5}	8	7	8	8	9	14	41
1.0×10^{-6}	9	9	9	9	10	19	91
1.0×10^{-7}	11	10	10	11	11	24	140
1.0×10^{-8}	12	12	12	12	13	28	189
1.0×10^{-9}	14	13	13	13	14	33	238
1.0×10^{-10}	15	14	15	15	16	38	288

3.9. *Comparison of the Jacobi, Red–Black and Gauss–Seidel*

We compare the performance of three relaxation methods: Jacobi, Red–Black and Gauss–Seidel. The initial condition is $\phi(x, y, 0) = 0.1\cos(\pi x)\cos(\pi y)$ on $\Omega = (0, 1) \times (0, 1)$. The parameters are $h = 1/128$, $\epsilon = 0.06$, $\Delta t = 10^{-7}$, $T = 100\Delta t$ and $tol = 1.0 \times 10^{-10}$. The SMOOTH relaxation numbers on the finest multigrid level (i.e., $\nu^1 = \nu^2$) are taken to be from 1 to 5 and those on the other multigrid levels are selected as 2 with the 128×128 mesh size. The relaxation numbers are rounded off to the nearest integer. Table 8 shows the average number of V-cycles for different relaxation numbers with the three methods. The relationship between the average numbers of V-cycles and $\nu^1 = \nu^2$ with the Jacobi, Red–Black and Gauss–Seidel method is plotted in Figure 12. The Gauss–Seidel method is observed to be the fastest. In the parallel multigrid method, the relaxation options are either Jacobi or Red–Black [33]. The Jacobi method requires approximately twice as many V-cycles as the Red–Black method does.

Table 8. Average numbers of V-cycles for various relaxation numbers. The SMOOTH relaxation numbers on the finest multigrid level (i.e., $\nu^1 = \nu^2$) are taken to be from 1 to 5.

Case	Jacobi	Red–Black	Gauss-Seidal
1	32	13	9
2	16	8	6
3	11	6	5
4	8	5	4
5	7	5	3

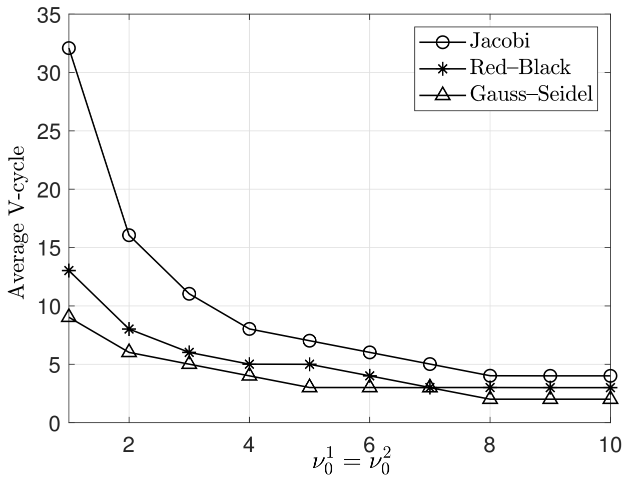


Figure 12. Plot of the average numbers of V-cycles versus $\nu^1 = \nu^2$ with the Jacobi ($\circ$), Red–Black ($*$) and Gauss–Seidel ($\triangle$) method.

3.10. Effect of ϵ

Next, we investigate the effect of $\epsilon = \epsilon_m$, which is related to the interface width. In this test, we perform a numerical simulation with the initial condition

$$\phi(x,y,0) = \begin{cases} 1 & \text{if } 0.15 \le x \le 0.85 \text{ and } 0.15 \le y \le 0.85, \\ -1 & \text{otherwise,} \end{cases}$$

on $\Omega = (0,1) \times (0,1)$. We use $h = 1/128$, $\Delta t = h$, SMOOTH relaxation $= 2$, $tol = 1.0 \times 10^{-10}$ and $T = 1000\Delta t$. Figure 13 presents the evolution of the CH equation with the three values ϵ_4, ϵ_8 and ϵ_{16}. As we have expected, the lower value of ϵ leads to a narrower interface width.

