Journal of Computational Physics: X 2 (2019) 100001

Contents lists available at ScienceDirect

Journal of Computational Physics: X

www.elsevier.com/locate/jcpx

Computing dynamics of thin films via large scale GPU-based simulations

Michael-Angelo Y.-H. Lam [1,2,4], Linda J. Cummings [1,2], Lou Kondic [*,1,2,3,4]

Department of Mathematical Sciences, New Jersey Institute of Technology, Newark, NJ, 07102, USA

ARTICLE INFO

Article history:
Received 19 June 2018
Received in revised form 4 December 2018
Accepted 5 December 2018
Available online 18 December 2018

Keywords:
Thin films
Long-wave approximation
Finite difference simulations
GPU computing
Film instabilities

ABSTRACT

We present the results of large scale simulations of 4th order nonlinear partial differential equations of diffusion type that are typically encountered when modeling dynamics of thin fluid films on substrates. The simulations are based on the alternate direction implicit (ADI) method, with the main part of the computational work carried out in the GPU computing environment. Efficient and accurate computations allow for simulations on large computational domains in three spatial dimensions (3D) and for long computational times. We apply the methods developed to the particular problem of instabilities of thin fluid films of nanoscale thickness. The large scale of the simulations minimizes the effects of boundaries, and also allows for simulating domains of the size encountered in published experiments. As an outcome, we can analyze the development of instabilities with an unprecedented level of detail. A particular focus is on analyzing the manner in which instability develops, in particular regarding differences between spinodal and nucleation types of dewetting for linearly unstable films, as well as instabilities of metastable films. Simulations in 3D allow for consideration of some recent results that were previously obtained in the 2D geometry [28]. Some of the new results include using Fourier transforms as well as topological invariants (Betti numbers) to distinguish the outcomes of spinodal and nucleation types of instabilities, describing in precise terms the complex processes that lead to the formation of satellite drops, as well as distinguishing the shape of the evolving film front in linearly unstable and metastable regimes. We also discuss direct comparison between simulations and available experimental results for nematic liquid crystal and polymer films.

0. Introduction

Simulating the dynamics of thin fluid films on substrates presents significant computational challenges, requiring consideration of complex setups that include evolving free surfaces and computational domains. Solving such problems using full numerical simulations of Navier–Stokes equations coupled with complex boundary conditions is computationally very

* Corresponding author.
E-mail address: kondic@njit.edu (L. Kondic).

[1] Research supported by the National Science Foundation under grant DMS-1211713.
[2] Research supported by the National Science Foundation under grant DMS-1815613.
[3] Research supported by the National Science Foundation under grant CBET-1604351.
[4] Research supported by the National Aeronautics and Space Administration under grant No. NNX16AQ79G.

https://doi.org/10.1016/j.jcpx.2018.100001

expensive. While there has been significant progress based on Volume of Fluid, phase field and related methods [1–7], it is desirable to be able to consider evolving thin films in a simpler and cheaper manner.

Motivated by the desire to reduce computational cost, and even to gain some insight based on analytical methods, thin fluid films are often considered using asymptotic approaches, in particular based on the long wave approximation. Such an approach, in its simplest form, leads to a highly nonlinear 4th order partial differential equation (PDE) of diffusion type (parabolic), which must then be solved numerically. While this is a much simpler task than that presented by the original problem, it still leads to significant computational challenges, particularly in configurations involving contact lines. Contact lines introduce short length scales in the problem, that result from the need to include additional physics resulting from liquid–solid interaction forces. These short length scales need to be resolved for the purpose of producing accurate results, leading to (in a finite difference setting) a large number of grid points. For evolving problems, the resulting outcome is still computationally expensive, typically limiting the simulations to relatively small computational domains and short times. Particularly when film instabilities are considered, simulating small computational domains is a significant restriction, since it is not obvious to which degree computational domain boundaries influence the results.

Numerical approaches that have been applied to the 4th order parabolic nonlinear partial differential equation (PDE) are numerous and we will not attempt to provide a comprehensive review. Only selected examples are mentioned, with a focus on the three dimensional setting that is of interest here. There is a number of finite difference based methods [8–12], pseudo-spectral approaches [13], and finite element type of methods [14]. Within finite difference based computations, significant progress has been achieved using the ADI (Alternate Direction Implicit) approach, which allows for efficient and accurate simulations, with good stability properties. Such an approach was discussed within the context of thin films [15] and later applied to a number of different problems, e.g. [11,16,17]. The main computational expense in this approach consists of solving a large number of linear algebra problems, involving in particular large sparse (pentadiagonal) matrices.

In this paper, we present a fast numerical method using a Graphics Processing Unit (GPU) and the Compute Unified Device Architecture (CUDA) Application Programming Interface (API). Using this approach, the most time consuming parts of the simulations are carried out on a GPU, producing as an outcome simulations that are more than an order of magnitude faster than a similarly coded serial Central Processing Unit (CPU) method.

The newly developed computational methods allow for consideration of problems that until now have been out of reach using reasonable computational resources. In this work we will focus on problems involving instabilities of thin films on the nanoscale, but it should be noted that the presented computational approach is quite general, and could be applied to a number of problems within the context of thin films, and also other physical problems that reduce to similar mathematical formulations that lead to the Cahn–Hilliard and Kuramoto–Sivashinsky type of equations.

Returning to the discussion of thin films, we note that such films are often unstable due to destabilizing liquid–solid interaction forces. The nature of these instabilities has been considered extensively in a variety of settings including polymer films [8,18,19], liquid crystal films [20–22], as well as liquid metals [23,24] (see also [25,26] for excellent reviews of this topic). The instability development leads to many questions even in relatively simple settings. One question involves differences between the instabilities due to the presence of a free surface, and those due to localized perturbations; the latter could be present due to some imperfection of either the fluid, or the substrate, or both. Free surface perturbations are typically global in character, random, and small in size (compared to the film thickness); the instabilities due to their presence are typically classified as spinodal dewetting, with the name evolving from mathematically similar spinodal decomposition. Localized perturbations, on the other hand, are typically large and lead to so-called nucleation type instability. One question is how to compare the instabilities resulting from these two mechanisms. While elaborate approaches based on Minkowski functionals have proved useful [8], one wonders whether, given a sufficiently large computational domain, one could analyze the results using some more straightforward approach. We will discuss one possible approach in this work.

There are additional aspects of thin film instabilities that require large scale simulations to resolve. One could ask, for example, whether the drop size resulting from film breakup due to spinodal and nucleation type instabilities are comparable. This question is of relevance to applications where instability is harnessed to produce drops of a specific size for use in some application, such as (for example) plasmonic resonance in the context of metal films [27]. Turning this around, if one is interested in the source of instability, one could analyze the resulting drop sizes, and use this information to infer the source. Furthermore, one could ask whether instability and resulting film breakup lead to the formation of small drops, often called satellite drops [28,29]. We will see that some of our findings could serve as a basis for answering these questions.

Clearly, a number of questions could be asked. We will focus on particular examples of nematic liquid crystal films and, to a smaller extent, on polymer films; however, the main features of our results are general, and will be of relevance to a number of problems involving thin films. What distinguishes different films, substrates and film thickness regimes, at least for slow evolution where inertial effects are not significant, is essentially the form of (effective) disjoining pressure that, in addition to liquid/solid interaction, may include the effects of anchoring in the context of liquid crystals; interactions of electric or magnetic type in the context of ferrofluids [29]; or composite substrates in the case of polymer films [18]. The influence of the functional form of such effective disjoining pressure on film stability was discussed in two spatial dimensions (2D) recently [28]. In that work, we presented numerical evidence for the formation of secondary (satellite) drops for positive values of the effective disjoining pressure, and discussed various regimes of instability development within linearly unstable as well as metastable regimes. The present paper will discuss some of these results in the 3D geometry.

The rest of this paper is organized as follows. Section 1 provides the basic mathematical formulation of the problems to be considered. The numerical methods are discussed in § 2, and the performance of the presented method in § 3. The

aspects related to implementation on a GPU are relegated to Appendix A. An application specific to nematic liquid crystal (NLC) films is discussed in § 4, and comparison to experimental results for both NLC and polymer films in § 5. We present our conclusions in § 6.

1. Governing equation

We consider equations describing nonlinear diffusion of the following general form

$$u_t + \nabla \cdot \left[f_0(u) \nabla \nabla^2 u + f_1(u) \nabla u \right] = 0 \,, \tag{1.1}$$

where $u(x, y, t)$ is the evolving quantity of interest; and $f_0(u)$, $f_1(u)$ are some smooth nonlinear functions of u. The square bracketed term may be interpreted as the flux that governs the diffusion process. Governing equations of the form stated in (1.1) appear in a variety of models, such as those leading to Cahn–Hilliard, Kuramoto–Sivashinsky and related equations.

1.1. Description of model problems

Equations of the form (1.1) commonly appear in the context of thin fluid films. In this context, the variable u describes the film thickness, h. For future reference, we specify here the governing equation for thin films in terms of (hatted) dimensional variables

$$\mu \hat{h}_{\hat{t}} + \hat{\nabla} \cdot \left[\hat{f}_0(\hat{h}) \hat{\nabla} \hat{\nabla}^2 \hat{h} + \hat{f}_1(\hat{h}) \hat{\nabla} \hat{h} \right] = 0 \,, \tag{1.2}$$

where $\hat{h}(\hat{x}, \hat{y}, \hat{t})$ is the fluid film thickness and μ is the viscosity. To nondimensonalize (1.2), four scaling factors are defined: H, a representative film thickness scale, L, the typical lengthscale of variations in the plane of the film, $(\hat{x}, \hat{y})$; $\delta = H/L \ll 1$, the small aspect ratio; and T, the timescale of fluid flow. Scaling $(\hat{x}, \hat{y})$, $\hat{t}$ and $\hat{h}$ in the obvious way, (1.2) becomes

$$h_t + \nabla \cdot \left[f_0(h) \nabla \nabla^2 u + f_1(h) \nabla h \right] = 0 \,, \quad \text{with} \quad f_0 = \frac{T F_0}{\mu L^4} \tilde{f}_0(h) \quad \text{and} \quad f_1 = \frac{T F_1}{\mu L^2} \tilde{f}_1(h) \,, \tag{1.3}$$

where F_0 and F_1 are some positive dimensional prefactors associated with $\hat{f}_0$ and $\hat{f}_1$, respectively, such that $\tilde{f}_0$ and $\tilde{f}_1$ are nondimensional functions of the dimensionless film height h, for example, $\hat{f}_0(\hat{h}) = F_0 \tilde{f}_0(h)$. The prefactors of $\tilde{f}_0$ and $\tilde{f}_1$ in (1.3) are nondimensional; therefore, for simplicity, we absorb these prefactors into the definitions of the relevant functions.

1.2. Linear Stability Analysis (LSA)

We now present a brief overview of the Linear Stability Analysis (LSA) of a flat film of thickness H_0 in 2D (two dimensions), which will be used to validate our numerical code. To derive the dispersion relation, the solution is assumed to be of the form $h(x, t) = H_0(1 + \epsilon e^{iqx + i\omega t})$, where $\epsilon \ll 1$. Substituting this ansatz into (1.1) yields

$$\omega = i \left[f_1(H_0) q^2 - f_2(H_0) \right] q^2 \,. \tag{1.4}$$

A film of thickness H_0 is linearly unstable if $f_2(H_0) > 0$, and in the unstable flat film thickness regime, the critical wavenumber (below which films are unstable), the most unstable mode, and the maximum growth rate are given by

$$q_c = \sqrt{\frac{f_2(H_0)}{f_1(H_0)}} \,, \quad q_m = \sqrt{\frac{f_2(H_0)}{2 f_1(H_0)}} \quad \text{and} \quad \omega_m = \frac{[f_1(H_0)]^2}{4 f_1(H_0)} \,, \tag{1.5}$$

respectively.

1.3. Thin film models

We consider in this paper three different models i.e., three different sets of $f_0(h)$ and $f_1(h)$ in (1.3). The first model considered is a test case leading to a simple linear partial differential equation, i.e.,

$$f_0(h) = c_0 \quad \text{and} \quad f_1(h) = c_1 \,, \tag{1.6}$$

where the sign of c_1 is chosen so that a flat film is linearly unstable. For the remaining two models, we assume f_0 and f_1 are of the form

$$f_0(h) = \mathcal{C} h^3 \quad \text{and} \quad f_1(h) = \frac{h^3 \Pi'(h)}{\mathcal{C}} \,, \tag{1.7}$$

where $\mathcal{C}$ is the inverse Capillary number (ratio of surface tension forces to viscous force) and $\Pi(h)$ is the disjoining pressure, typically describing the strength of fluid/solid interaction. In our second model, the disjoining pressure is specified as an 'effective' disjoining pressure derived in the context of nematic liquid crystal films [28,30]. For this model, $\Pi(h) = \Pi_{\text{NLC}}(h)$, with

$$\Pi_{\text{NLC}}(h) = \mathcal{K}\left[\left(\frac{b}{h}\right)^3 - \left(\frac{b}{h}\right)^2\right] + \frac{\mathcal{N}}{2}\left[\frac{m(h)}{h}\right]^2 , \tag{1.8}$$

where

$$m(h) = g(h)\frac{h^2}{h^2+\beta^2} ; \quad g(h) = \frac{1}{2}\left[1 + \tanh\left(\frac{h-2b}{w}\right)\right] . \tag{1.9}$$

The scales are chosen based on the experiments of Herminghaus et al. and Cazabat et al. for thin films of NLC [20,31]

$$H = 100\,\text{nm} , \quad L = 10\,\mu\text{m} , \quad T = 1\,\text{s} . \tag{1.10}$$

The values of the nondimensional parameters are also based on these experiments, as discussed in some detail in our earlier work [28], and are set to

$$\mathcal{C} = 0.0857 , \quad \mathcal{K} = 36.0 , \quad \mathcal{N} = 1.67 , \quad \beta = 1 , \quad w = 0.05 , \quad b = 0.01 . \tag{1.11}$$

These parameters are used throughout the paper except if specified differently. We leave the discussion of the details of this model to § 4, where we focus on extending previous results for 2D [28] to 3D films; however, for now, we note that the term with prefactor $\mathcal{K}$ in the disjoining pressure defined in (1.8) is the power-law form of disjoining pressure consisting of Born repulsion and the van der Waals force. The power-law form of the disjoining pressure is commonly used in the literature; see e.g., the review [25] for detailed discussion. The term with a prefactor $\mathcal{N}$ in (1.8) is due to the elastic response of Nematic Liquid Crystal (NLC), further discussed in § 4.

The third model describes polymeric films [18], where $\Pi(h) = \Pi_{\text{POL}}(h)$, with

$$\Pi_{\text{POL}}(h) = -\frac{\partial \psi_{\text{POL}}}{\partial h} , \quad \text{where} \quad \psi_{\text{POL}}(h) = \frac{C}{h^8} - \frac{A_{\text{SiOx}}}{12\pi h^2} + \frac{A_{\text{SiOx}} - A_{\text{Si}}}{12\pi (h+d)^2} , \tag{1.12}$$

and the coefficients in (1.7) and (1.12) are given by

$$\mathcal{C} = 0.00581 , \quad C = 1.181 , \quad A_{\text{SiOx}} = 41.25 , \quad A_{\text{Si}} = -243.75 , \quad d = 191 , \tag{1.13}$$

with the scalings

$$H = 1\,\text{nm} , \quad L = 100\,\text{nm} , \quad T = 60\,\text{s} \tag{1.14}$$

(here the physical parameters are taken from Seemann et al. [32]). The first term in (1.12) is due to steric effects (non-bonding intermolecular interactions), and the last two terms are van der Waals forces in a thin film of a polymer deposited on a silicon substrate (Si), coated with a silicon oxide (SiOx) layer of thickness d.

2. Numerical method

The numerical approach that we employ is based on the Alternate Direction Implicit method, discussed in the context of thin film flows [15] and implemented in recent works, such as [11,30]. In the present paper we provide a review of the method, both for completeness, and for the purpose of discussing particular issues involving implementation in a GPU computing environment. More precisely, in § 2.1 we discuss the spatial discretization scheme in terms of fluxes; temporal discretization is described in § 2.2; discretization of fluxes in § 2.3; and boundary conditions are discussed in § 2.4.

2.1. Conservation law

In terms of a conservation law, the governing equation may be expressed as

$$u_t + \nabla \cdot \mathbf{F}(u) = 0 , \quad \text{where} \quad \mathbf{F}(u) = f_0(u)\nabla\nabla^2 u + f_1(u)\nabla u \tag{2.1}$$

is the flux vector, with two components, i.e., $\mathbf{F}(u) = \{F_x(u), F_y(u)\}$. To simplify the results, we generalize the flux to linear combinations of terms of the form

$$\mathbf{F}(u) = f(u)\mathbf{L}[u], \tag{2.2}$$

where $f(u)$ is some smooth function of u and $\mathbf{L}$ is some linear differential operator with two components, i.e., $\mathbf{L}[u] = \{L_x[u], L_y[u]\}$. The results to be shown can be easily extended to the flux in our original governing equation (2.1).

The governing equation for $\bar{u}$, the average value of u on some sub-domain Ω, is given by

$$\bar{u}_t = -\frac{1}{A}\oint_{\partial\Omega} \mathbf{F}(u)\cdot\mathbf{n}\, ds\,, \tag{2.3}$$

where A is the area of the sub-domain Ω, $\partial\Omega$ denotes its boundary, and $\mathbf{n}$ is the outward-pointing normal to $\partial\Omega$. To subdivide the entire domain, the solution is discretized on an equipartitioned grid; specifically, the grid points in the x and y directions are defined as

$$\begin{aligned} x_i &= X_0 + i\Delta s, \quad 0 \le i \le I \quad \text{and} \\ y_j &= Y_0 + j\Delta s, \quad 0 \le j \le J\,, \end{aligned} \tag{2.4}$$

where Δs is the grid spacing; $I+1$ and $J+1$ are the numbers of points in the x and y directions, respectively; and X_0 and Y_0 are the initial points in x and y, respectively. Furthermore, the cell average points are given by

$$\bar{u}_{(i,j)} = \frac{1}{A}\oint_{\Omega_{(i,j)}} u(x,y)\, dA, \quad \text{where} \quad \Omega_{(i,j)} = [x_i, x_{i+1}]\times[y_j, y_{j+1}] \quad \text{and} \quad A_{(i,j)} = \Delta s^2\,, \tag{2.5}$$

for $0 \le i < I$ and $0 \le j < J$; and their respective governing equations are

$$\bar{u}_{t,(i,j)} = -\frac{1}{\Delta s^2}\left[-\int_{x_i}^{x_{i+1}} F_{y,(,j)}\, dx + \int_{y_j}^{y_{j+1}} F_{x,(i+1,)}\, dy + \int_{x_i}^{x_{i+1}} F_{y,(,j+1)}\, dx - \int_{y_j}^{y_{j+1}} F_{x,(i,)}\, dy \right], \tag{2.6}$$

where

$$F_{x,(i,j)} = F_x(u(x_i, y_j))\,, \quad F_{x,(,j)} = F_x(u(x, y_j))\,, \quad F_{x,(i,)} = F_x(u(x_i, y))\,, \tag{2.7}$$

and similar notation is used for F_y and $\bar{u}_t$. To solve (2.6), two second-order accurate approximations are implemented: 1) the integrals on the right-hand side of (2.6) are evaluated using the midpoint rule, e.g.,

$$\int_{x_i}^{x_{i+1}} F_{y,(,j)}\, dx = \Delta s\left[F_{y,(i+\frac{1}{2},j)} + O(\Delta s^2)\right], \tag{2.8}$$

where

$$\begin{aligned} x_{i+\frac{1}{2}} &= X_0 + \left(\frac{1}{2}+i\right)\Delta s \quad 0 \le i < I\,, \quad \text{and} \\ y_{j+\frac{1}{2}} &= Y_0 + \left(\frac{1}{2}+j\right)\Delta s \quad 0 \le j < J\,, \end{aligned} \tag{2.9}$$

are the center points of a cell; and 2) the cell averaged value is approximated by the cell-centered value, i.e.,

$$\bar{u}_{(i,j)} = u_{(i+\frac{1}{2},j+\frac{1}{2})} + O(\Delta s^2) = u(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}}) + O(\Delta s^2)\,. \tag{2.10}$$

Substituting (2.8) and (2.10) into (2.6) yields

$$u_{t,(i+\frac{1}{2},j+\frac{1}{2})} = -\frac{1}{\Delta s}\left[F_{x,(i+1,j+\frac{1}{2})} - F_{x,(i,j+\frac{1}{2})} + F_{y,(i+\frac{1}{2},j+1)} - F_{y,(i+\frac{1}{2},j)}\right] + O(\Delta s^2)\,. \tag{2.11}$$

2.2. Temporal discretization

Let us write (2.11) in the following general form

$$\mathbf{u}_t = -\mathbf{D}\mathbf{u}\,, \tag{2.12}$$

where $\mathbf{D}$ is a nonlinear matrix differential operator, representing the fluxes on the right-hand side of (2.11); and $\mathbf{u} = \{u_p\}$ is the collection of grid point values $u_{(i,j)}$. The grid points in $\mathbf{u}$ are ordered lexicographically and may be in row-major form, $p = Ji + j$; or column-major form, $p = Ij + i$. Note that vector notation will be used to denote vectors associated with cell centered quantities.

