Multigrid solution methods for nonlinear time-dependent systems

by

Feng Wei Yang

Submitted in accordance with the requirements for the degree of Doctor of Philosophy.

UNIVERSITY OF LEEDS

The University of Leeds
School of Computing

September 2014

The candidate confirms that the work submitted is his own, except where work which has formed part of jointly authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

Acknowledgements

I would like to express my deepest gratitude to my supervisors Peter Jimack and Matthew Hubbard. They have not only provided me the most incredible teaching, support, feedback and encouragement, but also shown me what a great researcher should be from their every word and action. I am grateful that they have indulged me with my habit in details. Whilst the work of mine was not always straightforward, I have enjoyed the past four years nevertheless. I would also like to thank Mark Walkley for giving up his time and expertise to help; Chris Goodyer for answering many puzzling programming-related questions, and his generosity for sharing his knowledge of Campfire and High Performance Computing; Peter Bollada for letting me study his research and eventually helping me to contribute to it.

I would like to thank my father Ru Jin Yang for his love and support that has guided me for so many years, without him this would have not been possible, and my mother Xue Mei Wen for her endless love. I am grateful to my wife Yi Yun Zhang who has always been encouraging, understanding and caring. I am indebted to my big family for all their help and support. I know all my relatives who are not familiar with my field of research will read this thesis while being proud of whom I have become, my sincere gratitude for their unconditional love that kept me going.

I would like to thank Aryana Tavanai, who is like a brother to me, and yes we have made it through since the foundation course eight years ago, Keeran Brabazon who shared tea and muffins with me, Thomas Ranner who was always patient to answer my many daft questions, Dave Harrison who kept me company in the pub, Christiana Panayi who brought me delicious cakes.

In the end, my special thanks go to people in the School of Computing, I have always been proud of being one of its members for the past eight years.

Abstract

An efficient, accurate and reliable numerical solver is essential for solving complex mathematical models and obtaining their computational approximations. The solver presented in this work is built upon nonlinear multigrid with the full approximation scheme (FAS). Its implementation is achieved, in part, using a complex, open source software library PARAMESH, and the resulting numerical solver, Campfire, also combines with adaptive mesh refinement, adaptive time stepping and parallelization through domain decomposition.

There are five mathematical models considered in this work, ranging from applications such as binary alloy solidification and fluid dynamics to a multi-phase-field model of tumour growth. These mathematical models consist of nonlinear, time-dependent and coupled partial differential equations (PDEs). Using our adaptive, parallel multigrid solver, together with finite difference method (FDM) and backward differentiation formulas (BDF), we are able to solve all five models in computationally demanding 2-D and/or 3-D situations.

Due to the choice of second order central finite difference and second order BDF2 method, we obtain, and demonstrate, solutions with an overall second order convergence rate and optimal multigrid convergence. In the case of the multi-phase-field model of tumour growth, this has not previously been achieved. The novelties of our work also include solving the model of binary alloy solidification with a time-dependent temperature field in 3-D for the first time; implementing non-time-dependent equations alongside the coupled time-dependent partial differential equations (and increasing the range of boundary conditions) to significantly increase the generality and range of applicability of the described multigrid solver; improving the efficiency of the implementation of the solver through multiple developments; and introducing penalty terms to smoothly control the behaviour of phase variables where their range of valid values is constrained.

To my parents, Mr. Ru Jin Yang, Mrs. Xue Mei Wen,
and my wife Mrs. Yi Yun Zhang,
who made everything possible for me...

Declarations

Some parts of the work presented in this thesis have been included in the following articles which are submitted for publication,

P. Bollada, C. Goodyer, P. Jimack, A. Mullis and F. Yang., "Thermal-solute phase field three dimensional simulation of binary alloy solidification.", *Journal of Computational Physics*, submitted August 2014.

In detail, the work from this jointly authored publication (submitted August 2014) is presented in Chapter 4. The contribution F. Yang made in this publication is explained in Sections 4.2 and 4.3 in detail. While P. Bollada is mainly credited for solving the mathematical model presented in this publication, P. Jimack and A. Mullis provided necessary guidance, as well as many insight into the model itself. C. Goodyer implemented the software used for this work.

Some parts of the work described in this thesis have also been presented in the UK & Republic of Ireland SIAM section, 25th Biennial Numerical Analysis Meeting Strathclyde 2013, and was awarded a Certificate of Recognition for an Outstanding Talk for the presentation "Towards the development and application of an optimal solver for continuum models of tumour growth".

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

There are five different mathematical models considered in this thesis, ranging from models of thin film flows, binary alloy solidification and Cahn-Hilliard-Hele-Shaw system of equations to multi-phase-field model of tumour growth. To respect the general convention used for notation, in this thesis, we include a local glossary for each model, and they are given in Chapter 1 when each of models are introduced, as well as in subsequent chapters when they are described in detail.

Here we summarise common notation used in this thesis. It is worth noting that a few of them are only relevant to short sections of text, therefore limited repetitions will occur, on the other hand, specific definitions of all notations used are denoted as clear as possible in the text.

A, C, M	Matrices
a	Array contains elements in the matrices
b	Right-hand side vector
E	Exact error or parallel efficiency
e	Approximate error
$f, \mathscr{F}, g$	Known functions for the purpose of demonstration
h	Distance between two adjacent grid points, and this notation is used for the variable of thin film thickness
I_1^2	Grid transfer operator which operates from grid 2 to grid 1
i, j, k	Indices of Cartesian grid points
$\mathscr{K}$	Number of dependent variables in a system
$\mathscr{L}$	Grid levels
l	Number of iterations
N	Total number of grid points in one direction of axes
n	Total number of internal grid points in one direction of axes, or the number of unknowns
p, u	General variables for the purpose of demonstration, and u is used to denote the true solution to a system
r	Residual
r^{BDF}	Ratio in the adaptive BDF method
S	Parallel speed up
T	End time
t	Time
v	Approximate solution to a system

vec, w	Vectors for the purpose of demonstration
x, y, z	Cartesian coordinates
Z_h	A set of grid points

Greek letters

$\triangle$	Laplace operator
Ω	Computational domain
$\partial\Omega$	Boundary of the domain
$\overline{\Omega}$	All grid points on the domain
ν	The outward-pointing normal to the boundary $\partial\Omega$
$\nabla\cdot$	Divergence operator
∇	gradient operator
δt	Time step size
ω	Weighted parameter
κ	Condition number of a matrix

Abbreviations

AMR	Adaptive mesh refinement
BDF	Backward differentiation formulas
CG	Conjugate gradient
CHHS	Cahn-Hilliard-Hele-Shaw
CPU	Central processing unit
DG	Discontinuous Galerkin
FAS	Full approximation storage
FDM	Finite difference method
FEM	Finite element method
FVM	Finite volume method
GCU	Guard cell update
GMRES	Generalized minimum residual method
GPU	Graphic processing unit
MIMD	Multiple instruction stream-multiple data stream
MINRES	Minimal residual method
MLAT	Multi-level adaptive technique
MPI	Message passing interface
ODE	Ordinary differential equation

PDE	partial differential equation
RHS	Right-hand side
SIMD	Single instruction stream-multiple data stream
SOR	Successive over-relaxation

Chapter 1

General Introduction

Recent scientific research has led to the use of complex mathematical models and their computational approximations. These models often consist of highly nonlinear, time-dependent and coupled partial differential equations (PDEs). Accurate, efficient and reliable numerical algorithms (and, frequently, great computational power) are necessary in order to obtain robust computational solutions.

This thesis is concerned with the novel application of advanced numerical methods to the efficient solution of nonlinear time-dependent systems of PDEs. Specifically, the focus is on parabolic systems. In this chapter these systems are introduced and described, including the introduction of some specific examples. In subsequent chapters, the numerical methods and the software that we have exploited and developed are discussed.

1.1 Background

In this section, a simple linear time-dependent parabolic system is firstly described, with subsequent generalizations to illustrate the introductions of nonlinearity. Following this, an example system of nonlinear PDEs is derived from the Navier-Stokes equations by using lubrication theory. Two of its variations are then described: the first one describes fully-developed flows over well-defined topography; the second one describes droplet spreading with a precursor film. The remainder of the section is used to introduce some phase-field models and some of their applications. One example model, for rapid solidi-

fication, is presented in detail. The second example model is a two-phase Cahn-Hilliard-Hele-Shaw system of equations that is used to study the mixing of fluids. The final example model is a multi-phase system for simulating tumour growth.

An example of a single parabolic PDE is the diffusion equation [188], that may take the following form:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}. \tag{1.1}$$

Here $u(x,y,z,t)$ has four independent variables. Physically u may represent a temperature field, whilst t represents time and (x,y,z) are Cartesian spatial coordinates. This equation is often written using the Laplace operator $\triangle$ for simplicity:

$$\frac{\partial u}{\partial t} = \triangle u, \tag{1.2}$$

where in 2-D

$$\triangle = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}, \tag{1.3}$$

or in 3-D

$$\triangle = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}. \tag{1.4}$$

A closed region Ω is defined to be the spatial domain, whereas $[0,T]$ is defined to be the temporal region of interest. The full domain for this PDE is $\Omega \times [0,T]$. For completeness, values on both the boundary of the spatial domain and the initial state of all spatial points need further specification. Two broad classes of boundary conditions are considered for models in this research: Dirichlet and Neumann. An example of the former is the following:

$$u = g \text{ on } \partial\Omega \times (0,T], \tag{1.5}$$

where g is a given function on the boundary $\partial\Omega$ of the spatial region Ω. A Neumann boundary condition is

$$\frac{\partial u}{\partial \nu} = f \text{ on } \partial\Omega \times (0,T], \tag{1.6}$$

where f is a given function and ν denotes the outward-pointing normal to the boundary $\partial\Omega$. There are models that require the Dirichlet boundary condition on part of the domain and Neumann boundary condition for the rest.

For the boundary condition that is required on the boundary of the temporal region of interest, this is typically presented for all of Ω at $t = 0$, and is known as an initial condition. For example, $u(x,y,z,t=0) = u_0(x,y,z)$.

The PDE described in Equation (1.2) is linear since the terms involving u and its

derivatives are all linear. There are several ways for nonlinearity to appear. One form involves adding a source term $f(u)$, which is a nonlinear function that depends upon u. Thus Equation (1.2) turns into the following:

$$\frac{\partial u}{\partial t} = \triangle u + f(u), \tag{1.7}$$

This nonlinearity can also exist within the following form:

$$\frac{\partial u}{\partial t} = \nabla \cdot (f(u)\nabla u), \tag{1.8}$$

where $f(u)$ is again a nonlinear function, $\nabla\cdot$ is the divergence operator and ∇ is the gradient operator, together $\nabla \cdot \nabla = \triangle$.

Equations (1.2), (1.7) and (1.8) that have been considered so far, are known as parabolic PDEs. There are two other types of PDEs: elliptic and hyperbolic. An elliptic PDE is non-time-dependent, and a hyperbolic PDE often involves oscillations. There are no hyperbolic PDE considered in this thesis. The interested reader is directed to [65, 188, 205] for more detailed description of PDEs.

Having described single nonlinear parabolic equations, an example of a system of coupled partial differential equations (system of PDEs) is introduced. At least some of the dependent variables in such a system will appear in more than one equation. Here we present an example of a fourth-order system, which takes the form of two coupled linear second-order equations:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \triangle p, \\ p &= -\triangle u. \end{aligned} \tag{1.9}$$

Here $u(x,y,z,t)$ and $p(x,y,z,t)$ are the two dependent variables. To complete this system, a spatial domain Ω and its boundary $\partial\Omega$ are defined, a boundary condition for each dependent variable needs to be imposed. For example, $u(x,y,z) = p(x,y,z) = 0$ on $\partial\Omega$ is the Dirichlet boundary condition, and $\frac{\partial u}{\partial \nu} = \frac{\partial p}{\partial \nu} = 0$ on $\partial\Omega$ is the Neumann boundary condition, where ν denotes the outward-pointing normal to the boundary $\partial\Omega$. Combinations of these are possible.

Initial conditions $u(x,y,z,t=0)$ and $p(x,y,z,t=0)$ are also required. In practice, after specifying the initial condition $u(x,y,z,t=0)$, it is then possible to obtain $p(x,y,z,t=0)$ using the second equation from the system of coupled Equations (1.9). Situations like this are often encountered in this thesis, and details with each specific problem are discussed

in later chapters.

For completeness, an equivalent form of the system of coupled Equations (1.9) is also presented, written as a single fourth-order diffusion PDE:

$$\frac{\partial u}{\partial t} = \triangle\left(\triangle u\right). \tag{1.10}$$

Clearly, Equations (1.9) and (1.10) are representations of a linear system. To introduce nonlinearity, we present the following fourth-order nonlinear coupled system, in the form of two second-order equations:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \underline{\nabla} \cdot (u^2 \underline{\nabla} p), \\ p &= -\triangle u. \end{aligned} \tag{1.11}$$

For this problem, Dirichlet and/or Neumann boundary conditions can be applied. Initial conditions for u and p must also be specified, for example, $u(x,y,z,t=0) = u_0(x,y,z)$ and $p(x,y,z,t=0) = p_0(x,y,z)$.

Having described the general structure of parabolic PDEs, and systems of coupled equations, before the actual mathematical models are introduced, in the following section, the methodology followed in this thesis is explained.

1.2 Methodology

These mathematical models introduced in the current chapter are selected for the purpose of achieving an incremental research methodology. The first model (model of droplet spreading) studied is selected because of its simplicity. Although it is a simple model, to accurately and efficiently solve it, many additional techniques are required, such as spatial adaptivity and adaptive time-stepping. Since it has been well studied, we can compare our solutions against those presented in literature.

The second model (model of fully-developed flows) can be viewed as an variation to the droplet model. The comparison between this model and the famous Navier-Stokes equations is still an ongoing research project.

The third model (model of binary alloy solidification) is included because many novel implementations and improvements to our software necessarily aided the work of Bollada in [27].

The fourth model (CHHS system of equations) is selected as a stepping stone towards the tumour model of Wise et al. This methodology is used by Wise et al., and it is our

intention to follow their foot steps.

The fifth model (model of tumour growth) is the main goal of this research.

1.3 Mathematical Models

In this section, five mathematical models that are considered in this thesis are introduced. Due to conventional notation used in the literature, we prefer to keep the same notation as much as possible for each of the problem, therefore, several local glossaries are used here to clearly indicate the notation.

1.3.1 Thin Film Flows

The flow of thin liquid films on surfaces is of both interest and importance in fields such as biophysics, physics and engineering. In the review paper [174], various mathematical models have been described, which concern different aspects of the physics of thin liquid films. In the context of this research, the dynamics of the contact lines between a thin film of liquid and a substrate are described, as well as fully-developed thin film flows over a prescribed topography. It is widely known that the Navier-Stokes equations provide a good representation of fluid dynamics in various situations. The models that we are interested in are derived from lubrication theory [174]. It provides justification for approximations that reduce the Navier-Stokes equations to a simpler system of PDEs [63]. Such a system provides an accurate description of these thin film flows [58], as long as the ratio between characteristic flow thickness and extent of the substrate is small [174]. The basic idea is to assume that the variation of the flow in the direction perpendicular to the film is negligible and therefore averaging may be undertaken in this direction. It is further assumed that this dimensional length is very small compared to the tangential length scales and so higher order terms in this ratio may be neglected. Consequently, the resulting dependent variables are functions of just two spatial dimensions, x and y.

A mathematical model of fully-developed flows over topography is described in the following subsection. A droplet spreading model is discussed afterwards. A local glossary is given below for the models of thin film flows. It is worth noting that since the notation h is occupied, when comes to described the distance between two adjacent grid points, we use dx, dy for x and y directions, respectively.

Glossary

Symbol	Description

H, h	Thin film thickness
h^*	Precursor film thickness
S, s	Topography
X, Y, Z, x, y, z	Cartesian coordinates
T, T_o	Time
U, V, W, u, v, w	Velocity components
ρ	Density
P, p	Pressure
g	Gravitational acceleration
μ	Viscosity
σ	Surface tension
α	Inclined angle of the substrate to the horizontal
Ca	Capillary number
N	Relative importance of the normal component of gravity
n, m	Exponents in the disjoining pressure term
Φ_e	Equilibrium contact angle
$V_x(Z)$	Parabolic velocity inside of the droplet
$\Pi(h)$	Disjoining pressure term
B_o	Bond number
H_o	Characteristic thin film flow/droplet thickness
L_o	Characteristic substrate length
ε	Ratio between H_o and L_o, and is assumed to be sufficient small by lubrication theory
S_0, s_0	Topography height/depth
γ	Steepness of the topography
l_t	Streamwise topography extent
w_t	Spanwise topography extent
A	Aspect ratio of the topography
dx, dy	distance between two adjacent grid points in the x and the y directions

1.3.1.1 Fully-Developed Flows

For fully-developed flows, it is assumed that fluid has already covered the domain in its initial condition. For the purpose of demonstration, Figure 1.1 is a sketch of the flow $H(X,Y,T)$, over an inclined substrate with a topography $S(X,Y)$, which is inclined at an angle α to the horizontal. The fluid is assumed to be Newtonian and incompressible, of

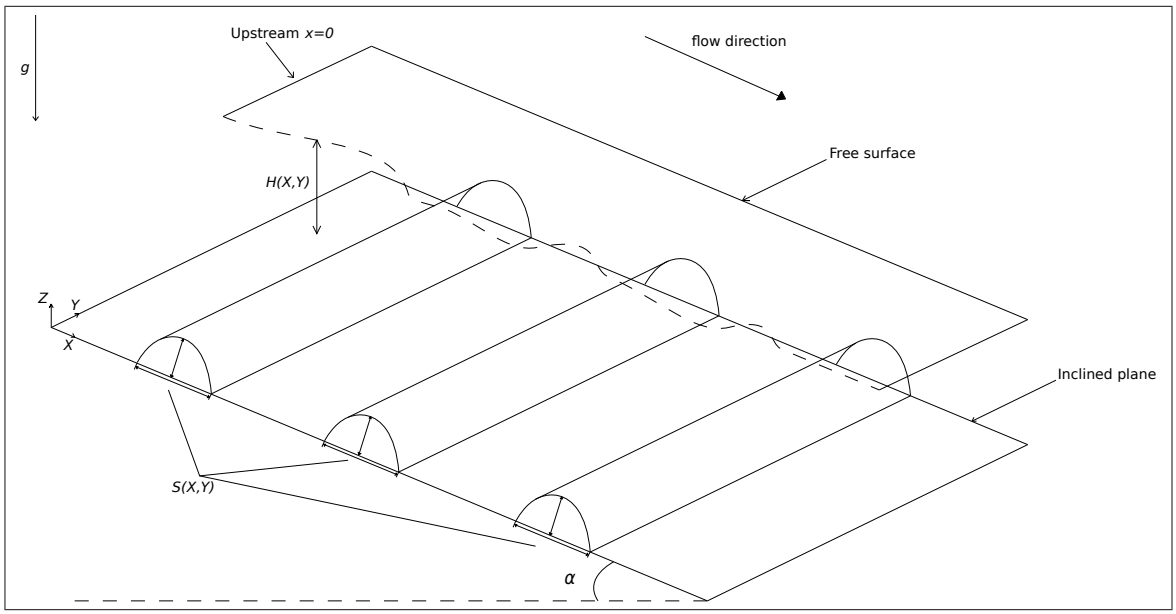

Figure 1.1: Sketch of thin film flow $H(X,Y,T)$ over a topography $S(X,Y)$ on a solid substrate inclined at an angle α to the horizontal.

constant density ρ and viscosity μ, with surface tension σ. The incompressible Navier-Stokes equations which govern the motion of a time-dependent flow in three dimensions are:

$$\rho\left(\frac{\partial \underline{U}}{\partial t}+\underline{U}.\nabla\underline{U}\right)=-\nabla P+\mu\triangle\underline{U}+\rho\underline{g}, \tag{1.12}$$

$$\nabla.\underline{U}=0, \tag{1.13}$$

where $\underline{U}=(U,V,W)$ is fluid velocity, P is pressure and $\underline{g}=g(sin\alpha,0,-cos\alpha)$, where $g=9.81ms^{-2}$ is the acceleration due to gravity and α is the angle of the solid substrate to the horizontal. Taking characteristic thin film flow thickness H_0 and the extent of the substrate L_0, in order to have a valid thin film approximation, the ratio $\varepsilon=H_0/L_0$ must be small. Thus a domain Ω is defined that is large enough to meet this requirement [81, 82].

The Navier-Stokes equations (1.12) and (1.13) may be simplified by using the lubrication theory (also called thin film or long-wave approximation). After all analyses are performed (for details, see [82] for example), results are obtained in terms of following non-dimensional (lower case) variables [82]:

$$\begin{aligned} h(x,y,t)=\frac{H(X,Y,T)}{H_0},\ \ s(x,y)=\frac{S(X,Y)}{H_0},\ \ (x,y)=\frac{(X,Y)}{L_0},\ \ z=\frac{Z}{H_0},\\ (u,v,\frac{w}{\varepsilon})=(U,V,W)\frac{T_0}{L_0},\ \ t=\frac{T}{T_0},\ \ T_0=\frac{\mu L_0}{\sigma\varepsilon^3},\ \ p(x,y)=\frac{2P(X,Y)}{\rho g L_0 sin\alpha}, \end{aligned} \tag{1.14}$$

where $h(x,y,t)$ is the thin film flow thickness, $s(x,y)$ is the non-dimensionalized topography of the substrate (e.g. three cylinder-like obstacles in Figure 1.1), (x,y,z) are non-dimensional Cartesian spatial coordinates, $(u,v,\frac{w}{\varepsilon})$ is the fluid velocity, $p(x,y,t)$ is the pressure vector, t represents time and T_0 is the time scale which has been derived in [173]. The derivation of the resulting Reynolds equation is outside the scope of this thesis however the interested reader is referred to Hayes et al. [107] for a Green's function approach to obtain this thin film equation:

$$\frac{\partial h}{\partial t} = \frac{\partial}{\partial x}\left[\frac{h^3}{3}\left(\frac{\partial p}{\partial x} - 2\right)\right] + \frac{\partial}{\partial y}\left[\frac{h^3}{3}\left(\frac{\partial p}{\partial y}\right)\right]. \tag{1.15}$$

Throughout this fully-developed flow, the pressure field satisfies

$$p = -\frac{\varepsilon^3}{Ca}\triangle(h+s) + 2\varepsilon(h+s-z)cot\alpha, \tag{1.16}$$

where Ca is the so-called capillary number, which reflects the ratio of viscosity μ to surface tension σ. As previously mentioned the ratio of thin film flow thickness H_0 to length scale L_0 is small [81, 82]. The choice of L_0 is, in the context of domain-filled fully-developed flow, equal to the capillary length L_c, from which Equation (1.16) may be further simplified (see [82] for more details) to

$$p = -6\triangle(h+s) + 2\sqrt[3]{6}N(h+s), \tag{1.17}$$

where $N = Ca^{1/3}cot\alpha$ measures the relative importance of the normal component of gravity [25]. For completeness, boundary and initial conditions are required and have to be defined for both dependent variables. Generally it is assumed that there exists zero flux at boundaries. Therefore zero Neumann boundary conditions are applied, with the exception of the upstream boundary, which represents a source of flow. For simplicity, a positive constant Dirichlet boundary condition can be used for the upstream boundary. Initial conditions for each variable can be defined as $h(x,y,t=0) = h_0(x,y)$ and $p(x,y,t=0) = p_0(x,y)$.

In summary, the system of PDEs for fully-developed thin film flow consists of Equations (1.15), (1.17) along with an additional numerical representation of the topography. We return to this system in Chapter 5 where an advanced solver is presented, along with its results. In the following section, a different variant of this model, which uses the lubrication theory to study droplet spreading, is introduced.

1.3.1.2 Droplet Spreading Precursor Film and Moving Contact Lines

The physical phenomenon of a liquid droplet spreading on a substrate has been studied in many scientific fields. In each case, a common demand is to obtain a relatively accurate numerical model for which the solution represents a good approximation to real-world experiments [148]. Many such models are suggested in review paper [174]. The model presented in this thesis is very close to the work of Schwartz, Bertozzi and their co-workers in [24, 63, 194, 195]. Similar to the fully-developed flow, as discussed in the previous section, lubrication theory is applied to the Navier-Stokes Equations (1.12) and (1.13). The non-dimensional variables that are used, are the same as those in Equation (1.14), with one exception of pressure p, which is given by

$$p(x,y) = \frac{L_0 P}{\sigma \varepsilon}. \tag{1.18}$$

The resulting lubrication model is described in detail by Gaskell et al. in [81]. The Reynolds equation for droplet spreading is the following:

$$\frac{\partial h}{\partial t} = \frac{\partial}{\partial x}\left[\frac{h^3}{3}\left(\frac{\partial p}{\partial x} - \frac{B_o}{\varepsilon} sin\alpha\right)\right] + \frac{\partial}{\partial y}\left[\frac{h^3}{3}\left(\frac{\partial p}{\partial y}\right)\right], \tag{1.19}$$

where $B_o = \rho g L_0^2/\sigma$ is the Bond number, measuring the relative importance of gravitational force to surface tension [195]. The main challenge to resolve a droplet spreading model is to accurately capture the moving contact lines between the thin film liquid and the solid substrate (see Figure 1.2). The above equation is based on the assumption of a no-slip condition at the substrate. This assumption gives zero velocity at the substrate because the no-slip condition on the substrate prevents movement, however a non-zero velocity is required at the interface between the air, the drop and the substrate to permit spreading. Figure 1.2 shows how this "paradox" may be resolved, illustrating a cross-section of the droplet on a substrate inclined at an angle α to the horizontal, as well as a precursor film of thickness h^* to over come the no-slip condition at the moving contact lines. The physical phenomenon of a thin precursor film has been detected in the real-world experiments presented in [20, 56]. Alternatively, there are slipping models [174]. Examples of the slipping models can be found in [115, 211], and results obtained from them suggest acceptable accuracy. Diez in [63] compared these two approaches (precursor film versus slipping models), and the same conclusion is reached in terms of model accuracy. However, the precursor film model shows an advantage over the slipping model in terms of efficiency.