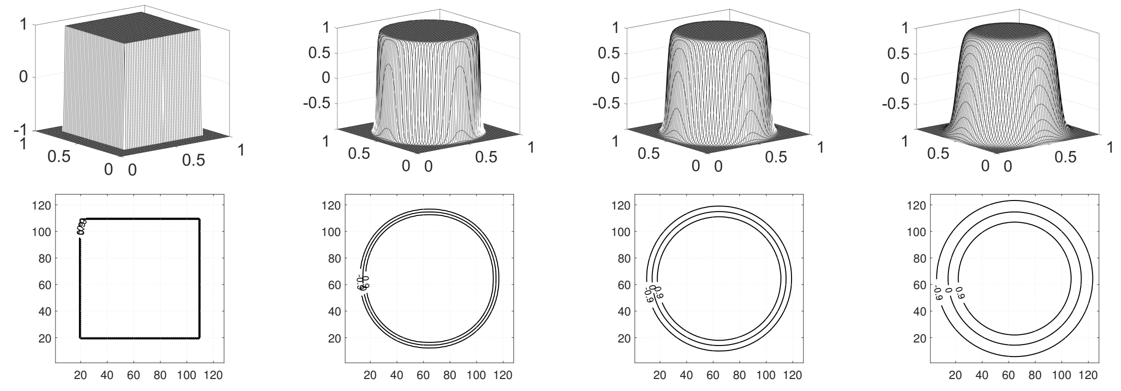

(**a**) initial (**b**) $m = 4$ (**c**) $m = 8$ (**d**) $m = 16$

Figure 13. Evolution of the Cahn–Hilliard (CH) equation with different $\epsilon = \epsilon_m$: (**a**) initial condition, (**b**–**d**) $m = 4$, $m = 8$ and $m = 16$ at $T = 1000\Delta t$, respectively.

3.11. Effect of mesh size, $N_x \times N_y$

In this test, we compare the CPU times with different mesh sizes $N_x \times N_y$. The initial condition is $\phi(x, y, 0) = 0.1\cos(\pi x)\cos(\pi y)$ on $\Omega = (0, N_x/32) \times (0, N_y/32)$. The parameters are $h = 1/32$, $\Delta t = h$, $T = 100\Delta t$, $\epsilon = 0.06$, SMOOTH relaxation $= 2$ and $tol = 1.0 \times 10^{-10}$. Table 9 shows the CPU times and their ratios (that is, the ratio of the CPU time with the mesh size $2N_x \times 2N_y$ to the CPU time with $N_x \times N_y$). We observe that the values converge to 4.

Table 9. CPU times for different mesh sizes.

Mesh Size	**32 × 32**		**64 × 64**		**128 × 128**		**256 × 256**
CPU time(s)	0.610		2.812		12.156		50.594
Ratio		4.610		4.323		4.162	

4. Conclusions

In this paper, we presented a nonlinear multigrid implementation for the CH equation in a two-dimensional space. Eyre's unconditionally gradient stable scheme was used to discretize the governing equation. The resulting discretizing equations were solved using the nonlinear multigrid method. We described the implementation of our numerical scheme in detail. We numerically showed the decrease in discrete total energy and the convergence of discrete total mass. We took a convergence test by studying the reductions in residual error on various mesh sizes in a single time step. The results of various numerical experiments were presented to demonstrate the effects of tolerance, SMOOTH relaxation, V-cycle and ϵ. The provided multigrid source code will be useful to beginners who needs the numerical implementation of the nonlinear multigrid method for the CH equation.

Author Contributions: All authors, C.L., D.J., J.Y., and J.K., contributed equally to this work and critically reviewed the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: The first author (C. Lee) was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2019R1A6A3A13094308). D. Jeong was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF-2017R1E1A1A03070953). J. Yang is supported by China Scholarship Council (201908260060). The corresponding author (J.S. Kim) expresses thanks for the support from the BK21 PLUS program.