To begin, a central difference discretization in time is applied to (2.12),

$$\mathbf{u}^{n+1} - \mathbf{u}^n = -\Delta t\mathbf{D}\mathbf{u}^{n+1/2} \tag{2.13}$$

where Δt is the time step, and n superscripts denote the value at the current time. Using the trapezoidal rule to evaluate the non-linear term at the half step, $\mathbf{D}\mathbf{u}^{n+1/2} = \left[\mathbf{D}\mathbf{u}^{n+1} + \mathbf{D}\mathbf{u}^{n}\right]/2 + O(\Delta t^2)$, (2.13) may be expressed as

$$\left(\mathbf{I} + \frac{\Delta t}{2}\mathbf{D}\right)\mathbf{u}^{n+1} = \left(\mathbf{I} - \frac{\Delta t}{2}\mathbf{D}\right)\mathbf{u}^{n}\,, \tag{2.14}$$

where $\mathbf{I}$ is the identity matrix. Equation (2.14) contains a nonlinear implicit term, i.e., nonlinear dependence on the unknown $\mathbf{u}^{n+1}$; therefore the above equation is solved numerically using a Newton iterative scheme.

2.2.1. Newton iterative scheme

To rewrite (2.14) as a root-solving problem, we define

$$\mathbf{G}(\mathbf{u}^{n+1}) = \left[\left(\mathbf{I} + \frac{\Delta t}{2}\mathbf{D}\right)\mathbf{u}^{n+1} - \left(\mathbf{I} - \frac{\Delta t}{2}\mathbf{D}\right)\mathbf{u}^{n}\right] = 0\,, \tag{2.15}$$

and find $\mathbf{u}^{n+1}$ satisfying this equation using Newton's iterative scheme. First, we compute the Jacobian of G with respect to $\mathbf{u}^{n+1}$, yielding

$$\mathbf{J}_{\mathbf{G}} = \mathbf{I} + \frac{\Delta t}{2}\mathbf{J}_{\mathbf{D}}\,, \tag{2.16}$$

where $\mathbf{J}_{\mathbf{D}}$ is the Jacobian of $\mathbf{D}\mathbf{u}^{n+1}$. The Newton iterative scheme may be expressed as

$$\begin{aligned} \mathbf{J}_{\mathbf{G},(k)}\mathbf{w} &= \left(\mathbf{I} + \frac{\Delta t}{2}\mathbf{J}_{\mathbf{D}}\right)\mathbf{w} = -\mathbf{G}(\mathbf{u}_{(k)}^{n+1}) \\ \mathbf{u}_{(k+1)}^{n+1} &= \mathbf{u}_{(k)}^{n+1} + \mathbf{w}\,, \end{aligned} \tag{2.17}$$

where $\mathbf{u}_{(k)}$ corresponds to the values at the current iterative step, $\mathbf{w}$ is an intermediate variable, and the (k) subscript in $\mathbf{J}_{\mathbf{G},(k)}$ indicates that the nonlinear $\mathbf{u}^{n+1}$ terms in $\mathbf{J}_{\mathbf{G},(k)}$ are evaluated using $\mathbf{u}_{(k)}^{n+1}$. The iterative scheme is initialized by setting $\mathbf{u}_{(0)}^{n+1} = \mathbf{u}^{n}$. Note that, assuming $\mathbf{G}$ has a solution, the Newton iterative scheme converges if the initial guess lies in some neighborhood about the solution, such that in that region: 1) $\mathbf{G}$ is continuously differentiable; 2) $\mathbf{J}_{\mathbf{G}}$ is Lipschitz continuous; and 3) $\mathbf{J}_{\mathbf{G}}^{-1}$ exists and $||\mathbf{J}_{\mathbf{G}}^{-1}||$ is bounded [33]. While demonstrating that these conditions are satisfied for our particular class of PDEs is not within the scope of this paper, it is reasonable to assume that for bounded and well-behaved $f_0(u)$ and $f_1(u)$, and sufficiently smooth u, the convergence conditions are satisfied for an appropriately chosen initial guess. To reduce computational complexity, we apply an Alternating Direction Implicit (ADI) type scheme, discussed next.

2.2.2. Alternating Direction Implicit (ADI) method

The ADI method splits the implicit differential operator (2.16) in two parts (implicit x derivatives for a fixed y and vice versa). Specifically, the right-hand side of (2.16) may be expressed as

$$\mathbf{I} + \frac{\Delta t}{2}\mathbf{J}_{\mathbf{D}} = \left(\mathbf{I} + \frac{\Delta t}{2}\mathbf{J}_{y}\right)\left(\mathbf{I} + \frac{\Delta t}{2}\mathbf{J}_{x}\right) + \frac{\Delta t}{2}\mathbf{J}_{xy} - \frac{\Delta t^2}{4}\mathbf{J}_{x}\mathbf{J}_{y}\,, \tag{2.18}$$

where $\mathbf{J}_x$ and $\mathbf{J}_y$ are the pure x and y derivative terms in $\mathbf{J}_{\mathbf{D}}$, respectively; and $\mathbf{J}_{xy}$ represents the remaining mixed derivative terms, i.e. $\mathbf{J}_{xy} = \mathbf{J}_{\mathbf{D}} - \mathbf{J}_x - \mathbf{J}_y$.

Neglecting the implicit mixed derivatives, $\mathbf{J}_{xy}$, we may substitute (2.18) into (2.17) reducing the method to first-order accuracy; however, as noted and confirmed numerically [15], implementing an appropriate iterative method, the ADI scheme will converge to second order accuracy in time. We will also show second order convergence for our implementation. The Newton iterative scheme in (2.17) becomes

$$\left(\mathbf{I} + \frac{\Delta t}{2}\mathbf{J}_{y,(k)}\right)\mathbf{w} = -\left[\left(\mathbf{I} + \frac{\Delta t}{2}\mathbf{D}\right)\mathbf{u}_{(k)}^{n+1} - \left(\mathbf{I} - \frac{\Delta t}{2}\mathbf{D}\right)\mathbf{u}^{n}\right]\,, \tag{2.19}$$

$$\left(\mathbf{I} + \frac{\Delta t}{2}\mathbf{J}_{x,(k)}\right)\mathbf{v} = \mathbf{w}\,, \tag{2.20}$$

$$\mathbf{u}_{(k+1)}^{n+1} = \mathbf{u}_{(k)}^{n+1} + \mathbf{v}\,, \tag{2.21}$$

where $\mathbf{w}$ and $\mathbf{v}$ are intermediate variables. This pseudo-Newton iterative method reduces (by an approximation) the left hand side of (2.18), from a large sparse matrix of size $IJ \times IJ$, to two block diagonal matrices (right hand side of (2.18)). In the implicit y step, using column-major ordering, we form a block diagonal matrix, $\mathbf{J}_y$, with I blocks of size $J \times J$. Each block can be inverted independently of others, thus reducing the computational complexity. Furthermore, each block is n-diagonal, where the width of the diagonal band, n, depends on the stencil used to evaluate spatial derivatives. For our implementation, $n = 5$ (penta-diagonal system). The n-diagonal matrices can be inverted with linear complexity, further

reducing computational cost, for example by using an extension of the Thomas algorithm for an n-diagonal matrix. Similarly, in the implicit x step, using row-major ordering, we form a block diagonal matrix, $\mathbf{J}_x$, with J blocks of size $I \times I$.

As noted by Witelski and Bowen [15], the ADI method with central difference in time discretization is unconditionally stable for the linear problem without diffusion, i.e., $f_0(u) = 1$ and $f_1(u) = 0$. Furthermore, it was noted that for the nonlinear problem, it is reasonable to expect convergence for a sufficiently smooth u, however, with additional restrictions on Δt.

2.2.3. Adaptive time stepping

To control the time step, we first specify the convergence conditions. Specifically, the solution at the next time step is accepted (i.e. $\mathbf{u}^{n+1} = \mathbf{u}^{n+1}_{(k+1)}$) if

$$\max_{0\le i<I, 0\le j<J} \left| \frac{v_{(i+\frac{1}{2}, j+\frac{1}{2})}}{u_{(k),(i+\frac{1}{2}, j+\frac{1}{2})}} \right| < \epsilon_{\text{Tol}} \tag{2.22}$$

where ϵ_{Tol} is the error tolerance. In addition, if the error tolerance is not satisfied in a specified maximum number of iterative steps, the scheme is assumed to have failed, triggering time step decrease.

Note 1: Our simulations show that setting the maximum number of iterative steps, K, to 10 gives the largest effective time steps, i.e., $\Delta t/K$ is maximized.

Note 2: Other restrictions besides the convergence of the iterative scheme may be placed on accepting the solution at the next iteration or next time step, e.g. sufficiently small truncation error, thus for generality, we will refer to the collection of such conditions as the *solution criteria*.

If the solution criteria are not met, then the time-step is reduced and the pseudo-Newton iterative scheme is performed with the new reduced time step. Furthermore, if the new reduced time step is below some minimum, the numerical scheme halts and ends in a failed state. This failed state is usually a result of either 1) the time step, Δt, being too large for the chosen spatial step size, Δx; or 2) slow convergence of the pseudo-Newton iterative scheme. Validation results (to be discussed in § 3.1) support this conjecture, since in that section we remove adaptive time stepping and find that obtaining convergence for a range of Δx values requires selecting a small enough Δt and increasing the maximum number of iterative steps to a sufficiently large value. In the case of a failed state, several modifications to simulation parameters or the solution criteria may be made. Naively, the minimum time step size could be reduced; however, too small of a time step may lead to excessively long run times. Alternatively, the spatial step size can be increased; however, this may not be an option due to numerical accuracy considerations, e.g., in the case that very thin films need to be resolved. Lastly, the maximum number of Newton iterations and/or the convergence error tolerance on the Newton iterative method, ϵ_{Tol}, may be increased. To improve the efficiency of the numerical scheme, the time step is increased if a minimum number of successful time steps have been consecutively computed. To remove possible numerical errors, the time step is bounded from above. Fig. 1 shows a flow chart of the adaptive time stepping procedure, coupled with the pseudo-Newton iterative method.

2.3. Flux discretization

To summarize the results so far, a second-order accurate scheme has been developed in terms of a non-linear spatial differential operator $\mathbf{D}$, representing the fluxes across cell boundaries. In this section, to complete the scheme, we specify the form of $\mathbf{D}$. Recalling (2.11), we may split $\mathbf{D}$ into two parts: the fluxes across the x cell boundaries $\mathbf{D}_x$, and the fluxes across the y cell boundaries $\mathbf{D}_y$. The values of the components of $\mathbf{D}$ are specified in terms of fluxes, i.e.,

$$\mathbf{D}_s = \frac{\mathbf{F}_{s,+} - \mathbf{F}_{s,-}}{\Delta s}\,, \tag{2.23}$$

where $\mathbf{D}_s$ is either $\mathbf{D}_x$ or $\mathbf{D}_y$, and $\mathbf{F}_{s,+}$ and $\mathbf{F}_{s,-}$ are non-linear matrix differential operators for the fluxes at the two boundaries. Similarly to § 2.1, the fluxes are generalized to a linear combination of functions of the form

$$\mathbf{F} = f(u)\mathbf{L}\,, \tag{2.24}$$

where $f(u)$ is a function of u, and $\mathbf{L}$ is a linear differential operator. We may express $\mathbf{D}$ as

$$\mathbf{D} = \frac{\mathbf{F}_{x,+} - \mathbf{F}_{x,-} + \mathbf{F}_{y,+} - \mathbf{F}_{y,-}}{\Delta s}\,. \tag{2.25}$$

To derive an expression for $\mathbf{J}_x$ and $\mathbf{J}_y$ in (2.19) and (2.20), we recall that mixed derivative terms have been ignored, and therefore,

$$\mathbf{J}_x = \frac{\partial}{\partial \mathbf{u}}\left[\frac{\mathbf{F}_{x,+} - \mathbf{F}_{x,-}}{\Delta s}\right] \quad \text{and} \quad \mathbf{J}_y = \frac{\partial}{\partial \mathbf{u}}\left[\frac{\mathbf{F}_{y,+} - \mathbf{F}_{y,-}}{\Delta s}\right], \tag{2.26}$$

where $\partial/\partial\mathbf{u}$ denotes the Jacobian, and the non-linear matrix differential operator, $\mathbf{F}$, only contains derivatives with respect to its subscript.

To complete the derivation, expressions for $\mathbf{F}_{x,+}$, $\mathbf{F}_{x,-}$, $\mathbf{F}_{y,+}$, and $\mathbf{F}_{y,-}$ are required. For brevity, we specify these expressions by the following descriptions:

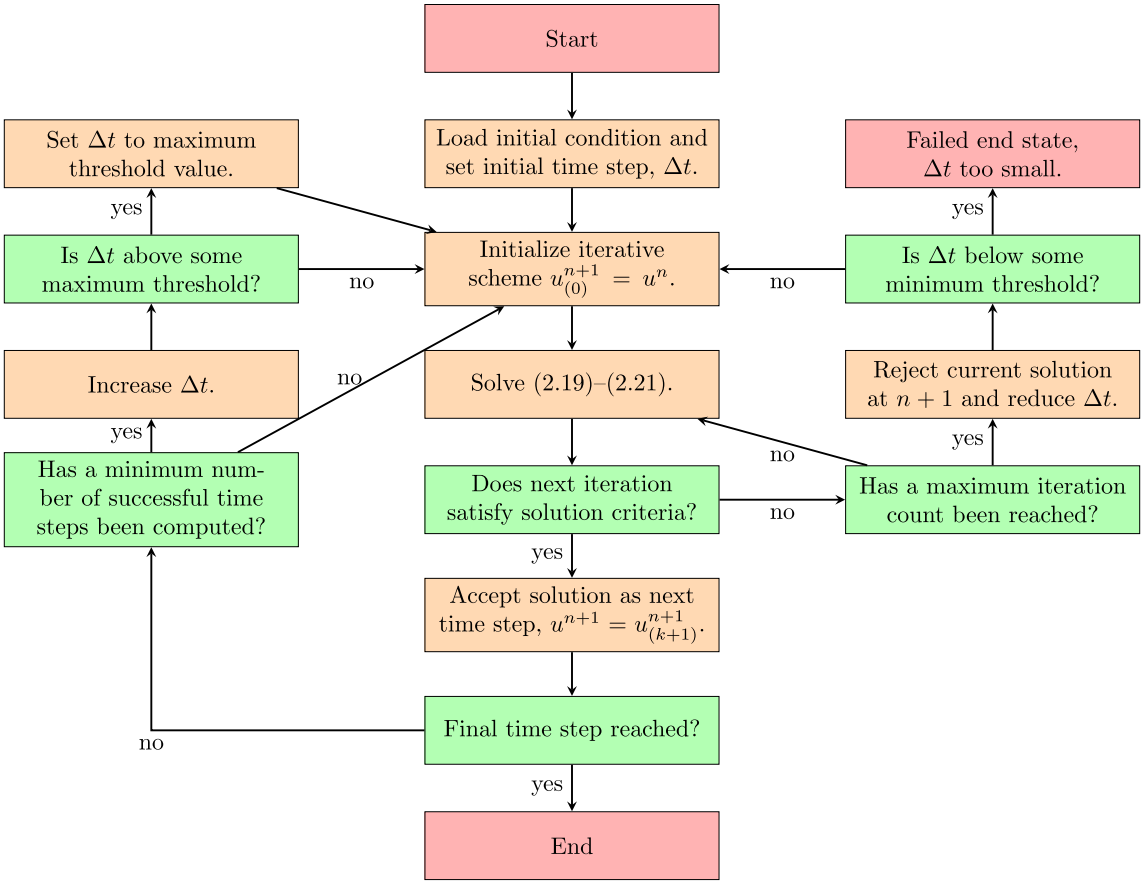


Fig. 1. Flow chart of the adaptive time stepping with a nested pseudo-Newton iterative scheme. Note that red rectangles correspond to the start or end of the flow chart, orange boxes designate processes, and green boxes denote decisions.

1. at the cell-centered point (i, j),
   - $\mathbf{F}_{x,+}$ is evaluated at $(x_{i+1}, y_{j+\frac{1}{2}})$,
   - $\mathbf{F}_{x,-}$ is evaluated at $(x_i, y_{j+\frac{1}{2}})$,
   - $\mathbf{F}_{y,+}$ is evaluated at $(x_{i+\frac{1}{2}}, y_{j+1})$, and
   - $\mathbf{F}_{y,-}$ is evaluated at $(x_{i+\frac{1}{2}}, y_j)$;
2. the linear matrix differential operator $\mathbf{L}$ is computed using cell-centered values;
3. $f(u)$ at a cell boundary is evaluated using interpolation, e.g.,
$$f(u_{(i+\frac{1}{2},j)}) = \frac{f(u_{(i+\frac{1}{2},j+\frac{1}{2})}) + f(u_{(i+\frac{1}{2},j-\frac{1}{2})})}{2} + O(\Delta s^2)\,; \tag{2.27}$$
4. the total combined stencil of all terms is shown in Fig. 2.

2.4. Boundary conditions

To complete the description of the numerical methods, we now discuss the boundary conditions. Typically, boundary conditions of the form

$$\begin{aligned} &u_x(x_0, y) = u_{xxx}(x_0, y) = 0\,, \qquad u_x(x_I, y) = u_{xxx}(x_I, y) = 0\,, \\ &u_y(x, y_0) = u_{yyy}(x, y_0) = 0\,, \qquad u_y(x, y_J) = u_{yyy}(x, y_J) = 0\,, \end{aligned} \tag{2.28}$$

are used, and may be interpreted as symmetry conditions. Furthermore, these boundary conditions correspond to zero flux at the boundary (recall (2.1)).

To implement the boundary conditions, the stencil shown in Fig. 2 includes ghost points at the boundaries. While ghost points may be used for explicit terms ($u^{n+1}_{(k)}$ and u^n), for the implicit term, $u^{n+1}_{(k+1)}$, the finite difference stencil must be modified. However, since implicit terms are derivatives of a single variable, the stencil modification is relatively simple.

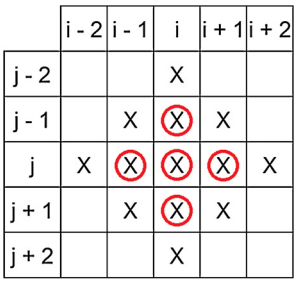


Fig. 2. X and ◯ denote the solution cell-center values $u_{(i,j)}$ and the function cell-center values, $f_{i,j}$, respectively. These values are required to numerically evaluate the flux differences (2.23) at the cell-center (i, j).

Using the boundary conditions to define the ghost points, the left-hand sides of the numerical scheme (2.19) and (2.20) are computable everywhere except at the corners of the domain, e.g., at the cell-center point $i = j = 0$. There are two choices to compute the ghost points in the corner $(x_i, y_j$ for $i, j = -2, -1)$: First, the y boundary conditions are applied, computing the solution at the ghost points (x_i, y_j) for $i = 0, 1$ and $j = -2, -1$, then the x boundary conditions are applied at the new ghost points, computing the solution at the corner ghost points. Alternatively, the order of operations may be reversed. It is relatively easy to show that both procedures lead to identical results.