Similar to fully-developed flow, [81] introduces an associated pressure equation for

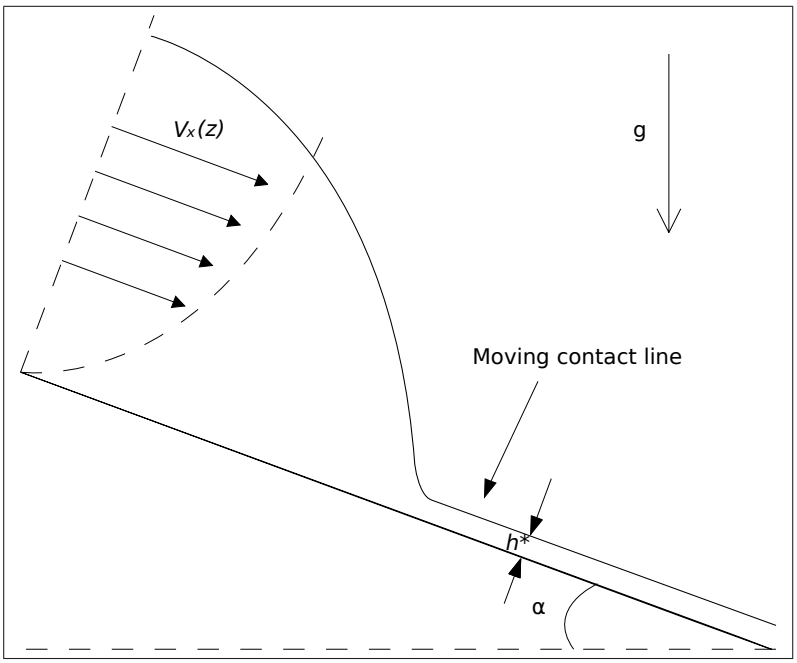

Figure 1.2: Sketch of precursor film model on an inclined substrate at angle α to the horizontal and the parabolic velocity $v_x(z)$ in the droplet liquid. Note the h^* represents a true thin film ahead of droplet, velocity is zero at the substrate.

the droplet spreading model as follows:

$$p = -\triangle(h+s) - \Pi(h) + B_o \cos\alpha(h+s-z), \tag{1.20}$$

where possible topographies may be included through $s(x,y)$, $\Pi(h)$ is a disjoining pressure term which is defined in [194, 195]. This term is used to alleviate the singularity at the moving contact line, and is given by

$$\Pi(h) = \frac{(n-1)(m-1)(1-\cos\Theta_e)}{h^*(n-m)\varepsilon^2}\left[\left(\frac{h^*}{h}\right)^n - \left(\frac{h^*}{h}\right)^m\right], \tag{1.21}$$

where n and m are the exponents of interaction potential and Θ_e is the equilibrium contact angle.

In summary, the droplet spreading model described here is based upon Equations (1.19) to (1.21) for the unknowns h and p. Note that the far-field Dirichlet boundary condition for h takes the form $h = h^*$ in this problem. This is described again in more detail in Chapter 5.

In the next section, phase-field models and the diffuse-interface methods are described.

1.3.2 Phase-Field Models

Modeling the evolution of interfaces between materials is a challenge arising in many scientific fields. In Section 1.3.2.1, we introduce the concept of phase-field models which have been used to solve such interface problems, (along with very brief descriptions of other types of approaches). In Section 1.3.2.2, one of the applications from the field of rapid solidification, which has recently been solved using phase-field software developed in the Scientific Computing research group, at the University of Leeds is described. In Section 1.3.2.3, a two-phase-field Cahn-Hilliard-Hele-Shaw (CHHS) model is discussed. Finally, a more complex multi-phase-field model for tumour growth is presented in Section 1.3.2.4.

1.3.2.1 Introduction

In this subsection, mathematical models are considered which may be used to model an evolving interface between different materials, or different material states (e.g. solid and liquid). Many types of techniques have been developed over the years. Here three main approaches are summarised: free boundary problems with a sharp interface [124]; models with level set methods [133]; and phase-field models [138].

For free boundary problems, real physical descriptions are used to specify the interface region which is then tracked explicitly. This normally results in sharp interfaces and steep gradients in the interface region, see for example [212, 213]. The systems of equations in these models are difficult to solve from a numerical point of view, and require explicitly tracking the interface [7]. Techniques such as a front-tracking method and moving mesh approaches, have been developed to improve this tracking process, applications can be found in [18, 32, 124].

An alternative to explicit tracking is to have an implicit representation of the interface. This can be done using a level set function, and is often referred as the level set method. Examples of this method can be found in [48, 116, 133, 176]. The benefit of using the level set method is the evolving interface can be captured by using a given fixed grid. On the other hand, this representation requires one extra level set variable, which defines the interface when equal to a constant value (typically the choice of value is zero) [198].

The use of diffuse-interface methods as described in [43], provide an alternative technique for solving the interface implicitly. The resulting models are called phase-field models. Generally phase-field variables are smooth (differentiable) and are nearly constant valued throughout most of the domain. However, at the interface regions, between different phases, values of phase-field variables vary smoothly to represent the change in

material or state. Figure 1.3 shows a phase variable ϕ in a two-phase-field system. $\phi = 1$ and $\phi = -1$ represent two different materials or states, respectively. These two phases are separated by a diffuse interface with a thickness ε.

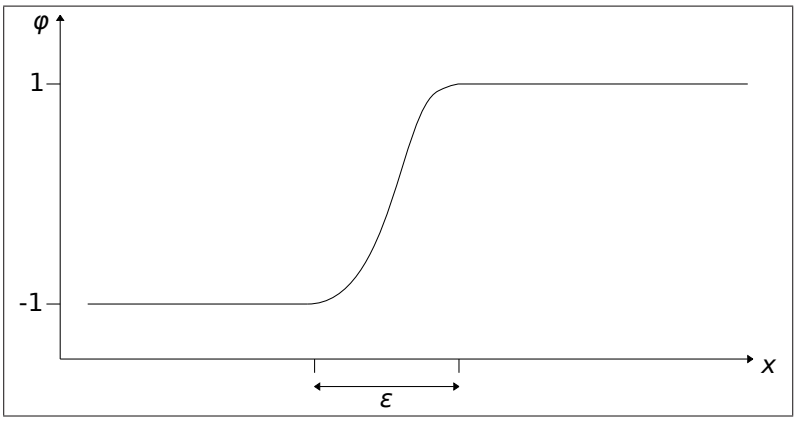

Figure 1.3: Sketch of a phase-field variable ϕ in a two-phase-field system, representing two phases (i.e. materials or states) by values of 1 and -1. Two phases are separated by a diffuse interface with a thickness ε.

If the value of ε in Figure 1.3 becomes very small, models reduce to free boundary problems with the sharp interface issues. However, as long as the interface is relatively diffusive in phase-field models, there is no shape-interface tracking required. This may leads to an enormous numerical advantage. On the other hand, a very diffusive interface may lower the accuracy of the model. Therefore the choice of ε should be made with consideration of such a trade-off.

One may argue phase-field variables do not explicitly represent their subjects from real physics, but only give a phenomenological description. However, phase-field variables are coupled with other variables that do represent real physical properties, such as energy, temperature, concentration and velocity [189]. Furthermore the phase-field equations are derived from physical energy minimization principles [167]. Many applications are described in [13, 41, 47, 138, 142, 167]. Results suggest phase-field models are able to simulate complex morphological evolutions with acceptable accuracy.

In the following section, a 3-D two-phase-field model is introduced, which models binary alloy solidification.

1.3.2.2 A Two-Phase-Field Model For Binary Alloy Solidification

A glossary for the model of binary alloy solidification is give below.

Glossary

Symbol	**Description**
ϕ	Phase-field variable
A	Anisotropy function
τ	Relaxation time function
θ	Temperature
U	Solute concentration
t	Time
x, y, z	Cartesian coordinates
M	Scaled magnitude of the liquidus slope
c_∞	Solute concentration far from the interface
λ	Coupling parameter
k	Equilibrium partition coefficient
D	Dimensionless solute diffusivity
α	Thermal diffusivity
c	Concentration
ε	Strength of the anisotropy
L_e	Lewis number
$\Omega_{undercooling}$	Undercooling parameter

The mathematical model that is presented here has been developed by Karma and Rappel in [127, 128]. Goodyer et al. in [91] solved a 3-D phase-field isothermal system with two dependent variables: a phase-field, $\phi(x,y,z,t)$, and a solute concentration, $U(x,y,z,t)$. Ramirez et al. presented a similar model in 2-D [184], where an additional dependent variable was added, temperature, $\theta(x,y,t)$. Then Bollada et al. in [27] extended the work of [91, 184] to non-isothermal in 3-D. This phase-field model has three dependent variables: a phase-field variable $\phi(x,y,z,t)$, which takes values of $+1$ and -1 in the solid and liquid phases respectively; a dimensionless solute concentration $U(x,y,z,t)$; and temper-

ature $\theta(x,y,z,t)$. The equation for $\phi(x,y,z,t)$ is given as the following [1]:

$$\begin{aligned}
&\tau(c,\phi)A^2(x,y,z)\frac{\partial\phi}{\partial t} = \\
&\nabla\cdot\left(A^2(x,y,z)\nabla\phi\right) - \left[\phi^3-\phi+\lambda\left(\theta+c_\infty U\right)\left(1-2\phi^2+\phi^4\right)\right] \\
&+\frac{\partial}{\partial x}\left[A(x,y,z)\left(-\frac{\partial A(x,y,z)}{\partial\phi_x}\frac{\phi_y|\nabla\phi|^2}{\phi_x^2+\phi_y^2}+\frac{\partial A(x,y,z)}{\partial\phi_x}\frac{\phi_z\phi_x}{\sqrt{\phi_x^2+\phi_y^2}}\right)\right] \\
&+\frac{\partial}{\partial y}\left[A(x,y,z)\left(\frac{\partial A(x,y,z)}{\partial\phi_y}\frac{\phi_x|\nabla\phi|^2}{\phi_x^2+\phi_y^2}+\frac{\partial A(x,y,z)}{\partial\phi_y}\frac{\phi_z\phi_y}{\sqrt{\phi_x^2+\phi_y^2}}\right)\right] \\
&-\frac{\partial}{\partial z}\left[A(x,y,z)\left(\frac{\partial A(x,y,z)}{\partial\phi_z}\sqrt{\phi_x^2+\phi_y^2}\right)\right].
\end{aligned} \tag{1.22}$$

$A(x,y,z)$ is an anisotropy function which is used to identify preferred growth directions of solid alloy, expressed as

$$A(x,y,z) = A_0\left[1+\varepsilon\left(\frac{\phi_x^4+\phi_y^4+\phi_z^4}{|\nabla\phi|^4}\right)\right],$$

and $\tau(c,\phi)$ is a dimensionless relaxation time function which is defined by

$$\tau(c,\phi) = \frac{1}{L_e}+Mc_\infty[1+(1-k)U].$$

Here L_e the Lewis number and $L_e = \alpha/D$ is the ratio of the thermal diffusivity, α, and the dimensionless solute diffusivity, D. It is worth noting that the Lewis number may be very large: this makes the model very difficult to solve as the problem is especially "stiff". On the other hand, a small Lewis number does not accurately represent the realistic parameters for typical metal alloys. Discussions around this Lewis number and relevant applications can be found in [190, 191]. The variable c is the concentration of the secondary component of the alloy, k is the equilibrium partition coefficient, M is a known constant (the scaled magnitude of the liquidus slope), c_∞ is the solute concentration far from the interface, λ is a coupling parameter, and ε is a small parameter that governs the strength of the anisotropy.

[1] Bollada et al. in [27] used different notation to present this model, however, the presentation used here is mathematically equivalent to the one in [27].

The equation for the dimensionless concentration field $U(x,y,z,t)$ is given by

$$\left(\frac{1+k}{2}-\frac{1-k}{2}\phi\right)\frac{\partial U}{\partial t} =\nabla\cdot\left\{D\frac{1-\phi}{2}\nabla U+\frac{1}{2\sqrt{2}}\left[1+(1-k)U\right]\frac{\partial\phi}{\partial t}\frac{\nabla\phi}{|\nabla\phi|}\right\} + \frac{1}{2}\left[1+(1-k)U\right]\frac{\partial\phi}{\partial t}, \tag{1.23}$$

where the non-dimensional concentration field $U(x,y,x,t)$ is related to the concentration c via

$$U=\frac{\left(\frac{2c/c_\infty}{1+k-(1-k)\phi}\right)}{1-k}.$$

The equation for the temperature field $\theta(x,y,z,t)$ is given as the following:

$$\frac{\partial\theta}{\partial t}=\alpha\triangle\theta+\frac{1}{2}\frac{\partial\phi}{\partial t}. \tag{1.24}$$

In order to complete this model, a domain Ω is defined, which is further assumed to be sufficiently large that its boundaries $\partial\Omega$ are far from solid alloy, thus the boundaries have no effect on the evolution of the solid-liquid interface away from these far-field boundaries. Zero Neumann boundary conditions are therefore applied for simplicity:

$$\frac{\partial\phi}{\partial\nu}=\frac{\partial U}{\partial\nu}=\frac{\partial\theta}{\partial\nu}=0 \quad \text{on} \quad \partial\Omega, \tag{1.25}$$

where ν denotes the outward-pointing normal to the boundary $\partial\Omega$. Initially, $\phi(x,y,z,t=0)$ can be defined as a small spherical seed of solid phase in the middle of the domain, in order to initiate the solidification process [91]. Reasonable choices for $U(x,y,z,t=0)$ and $\theta(x,y,z,t=0)$, are straightforward to obtain given $\phi(x,y,z,t=0)$, see [27].

As mentioned previously, the results of this model that are presented in this thesis are selected to demonstrate the new software functionality of our solver that is described in Sections 4.2 and 4.3. In the next section, a two-phase-field model for studying a mixture of fluids is described.

1.3.2.3 A Two-Phase-Field Model: Cahn-Hilliard-Hele-Shaw System of Equations

A glossary for the Cahn-Hilliard-Hele-Shaw system of equations is given below.

Glossary

Symbol	Description
ϕ	Phase-field variable

μ	Chemical potential
p	Pressure
$\boldsymbol{u}$	Advective velocity
γ	Parameter in pressure equation
ε	Interface thickness

The Cahn-Hilliard (CH) equations originate from the work in [42, 43], and are used to model spinodal decomposition. Then Shinozaki and Oono [201] used a variation of the CH equations to simulate binary-fluid spinodal decomposition in a Hele-Shaw cell. The resulting model is called Cahn-Hilliard-Hele-Shaw (CHHS) system of equations. This CHHS model that is describe in this thesis is from the work of Wise [225] which is a simplified version of the model derived by Lee et al. in [144, 145]. This model can be used to study a mixture of two binary fluids, and is given by

$$\frac{\partial \phi}{\partial t} = \triangle \mu - \nabla \cdot (\phi \boldsymbol{u}), \tag{1.26}$$

$$\mu = \phi^3 - \phi - \varepsilon^2 \triangle \phi, \tag{1.27}$$

$$\boldsymbol{u} = -\nabla p - \gamma \phi \nabla \mu, \tag{1.28}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{1.29}$$

where phase-field variable $\phi(x,y,z,t) = \pm 1$ is the pure fluids, $\mu(x,y,z,t)$ represents the chemical potential, $\boldsymbol{u}(x,y,z,t)$ is the advective velocity, $p(x,y,z,t)$ is a pressure and ε is a constant. Additionally, in the CHHS model we take $\gamma > 0$. If $\gamma = 0$, Equations (1.26), (1.28) and (1.29) reduces to the CH equations [42].

A domain Ω and its boundary $\partial\Omega$ are defined, and no-flux boundary conditions are assumed. Therefore the boundary conditions for the CHHS system of equations are the following:

$$\frac{\partial \phi}{\partial \nu} = \frac{\partial \mu}{\partial \nu} = \frac{\partial p}{\partial \nu} = 0 \text{ on } \partial\Omega, \tag{1.30}$$

where ν denotes the outward-pointing normal to the boundary $\partial\Omega$.

Benefiting from using the diffuse-interface method, the initial condition $\phi(x,y,z,t = 0)$ may have shape changes. However, a diffusive initial condition is used in Chapter 6 in order to represent diffusive interface of fluids. Using $\phi(x,y,z,t = 0)$ to obtain $\mu(x,y,z,t = 0)$ is straightforward from Equation (1.27). On the other hand, since $p(x,y,z,t)$ only appears in its gradient form in the CHHS system, a solver is required to obtain $p(x,y,z,t = 0)$, details are given in Chapter 6.

Based upon this CHHS system of equations, a multi-phase-field model is proposed

in [227] to simulate tumour growth. We explain such a model in the next section.

1.3.2.4 A Multi-Phase-Field Model of Tumour Growth

A glossary for the multi-phase-field model of tumour growth is given below.

Glossary

Symbol	**Description**
ϕ_W	Volume fraction of extracellular fluid
ϕ_V	Volume fraction of viable tumour tissue
ϕ_D	Volume fraction of dead tumour tissue
ϕ_H	Volume fraction of healthy tissue
ϕ_T	Total tumour volume fraction
$\underline{u}_S$	Tissue velocity
p	Cell-to-cell (solid) pressure
n	Concentration of nutrient
μ	Intermediate variable
S_T	Net source of tumour cells
S_D	Net source of dead cells
M	Mobility constant
$f(\phi_T)$	Quartic double-well potential
ε	Interface thickness
κ	Tissue motility function
γ	Excess adhesion force at the diffuse tumour/host-tissue interface
D	Diffusion coefficient
D_H	Nutrient diffusivity in the healthy tissue
T_c	Nutrient capillary source term
V_P^H	Rate of nutrient transfer from vasculature in healthy cells
V_P^T	Rate of nutrient transfer from vasculature in tumour cells
n_N	Necrotic limit (if the nutrient is lower then dead cells start to appear)
n_C	Nutrient level in the capillaries
λ_L	Birth rate of tumour cells from mitosis
λ_A	Death rate of cells from apoptosis
λ_N	Death rate of cells from necrosis
$G(\phi_T)$	Continuous cut-off function
$\mathscr{H}$	Heaviside function

$\mathscr{H}^{\text{smooth}}$	Smoothed Heaviside function
$\varepsilon^{\text{Heaviside}}$	Interface thickness in the smoothed Heaviside function
$Q(\phi_T)$	An interpolation function in nutrient capillary source term
x, y, z	Cartesian coordinates

The mathematical model from Cristini et al. [51] is extended by Wise et al. in [227]. It is a multi-phase-field model, based upon the CHHS system of equations [225] from the previous section, which simulates tumour growth. This model consists of fourth-order nonlinear advection-reaction-diffusion equations for multiple cell-species, as well as coupling with reaction-diffusion equations for substrate components. It is described from a numerical point of view in [226] by Wise, Lowengrub and Cristini. This model is described here. Furthermore it is discussed again in Chapter 7 in detail.

There are in total five phase-field variables which represent volume fractions in this model, and they are summarised in the glossary at the beginning of this subsection. In addition, there are three assumptions amongst these volume fractions. Firstly, it is assumed the extracellular fluid volume fraction is everywhere constant, $\phi_W(x,y,z,t) = \phi_{W,0} = \text{constant}$. After this assumption, the tumour model consists of multiple solid cell fractions. Secondly, cells are assumed to be close-packed, and this leads to the sum of the healthy cell volume fraction ϕ_H, the viable tumour cell volume fraction ϕ_V and the dead tumour cell volume fraction ϕ_D equals to 1 (i.e. $\phi_H + \phi_V + \phi_D = 1$). Thirdly, it is further assumed that inside of tumour there are only two types of cells: viable and dead. This indicates the total tumour cell volume fraction ϕ_T is the sum of ϕ_V and ϕ_D (i.e. $\phi_T = \phi_V + \phi_D$). Based upon these three assumptions, there are only two phase-field variables that are required to be solved, and they are ϕ_T and ϕ_D. Once the solutions of these two variables are obtained, other variables may be extrapolated from the assumptions made.

The ϕ_T is evaluated by the following Cahn-Hilliard-type advection-reaction-diffusion equations.

$$\frac{\partial \phi_T}{\partial t} = M\nabla\cdot(\phi_T\nabla\mu) + S_T - \nabla\cdot(\underline{u}_S\phi_T), \tag{1.31}$$

$$\mu = f'(\phi_T) - \varepsilon^2\nabla^2\phi_T, \tag{1.32}$$

where $M > 0$ is the mobility constant, $f(\phi) = \phi^2(1-\phi)^2/4$ is the quartic double-well potential, $\varepsilon > 0$ is the interface thickness parameter between healthy and tumour tissue, S_T and S_D are the net sources of tumour cells which depend on ϕ_V and ϕ_D (see Chapter 7 for full details).

A dynamical equation for solving the volume fraction of dead tissue ϕ_D is used:

$$\frac{\partial \phi_D}{\partial t} = M\nabla \cdot (\phi_D \nabla \mu) + S_D - \nabla \cdot (\underline{u}_S \phi_D). \tag{1.33}$$

According to the assumption that cells are close-packed, the viable tumour tissue volume fraction ϕ_V can be computed through $\phi_V = \phi_T - \phi_D$ and $\phi_H = 1 - \phi_T$.

The tissue velocity $\underline{u}_S$ is assumed to obey Darcy's law. In order to obtain the velocity $\underline{u}_S$, the cell pressure p is computed first, then back-calculate for $\underline{u}_S$. The tissue velocity is given by

$$\underline{u}_S = -\kappa(\phi_T, \phi_D)(\nabla p - \frac{\gamma}{\varepsilon}\mu\nabla\phi_T),$$
$$\nabla \cdot \underline{u}_S = S_T,$$

where $\kappa > 0$ is the tissue motility function and $\gamma \geq 0$ is a measure of the excess adhesion. Together with the two equations above, a Poisson equation for the cell pressure p can be constructed:

$$-\nabla \cdot (\kappa(\phi_T, \phi_D)\nabla p) = S_T - \nabla \cdot (\kappa(\phi_T, \phi_D)\frac{\gamma}{\varepsilon}\mu\nabla\phi_T). \tag{1.34}$$

A quasi-steady equation is given for the nutrient concentration through diffusion:

$$0 = \nabla \cdot (D(\phi_T)\nabla n) + T_c(\phi_T, n) - n(\phi_T - \phi_D), \tag{1.35}$$

where $D(\phi_T) = D_H(1 - Q(\phi_T)) + Q(\phi_T)$ is the diffusion coefficient, D_H is the nutrient diffusivity in the healthy tissue, $Q(\phi_T)$ is an interpolation function, and $T_c(\phi_T, n) = (v_P^H(1 - Q(\phi_T)) + v_P^T Q(\phi_T))(n_C - n)$ is the nutrient capillary source term, $v_P^H \geq 0$ and $v_P^T \geq 0$ are constants specifying the degree of pre-existing uniform vascularization, $n_C \geq 0$ is the nutrient level in capillaries.

The model equations are valid throughout a regular domain Ω, there are no internal boundary conditions for the solid tumour, the necrotic core or other variables. Therefore, only one set of outer boundary conditions is imposed:

$$\mu = p = 0, \ n = 1, \ \frac{\partial \phi_T}{\partial \nu} = \frac{\partial \phi_D}{\partial \nu} = 0 \text{ on } \partial\Omega, \tag{1.36}$$

where ν denotes the outward-pointing normal to the boundary $\partial\Omega$.

Initial conditions are also required. A specific set for $\phi_T(x, y, z, t = 0)$ and $\phi_D(x, y, z, t = 0)$ is defined later on in this thesis. After both of these are given, $\mu(x, y, z, t = 0)$, $p(x, y, z, t = 0)$ and $n(x, y, z, t = 0)$ can be determined. This model is further discussed in detail in Chapter 7. In the following section, an overview on the structure of this thesis is given.

1.4 Overview of Thesis

In this section, an overview of this thesis is presented. In Chapter 2, a number of methods are introduced which are used to obtain the discretizations of these models. Within these descriptions, we emphasize the focus in this thesis which is using finite difference methods (see Section 2.1.1.1), cell-centred Cartesian grids with Neumann, Dirichlet boundary conditions (see Section 2.1.1.2) and backward differentiation formulae (see Section 2.1.2.2). In order to solve the resulting system, some common solution methods are described. We start with solution methods for linear problem which consist of sparse matrices and their storage, direct methods like Gaussian elimination via the LU factorisation, a number of iterative methods and the technique of preconditioning. We go on to explain the solution methods for nonlinear problems. The main focus of this thesis, the multigrid methods, are then described in detail. To further improve the efficiency, parallel computing may be employed, and we link this with the multigrid methods.

Having discussed a number of scientific computing techniques in Chapter 2, we introduce the software tools used in this thesis in Chapter 3 which consist of a software library PARAMESH (see Section 3.2) and a multigrid solver Campfire (see Section 3.3). These two are firstly introduced for their backgrounds and connections. Campfire is the main software tool used as the numerical solver in this thesis, and we summarise the implemented adaptive multigrid solver in detail.

Chapter 4 is where we discuss the first of five models: the model of binary alloy solidification. The model is introduced again, and a novel implementation made with Campfire is discussed. Results from solving this model using the multigrid solver in Campfire are presented. These results are mainly generated by Bollada and are also presented in [27].

The models of Thin film flows are presented in Chapter 5. Its discretizations are included, and novel implementations of this model with Campfire are discussed. Firstly, non-time-dependent equations are implemented in Campfire for the first time. Then we described the use of Dirichlet boundary conditions in Campfire which has not been achieved before. Finally, the convergence test based upon solution restriction is explained, and these tests are later on employed to demonstrate second order convergence rate. Results on the droplet spreading models are summarised in Section 5.4. One of the derivation of the thin film models is the model of fully-developed flows, and it is presented in Section 5.5. We include its discretizations and the results for validation and convergence tests.

The Cahn-Hilliard-Hele-Shaw system of equations are presented Chapter 6. Two different temporal discretization schemes are introduced. The first one is from [225] and it is a semi-implicit scheme. The second one is our fully-implicit scheme with the BDF2

method. The implementation of this problem in Campfire is summarised. We present the discrete systems followed by these two different schemes, and the implementation of two multigrid solvers is included. Firstly, we explain our multigrid solver with the fully-implicit scheme. Then we discuss the implementation used in [225]. In addition, both solvers are implemented in Campfire for the purpose of comparison. Results from solving this model are presented in Section 6.4. Chapter 6 is concluded with a discussion on different Gauss-Seidel iteration strategies.

The multi-phase-field model of tumour growth is discussed in Chapter 7. First of all, a brief literature review on tumour modeling is summarised. The model of tumour growth that is studied in this thesis is from [226] and is presented again. We describe the implementation. A discussion is presented on the implementation used in [226] and the reasons why a second order convergence rate is not obtained. Having followed the suggestions from Wise et al. stated in [226], we present our implementation in Campfire. Results of this tumour growth model are presented in Section 7.4. Chapter 7 is concluded with a discussion on the issues of multigrid convergence from the pressure variable.

In Chapter 8, conclusions and further work are discussed.

1.5 Main Achievements of the Thesis

The main achievements of this thesis can be separated into two parts. The first part is the achievements towards the software implementation (this software, Campfire, is described in Chapter 3), and these can be summarised as

- extending Campfire to non-time-dependent equations,
- expanding the choices of Campfire's boundary conditions,
- increase the restart capacity of Campfire,
- improvement to the parallel implementation of the software library, PARAMESH (which the Campfire is built upon).

The second part is that we obtained a second-order convergence rate from the multi-phase-field model of Wise et al. in [226]. This is done by combining second-order FDM, BDF2, penalty terms and smoothing Heaviside function all together.

These achievements are described in detail in the thesis.

Chapter 2

Introduction to Scientific Computing Techniques

In the previous chapter, a selection of parabolic PDEs and systems were presented. These ranged from simple linear and nonlinear diffusion equations through to systems for thin film flows and phase-field models. In order to obtain approximations of the true solutions of these mathematical models, a number of scientific computing techniques are required. Within Section 2.1, spatial and temporal discretization schemes are described, for obtaining discrete systems from mathematical models that are originally continuous in space and time. Approximations using spatial and temporal discretizations can be further improved in terms of efficiency by applying adaptivity, this is discussed in Section 2.2. In order to solve these discrete systems, a number of standard linear and nonlinear solvers are discussed in Section 2.3, and a state-of-the-art multigrid algorithm is described in Section 2.4, along with four of its variations. A large amount of numerical computations are typically required for large simulations (e.g. using very fine spatial discretizations), even when multigrid methods are used. To further improve the computational performance, the use of parallel computing is introduced in Section 2.5.