Acknowledgments: The authors thank the editor and the reviewers for their constructive and helpful comments on the revision of this article.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

The *C* code and MATLAB postprocessing code are given as follows, and the parameters are enumerated in Table A1.

Table A1. Parameters used for the 2D Cahn–Hilliard equation.

Parameters	Description
nx, ny	maximum number of grid points in the x-, y-direction
n_level	number of multigrid level
c_relax	number of times being relax
dt	Δt
xleft, yleft	minimum value on the x-, y-axis
xright, yright	maximum value on the x-, y-axis
ns	number of print out data
max_it	maximum number of iteration
max_it_mg	maximum number of multigrid iteration
tol_mg	tolerance for multigrid
h	space step size
h2	h^2
gam	ϵ
Cahn	ϵ^2

The following *C* code is available on the following website:

http://elie.korea.ac.kr/~cfdkim/codes/

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <malloc.h>
#include <time.h>
#define gnx 32
#define gny 32
#define PI 4.0*atan(1.0)
#define iloop for(i=1;i<=gnx;i++)
#define jloop for(j=1;j<=gny;j++)
#define ijloop iloop jloop
#define iloopt for(i=1;i<=nxt;i++)
#define jloopt for(j=1;j<=nyt;j++)
#define ijloopt iloopt jloopt
int nx,ny,n_level,c_relax;
double **ct,**sc,**smu,**sor,h,h2,dt,xleft,xright,yleft,yright,gam,Cahn,**mu,**mi;
double **dmatrix(long nrl,long nrh,long ncl,long nch){
double **m;
long i,nrow=nrh-nrl+2,ncol=nch-ncl+2;
m=(double **) malloc((nrow)*sizeof(double*)); m+=1;m-=nrl;
m[nrl]=(double *) malloc((nrow*ncol)*sizeof(double)); m[nrl]+=1; m[nrl]-=ncl;
for (i=nrl+1; i<=nrh; i++) m[i]=m[i-1]+ncol;
return m;
}
void free_dmatrix(double **m,long nrl,long nrh,long ncl,long nch){
free(m[nrl]+ncl-1); free(m+nrl-1);
}
void zero_matrix(double **a,int xl,int xr,int yl,int yr){
int i,j;
for (i=xl; i<=xr; i++){ for (j=yl; j<=yr; j++){ a[i][j]=0.0; }}
}
```

```
void mat_add2(double **a,double **b,double **c,double **a2,
double **b2,double **c2,int xl,int xr,int yl,int yr){
int i,j;
for (i=xl; i<=xr; i++)
for (j=yl; j<=yr; j++){ a[i][j]=b[i][j]+c[i][j]; a2[i][j]=b2[i][j]+c2[i][j]; }
}
void mat_sub2(double **a,double **b,double **c,double **a2,
double **b2,double **c2,int nrl,int nrh,int ncl,int nch){
int i,j;
for (i=nrl;i<=nrh;i++)
for (j=ncl; j<=nch; j++){ a[i][j]=b[i][j]-c[i][j];a2[i][j]=b2[i][j]-c2[i][j]; }
}
void mat_copy(double **a,double **b,int xl,int xr,int yl,int yr){
int i,j;
for (i=xl; i<=xr; i++){ for (j=yl; j<=yr; j++){ a[i][j]=b[i][j]; }}
}
void mat_copy2(double **a,double **b,double **a2,double **b2,int xl,int xr,int yl,int yr){
int i,j;
for (i=xl; i<=xr; i++)
for (j=yl; j<=yr; j++){a[i][j]=b[i][j]; a2[i][j]=b2[i][j];}
}
void print_mat(FILE *fptr,double **a,int nrl,int nrh,int ncl,int nch){
int i,j;
for(i=nrl; i<=nrh; i++){ for(j=ncl; j<=nch; j++)
fprintf(fptr," %16.15f",a[i][j]); fprintf(fptr,"\n"); }
}
void print_data(double **phi) {
FILE *fphi;
fphi=fopen("phi.m","a"); print_mat(fphi,phi,1,nx,1,ny); fclose(fphi);
}
void laplace(double **a,double **lap_a,int nxt,int nyt){
int i,j;
double ht2,dadx_L,dadx_R,dady_B,dady_T;
ht2=pow((xright-xleft)/(double) nxt,2);
ijloopt {
if (i>1)   dadx_L=a[i][j]-a[i-1][j];
else       dadx_L=0.0;
if (i<nxt) dadx_R=a[i+1][j]-a[i][j];
else       dadx_R=0.0;
if (j>1)   dady_B=a[i][j]-a[i][j-1];
else       dady_B=0.0;
if (j<nyt) dady_T=a[i][j+1]-a[i][j];
else       dady_T=0.0;
lap_a[i][j]=(dadx_R-dadx_L+dady_T-dady_B)/ht2;}
}
void source(double **c_old,double **src_c,double **src_mu){
int i,j;
laplace(c_old,ct,nx,ny);
ijloop{src_c[i][j]=c_old[i][j]/dt-ct[i][j]; src_mu[i][j]=0.0;}
}
double df(double c){return pow(c,3);}
```