3. Numerical performance

We now switch focus to the numerical performance and accuracy of our implementation. The details related to implementation in the GPU computing environment are discussed in Appendix A. Here, we discuss first convergence, followed by confirmation of conservative properties, and the comparison with LSA. Then, we discuss the performance by comparing the GPU and serial CPU codes.

For the purpose of the numerical tests discussed in this section, we choose an initial condition of the form

$$u(x, y, t = 0) = H_0 \left(1 + \left[\epsilon_x \cos\left(\frac{\pi x}{\lambda}\right) + \epsilon_y \cos\left(\frac{\pi y}{\lambda}\right)\right]\right) \quad (x, y) \in [0, P\lambda] \times [0, P\lambda], \quad \lambda = 2\pi/q\,, \tag{3.1}$$

where $\epsilon_x = \epsilon_y = 0.1$ and q is the wave number. The initial film thickness, H_0, is fixed at 1 for the linear model (for simplicity), 0.5 for the NLC model, and 3.9 for the polymer model. For the last two models, the values are motivated by the experiments [18,34].

3.1. Validation

To test the convergence properties, in § 3.1.1 and § 3.1.2, the adaptive time stepping is removed and we set the maximum number of Newton iterative steps to 5000 (allowing for a more robust selection of spatial step size). In addition, we choose $q = q_m$ and $P = 6$ in (3.1) where q_m is given by (1.5). The spatial step size, $\Delta s = P\lambda_m/I$, where $\lambda_m = 2\pi/q_m$ is the most unstable wavelength, I is the number of grid points, and simulations are carried out on a square computational domain, $I = J$; therefore, choosing I (discussed in the next sections) determines Δs. Note that the choice $P = 6$ is made so as to maximize the range of grid sizes that fit into the GPU's memory and maintain a sufficiently large spatial step size to maximize the (common) stable time step. In § 3.1.3, simulation results are compared with the predictions of LSA, where adaptive time stepping is utilized, the maximum number of Newton iterative steps is reverted back to 10, we set $P = 1$, and we vary q according to the dispersion curve from LSA.

3.1.1. Convergence

To study the convergence properties, we first define I_0, the initial spatial grid size, i.e., $I = J = I_0$, and Δt_0, the initial time step size. In addition, since obtaining an analytical solution for the non-linear models is not possible, we compute the L_2 norm of the error, relative to the numerical solution obtained for the most refined partition. Specifically, we compute

$$\|u_{(r)}\|_2 = \sqrt{\sum_{i=0}^{I_0-1}\sum_{j=0}^{I_0-1}\left(\frac{u_{(i,j),(r)} - u_{(i,j),(R)}}{u_{(i,j),(R)}}\right)^2}\,, \tag{3.2}$$

where we use subscript notation (r), $0 \le r \le R$, to denote the refinement level; and R to specify the maximum level of refinement.

For the purpose of checking temporal convergence, we fix $I_0 = 256$ for all simulations in this section, and set $\Delta t_0 = 10^{-3}\omega_m^{-1}$, i.e., we scale the time step with the growth rate of the most unstable mode. The initial simulation is carried out for one time step, Δt_0, and for each subsequent simulation, the time step is halved, i.e., $\Delta t_i = 2^{-i}\Delta t_0$, where i is the

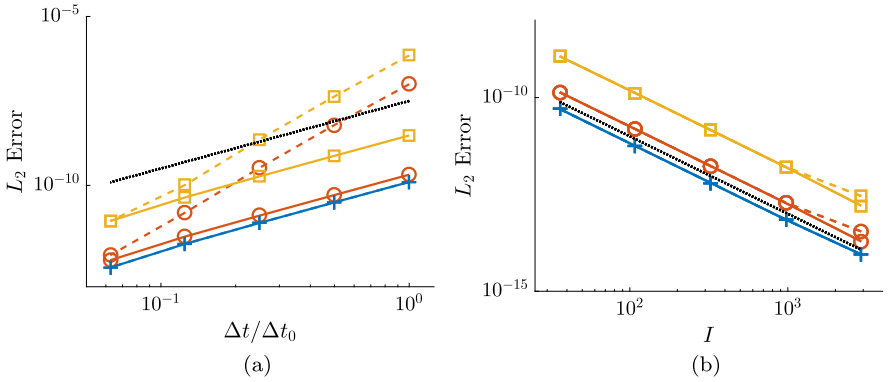


Fig. 3. L_2 norm of the error as a function of (a) time step size and (b) number of grid points in the x direction for: the linear model (blue curves with '+' symbols), the NLC model (red curves with '∘' symbols), and the polymer model (yellow curves with '□' symbols). Solid curves denote simulations where Newton iterative convergence error tolerance is close to machine precision, $\epsilon_{\text{Tol}} = 10^{-14}$; and dashed curves denote simulations with error tolerances set to (a) $\epsilon_{\text{Tol}} = \max(\Delta s^2, 10^{-14})$ and (b) $\epsilon_{\text{Tol}} = \max(\Delta t^2, 10^{-14})$. The dotted black line denotes second order convergence. Note that in both figures for the linear model the two curves (solid and dashed) lie on top of each other, as expected.

refinement level. Note that at each refinement level the total number of time steps doubles. Fig. 3(a) shows the L_2 norm of the error for all three considered problems for two implementations: i) the Newton iterative convergence error tolerance, ϵ_{Tol} in (2.22), is set close to machine precision (solid curves), $\epsilon_{\text{Tol}} = 10^{-14}$; and ii) the error tolerance is set to scale with the temporal error (dashed curves), $\epsilon_{\text{Tol}} = \max(\Delta t^2, 10^{-14})$. Note that we limit $\epsilon_{\text{Tol}} \geq 10^{-14}$ since if the relative error in the Newton iterative scheme is smaller than machine error, the iterative method will always fail. We observe that in the first case, second-order temporal accuracy is achieved (compare to the dotted black line). Note that for the linear model, the two curves lie on top of each other, as expected. Furthermore, the results show that it is sufficient to set the Newton error tolerance to be proportional to the temporal accuracy to maintain second-order accuracy, and even higher order convergence is achieved for the nonlinear models, although the error is larger.

To confirm the spatial convergence, we fix $\Delta t_0 = 10^{-8}\omega_m^{-1}$ for all simulations and set the coarsest grid size $I_0 = 36$. Recalling § 2.1, the numerical solution is evaluated at the cell-centers of the spatial grid; therefore, if the spatial step size is halved, the cell-centered points on the refined grid are not contained in the coarse grid. To avoid errors associated with interpolating the numerical solutions on different grid sizes to a common grid size, we instead triple the grid size (e.g., $I_r = 3^r I_0$, where r is the level of refinement) so that cell-centered points on the coarser grid are contained in the refined grid. To compute the L_2 norm defined in (3.2), we map the indices from refined grids to the coarsest grid as follows,

$$\tilde{u}_{(i,j),(r)} = u_{(d(i),d(j)),(r)}\,, \quad d(i) = \frac{3^r - 1}{2} + 3^r i, \tag{3.3}$$

where tilde notation denotes the numerical solution on the (common) coarsest grid. Fig. 3(b) confirms second-order spatial accuracy. Furthermore, we observe that there is little difference between setting the Newton error tolerance to $\epsilon_{\text{Tol}} = \max(\Delta s^2, 10^{-14})$ or close to machine precision, $\epsilon_{\text{Tol}} = 10^{-14}$.

To summarize, the results show that the implemented method is second-order accurate in time and space, and furthermore, that it is sufficient to set the error tolerance to scale with the order of the spatial and temporal accuracy. Such implementation significantly reduces computational time.

3.1.2. Conservation of mass

To verify conservation of mass, we carry out the simulations until the final time, $1000\Delta t_0$, and collect data in intervals of $100\Delta t_0$. Furthermore, motivated by the results of § 3.1.1, we fix $\epsilon_{\text{Tol}} = \max(\Delta s^2, 10^{-14})$ for spatial convergence simulations, and $\epsilon_{\text{Tol}} = \max(\Delta t^2, 10^{-14})$ for temporal convergence simulations. To compute the average mass of the solution, recall that the cell average is given by the cell-centered value (with second order accuracy), therefore, the average mass on the entire domain is given by

$$\bar{U} = \frac{1}{IJ}\sum_{i=0}^{I-1}\sum_{j=0}^{J-1} u_{(i+\frac{1}{2},j+\frac{1}{2})}\,. \tag{3.4}$$

Fig. 4 shows the change (relative to the initial condition) in total mass for the linear model (left column), NLC model (central column) and polymer model (right column), for fixed time step and varied spatial step (top row), and fixed spatial step and varied time step (bottom row). The results obtained when the spatial step size is varied (top row) show that there is no discernible trend in the mass error (numerical noise), with the exception of the smallest step size for the NLC model and the polymer model, indicated by the arrows in panels (b) and (c), respectively. It is interesting to note that the conservation

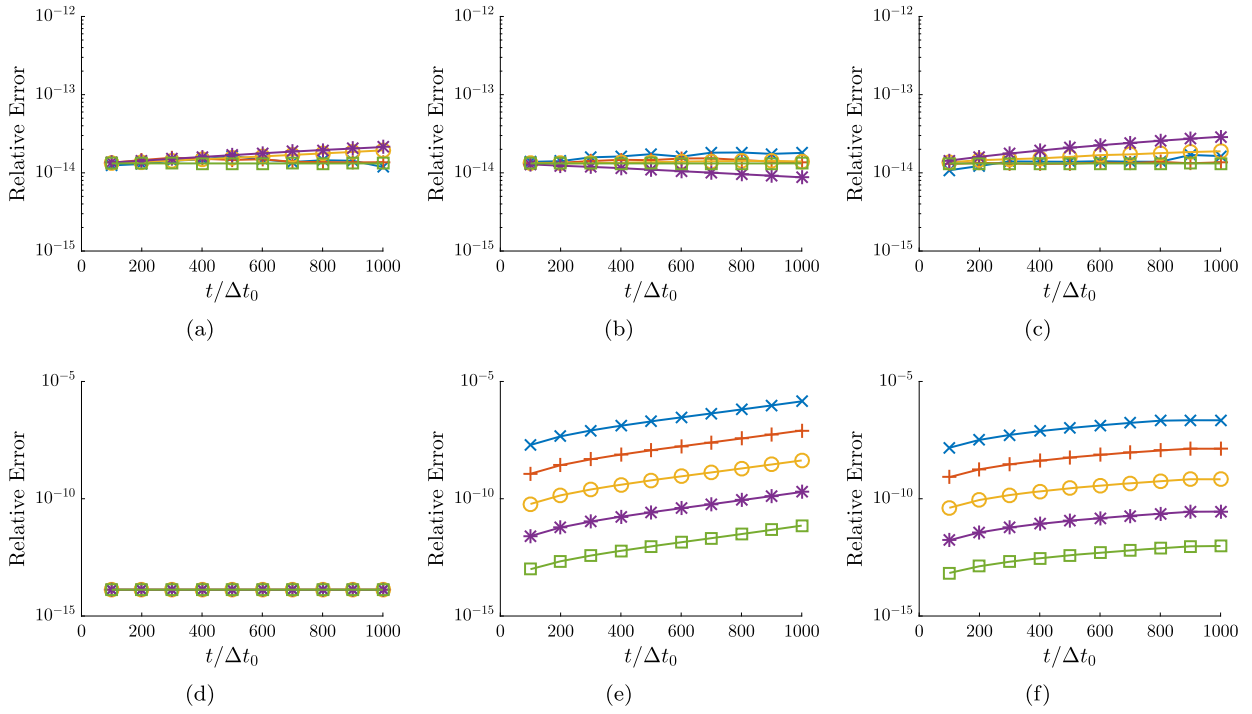


Fig. 4. The relative mass error as a function of time for the linear model (left column), the NLC model (central column), and the polymer model (right column) for decreasing spatial step size (top row) and decreasing time step (bottom row). Spatial step sizes and time step sizes are the same as Fig. 3 and '×', '+', '∘', '∗', and '□' symbols denote progressively decreasing spatial and temporal step size. Note that for the top row of figures $\Delta t = \Delta t_0 = 10^{-8}\omega_m^{-1}$ and $I = I_0 3^i$, where $I_0 = 36$ and $i = 0, 1, 2, 3, 4$; and for the bottom row of figures $I = 1024$ and $\Delta t = 2^{-i}\Delta t_0$ where $\Delta t_0 = 10^{-3}\omega_m^{-1}$ and $i = 0, 1, 2, 3, 4$ is the refinement depth (decreasing step size).

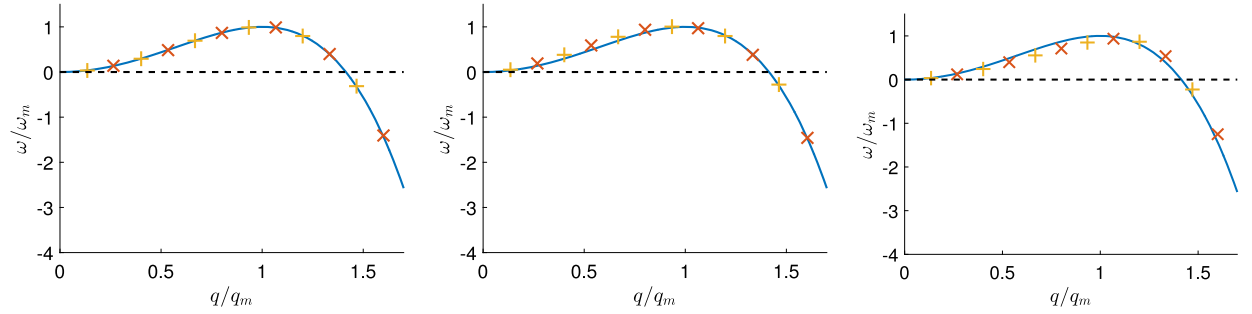


Fig. 5. Comparison between dispersion relations (blue curves) and growth rates extracted from numerical simulations (symbols) for (a) the linear model, (b) the NLC model, and (c) the polymer model. '×' symbols (red) denote initial condition perturbed in the x direction and '+' symbols (gold) denote a perturbation in the y direction.

law formulation of the governing equation in terms of fluxes (recall equation (2.11) in § 2.1) indicates that conservation of mass should be second order accurate in space. However, while the approximations of the fluxes are second-order accurate, cancellation of these errors across a cell boundary leads to higher order accuracy.

When time step is varied, with the spatial step fixed, the bottom row of Fig. 4 shows that conservation of mass is at least second order accurate. For the linear model, panel (d), the error is close to machine precision, so no trend is observed.

3.1.3. Comparison to LSA

Here, we validate our model by showing agreement with LSA. This is done by comparing the growth rates extracted from simulations to the dispersion relation (1.4). For each model, simulations are carried out for several q values for perturbations in either the x or y directions. In addition, the linear domain is fixed to λ_m, i.e. $P = 1$ in (3.1) with 32 points ($I = J = 32$). The final solution time is fixed at $T = \log(1.3)|\omega(q)^{-1}|$, where $\omega(q)$ is defined in (1.4). Fig. 5 confirms that the numerical results agree with the LSA predictions, validating the code.

Table 1
Comparison between wall times for simulations performed on the CPU and GPU. Simulations were carried out with initial condition specified in (3.1) with $P = I/64$ where I is the linear domain size and to a final time, $T = 20 \approx \omega_m^{-1}$, where $\omega_m = 0.05169$ for $H_0 = 0.5$.

Domain size	CPU wall time (s)	GPU wall time (s)
64×64	1.83	1.26
128×128	4.99	1.37
256×256	21.4	2.20
512×512	97.3	3.96
1024×1024	395	8.70
2048×2048	1669	22.4
4096×4069	11756	76

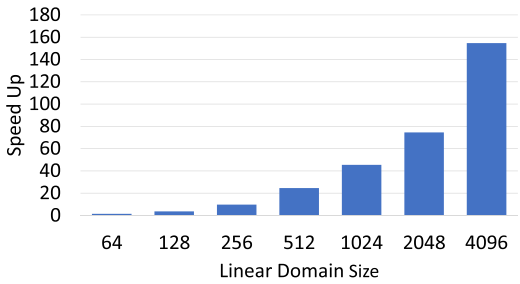


Fig. 6. Plot of relative speedup for simulations described in Table 1.

3.2. Performance comparison

Having validated the accuracy, convergence, and mass conservation of our numerical method, we here discuss the computational performance of our GPU implementation (details are given in Appendix A) in comparison to our serial CPU code. The comparison is not straightforward since some parts of the existing serial CPU code are implemented differently from the GPU implementation. To provide a complete picture, we provide two measures: (i) comparison of the wall times required for the simulations to reach a certain specified time; and (ii) comparison of the computing time used for solving the linear algebra problem (penta-diagonal system of equations in (A.1) and (A.2)). The penta-diagonal solver was chosen for comparison since solving it uses a significant part of the computing resources, and furthermore it is coded similarly in both GPU and CPU implementations. The CPU computations were performed on an Intel© Core™i7-6700 K and GPU computations were performed on a Nvidia Geforce© GTX Titan X Maxwell.

To begin, we compare simulation wall times for the GPU code and the in-house serial CPU code. For brevity, the performance results are presented only for the NLC model. Simulations were carried out with the initial condition specified in (3.1) with $P = I/64$ where I is the linear domain size and a final time of $T = 20 \approx \omega_m^{-1}$, where $\omega_m = 0.05169$ for $H_0 = 0.5$. The results are shown in Table 1 and the relative speedup as a function of the linear domain size is plotted in Fig. 6. It may be seen that for small domains, e.g., 64×64, there is only a minor difference in computing times between GPU and CPU simulations; however, for the largest domain sizes, the GPU code can perform up to 155 times faster than the CPU code.

Regarding penta-diagonal, solvers, the serial CPU implementation is based on the approach found in Numerical Recipes in FORTRAN [35]. Fig. 7 and Table 2 show that for the smallest domain size, the GPU and CPU codes are again of comparable performance; however, for larger domains, the speedup of the GPU over the CPU code reaches factor of 40. The saturation of the speedup occurs when the total number of independent linear systems is greater than the total core count (3072 for the GPU used in the reported computations).

To summarize, we have shown that on large enough domain sizes, our GPU implementation is considerably faster than the CPU one. The linear algebra problem (inverting penta-diagonal solver), which is a major component of the numerical scheme, shows speedup of 40 times and is limited by the number of GPU cores. The overall performance of our GPU implementation shows even larger speedup, however, it should be pointed out that there are differences in coding algorithms between the GPU and CPU implementations.

4. Nematic liquid crystals

In this section, we focus on the thin film model derived for nematic liquid crystals, i.e., the governing equation given by (1.1) and (1.8). The details of the model itself are discussed in detail in [28], where a weak anchoring model was presented. In [28] it was shown that the nematic contribution to the thin film model results in an effective disjoining pressure, given

Table 2
Table of wall times the original in-house CPU code and GPU code for various domain sizes.

Domain size	CPU wall time (s)	GPU wall time (s)
64×64	0.00135	0.000235
128×128	0.00580	0.000400
256×256	0.0256	0.00105
512×512	0.122	0.00228
1024×1024	0.465	0.00611
2048×2048	2.25	0.0182
4096×4069	12.5	0.0643
8192×8192	52.8	0.242

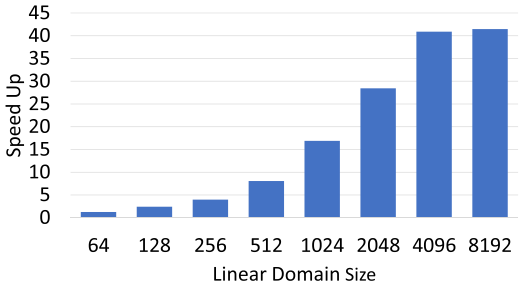


Fig. 7. Plot of relative speedup to solve penta-diagonal system in the GPU code over the original in-house CPU code for various linear domain sizes.