2.1 Discretization Schemes

In this section, possible spatial and temporal discretization schemes for the parabolic models which are presented in the previous chapter are described. In Section 2.1.1 several options for spatial discretization are introduced. In Section 2.1.2 concepts of explicit and implicit temporal discretization schemes, with examples are discussed. Furthermore, the family of Backward Differentiation Formulae (BDF), which is used extensively in this thesis, is described in detail.

2.1.1 Spatial Discretizations

The mathematical models described in the previous chapter are continuous in space and time. In order to obtain numerical approximations to the solutions of these models, methods of spatial discretization are applied. These discretization techniques seek to approximate the spatial parts of the problem with a finite number of degrees of freedom. Typically these are defined based upon decomposing the spatial domain Ω into a set of points or cells (however other approaches, such as spectral methods [89] and collocation methods [139] do exist). Here the three most common choices of point/element based methods are introduced, with the most detail coming in Section 2.1.1.1, where a Finite Difference Method (FDM) is described, which is extensively used in this thesis. The outcome of each of these spatial discretization schemes, when applied to a parabolic PDE, is to reduce the problem to an initial value system of ordinary differential equations (ODEs). Such systems are discrete in space, however still continuous in time (temporal discretization schemes are introduced later in Section 2.1.2).

Alternative approaches, which will not be considered further, include the Finite Element Method (FEM). The FEM was proposed in the 1940s [214], and one of the initial applications was developed for the solution of aircraft structural problems [185]. Thereafter, its potential for solutions of a variety of applied science and engineering problems, as well as solving different types of PDEs were recognized [208]. Until today, it is considered to be one of the best methods for efficiently solving a large range of practical problems [185], especially if the domain is geometrically complex, although such problems are not considered in this thesis. The FEM decomposes the spatial domain into a number of non-overlapping elements, each of which represents a sub-section of the domain. Then the FEM uses integrals over the individual elements to form a full approximation across the domain. A typical triangle element is often used in 2-D, where three grid points are required to define such an element. Based upon the shape of elements and their spatial dimension, more grid points are needed for each element. Commonly, the basis function

of each grid point is defined to be 1 at the point and linearly reduces to 0 upon reaching adjacent grid points along the edges. This is referred to as a low-order polynomial, although higher-order polynomial approximations are possible by employing extra points (e.g. edge-centred points and cell-centred points). [187, 231] provide good introductory guides to the FEM, and some of its applications can be found in [19, 106, 196].

Another popular discretization scheme is the Finite Volume Method (FVM). This method also decomposes the domain into a finite number of elements, then uses averaged values and volume integrals over elements to form an approximation of the solution [218]. The approximation within individual element is local, thus imposing no requirements on the grid structure, and therefore unstructured grids can also be used. The FVM is often applied to hyperbolic conservation laws, and an application of this method on non-uniform grids can be found in [120].

The Discontinuous Galerkin (DG) method combines the FEM and the FVM together, and overcomes the limitation of the FVM on achieving higher-order accuracy on general unstructured grids by applying higher-order polynomial basis functions [114]. The DG method also maintains local conservation and flexibility in the choice of the numerical flux. However, one of the trade-offs of using the DG method is having to increase the total degrees of freedom. An application of the DG method can be found in [55].

In the following section, the FDM is described.

2.1.1.1 Finite Difference Methods

The FDM decomposes the continuous spatial domain into a finite number of grid points, and a discrete difference operator is used in place of the differential operator at each grid point. On a cubic domain Ω in 3-D, a set of uniform grid points are defined as the following:

$$Z_h = \{(x,y,z) : x = ih, y = jh, z = kh \quad i,j,k = 0,...,N-1\}, \tag{2.1}$$

where (x,y,z) are Cartesian spatial coordinates, N is the number of grid points in each coordinate direction and $h = 1/(N-1)$ is the equal distance between the adjacent points. These grid points are further separated into two categories: internal points and boundary points. The set of internal points is denoted by $\Omega_h = \Omega \cap Z_h$, and $n = N-2$ is the number of internal grid points in each coordinate direction. The set of boundary points is denoted by $\partial\Omega_h = \partial\Omega \cap Z_h$, and $\overline{\Omega}_h = \Omega_h \cup \partial\Omega_h$ is the set of all grid points on the domain Ω and its boundary. An example shown in Figure 2.1 has $N = 10$ and 8×8 internal grid points. The FDM is only applied to the internal points Ω_h, and with the assumption that boundary

conditions (e.g. Neumann and/or Dirichlet boundary conditions from Equations (1.6) and (1.5)) have been applied to boundary points $\partial\Omega_h$.

As an example, to approximate the 2-D Laplace operator which is previously introduced in Equation (1.3),

$$\triangle u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}, \tag{2.2}$$

a standard second-order central finite difference scheme is used for the second derivative of u in the x and the y directions. The resulting discrete system is given as the following:

$$\begin{aligned} \triangle_h u_h &= \frac{1}{h^2}\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right) + \frac{1}{h^2}\left(u_{i,j-1} - 2u_{i,j} + u_{i,j-1}\right) \\ &= \frac{1}{h^2}\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}\right), \end{aligned} \tag{2.3}$$

where $u_h = \underline{u}_h(\boldsymbol{x})$ is the so-called grid point function, $\boldsymbol{x} \in \Omega_h$ and the element of $\underline{u}_h$ corresponding to (ih, jh) is denoted as $u_{i,j}$. Equation (2.3) is also called five-point stencil scheme, where for each internal grid point Ω_h, four neighbours and itself are involved in computing the approximation, as shown at a grid point a in Figure 2.1. The grid points which are needed for the computation in each stencil are marked as $\circ$. This approximation and its equivalent seven-point stencil scheme in 3-D, are needed for many terms in these mathematical models that are described in Chapter 1, for example, variable p in Equation (1.15); $-6\triangle(h+s)$ term in Equation (1.17); and both $\triangle\mu$ and ϕ terms in the CHHS which is represented in Equations (1.26) to (1.29) etc. The resulting systems after applying the spatial discretization schemes are systems of ordinary differential equations (ODEs) for the unknowns $u_{i,j}(t)$ in 2-D or $u_{i,j,k}(t)$ in 3-D.

There is also a central finite difference scheme for the first derivative of variables. For example, such a scheme can be applied to $\nabla \cdot (\underline{u}_S \Phi_T)$ term in Equation (1.31) of the tumour model. For the purpose of demonstration, the first derivative of u in the x and the y directions of using this scheme is approximated as the following:

$$\nabla_h \cdot u_h = \frac{u_{i+1} - u_{i-1}}{2h}. \tag{2.4}$$

To increase the order of accuracy of the finite difference discretization scheme, larger stencils can be used. Figure 2.1 shows a nine-point stencil on another vertex-centred grid point b, and points that are needed by this nine-point stencil are marked as $\circ$. Although, it is advantageous to use a higher-order scheme provided the solution is sufficiently smooth, issues may arise if parallel computing and mesh partitioning are taken into account. In the context of this thesis, due to the use of parallel computing, only the five-point stencil

in 2-D (and its equivalent seven-point stencil in 3-D) is applied, the reasons are explained in Section 2.5.3.

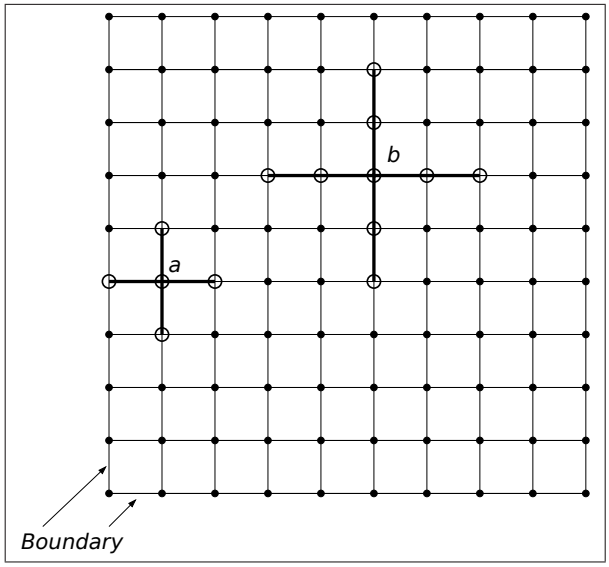

Figure 2.1: Sketch shows a 2-D vertex-centred grid with 8×8 internal points; and a five-point stencil on a vertex-centred grid point a; and a nine-point stencil on another vertex-centred grid point b. The grid points which are needed for the computation in each stencil are marked as $\circ$.

In Figure 2.1, an example is shown in a vertex-centred grid, where a boundary condition can be specified on boundary points which are directly positioned on the boundary. There are also other types of grid points, such as face-centred, edge-centred and cell-centred points. The Cartesian grids with cell-centred points have been used extensively in this thesis, and an example of such a grid type is shown in Figure 2.2. Although it is possible to use vertex-centred grid points as the boundary points (which are geometrically on the boundary) for cell-centred grids, the use of ghost cells (shown in Figure 2.2 as $\circ$) is also common and is our approach. Following the definition of vertex-centred points in Equation 2.1, the cell-centred grid points (including these ghost cells) on a cubic domain Ω in 3-D are defined as

$$Z_h = \left\{ (x,y,z) : x = \left(i - \frac{1}{2}\right) h, y = \left(j - \frac{1}{2}\right) h, z = \left(k - \frac{1}{2}\right) h \quad i,j,k = 0,...,N-1 \right\}, \tag{2.5}$$

where (x,y,z) are Cartesian spatial coordinates, N is the number of grid points in each coordinate direction, $h = 1/(N-1)$ is the equal distance between the adjacent points and $n = N-2$ is the number of internal grid points in each coordinate direction. On the same Cartesian grid shown in both Figures 2.1 and 2.2, there are 10×10 vertex-centred grid points and 8×8 of them are internal points; on the other hand, there are 11×11 cell-

centred grid points (including ghost cells) and 9×9 of them are internal points. Spatial discretization schemes of FDM, like the five-point stencil can be applied in the same way as to the vertex-centred grids. However, ghost cells that provide boundary condition on cell-centred grid are not positioned on the boundary itself. In the following Section, Dirichlet and Neumann boundary conditions on cell-centred grids and the use of ghost cells are described.

2.1.1.2 Boundary Conditions

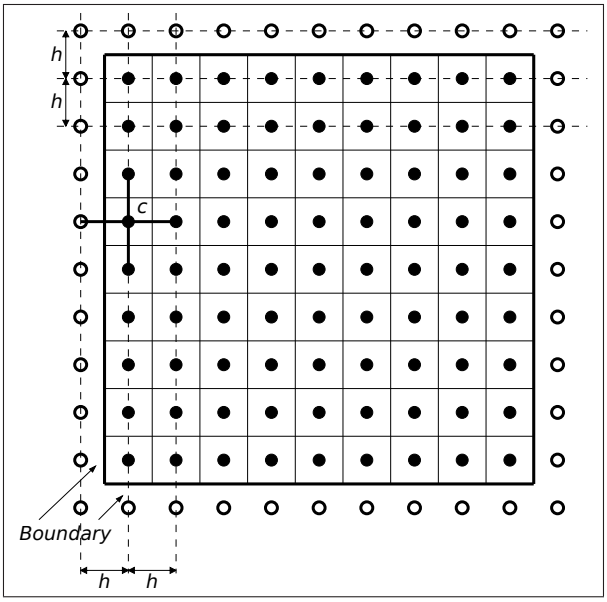

Figure 2.2: Sketch shows a 2-D cell-centred grid with 9×9 internal grid points (marked as $\bullet$), and the ghost cells (marked as $\circ$) which are outside the actual boundary. This type of ghost cells are used to provide the boundary conditions on the cell-centred grids.

In Figure 2.2, ghost cells (marked as $\circ$) on a cell-centred grid which are used to represent boundary points are illustrated. These ghost cells are set outside the actual boundary. Therefore, values at these points that are used to indicate boundary conditions have to be treated very carefully.

The zero Neumann boundary condition, $\partial u/\partial \nu = 0$ on the boundary $\partial\Omega$, requires boundary points to copy the exact values from the nearest internal grid points. This has no conditions on the position of the boundary points, therefore, for boundary points in both types of grids (i.e. vertex-centred grid in Figure 2.1 and cell-centred grid in Figure 2.2), the zero Neumann boundary condition can be applied in the same way. The computation which needs the values from boundary points (e.g. applying the five-point stencil scheme on near-boundary grid point) can also be done in the same way on the cell-centred grids.

Previously, the five-point stencil scheme is applied to a near-boundary vertex-centred grid point a in Figure 2.1. Here the equivalent five-point stencil on a near-boundary cell-centred grid point c is demonstrated in Figure 2.2, which takes values from one boundary point, three internal grid points and itself. Although, this boundary point is not positioned on the boundary, as long as it mirrors the value on point c, the zero Neumann boundary condition is correctly imposed. This zero Neumann boundary condition is employed in all five mathematical models that are described in Chapter 1, for fully-developed flow and droplet spreading problem, it is discussed in Sections 1.3.1.1 and 1.3.1.2; and for other three phase-field models, see Equations (1.25), (1.30) and (1.36). A non-zero Neumann boundary condition increases the complexity in the case of cell-centred grids, however this is neglected since no such boundary condition is used in this thesis.

On the other hand, the Dirichlet boundary condition $u = g$, which is defined in Equation (1.5) generally requires a specific value to be applied on boundary $\partial\Omega$. For the vertex-centred grid, values of the function g can be directly applied to the boundary points which are positioned at the boundary (see Figure 2.1). However, values for the ghost cells on a cell-centred grid need to be set to reflect the correct values on the boundary (see Figure 2.2). For example, the function $g(x,y,t) = 0$ yields the average between the ghost cells and their nearest internal points equals to 0. Or if the function $g(x,y,t) = g$, implies

$$\frac{u_{\text{ghost cell}} + u_{\text{nearest internal point}}}{2} = g,$$

which leads to

$$u_{\text{ghost cell}} = 2g - u_{\text{nearest internal point}}. \tag{2.6}$$

The Dirichlet boundary condition is needed in the model of tumour growth for variable μ, p and n (see Equation (1.36)), and the model of fully-developed flow (discussion is included in Section 1.3.1.1).

Having described spatial discretization schemes and the resulting spatial discrete systems, in the next section, temporal discretization schemes are discussed. They are used to discretize the initial value system of ODEs that are still continuous in time.

2.1.2 Temporal Discretizations

In the previous section, spatial discretization schemes are described and used to discretize parabolic PDEs into initial value systems of ODEs. This semi-discretization in space results in systems that are continuous in time, therefore temporal discretization schemes are needed to obtain the fully-discrete systems which can be used for numerical approxima-

tions.

There are two main types of temporal discretization scheme: explicit methods and implicit methods [34]. In order to present these temporal discretization methods clearly, a single initial value ODE is considered, and is written in the following form:

$$\frac{du}{dt} = f(t,u). \tag{2.7}$$

Applying a spatial discretization scheme (e.g. FDM) will produce a similar equation in vector form, in terms of vectors $u_h(t)$ and $f_h(t,u_h)$. However, for simplicity, they are presented as scalars here to demonstrate the temporal discretization schemes. In this section, both explicit and implicit methods are introduced.

2.1.2.1 Explicit Versus Implicit

The concept of explicit methods is based upon using only known values from the previous time steps to compute values at the current time step. A straightforward example of an explicit method is the forward Euler method. When this is applied to Equation (2.7), the resulting equation is

$$u^{\tau+1} = u^{\tau} + \delta t f(t^{\tau}, u^{\tau}), \tag{2.8}$$

where superscript τ indicates the solution from the previous time step, $\tau+1$ means the unknown solution at the current time step, and δt is the size of the time step. The forward Euler method of Equation (2.8) is first-order accurate in time [34]. There are other types of explicit methods that can provide a higher order of accuracy, such as the explicit Runge-Kutta methods, and these methods are examples of so-called "one-step" methods where $u^{\tau+1}$ is determined only from knowledge of u^{τ}. Higher order explicit methods may also be derived from multi-step schemes, where u^{τ}, $u^{\tau-1}$, $u^{\tau-2}$, etc. are used to determine $u^{\tau+1}$. An application with a second-order Runge-Kutta method can be found in [150], and a description of a fourth-order Runge-Kutta method is given in [34].

The discrete equations resulting from the application of explicit methods are easy to solve, and this is the primary advantage of explicit methods. However, explicit methods generally have the property that they are only conditionally stable. This means that the stability of the solution is only guaranteed when the size of time step δt is sufficiently small. Furthermore, the finer the mesh resolution that is used in the spatial discretization methods, the smaller time step δt needs to be. For second-order parabolic systems or equations, $\delta t = \mathscr{O}(h^2)$, such as Equation (1.2); and for fourth order parabolic systems, $\delta t = \mathscr{O}(h^4)$, for example, the model of fully-developed flow in Equations (1.15) and

(1.17) [34], this gives an enormous computational disadvantage when large (very high spatial resolution) simulations are required.

The use of implicit methods is typical in order to overcome these issues with conditional stability. This is because such methods typically have much greater regions of stability than explicit methods, and sometimes they can even be unconditionally stable. Discussions on this can be found in [59, 119, 225]. A straightforward example of an implicit method is the first-order backward Euler method. When this is applied to Equation (2.7), the resulting equation is

$$u^{\tau+1} = u^{\tau} + \delta t f(t^{\tau+1}, u^{\tau+1}). \tag{2.9}$$

This backward Euler method is called implicit, because the function $f(t,u)$ needs to be evaluated at $u^{\tau+1}$ in order to obtain $u^{\tau+1}$. Another example is the second-order Crank-Nicolson method [50], where Equation (2.7) is reduced to the following:

$$u^{\tau+1} = u^{\tau} + \frac{\delta t}{2} f(t^{\tau}, u^{\tau}) + \frac{\delta t}{2} f(t^{\tau+1}, u^{\tau+1}). \tag{2.10}$$

Applications of this method can be found in [81,226]. Other examples of implicit schemes include implicit Runge-Kutta schemes [92], and implicit multi-step methods [141].

In comparison to the explicit methods, the system resulting from using an implicit method becomes an implicit equation or, in the case of ODE systems, a large algebraic system of equations. If the given function f_h is nonlinear, the resulting algebraic systems become nonlinear as well. Furthermore, common spatial discretization schemes only use a relatively few neighbouring points to compute each internal point (e.g. the five-point stencil in 2D). This leads to large sparse systems of linear or nonlinear algebraic equations (see Sections 2.3.1 and 2.3.2 for more details). To efficiently solve these systems, sparse/iterative solvers are employed including Krylov methods and multigrid techniques. The detail of such methods is described later. In the following section, a family of implicit temporal discretization schemes are discussed. The variant with second-order accuracy in time is extensively used in this research.

Before we proceed to the next section, it is worth noting that the implicit schemes may be further separated into two categories: fully-implicit and semi-implicit. The presented backward Euler method in Equation (2.9) is a fully-implicit scheme, because for the known function $f(t,u)$, it uses the unknown solution from the current time step. On the other hand, the presented Crank-Nicolson method is an implicit scheme, where solutions from two time steps are applied to the function $f(t,u)$.

2.1.2.2 Family of Backward Differentiation Formulae

The Backward Differentiation Formulae (BDF) are implicit temporal discretization schemes. The backward Euler method, which is previously presented in Equation (2.9), belongs to the family of BDF methods, and is termed BDF1. The number 1 indicates the order of accuracy (and also the number of previous time steps from which solutions are needed). Given Equation (2.7), a general form for the BDF family can be presented as the following

$$u^{\tau+1} + \sum_{j=0}^{p-1} \alpha_j u^{\tau-j} = \beta \delta t f(t^{\tau+1}, u^{\tau+1}), \tag{2.11}$$

where the superscript $\tau+1$ indicates the unknown solution at current time step, $\tau-j$ means the known solution from one of the previous time steps, δt is the size of time step (and is assumed constant for this description), p indicates the order of accuracy in time, α_j and β are parameters as defined in Table 2.1 up to $p=4$ [119]. It is proven by Hundsdorfer and Verwer in [119] that the BDFs of order $1-4$ are A-stable. Details of the family of BDF methods can also be found in [83, 105, 141].

p	β	α_0	α_1	α_2	α_3
1	1	-1			
2	$\frac{2}{3}$	$-\frac{4}{3}$	$\frac{1}{3}$		
3	$\frac{6}{11}$	$-\frac{18}{11}$	$\frac{9}{11}$	$-\frac{2}{11}$	
4	$\frac{12}{25}$	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$

Table 2.1: Coefficients α_j, β and order p up to $p=4$ for the family of BDF methods that is presented in Equation (2.11). The coefficients are presented with the assumption of constant size of time step δt.

It can be shown from Table 2.1 that when order $p=1$, values of α and β reduce the general form of BDF methods in Equation (2.11) to the backward Euler method in Equation (2.9). It is also clear from Table 2.1 that the higher order the method, the more previous time step solutions are needed.

The member of the family of BDF methods when $p=2$ is termed the BDF2. With second order of accuracy in time, this method is compatible with the second-order central finite difference scheme which is previously described in Section 2.1.1.1. An application of this BDF2 method can be found in [91]. This BDF2 method is extensively used in this thesis.

Having described spatial discretization with uniform grids and temporal discretization with fixed δt, in the following section, the concept of adaptivity is described, which plays a major role in the efficiency improvement of the discretization schemes.

2.2 Adaptivity

In the previous section, a number of spatial and temporal discretization schemes are described. Of these schemes, the FDM on cell-centred grids and the BDF2 method are extensively used in this thesis. In the following sections, the concept of adaptivity is described, which improves the discretization schemes in terms of their efficiency. To demonstrate the adaptivity, examples from using the FDM on cell-centred grids and the family of BDF methods are given.

2.2.1 Spatial Adaptivity

Considering the examples of the droplet and the phase-field models that are illustrated previously in Figures 1.2 and 1.3, observations and numerical experiments suggest that the finest mesh resolution (and the resulting heavy computation) are only needed in parts of the domain: areas around the moving contact line in the droplet model; and around the diffuse interfaces in the phase-field models. If uniform meshes (e.g. Figure 2.3(a)), which have the same level of mesh resolution everywhere, are employed in such situations, a lot of the finest grid computations, that are not in the areas of this need, will have little effect on the overall accuracy of the simulation.

The idea of spatial adaptivity then naturally emerges, which suggests a set of different mesh resolutions to be used in different regions of the domain. This concept is illustrated in Figure 2.3(b) where a 2-D cell-centred grid with three levels of mesh refinement is presented. The centre of the mesh is the area of the greatest resolution: the refinement level 3 has the equivalent mesh resolution of the uniform grid in Figure 2.3(a). From the comparison of the two sketches (a) and (b) in Figure 2.3, it suggests that the use of adaptive meshes can lead to a great improvement in terms of efficiency. Especially when there are small regions which require greater resolution (either to reduce the level of the local error to that the rest of the domain, or because a quantity of interest is most influenced by aspects of the solution in that region [217]). Some applications of spatial adaptivity can be found in [18, 23, 123, 130, 147, 223].

From the implementation point of view, one of the most common approaches is for the user to define a maximum and a minimum mesh resolution. The entire domain is covered

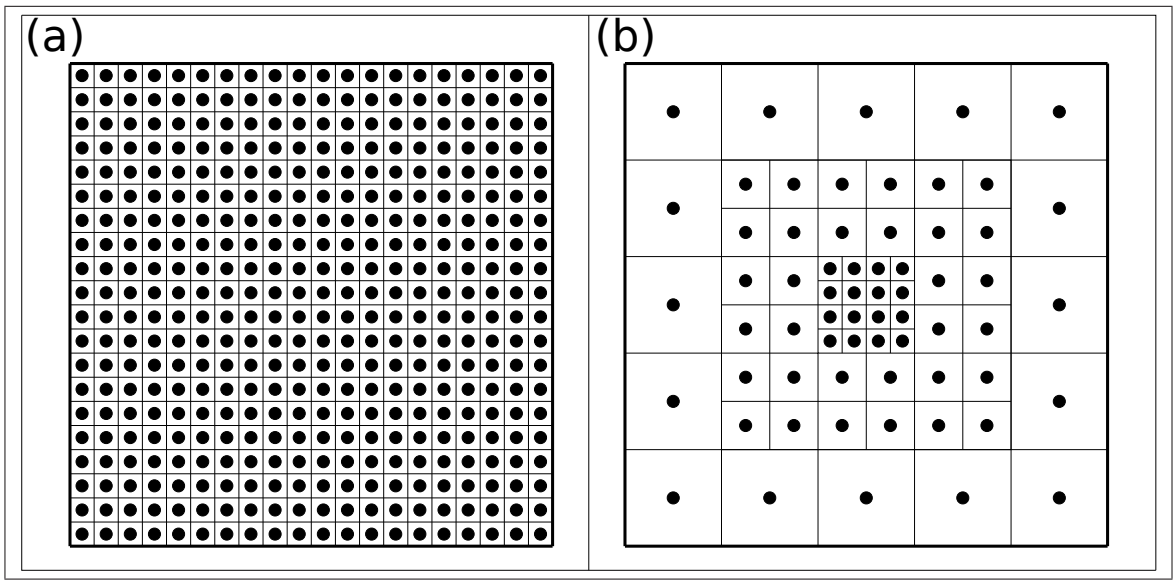

Figure 2.3: Sketch shows (a) a 2-D uniform cell-centred grid; (b) a 2-D cell-centred grid with three levels of mesh refinement, in which the finest refinement level has the equivalent mesh resolution of the uniform grid in (a).

by the coarsest mesh resolution, referred to here as refinement level 1, and the number of the level increments by one with each subsequent local refinement. The grid points that have refinements from them are called parents points, and others are called children points. Based upon problem-specific refinement criteria, regions where the greatest resolution is required are refined to have higher mesh level until the sufficient accuracy is obtained or the maximum resolution is reached. These refinement criteria may depend upon error estimation [44], the actual values of certain variables, the magnitude of the gradient of variable values [122], etc. The specific choices used in this thesis are described along with the detail of each model in later chapters. Algorithm 1 describes the main procedures of a point-wise adaptive mesh refinement routine, based upon a problem-specific user-defined refinement criterion. When the marking procedures end, a "Spatial adaptive routine" is called to actually perform the refining and/or coarsening (this routine is defined in Algorithm 2).

The introduction of variable mesh resolution means that structured finite difference approximations on uniform meshes cannot be used without careful consideration of the interfaces between mesh levels. For example, without further adjustment the five-point stencil scheme which is described in Equation (2.3) experiences difficulty if applied to those grid points that are at adjacent edges of refinement levels shown in Figure 2.3(b). In Chapter 3 Section 3.2.4, a specific approach which deals with the data exchange across the refinement edges is described and applied throughout this thesis.

Algorithm 1 Point-wise adaptive mesh refinement

```
1: for All children points, n do
2:     Call "Problem-specific user-defined refinement criterion" with n
3:     if n.criterion = refine and n.level != max level then
4:         n.refine = true
5:     else if n.criterion = coarsen and n.level != min level then
6:         Call "Problem-specific user-defined refinement criterion" with n.neighbour
7:         if n.neighbour.criterion = coarsen then
8:             n.coarsen = true
9:             n.neighbour.coarsen = true
10:        end if
11:    end if
12: end for
13: for Each children points, n do
14:    Call "Spatial adaptive routine" with n
15: end for
```

In Figure 2.3(b), the region which requires highest refinement lies in the centre of the mesh. Furthermore, when solving parabolic problems, the solution may evolve through time. In this case the region of the spatial domain that requires the highest refinement may also evolve, a dynamic adaptive technique is required. This requires refinement and coarsening processes to be carried out during the simulation, typically at the start of each time step. Algorithm 2 describes the main procedures for a point-wise spatial adaptive routine which is sequentially executed on all children points. This algorithm is also recursively performed if necessary, to preserve the mesh validity, which is to ensure that, on the structured Cartesian meshes, neighbouring regions have only one level of difference in refinement.