```
double d2f(double c){return 3.0*c*c;}
void relax(double **c_new,double **mu_new,double **su,double **sw,int ilevel,
int nxt, int nyt){
int i,j,iter;
double ht2,x_fac,y_fac,a[4],f[2],det;
ht2=pow((xright-xleft)/(double) nxt,2);
for (iter=1; iter<=c_relax; iter++){
ijloopt {
if (i>1 && i<nxt) x_fac=2.0;
else              x_fac=1.0;
if (j>1 && j<nyt) y_fac=2.0;
else              y_fac=1.0;
a[0]=1.0/dt; a[1]=(x_fac+y_fac)/ht2;
a[2]=-(x_fac+y_fac)*Cahn/ht2-d2f(c_new[i][j]); a[3]=1.0;
f[0]=su[i][j]; f[1]=sw[i][j]+df(c_new[i][j])-d2f(c_new[i][j])*c_new[i][j];
if (i>1) { f[0]+=mu_new[i-1][j]/ht2; f[1]-=Cahn*c_new[i-1][j]/ht2; }
if (i<nxt) { f[0]+=mu_new[i+1][j]/ht2; f[1]-=Cahn*c_new[i+1][j]/ht2; }
if (j>1) { f[0]+=mu_new[i][j-1]/ht2; f[1]-=Cahn*c_new[i][j-1]/ht2; }
if (j<nyt) { f[0]+=mu_new[i][j+1]/ht2; f[1]-=Cahn*c_new[i][j+1]/ht2; }
det=a[0]*a[3]-a[1]*a[2];
c_new[i][j]=(a[3]*f[0]-a[1]*f[1])/det;
mu_new[i][j]=(-a[2]*f[0]+a[0]*f[1])/det; }}
}
void restrictCH(double **uf,double **uc,double **vf,double **vc,int nxc,int nyc) {
int i,j;
for (i=1; i<=nxc; i++)
for (j=1; j<=nyc; j++){
uc[i][j]=0.25*(uf[2*i][2*j]+uf[2*i-1][2*j]+uf[2*i][2*j-1]+uf[2*i-1][2*j-1]);
vc[i][j]=0.25*(vf[2*i][2*j]+vf[2*i-1][2*j]+vf[2*i][2*j-1]+vf[2*i-1][2*j-1]);}
}
void nonL(double **ru,double **rw,double **c_new,double **mu_new,int nxt,int nyt) {
int i,j;
double **lap_mu,**lap_c;
lap_mu=dmatrix(1,nxt,1,nyt); lap_c=dmatrix(1,nxt,1,nyt);
laplace(c_new,lap_c,nxt,nyt); laplace(mu_new,lap_mu,nxt,nyt);
ijloopt{ ru[i][j]=c_new[i][j]/dt-lap_mu[i][j];
rw[i][j]=mu_new[i][j]-df(c_new[i][j])+Cahn*lap_c[i][j]; }
free_dmatrix(lap_mu,1,nxt,1,nyt); free_dmatrix(lap_c,1,nxt,1,nyt);
}
void defect(double **duc,double **dwc,double **uf_new,double **wf_new,double **suf,
double **swf,int nxf,int nyf,double **uc_new,double **wc_new,int nxc,int nyc) {
double **ruf,**rwf,**rruf,**rrwf,**ruc,**rwc;
ruc=dmatrix(1,nxc,1,nyc);rwc=dmatrix(1,nxc,1,nyc);ruf=dmatrix(1,nxf,1,nyf);
rwf=dmatrix(1,nxf,1,nyf);rruf=dmatrix(1,nxc,1,nyc);rrwf=dmatrix(1,nxc,1,nyc);
nonL(ruc,rwc,uc_new,wc_new,nxc,nyc);nonL(ruf,rwf,uf_new,wf_new,nxf,nyf);
mat_sub2(ruf,suf,ruf,rwf,swf,rwf,1,nxf,1,nyf);
restrictCH(ruf,rruf,rwf,rrwf,nxc,nyc);
mat_add2(duc,ruc,rruf,dwc,rwc,rrwf,1,nxc,1,nyc);
free_dmatrix(ruc,1,nxc,1,nyc); free_dmatrix(rwc,1,nxc,1,nyc);
free_dmatrix(ruf,1,nxf,1,nyf); free_dmatrix(rwf,1,nxf,1,nyf);
free_dmatrix(rruf,1,nxc,1,nyc); free_dmatrix(rrwf,1,nxc,1,nyc);
}
```