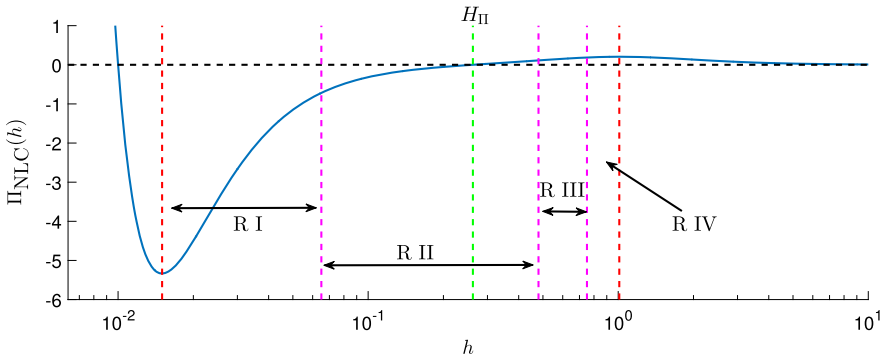


Fig. 8. The effective disjoining pressure (solid blue curve) for a NLC, (1.8), as a function of the film thickness. The region between the outer dashed red vertical lines denotes linearly unstable film thicknesses, and the central dashed green line denotes the zero of the disjoining pressure, H_Π (within the linearly unstable regime). Stability regimes R I, R II, R III, and R IV, determined numerically for 2D films in [28], are discussed in § 4.3.2.

by (1.8), and plotted in Fig. 8. Similar functional forms of the disjoining pressure, characterized by a local maximum for nonzero film thicknesses, are found in other contexts, such as polymer films [18], or even simpler setups, such as water on quartz [36].

In the present paper, we use large-scale simulations to discuss some of the main features of the results from [28] in the three dimensional context. These include formation of satellite (secondary) drops, nucleation versus spinodal type instability, and metastability of films. The fact that we are able to simulate accurately large domains of linear dimensions measured in tens of wavelengths of maximum growth, allows us to obtain results that are only very weakly influenced by the presence of the domain boundaries. This section is organized as follows: in § 4.1 we present the model in terms of the gradient dynamics formulation, § 4.2 gives a model outline, and the main part, § 4.3 presents the computational results.

4.1. Gradient dynamics formulation

It is convenient to express the governing equation (1.1) in terms of the gradient dynamics formulation; in particular, it is advantageous to express analytical results in terms of a disjoining pressure, $\hat{\Pi}(\hat{h})$. The benefit of this formulation is that previous analytical results for 2D films [28] show that the form of the disjoining pressure primarily determines the transition between different stability types of a thin film as a function of its thickness.

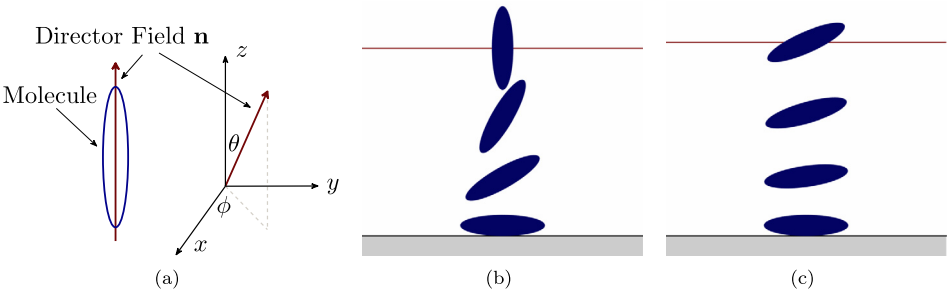


Fig. 9. (a) Schematic of director field relative to liquid crystal molecule. (b) Strong homeotropic free surface anchoring model example. (c) Weak homeotropic free surface anchoring model.

To motivate this assertion, we perform LSA on the gradient dynamics formulation of the governing equation, which is given by

$$\mu\hat{h}_t + \hat{\nabla}\cdot\left[\hat{Q}(\hat{h})\nabla\frac{\delta\hat{F}}{\delta\hat{h}}\right] = 0\,, \quad \text{where} \quad \hat{F}(\hat{h}) = \gamma\left[1+\frac{\hat{\nabla}\hat{h}\cdot\hat{\nabla}\hat{h}}{2}\right] + \hat{\psi}(\hat{h})\,, \quad \hat{\Pi}(\hat{h}) = -\frac{\partial\hat{\psi}}{\partial\hat{h}}\,, \tag{4.1}$$

$\hat{Q}(\hat{h})$ is the mobility function, $\hat{F}(\hat{h})$ is total interfacial energy (Gibbs energy), γ is the surface tension, and $\hat{\Pi}(\hat{h})$ is disjoining pressure. Relating (4.1) to the nondimensional governing equation (1.3),

$$\tilde{f}_0(h) = \tilde{Q}(h)\,, \quad \hat{F}_0 = \gamma\hat{M}\,, \quad \tilde{f}_1(h) = \tilde{Q}(h)\tilde{\Pi}'(h) \quad \text{and} \quad \hat{F}_1 = \frac{\hat{M}\hat{\Psi}}{H^2}\,, \tag{4.2}$$

where, similarly to before, $\hat{M}$ and $\hat{\Psi}$ are dimensional constant prefactors derived from expressing $\hat{Q}(\hat{h})$ and $\hat{\Psi}(\hat{h})$ as dimensionless functions of the dimensionless variable h, e.g., $\hat{Q}(\hat{h}) = \hat{M}\tilde{Q}(h)$. The nondimensional linear stability analysis results (1.5) simplify to

$$q_m = \sqrt{\frac{\Pi'(H_0)}{2\Gamma}} \quad \text{and} \quad \omega_m = \frac{Q(H_0)[\Pi'(H_0)]^2}{4\Gamma}\,, \quad \text{where} \quad \Gamma = \frac{\hat{\Psi}}{\gamma\delta^2} \tag{4.3}$$

is a dimensionless constant and may be interpreted as the ratio of disjoining pressure to surface tension forces. We note that the disjoining pressure determines the transition between linear stability ($\Pi'(H_0) < 0$) and instability ($\Pi'(H_0) > 0$). The instability wavelength is independent of the mobility function, which only affects the growth rate of instabilities.

4.2. Model description

To motivate the form of the disjoining pressure shown in Fig. 8 and defined by (1.8), here we provide a brief outline of the main features of NLC, and we refer the reader to our previous work [28] for a more detailed derivation. Typically, NLC molecules are rod-like structures with an (electrical) dipole moment, which gives rise to a state of matter intermediate between a crystal and a liquid. Specifically, similarly to a Newtonian fluid, there is no positional ordering to the NLC molecules; however, similarly to a crystal, molecules have short-range orientational order due to interactions between dipoles (elastic response). To model NLC, in addition to modeling the velocity field of molecules (as with most fluids), short-range ordering is often modeled with a director field. The director field is a unit vector, aligned with the long axis of the liquid crystal, see Fig. 9(a), and it is often convenient to consider the polar angle, θ, and azimuthal angle, ϕ, of the director field orientation as functions of Cartesian space variables (x, y, z).

The evolution of NLC may be modeled using the Leslie–Ericksen equations, an extension of the Navier–Stokes equations, with an additional equation modeling conservation of energy. Under the long wave approximation, and assuming that the time scale of fluid flow is much faster than that of the elastic reorientation, the conservation of energy equation decouples from the remaining equations and only depends on the director field. Using energy minimization [28,30,37], the polar and azimuthal angles of the director field are determined to be of the form

$$\theta(x, y, z) = a(x, y)z + b(x, y) \quad \text{and} \quad \phi(x, y) = c(x, y)\,, \tag{4.4}$$

where $a(x, y)$, $b(x, y)$, and $c(x, y)$ are constant with respect to z and are chosen to satisfy appropriate so-called anchoring boundary conditions at both the substrate and the free surface, thus the director field is a function of the instantaneous fluid height.

At the interface between the NLC film and another material, liquid crystal molecules satisfy certain anchoring conditions. Typically, at the free surface (air/NLC interface), molecules are normal to the free surface (a condition known as homeotropic anchoring; within the long-wave approximation, $\theta(x, y, z = h) = \pi/2$), while at the NLC/substrate interface, planar anchoring

is appropriate, so $\theta(x, y, z = 0) = 0$, see Fig. 9(b). However, for very thin films, or close to a contact line, this configuration induces a large energy penalty in the bulk due to rapid spatial variations in the director field; therefore, we implement a novel weak free surface anchoring model. In practice, the substrate anchoring is stronger than the free surface anchoring, thus we relax the free surface anchoring to that of the substrate for very thin films (weak anchoring model) to alleviate the large energy penalty in the bulk (compare Figs. 9(b) and 9(c)). Specifically,

$$\theta(x, y, z) = \frac{\pi}{2}\left(1 - \frac{m(h)}{h}z\right) \tag{4.5}$$

where $m(h) \in [0, 1]$ and is defined in (1.9). The azimuthal anchoring ϕ in (4.4) is independent of z; therefore, assuming the substrate anchoring dominates, the azimuthal anchoring is determined by the substrate. The governing equation (1.1) is a simplification of the long wave approximation of the Leslie–Ericksen equations that ignores substrate anchoring. In the full (long wave) model, the mobility function may be expressed as

$$Q(h) = \left[\lambda \mathbf{I} + \nu \begin{pmatrix} \cos 2\phi & \sin 2\phi \\ \sin 2\phi & -\cos 2\phi \end{pmatrix}\right] h^3 , \tag{4.6}$$

where $\mathbf{I}$ is the identity matrix, and λ and ν are anisotropic viscosities. To simplify the model, we fix $\lambda = 1$ and $\nu = 0$, which removes the dependence on ϕ. Note that by definition, $\lambda > \nu$, therefore the mobility function is positive definite and does not change stability properties. We leave the investigation of the effects of substrate anchoring for future work.

4.3. Simulations

For the remainder of this section, we focus on considering some aspects of the previous analysis carried out for 2D films [28] in the 3D geometry. We note that there has been a significant amount of work discussing various aspects of instability development, see, e.g. [8,12,38–41] and the relevant part of the review [25]. The simulation results that we present here are distinguished by significantly larger computational domains that allow for more elaborate analysis of the results. Direct comparison with some of the available experimental results is presented in § 5.

This section is divided into three parts; in the first two parts we focus on the linearly unstable regime, investigating coarsening and satellite drop formation in § 4.3.1; and the nucleation dominated regime in § 4.3.2. In § 4.3.3, the metastable regime is examined. Animations of the simulation results are available, and are provided as Supplementary Material, see Appendix C. Before presenting these results, we discuss the domain size, the spatial step size of simulations, and also the tools used to quantify simulation results: (i) the Fourier transform of the film profile, and (ii) the Betti numbers, topological measures of the film profile.

In § 3.1 the spatial step size was determined by I, λ_m, and P, in this section, we instead fix Δs, λ_m, and P and then set the grid size $I = P\lambda_m/\Delta s$, with I ranging between 1000 and 4000, giving the total number of grid points of the order of 10^6–10^7. For all simulations, we fix the spatial step size to $\Delta s = 0.05$ except for the film thicknesses $H_0 = 0.05,\ 0.01$, where $\Delta s = 0.01,\ 0.02$, respectively. The spatial step size was changed for these film thicknesses so as to maintain a sufficient number of points per period of the most unstable wavelength, λ_m. Note that for thicker films, while a larger Δs may be chosen while maintaining a sufficient number of points per period (say 50), to adequately resolve the contact line, Δs is chosen to be close to the precursor thickness, $b = 0.01$. In addition, simulations in the linearly unstable regime are scaled in two ways: (i) the linear domain size is set to $P\lambda_m$ where P is an integer, i.e., domains are scaled by P periods of the most unstable wavelength from LSA; and (ii) when comparing results for different film thicknesses as a function of time, it is convenient to define a new timescale, $\tau = t\omega_m$, i.e., we scale time by the most unstable growth rate obtained using LSA. We fix $P = 40$ for all (linearly unstable) thicknesses.

One aspect of the analysis of the results focuses on the extraction of the dominant length scales. This goal is achieved by using a 2D Fourier transform. Assuming that the instability pattern is isotropic, the magnitude of the 2D Fourier transform may be mapped to a one-dimensional (radial) function of the magnitude of the wave vector $q_r = \sqrt{q_x^2 + q_y^2}$, where q_x and q_y are the x and y components of the wave vector. In addition, several smoothing techniques are required to reliably extract local maxima. The details of the procedure are not trivial if one wishes to extract accurate results, and are described in Appendix B.

Next, we discuss the use of Betti numbers to further quantify the simulation results. The Betti numbers, b_n, are n topological invariants that characterize the connectivity of n-dimensional objects. For our purposes, $n = 2$ since the film thickness is a function of two spatial variables. To illustrate the meaning of b_0 and b_1, we use landscape analogy: consider a landscape sliced by a horizontal plane at the height H^*. Then, b_0 counts the number of mountain peaks (components) above H^*, and b_1 counts the number of valleys (loops) surrounding the peaks. In any given landscape, the values of b_0 and b_1 change with the threshold level H^*. In the present context, we will think of the film thickness, h, as the variable describing the landscape; by computing Betti numbers for all relevant thresholds (ranging from precursor film thickness to the values larger than the initial film thickness), we will be in the position to describe in precise terms the topology of the evolving films, and to analyze the appearance of topological objects across scales. To minimize the influence of boundaries, it is important to carry out simulations in large domains such that boundary-related effects are minor; the computational

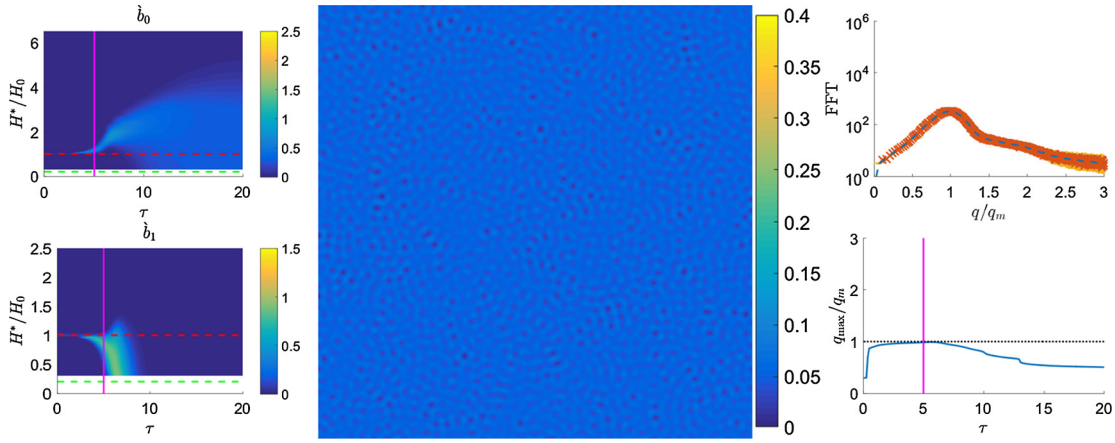


Fig. 10. Evolution of film of thickness $H_0 = 0.05$ perturbed by global perturbations as described in the text. The central part shows a contour plot of the free surface height at $\tau = 5$. The domain size is $40\lambda_m \times 40\lambda_m$. The right-hand side includes the corresponding radial Fourier transform of the film height (top), and local maxima of radial Fourier transform, $q_{\max}$ (bottom) as a function of τ. The left-hand side shows the contour plots of the average Betti numbers $\grave{b}_0$ (top) and $\grave{b}_1$ (bottom) as functions of τ and scaled threshold value H^*/H_0. The vertical magenta line corresponds to the solution time shown in the central panel. The grid size is 1637×1637. For animations, see Appendix C, movie1.

domains that we choose are sufficiently large for the purpose. The calculations of Betti numbers are carried out using the Computational Homology Project (CHomP) software package.[5]

To examine the Betti numbers for various film thicknesses, we find it convenient to normalize them by the area corresponding to the most unstable wavelength, λ_m. Therefore, we define normalized Betti numbers, $\grave{b}_0$ and $\grave{b}_1$ by $\grave{b}_{0,1} = b_{0,1}/P^2$ where $P\lambda_m$ is the linear domain size, see (4.7). The normalized Betti numbers give a measure of the number of features (components, loops) per λ_m^2, and will be shown as functions of the scaled threshold value H^*/H_0, where H_0 is the average film thickness, and the rescaled time, τ.

4.3.1. Evolution of films exposed to infinitesimal perturbations of global character

Here we analyze the evolution of randomly perturbed films of thicknesses H_0 that are in the linearly unstable regime. The initial condition is set to a flat film that has been randomly perturbed, and to excite all modes in the 2D Fourier transform independently (combinations of q_x and q_y), pseudo-Perlin noise is used. Specifically, the initial condition is of the form

$$u(x, y, t = 0) = H_0(1 + \epsilon\,|\zeta(x, y)|)\,, \quad (x, y) \in [0, P\lambda_m] \tag{4.7}$$

where $\lambda_m = 2\pi/q_m$, q_m is given in (1.5), and $\zeta(x, y)$ is the inverse Fourier transform of

$$\zeta(q_x, q_y) = \left| \left[q_x^2 + q_y^2\right]^{-\alpha/2} \exp\left(2\pi\,\mathrm{i}a(q_x, q_y)\right) \right| . \tag{4.8}$$

Here, $\epsilon = 0.01$, α is some positive constant, and $a(q_x, q_y)$ is a random variable, uniformly distributed on $[-1, 1]$, for each (q_x, q_y). In addition, $\zeta(x, y)$ is scaled so that $|\zeta(x, y)| \leq 1$ and we fix $\alpha = 200/I$, where I is the number of discretization points in the x and y directions, so that the spatial scale of the noise is proportional to λ_m.

Figs. 10 and 11 show the results for the thinnest film that we consider, $H_0 = 0.05$, and for times $\tau = 5$ and 20, respectively. The central parts of the figures show the free surface thickness. The corresponding radial Fourier Transform is shown at the top right. We see that the most unstable wavelength dominates the morphology of the film at $\tau = 5$. On the bottom right, the local maxima of the radial Fourier transform are plotted as functions of time, showing consistently that for $1 < \tau < 6$, q_m dominates the wave pattern. At later times drops form and coarsening is observed in the radial Fourier transform. Similar coarsening effects are also observed in Volume of Fluid based simulations of nanoscale metal films [42]. Additional simulations (figures not shown for brevity) uncover consistent results for all linearly unstable film thicknesses considered, in the range $[0.05 : 0.8]$.

Figs. 10 and 11 also plot the normalized Betti numbers $\grave{b}_0$ (top left) and $\grave{b}_1$ (bottom left) as contour plots with independent variables being H^*/H_0 (vertical axis) and the scaled time, τ (horizontal axis). (Note that the only difference between the Betti plots shown in these two figures is the position of vertical (magenta) lines, that show the current value of τ depicted in the central part of the figures.) The basic interpretation of these Betti numbers figures is straightforward for both early and late times. For the early times, $\tau \lessapprox 3$, the perturbations are small and are not captured by the level of thresholding

[5] http://chomp.rutgers.edu/.

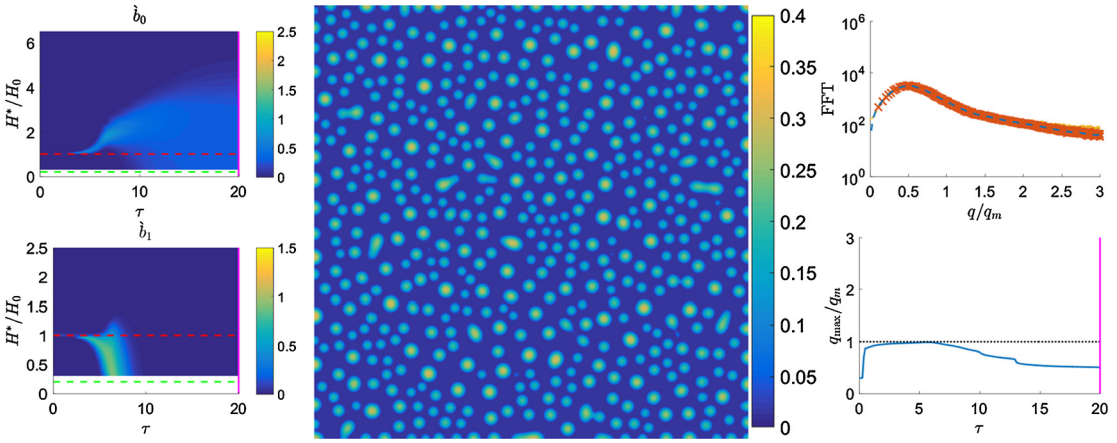


Fig. 11. Film of thickness $H_0 = 0.05$ perturbed by global perturbations. The central part shows a contour plot of the free surface height at $\tau = 20$. The rest of the figure is as described in the caption of Fig. 10. The grid size is 1637×1637. For animations, see Appendix C, movie1.