Algorithm 2 Point-wise spatial adaptive routine

```
1: Input parameter: n – a grid point
2: if n.refine = true then
3:     Refining n into four new grid points
4:     if More than one refinement level exists with n.neighbour then
5:         n.neighbour.refine = true
6:         Call 'Spatial adaptive routine' with n.neighbour
7:     end if
8: else if n.coarsen = true and n.neighbour.coarsen = true then
9:     if Resulting child point has no more than one refinement level between surround-
   ing children points then
10:        Coarsening n and its neighbours to obtain such new child point
11:    end if
12: end if
```

In the following subsections, the preservation of mesh validity, both the refinement and the coarsening processes are described in detail, along with interpolation and restriction operators which are used to transfer values between levels of refinement during the dynamic adaptive process.

2.2.1.1 Refinement and Interpolation

Once a sub-region of the domain is marked for refining, the refinement process separates each marked cell. In a 2-D situation, one marked cell is separated into four equal-space cells, and four new cell-centred children points are generated while the old point is marked as a parent point. Figure 2.4(a) shows this process in 2-D by separating a coarsest grid point (marked as $\bigtriangledown$) on a cell-centred grid, into four points (marked as $\circ$), and these four points are further refined into sixteen grid points (marked as $\bullet$). This procedure can also be carried out in a similar manner on a 3-D cell-centred grid. That is one grid point can be refined into eight new points, and if these eight points are refined again, the number of resulting children points is sixty-four (and in total seventy-two grid points from all three refinement levels in 3-D).

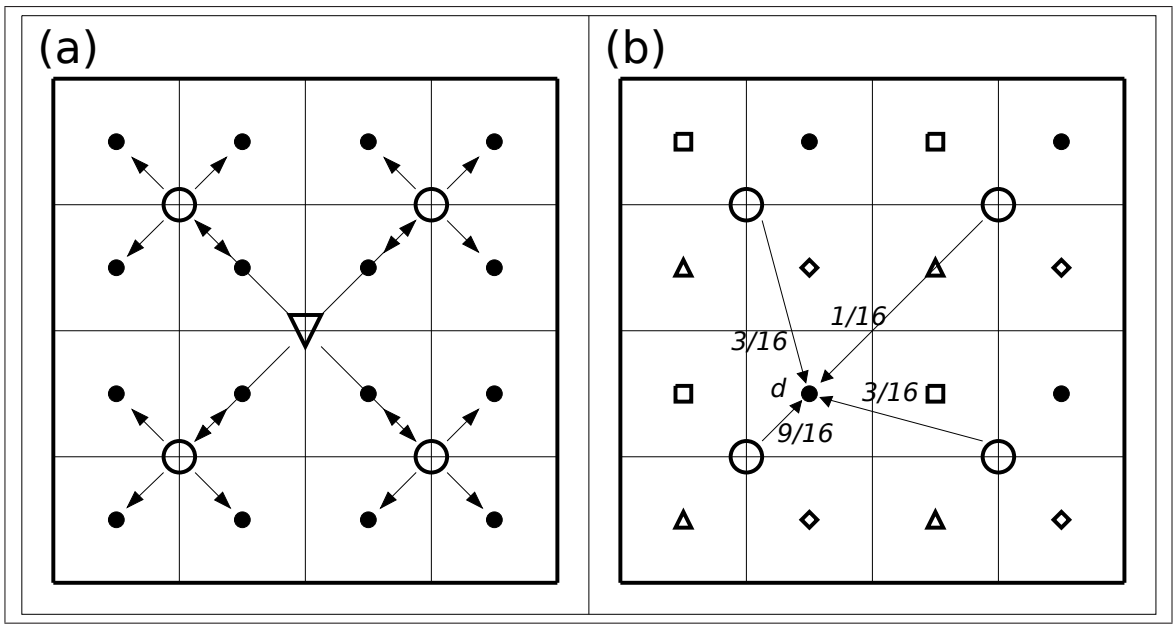

Figure 2.4: Sketch shows (a) the refinement process applied on a 2-D cell-centred point (marked as $\bigtriangledown$), which results in four new grid points (marked as $\circ$), and these four points are futher refined into sixteen grid points (marked as $\bullet$); (b) points on a fine cell-centred grid presented with symbols $\square$, $\bullet$, $\triangle$ and $\diamond$ are used to explain the bilinear interpolation in Equation (2.12) which transfer values from coarse grid points (marked as $\circ$) to the fine grid points.

During a simulation, this refinement process needs to be operated in a dynamic manner. Thus, an interpolation operator is required to transfer values from coarse grid points to the newly-generated fine grid points. This interpolation operator is written as I_{2h}^{h}, where h and $2h$ are the distances between two adjacent points on the fine and coarse refinement levels, respectively. A bilinear interpolation for cell-centred grid points from [216] is introduced here as an example. Figure 2.4(b) is used to help explaining this process, where symbols $\square$, $\bullet$, $\triangle$ and $\diamond$ are used to identify grid points on the fine level, and $\circ$ represents grid points that are on the coarse level. In addition, the amount of values taken from the surrounding coarse level grid points are identified as fractions at a particular grid point d in Figure 2.4(b). This bilinear interpolation is given by

$$I_{2h}^{h}u_{2h}(x,y) = \begin{cases} \frac{1}{16}\Big[9u_{2h}\left(x-\frac{h}{2},y-\frac{h}{2}\right)+3u_{2h}\left(x-\frac{h}{2},y+\frac{3h}{2}\right) & \\ +3u_{2h}\left(x+\frac{3h}{2},y-\frac{h}{2}\right)+u_{2h}\left(x+\frac{3h}{2},y+\frac{3h}{2}\right)\Big] & \text{for } \bullet; \\ \frac{1}{16}\Big[3u_{2h}\left(x-\frac{3h}{2},y-\frac{h}{2}\right)+u_{2h}\left(x-\frac{3h}{2},y+\frac{3h}{2}\right) & \\ +9u_{2h}\left(x+\frac{h}{2},y-\frac{h}{2}\right)+3u_{2h}\left(x+\frac{h}{2},y+\frac{3h}{2}\right)\Big] & \text{for } \square; \\ \frac{1}{16}\Big[3u_{2h}\left(x-\frac{h}{2},y-\frac{3h}{2}\right)+9u_{2h}\left(x-\frac{h}{2},y+\frac{h}{2}\right) & \\ +u_{2h}\left(x+\frac{3h}{2},y-\frac{3h}{2}\right)+3u_{2h}\left(x+\frac{3h}{2},y+\frac{h}{2}\right)\Big] & \text{for } \diamond; \\ \frac{1}{16}\Big[u_{2h}\left(x-\frac{3h}{2},y-\frac{3h}{2}\right)+3u_{2h}\left(x-\frac{3h}{2},y+\frac{h}{2}\right) & \\ +3u_{2h}\left(x+\frac{h}{2},y-\frac{3h}{2}\right)+9u_{2h}\left(x+\frac{h}{2},y+\frac{h}{2}\right)\Big] & \text{for } \triangle, \end{cases} \tag{2.12}$$

where the array u stores values at the grid points, and (x,y) are the Cartesian coordinates of the grid points. In 3-D cell-centred grids, the generalization of the bilinear interpolation is called trilinear interpolation. Both of the 2-D bilinear and 3-D trilinear interpolations are $\mathscr{O}(h^2)$ accurate. A description of the trilinear interpolation can be found in [216], and an example can be found in [161]. The interpolation operators with higher-order accuracy generally derived from multi-linear schemes, the fundamental difference to the second-order interpolation is that, the multi-linear interpolations extends their stencils, so a greater number of coarse grid points are considered, see for example [136].

In the following section, the coarsening process and a corresponding restriction operator are introduced.

2.2.1.2 Coarsening and Restriction

The coarsening process preforms a reverse operation to the previously described refinement process. It is shown in Figure 2.5, where a group of four cell-centred points on the

fine level of refinement reduce to one point on the coarse level of refinement. In order to preserve the validity of the mesh, for the structured cell-centred grids that are used in this thesis, to obtain one grid point on the coarser level of refinement, a group of four grid points are required from the finer level of refinement in 2-D (and a group of eight grid points from the finer level of refinement are needed in 3-D). All of points in a group should be marked for coarsening by the refinement criterion, in order to carry out the coarsening operator.

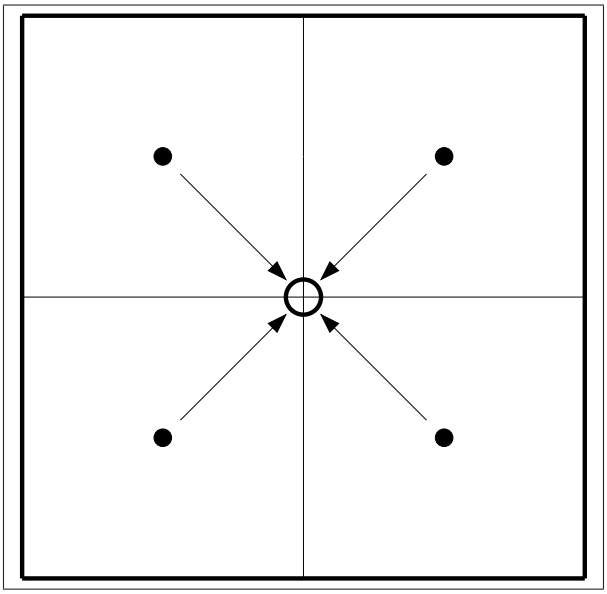

Figure 2.5: Sketch shows the coarsening process on a 2-D cell-centred grid, which reduces a group of four points (marked as •) on the fine level of refinement to one point (marked as ○) on the coarse level of refinement.

The coarsening process is operated dynamically during the simulation, a restriction operator is required for these newly-generated points on the coarse level of refinement. This restriction operator is written as I_h^{2h}, where h and $2h$ are the distances between two adjacent points on the fine and coarse refinement levels, respectively. A simple four-point averaging approach from [216] can be applied. It is given by

$$\begin{aligned} I_h^{2h} u_h(x,y) = & \\ & \frac{1}{4}\left[u_h\left(x-\frac{h}{2},y-\frac{h}{2}\right) + u_h\left(x-\frac{h}{2},y+\frac{h}{2}\right) \right. \\ & \left. u_h\left(x+\frac{h}{2},y-\frac{h}{2}\right) + u_h\left(x+\frac{h}{2},y+\frac{h}{2}\right) \right], \end{aligned} \tag{2.13}$$

where the array u stores values of grid points, and (x,y) are the Cartesian coordinates of

the grid points. For the restriction on 3-D cell-centred grids, the equivalent eight-point averaging approach can be used [216]. Both averaging approaches are $\mathcal{O}(h^2)$ accurate. The restrictions with higher-order accuracy can be derived from multi-linear schemes, where more fine grid points are considered by the operator.

In the following section, adaptivity in the temporal discretization schemes is described.

2.2.2 Temporal Adaptivity

The temporal discretization schemes (i.e. BDF methods) from Equation (2.11) and Table 2.1 are described with an assumption of a fixed time step size δt. For temporal adaptivity to appear, variable time step sizes δt^n are used [1]. In this section, temporal adaptivity with the BDF2 method is explained in detail, since it is applied in this thesis.

During the simulation, a user-defined, problem-specific criterion may be used to determine for each time step, if the $\delta t^{\tau+1}$ should be increased, decreased or kept the same compared to δt^{τ}, providing the solution is converging at the current time step. There is another circumstance where $\delta t^{\tau+1}$ is too large for the solution to converge at a given time step. Then this time step needs to be reset and computed again with a smaller $\delta t^{\tau+1}$. Authors of [81] suggested a variable amount for adaptive $\delta t^{\tau+1}$ to change between time steps, based upon the error estimate. Examples of using a local error estimate for the criterion can also be found in [90, 189]. The criterion used in this thesis is very similar to [91], where the convergence rate of the implicit solver is used, as a local indicator of time step selection, the detail of which are explained in Chapter 3 Section 3.3.3.

The second-order BDF2 method for variable time step sizes is presented in [69, 119] for the ODE in Equation (2.7), and it is given by

$$u^{\tau+1} - \left(\frac{(r^{BDF}+1)^2}{1+2r^{BDF}} u^{\tau} - \frac{\left(r^{BDF}\right)^2}{1+2r^{BDF}} u^{\tau-1} \right) = \frac{1+r^{BDF}}{1+2r^{BDF}} \delta t^{\tau} f(t^{\tau+1}, u^{\tau+1}), \qquad (2.14)$$

where ratio $r^{BDF} = \delta t^{\tau} / \delta t^{\tau-1}$, superscripts $\tau+1$, τ and $\tau-1$ indicate the current, previous and the one-before-previous time steps, respectively. Applications of the BDF2 method with variable time step sizes can be found in [90, 91]. The choices of variable $\delta t^{\tau+1}$ may have effects on the stability of the scheme. Author of [69] suggested that, for semi-linear parabolic problems, the ratio of r^{BDF} is restricted to be less than 1.91 for maintaining the stability of the BDF2 methods. Additional information can be found in [168].

[1] Superscript $n = 1, 2, \ldots, \tau+1$ indicates the number of time steps.

Having described spatial and temporal discretization schemes and their adaptivity, in the next section, several common solution methods are briefly introduced, in order to solve the fully-discrete systems that arise at each time step of an implicit scheme.

2.3 Common Solution Methods for Algebraic Systems Arising from Local Discretizations of Partial Differential Equations

Having explained the derivation of the discrete algebraic systems arising from using spatial (and temporal) discretization schemes for elliptic (and parabolic) PDEs, in this section, several common solution methods are briefly introduced for these systems. In Section 2.3.1, solution methods for linear problems are described, including direct methods, iterative methods and technique of preconditioning. In Section 2.3.2, solution methods for nonlinear problems are discussed, including Newton's method and nonlinear iterative methods.

2.3.1 Solution Methods for Linear Partial Differential Equations

Let's consider a simple linear elliptic PDE, such as Poisson's equation given as

$$\triangle u = f, \tag{2.15}$$

where f is a known function that is independent of u. For simplicity, initially Equation (2.15) is considered in 1-D. A set of 1-D cell-centred grid points is defined as

$$Z_h = \left\{ x : x = \left(i - \frac{1}{2} \right) h \quad i = 0, ..., N-1 \right\}, \tag{2.16}$$

where the elements of Z_h are the Cartesian spatial coordinates. N is previously defined as the number of grid points in each coordinate direction, so in this 1-D case, N is the total number of grid points. Let n represent the internal grid points in each coordinate direction. In other words, n is the number of unknowns for a standard finite difference discretization of Equation (2.15) with Dirichlet boundary conditions, and $h = 1/(N-1)$ is the equal distance between each grid point. On this 1-D Cartesian grid, there are two boundary points: $x_{i=0}$ and $x_{i=N}$. It is assumed for simplicity that Dirichlet boundary condition have been applied, so $u(x_{i=0})$ and $u(x_{i=N})$ are specified. For all internal points, the 1-D central finite difference approximation using a three-point stencil shown in Equation (2.17) can

be used (cf. the 2-D five-point stencil in Equation (2.3)):

$$\triangle_h u_h = \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}. \tag{2.17}$$

When discretized by this three-point scheme, Equation (2.15) can be written in the following form:

$$Au = b, \tag{2.18}$$

where $u \in \mathbb{R}^n$ is the vector of unknowns, $b \in \mathbb{R}^n$ is the right-hand side vector, and $A \in \mathbb{R}^{n \times n}$ is a coefficient matrix. Moreover, A is a sparse matrix, because the three-point stencil that is shown in Equation (2.17), is based on local approximations, where the dependence of updating a grid point is limited to its local neighbours. In a sparse matrix, the number of non-zero elements per row is only dependent upon the stencil, and is independent of h or n. Thus the number of non-zero elements is $\mathcal{O}(n)$ as $n \to \infty$.

Let's consider the example PDE shown in Equation (2.15) in a 2-D situation, with a set of 2-D cell-centred grid points defined as Equation (2.5) (although Equation (2.5) defines a 3-D set of grid points, it is straightforward to define a set of 2-D points from the definition), where $N = 6$ and therefore $n = 4$. This 6×6 2-D cell-centred grid is shown in Figure 2.6.

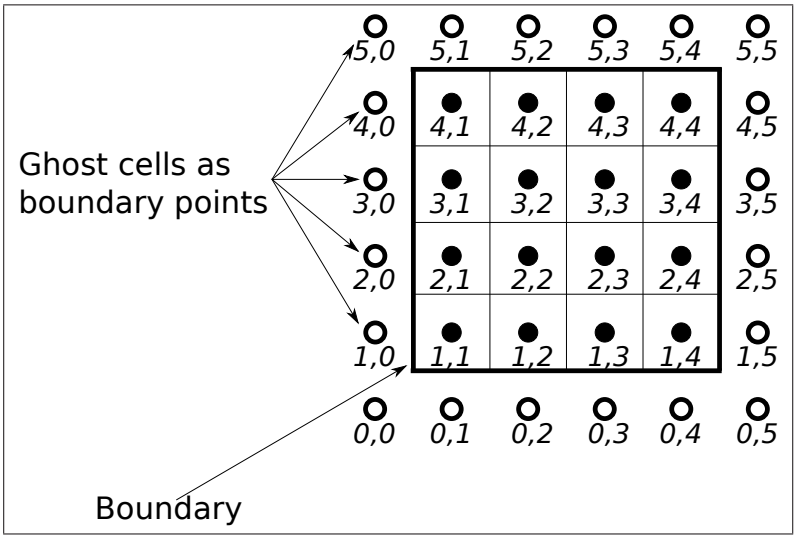

Figure 2.6: Sketch shows a 6×6 2-D cell-centred Cartesian grid, $\circ$ represents 20 ghost cells that are used as boundary points, $\bullet$ represents 16 unknowns, $i = 0, \ldots, 5$ is the number of rows and $j = 0, \ldots, 5$ is the number of columns for each grid points (i, j).

Equation (2.15) is then discretized by the standard five-point stencil from Equation (2.3) on this given grid. It is assumed the values of boundary points are known and can be re-arranged into the RHS vector b. The resulting $Au = b$ for the 16 unknowns ($u_{1,1}$ to

$u_{4,4}$) can be written as

$$-\frac{1}{h^2}\begin{pmatrix}
4 & -1 & & & -1 & & & & & & & & & & & \\
-1 & 4 & -1 & & & -1 & & & & & & & & & & \\
 & -1 & 4 & -1 & & & -1 & & & & & & & & & \\
 & & -1 & 4 & & & & -1 & & & & & & & & \\
-1 & & & & 4 & -1 & & & -1 & & & & & & & \\
 & -1 & & & -1 & 4 & -1 & & & -1 & & & & & & \\
 & & -1 & & & -1 & 4 & -1 & & & -1 & & & & & \\
 & & & -1 & & & -1 & 4 & & & & -1 & & & & \\
 & & & & -1 & & & & 4 & -1 & & & -1 & & & \\
 & & & & & -1 & & & -1 & 4 & -1 & & & -1 & & \\
 & & & & & & -1 & & & -1 & 4 & -1 & & & -1 & \\
 & & & & & & & -1 & & & -1 & 4 & & & & -1 \\
 & & & & & & & & -1 & & & & 4 & -1 & & \\
 & & & & & & & & & -1 & & & -1 & 4 & -1 & \\
 & & & & & & & & & & -1 & & & -1 & 4 & -1 \\
 & & & & & & & & & & & -1 & & & -1 & 4
\end{pmatrix}\begin{pmatrix} u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ u_{1,4} \\ u_{2,1} \\ u_{2,2} \\ u_{2,3} \\ u_{2,4} \\ u_{3,1} \\ u_{3,2} \\ u_{3,3} \\ u_{3,4} \\ u_{4,1} \\ u_{4,2} \\ u_{4,3} \\ u_{4,4} \end{pmatrix} = \begin{pmatrix} b_{1,1} \\ b_{1,2} \\ b_{1,3} \\ b_{1,4} \\ b_{2,1} \\ b_{2,2} \\ b_{2,3} \\ b_{2,4} \\ b_{3,1} \\ b_{3,2} \\ b_{3,3} \\ b_{3,4} \\ b_{4,1} \\ b_{4,2} \\ b_{4,3} \\ b_{4,4} \end{pmatrix}. \quad (2.19)$$

Only non-zero elements are presented in the coefficient matrix A. If a 3-D situation is considered, the use of the corresponding seven-point stencil on Equation (2.15) is straightforward.

When a very fine grid is used, and especially for problems in higher dimensions, this coefficient matrix A can become very large. Thus, before solving the linear system, this large but sparse matrix needs to be stored via the use of some efficient storage schemes. The simplest storage scheme is the so-called coordinate format [89]. To store a $n \times n$ matrix, it uses three arrays, each with size of the number of non-zero elements. They are, firstly a "value" array which contains all the real non-zero values of the matrix in any order; secondly, an integer "row" array which contains the corresponding row indices of these values; and finally an integer "column" array which contains the corresponding column indices of the values.

This coordinate format can be further improved by compressing the "row" array. Thus it only contains a position which indicates the range of each row within the "value" array. This requires the "value" array to be modified, so that it stores the non-zero values row by row, from row 1 to n. This is called the compressed sparse row format, and an equivalent scheme which compresses the "column" array instead, is called the compressed sparse column format [89]. There are other storage schemes that exist, for example the modified sparse row format, which exploits the fact that the diagonal elements of common matrices are usually non-zero, and are often required (therefore are accessed without indirect addressing). Detailed descriptions of these storage schemes can be found in [192], and an application which describes several storage schemes in a parallel environment can be found in [99].

The common solution methods to solve a sparse linear system fall into two general categories: direct methods and iterative methods. The former, in the absence of rounding

errors, may provide the true solution (to the fully-discrete system) after a finite number of computations. On the other hand, the latter aims to move closer towards the true solution after each iteration from a provided initial guess. Typically, there is no attempt to gain the true solution when using iterative methods, instead, an acceptable approximation is obtained.

Gaussian elimination via LU factorisation is arguably the most widely used direct method for solving general linear systems. This may be modified to work on sparse systems, however, before this is described, an important concept of matrix structure which is preferred by the direct methods (reasons are discussed later), needs to be explained. The desired matrix structure is for all non-zero elements to gather around the diagonal. This type of matrix is termed a banded matrix, and can be defined as the following:

$$a_{ij} \neq 0 \text{ only if } i - m_l \leq j \leq i + m_u, \text{where } m_l, m_u \geq 0,$$

where a_{ij} are the elements of a banded matrix A. The number $m_l + m_u + 1$ is called the bandwidth. Matrices with larger bandwidth typically occur from the discretizations of PDEs in higher dimensions and/or from using larger stencils. For example, using the three-point stencil in Equation (2.17) leads to a matrix with a bandwidth of 3. It is often called a tridiagonal matrix. The sparse matrix A from discretising Equation (2.15) in 2-D using the standard five-point stencil, as shown in Equation (2.19), has the bandwidth of 9. The bandwidth increases as the grid becomes finer. This is because both upper and lower diagonals of non-zero elements become further away from the leading diagonal. With the row-by-row ordering shown in Figure 2.6 and Equation (2.19), a 2-D matrix has a bandwidth of $2n^{1/2} + 1$. In addition, if a seven-point stencil is used in 3-D, the bandwidth of a sparse matrix is $2n^{2/3} + 1$.

The reason for this interest in the bandwidth of a matrix is because, during Gaussian elimination, many original zero elements may become non-zero. This is termed fill-in, and the bandwidth gives an upper bound on the number of zero elements that can become non-zero during the Gaussian elimination process. In some extreme cases, a sparse matrix that is not banded can turn into a dense matrix because of the fill-in. Therefore, generally the first step of applying a sparse direct solve (e.g. Gaussian elimination with LU factorisation) is pre-ordering using an algorithm that aims to minimize fill-in, this process is also called permutation. To name a few examples, the band method, envelope method and Cuthill-McKee algorithm can be used as a pre-ordering algorithm. The interested reader is directed to [54, 72, 84] for more information.

Actually pre-ordering to minimize fill-in is sufficient for symmetric, positive-definite

matrices (this type of matrix is described in detail later). However, it is not quite sufficient for general sparse systems. This is because the Gaussian elimination without pivoting is known to be numerically unstable in general. Hence the ordering may have to be changed again during the elimination process, based upon the sizes of the numerical values that appear in the pivot position. This is an additional complication that has to be managed by algorithms such as SuperLU [210].

Having obtained a well-ordered matrix A, it can be rewritten in terms of lower and upper triangular component matrices, so that $A = LU$. Two systems can then be created from separating Equation (2.18); they are

$$Lz = b, \tag{2.20}$$

$$Uu = z. \tag{2.21}$$

Firstly Equation (2.20) is solved by a forward substitution for the vector z, then Equation (2.21) can be solved by a backward substitution for the solution u. The implementation of Equations (2.20) and (2.21) needs to take advantage of using the sparse matrices, L and U. There exists general purpose software for solving large, sparse systems of linear equations using such direct methods, for example, SuperLU [210] and MUMPS [169]. Applications of these software packages (and further references) can be found in [10, 151].

There are other direct methods that exist for less general sparse systems. For example, the frontal method, which is extended from the band method. This method can perform well as long as the matrix bandwidth is relatively small and is commonly used with applications of the FEM with very fine structured meshes. An application of the frontal method can be found in [66] where detailed programming code is given. The interested reader is directed to [64, 89, 154] for more information about the direct methods.

For the demonstrated 1-D problem with the use of three-point stencil, the described LU factorisation only requires $\mathscr{O}(n)$ computations. However, for the problems with sparse matrices in higher dimensions, the direct methods generally requires $\mathscr{O}(n^2)$ computations. This becomes excessively expensive in practice when n is very large (i.e. having a very fine grid).

An alternative to the direct methods is to use iterative methods for the solution of sparse linear systems. One of the simplest iterative methods for the linear system that is presented in Equation (2.18) is the Jacobi iteration. In order to update each unknown u_i,

a point-wise Jacobi iteration is given by [89]

$$u_i^{l+1} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} u_j^l - \sum_{j=i+1}^{n} a_{ij} u_j^l \right) \Big/ a_{ii}, \tag{2.22}$$

where a_{ij} are the elements of coefficient matrix A, b_i are the elements of the right-hand side vector, and superscripts $l+1$ and l indicate values from the current and previous iterations, respectively. Note elements of a_{ii} belong to the diagonal of the matrix A. Therefore the Jacobi iteration only works on non-zero diagonal matrices. In fact, it only converges for diagonally-dominant matrices [34].

The Jacobi iteration only uses the values from the previous iteration. Another iterative method, the Gauss-Seidel iteration, improves this by taking the most up-to-date values of u. Therefore, a point-wise Gauss-Seidel iteration has the following form:

$$u_i^{l+1} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} u_j^{l+1} - \sum_{j=i+1}^{n} a_{ij} u_j^l \right) \Big/ a_{ii}. \tag{2.23}$$

In general, because of using latest values, the Gauss-Seidel iteration converges faster than the Jacobi iteration. However, it shares the same requirement on the matrix structure (i.e. it only works on matrices that are diagonally-dominant).