```
void prolong_ch(double **uc,double **uf,double **vc,double **vf, int nxc,int nyc){
int i,j;
for (i=1; i<=nxc; i++)
for (j=1; j<=nyc; j++){
uf[2*i][2*j]=uf[2*i-1][2*j]=uf[2*i][2*j-1]=uf[2*i-1][2*j-1]=uc[i][j];
vf[2*i][2*j]=vf[2*i-1][2*j]=vf[2*i][2*j-1]=vf[2*i-1][2*j-1]=vc[i][j];}
}
void vcycle(double **uf_new,double **wf_new,double **su,double **sw,int nxf,int nyf,
int ilevel) {
relax(uf_new,wf_new,su,sw,ilevel,nxf,nyf);
if (ilevel<n_level) {
int nxc,nyc;
double **duc,**dwc,**uc_new,**wc_new,**uc_def,**wc_def,**uf_def,**wf_def;
nxc=nxf/2; nyc=nyf/2;
duc=dmatrix(1,nxc,1,nyc); dwc=dmatrix(1,nxc,1,nyc);
uc_new=dmatrix(1,nxc,1,nyc); wc_new=dmatrix(1,nxc,1,nyc);
uf_def=dmatrix(1,nxf,1,nyf); wf_def=dmatrix(1,nxf,1,nyf);
uc_def=dmatrix(1,nxc,1,nyc); wc_def=dmatrix(1,nxc,1,nyc);
restrictCH(uf_new,uc_new,wf_new,wc_new,nxc,nyc);
defect(duc,dwc,uf_new,wf_new,su,sw,nxf,nyf,uc_new,wc_new,nxc,nyc);
mat_copy2(uc_def,uc_new,wc_def,wc_new,1,nxc,1,nyc);
vcycle(uc_def,wc_def,duc,dwc,nxc,nyc,ilevel+1);
mat_sub2(uc_def,uc_def,uc_new,wc_def,wc_def,wc_new,1,nxc,1,nyc);
prolong_ch(uc_def,uf_def,wc_def,wf_def,nxc,nyc);
mat_add2(uf_new,uf_new,uf_def,wf_new,wf_new,wf_def,1,nxf,1,nyf);
relax(uf_new,wf_new,su,sw,ilevel,nxf,nyf);
free_dmatrix(duc,1,nxc,1,nyc); free_dmatrix(dwc,1,nxc,1,nyc);
free_dmatrix(uc_new,1,nxc,1,nyc); free_dmatrix(wc_new,1,nxc,1,nyc);
free_dmatrix(uf_def,1,nxf,1,nyf); free_dmatrix(wf_def,1,nxf,1,nyf);
free_dmatrix(uc_def,1,nxc,1,nyc); free_dmatrix(wc_def,1,nxc,1,nyc); }
}
double error2(double **c_old,double **c_new,double **mu,int nxt,int nyt){
int i,j;
double **rr,res2,x=0.0;
rr=dmatrix(1,nxt,1,nyt);
ijloopt { rr[i][j]=mu[i][j]-c_old[i][j]; }
laplace(rr,sor,nx,ny);
ijloopt { rr[i][j]=sor[i][j]-(c_new[i][j]-c_old[i][j])/dt; }
ijloopt { x=(rr[i][j])*(rr[i][j])+x; }
res2=sqrt(x/(nx*ny));
free_dmatrix(rr,1,nxt,1,nyt);
return res2;
}
void initialization(double **phi){
int i,j;
```