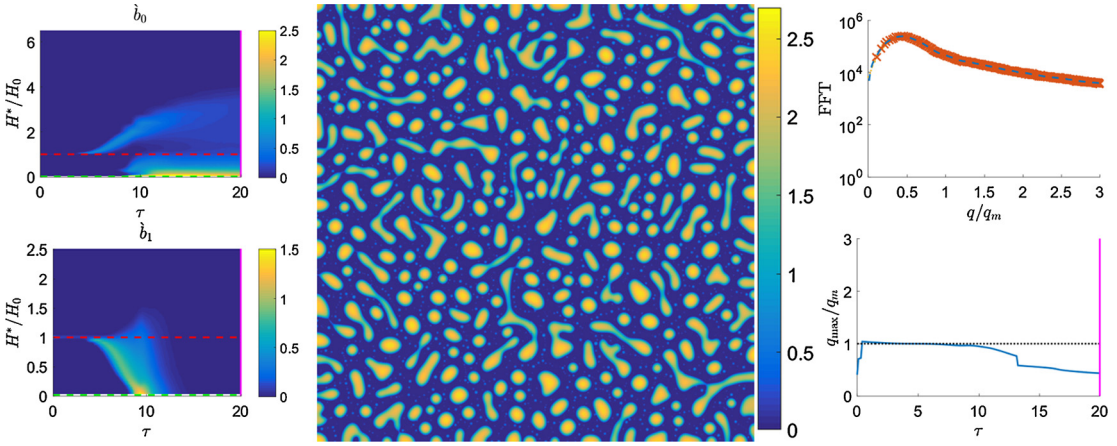


Fig. 12. Film of thickness $H_0 = 0.6$ perturbed by global perturbations. The central part shows a contour plot of the free surface height at $\tau = 20$. The rest of the figure is as described in the caption of Fig. 10. The grid size is 3882×3882. For animations, see Appendix C, movie2.

implemented; therefore both $\grave{b}_0$ and $\grave{b}_1$ are small (the exact values depend on the treatment of the boundaries, but this is not of particular interest here). For long times, $\tau \gtrapprox 10$, the drops form and there are no separate valleys/loops (they are all connected by the precursor); therefore $\grave{b}_1$ is small. For H^* in the range $1.5b$ (the smallest value considered) to $\approx 4H_0$, for long times we see essentially constant values of $\grave{b}_0$, suggesting that the most of the drops are of approximately similar size; small values of $\grave{b}_0$ for $H^*/H_0 \gtrapprox 4$ show that there are essentially no drops exceeding height $4H_0$.

For intermediate times, we observe a peninsula-like light blue structure in $\grave{b}_0$ (top row), the tip of which starts at $\tau \approx 3$ and $H^*/H_0 \approx 1$. This corresponds to a local increase of film thickness (therefore formation of isolated hill tops, that are not yet drops). These hill tops are associated with the formation of surrounding valleys (loops), indicated by an increase of $\grave{b}_1$ around the same time. For slightly longer times, following the contours of $\grave{b}_0$ and $\grave{b}_1$, we observe an increase in the height of the drops, measured by $\grave{b}_0$; and the formation of deeper and deeper valleys, measured by $\grave{b}_1$. The evolution is essentially complete by $\tau \approx 10$, at which time we reach the long time regime already discussed above. We note that one important utility of Betti numbers is that they provide an insight into the topology of emerging patterns across a range of thresholds, allowing one to follow the details of instability development.

Our previous analysis of 2D films [28] showed the presence of satellite drops persisting on a timescale longer than that of dewetting ($\tau \gg 1$) for those linearly unstable film thicknesses corresponding to positive values of the disjoining pressure [28]. For the present choice of parameters, $\Pi(H_0) = 0$ for $H_0 = 0.2624$, see Fig. 8 [28]. We are now in a position to discuss whether the satellite drops observed in 2D simulations persist in the 3D geometry. Fig. 12 shows the results for $H_0 = 0.6$, for which $\Pi(H_0) > 0$. We note that the Fourier transform (in particular, the dominant wave number) shows results similar to those for thinner films presented in Figs. 10 and 11. However, there is a significant difference in the normalized Betti numbers for threshold values near the precursor thickness; specifically we note the presence of the thin rectangular bright yellow region in $\grave{b}_0$ bounded by $10 < \tau < 20$. This region corresponds to the presence of features characterized by

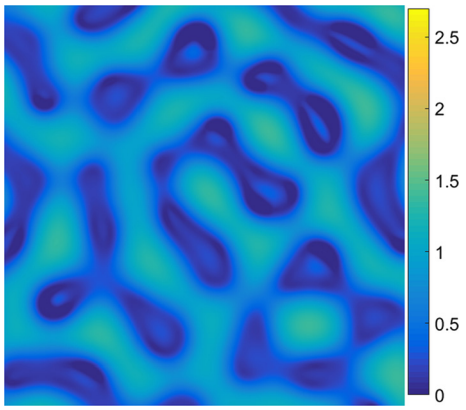


Fig. 13. Zoom-in for the film shown in Fig. 12 at $\tau = 9$. Linear domain size is $5\lambda_m$ and the sub-grid size is 485×485. Note formation of satellite drops (light blue islands) in the centers of growing holes.

heights that are much smaller than H_0: these features are readily identified as small satellite drops. Therefore, our earlier finding that positive values of the disjoining pressure lead to the existence of satellite drops for 2D films, is found to extend to 3D films. We note in passing that the configuration shown in the central part of Fig. 12 at $\tau = 20$ is still evolving; the evolution has not yet concluded. For longer times, the main drops form and the satellite drops persist for the entire simulated time interval ($\tau = 40$).

The results for $\grave{b}_1$ provide additional insight about satellite drops and the nature of film breakup. Note the bright yellow (approximately) circular region in the plot of $\grave{b}_1$ in Fig. 12, bounded by $9 < \tau < 10$, which indicates the formation of holes for $H^*/H_0 \approx 0.1$. It turns out that this region is due to formation of satellite drops in the centers of holes, which formed due to the film instability. To illustrate this process further, Fig. 13 shows a small part of the domain where one can clearly see these satellite drops (see also Supplementary Material Appendix C). These drops are surrounded by a thinner film layer, leading therefore to a temporary increase of $\grave{b}_1$. For later times, the holes disintegrate, and $\grave{b}_1$ decreases to small values again.

4.3.2. Evolution of films exposed to both localized and global perturbations

We now switch focus to linearly unstable film thicknesses exposed to both localized and global (random) perturbations. Previous results for 2D films [28] show that a localized perturbation propagates into the flat regime, dewetting the film and successively forming drops. Based on the average distance between the drop centers, four regimes were identified numerically: regions R I and R III, where the mean distance between the drop centers is characterized by λ_m, the most unstable wavelength (spinodal dewetting); and regions R II and R IV, where the mean distance between drops is not characterized by λ_m (nucleation dominated regimes). These stability regions, for 2D films, can be related to the disjoining pressure as shown in Fig. 8. In this section we will use these 2D results as a guide to analyze 3D film instability in the presence of both local and global perturbations.

To begin, we consider the evolution of a flat film of thickness H_0, perturbed by global random perturbations, with superimposed larger localized perturbations, with a specified average mean distance apart. Specifically, the initial condition is given by

$$u(x, y, t = 0) = H_0(1 + \epsilon|\zeta(x, y)| + 10\epsilon\eta(x, y)) , \tag{4.9}$$

where $\zeta(x, y)$ is the pseudo-Perlin noise (4.8), and localized perturbations are defined by $\eta(x, y)$. We will not attempt here a careful study of the effect of the choice of functional form $\eta(x, y)$ on results, but consider instead just a single realization that specifies the functional form of each perturbation in a manner similar to [28], with perturbations distributed at an average distance $\bar{d}$ that is large compared to λ_m. As an illustrative example, we choose $\bar{d} = 8$ in the present study, a choice motivated by our desire to place the local perturbation centers at a reasonable distance from each other, while simultaneously allowing a relatively large number of such perturbations. The form chosen for $\eta(x, y)$ is

$$\eta(x, y) = \sum_{i=1}^{4}\sum_{j=1}^{4} \exp\left(-\frac{(x - \chi_i)^2 - (y - \xi_j)^2}{0.04\lambda_m^2}\right) , \tag{4.10}$$

where $\chi_i = (8i - 4 + 4a_i)\lambda_m$, $\xi_j = (8j - 4 + 4b_j)\lambda_m$, and a_i and b_j are random variables uniformly distributed on $[-1, 1]$ (i.e., we generate a two-dimensional array of localized perturbations with mean distance $8\lambda_m$ apart). We note that the motivation for considering random global perturbations alongside local ones is that random perturbations are always inevitably present in experiments, hence carrying out simulations where both local and global perturbations are present brings us closer to understanding experimental results.

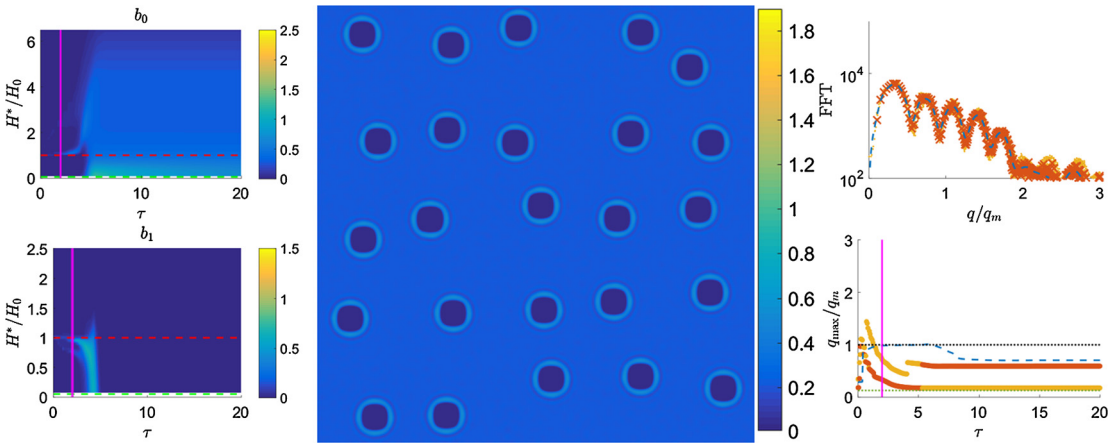


Fig. 14. Film of thickness $H_0 = 0.2$ perturbed by local and global random perturbations as described in the text. The central part shows a contour plot of the free surface height at $\tau = 2$. The corresponding radial Fourier transform (top right) of the free surface height in the center of the domain, and the large local maximum (red dots) and second largest local maximum (yellow dots) of the radial Fourier transform (bottom right) are plotted as functions of τ. Here the dashed blue curve corresponds to the local maximum of the radial Fourier transform for a simulation carried out with the same film thickness, but with random perturbations only, and the horizontal green line is the wave number associated with the average distance between localized perturbations. Left panels plot the Betti numbers $\grave{b}_0$ (top) and $\grave{b}_1$ (bottom). The vertical magenta line corresponds to the solution time in the respective panels. The grid size is 1970×1970. For animations, see Appendix C, movie3.

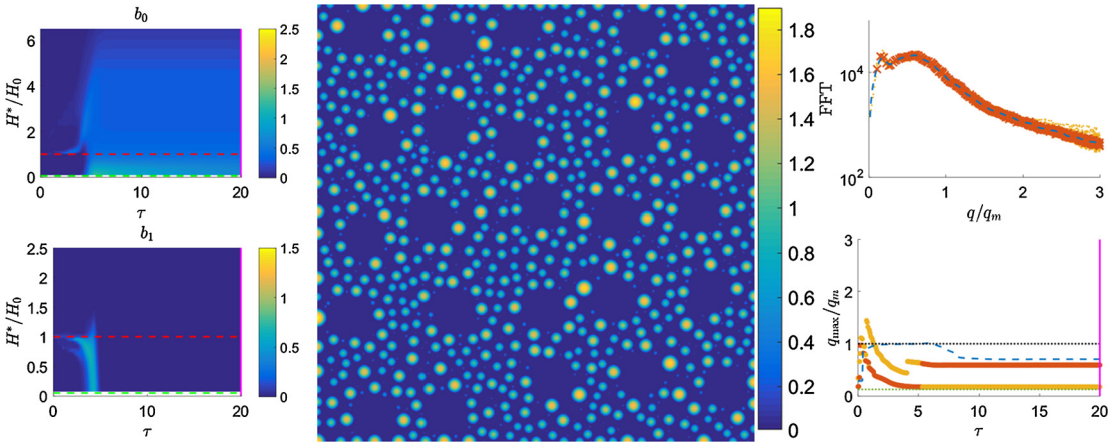


Fig. 15. Film of thickness $H_0 = 0.2$ perturbed by local and global perturbations as described in the text. The central part shows a contour plot of the free surface height at $\tau = 20$. The rest of the figure is as described in the caption of Fig. 14. The grid size is 1970×1970. For animations, see AppediX C, movie3.

Fig. 14 shows the simulation results for $H_0 = 0.2$ (nucleation dominated regime) at $\tau = 2$. The figure is presented after the fashion of Figs. 10–12, except that Fourier spectra figures (right hand side) also show the results obtained in the corresponding simulation with only random perturbations applied (dashed blue lines). From the central panel of this figure, it may be seen that the localized perturbations grow into large holes and dominate the radial Fourier transform for $1 < \tau < 5$. To further clarify the influence of random perturbations, in the Fourier transform plots (right hand side) we plot not only the main maximum (red dots), but also the second largest maximum, if well defined (yellow dots). For the early times shown in the central panel of this figure, the dominant mode corresponds to the long wavelengths that are due to the localized perturbations. In Fig. 14 (and in Fig. 16) the second maximum is plotted only if its amplitude is at least 5% of the largest maximum.

Fig. 15 shows the results of the same simulation at a later time. We note that, while drops form, the holes are still visually distinguishable (see central panel). Furthermore, by considering Fourier spectra again, we observe that for $\tau > 5$, there is a switch in the dominant wave number from the long wavelength one (corresponding to localized perturbations) to the shorter wavelength one corresponding to random perturbations; the latter is clearly very close to that found if only random perturbations are imposed (see green dashed line in the bottom right panel of Fig. 15). These results indicate that it is possible to identify the nucleation-dominated regime by considering Fourier transforms, as long as the considered domains are sufficiently large to be able to extract the second, smaller, wavenumber from the radial Fourier transform. We note that a more elaborate approach, based on the use of Minkowski functionals, was considered for this purpose previously [8]. We should also note that, as with our previous work for 2D films [28], the nucleation-dominated regime is

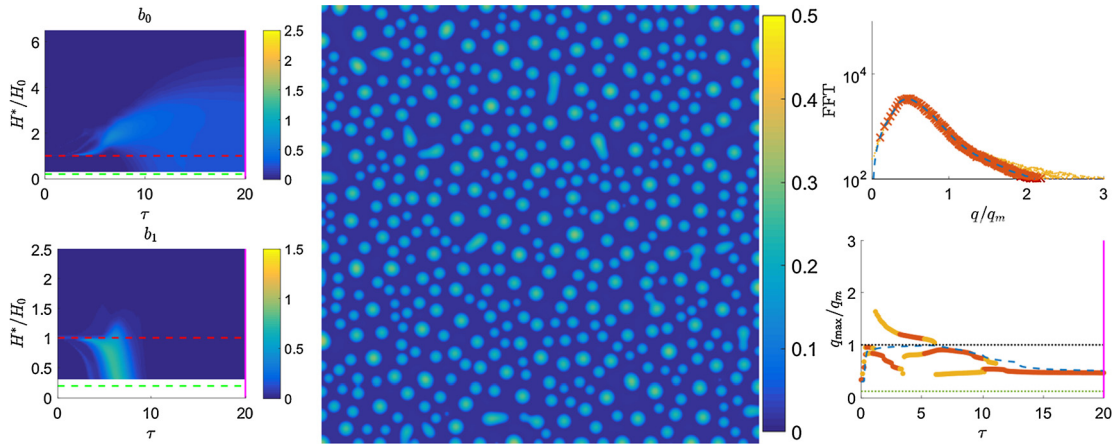


Fig. 16. Film of thickness $H_0 = 0.05$ perturbed by local and global perturbations. The central part shows a contour plot of the free surface height at $\tau = 20$. The rest of the figure is as described in the caption of Fig. 14. The grid size is 1637×1637. For animations, see Appendix C, movie4.

confirmed only numerically: we do not yet have an analytical framework for identifying this regime. For all considered film thicknesses, the coarsened wavenumber appears much earlier than in the simulations with global random perturbations only (compare yellow/red dotted lines with the blue dashed line in Fig. 14); this may be due to the localized perturbations being much larger than global random perturbations, initializing dewetting at an earlier time.

Fig. 16 plots, for comparison, the results for $H_0 = 0.05$ (spinodal dominated regime) at $\tau = 20$. The results show that for long times, the localized perturbations have negligible effect on the film morphology, and only a single wave number (which is the same as in the simulation with random perturbations only) dominates the wave pattern. Simulations were also carried out for $H_0 = 0.3$ (nucleation-dominated) and $H_0 = 0.6$ (spinodal-dominated), confirming the results (figures not shown for brevity). Therefore, our simulations are able to distinguish between spinodal-dominated and nucleation-dominated regimes. This achievement is difficult if not impossible to reach without resorting to the kind of large-scale simulations presented here.

Next we discuss briefly the Betti numbers for the simulations where both local and global perturbations are present, with the focus on a few revealing features of the results. The comparison of results for the Betti number $\grave{b}_0$ between Figs. 14 and 16 shows two main differences. First, there are some much larger (taller) drops in the nucleation regime: in Fig. 14, top left panel, we see a relatively large number of components (drops), even for thresholds as large as $5H_0$, while in the spinodal regime shown in Fig. 16, such tall drops are not present. The second difference, which is perhaps even more interesting, is the gradient (with respect to the threshold value) of the number of components (drops) for films in the nucleation regime, see Fig. 14, top left panel. We see a larger number of components (drops) for smaller values of the threshold, all the way down to the precursor film thickness. This finding suggests (i) larger differences in the drop sizes in the nucleation regime; and (ii) a significant presence of small satellite drops, similar to the RII regime in the case of global perturbations only, see Fig. 12. Further inspection (see also Appendix C) shows that the variety of drop sizes, including rather small ones, is due to breakup of the ridges between growing holes. Furthermore, there are no satellite drops forming inside of the growing holes, in contrast to the results shown in Fig. 12.

Fig. 17 shows another example of interesting and non-trivial dynamics. Here, the expanding fluid film breaks up into a series of ridges. The innermost of these ridges stops its expansion and reverses its motion, collapsing back into itself. This process resembles the collapse of liquid rings, discussed recently by Gonzalez et al. [43]. The outcome is the formation of a large drop, instead of a small satellite at a perturbation center. This type of unusual dynamics is referred to below as an 'expanding/collapsing ridge'.

In our previous work considering the 2D geometry [28] we have shown that the evolution of unstable films may be very complex; some aspects of that complexity could be analyzed in detail in 2D since we were able to run simulations in large domains for long times. Doing such detailed simulations is not feasible in 3D, even with our GPU code, so we limit ourselves here to illustration of the main features of the results. Still, the simulations that we have carried out for various film thicknesses and perturbation types produce consistent results, in particular with respect to the sign of the disjoining pressure, the instability type, and the type of perturbation applied. Table 3 illustrates some of these common features. While new simulations may uncover additional features of the results and patterns that form, we believe that the generic features of the results described so far already provide significant insight.