There is another alternative to the Jacobi and Gauss-Seidel iterations. This is based upon a parameterized splitting, and can be seen as an extension to the Gauss-Seidel iteration. It is called the method of successive over-relaxation (SOR), and is based upon ω, a weighted parameter that takes values strictly in the range of 0 to 2. A point-wise SOR method has the following form:

$$u_i^{l+1} = \omega \left[\left(b_i - \sum_{j=1}^{i-1} a_{ij} u_j^{l+1} - \sum_{j=i+1}^{n} a_{ij} u_j^l \right) \Big/ a_{ii} \right] + (1-\omega) u_i^l. \tag{2.24}$$

Note if $\omega = 1$, the SOR method turns back to the Gauss-Seidel iteration. When $0 < \omega < 1$, it can be seen as a "conservative" relaxation, whereas for $\omega > 1$, it is over-relaxation. Typically a value in the range of $(1,2)$ is selected to accelerate the convergence of the Gauss-Seidel iteration.

There is one important feature which is shared by these three iterative methods, that is the so-called "smoothing property". This feature describes that these iterative methods (with an appropriate choice of ω in case of SOR) are able to reduce the higher frequency error components very quickly (generally in a few iterations), and it is well known they are less effective against lower frequency error components, the error being the differ-

ence between the true solution and the approximate solution. The high frequency error components are defined as the highest frequency oscillations that are representable on the given grid (with grid spacing h). This "smoothing property" is exploited by the multigrid methods that are described in Section 2.4.

For general sparse matrices arising from low order PDE discretizations, Jacobi, Gauss-Seidel and SOR methods typically require $\mathscr{O}(n^2)$ computations to converge (i.e. $\mathscr{O}(n)$ iterations at a cost of $\mathscr{O}(n)$ per iteration). Detailed descriptions of these three methods can be found in [103, 104, 228], and applications of these in a parallel environment can be found in [11, 126].

There are other types of iterative methods, many of which solve the linear system by minimising the residual (which is shown in Equation (2.27)) over larger and larger subspaces of $\mathbb{R}^n$. Before describing the methods, let's consider a minimization problem with a quadratic form. The general quadratic form can be written as the following:

$$f(u) = \frac{1}{2}u^T Au - b^T u + c, \tag{2.25}$$

where $A \in \mathbb{R}^{n\times n}$ is a symmetric (i.e. the transpose of A is equal to A) and positive-definite (i.e. for every non-zero vector vec, $vec^T Avec > 0$) matrix, u and b are two vectors in $\mathbb{R}^n$, and c is a scalar constant ($vec^T w$ represents the inner product of two vectors vec and w). The gradient of function f in Equation (2.25) with respect of u can be obtained as the following:

$$f'(u) = Au - b, \tag{2.26}$$

where $f'(u) \in \mathbb{R}^n$ represents the gradient of the quadratic function $f(u)$. If a solution u minimises the gradient, so that $f'(u) = 0$, then this solution u is also the solution of the minimization problem in Equation (2.25).

The gradient shows the direction of the greatest increase of $f(u)$. In order to find a solution which minimises $f(u)$, the opposite direction of the gradient is used as a descent direction, this is called the residual and is defined by

$$r = -f'(u) = b - Au. \tag{2.27}$$

Here a specific iterative method which minimises the residual is described. It is called the steepest descent method, and is an iterative method. Like the three iterative methods described earlier, the steepest descent method also requires a series of iterations, such as $u^{l=1}$, $u^{l=2}$, etc., where l is the number of iteration and $u^{l=0}$ is a known initial guess. The method stops when the approximate solution satisfies a pre-defined stopping criterion.

By using the residual as described above, the steepest descent method updates the solution u as the following:

$$u^{l+1} = u^l + \alpha^l r^l, \tag{2.28}$$

where the value of the scalar α^l determines the distance taken along the direction of the residual within the iteration l. α^l minimises $f(u^l)$ when the derivative $\frac{d}{d\alpha} f(u^l) = 0$. This leads to r^{l-1} and $f'(u^l)$ being orthogonal, i.e. $f'(u^l)^T r^{l-1} = 0$. In addition, since $f'(u^l) = -r(u^l)$ from Equation (2.27), α^l can be defined as

$$\alpha^l = \frac{r^{lT} r^l}{r^{lT} A r^l}. \tag{2.29}$$

This expression for α^l only holds if the matrix A is symmetric and positive-definite.

For general problems with symmetric and positive-definite matrices, the complexity of the steepest descent method is still $\mathcal{O}(n^2)$, if it is assumed $\mathcal{O}(n)$ iterations are required to converge. Details about this method can be found in [89, 192] and some applications can be found in [16, 162].

Note that because of $f'(u^l)^T r^{l-1} = 0$ in the steepest descent method, the new search direction is always orthogonal to the previous one. This creates these so-called "zigzag" routes, which often repeat the same direction as earlier ones. It can be improved by making the new search direction A-orthogonal to the previous one. Considering two residual vectors r^l and r^{l+1}, which also indicate the search direction, they are A-orthogonal (also are called A-conjugate) if

$$r^{lT} A r^{l+1} = 0. \tag{2.30}$$

The method that exploits this orthogonality, and uses it to improve the steepest descent method, is called the Conjugate Gradient (CG) method. An important feature in the CG method is the set of search direction, which is defined as the following:

$$K^l = \text{span}\{r^{l=0}, A r^{l=0}, A^2 r^{l=0}, \ldots, A^{l-1} r^{l=0}\}, \tag{2.31}$$

where superscript l denotes the number of iterations, and $r^{l=0}$ is the residual calculated from using the initial guess $u^{l=0}$. This set is called a Krylov subspace. Each new subspace K^{l+1} is formed from the union of the previous subspace K^l and the subspace AK^l. Additionally, because AK^l is included in K^{l+1}, the next residual r^{l+1} can be proven to be A-orthogonal to K^l. Thus, stepping through the subspaces K^l, the CG method can be used as an iterative method to minimise the residual r.

The CG method was first introduced in [113]. Details and analyses of the CG method

can be found in [112, 170]. There are many CG-related methods, and one example can be found in [33]. Like the steepest descent method, the CG method only works on matrices that are symmetric and positive-definite. A method which is called Minimal Residual Method (MINRES) extends the idea, so it works on symmetric and indefinite matrices (matrix A is an indefinite matrix, if there are two vectors *vec* and w, such that $vec^T Avec > 0 > w^T Aw$). A discussion comparing the CG method and MINRES can be found in [140]. For problems with non-symmetric matrices, the Generalized Minimum Residual Method (GMRES) is often used. The CG method, the MINRES and the GMRES all belong to the family of Krylov subspace methods. However, the described Krylov subspace methods have the property of converging more and more slowly (for problems arising from discretizations of PDEs) as $n \to \infty$ (i.e. $h \to 0$). More details about the Krylov subspace methods can be found in [64, 152, 192, 193].

The rates of convergence of the described Krylov methods can be further improved. This is done by using a technique which is called preconditioning. Considering the linear problem $Au = b$, the aim of the preconditioning is to find a nonsingular matrix M, so that, $M^{-1}Au = M^{-1}b$ has the same solution as the original problem but is much easier to solve [89, 192]. Let's use the CG method with a symmetric and positive-definite matrix A as an example. The convergence of the CG method is well understood and is based on the condition number $\kappa(A)$ of the coefficient matrix A. The condition number is the ratio of the matrix's largest and smallest eigenvalues. Because of the assumption that A is a symmetric and positive-definite matrix, its eigenvalues are positive real numbers. Theorem 10.2.6 from Golub and Van Loan [88, pg.530] describes the convergence of the CG method, which is given as the following. Note the Algorithm 10.2.1 mentioned in this Theorem is from [88, pg.527], and is the algorithm for the CG method.

Theorem 1. (from Golub and Van Loan [88, pg.530]) *Suppose* $A \in \mathbb{R}^{n\times n}$ *is symmetric positive definite and* $b \in \mathbb{R}^n$. *If Algorithm* 10.2.1 *produces iterates* $\{x_k\}$ *and* $\kappa = \kappa_2(A)$ *then*

$$||x - x_k||_A \leq 2||x - x_0||_A \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k.$$

Proof. See [156, pg.187] from Luenberger. □

$\kappa_2(A)$ in the presented theorem above is the spectral condition number measured in two norm. Therefore, if through the use of preconditioning, a matrix M can be found, so that $\kappa(M^{-1}A) \approx 1$, then the CG method may have an optimal convergence. The matrix M is called a preconditioner. If $\kappa(M^{-1}A) \approx 1$ is not achievable, then using the preconditioning to reduce the spectral range of the eigenvalues can also improve the rate of convergence.

For more general matrices (e.g. indefinite and non-symmetric) and their solvers (e.g. MINRES and GMRES), although they are less well understood, the use of preconditioning still can improve their rates of convergence. This has been seen in practice, and the interested reader is directed to [22, 89] for more information. There are many algorithms that can be used to find such preconditioner. To name a few examples, incomplete LU factorization and sparse approximate inverse are commonly applied. Their applications can be found in [45, 49].

It is worth noting that in this section, a single linear elliptic PDE is chosen for the purpose of demonstration. For a linear parabolic PDE which is time-dependent, it may also be written in the form of $Au = b$ for the problem arising from each time step by using the implicit temporal discretization schemes that are described in Section 2.1.2. In addition, these described solution methods may be used to solve this $Au = b$ system from each time step.

Having described some common methods and techniques for the linear PDEs, in the following section, solution methods for nonlinear PDEs are introduced.

2.3.2 Solution Methods for Nonlinear Partial Differential Equations

For solving a nonlinear algebraic system arising from the discretization of a nonlinear PDE, a nonlinear version of the Jacobi iteration can be derived. In order to describe this method, a single equation which is a nonlinear elliptic boundary value problem (e.g. the Poisson equation with a nonlinear source term for the sake of argument) is considered here. When discretized, the equation can be written as the following algebraic system:

$$A(u) = f, \tag{2.32}$$

where A, instead of being a single matrix as in the linear case, is a vector-valued nonlinear function that takes u as input. Hence u is the solution for nonlinear problem and f is a vector of RHS values (independent of u). Equation (2.32) has the following equivalent form:

$$\mathscr{F}(u) = 0, \text{ where } \mathscr{F}(u) = A(u) - f. \tag{2.33}$$

A commonly used nonlinear Jacobi iteration which is described in [31] is applied. A solution u_i (generally this solution is an approximation to the true solution) is obtained as follows

$$u_i^{l+1} = u_i^l - \frac{\mathscr{F}_i(u^l)}{(\mathscr{F}_i)'(u^l)}, \tag{2.34}$$

where the superscripts $l+1$ and l are the current and previous iterations of the nonlinear Jacobi sweeps, respectively, $(\mathscr{F}_i)'(u^l)$ is the first derivative of $\mathscr{F}_i(u^l)$ with respect to u_i^l and $u^{l=0}$ is an initial guess. Note that this update must be applied in turn for each unknown component of u_i^l, and Equation (2.33) represents the nonlinear Jacobi form. The nonlinear Gauss-Seidel form always uses the most up-to-date components of u in the evaluation of the quotient on the right-hand side.

Having described the nonlinear Jacobi iteration for solving the discretization of a single nonlinear elliptic PDE, it is worth noting that the nonlinear Jacobi iteration can also be applied in a block-wise manner as well as a point-wise manner. Hence for solving the algebraic system arising from discretization of a nonlinear system of PDEs, we may update multiple unknowns corresponding to the same grid point simultaneously within one nonlinear Jacobi iteration. Typically, all variables associated with one grid point are grouped together and are updated one grid point at a time. The reason for using the nonlinear block Jacobi approach for solving such systems is that it is typically much more robust than a point-wise iteration and so, in the context of time-dependent PDEs, typically allows much larger time steps to be selected (and still lead to a convergent iteration).

This block Jacobi method is based upon Newton's method (which is described later in this section), for the small nonlinear system corresponding to the unknowns on each grid point. By approximately solving this small system as a whole, with all unknowns at all other grid points frozen, all variables at the "visited" grid point can be updated simultaneously.

In order to demonstrate the use of this nonlinear block Jacobi method, consider a finite difference discretization of a system of elliptic nonlinear PDEs: $\mathscr{F}(u) = 0$. Let $u_{i,k}$ be the solution on grid point i for unknown variable k, where we assume $\mathscr{K}$ unknowns at each grid point i. The system $\mathscr{F}(u) = 0$ is made up of $n \times \mathscr{K}$ coupled nonlinear algebraic equations,

$$\mathscr{F}_{i,k}(u_{i,k}) = 0, \tag{2.35}$$

where $u_i \in \mathbb{R}^{\mathscr{K}}$ and $\mathscr{F}_i \in \mathbb{R}^{\mathscr{K}}$, and for $i = 1,\ldots,n$ and $k = 1,\ldots,\mathscr{K}$ (to clarify the notation $u_{i,k}$ is the k^{th} component of $u_i \in \mathbb{R}^{\mathscr{K}}$ and $\mathscr{F}_{i,k}$ is the k^{th} component of $\mathscr{F}_i \in \mathbb{R}^{\mathscr{K}}$). On one grid point i, all $\mathscr{K}$ variables may be updated simultaneously as

$$u_i^{l+1} = u_i^l - C_i^{-1}\mathscr{F}_i(u_i^l), \tag{2.36}$$

where C_i^{-1} is the inverse of the $\mathscr{K} \times \mathscr{K}$ Jacobian matrix C_i, which is given as

$$C_i = \begin{pmatrix} \frac{\partial \mathscr{F}_{i,1}}{\partial u_{i,1}} & \frac{\partial \mathscr{F}_{i,1}}{\partial u_{i,2}} & \cdots & \frac{\partial \mathscr{F}_{i,1}}{\partial u_{i,K}} \\ \frac{\partial \mathscr{F}_{i,2}}{\partial u_{i,1}} & \frac{\partial \mathscr{F}_{i,2}}{\partial u_{i,2}} & \cdots & \frac{\partial \mathscr{F}_{i,2}}{\partial u_{i,K}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathscr{F}_{i,K}}{\partial u_{i,1}} & \frac{\partial \mathscr{F}_{i,K}}{\partial u_{i,2}} & \cdots & \frac{\partial \mathscr{F}_{i,K}}{\partial u_{i,K}} \end{pmatrix}. \tag{2.37}$$

The Gauss-Seidel form of this method is straightforward as it uses the most up-to-date values of u on the RHS of Equation (2.36), rather than only using u^l.

The described point-wise and block Jacobi methods are so-called "local relaxation-type" methods which perform local linearization at each grid point. On the other hand, the well-known Newton's method applies a global linearization of the nonlinear problem. Let's consider the same general form of the discrete nonlinear system arising from application of FDM for example, to a system of nonlinear elliptic PDEs: $\mathscr{F}(u) = 0$. Again, let $u_{i,k}$ be the solution on grid point i for each variable k. Then u^l can be updated in the following iterative manner,

$$u^{l+1} = u^l - J^{-1}\mathscr{F}(u^l), \tag{2.38}$$

where J is the full Jacobian matrix. Let $\delta u = u^{l+1} - u^l$ then Equation (2.38) becomes

$$\delta u = -J^{-1}\mathscr{F}(u^l), \tag{2.39}$$

which leads to the equation

$$J\delta u = -\mathscr{F}(u^l). \tag{2.40}$$

It is worth noting that the Jacobian matrix J is much larger than the local C_i matrices which appeared for each grid point i in the nonlinear block Jacobi method. Considering there are $\mathscr{K}$ variables and n unknowns in the system, J is a $n\mathscr{K} \times n\mathscr{K}$ matrix, and can be defined as

$$J = \begin{pmatrix} \frac{\partial \mathscr{F}_{1,1}}{\partial u_{1,1}} & \cdots & \frac{\partial \mathscr{F}_{1,1}}{\partial u_{n,1}} & \frac{\partial \mathscr{F}_{1,1}}{\partial u_{1,2}} & \cdots & \frac{\partial \mathscr{F}_{1,1}}{\partial u_{n,\mathscr{K}}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial \mathscr{F}_{n,1}}{\partial u_{1,1}} & \cdots & \frac{\partial \mathscr{F}_{n,1}}{\partial u_{n,1}} & \frac{\partial \mathscr{F}_{n,1}}{\partial u_{1,2}} & \cdots & \frac{\partial \mathscr{F}_{n,1}}{\partial u_{n,\mathscr{K}}} \\ \frac{\partial \mathscr{F}_{1,2}}{\partial u_{1,1}} & \cdots & \frac{\partial \mathscr{F}_{1,2}}{\partial u_{n,1}} & \frac{\partial \mathscr{F}_{1,2}}{\partial u_{1,2}} & \cdots & \frac{\partial \mathscr{F}_{1,2}}{\partial u_{n,\mathscr{K}}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial \mathscr{F}_{n,\mathscr{K}}}{\partial u_{1,1}} & \cdots & \frac{\partial \mathscr{F}_{n,\mathscr{K}}}{\partial u_{n,1}} & \frac{\partial \mathscr{F}_{n,\mathscr{K}}}{\partial u_{1,2}} & \cdots & \frac{\partial \mathscr{F}_{n,\mathscr{K}}}{\partial u_{n,\mathscr{K}}} \end{pmatrix}. \tag{2.41}$$

When using a discretization scheme that is based upon local approximations, such as the FDM and FEM, this full Jacobian matrix J becomes a sparse matrix. Hence, the resulting sparse linear system from Newton's method which is presented in Equation (2.40) can be solved for δu by the linear solvers which are described in Section 2.3.1. The combination of Newton's method and the Krylov subspace methods is well known, and an example can be found in [137]. Once δu has been obtained by a linear solver, a correction step $u^{l+1} = u^l + \delta u$ can be carried out easily by Newton's method. More details of Newton's method and the nonlinear Jacobi and Gauss-Seidel iterations can be found in [61, 175, 216].

Similar to the previous section, a single nonlinear elliptic PDE is chosen for the purpose of demonstration. The use of the implicit temporal discretization schemes that are described in Section 2.1.2 reduces nonlinear parabolic PDEs which are time-dependent into the form of $A(u) = f$. In addition, this system arises at each time step. So for each time step, the described nonlinear solution methods may be applied to solve the resulting nonlinear system $A(u) = f$.

Both described linear and nonlinear solvers do not have a complexity of $\mathcal{O}(n)$ when applied to general algebraic systems arising from local discretizations of PDEs. On the other hand, multigrid methods appear to be one of the fastest for solving the algebraic systems of equations for elliptic and parabolic problems. In the following section, multigrid methods which are the main focus of this thesis, are described in detail.

2.4 Multigrid

Having briefly introduced some common solution methods for the discrete algebraic systems arising from the local discretization of PDEs, in this section multigrid methods, which are the main focus in this thesis, are described in detail. Firstly in Section 2.4.1, the general idea of multigrid is introduced, along with the exploitation of the so-called "smoothing property" of some of the iterative methods that are summarized in the previous section. In Section 2.4.2, a linear multigrid method is described. Then in Section 2.4.3, Newton's method is employed to linearise a nonlinear algebraic system of PDEs arising from the local discretization of a nonlinear PDE, and then application of the linear multigrid method is described for the resulting linear system at each Newton iteration. Alternatively, for solving nonlinear PDEs, a nonlinear multigrid method called the Full Approximation Scheme (FAS) is described in Section 2.4.4. This nonlinear FAS multigrid method is the key method in this thesis. Finally, to benefit from using spatial adaptivity, which is discussed previously in Section 2.2.1, the extension of the nonlinear FAS multigrid method to include adaptivity, the Multi-Level Adaptive Technique (MLAT), is

described in Section 2.4.5.

2.4.1 Introduction to Multigrid

The idea of using multiple grids to enhance the computation can be traced back to 1946, Southwell in his publication [207] described an application which solves on a coarse grid and then interpolates the solution to improve the initial guess on fine grid. However, multigrid methods operate differently. Brandt in his 1977 paper, entitled "Multi-Level adaptive solutions to boundary-value problems" [29], systematically describes the first multigrid methods, and some of their applications. The paper is known as one of the first, and one of the most important, publications on this subject (and it is certainly one of the most highly cited publications).

The multigrid method is commonly accepted as being one of the fastest numerical methods for the solution of the systems of equations that arise from locally discretising elliptic PDEs and elliptic systems of PDEs. This is because it is able to solve a linear algebraic system of equations for elliptic and parabolic problems involving n unknowns with a computational cost of $\mathscr{O}(n)$ [216]. Although the multigrid methods are now known as a single algorithm, they are in fact a combination of solution methods, and the multigrid methods employ them in such a way that solutions can be obtained in the most efficient, and accurate manner.

The multigrid methods operate on a hierarchy of grids. In Figure 2.7, such a hierarchy of 2-D uniform Cartesian grids is shown. These grids are defined such that on a grid level, $\mathscr{L}$, two adjacent grid points are a distance $h^{\mathscr{L}} = \Omega_x/2^{\mathscr{L}}$ apart, where Ω_x is the size of the computational domain in x direction (and $\Omega_x = \Omega_y$ in this case). The multigrid methods that are considered in this thesis, which operate on a hierarchy of grids such as that shown in Figure 2.7, all belong to the family of methods known as geometric multigrid. There are multigrid methods that do not require an actual hierarchy of meshes to be generated, and these methods are called algebraic multigrid. This type of multigrid method is not considered in this thesis, therefore no further detail is given, however, the interested reader is directed to [31, 216] for detailed descriptions, and some applications of algebraic multigrid can be found in [60, 204]. Multigrid methods can also be used as preconditioners, see [5, 73] for examples.

Due to the nature of many classical iterative methods, such as the Jacobi and the Gauss-Seidel iterations, which are discussed in the previous section, they may possess the so-called "smoothing property", which multigrid is then able to exploit. An iterative method is said to possess this "smoothing property" if it tends to converge much quicker

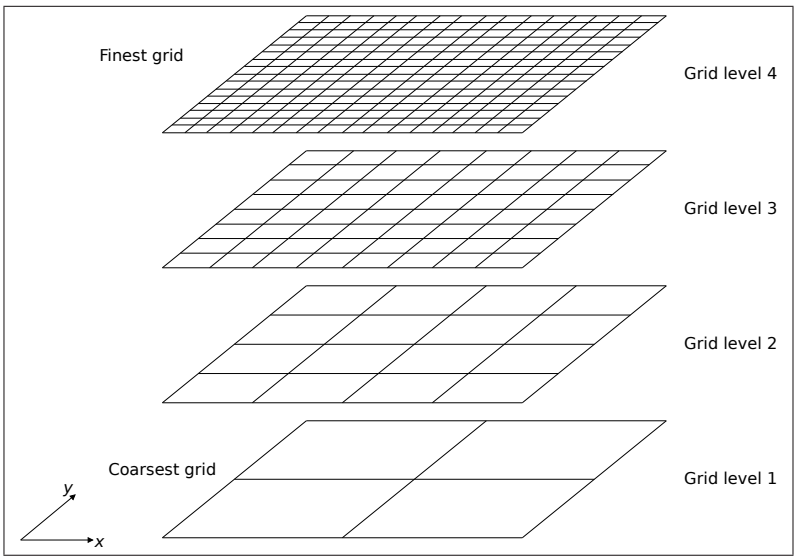

Figure 2.7: Sketch shows a hierarchy of four 2-D uniform Cartesian grids, each of which has grid spacing $h^{\mathscr{L}} = \Omega_x/2^{\mathscr{L}}$, where $\mathscr{L}$ is the level of grid and Ω_x is the size of the computational domain in x direction (and $\Omega_x = \Omega_y$ in this case).

when the error has high frequency components. More specifically, when all Fourier components (up to the resolution of the grid) are present in the initial error, smoothing methods will damp out the highest frequency components of the error in a very small number of iterations. Therefore, by applying a few sweeps of such an iterative method on a fine grid, a large reduction of the high frequency components of the error is achieved. In order to remove the remaining low frequency components of the error on the fine grid, the algebraic system is moved down to a coarser grid in the grid hierarchy (see Figure 2.7). Since the number grid points is reduced on such coarser grid, part of the error now becomes high frequency. Therefore the iterative method can still quickly reduce the high frequency components of the error within a few number of sweeps. This is repeated until the coarsest grid is reached (grid level 1 in Figure 2.7 for example). A large amount of computations can be performed to obtain an "exact" solution on the coarsest grid, with a reasonable time cost. This "exact" solution can then be used to improve the fine grid solution. The computations that are performed on these coarser grids are termed the coarse grid correction, whereas the solver on the coarsest grid is termed the coarse grid solver. The iterative method used on each level of grids (may exclude coarse grid solver if different from this iterative method) is termed the smoother.

It is known that the convergence rate of a multigrid method is independent from the size of finest grid, based upon two conditions: a well-functioning smoother with the "smoothing property" and a well-functioning coarse grid solver which can obtain the "ex-

act" solution on the coarsest grid. The cost of performing multigrid methods may thus be proven to be $\mathcal{O}(n)$ in such situation. Comprehensive introductions to multigrid analysis can be found in [31, 216]. Some applications of multigrid methods can be found in, for example, [6, 23, 30, 81, 85, 90, 91, 147, 199, 225, 226]. In the following section, the linear multigrid method is introduced.

2.4.2 Linear Multigrid

The use of the multigrid methods to solve algebraic systems arising from local discretizations of linear elliptic PDEs is based upon an important relation between residual and error for linear equations. These two concepts are briefly discussed in Section 2.3.1, and they are described here in detail. However before that, a simple 2-D linear boundary value problem is introduced in order to present the linear multigrid method clearly. For example, the Poisson's equation, when discretized (e.g. by applying the FDM that is described previously in Section 2.1.1.1) leads to a linear algebraic system that my be written in the following form:

$$Au = b. \tag{2.42}$$

Here A is the coefficient matrix, u denotes the vector of the solution values for this problem, and b denotes the right-hand side (RHS) vector. Now consider an approximation to the solution of Equation (2.42) obtained by using a small number of iterations of a fixed point iterative method (e.g. Jacobi from Equation (2.22) or Gauss-Seidel from Equation (2.23)). Let v denote such an approximate solution of Equation (2.42). Hence, the exact error E, defined by the difference between u and v, is given by

$$E = u - v. \tag{2.43}$$

On the other hand, the amount by which the approximate solution fails to satisfy Equation (2.42) is known as the residual (or defect), and is shown in the following residual equation:

$$r = b - Av. \tag{2.44}$$

Note that the error E is generally unknown, however the residual may always be computed. Having defined the concept of error and residual, now the relation between them

can be observed:

$$\begin{aligned} AE &= A(u-v) \\ &= Au - Av \\ &= b - Av \\ &= r. \end{aligned} \tag{2.45}$$

The Equation (2.45) is called the error equation. However, to obtain the exact error E from solving this error equation is just as difficult as exactly solving the original problem in Equation (2.42). If v was computed using a small number of iterations of an iterative method with the "smoothing property", we know that E may be accurately represented on a coarser grid (as its highest frequency components will be negligible). Therefore, instead of trying to find the exact error E by solving Equation (2.45) directly, an error approximation e is obtained by approximating the system (2.45) on a coarser grid and solving that (smaller) system instead. Using this error approximation, a better approximation of v can then be computed as the following:

$$v^{\text{better approximation}} = I_{2h}^{h} e + v, \tag{2.46}$$

where I_{2h}^{h} is an interpolation operator, such as Equation (2.12).