```
double x,y;
ijloop {x=(i-0.5)*h; y=(j-0.5)*h; phi[i][j]=cos(PI*x)*cos(PI*y);}
}
void cahn(double **c_old,double **c_new){
FILE *fphi2;
int i,j,max_it_CH=10000,it_mg2=1;
```

```
double tol=1.0e-10, resid2=1.0;
source(c_old,sc,smu);
while (it_mg2<=max_it_CH && resid2>tol) {
it_mg2++; vcycle(c_new,mu,sc,smu,nx,ny,1);
resid2=error2(c_old,c_new,mu,nx,ny);
printf("error2 %16.15f %d \n",resid2,it_mg2-1);
fphi2=fopen("phi2.m","a");
fprintf(fphi2,"%16.15f %d \n",resid2,it_mg2-1); fclose(fphi2);}
}
int main(){
int it=1,max_it,ns,count=1,it_mg=1;
double **oc,**nc,resid2=1.0;
FILE *fphi,*fphi2;
c_relax=2; nx=gnx; ny=gny; n_level=(int)(log(nx)/log(2.0)+0.1);
xleft=0.0; xright=1.0; yleft=0.0; yright=1.0; max_it=100; ns=10; dt=0.01;
h=xright/(double)nx; h2=pow(h,2); gam=0.06; Cahn=pow(gam,2);
printf("nx=%d,ny=%d\n",nx,ny); printf("dt=%f\n",dt);
printf("max_it=%d\n",max_it); printf("ns=%d\n",ns); printf("n_level=%d\n\n",n_level);
oc=dmatrix(0,nx+1,0,ny+1); nc=dmatrix(0,nx+1,0,ny+1); mu=dmatrix(1,nx,1,ny);
sor=dmatrix(1,nx,1,ny); ct=dmatrix(1,nx,1,ny); sc=dmatrix(1,nx,1,ny);
mi=dmatrix(1,nx,1,ny); smu=dmatrix(1,nx,1,ny); zero_matrix(mu,1,nx,1,ny);
initialization(oc); mat_copy(nc,oc,1,nx,1,ny);
fphi=fopen("phi.m","w"); fclose(fphi); print_data(oc);
for (it=1; it<=max_it; it++) {
cahn(oc,nc); mat_copy(oc,nc,1,nx,1,ny);
if (it % ns==0) {count++; print_data(oc); printf("print out counts %d \n",count);}
printf(" %d \n",it);}
return 0;
}
```

The following MATLAB code produces the results shown in Figure 5. The code can also be downloaded from

http://elie.korea.ac.kr/~cfdkim/codes/