We also comment briefly on the speed at which the localized perturbation propagates into the linearly unstable flat film regime investigated in previous work for 2D geometry [28]. Our previous work applied the Marginal Stability Criterion (MSC) to derive an analytical expression for the speed at which the localized perturbation propagates (the details are beyond the scope of the present paper and we refer the reader to other works [28,44] for a more detailed discussion). However, we note that when we extract this speed from simulations of a single localized perturbation for $H_0 = 0.2$ and 0.6 in 3D geometry, the numerically-extracted speeds agree with the analytical MSC result, demonstrating that the MSC result carries over from

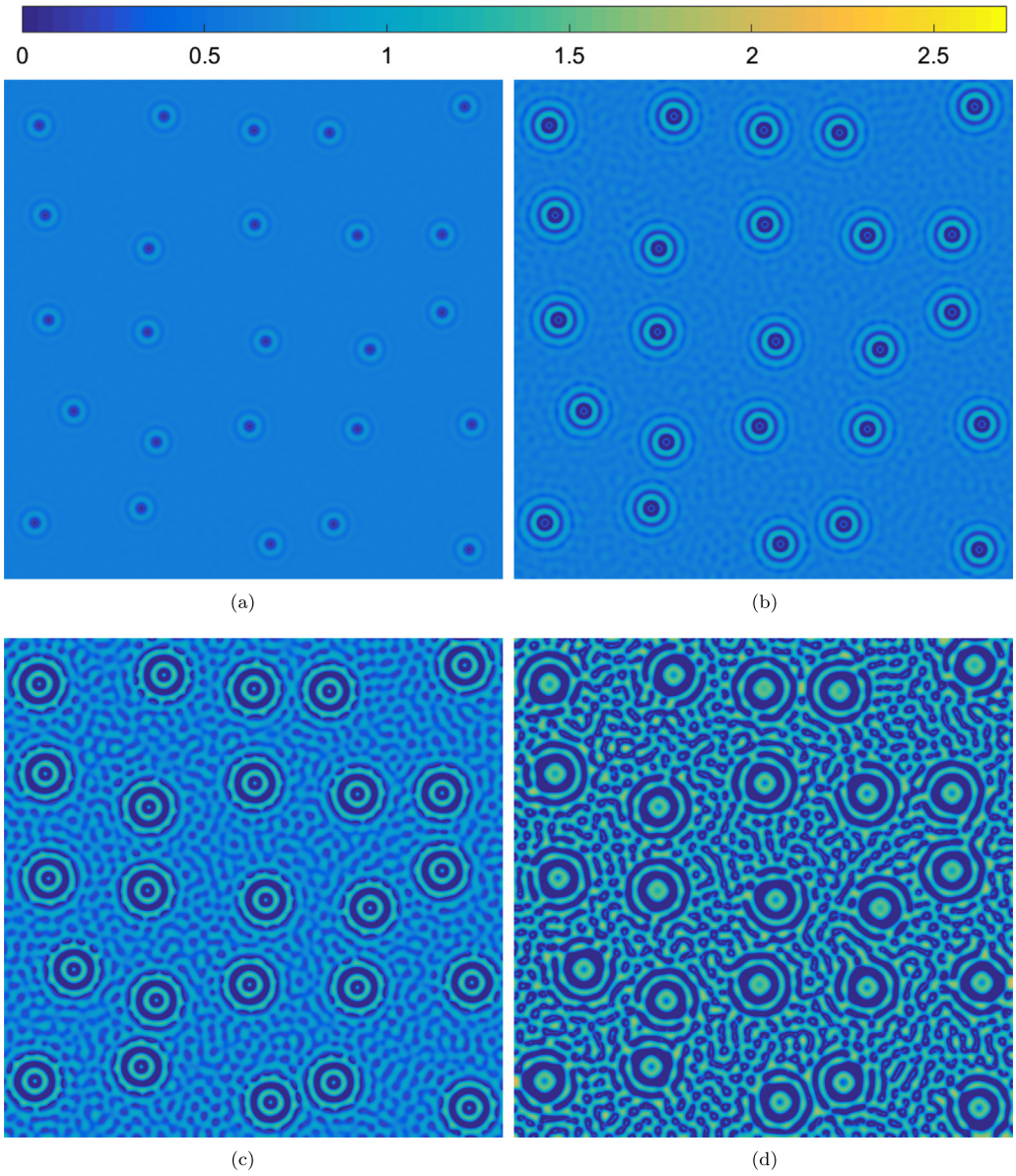


Fig. 17. Film of thickness $H_0 = 0.6$ perturbed by local and global perturbations shown at $\tau = 2.5,\ 5.0,\ 7.5,\ 10$. Note the expanding/collapsing ridges close to the perturbation centers, resulting in formation of large drops there at the last time shown. The grid size is 3882×3882. For animations, see Appendix C, movie5.

the 2D geometry. In addition, as noted in our previous work [28], MSC can also be applied to gravity driven flow of a NLC film down an inclined substrate [45,46] (where van der Waals forces are ignored); to flow of Newtonian films down an inverted substrate [11,47] and down the outside of a vertical cylinder [48]; and may also be applied to other models such as the Cahn–Hilliard and Kuramoto–Sivashinsky type of equations [44].

4.3.3. Metastable regime

Here we briefly investigate the metastable regime previously studied for 2D films [28]. This regime corresponds to film thicknesses that are linearly stable; but unstable with respect to perturbations of finite amplitude. Previous analytical results have shown that there exists a thickness regime, $H_0 \in (H_+, H_{m_3}) \approx (1.01, 3.04)$ (using the notation from [28] here) that is metastable, and was derived analytically by extending previous results for a Newtonian flow under gravity [49]. Furthermore, for 2D films, extensive numerical simulations have shown that for an initial condition of the form

$$h(x, y, t = 0) = H_0 \left[1 - d \exp\left(-\frac{x^2}{0.04 W^2} \right) \right], \quad W = 4\pi , \tag{4.11}$$

Table 3
Summary of the results for linearly unstable films.

H_0	Perturbation	Regime	Π	Satellites	FFT	Comment
0.05	global	spinodal	<0	no	single mode	agreement with LSA and long time coarsening
0.05	local	spinodal	<0	no	single mode	no influence of local perturbations
0.05	mixed	spinodal	<0	no	single mode	no influence of local perturbations
0.2	global	nucleation	<0	yes	single mode	satellites due to ridge breakup
0.2	local	nucleation	<0	yes	bimodal	satellites due to ridge breakup
0.2	mixed	nucleation	<0	yes	bimodal	satellites due to ridge breakup
0.3	global	nucleation	>0	yes	single mode	satellites at perturbation centers and due to ridge breakup
0.3	local	nucleation	>0	yes	bimodal	satellites at perturbation centers and due to ridge breakup
0.3	mixed	nucleation	>0	yes	bimodal	satellites at perturbation centers and due to ridge breakup
0.6	global	spinodal	>0	yes	single mode	satellites at perturbation centers and due to ridge breakup
0.6	local	spinodal	>0	yes	single mode	satellites at perturbation centers and due to ridge breakup
0.6	mixed	spinodal	>0	yes	single mode	satellites at perturbation centers and due to ridge breakup

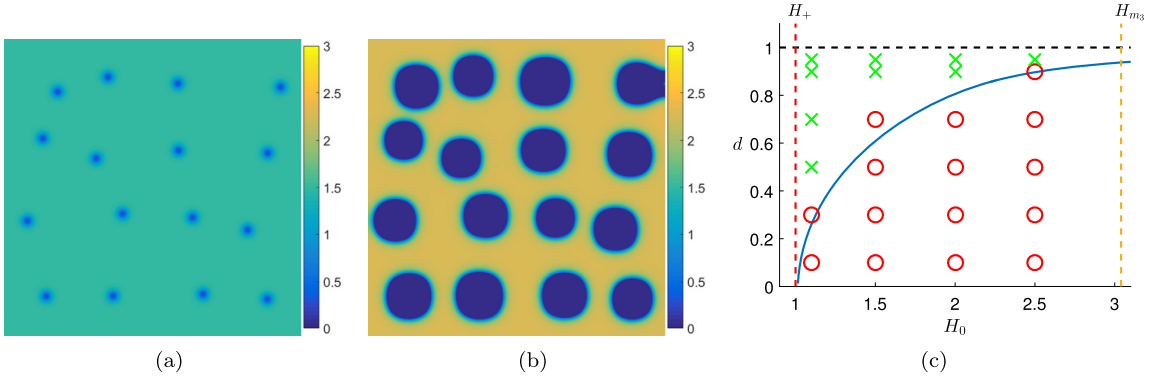


Fig. 18. Contour plot of metastable film, with (a) the initial condition given by (4.12), and (b) the free surface at $t = 1000$. (c) Metastability results for initial condition (4.11) as a function of the initial film thickness, H_0, and initial magnitude of the localized perturbation, d. The blue curve denotes d_c, the minimum value of d required to induce dewetting in 2D films [28]. For the 3D films, $\times$ in green font denotes initial condition parameters that induce dewetting, and $\bigcirc$ in red font denotes parameters that do not induce dewetting. The grid size is 2000×2000. For animations, see Appendix C, movie6.

for dewetting to occur, the minimum magnitude of the perturbation required for dewetting to occur, d_c, approaches 1 as $H_0 \to H_{m_3}$. In other words, for films of thickness close to H_{m_3}, the initial perturbation has to essentially reach the precursor thickness to produce dewetting.

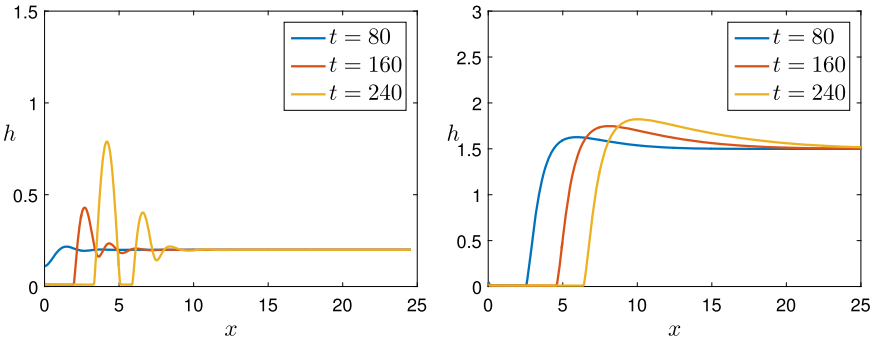
For 3D films, we use an initial condition of the form

$$h(x, y, t = 0) = H_0 [1 - d\eta(x, y)] \,, \tag{4.12}$$

where the linear domain size is 100, $H_0 = 1.5$, $d = 0.85$, and $\eta(x, y)$ is given by (4.10) with $\lambda_m = 2.5$ (for convenience, we use the value from (4.10) although λ_m is not relevant for metastable films). Fig. 18(a) shows the contour plot of the initial condition. The imposed local perturbations grow, leading to the profile characterized by growing holes shown in Fig. 18(b) at $t = 1000$ (note that in the linear stable regime, ω_m is undefined, therefore, we use t instead of τ in this section). We note that the profile shown in this figure is transient: the holes continue to expand for later times. In this work we focus only on the regime during which the perturbations evolve essentially independently of each other.

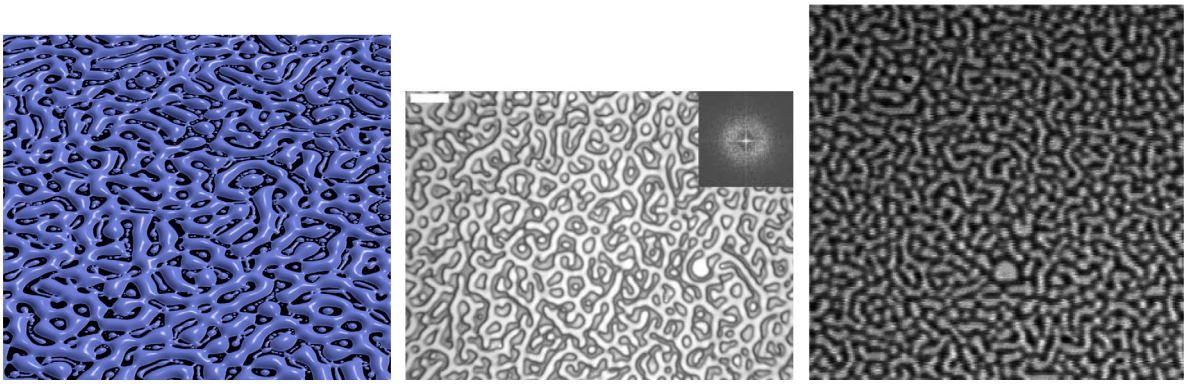
Fig. 18(c) focuses on the comparison between the minimum thicknesses, d_c, needed to cause dewetting, for 2D and 3D films. In 3D, we carry out simulations with a single perturbation of the form specified by (4.12), with the perturbation centered in the middle of the domain. In Fig. 18(c), crosses and circles denote the parameters that do/do not induce dewetting, respectively, in 3D, and the blue curve shows the analogous (analytical) results for 2D films. It may be seen that the 2D findings essentially extend to 3D. The minor differences seen are to be expected, due to the different geometry of perturbations in 2D and 3D: a localized perturbation in 2D is essentially a 'trench' since the perturbation is invariant with respect to the y direction.

Finally, we briefly comment on the shape of the expanding fronts away from localized perturbations for linearly unstable nucleation-dominated films versus metastable ones. Fig. 19 shows a few cross-sectional snapshots of one of perturbations from Figs. 14 and 18. The expanding front for a linearly unstable film is characterized by the presence of a well-developed capillary ridge, followed by an oscillatory damped profile, similar to the 2D films considered previously in [28]. For metastable films, however, the ridge is much less prominent and the connection to the rest of the film is monotonous. We are not aware that these different front shapes have been discussed in this context in the literature so far.



(a) Linearly unstable regime, $H_0 = 0.2$. (b) Metastable regime, $H_0 = 1.5$.

Fig. 19. Cross-section of the film profile as a function of the distance from localized perturbation centered at $x = 0$. The grid sizes are (a) 492×492 and (b) 750×750.



(a) Our simulation (b) Schlagowski *et al.* (2002) (c) Vandenbrouck *et al.* (1999)

Fig. 20. (a) Simulation result for a 50 nm ($H_0 = 0.5$) thick film of 4-Cyano-4′-pentylbiphenyl (5CB) nematic liquid crystal vs experimental results for (b) a 85 nm thick 4-Octyl-4-Cyanobiphenyl (8CB) film and (c) a 43 nm thick 5CB film. The grid size in a) is 3400×3400.

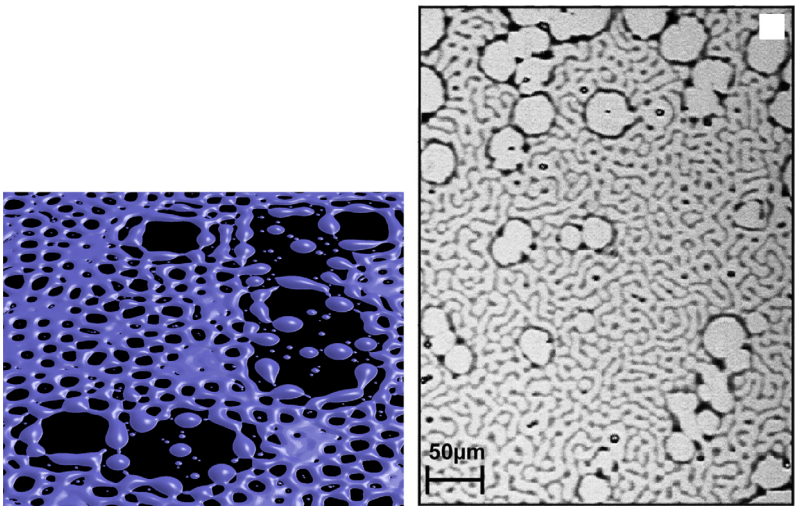
5. Comparison to experimental results

In this section, we compare simulation results for NLC and polymeric films to available experimental results. The disjoining pressures considered are given by (1.8) and (1.12), respectively. To scale the domain and initial condition for different film thicknesses, simulations are performed on a square domain, and the linear domain size is scaled with λ_m. In addition, we fix $\Delta s = 0.05$ for both models. The initial condition is given by (4.7). For all simulations in this section, we fix $P = 40$.

The first set of experimental results considered is dewetting experiments for NLC films by Schlagowski et al. [22] and Vandenbrouck et al. [34]. In the experiments by Vandenbrouck et al. [34], a flat film is formed by increasing the temperature of the NLC such that a flat film is stable at the considered film thickness of interest (isotropic phase). The flat film is formed by using a spin coating method, and then cooled to a temperature so that it becomes unstable (nematic phase). Schlagowski et al. [22] varied the temperature of the NLC sample, alternating between the isotropic and nematic phases, observing the morphology of the film as the temperature crosses the threshold between the isotropic phase and nematic phase. Fig. 20 compares a 3D rendering of our numerical results to experimental results of two different types of NLC. In both experiments, the characteristic wavelength of undulations is appropriately 30 μm, which is similar to $\hat{\lambda}_m = 42$ μm for the parameters stated in (1.10) and (1.11).

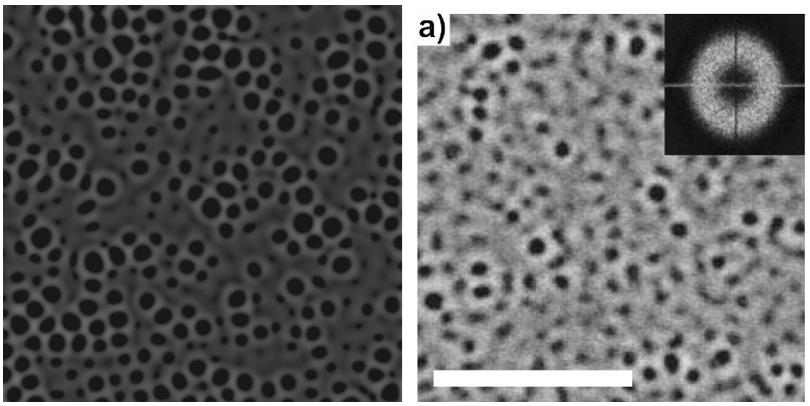
The next experiment we consider is that by Herminghaus et al. [20], who prepared flat films of NLC in a Langmuir trough. Unlike the spin coating method, transferring the flat film from the Langmuir trough to the solid substrate may induce defects in the film (localized perturbations). To simulate this initial state, large localized perturbations at random locations are added to the initial condition (4.7). Fig. 21 demonstrates that the localized perturbations can induce the formation of large holes, leading to patterns that are visually similar to the experimental images.

The last experiment we consider is a dewetting experiment with polymeric films by Jacobs et al. [18]. Fig. 22 compares simulation results with experimental results for a 3.9 nm thick polymeric film on a silicon substrate coated with a 191 nm layer of silicon oxide. It may be seen that the dominant wavelength in our simulations matches the experimental results; however, the timescales of dewetting differ. This difference may be due to the choice of the initial perturbation size.



(a) Our simulation (b) Herminghaus *et al.* (1998)

Fig. 21. (a) Simulation result for a 20 nm ($H_0 = 0.2$) thick film of 4-Cyano-4′-pentylbiphenyl (5CB) nematic liquid crystal film vs (b) an experimental result from Herminghaus et al. [20] for a 40 nm thick tris(trimethylsiloxy) silane-ethoxycyanobiphenyl ($5AB_4$) film. The grid size in a) is 1970×1970.



(a) Our simulation (b) Jacobs *et al.* (2008)

Fig. 22. Comparison between (a) our numerical simulations at $t = 60$ s and (b) the experiment by Jacobs et al. (2008) [18] for a 4.9 nm film on a silicon substrate coated with a 191 nm layer of silicon oxide. Note color scales on both panels are the same; specifically, black corresponds to 0 nm and white corresponds to 20 nm. The linear domain size in both panels is 8 μm and the scale bar for (b) corresponds to 5 μm. The grid size in a) is 1600×1600.