To summarise a two-grid linear multigrid method, firstly a small number of sweeps of a smoother (e.g. Jacobi from Equation (2.22) or Gauss-Seidel from Equation (2.23)) are performed on the fine grid, using the initial guess of v to obtain an approximate solution, v^f. The superscript f denotes values (or operations that are carried out) on the fine grid. This is termed a pre-smoother in the context of the multigrid methods. Then residual r^f is calculated on the fine grid as shown in Equation (2.44). A restriction operator I_f^c (formerly defined as I_h^{2h} in Section 2.2.1.2) is now required to transfer the values of the fine grid residual r^f to the coarse grid. The approach which is shown previously in Equation (2.13) can be used to obtain r^c on the coarse grid. The superscript c denotes values (or operations that are carried out) on the coarse grid. On the coarse grid, the error equation $A^c e^c = r^c$ can then be solved with an initial guess $e^c = 0$ by a coarse grid solver. Standard direct or iterative methods, which are described in Section 2.3.1, can be used here, however, these may still be quite slow if the coarse grid has a large number of degrees of freedom. After applying the coarse grid solver, e^c is interpolated back to the fine grid. A interpolation operator I_c^f (formerly defined as I_{2h}^h in Section 2.2.1.1) is required, and the approach that is shown previously in Equation (2.12) can be used to obtain e^f. On the fine grid, a correction process is carried out as illustrated in Equation (2.46). Finally,

a few sweeps of the smoother are applied to re-adjust the solution for possible high frequency error components that may be introduced in the correction process. This is termed a post-smoother in the context of the multigrid methods. This description summarises one iteration of the linear multigrid method with a two-grid approach.

In practice, if the number of degrees of freedom on the fine grid is very large, to apply an "exact" coarse grid solver on a relatively coarser grid in a two-grid approach, is very inefficient. Therefore, multiple grids are used, this then requires the two-grid method to be performed recursively. Algorithm 3 illustrates a standard V-cycle linear multigrid method (more details on multigrid cycle strategies are described in Section 2.4.2.1). Within the algorithm, p_1 and p_2 are the numbers of sweeps that are performed by pre- and post-smoothers, respectively. Their values can be problem-specific, however, they are typically in range of zero to four. Additionally, in Algorithm 3, superscript h is the distance between two adjacent points on a given structured grid (e.g. grids shown in Figure 2.7), and Ω_h denotes such a grid.

Algorithm 3 V-cycle linear multigrid method [216]

p_1 and p_2 are the number of sweeps performed by pre- and post-smoothers respectively; and superscript h is the distance between two adjacent points on a given grid Ω_h.

Function: v^h = V-cycleLMG(h, v^h, b^h, A^h)

1. Apply p_1 iterations of the pre-smoother on $A^h v^h = b^h$
$v^h = \text{PRE-SMOOTH}(p_1, v^h, A^h, b^h)$

2. Compute the residual r^h
$r^h = b^h - A^h v^h$

3. Restrict the residual r^h from Ω_h to Ω_{2h} to obtain r^{2h}
$r^{2h} = I_h^{2h} r^h$

4. Set the initial guess for e^{2h} to be 0
if Ω_{2h} = coarsest grid **then**
Perform an "exact" coarse grid solver on $A^{2h} e^{2h} = r^{2h}$
else
e^{2h} = V-cycleLMG$(2h, e^{2h}, r^{2h}, A^{2h})$
end if

5. Interpolate the error e^{2h} from Ω_{2h} to Ω_h to obtain e^h
$e^h = I_{2h}^h e^{2h}$

6. Perform correction
$v^h = v^h + e^h$

7. Apply p_2 iterations of the post-smoother on $A^h v^h = b^h$
$v^h = \text{POST-SMOOTH}(p_2, v^h, A^h, b^h)$

This process illustrated in Algorithm 3 can be repeated until a user-specified stopping criterion is satisfied, and commonly a suitable norm of the residual is considered. For different models that are presented in this thesis, stopping criteria are slightly different

from one another, and are individually described in detail in later chapters.

Although here an elliptic equation is used as an example, the linear multigrid method can be applied to time-dependent problems (e.g. linear parabolic PDEs). For example, when an implicit temporal discretization scheme (e.g. BDF2 method that is described in Section 2.1.2.2) is applied (following a standard spatial discretization), each time step consists of a fully-discrete system. Such a system can be solved by using the linear multigrid method. One important feature for multigrid methods in general is that the number of V-cycles that are needed for convergence is independent from the grid size, and this is demonstrated across a range of applications in later chapters.

Having mentioned the standard V-cycle for the linear multigrid method, in the following subsection, multigrid cycle strategies, including the V-cycle, are described in more detail.

2.4.2.1 Multigrid Cycle Strategies

Algorithm 3 in the previous section illustrates a standard V-cycle strategy, which consists of only one coarse grid correction within each level of the multigrid cycle. In this section, some other multigrid cycle strategies are described. A complete multigrid cycle starts from applying the pre-smoother on the finest grid, and ends with the post-smoother on the finest grid.

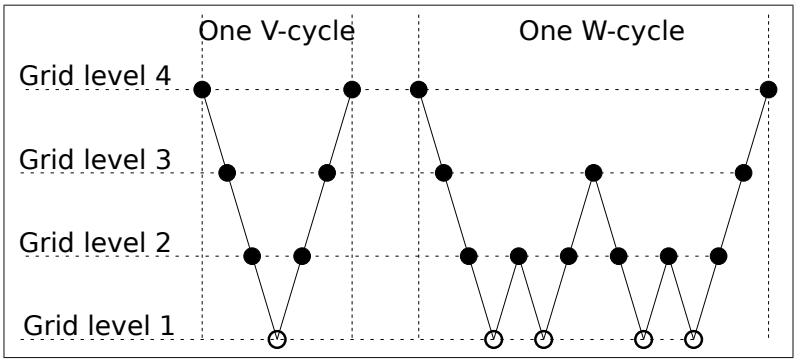

Figure 2.8: Sketch shows a standard V-cycle and a W-cycle in a four-grid multigrid method. Within the cycles, • shows where smoother is used; ○ denotes the "exact" coarse grid solver; \ denotes the fine-to-coarse restriction and / denotes the coarse-to-fine interpolation.

In the left half of Figure 2.8, one V-cycle is illustrated for a four-grid multigrid method. Note that between two smoothers on each grid, only one coarse grid correction is used.

An alternative is the so-called W-cycle, which is shown in the right half of Figure 2.8. The same operations and notations (i.e. •, ○ \ and /) are used as in the V-cycle case. The

idea of the W-cycle is that after the pre-smoother is applied, at each level other than the finest, two coarse grid corrections are then applied consecutively. The W-cycle that is shown in Figure 2.8 is also termed W2, which indicates that one additional coarse grid correction is needed after each new interpolation. Note W1 is the original V-cycle, and W3 runs two additional coarse grid corrections after each new interpolation, etc.

The improvement of the W-cycle is the increased number of coarse grid solvers applied in one cycle, along with an increased number of pre- and post-smoothers at each grid level. In many cases this can cause the iteration to converge faster. On the other hand, the trade-off of the W-cycle is the extra cost per cycle. The use of the W-cycle can be found in [153], for example.

Another multigrid cycle strategy is called Full Multigrid (FMG), which is shown in Figure 2.9. Unlike the V- and W-cycles, FMG starts on the coarsest grid,

This computation begins with a coarse grid solver, followed by interpolation of the solution to the next finer level. Then a single V-cycle (or W-cycle) is carried out at each level, followed by another interpolation of the solution to the next level. Finally, a standard cycle is repeated. These new interpolations at the end of each stage in the FMG require extra attention and are marked as // in Figure 2.9. In contrast to the interpolation that is used in standard cycles of the linear multigrid method, the FMG interpolation transfers approximations of the solution u from Ω_{2h} to Ω_h, instead of error e. It is worth noting that the FMG is not a complete cycle, thus it may only be used once at the beginning of the computation for solving an elliptic problem (or at the beginning of each time step for a time-dependent parabolic problem). An application that combines the FMG and the W-cycle can be found in [53].

Apart from those multigrid cycle strategies described above, it is worth noting there are other cycle strategies too. For example, the F-cycle starts on the finest grid but performs a similar pattern as the FMG cycle after the first coarse grid solver. Unlike the FMG, the F-cycle performs complete cycles. The description of this can be found in [216], however no further discussions are made here.

It is noteworthy that when parallel computing is used (discussed in Section 2.5), the communication on the coarsest grid may prevent the parallel algorithm from executing with good parallel efficiency. In such cases, multiple visits to the coarsest grid in one cycle (e.g. W-cycle and FMG) may not be necessarily the best option in terms of efficiency. This is one of the main reasons that only the V-cycle is used in this thesis (the multigrid algorithm in parallel is discussed further in Section 2.5.5, where this issue is discussed in detail).

In the following sections, nonlinear PDEs are considered, initially by combining New-

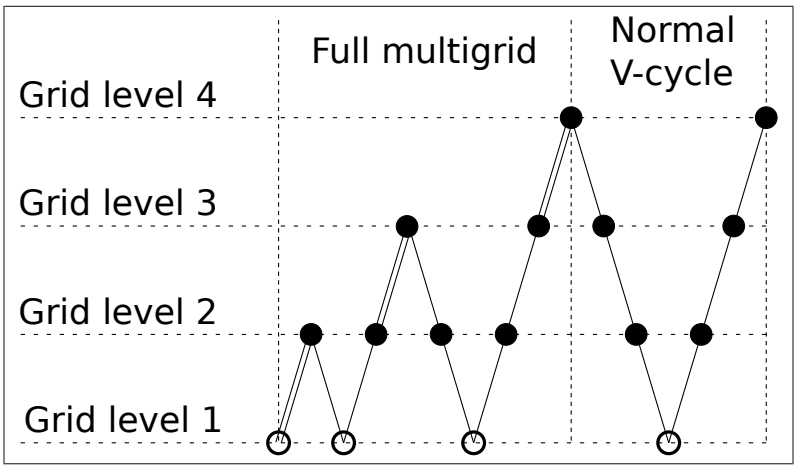

Figure 2.9: Sketch shows the full multigrid (FMG) cycle strategy, where computation starts on the coarsest grid, and after each new interpolation, a coarse grid solver is required. Upon reaching the finest grid, V-cycle (or W-cycle if is needed) can be carried out normally with an improved initial guess. Note • denotes where smoother is used; ∘ denotes the "exact" coarse grid solver; \ denotes the fine-to-coarse restriction; / denotes the coarse-to-fine interpolation and // denotes the FMG interpolation which interpolates the approximate solution u from a coarse grid to a fine grid.

ton's method with linear multigrid, and then by introducing a nonlinear multigrid scheme.

2.4.3 Newton Multigrid

Newton's method is a well known technique for solving nonlinear algebraic equations and systems. It is introduced in Section 2.3.2. The use of Newton's method provides a global linearization of a nonlinear system. The resulting linear system for the correction term δu, as demonstrated in Equation (2.40), needs to be solved by some linear solver. In the previous section, the described linear multigrid is applied to the solution of discretizations of linear PDEs, providing a suitable smoother and coarse grid correction are used. In this section, we generalise the linear multigrid approach based upon the combination of Newton's method and the linear multigrid method.

Consider a simple nonlinear boundary value problem consisting of a single nonlinear PDE. When discretized, the equation can be written as

$$\mathscr{F}(u) = 0, \tag{2.47}$$

where $\mathscr{F}$ is a known nonlinear function, and u is the true solution of this discretized nonlinear problem. An approximate solution v of the true solution u can be obtained by following an iterative step, and this is shown in Equation (2.38). From Equations (2.38) to

(2.40), the linear system which needs to be solved is derived. This linear system is given as

$$J\delta v = -\mathscr{F}(v), \tag{2.48}$$

where J is the Jacobian matrix and is defined in Equation (2.41), and δv is the correction term in Newton's method.

Given an initial guess v^0, the Jacobian matrix J^0 can be formed and the RHS values $-\mathscr{F}(v^0)$ can also be calculated. Since δv is the correction term, the initial guess $\delta v^0 = 0$ is commonly used in Newton's method. Applying the linear multigrid to this linear problem, δv^0 can be obtained with the optimal amount of computations. Then the solution, v, to the original nonlinear problem can be updated as $v^1 = v^0 + \delta v^0$. If the approximate solution v^1 is not satisfactory, then the Jacobi matrix J^0 is updated to J^1 using the solution v^1, and the linear problem $J^1\delta v^1 = -\mathscr{F}(v^1)$ is solved, again by the linear multigrid method. This process repeats until an acceptable solution v^l is obtained after l iterations. The above description briefly summarises the Newton multigrid method without repeating the linear multigrid method which is already explained in the previous section. Algorithm 4 uses the linear multigrid function which is defined in Algorithm 3 to illustrate the Newton multigrid method.

It is worth noting that there are several changes required in order to carry out Algorithm 4 efficiently in practice. One of them is associated with forming the Jacobian of the coarse grid approximations (i.e. A^{2h} in step 4 from Algorithm 3). It is very computationally expensive to directly approximate the Jacobian on coarser grids, so typically $A^{2h} = I_h^{2h} A^h I_{2h}^h$ is used in the Newton multigrid version of Algorithm 3.

For time-dependent nonlinear problems, if an implicit temporal discretization scheme is required, then the Newton multigrid function that is presented in Algorithm 4, may be carried out for the discrete system at each implicit time step. For simplicity, after the first time step, the most up-to-date solution v from the latest time step can be used as the initial guess in the next time step.

In this thesis, the Newton multigrid method is not applied. The interested reader is directed to [31, 216] for more information. An application of the Newton multigrid method can be found in [28], where the authors compare the Newton multigrid with a nonlinear multigrid method. They also discuss a number of practical aspects of its efficient implementation and of convergence.

As an alternative for solving a nonlinear problem, the nonlinear multigrid method is described in the following section.

Algorithm 4 Newton multigrid method [216]

v^0 is an initial guess for the discrete nonlinear equation $\mathscr{F}(v) = 0$ on a given grid Ω. Neighbouring grid points have h distance on the grid Ω. The superscript l is the number of iterations. This NewtonMG function requires the linear multigrid function V-cycleLMG which is presented in Algorithm 3

Function: u = NewtonMG$(v^0, \mathscr{F}(v))$

1. Initialisation

$l = 1$

$v^l = v^0$

while u^l does not satisfy the stopping criterion **do**

2. Generate the Jacobian matrix J^l based upon the solution v^l

3. Calculate the RHS vector $\mathscr{F}(v^l)$

4. Initialise the correction term δv for Newton's method

$\delta v = 0$

while δu does not provide a good enough correction **do**

5. Perform a linear multigrid solver

$\delta v =$ V-cycleLMG$(h, \delta v, -\mathscr{F}(v^l), J^l)$

end while

6. Perform the correction of Newton's method

$v^{l+1} = v^l + \delta v$

7. Increases the number of iterations

$l \mathrel{+}= 1$

end while

2.4.4 Nonlinear Multigrid

Having described the use of a linear multigrid method in combination with Newton's method, in this section, a nonlinear multigrid method, known as the Full Approximation Scheme (FAS), is introduced. This scheme is the main focus of this thesis, and is designed to treat the nonlinearity directly, based upon a nonlinear smoother followed by a modified coarse grid correction.

The major distinction between linear and nonlinear multigrid methods is that the relation between error and residual (shown in Equation (2.45)) does not hold in the nonlinear case, and therefore no error equation (i.e. $AE = r$) can be formed. Instead of the coarse grid correction being based upon solving for the error approximation e, the FAS uses a coarse grid correction that is based upon solving for the solution itself on the coarsest grid. To achieve this, the FAS restricts both the residual r and the approximate solution v to the coarser grids. Thus the original nonlinear problem is approximated directly on these coarser grids, however with a modified Right-Hand Side (RHS).

In order to describe a two-grid nonlinear multigrid method clearly, a single PDE which is a nonlinear boundary value problem is considered here. When discretized on the fine grid, the equation can be written as in Equation (2.49). Note the superscript f denotes these values on the fine grid, and later on the superscript c denotes values on the coarse grid. This notation is used throughout this section. The discrete nonlinear system is

$$A^f(u^f) = f^f, \tag{2.49}$$

where A^f, instead of being a single matrix as in the linear case, is a vector-valued nonlinear function that takes u^f as input, where u^f is the true solution for the nonlinear problem on the fine grid, and f^f is a vector of the RHS values for the fine grid problem. Equation (2.49) has a following equivalent form:

$$\mathscr{F}^f(u^f) = 0, \quad \text{where} \quad \mathscr{F}^f(u^f) = A^f(u^f) - f^f. \tag{2.50}$$

A nonlinear pre-smoother is required in the nonlinear multigrid method: firstly to obtain an approximate solution v^f of the true solution u^f; secondly to be able to remove the high-frequency error components in a small number of sweeps (in other words, it should have the "smoothing property"). The point-wise nonlinear Jacobi iteration which is presented in Equation (2.34) can be used. Applying this iterative method to the equation on the fine grid, an approximate v^f can be obtained.

In this nonlinear case, the residual on the fine grid r^f can be calculated in a similar

manner to the linear case (and also can be calculated at any given time). The following equation demonstrates this process:

$$r^f = -\mathscr{F}^f(v^f) = f^f - A^f(v^f). \tag{2.51}$$

Next, as mentioned previously, the FAS restricts both residual r^f and the approximate solution v^f to the coarse grid by applying a restriction operator I_f^c. Since only the FDM is used in this thesis, the approach shown in Equation (2.13) can be used for restricting both the fine grid residual and the fine grid solution. If, for instance, the FEM which is briefly discussed in Section 2.1.1 is used, then two different restriction operators are required. This is because for restricting the solution to the coarse grid, it is the integrals of those fine grid elements that need to be considered, instead of the values on grid points. No further discussion is made on the multigrid with the FEM, more detailed descriptions can be found in [31, 216].

A restricted residual r^c and a restricted solution w^c are obtained on the coarse grid through the use of the restriction operator I_f^c

$$\begin{aligned} r^c &= I_f^c r^f, \\ w^c &= I_f^c v^f. \end{aligned} \tag{2.52}$$

Having the restricted solution w^c, the original nonlinear problem from the fine grid is now solved on the coarse grid, however, with a modified RHS. The following equation demonstrates the coarse grid problem with such a modified RHS:

$$\begin{aligned} A^c(v^c) &= f^c + [r^c - (f^c - A^c(w^c)] \\ &= r^c + A^c(w^c), \end{aligned} \tag{2.53}$$

where A^c is a vector-valued nonlinear function from the discretization on the coarse grid with the same boundary conditions as on the fine grid. Note the approximate solution v^c on the coarse grid can be solved with a RHS that involves only the known restricted values r^c and w^c. The coarse grid solver may use the nonlinear Jacobi iteration presented in Equation (2.34), for example, however many sweeps may be needed to obtain a converged solution. Upon obtaining the solution v^c on the coarse grid, an error approximation e^c can then be calculated on the coarse grid as

$$e^c = v^c - w^c. \tag{2.54}$$

The same interpolation operator from Equation (2.12) can be applied on e^c to obtain e^f on the fine grid. This allows the correction process to be carried out in the same manner as the linear multigrid method, shown as the following:

$$v^{f\text{ better approximation}} = v^f + e^f. \tag{2.55}$$

This is followed by a few sweeps from a post-smoother (i.e. the nonlinear Jacobi or Gauss-Seidel iteration) to remove possible high frequency error components that are potentially introduced during the correction process. The above description summarises one cycle of the two-grid nonlinear multigrid method, and can be repeated if a user-specified stopping criterion is not satisfied.

For the same reason as the linear multigrid method that is described before, the nonlinear multigrid method normally operates on multiple grids. This requires a recursive algorithm, and in Algorithm 5, a standard V-cycle nonlinear FAS multigrid method is illustrated, where the notation is the same as in the linear case.

Algorithm 5 illustrates one V-cycle of the nonlinear FAS multigrid method, however, multiple cycles can be performed if a user-specified stopping criterion is not satisfied. Considerations associated with the choice of stopping criteria are discussed later in this thesis.

Although a nonlinear boundary value problem is used as an example here, the nonlinear multigrid method can be extended to time-dependent problems (e.g. nonlinear parabolic PDEs). This is done by applying the nonlinear multigrid method to the fully-discrete system that arises at each time step from using an implicit temporal discretization.

There is an alternative nonlinear multigrid method, according to Hackbusch [102] which is different from the nonlinear FAS multigrid. The main difference is that for the nonlinear FAS multigrid, the initial approximation on the coarse grid w^{2h} comes from restricting v^h on the fine grid (as illustrated in Algorithm 5). However, Hackbusch's method uses values from the FMG process as the first approximation on the coarse grid, since this approximation is a solution of the nonlinear coarse grid equation. In addition, Hackbusch's method includes two scaling factors c and $1-c$, in the restriction of the residual and the interpolation of the correction respectively. Therefore, the residual from the fine grid and the correction from the coarse grid do not take the full weight. This is used to ensure the solvability of the coarse grid equation [216]. Details of the Hackbusch's nonlinear multigrid method and the choices of the scaling factors can be found in [102].

This thesis is primarily concerned with the development and application of multigrid methods for the solution of nonlinear parabolic systems of PDEs. Each of these models

Algorithm 5 V-cycle nonlinear FAS multigrid method for a single nonlinear PDE [216]

p_1 and p_2 are the number of sweeps performed by pre- and post-smoothers respectively; and superscript h is the distance between two adjacent points on a given grid Ω_h.

Function: v^h = V-cycleFASMG$(h, v^h, f^h, A^h(v^h))$

1. Apply p_1 iterations of the pre-smoother on $A^h(v^h) = f^h$
$v^h = \text{PRE-SMOOTH}(p_1, v^h, A^h(v^h), f^h)$

2. Compute the residual r^h
$r^h = f^h - A^h(v^h)$

3. Restrict the residual r^h from Ω_h to Ω_{2h} to obtain r^{2h}
$r^{2h} = I_h^{2h} r^h$

4. Restrict the fine grid approximate solution v^h from Ω_h to Ω_{2h} to obtain w^{2h}
$w^{2h} = I_h^{2h} v^h$

5. Compute the modified RHS
$f^{2h} = r^{2h} + A^{2h}(w^{2h})$

6. if Ω_{2h} = coarsest grid **then**
Perform an "exact" coarse grid solver on $A^{2h}(v^{2h}) = f^{2h}$
else
v^{2h} = V-cycleFASMG$(2h, w^{2h}, f^{2h}, A^{2h}(v^{2h}))$
end if

7. Compute the error approximation e^{2h} on Ω_{2h}
$e^{2h} = v^{2h} - w^{2h}$

8. Interpolate the error approximation e^{2h} from Ω_{2h} to Ω_h to obtain e^h
$e^h = I_{2h}^h e^{2h}$

9. Perform correction
$v^h = v^h + e^h$

10. Apply p_2 iterations of the post-smoother on $A^h(v^h) = f^h$
$v^h = \text{POST-SMOOTH}(p_2, v^h, A^h(v^h), f^h)$

consists of multiple coupled equations, and some variables appear in several equations. The nonlinear multigrid method with the FAS, which is described earlier in this section for a single PDE can still be applied to these complex systems. Almost all operators from the nonlinear FAS multigrid method for singe equation, like restriction, interpolation and coarse grid correction have to be extended to work on all variables in such a system, however the fundamental principles are not changed. The main difference comes from the nonlinear smoother and the extra complexity of the coarse grid solver. For the nonlinear system of PDEs, the nonlinear block Jacobi iteration which is presented in Equation (2.36) and discussed in Section 2.3.2 can be used as the smoother, and also as the coarse grid solver. Note this method updates all variables simultaneously on a grid point. In this case, the coarse grid solver performs the same iteration with a "block" structure, and often with many more sweeps in order to obtain an "exact" solution on the coarsest grid. Similar methods that can be used as replacements are, for example, the block nonlinear Gauss-Seidel iteration and the Gauss-Picard method, whose descriptions can be found in [61, 216].

So far the spatial adaptivity has not yet been considered in the described multigrid methods and the resulting adaptive grids (see for example Figure 2.3). In the following section, an adaptive multigrid method that is based upon the nonlinear FAS multigrid method is introduced.

2.4.5 Adaptive Multigrid

The spatial adaptivity that is described in Section 2.2.1, and the resulting adaptive grids (see for example Figure 2.3) plays a major role in terms of improving the computational efficiency. Therefore a mechanism for applying multigrid methods in combination with grid adaptivity is needed. In this section, the Multi-Level Adaptive Technique (MLAT) from Brandt [29] is described in detail since this is used throughout this thesis. The MLAT is applied as part of a nonlinear multigrid scheme with the FAS, which is discussed in Section 2.4.4. The major differences that are introduced by the inclusion of adaptivity are handled by the MLAT, through the use of temporary Dirichlet boundary points.

In order to demonstrate the MLAT clearly 1-D cell-centred Cartesian grids are used, as illustrated in Figure 2.10 where a three-level hierarchy of grids is shown as an example. On these grids, a nonlinear problem $A(u) = f$ is considered, and v is the approximate solution of u. Grid level 3 in Figure 2.10 is the finest grid, and only covers a part of the whole domain. By setting the two end-points of this grid to be temporary Dirichlet boundary points (i.e. fix their own values), the nonlinear Jacobi iteration (or the nonlinear

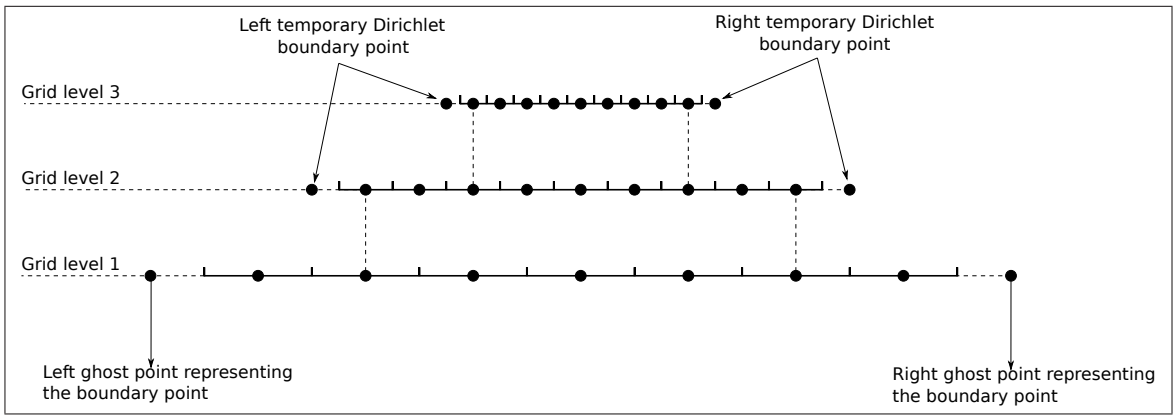

Figure 2.10: Sketch shows a three-level hierarchy of 1-D adaptive cell-centred grids, the MLAT with nonlinear multigrid method can be applied to these grids. For these grids that only cover some regions of the domain, temporary boundary points are illustrated.

block Jacobi iteration if there is a system of PDEs) can be applied to the interior of grid level 3 as usual. It is worth noting that due to the use of cell-centred grid points, the two end-points are positioned outside the temporary boundary. We refer these points as ghost points in Section 2.1.1.2. Upon having an approximate solution $v^{\mathscr{L}=3}$, the residual on internal points of grid level 3 can be calculated. The approximate solution and the residual are then restricted via the same restriction operator that is used in the nonlinear FAS multigrid. However, the restricted solution and residual occur in the region that belongs to both grid levels 3 and 2 (i.e. $\Omega^{\mathscr{L}=3} \cap \Omega^{\mathscr{L}=2}$). For those regions that only belong to grid level 2 but not grid level 3, two conditions may occur. Firstly, if it is the first V-cycle, then the initial guess is considered as the coarser grid solution $v^{\mathscr{L}=2}$ on grid level 2. For time-dependent problems, this happens to the first V-cycle at every time step. Secondly, on the subsequent V-cycles, $v^{\mathscr{L}=2}$ takes the most up-to-date values as the solution on grid level 2. The next step in the nonlinear FAS multigrid is to compute the modified RHS, and this can be done on the intersection region, by using the restricted solution and residual. For those regions that have no restricted values, the original RHS $f^{\mathscr{L}=2}$ on grid level 2 is used.