```
clear; clc; close all;
ss=sprintf('./phi.m'); phi=load(ss); nx=32; ny=32; n=size(phi,1)/nx;
x=linspace(0,1,nx); y=linspace(0,1,ny); [xx,yy]=meshgrid(x,y);
for i=1:n
pp=phi((i-1)*nx+1:i*nx,:);
figure(i); mesh(xx,yy,pp'); axis([0 1 0 1 -1 1]); view(-38,42);
end
```

References

1. Trottenberg, U.; Schüller, A.; Oosterlee, C.W. *Multigrid Methods*; Academic Press: Cambridge, MA, USA, 2000.
2. Cahn, J.W.; Hilliard, J.E. Free energy of a nonuniform system. I. Interfacial free energy. *J. Chem. Phys.* **1958**, *28*, 258–267. [CrossRef]
3. Copetti, M.I.M.; Elliott, C.M. Kinetics of phase decomposition processes: Numerical solutions to Cahn–Hilliard equation. *Mater. Sci. Technol.* **1990**, *6*, 273–284. [CrossRef]
4. Honjo, M.; Saito, Y. Numerical simulation of phase separation in Fe-Cr binary and Fe-Cr-Mo ternary alloys with use of the Cahn–Hilliard equation. *ISIJ Int.* **2000**, *40*, 914–919. [CrossRef]
5. Bertozzi, A.L.; Esedoglu, S.; Gillette, A. Inpainting of binary images using the Cahn–Hilliard equation. *IEEE Trans. Image Process.* **2007**, *16*, 285–291. [CrossRef]
6. Bosch, J.; Kay, D.; Stoll, M.; Wathen, A.J. Fast solvers for Cahn–Hilliard inpainting. *SIAM J. Imaging Sci.* **2014**, *7*, 67–97. [CrossRef]
7. Choksi, R.; Peletier, M.A.; Williams, J.F. On the phase diagram for microphase separation of diblock copolymers: An approach via a nonlocal Cahn–Hilliard functional. *SIAM J. Appl. Math.* **2009**, *69*, 1712–1738. [CrossRef]
8. Tang, P.; Qiu, F.; Zhang, H.; Yang, Y. Phase separation patterns for diblock copolymers on spherical surfaces: A finite volume method. *Phys. Rev. E* **2005**, *72*, 016710. [CrossRef] [PubMed]
9. Hu, S.Y.; Chen, L.Q. A phase-field model for evolving microstructures with strong elastic inhomogeneity. *Acta Mater.* **2001**, *49*, 1879–1890. [CrossRef]
10. Yu, P.; Hu, S.Y.; Chen, L.Q.; Du, Q. An iterative-perturbation scheme for treating inhomogeneous elasticity in phase-field models. *J. Comput. Phys.* **2005**, *208*, 34–50. [CrossRef]
11. Gurtin, M.E.; Polignone, D.; Vinals, J. Two-phase binary fluids and immiscible fluids described by an order parameter. *Math. Models Methods Appl. Sci.* **1996**, *6*, 815–831. [CrossRef]
12. Lee, T. Effects of incompressibility on the elimination of parasitic currents in the lattice Boltzmann equation method for binary fluids. *Comput. Math. Appl.* **2009**, *58*, 987–994. [CrossRef]
13. Jeong, D.; Kim, J. Phase-field model and its splitting numerical scheme for tissue growth. *Appl. Numer. Math.* **2017**, *117*, 22–35. [CrossRef]
14. Kim, J.; Lee, S.; Choi, Y.; Lee, S.M.; Jeong, D. Basic Principles and Practical Applications of the Cahn–Hilliard Equation. *Math. Probl. Eng.* **2016**, *2016*, 9532608. [CrossRef]
15. Colli, P.; Gilardi, G.; Sprekels, J. A distributed control problem for a fractional tumor growth model. *Mathematics* **2019**, *7*, 792. [CrossRef]
16. Myśliński, A.; Wroblewski, M. Structural optimization of contact problems using Cahn–Hilliard model. *Comput. Struct.* **2017**, *180*, 52–59. [CrossRef]