6. Conclusions

In this paper, we describe a GPU implementation of an ADI numerical scheme applied to evolution of thin films in a three dimensional setting. The scheme has been generalized to conservation equations with nonlinear fluxes, and validated by extensive testing of its convergence and conservative properties, as well as by comparison with linear stability analysis. The computational gains obtained by carrying out simulation on a GPU are substantial, and allow for carrying out simulations in large domains with rather basic computational resources.

The main set of results presented in this paper focuses on three dimensional (3D) simulations of unstable films. We focus in particular on instabilities of nematic liquid crystal films, and to a lesser extent, those of spreading polymer films. Direct comparison of the computational results to available experimental data has been carried out, with striking visual similarity observed between the simulations and experiments.

The large scale simulations that we present in the paper have uncovered a number of results. Regarding linearly unstable films, we find that both evolution of instability and the final outcome may depend strongly on the source of the instability, and on film thickness. In particular, films that are dominated by a spinodal instability mechanism lead to formation of drops whose typical distances and sizes are consistent with the predictions of linear stability analysis, independently of the source of instability. In the nucleation dominated regime, however, we observe the signature of the source of instability both in the manner in which the instability evolves, and in the final outcome. While we have not presented an extensive study of this

aspect of the results, our results suggest that one may be able to deduce the instability source by observing the outcome. Attaining such a goal requires large computational domains, as used in this work, or comparably large experimental domains.

Another outcome of our results that is made possible by large computational domains is analysis of satellite drop formation. Our previous works [28] on 2D films predicted formation of satellite drops during breakup of films characterized by positive values of (effective) disjoining pressure. The 3D results in the present paper are consistent with the 2D results, showing that the number of physical dimensions does not influence the basic features of instability development. The details of the results are influenced by the geometry however; in particular, we find two possible ways for satellite drops to form: they can form either at the centers of imposed perturbations (equivalently, at the centers of the expanding holes), or they can form during the process of ridge breakup. We find that both the instability regime, and the sign of the disjoining pressure, influence the process of satellite drop formation, with only the thinnest of the considered films not showing satellites. In some cases, typically for thicker films, we also observe expanding/collapsing ridges – a ridge growing radially away from a perturbation center may detach from the rest of the film and collapse back to its center due to azimuthal curvature.

For the purpose of analysis of results, we have found that careful Fourier analysis combined with computing topological invariants (Betti numbers) provides insight. Betti numbers, in particular, are computed across all relevant film thicknesses, giving a detailed picture of the film evolution.

We have also analyzed metastable films, which are linearly stable, but unstable with respect to finite size perturbations. The most interesting aspect of the results here includes the distinctive shape of the expanding holes due to imposed local perturbations. While for linearly unstable films one observes the formation of capillary ridges at the expanding fronts, for metastable films we find a monotonous increase of thickness from the equilibrium layer to the scale of the original film thickness.

There are several possible avenues for extending the work presented in this paper. Regarding satellite drop formation, and the nucleation-dominated and metastable regimes, investigation of other thin film models would verify the generalization of our results in terms of a form of disjoining pressure. There are also several possible improvements that may be made to our GPU code, for example, a multi-GPU implementation; implementing adaptive mesh refinement; or improving numerical accuracy. Another important improvement would be to improve the parallelization of the linear solver using techniques such as the SPIKE method [50] or cyclical reduction [51,52].

Regarding future applications of the presented computational methods, there are numerous possible directions. In our work, we have so far considered relatively simple setups where the film evolution could be described by a single partial differential equation. An extension of the presented approach to problems involving multiple miscible or immiscible films, or to problems involving thermal effects, for example, is possible.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. GPU implementation

In this Appendix, the specifics of the GPU implementation are discussed. To begin, we give an overview on key concepts of GPU computing utilized in our implementation. The GPU is a vector type processor which processes data in the Single Instruction, Multiple Data (SIMD) framework of parallel computing [53], i.e., the same finite difference scheme (single instruction) is applied at each grid point (multiple data). A GPU consists of several multi-core processors (MPs) which share a *global* memory bank (analogous to the CPU memory). Analogous to the levels of cache memory on CPUs,[6] each MP has a fast (compared to global memory) *shared* (between cores of a MP) memory bank, and an even faster *register* memory (of the individual cores); however, unlike CPU cache memory, the shared memory bank has to be explicitly handled in a code in order to reduce redundant access to the (slow) global memory. Use of register memory is automatic when defining variables within the scope of the kernel (similar to the automatic use of cache memory on a CPU). Typically, the shared memory bank is used to store data that will be reused by other threads within a thread block, whereas register memory is used to store values that will only be used by an individual thread.

In CUDA, instructions on the GPU (*device*) are executed from the CPU (*host*) in terms of *kernel* calls. There are two input parameters unique to a kernel call, each composed of 3 numbers, specifying the size of a vector, matrix, or tensor. The purpose of these parameters is to divide a domain into sub-lengths, sub-squares or sub-cubes; and assign each *sub-domain* a vector, matrix, or tensor of threads, which we will refer to as a *thread block*. The input parameters provide an intuitive way of sub-dividing a multi-dimensional domain for parallelization (sub-domains), and a natural (geometric) way of mapping threads to points on the sub-domain. On the hardware level, threads within a thread block are further sub-divided into groups called warps (32 threads to each warp for the GPU used in this paper). Instructions to the MPs are batched in terms of warp (vectors) and are executed in the SIMD architecture framework.

[6] Technically, the shared memory bank is not a level of cache memory, and MPs already have a L1 and a L2 cache (register memory).

The last important concept for GPU computing (for our implementation) is *coalesced* data access to the global memory, i.e., adjacent data values that are adjacent in memory may be accessed in a single SIMD transaction (as opposed to multiple transactions), improving efficiency. This is an important issue for the ADI method, in particular, solving the system of penta-diagonal matrices in each direction, the details of which will be discussed in A.2.1.

Note 1: It is important to note that to fully utilize all threads on a MP, the size of the thread block is typically an integer multiple of the warp size.

Note 2: Since warps are executed in the SIMD framework, warps with threads with different instructions (*warp divergence*) are not computed in parallel, but each different instruction is executed in series, e.g., for n different sets of instructions, the warp is computed in n serial steps (each sub-group of threads, within the warp, with the same instructions, are computed in parallel). Ghost points remove warp divergence at the boundaries; however, this is only an order $I + J$ improvement (the complexity of the complete scheme is $I\,J$).

A.1. GPU and CPU interactions

In addition to the three levels of GPU memory, the GPU may also access the CPU memory; however, the access time for this level of memory (with respect to the GPU memory) is prohibitively slow and may drastically affect the performance of a GPU code, e.g., for simple computations, the time required for transferring the data between the GPU and CPU can be comparable to (or even greater than) the time required to perform the computation on the GPU. Furthermore, data have to be explicitly (in code) transferred between the GPU and CPU, and memory must be allocated on both the CPU and GPU.[7,8] To remove this bottleneck, unless necessary to do otherwise, all computations are performed on the GPU. Recalling the outline of the numerical scheme, controlling the adaptive time stepping is independent of grid size, and is therefore performed on the CPU. Computationally expensive steps (center column of steps in Fig. 1) may be (naively) parallelized on the GPU, with the exception of evaluating the convergence criteria (restrictions on accepting the solution at the new time).

To compute the convergence criteria, we first note that for serial computations, it is simple to determine if the convergence criteria are violated at any of the grid points, i.e., the serial code may iterate through the grid points until one of the convergence criteria is not satisfied; once determined, the appropriate changes may be made to the time step. However, this procedure is slightly non-trivial on a GPU; specifically, a thread cannot terminate the execution of warps in the queue (remaining grid points at which the convergence criteria are to be computed) and control cannot be immediately returned to the CPU. Therefore, once one of the convergence criteria is determined to be violated at a grid point, this information (state) cannot be immediately returned to the CPU, nor can control be immediately returned to the CPU in this manner; thus, all warps must be executed before appropriate changes to the time step can be made. The naive approach is to transfer the data from the GPU's memory to the CPU's memory. However, as mentioned before, this transfer is slow; therefore, a reduction method is used to reduce the amount of data to transfer. In more exact terms, each thread evaluates the convergence criteria at a grid point. The data are reduced on each thread block (sub-domain) by determining if any of the convergence criteria are not met within the thread block, and the reduced data (information for each thread block) are transferred to the CPU.

In our implementation, the size of the thread block is 16×16, thus reducing the total number of data to be transferred by a factor of 256. To reduce further the data volume to be transferred, we do not store the 8 byte double precision value for the point that violates the convergence criteria, but instead store an integer value (an ID representing the convergence criterion that was not satisfied) in a 1 byte char (flag variables). In total, this method reduces the total volume of data to be transferred by a factor of 2048, and the cost of transfer is therefore negligible compared to the GPU computations. The relevant information regarding convergence criteria may then be compressed (reduction) by the GPU, transferred quickly to the CPU, and using a simple loop, the CPU checks the flag on each thread block and adjusts the time-step accordingly.

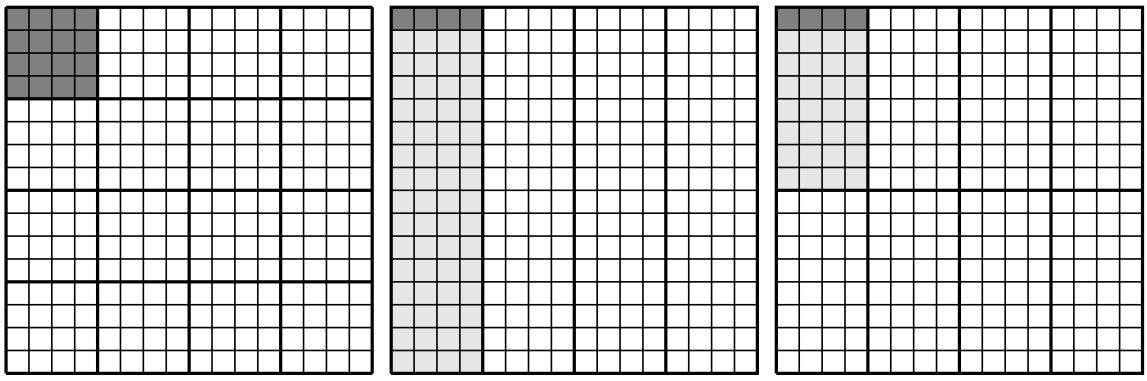
Note 1: To the authors' knowledge, on a GPU, each entry in a boolean array is stored as a byte (8 bits) so is not a true bit array (1 bit per entry); thus there is no reduction in memory usage when using a boolean array instead of a char array. While a bit array could have been implemented, this was not done as: a) memory has to be carefully accessed by threads so as to not incur inefficiencies; b) the use of a char datatype allows information about the failure state on a sub-domain to be passed back to the host CPU (thus logged); and c) the total data size has been sufficiently reduced.

Note 2: The reduction on the GPU may be done recursively, i.e. one could further divide the flags into sub-domains and check whether each flag on the sub-domains (of flags) is equal to one of the convergence condition IDs. However with each recursive step, there are diminishing returns in efficiency (less parallelization due to recursively smaller domain sizes), and furthermore, a sufficient reduction in the size of the data has already been achieved.

Having developed a hybrid method (CPU and GPU) to evaluate the convergence criteria, all computationally expensive operations (center column of steps in Fig. 1) can be parallelized on the GPU, and only a small volume of data (relative to

[7] Newer GPU models support NVDIA's *Unified Memory* model, where explicit transferring of data and allocation of memory on both the CPU and GPU is no longer necessary. Our implementation was originally developed on an older GPU that did not support the Unified Memory model.

[8] Data in memory are typically organized in blocks of fixed length, often called 'pages'. For higher data bandwidth between CPU and GPU, the CPU memory is page-locked to GPU memory (called 'pinned' memory in CUDA), i.e., similar to coalesced data access, page locking provides a direct mapping between pages in the GPU's memory and pages in the CPU's memory, removing offsets between pages, allowing CUDA to improve the data bandwidth by transferring the page in a single instruction (as opposed to multiple instructions).



(a) Generic (b) Linear System Solver (c) Form Linear System

Fig. A.23. Graph of the various sub-domain block and thread block combinations used in this GPU implementation. Thin lines denote borders between cell centers and thicker lines denotes sub-domain blocks. Light gray shaded areas mark an example subdomain, with darker gray shared areas marking thread block. Note in (b) thread block and sub-domain block are the same.

size of the solution) has to be transferred between the CPU and the GPU at every time step and iteration step. While the remaining computationally expensive operations (center column of steps in Fig. 1) may have a naive parallelization, for fast computation, effective uses of the various GPU memory types is required. This applies in particular to solving (2.19)–(2.21), on which we focus in the next section.

A.2. GPU kernels

Analogous to loop fusion (removal of loop overheads by combining similar loops), reducing the number of kernel calls (kernel fusion) can improve efficiency, in particular by reducing repeated requests to the same data in global memory, and reducing the use of global memory to store intermediary data. Before discussing the details of the kernels solving (2.19) and (2.20), we first present the remaining GPU computations, which are computed with two kernels: the *preprocessing* kernel, which computes the solution at the ghost points and the nonlinear functions and their derivatives at the cell-center points $(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}})$ (see § 2.3 and 2.4); and the *reduction and update* kernel, which computes the solution criteria and reduces the dataset, and updates the iteration step (2.21).[9] In both kernels, the domain is sub-divided into squares (16×16 in our implementation) and assigned a thread block of the same size, see Fig. A.23(a). Shared memory and register memory are naively used to remove repeated access to global memory.

To solve efficiently the penta-diagonal system defined in (2.19) and (2.20) requires care. Recall that at each time step and each iteration step, there are I penta-diagonal matrices of size $J \times J$ and J penta-diagonal matrices of size $I \times I$ to invert, (2.19) and (2.20), respectively. To show this, we arrange solution values $u_{(i,j)}$ in terms of a matrix, $\mathbf{u}$; and use the subscripts $(,j)$ and $(i,)$ to refer to the jth column of $\mathbf{u}$ (fixed y) and the ith row of $\mathbf{u}$ (fixed x), respectively. Similar notation will be used for $\mathbf{v}$ and $\mathbf{w}$. We may then express (2.19) to (2.21) as

$$\mathbf{A}_j \left[\mathbf{w}_{(,j)}\right]^T = \left[\mathbf{a}^{n+1}_{(k),(,j)} + \mathbf{b}^n_{(,j)}\right]^T \quad 0 \le j < J\,, \tag{A.1}$$

$$\mathbf{B}_i \mathbf{v}_{(i,)} = \mathbf{w}_{(i,)} \qquad 0 \le i < I\,, \tag{A.2}$$

$$\mathbf{u}^{n+1}_{(k+1)} = \mathbf{u}^{n+1}_{(k)} + \mathbf{v}\,, \tag{A.3}$$

where

$$\mathbf{A}_j = \mathbf{I} + \frac{\Delta t}{2}\mathbf{J}_{y_j}\,,\ \ \mathbf{B}_i = \mathbf{I} + \frac{\Delta t}{2}\mathbf{J}_{x_i}\,,\ \ \mathbf{a}^{n+1}_{(k),(,j)} = -\left(\mathbf{I} - \frac{\Delta t}{2}\mathbf{D}_{(,j)}\right)\mathbf{u}^{n+1}_{(k),(,j)}\,,\ \ \mathbf{b}^n_{(,j)} = \left(\mathbf{I} - \frac{\Delta t}{2}\mathbf{D}_{(,j)}\right)\mathbf{u}^n_{(,j)}\,. \tag{A.4}$$

Note that $\mathbf{A}_j$ and $\mathbf{B}_i$ are penta-diagonal matrices representing the implicit y derivatives for a fixed x_i, and the implicit x derivatives for a fixed y_j, respectively.

We therefore solve (A.1) and (A.2) in four kernels: 1) Compute $\mathbf{A}_i$ for each i, $\mathbf{a}^{n+1}_{(k)}$ and $\mathbf{b}^n$; 2) Invert the penta-diagonal system in (A.1) and compute the transpose (coalesce data access); 3) Compute $\mathbf{B}_j$ for each j; and 4) Invert penta-diagonal systems in (A.2) and compute the transpose.

[9] At this point it is unknown if the iteration step satisfies the solution criteria; therefore, this calculation is possibly redundant if the solution is rejected, however, this inefficiency is negligible in comparison to splitting the kernel.

Note 1: $\mathbf{b}^n$ is independent of the iteration step, k, and is therefore only computed at the initial iteration step (for each time step), and stored in memory for use in future iterations. Furthermore, when initializing the iterative method, $\mathbf{u}^{n+1}_{(0)} = \mathbf{u}^n$, we do not explicitly copy $\mathbf{u}^n$ into $\mathbf{u}^{n+1}_{(0)}$, but instead manipulate array pointers to access the correct terms.

Note 2: While there are pre-existing subroutine libraries to perform a matrix transpose, to remove redundant global memory access, the inverse solver is fused with an efficient transpose method.

Note 3: Kernel fusion may be used to combine steps 1) and 3); however, the time saving is negligible compared to the prohibitive total global memory cost, e.g., if M bytes are needed to store the solution, $6M$ bytes are required to store one set of penta-diagonal linear systems, plus an additional M bytes for the fixed implicit term in (A.4), $\mathbf{b}^n$. Our current implementation restricts the total domain size to be able to fit into global memory; therefore, not fusing the kernels allows almost 25% more memory to be utilized (an additional $4M$ bytes are needed for storing the nonlinear functions and their derivatives).[10]

A.2.1. Linear system solver

We begin with the most difficult computation to parallelize, inverting the penta-diagonal linear systems, (A.1) and (A.2). At best, a single penta-diagonal linear system may be solved with linear complexity; however, the scheme is highly non-parallelizable. To parallelize a single penta-diagonal system would require a reduction type method; however, such a method would increase the total number of calculations. To avoid this issue, we instead assign to each thread on a GPU a linear system to solve.

To invert an individual penta-system, a LU factorization[11] is implemented e.g., $\mathbf{A}_i = \mathbf{L}_i\mathbf{U}_i$. In general, solving the linear system $\mathbf{L}_i\mathbf{U}_i\mathbf{a}_i = \mathbf{c}_i$ may be separated into two operations,

$$\mathbf{U}_i\mathbf{a}_i = \mathbf{b}_i \ , \text{ and } \mathbf{L}_i\mathbf{b}_i = \mathbf{c}_i \ , \tag{A.5}$$

which may be inverted with linear complexity. To sub-divide the (x, y) domain for parallelization, the domain is divided according to the spatial direction, i.e., the domain is divided into strips. The direction that is sub-divided depends on the current direction of the ADI method being computed, e.g., solving the implicit y derivatives for fixed x, the domain is divided in the x direction. The width of the strips (16 in our implementation) is the number of penta-diagonal systems in each sub-domain (strip) e.g., different i in (A.5). Each sub-domain is assigned a vector of threads (equal to the width of the strips) which invert the subset of penta-diagonal systems by LU factorization. Fig. A.23(b) gives an example of a 16×16 domain with strips of length 4. Since each penta-diagonal system is independent (for a given ADI direction), each step of the LU factorization is essentially a three point iterative method, therefore, register memory is used to store the previous two iterative points.

As mentioned previously, to utilize memory coalescence (threads access data points that are adjacent to each other in global memory), data must be transposed in memory. To fuse the transpose with LU factorization, the transpose is computed while inverting the second step of the LU factorization, e.g., inverting $\mathbf{U}_i a_i = b_i$ in (A.5). Since coalesced memory access is not relevant for shared memory access, it may be used to temporarily store data. For example, in the y implicit direction of the ADI method, 16 threads (each x value) compute an iteration step (fixed y value) of the U factorization, and then store the data row-wise (fixed y) in a square block of shared memory, the size of which depends on the strip width (16×16 in our implementation). Repeating the U factorization for the next 15 iteration steps, the solution is computed on a 16×16 sub-block. The transpose may be performed in a block-wise fashion by accessing the shared memory column-wise (fixed x) without any significant inefficiency, and then stored in global memory with coalesced access.