Although the example from Figure 2.10 is using 1-D cell-centred grids, to extend this to higher dimensions is straightforward.

The above process repeats from grid level 2 to grid level 1. It is assumed that the coarsest grid will always cover the whole domain. So there is no need for temporary Dirichlet boundary points on grid level 1, instead, the original boundary points are used. The coarse grid solver can then be applied as usual. The nonlinear Jacobi iteration from Equation (2.34) can be used (or the nonlinear block Jacobi iteration from Equation (2.36)

if there is a system of PDEs). Many sweeps may be needed to obtain a converged solution on the coarsest grid.

Upon obtaining the "exact" solution on grid level 1, the error approximation $e^{\mathscr{L}=1}$ is calculated, but only on regions which are shared with grid level 2. Then the error approximation is interpolated to grid level 2 via the same interpolation operator which is used in the nonlinear FAS multigrid. This is followed by the usual correction process and a few sweeps of a post-smoother. For the remaining regions on grid level 1, the original solution is replaced by the "exact" coarse grid solution. This process ends when the post-smoother on the finest grid (grid level 3) is carried out. The above description summarises a three-grid V-cycle of nonlinear multigrid method with the combination of FAS and MLAT. In Algorithm 6, a recursive approach for a more general V-cycle adaptive multigrid is illustrated.

Algorithm 6 only considers the algebraic system arising from the discretizations of a single nonlinear PDE. We have discussed a nonlinear FAS multigrid which deals with a system of nonlinear PDEs in the previous section. The adaptive multigrid with MLAT that is presented Algorithm 6 can be extended to solve a system of PDEs in a similar manner. Details about the software implementation of the adaptive multigrid method with FAS and MLAT are discussed in the next chapter.

Apart from the MLAT, there is another common scheme for handling load adaptivity that is also based upon the nonlinear FAS multigrid method. This is called the Fast Adaptive Composite Grid (FAC) method. The FAC method uses a so-called conservative discretization at the interface. During the computation, conservation interpolation is applied, so that finer grids which cover sub-regions of the domain may have correct temporary boundary values around them. Thus their interior can be updated by nonlinear iterative methods. The use of the FAC methods is generally associated with the FVM. The interested reader is directed to [164,202] for more information about the FAC method. Other methods for the adaptive multigrid also exist, for example, applications which are associated with the FEM can be found in [21, 122].

Having described multigrid methods, in the next section, parallel computing is introduced which, from a computational point of view, can greatly improve the efficiency.

2.5 Parallel Computing

Having described techniques that aim to gain efficiency based upon advanced mathematical algorithms such as adaptivity and multigrid to solve PDEs, here a further improvement is discussed from a computational point of view. Large numerical simulations of scien-

Algorithm 6 V-cycle MLAT nonlinear FAS multigrid method [216]

p_1 and p_2 are the number of sweeps performed by pre- and post-smoothers respectively; and superscript h is the distance between two adjacent points on a given grid Ω_h.

Function: v^h = V-cycleMLATMG$(h, v^h, v^{2h}, f^h, f^{2h}, A^h(v^h), A^{2h}(v^{2h}))$

1. Apply p_1 iterations of the pre-smoother on $A^h(v^h) = f^h$

$v^h = \text{PRE-SMOOTH}(p_1, v^h, A^h(v^h), f^h)$

2. Compute the residual r^h on Ω_h

$r^h = f^h - A^h(v^h)$

3. Restrict the residual r^h from Ω_h to $\Omega_{2h} \cap \Omega_h$ to obtain r^{2h}

$r^{2h} = I_h^{2h} r^h$

4. Restrict the fine grid approximate solution v^h from Ω_h to Ω_{2h} to obtain w^{2h}

$$w^{2h} = \begin{cases} I_h^{2h} v^h & \text{on } \Omega_{2h} \cap \Omega_h \\ v^{2h} & \text{on the remaining part of } \Omega_{2h} \end{cases}$$

5. Compute the modified RHS

$$f^{2h} = \begin{cases} r^{2h} + A^{2h}(w^{2h}) & \text{on } \Omega_{2h} \cap \Omega_h \\ f^{2h} & \text{on the remaining part of } \Omega_{2h} \end{cases}$$

6. if Ω_{2h} = coarsest grid **then**

Perform an "exact" coarse grid solve on $A^{2h}(v^{2h}) = f^{2h}$

else

v^{2h} = V-cycleMLATMG$(2h, w^{2h}, v^{4h}, f^{2h}, f^{4h}, A^{2h}(v^{2h}), A^{4h}(v^{4h}))$

end if

7. Compute the error approximation e^{2h} on $\Omega_{2h} \cap \Omega_h$

$e^{2h} = v^{2h} - w^{2h}$

8. Update solution on the remaining part of Ω_{2h}

$v^{2h} = v^{2h}$ latest

9. Interpolate the error approximation e^{2h} from Ω_{2h} to Ω_h to obtain e^h

$e^h = I_{2h}^h e^{2h}$

10. Perform correction

$v^h = v^h + e^h$

11. Apply p_2 times of post-smoother on $A^h(v^h) = f^h$

$v^h = \text{POST-SMOOTH}(p_2, v^h, A^h(v^h), f^h)$

tific and engineering problems generally demand very substantial computational power. A finite difference or finite element mesh may contain hundreds of millions of points to satisfy the needs of accuracy and reliability [80, 91, 226]. This presents significant computational challenges both in terms of the memory required to work with such a large mesh and the execution times needed to obtain the solution of the corresponding discrete systems of equations.

The idea of parallel computing comes from natural logic: if a job is too big for one person (or one computer) to handle, we should, if we can, split it up. In the case of a computer, the split needs to take place across multiple Central Processing Units (CPUs), also referred to as processors. In more modern parallel architectures the split can also take place across the cores on each CPU. Today's computers have multiple cores, and frequently multiple CPUs or nodes [98], a node consists of multiple cores which are combined together. For example, a description of "two quad cores machine" means it has two nodes, each with four cores and so in total there are eight cores. According to the January 2014 TOP500 [209], the largest supercomputer at that time was the Tianhe-2, with over 3 million cores.

In addition to computational cores, parallel computers also need to address another component: that is memory. Generally speaking, memory is a physical device used to store running programs and their associated data until these programs terminate, where it will be made available for new programs. It is mentioned previously in Section 2.1.1, that in order to discretize PDEs (e.g. using the FEM), mesh information and coefficients of the discrete equations need to be stored. Such data needs to stay in the memory as long as the program is running, and can consume a very large amount of memory depending upon the mesh size.

In the following section, parallel architectures and assessments of parallel performance are introduced.

2.5.1 Introduction to Parallel Computing

Having introduced the concepts of CPUs, cores and memory, in the next section, three traditional parallel architectures are described. Then several general measures of parallel performance are introduced, which include speed-ups from different points of views and efficiency. In Section 2.5.1.3, parallel architectures that consist of Graphics Processing Units (GPUs) are briefly explained.

2.5.1.1 Traditional Parallel Architectures

The combinations and organizations of cores and memory lead to different parallel architectures. The common personal computers use shared memory, which means all cores on this computer gain direct access to the entire memory address space, regardless of the number of cores. Another architecture type is distributed memory, where different subsets of the cores each have their own memory. If data is required that is not stored locally to a core then exchange is needed, so cores have to communicate: this is done by sending controlled signals through a physical communication band that connects these cores. Ultimately, a hybrid that combines these two is commonly used in most supercomputers. That is, each node consists of multiple cores that share a common section of memory, however every node is linked to the other nodes and their sections of memory to form a large distributed system. Examples can be found in [26, 70]. Figure 2.11 shows the difference between shared and distributed memory architecture, as well as the hybrid of the two.

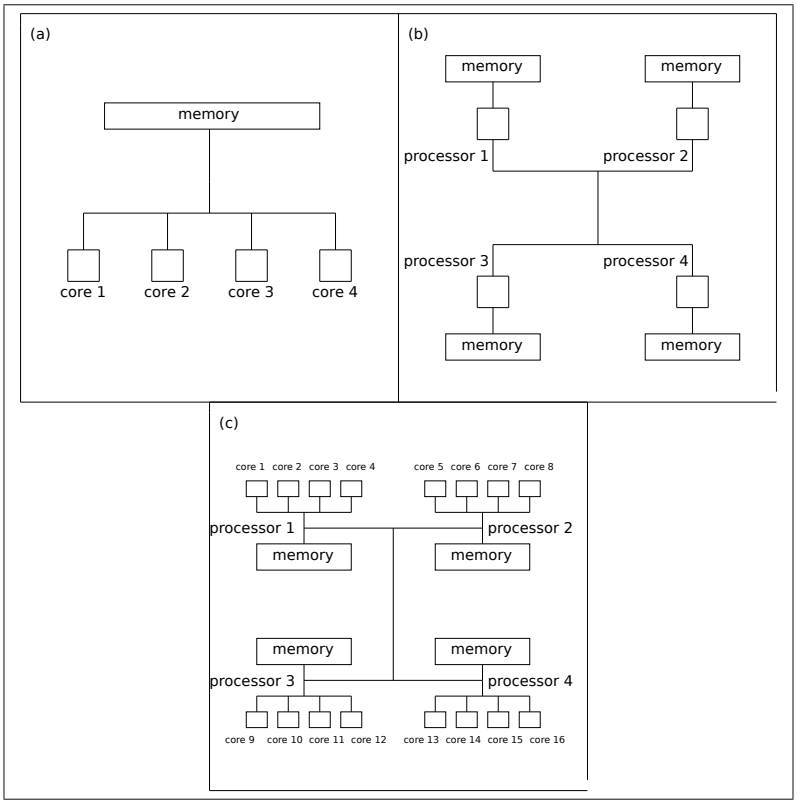

Figure 2.11: Sketch of (a) a four-core shared memory architecture; (b) a completely distributed memory architecture; and (c) the hybrid architecture.

Another classification of the parallel architectures is known as Flynn's taxonomy, which was proposed by Flynn in [74], based upon single or multiple instruction and

data streams. A stream of instructions is generated by the program execution, and the instructions operate upon the data. All three parallel architectures in Figure 2.11 belongs to the class of multiple instruction stream-multiple data stream (MIMD). They all consist of multiple interconnected processors and these processors simultaneously and independently execute different instructions on different data. In Section 2.5.1.3, a single instruction stream-multiple data stream (SIMD) is described, although its application is not considered in this thesis. Detail of the Flynn's taxonomy can also be found in [75].

In the following section, common measures of parallel performance are described.

2.5.1.2 Measures of Parallel Performance

Distributed memory architectures permit memory scalability in a natural manner as larger and larger jobs are solved on increasing numbers of cores. A further goal of parallel computing is the scalability of the execution time as the core count and/or the problem size is increased. For example, in an ideal world, when p cores are used to solve a computational job the execution time should reduce to a fraction $1/p$ of the execution time on a single core. This leads us to define the concept of speed-up [178, 186]

$$S(p) = \frac{\text{Execution time of a job using single core}}{\text{Execution time using a multicore system with } p \text{ cores}}. \tag{2.56}$$

In the ideal situation described above, $S(p) = p$. In practice this perfect speed-up can be difficult to achieve since there are overheads associated with parallel execution on p cores (e.g. synchronization points; unequal work per core, etc.).

Even more importantly however, speed-up will be less than p whenever there is a section of the code that cannot be executed in parallel. Figure 2.12 illustrates how a problem which consists of both serial and parallelizable sections has limitations on the possible speed-up from using parallel computing. It suggests that the more serial sections there are, the less efficiency can be gained through using parallel computing. This may be quantified by Amdahl's law [9], which notes that:

$$S(p) = \frac{t_s}{ft_s + (1-f)t_s/p} = \frac{p}{1+(p-1)f}, \tag{2.57}$$

where f is the fraction of the computation that must be done sequentially, p is the number of cores, t_s is the total time required if the problem is solved by single core machine, and parallel execution time, t_p, is rewritten as $ft_s + (1-f)t_s/p$. Amdahl's law assumes t_s to be constant. The speed-up factor S from Amdahl's law tends to p as $f \to 0$ and tends to 1 as $f \to 1$. Thus, to obtain good parallel speed-ups the sequential fraction, f, has to be

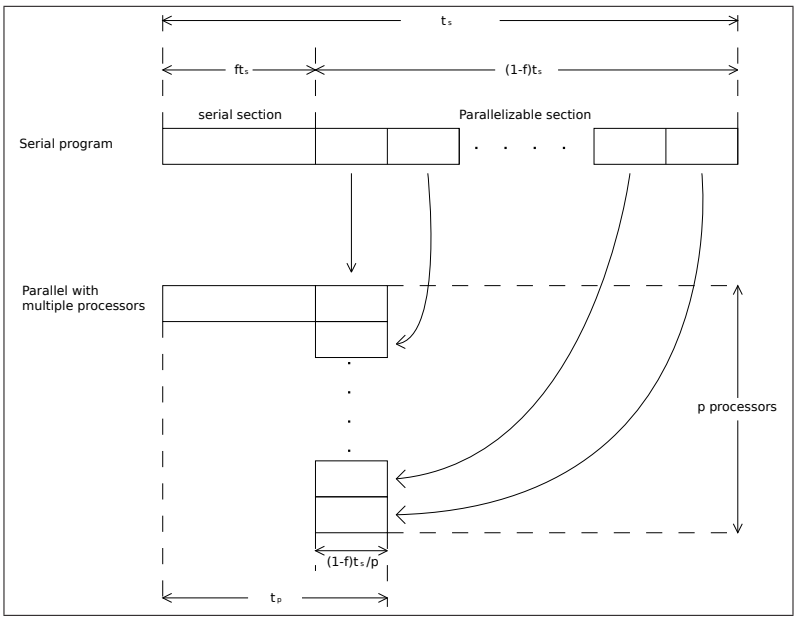

Figure 2.12: Diagram that shows how a problem with serial part at beginning gains efficiency by using parallel (Section 1.2, [222]).

very small.

From a different point of view, one may be interested in how much larger problems can be solved within a fixed time period by applying more cores. This information is provided by Gustafson's law [101]. Assumptions made by Gustafson that the parallel execution time, t_p, and serial section execution time, ft_s, are constant. Therefore, sequential execution time, t_s, is separated into $ft_s + (1-f)t_s$; it is further assumed $t_p = 1$ for simplicity. Hence the Gustafson's law is given by

$$S(p) = \frac{ft_s + (1-f)t_s}{ft_s + (1-f)t_s/p} = p + (1-p)ft_s. \tag{2.58}$$

Another assessment for parallel performance is efficiency, which indicates the average usage from all the cores [186, 222]. The efficiency, E, which is given as a percentage, is defined as

$$E = \frac{t_s}{t_p \times p} \times 100\%. \tag{2.59}$$

In practice, even when there is good speed-up with a low number of cores, most problems hit a threshold where having more cores only has a negative impact on efficiency. This generally means the problem is split into too small fractions, and that the time needed for computations on each fraction becomes dominated by the communications between them. Details about parallel communication are in Section 2.5.2. In the following sec-

tion, another parallel architecture which is built on graphics processing units is briefly explained.

2.5.1.3 Many-Core Parallel Architectures

Graphics Processing Units (GPUs) are an well-known example of the many-core architectures, or the so-called "accelerator" (e.g. Intel MIC, etc.). The GPUs were originally designed for image rendering on graphic cards. Comparing to CPUs, GPUs have relatively greater numbers of cores, however these cores are individually smaller. According to the January 2014 TOP500 [209], Titan was the "largest" supercomputer at that time (as measured by the number of Floating Point Operation Per Second (FLOPS) achieved on certain dense matrix benchmark problems) containing 18,688 GPUs in its architecture with 27 petaFLOPS. As mentioned in Section 2.5.1.1, Flynn's taxonomy classifies the architecture within each GPU as single instruction stream-multiple data stream (SIMD), which means multiple cores in one GPU simultaneously execute a single instruction on different data. To program more general applications on the GPUs, a technique that is called GPGPU can be used, which is short for general-purpose programming using a graphics processing unit [134]. An example can be found in [177].

Due to the fact that each GPU processor has relatively greater number of cores, GPU cores compute relatively faster [146], provided they have access to the necessary data. The parallel architectures that are built using GPUs tend to have more processing units, and have very large theoretical computational power. For certain tasks they can therefore allow excellent performance [134].

On the other hand, a major disadvantage of using GPUs comes from the limited memory space, and the large amount of time it takes to move data between the CPU and the GPU cache. Furthermore, because of the larger number of cores on one GPU processor, memory space per core is even smaller. Similarly, there are circumstances, such as using the FEM with a very fine resolution of mesh, where the amount of data to be moved to the GPU can be a major bottleneck. There are also circumstances that applications cannot be highly parallelized, because of inevitable serial sections, synchronization, sequential order on execution etc. These situations happen in the multigrid methods which are the main focus of this thesis (see Section 2.4). Instead of using GPUs for the multigrid methods, they can be better handled by the CPUs. In addition, as described in Section 2.3, discretization schemes and solution methods that are used in this thesis are usually considering sparse matrices, which the GPUs do not work on well. In general, it is known that the SIMD model is much more restrictive than the MIMD model (model with the use of CPUs), so fewer problems are really well suited. Due to the disadvantages that are

stated above, the many-core architectures and the use of GPUs are not considered in this thesis.

However, a possible alternative to the CPU architecture is the hybrid architecture which combines CPUs and GPUs together. This hybrid architecture executes the appropriate parts on the CPUs and runs numerically intensive and highly parallelizable parts on the GPUs. Programming languages such as CUDA and OpenCL are designed to achieve this hybrid implementation, applications can be found in [131, 230]. Some performance analyses between CPUs and GPUs can be found in [146, 149, 203].

In the following section, parallel communication is introduced.

2.5.2 Parallel Communication

The programming and architectural paradigm used throughout this thesis is based upon distributed memory and message passing, which means passing and exchanging information between cores. This is done here using a software library called Message Passing Interface (MPI). MPI was proposed and designed by a group of researchers from academia and industry, [166] is its official website. The main purpose of MPI was to create a portable and standard library for message passing: nowadays MPI and its interface are widely acknowledged as a *de facto* standard and commonly used. However, MPI only defines detailed function definitions and operation principles, not their actual implementations in software. Therefore, in order to program using MPI, a specific software library that defines the implementations of MPI protocols is required. Many such libraries exist and some of them are free to the public. For example, Open MPI and MPICH, are both widely accepted and applied to various parallel computers. The results of using these software libraries are a sequence of MPI processes. These processes can be carried out by any number of computer processors. For simplicity, here it is assumed one MPI process is allocated to one computer core.

Here a pair of basic MPI functions are demonstrated as an example to illustrate how MPI works. They are MPI_Send and MPI_Recv [98]. The former tells the process executing it to send a message that contains either one item or an array of data, to a specific destination process; the latter pauses the executing process and waits for desired data to be received. This type of communication is called blocking. There are other types of communications, for example, nonblocking routines proceed without confirming the completion of message exchange. Only the blocking routines are used in this thesis. The

syntax used to define MPI_Send and MPI_Recv is as follows:

$$\text{MPI\_Send}\,(\text{address, count, data type, destination, tag, comm})\,,$$
$$\text{MPI\_Recv}\,(\text{address, count, data type, source, tag, comm, status})\,,$$

where

- (address, count, data type) specifies the data and its size and type;
- destination specifies the process (within the given communicator) where data is sent to;
- source specifies the process (within the given communicator) where data is expected to come from;
- tag specifies the ID of message, which enables multiple data to be sent separately;
- comm specifies the communicator (essentially a pre-defined set of processes) to be used for this transmission;
- status provides information about the received message.

There are many other global operations defined by MPI to allow programming parallel code. For example, MPI_Bcast broadcasts a message from a root process to others in the communicator, and MPI_Reduce gathers values from all processes in a communicator and reduce to a single value (e.g. performing maximize) and return to a root process.

So far MPI and communications between processes have been described, however there are other forms of parallel computing that exist without message passing. One of the most common examples comes from shared memory concurrent programming via threading [79]. A thread is a sequence of instructions which cores can operate on [206]. Across multiple cores, it is possible to distribute multiple threads to achieve parallel computing. This is called multithreading. One of the distinct differences between multithreading and message passing is that within multithreading, no messages are passed between threads. It is assumed that each thread, and therefore every core, has access to the entire memory address space (i.e. parallel architecture with shared memory, see Figure 2.11(a)).

It is very inconvenient for programmers to deal with each thread and its low level interaction with computer hardware. Thus like the MPI for message passing, software libraries are generally used. For multithreading, the most widely used libraries are called Open Multi-Processing (OpenMP) and Pthreads. The use of these software libraries will break sections of code which contain the additional multithreading commands into threads for different cores to execute in parallel. Memory contention issues may arise if shared variables are involved, or if data dependencies require a certain order of loop execution, and these challenges to program using multithreading can add considerable complexity.

Since the parallel executions discussed in this thesis are exclusively undertaken with the MPI, no further details of OpenMP or Pthreads are provided here. The interested reader is referred to [17, 46], for example. As a final note, in this subsection however, architectures such as the hybrid shown in Figure 2.11(c) could be ideally suited for a combination of shared and distributed memory libraries. For example, MPI and OpenMP can be combined such that external communications between nodes is handled by MPI software, and within each node, coding can make use of OpenMP with multiple threads. An application of such a hybrid system can be found in [100], where a parallel multigrid implementation is used to simulate alloy dendrite growth. In their parallel software, both MPI and OpenMP are used, and tested on a 640-core computing cluster.

Having described parallel communication, in the next section, partitioning a discrete PDE system across multiple processes is demonstrated, and the use of message passing in the resulting parallel solver is described.

2.5.3 Mesh Partitioning in Parallel

As mentioned in Section 1.1, systems of PDEs are defined in closed regions. The use of discretization schemes such as those described in Section 2.1 allows the solutions in such regions to be approximated by a finite number of values. For each unknown value, computations for which only local values are involved are carried out in order to obtain the solution. For example, consider a cell-centred finite difference approximation to a single PDE on a two-dimensional square region. The left diagram in Figure 2.13 illustrates the four hundred unknown values within the computational domain on a 20×20 uniform mesh. If MPI is to be used to parallelize this problem then a common way of partitioning the data across the processes is to use a geometric partition (and is often referred as domain decomposition in the literature). This assigns ownership of the unknowns within each sub-region of the full mesh to a unique process in the MPI communicator. Figure 2.13 illustrates a natural geometric partition into four sub-regions, corresponding to four MPI processes, and partition edges and boundary are also identified. These sub-regions are often referred as blocks, however, a sub-region of the domain may contain multiple blocks.

The block partitioning approach used in Figure 2.13 can be further applied to larger numbers of sub-regions (e.g. 16 blocks of size 5×5), leading to more blocks with smaller size as the number of processes increases. This partition strategy eventually requires the work of load balancing, which allocates an equal number of points (or blocks) to each MPI process in order to maximize parallel efficiency. Uniform meshes certainly are easier to

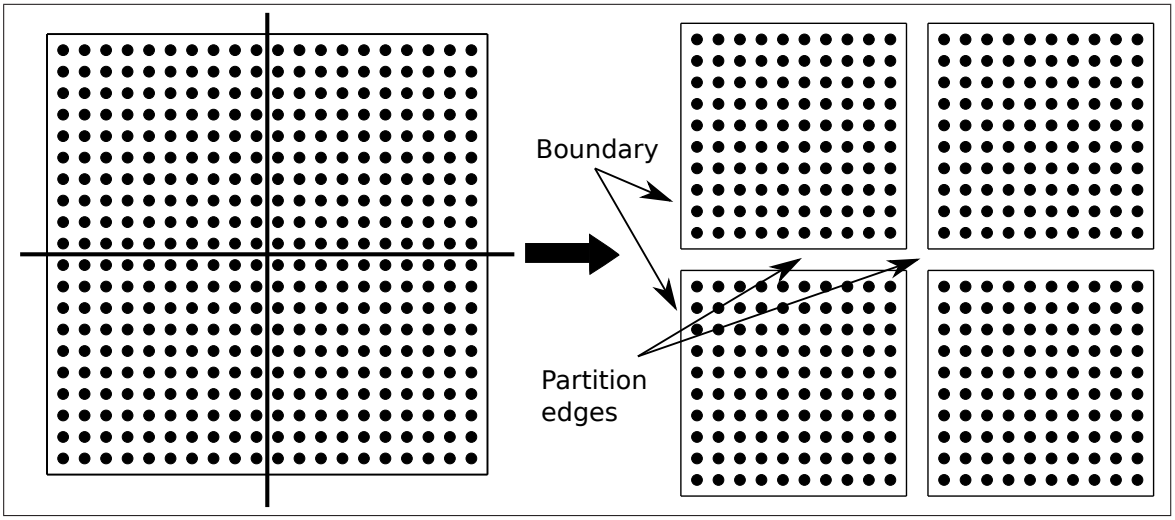

Figure 2.13: Sketch shows a 2-D uniform mesh with four hundred internal grid points split into four blocks, each with one hundred internal grid points.

achieve such a balance, however this is also required for non-uniform and unstructured meshes. This general load-balancing problem may be expressed formally as a graph partitioning problem which is believed to be NP-hard [62]. There are many software tools (e.g. Jostle [219], METIS [129], SCOTCH [181] etc.) and heuristic algorithms (e.g. greedy, recursive coordinate bisection, recursive spectral bisection etc. [135]) that have been proposed over the years, aimed at finding "good" partitions (though, generally these are not optimal).

Having partitioned the unknowns in the mesh across the MPI processes, the next step is to compute the discrete solution in parallel. Here an iterative solution process in which the update value at each grid point depends upon its four neighbours is considered (e.g. the point-wise Jacobi iteration that is presented in Equation (2.22) with the use of the five-point stencil from Equation (2.3)). Updates of interior points can be carried out easily, however near partition edge points will require data from one or more neighbouring blocks which are held by other processes. To fulfill this requirement, MPI is used to facilitate message transmission between processes.

To acquire values from neighbouring processes, the concept of ghost cells which is introduced in Figure 2.2 for representing boundary points on cell-centred grids can be extended. In practice, the most common approach is to introduce an extra layer of guard cells around each block of the partition. Figure 2.14 illustrates the use of one layer of guard cells (marked as $\circ$) on one block of the partition from Figure 2.13.