17. Eyre, D.J. Unconditionally gradient stable time marching the Cahn–Hilliard equation. *MRS Proc.* **1998**, *529*, 39. [CrossRef]
18. Yang, S.; Lee, H.; Kim, J. A phase-field approach for minimizing the area of triply periodic surfaces with volume constraint. *Comput. Phys. Commun.* **2010**, *181*, 1037–1046. [CrossRef]
19. Kim, J. A numerical method for the Cahn–Hilliard equation with a variable mobility. *Commun. Nonlinear Sci. Numer. Simul.* **2007**, *12*, 1560–1571. [CrossRef]
20. Shin, J.; Jeong, D.; Kim, J. A conservative numerical method for the Cahn–Hilliard equation in complex domains. *J. Comput. Phys.* **2011**, *230*, 7441–7455. [CrossRef]
21. Jeong, D.; Kim, J. A practical numerical scheme for the ternary Cahn–Hilliard system with a logarithmic free energy. *Phys. A* **2016**, *442*, 510–522. [CrossRef]
22. Lee, H.; Kim, J. A second-order accurate non-linear difference scheme for the N-component Cahn–Hilliard system. *Phys. A* **2008**, *387*, 4787–4799. [CrossRef]
23. Lee, H.G.; Shin, J.; Lee, J.Y. A High-Order Convex Splitting Method for a Non-Additive Cahn–Hilliard Energy Functional. *Mathematics* **2019**, *7*, 1242. [CrossRef]
24. Shin, J.; Choi, Y.; Kim, J. An unconditionally stable numerical method for the viscous Cahn–Hilliard equation. *Discret. Contin. Dyn. Syst. Ser. B* **2014**, *19*, 1737–1747. [CrossRef]
25. Kim, J. Phase-field models for multi-component fluid flows. *Commun. Comput. Phys.* **2012**, *12*, 613–661. [CrossRef]

26. Kim, J.; Bae, H.O. An unconditionally gradient stable adaptive mesh refinement for the Cahn–Hilliard equation. *J. Korean Phys. Soc.* **2008**, *53*, 672–679. [CrossRef]
27. Wise, S.; Kim, J.; Lowengrub, J. Solving the regularized, strongly anisotropic Cahn–Hilliard equation by an adaptive nonlinear multigrid method. *J. Comput. Phys.* **2007**, *226*, 414–446. [CrossRef]
28. Li, Y.; Jeong, D.; Shin, J.; Kim, J. A conservative numerical method for the Cahn–Hilliard equation with Dirichlet boundary conditions in complex domains. *Comput. Math. Appl.* **2013**, *65*, 102–115. [CrossRef]
29. Lee, H.G.; Kim, J. Accurate contact angle boundary conditions for the Cahn–Hilliard equations. *Comput. Fluids* **2011**, *44*, 178–186. [CrossRef]
30. Shin, J.; Kim, S.; Lee, D.; Kim, J. A parallel multigrid method of the Cahn–Hilliard equation. *Comput. Mater. Sci.* **2013**, *71*, 89–96. [CrossRef]
31. Lee, C.; Jeong, D.; Shin, J.; Li, Y.; Kim, J. A fourth-order spatial accurate and practically stable compact scheme for the Cahn–Hilliard equation. *Phys. A* **2014**, *409*, 17–28. [CrossRef]
32. Choi, J.W.; Lee, H.G.; Jeong, D.; Kim, J. An unconditionally gradient stable numerical method for solving the Allen–Cahn equation. *Phys. A* **2009**, *388*, 1791–1803. [CrossRef]
33. Baker, A.H.; Falgout, R.D.; Kolev, T.V.; Yang, U.M. Scaling hypre's multigrid solvers to 100,000 cores. In *High-Performance Scientific Computing*; Springer: London, UK, 2012; pp. 261–279.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).