A.2.2. Computing the linear system

We now switch focus to the remaining two kernels not yet defined, computing $\mathbf{A}_j$, $\mathbf{a}^{n+1}_{(k),(,j)}$, $\mathbf{b}^n_{(,j)}$, and $\mathbf{B}_i$ in (A.4). It is mostly trivial to parallelize the computation of $\mathbf{A}_j$, $\mathbf{B}_i$, $\mathbf{a}^{n+1}_{(k),(,j)}$ and $\mathbf{b}^n_{(,j)}$, as at a given time step, threads do not depend on the data computed by other threads; however, to compute numerical derivatives, adjacent points are required. It is therefore prudent to use shared memory to reduce redundant global memory access.

To compute $\mathbf{B}_i$ in (A.4), the domain is divided into squares and assigned a thread block of equal size (16×16 in our implementation). While we may define a shared memory block of equal size to the sub-domain, threads near the x boundaries of the sub-domain depend on data in the adjacent sub-domain; therefore analogous to the idea of ghost points, we padded that shared memory block with ghost points (corresponding to solution values in adjacent sub-domains). In our implementation, adding ghost points (2 points at each boundary) results in a 20×16 block of shared memory. Note that data must be in the transpose form; however, since shared memory is already being used and a square thread block is operating on the sub-domain, the transpose is performed block-wise when transferring the data to global memory.

[10] Our implementation uses synchronous kernel calls i.e. kernels are executed in serial by the CPU. By sub-dividing the domain into partitions (which would be further sub-divided by the kernels), clever use of asynchronous kernel calls may be used to reduce the total memory requirement, e.g., if each partition contains $M_1 \ll M$ bytes of the solution, then $5M$ plus some small multiple of $10M_1$ bytes would be required, i.e., the total memory usage may be halved (approximately).

[11] While Gaussian elimination may be used, it requires more divisions to be computed, which, on a GPU, is more expensive to evaluate when compared to a multiplication. On a CPU, typically, there is no significant difference.

While we may apply the same procedure for computing $\mathbf{A}_j$, $\mathbf{a}^{n+1}_{(k),(,j)}$, and $\mathbf{b}^{n}_{(,j)}$ in (A.4), unlike computing $\mathbf{B}_i$, threads near either the x or y boundaries of the sub-domain depend on data in the adjacent sub-domain; therefore, the shared memory bank is padded with ghost points in both directions (20×20 in our implementation). For the $\mathbf{B}_j$ shared memory block, ghost points could be transferred from global memory to shared memory with little inefficiency; however, for the $\mathbf{A}_i$, $\mathbf{a}^{n+1}_{(k),(,j)}$, and $\mathbf{b}^{n}_{(,j)}$ shared memory blocks, memory transfer is inefficient with the additional ghost points. A faster approach is to subdivide the domain into sub-rectangles and assign each sub-domain a vector of threads, e.g., in our implementation the sub-domain size is 32×15, where the 32 threads are mapped to 32 x-values, and each thread loops through 15 y-values, see Fig. 2. Furthermore, we note that the y derivatives depend on at most five points ($j-2,\ j-1,\ j,\ j+1$ and $j+2$); therefore, instead of a $(32+4)\times(15+4)$ (sub-domain plus ghost points) shared memory block, we implement five 36×1 shared memory blocks. Similarly, function values and their derivatives require three points, therefore six (three each) 36×1[12] shared memory blocks are used for each nonlinear function (there are two in our model). While the initial iteration (first y-value on sub-domain) requires transferring seventeen 36×1 blocks of data from global memory to shared memory, in subsequent iterations, only five 36×1 blocks of data are transferred from the global memory to shared memory: the solution, the two non-linear functions and their derivatives.

Note: Numerical tests show that the choice of 15 in the sub-block size for $\mathbf{A}_i$, $\mathbf{a}^{n+1}_{(k),(,j)}$, and $\mathbf{b}^{n}_{(,j)}$ leads to the fastest computation time (seemingly independent of the thread block size), which is related to our implementation. To remove explicitly copying data between shared memory location, computations for each looped y value are grouped into a subroutine, and at each loop step, the input parameters are alternated by pointer manipulation. For example, for a generic three point method, first define $v_{-1}=u_{j-1}$, $v_0=u_j$, and $v_1=u_{j+1}$, where u is the global memory variable and v is the shared memory variable. Performing the following operations at the next three iteration steps (three cycle):

1. Compute $f(v_{-1},v_0,v_1)$, and then set $v_{-1}=u_{j+2}$;
2. Compute $f(v_0,v_1,v_{-1})$, and then set $v_0=u_{j+3}$;
3. Compute $f(v_1,v_{-1},v_0)$, and then set $v_1=u_{j+4}$;

it may be seen that by the third step, $v_{-1}=u_{j-1+3}$, $v_0=u_{j+3}$, and $v_1=u_{j+1+3}$, i.e., the definitions of the shared memory variables are realigned but offset by 3. Similarly, a five point method results in a five cycle. Recalling the combined stencil in Fig. 2, the method is a three-point method with respect to function values, and a five-point method with respect to solution values; therefore, combining the 3-cycle and 5-cycle, results in a 15-cycle, the most efficient choice for the y domain size according to numerical tests.

Appendix B. Radial Fourier transform

Here we give an overview of the procedure for computing the radial Fourier Transform used to analyze simulations in § 4.3. To map the magnitude of the 2D Fourier transform to the radial Fourier Transform, a one dimensional (1D) function of the magnitude of the wave vector $q_r=\sqrt{q_x^2+q_y^2}$, six processing steps are performed. The first four steps smooth the magnitude of the Fourier transform and map the smoothed data from a 2D Cartesian definition of the wavenumber to a 1D function of the magnitude of the wave number. The remaining two steps further smooth the data so as to reliably extract the local maxima.

First, the peak at the zero wave number is removed by subtracting the initial film thickness H_0, i.e., the magnitude of the Fourier transform of $\zeta(x,y)=h(x,y)-H_0$ is computed, an example of which is shown in Fig. B.24(a). Second, to reduce noise and smooth the data, the magnitude of the Fourier transform data is convoluted with a Gaussian function/filter. Third, the smoothed data, $\bar{\zeta}$, are mapped from the Cartesian coordinates (q_{x_i},q_{y_j}) to polar coordinates (q_{r_i},q_{θ_j}). Fourth, the polar coordinate data are mapped to a (radial) function, $\bar{\bar{\zeta}}(q_{r_i})$, by averaging over the θ direction for a fixed q_{r_i}, i.e., we compute

$$\bar{\bar{\zeta}}(q_{r_k})=\frac{1}{N_k}\sum_{(i,j)\in P_k}\bar{\zeta}(q_{r_i},q_{\theta_i})\,,\quad P_k=\{q_{r_i},q_{\theta_i}\in\mathbb{R}|q_{r_i}=q_{r_k}\},\tag{B.1}$$

where N_k is the number of points in the set P_k. An example of (B.1) is given by the yellow scatter plot in Fig. B.24(c) with Fourier transform data in Fig. B.24(a). To extract a local maximum (dominant wavenumber) numerically, sufficiently smooth data are required. To smooth the scattered data two final processing steps are performed: first, using a linear-least squares method, the scattered data are fitted to a piece-wise continuous function of the form

$$\bar{\bar{\zeta}}(q)=a_k+b_k\frac{s-q_k}{q_{k+1}-q_k}\,,\quad\text{where}\quad s\in[0,1]\,,\quad q=q_k+s(q_{k+1}-q_k)\,,\tag{B.2}$$

[12] While only a 34×1 shared memory block is required to store function values plus ghost points, a 36×1 domain is chosen to align shared memory indices to those of the shared memory used to store the solution, simplifying coding.

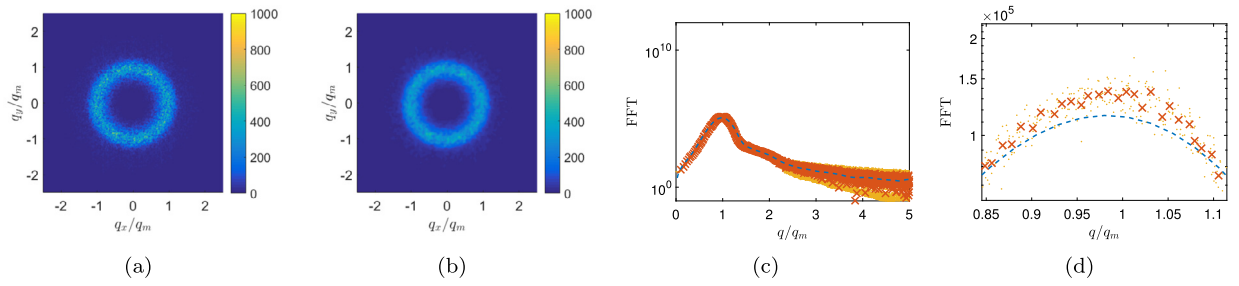


Fig. B.24. (a) An example of the log of the magnitude of the two-dimensional Fourier transform. (b) An example of smoothing the data in (a) using a convolution with a Gaussian filter. (c) A plot of: i) the data in (b) mapped to a radial one-dimensional function (yellow scatter plot) using equation (B.1); ii) a linear least-squares fit of the scatter plot data to the piece-wise linear function (red '×' symbols); and iii) the piece-wise linear function smoothed with a moving average filter (dashed blue curve). (d) Zoom of panel (c) demonstrating the improved smoothness using the moving average filter.

and the domain is restricted to $q \in [0, 5q_m]$ (reducing computational cost). Note that we choose q_k's such that each interval contains 10 points. Denoting the number of intervals as K, continuity implies $b_k = a_{k+1} - a_k$ for $0 \le k < K-1$, and choosing $a_0 = 0$ (i.e. $\zeta(q=0)=0$) leaves K free parameters: b_{K-1} and a_k for $1 \le k \le K-1$, therefore the fitting procedure is fully determined.

An example of this fitting procedure is given by the red '×' scatter plot in Fig. B.24(c). This step accurately captures the trend of the scattered data; however, we could not produce a sufficiently smooth function, see Fig. B.24(d) for a comparison between the piece-wise continuous function (red '×' symbols), and the additional smoothing step (dashed blue curve) discussed next. To smooth the piece-wise continuous function, the next step evaluates (B.2) on an equipartitioned grid of 400 points and smooths the data using a moving average filter. Specifically, we compute

$$\tilde{\tilde{\zeta}}(q_k) = \frac{1}{2P+1} \sum_{p=-P}^{P} \tilde{\tilde{\zeta}}(q_{k+p}) \tag{B.3}$$

where P is half the filter width, and the filter is applied three times with $P = 3, 2$, and then 1.

To numerically extract the local maxima, a second order method is used to compute the derivative of the radial Fourier transform, and a combination of a bisection method and interpolation is used to find the roots of the derivative (critical points). To extract the local maxima, the first derivative test is used where the second derivative is computed numerically at the roots.

Appendix C. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jcpx.2018.100001.

References

[1] J. Jacqmin, Calculation of two-phase Navier–Stokes flows using phase field modeling, J. Comput. Phys. 155 (1999) 96.
[2] J. Jacqmin, Contact-line dynamics of a diffuse fluid interface, J. Fluid Mech. 402 (2000) 57.
[3] K. Mahady, S. Afkhami, L. Kondic, A volume-of-fluid method for simulating fluid/fluid interfaces in contact with solid boundaries, J. Comput. Phys. 294 (2015) 243.
[4] M. Renardy, Y. Renardy, J. Li, Numerical simulation of moving contact line problems using a volume-of-fluid method, J. Comput. Phys. 171 (2001) 243.
[5] I. Seric, S. Afkhami, L. Kondic, Direct numerical simulation of variable surface tension flows using a volume-of-fluid method, J. Comput. Phys. 352 (2018) 615.
[6] P. Yue, C. Zhou, J. Feng, Sharp-interface limit of the Cahn–Hilliard model for moving contact lines, J. Fluid Mech. 645 (2010) 279.
[7] C. Zhou, P. Yue, J.J. Feng, C.F. Ollivier-Gooch, H.H. Hu, 3D phase-field simulations of interfacial dynamics in Newtonian and viscoelastic fluids, J. Comput. Phys. 229 (2010) 498.
[8] J. Becker, G. Grün, R. Seemann, H. Mantz, K. Jacobs, K.R. Mecke, R. Blossey, Complex dewetting scenarios captured by thin-film models, Nat. Mater. 2 (2003) 59.
[9] J.A. Diez, L. Kondic, Computing three-dimensional thin film flows including contact lines, J. Comput. Phys. 183 (2002) 274.
[10] R. Fetzer, K. Jacobs, A. Münch, B. Wagner, T.P. Witelski, New slip regimes and the shape of dewetting thin liquid films, Phys. Rev. Lett. 95 (2005) 127801.
[11] T.-S. Lin, L. Kondic, A. Filippov, Thin films flowing down inverted substrates: three dimensional flow, Phys. Fluids 24 (2012) 022105.
[12] A. Sharma, Many paths to dewetting in thin films: anatomy and physiology of surface instability, Eur. Phys. J. E 12 (2003) 397.
[13] U. Thiele, L. Brusch, M. Bestehorn, M. Bär, Modelling thin-film dewetting on structured substrates and templates: bifurcation analysis and numerical simulations, Eur. Phys. J. E 11 (2003) 255.
[14] S. Engelnkemper, M. Wilczek, S.V. Gurevich, U. Thiele, Morphological transitions of sliding drops – dynamics and bifurcations, Phys. Rev. Fluids 1 (2016) 073901.
[15] T. Witelski, M. Bowen, ADI schemes for higher-order nonlinear diffusion equations, Appl. Numer. Math. 45 (2003) 331.
[16] L. Kondic, J.A. Diez, P.D. Rack, Y. Guan, J. Fowlkes, Nanoparticle assembly via the dewetting of patterned thin metal lines: understanding the instability mechanism, Phys. Rev. E 79 (2009) 026302.
[17] N. Murisic, L. Kondic, On evaporation of sessile drops with moving contact lines, J. Fluid Mech. 679 (2011) 219.
[18] K. Jacobs, R. Seemann, S. Herminghaus, Stability and Dewetting of Thin Liquid Films, World Scientific, New Jersey, 2008, p. 243.

[19] R. Seemann, S. Herminghaus, K. Jacobs, Dewetting patterns and molecular forces: a reconciliation, Phys. Rev. Lett. 86 (2001) 5534.
[20] S. Herminghaus, K. Jacobs, K. Mecke, J. Bischof, A. Fery, M. Ibn-Elhaj, S. Schlagowski, Spinodal dewetting in liquid crystal and liquid metal films, Science 282 (1998) 916.
[21] C. Poulard, A.M. Cazabat, Spontaneous spreading of nematic liquid crystals, Langmuir 21 (2005) 6270.
[22] S. Schlagowski, K. Jacobs, S. Herminghaus, Nucleation-induced undulative instability in thin films of nCB liquid crystals, Europhys. Lett. 57 (2002) 519.
[23] J.D. Fowlkes, L. Kondic, J.A. Diez, P.D. Rack, Self-assembly versus directed assembly of nanoparticles via pulsed laser induced dewetting of patterned metal films, Nano Lett. 11 (2011) 2478.
[24] A. González, J.A. Diez, Y. Wu, J.D. Fowlkes, P.D. Rack, L. Kondic, Instability of liquid Cu films on a SiO_2 substrate, Langmuir 29 (30) (2013) 9378.
[25] R. Craster, O. Matar, Dynamics and stability of thin liquid films, Rev. Mod. Phys. 81 (2009) 1131–1198.
[26] A. Oron, S.H. Davis, S.G. Bankoff, Long-scale evolution of thin liquid films, Rev. Mod. Phys. 69 (1997) 931.
[27] H. Atwater, A. Polman, Plasmonics for improved photovoltaic devices, Nat. Mater. 9 (2010) 9.
[28] M.-A.Y.-H. Lam, L. Cummings, L. Kondic, Stability of thin fluid films characterised by a complex form of effective disjoining pressure, J. Fluid Mech. 841 (2018) 925.
[29] I. Seric, S. Afkhami, L. Kondic, Interfacial instability of thin ferrofluid films under a magnetic field, J. Fluid Mech. Rapids 755 (2014) R1.
[30] T.-S. Lin, L. Kondic, U. Thiele, L.J. Cummings, Modeling spreading dynamics of liquid crystals in three spatial dimensions, J. Fluid Mech. 729 (2013) 214.
[31] A.M. Cazabat, U. Delabre, C. Richard, Y.Y.C. Sang, Experimental study of hybrid nematic wetting films, Adv. Colloid Interface Sci. 168 (2011) 29.
[32] R. Seemann, S. Herminghaus, K. Jacobs, Gaining control of pattern formation of dewetting liquid films, J. Phys. Condens. Matter 13 (2001) 4925.
[33] J.E. Dennis, R.B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, SIAM edition, Society for Industrial and Applied Mathematics, Philadelphia, 1996.
[34] F. Vandenbrouck, M.P. Valignat, A.M. Cazabat, Thin nematic films: metastability and spinodal dewetting, Phys. Rev. Lett. 82 (1999) 2693.
[35] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd ed., Cambridge University Press, New York, NY, USA, 1992.
[36] V. Ajaev, G. Homsy, Modeling shapes and dynamics of confined bubbles, Annu. Rev. Fluid Mech. 38 (2006) 277.
[37] L.J. Cummings, Evolution of a thin film of nematic liquid crystal with anisotropic surface energy, Eur. J. Appl. Math. 15 (2004) 651.
[38] C. Neto, K. Jacobs, R. Seemann, R. Blossey, J. Becker, G. Grün, Satellite hole formation during dewetting: experiment and simulation, J. Phys. Condens. Matter 15 (2003) 3355.
[39] A. Pototsky, M. Bestehorn, D. Merkt, U. Thiele, Morphology changes in the evolution of liquid two-layer films, J. Chem. Phys. 122 (2005) 224711.
[40] U. Thiele, Open questions and promising new fields in dewetting, Eur. Phys. J. E 12 (2003) 409.
[41] U. Thiele, M.G. Velarde, K. Neuffer, Dewetting: film rupture by nucleation in the spinodal regime, Phys. Rev. Lett. 87 (2001) 016104.
[42] K. Mahady, S. Afkhami, L. Kondic, A numerical approach for the direct computation of flows including fluid–solid interaction: modeling contact angle, film rupture, and dewetting, Phys. Fluids 062002 (2016) 28.
[43] A.G. Gonzalez, J.D. Diez, L. Kondic, Stability of a liquid ring on a substrate, J. Fluid Mech. 718 (2013) 213.
[44] W. van Saarloos, Front propagation into unstable states, Phys. Rep. 386 (2003) 29.
[45] M.A. Lam, L.J. Cummings, T.-S. Lin, L. Kondic, Modeling flow of nematic liquid crystal down an incline, J. Eng. Math. 94 (2014) 97.
[46] M.A. Lam, L.J. Cummings, T.-S. Lin, L. Kondic, Three-dimensional coating flow of nematic liquid crystal on an inclined substrate, Eur. J. Appl. Math. 25 (2015) 647.
[47] T.-S. Lin, L. Kondic, Thin films flowing down inverted substrates: two dimensional flow, Phys. Fluids 22 (2010) 052105.
[48] L.C. Mayo, S.W. McCue, T.J. Moroney, Gravity-driven fingering simulations for a thin liquid film flowing down the outside of a vertical cylinder, Phys. Rev. E 87 (2013) 053018.
[49] J. Diez, L. Kondic, On the breakup of fluid films of finite and infinite extent, Phys. Fluids 19 (2007) 072107.
[50] E. Polizzi, A.H. Sameh, A parallel hybrid banded system solver: the spike algorithm, Parallel Comput. 32 (2006) 177.
[51] D. Heller, Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems, SIAM J. Numer. Anal. 13 (1975) 484–496.
[52] U. Meier, A parallel partition method for solving banded systems of linear equations, Parallel Comput. 2 (1985) 33.
[53] M.J. Flynn, Some computer organizations and their effectiveness, IEEE Trans. Comput. C-21 (9) (1972) 948.