In Figure 2.14, these guard cells at partition edges are used to store the values of grid points from neighboring blocks; and curved lines are used to link these guard cells with

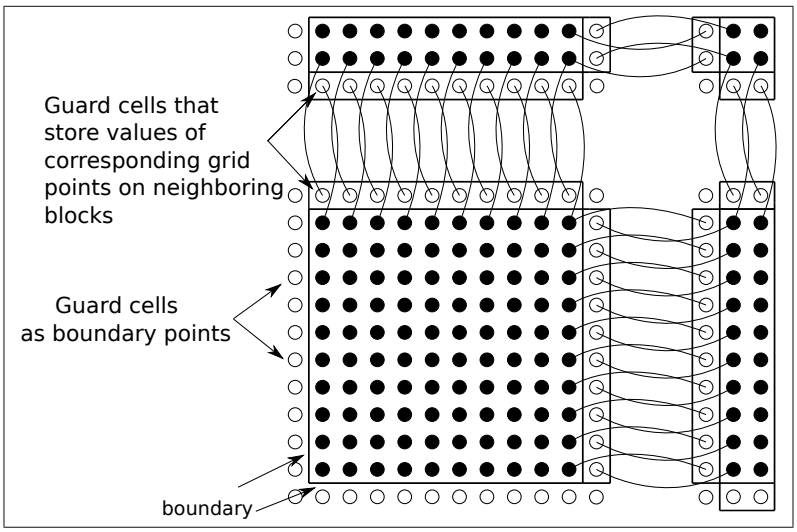

Figure 2.14: Sketch shows the role of guard cells (marked as ∘) in a 2-D block partition; guard cells which are used as boundary points and as duplication of grid points from neighboring blocks are illustrated; for these guard cells that are near the partition edge, curved lines are used to identify their corresponding grid points on neighboring blocks.

their corresponding internal grid points from neighboring blocks. At the actual boundary, guard cells are used as the ghost cells for representing boundary points. For the standard five-point stencil in 2-D, or equivalent seven-point stencil in 3-D, one layer of guard cells is sufficient. Larger stencils may require additional layers of guard cells and/or additional messages to be sent. In either case, this leads to communication of more data.

The idea of load balancing which is described earlier suggests the size of block should be as small as possible, especially for non-uniform meshes, so it can be flexible to manage an equal workload on each MPI process. However, the choices of block size and the size of guard cells become a trade-off. If the block size is too small the number of guard cells becomes overwhelming. For example, a 4×4 block (where 4 is the number of internal grid points in x and y directions) has more guard cells (i.e. 20) than internal points (i.e. 16). These guard cells do not contribute to the computation directly and require a large amount of time to update. The choice of block size in applications of this thesis is further discussed in Chapter 3.

In [200], Shin et al. described a parallel multigrid method that is used to solve the Cahn-Hilliard equations with finite difference and stabilized splitting schemes. Only uniform grids were considered, and they applied the same idea of having a layer of guard cells around mesh partitions to help the communication between processes. Likewise, Guo et al. in their parallel multigrid solver [100], also make use of guard cells around uniform mesh blocks to achieve data transmission. A different example can be found

in [80], where Gaskell et al. partition a rectangular domain into a strip of equal-sized rectangular sub-blocks and distributed each to one process.

The guard cell approach may also be applied to other than square (cubic) meshes. For example, Heikes et al. [110] discuss a multigrid solver with meshes that consist of hexagonal and triangular grids. Other examples of partitioning unstructured meshes can also be found in [87, 220]. One describes a multilevel unstructured mesh and its parallel optimization with three algorithms. Another partitions hierarchical hybrid grids with an unstructured mesh generation package.

The above discussion refers to the partitioning of a single mesh (static mesh partitioning) however the problems studied in this thesis are time-dependent and the mesh will be adapted dynamically to the moving features of the solution. In the following section, combining spatial adaptivity on a structured mesh with parallel partitioning is described and balancing the workload in parallel using dynamic load balancing is discussed.

2.5.4 Spatial Adaptivity in Parallel and Dynamic Load Balancing

In the previous section, partitioning the domain into a number of sub-regions is described. Inside each sub-region, there may be one or many mesh blocks. The spatial adaptivity which is previously discussed in Section 2.2.1, can be achieved in parallel by allowing each MPI process to adapt those mesh blocks contained in its sub-regions of the domain. Some problem-specific, user-defined, refinement criteria that are mentioned in Section 2.2.1 can be applied to the individual mesh blocks. Note that some communication may be required when a block is refined or coarsened since this has an impact in neighbouring blocks (which may not be owned by the same process).

When a block data structure is used then the natural way to achieve the spatial adaptivity is by refining the whole mesh block. In this case, mesh validity is preserved by ensuring there is no more than one refinement level difference between adjacent blocks.

To determine whether a mesh block needs adapting, solutions on all the grid points in that block are considered. The refinement criteria (based upon error estimation, the actual values of certain variables and/or the magnitude of the gradient of variable values) need only be satisfied for a single cell in the block, whereas the coarsening criteria must be satisfied for every cell in the block. In either situation the block is marked for possible refinement or coarsening. The marking procedures that are undertaken in this thesis are described in the next chapter in detail.

After the marking procedures, mesh blocks may be refined and/or coarsened accordingly. The refining and coarsening processes that are described previously in Sections

2.2.1.1 and 2.2.1.2 can be performed on all grid points in a block, and correct guard cells are assigned. An example is shown in Figure 2.15, when the refining process is applied to the 10×10 coarse block on the left, the resulting 20×20 mesh is composed of four 10×10 fine blocks, as illustrated on the right. Furthermore, the coarsening process turns the fine grid on the right to the coarse grid on the left by replacing all four 10×10 blocks with a single 10×10 coarse block. This will only occur if all four fine blocks are marked for possible coarsening. If the adaptivity is carried out dynamically during the simulation, then interpolation and restriction operators are needed for transferring values between refinement levels. As for the example shown in Figure 2.15, operators from Equations (2.12) and (2.13) can be used to transfer values of these cell-centred grid points.

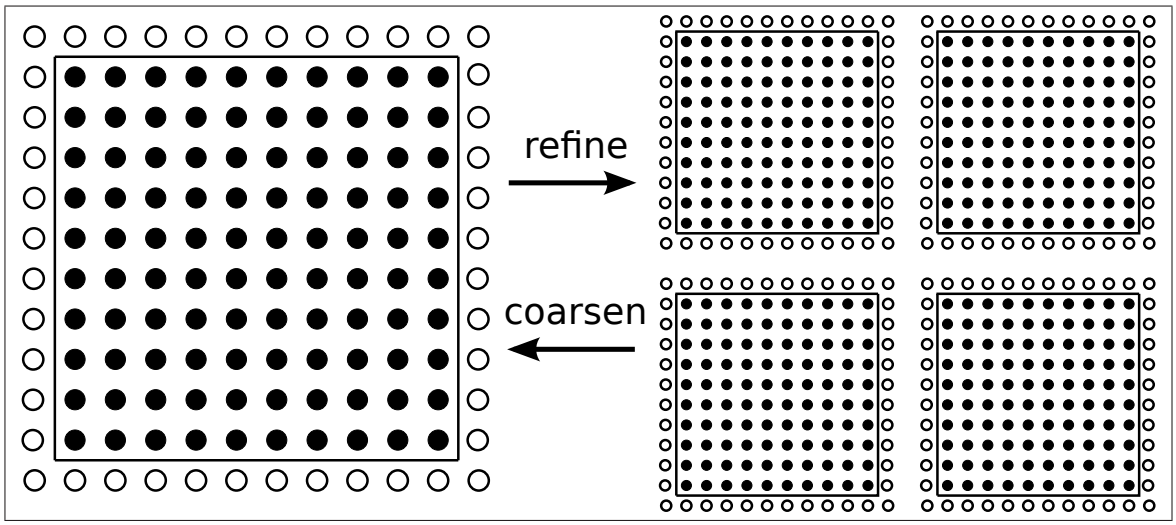

Figure 2.15: Sketch shows the refining process that transfers a 10×10 block on the left to four 10×10 blocks on the right, and the coarsening process turns all four blocks on the right to the single 10×10 block on the left.

When the adaptive process is finished, the guard cells need to be updated. This is to ensure the guard cells are containing the correct values from corresponding grid points. The previous example in Figure 2.14 shows the update process in a uniform grid situation. However, if two adjacent blocks have one level difference in refinement, some special operators are required. In Chapter 3 Section 3.2.4, a specific procedure that deals these situations is introduced, which is applied in this thesis.

An important additional challenge that occurs during the dynamic adaptive process in parallel is balancing the workload. As mentioned previously, when solving parabolic problems, the solution may evolve through time. When the adaptive process takes place, there are MPI processes where a number of new mesh blocks are generated and possibly other processes where blocks are removed. Consequently, after an adaptive step, even if

each process had an equal number of blocks to begin with, they will not generally retain the same number of blocks as each other. There is therefore a need to re-distribute some of the blocks between processes. This procedure is termed dynamic load balancing.

Even when all MPI processes receive roughly same number of blocks, there may be further challenges. For example, all blocks in one MPI process are preferred to be around a same geometric location. In this way, it saves a large amount of time for synchronising the layers of guard cells. In contrast, if one MPI process has its blocks all scattered over the domain, then almost all the values of guard cells require communication with other MPI processes. Hence dynamic load balancing seeks to improve the partition subject to this low communication objective. Ideally this will be achieved with a minimal amount of blocks being relocated as well. These constraints cannot all be satisfied but there are some heuristics which aim to find good new partitions at low overhead (e.g. based upon the use of space-filling curves or parallel version of METIS [197]).

In the following section, the parallelization of the multigrid algorithm is introduced.

2.5.5 Multigrid Algorithms in Parallel

Details about the multigrid algorithm and its variants are discussed in Section 2.4. These multigrid methods are previously described from a purely numerical point of view. In this section, these multigrid methods are also discussed from a parallel point of view.

The multigrid algorithm, as a whole, is not considered to be fully parallelizable in terms of parallel computing. This is because computations on each grid have to be carried out in a sequential order. In addition, the use of a hierarchy of grids imposes a different degree of potential parallelism on each level of grid [216]. On the other hand, each of the main components of the multigrid algorithm may be parallelized. Here, we consider the parallelization of multigrid methods with the use of block mesh partitioning that is previously explained in Section 2.5.3.

The main components of the multigrid algorithm can be divided into three categories. The first category includes the solution methods. They are used as the pre-smoother, the post-smoother and the coarse grid solver in the multigrid algorithm. Since only iterative methods (e.g. the linear point-wise Jacobi, the nonlinear point-wise Jacobi and the nonlinear block version of Jacobi for a system of PDEs, they are presented in Equations (2.22), (2.34) and (2.36) respectively) are used in the multigrid algorithm in this thesis, we are able to neglect other variants (e.g. Gaussian-Elimination with LU factorization as the coarse grid solver). The iterative methods, combined with a discretization scheme that is a local approximation (e.g. the FDM), update each grid point only based upon local values.

The operations in this category require neighbour-to-neighbour MPI communications to update the guard cells, typically, one update after each iteration for a Jacobi solver.

The second category includes a highly parallelizable component of the multigrid algorithm. This component is the residual calculation. The calculation can be performed independently on each block of grid points, providing both up-to-date values of the RHS vectors f and left-hand side Au (or $A(u)$ if the problem is nonlinear) are stored. Otherwise, f and Au (or $A(u)$) need to be re-calculated, and this may require MPI communications if guard cells are not up-to-date.

The third category includes grid transfer operators, they are the restriction and the interpolation. These operators may require guard cells (e.g. the bilinear interpolation presented in Equation (2.12)). Thus neighbour-to-neighbour communications are required prior to these operators. During the restriction and the interpolation, if the coarse block and fine blocks are owned by the same process, then no further communication is required. However, in general, communications to transfer blocks of data between processes are required. After values are interpolated via the interpolation operator, coarse grid correction can be carried out within each individual block.

One issue which may arise using this block mesh partitioning is when the number of MPI processes is greater than the number of partitions (e.g. mesh blocks) on the coarsest grid. These extra MPI processes may share workload from the finer grids, but on the coarsest grid, they become idle. This deteriorates the parallel efficiency, and is well known as the bottleneck of the multigrid methods in parallel computing.

There are two possible solutions to this issue [216]. The first one is to avoid very coarse grids. Shin et al. in [200] implemented this solution to their parallel multigrid solver. An extra stopping criterion is added, so that when the criterion is satisfied on a coarse grid, then no coarser grids are visited. The second solution is from [111], Hempel and Schuller suggest these coarser grids which are not suitable for parallelization are assigned to one or a few MPI processes. These finer grids are parallelized normally. This is termed agglomeration.

Having described the parallelizations of the main components from the general multigrid algorithm, there are components which only belong to specific methods. They can be parallelized as well. For the linear multigrid, the specific component is the initialization of the error approximation e on the coarse grids, which is shown in Algorithm 3 at line 4. This component is highly parallelizable and does not require MPI communications.

For Newton multigrid, the calculations for the Jacobian matrix and the RHS vector at lines 2 and 3 in Algorithm 4 generally need values from the guard cells. Then the linear multigrid solver can be parallelized accordingly. As mentioned previously in Section

2.4.3, the Jacobian matrices on coarser grids are computed by using the ones from finer grids (e.g. $A^{2h} = I_h^{2h} A^h I_{2h}^h$). However, this may be performed through the standard restriction and interpolation operators. Other specific components in Algorithm 4 can be carried out without the MPI communications. They include the initializations at lines 1 and 4, the Newton's method correction at line 6 and modifying the iteration number at line 7.

For the nonlinear FAS multigrid, calculation of the modified RHS may require values from the guard cells. Thus MPI communications are needed for updating the guard cells. On the other hand, calculation of the error approximation on the coarse grids can be done without MPI communications. If it is a system of nonlinear PDEs, then calculations need to be applied to all variables. Therefore, the amount of MPI communications grows accordingly. These two special components correspond to lines 5 and 7 in Algorithm 5. The adaptive multigrid with the use of MLAT is based upon the nonlinear FAS multigrid. It can be parallelized as a nonlinear FAS multigrid, providing mesh blocks and the temporary Dirichlet boundary points are handled correctly. In the following chapter, a software tool that is used to implement some of the described scientific techniques is introduced.

Chapter 3

An Overview of the Software Tools

In the previous chapter, a number of scientific computing techniques of relevance to the subsequent research in this thesis are described. In order to obtain implementations of a number of these techniques, a software tool has been developed in the Scientific Computing group at the University of Leeds, and is introduced in this chapter. It is called Campfire, and is further dependent upon a software library for mesh generation and adaptivity, called PARAMESH [180]. In Section 3.1, brief overviews on the development of PARAMESH and Campfire are introduced. PARAMESH is then described in more detail in Section 3.2. Its adaptive mesh refinement and data structures are explained in Section 3.2.1 and Section 3.2.2 respectively. Grid transfer operators and dynamic load balancing in PARAMESH are discussed in Section 3.2.3. Then an important component in PARAMESH, known as guard cells, and their update subroutines are described in Section 3.2.4. In Section 3.3, Campfire is described. A significant modification to PARAMESH associated with the ordering and the parallel distribution of mesh blocks is explained in Section 3.3.1. Then adaptive mesh refinement and adaptive time stepping in Campfire are summarised in Sections 3.3.2 and 3.3.3 respectively. We conclude this chapter with a description of the adaptive multigrid solver that is implemented in Campfire in Section 3.3.4.

3.1 Introduction to PARAMESH and Campfire

The software that is used as the starting point for this thesis, Campfire, is dependent upon a software library, which is called PARAMESH [180]. PARAMESH was developed at NASA Goddard Space Flight Center and Drexel University under NASA's HPCC and ESTO/CT projects, and under grant NNG04GP9G from the NASA/AISR project. Its current website can be found in [180], including several useful guides for programming PARAMESH. PARAMESH itself is an open source software and can be downloaded from [179]. It is programmed using Fortran 90. The implementation provides a user-friendly interface, where only the application-specific routines are exposed to the user by default. This software library generates structured Cartesian meshes with the use of a block partitioning strategy (see Section 2.5.3), and obtains spatial adaptivity by having multiple layers of mesh refinements. This is termed adaptive mesh refinement (AMR) and is introduced previously in Section 2.5.4. Parallelism is achieved through distributing mesh blocks to multiple MPI processes, and using MPI to communicate between individual MPI processes to exchange data (MPI communication is previously described in Section 2.5.2). Applications that are implemented using PARAMESH and additional related information can be found in [160, 171, 172].

There were intentions to implement the multigrid methods in PARAMESH while it was developed at NASA. However, only a test version of the linear multigrid method was considered. On the other hand, the AMR certainly provides the potential functionalities to obtain a hierarchy of grids that the geometric multigrid algorithm needs (such as shown in Figures 2.7 and 2.10). Therefore, a 2-D adaptive multigrid solver with FAS and MLAT technique was developed by Green within the Scientific Computing group at the University of Leeds. The resulting program is used to solve phase-field models arising from the field of solidification, and is described in [93]. It also provided a provisional implementation for 3-D problems.

Later on, Green and Goodyer extended the 2-D solver to tackle more computationally challenging phase-field models in 3-D. The resulting program was repackaged by Goodyer into today's Campfire. Its application can be found in [91]. In the following sections, main procedures that are provided in PARAMESH are explained. However, before we proceed, it is worth noting there are alternative software packages. To name a few, DUNE, DEAL.II and AFEPack all offer adaptive mesh generation and parallelism. However, these software packages are mainly used with FEM and unstructured meshes. DUNE and AFEPack have multigrid methods implemented. The interested reader is directed to their websites [4, 57, 67] for more information.

3.2 PARAMESH

In this section, main procedures and components that are provided by the PARAMESH library are explained. In Section 3.2.1, AMR in PARAMESH is described. Its associated data structures are explained in Section 3.2.2. When dynamically adapting the meshes, grid transfer operators are typically needed. These operators (i.e. restriction and interpolation) are previously explained in Sections 2.2.1.1 and 2.2.1.2. In Section 3.2.3, these procedures and the associated issues with dynamic load balancing in PARAMESH are described. As mentioned in the previous section, PARAMESH uses a block partitioning strategy, and this is generally associated with the intrinsic use of guard cells. This is previously described in Section 2.5.3. In Section 3.2.4, a PARAMESH subroutine which updates guard cells globally in a parallel environment through the use of MPI communication is explained in detail.

3.2.1 Adaptive Mesh Refinement

PARAMESH defines its meshes as the union of mesh blocks, and this strategy is previously described in Section 2.5.3. In this section, mesh generation and AMR in PARAMESH are explained.

In PARAMESH, Cartesian meshes are created by combining structured mesh blocks together. These blocks may be uniformly refined (one example is shown from left to right in Figure 2.15) to generate finer meshes. If the refinement is only carried out locally, the adaptive meshes are obtained. The Cartesian meshes that are generated using PARAMESH consist of 2-D square/rectangular or 3-D cubic/cuboid blocks of cells. This may result in different mesh geometries and topologies. For example, in Figure 3.1(a), a 2-D mesh consists of 2×2 blocks and each block is a rectangle. This figure illustrates square topology but not square geometry. On the other hand, in Figure 3.1(b), another 2-D mesh consists of 3×2 blocks and each block is also a rectangle. This figure shows square geometry but not square topology.

In PARAMESH, each block has the same number of grid points, even when they are in different levels of refinement. This is the so-called logically identical size but the sizes of blocks may be geometrically different. For the purpose of demonstration, let's consider 2-D meshes which are square geometry and square topology. In Figure 3.2, a 2-D Cartesian grid is shown, which has a block size of 2×2. It is worth noting that this block size is only for demonstration, and is not practical. The reason is caused by its associated guard cells, and this is described previously in Section 2.5.3. In Figure 3.2, each block is enumerated from number 1 to 17, with a specific ordering strategy from PARAMESH, that is called

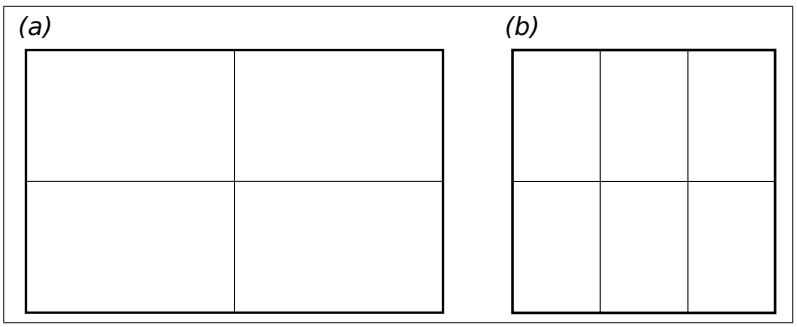

Figure 3.1: Sketch shows (a) a 2-D mesh consists of 2×2 blocks, this mesh is therefore square topology but not square geometry; (b) another 2-D mesh consists of 3×2 blocks, and it square geometry but not square topology.

Morton order [180]. These indices of blocks are positioned near the centre of each block. The heavy lines in the grid indicate the boundaries of each block, and the lighter lines indicate individual grid cells. Block 1 is the coarsest grid that covers the whole domain. Blocks $4, 5, 6$ and 7 have the finest refinement level. Although it is not necessary to just use one block as the coarsest grid. Grids that are shown in Figure 3.1 also can be used as the coarsest grids. Other blocks are further refined in local regions, and they have the same logically identical size (i.e. 2×2). It is worth noting in Figure 3.2, guard cells are neglected, although in practice, each of these 2×2 blocks will typically have a layer of guard cells. PARAMESH permits multiple layers of guard cells, however this is not applied in this thesis. We refer back to Figure 2.14 for the description and implementation of guard cells.

Although the example shown in Figure 3.2 is in 2-D, generating a 3-D mesh with cubic/cuboid blocks is straightforward. In the following section, the data structure that is associated with the adaptive meshes in PARAMESH is described.

3.2.2 Data Structure

In PARAMESH, refining one block generates a fixed number of new "children" blocks. In 2-D, there are four newly-generated blocks, and there are eight "children" blocks per "parent" block in 3-D. Let's consider the example mesh shown in Figure 3.2. A tree structure can be used to describe this example adaptive mesh. It starts with the root block which represents the coarsest grid. Then each local refinement generates new branches. This tree structure is presented in Figure 3.3 which is corresponding to the example mesh shown in Figure 3.2, and it is called a quad-tree. If cubic/cuboid mesh blocks from 3-D are considered, the data structure becomes an oct-tree. Each node in this tree structure represents a mesh block, the root of the tree is the coarsest block 1. As already noted, all

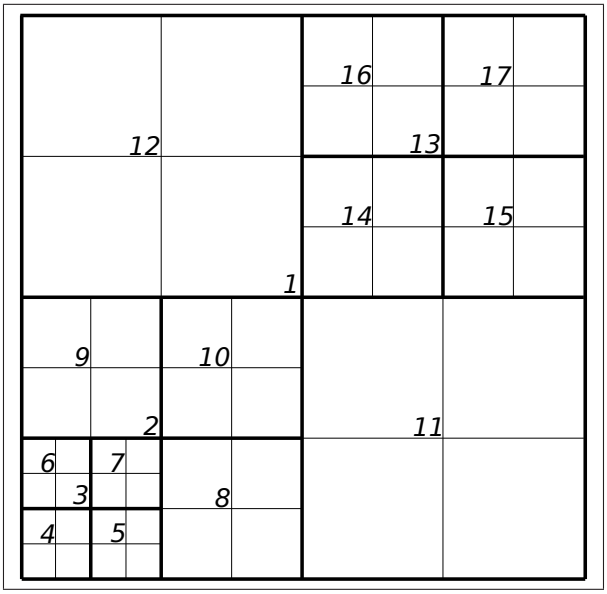

Figure 3.2: Sketch shows a 2-D Cartesian grid with a block size of 2×2, blocks are enumerated from number 1 to 17, with a specific ordering strategy from PARAMESH, that is called Morton order [180], the heavy lines in the grid indicate the boundaries of each block, and the lighter lines indicate individual grid cells.

blocks have the same logical size. The indices of blocks are presented next to each node, and are used to show the "parent-children" relations between these blocks.

Furthermore, let's consider distributing this tree structured data to multiple MPI processes to allow parallelism. Four different shapes (i.e. $\square$, $\triangle$, $\bullet$ and $\circ$) are used to indicate four MPI processes. In Figure 3.3, four MPI processes all receive near-optimal number of blocks, and this balances the workload. PARAMESH also aims on maximising block locality. This means each MPI processes indicated in Figure 3.3 has as many as possible blocks from the same "parent-children" relation.

To store the data from these mesh blocks, arrays are typically employed. The main arrays that are used in PARAMESH for storage are the so-called "unk" array. This "unk" array is used to store values for cell-centred grid points, and the guard cells. Other types of grid points are supported by PARAMESH, such as face-centred and edge-centred. However, these are not used in this thesis, thus no further discussions are made here.

When mesh blocks are parallelized to different MPI processes, these "unk" arrays are split up and follow their corresponding blocks. Thus they are highly parallelizable, just as these mesh blocks. The "unk" arrays are multi-dimensional, and also suitable for multiple dependent variables. They are given by

$$\mathrm{unk}(var, i, j, k, lb), \tag{3.1}$$

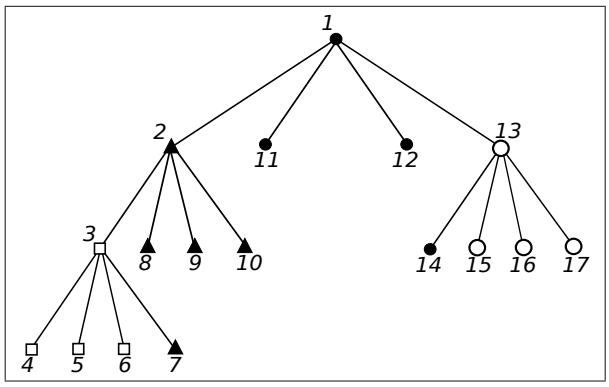

Figure 3.3: Sketch shows a quad-tree [180] with indices of blocks from the corresponding mesh shown in Figure 3.2, furthermore, four different shapes (i.e. □, △, • and ○) are used to indicate a possible distribution in a parallel environment to balance the workload.

where $var = 1, \ldots, \mathscr{K}$ indicate the indices of dependent variables, i, j, k represents each grid point in a block. These points are ordered row by row (such as shown in Figure 2.6), and lb denotes the indices of blocks. It is the lb dimension of the array that is partitioned across the MPI processes therefore.

PARAMESH includes a number of supporting data structures. There include data structures used to hold the "parents-children" relations between blocks. Some others serve as temporary data storage. Two additional data structures are worth noting. They are two logical arrays which are associated with each mesh block. They are used to indicate whether this block has been marked as a target for refining or coarsening, and are given by

$$\begin{aligned} \text{refine}(lb) &= \textit{true} \text{ or } \textit{false}; \\ \text{derefine}(lb) &= \textit{true} \text{ or } \textit{false}. \end{aligned} \tag{3.2}$$

These two arrays are used in a marking process in AMR, which typically depends upon a problem-specific, user-defined, criterion. There are data that are associated with other types of grid points. However, these data are not used in the work described in this thesis. This is because cell-centred discretization schemes are used throughout.

The described AMR may be dynamically carried out in PARAMESH. However, there are two issues associated: grid transfer and dynamic load balancing. In the following section, these two issues are discussed.

3.2.3 Dynamic Load Balancing and Grid Transfer Operators

One of the issues that arises from applying AMR during computations in a parallel environment is dynamic load balancing. This is generally described in Section 2.5.4. Let's consider an example which requires the mesh shown in Figure 3.2 to be adapted dynamically. More specifically, block 4 is refined while blocks $14, 15, 16$ and 17 are coarsened. The resulting new adaptive mesh is shown in Figure 3.4. Note these adaptive changes do not require preservation of mesh validity, so they are chosen deliberately for the demonstration and discussion made here.

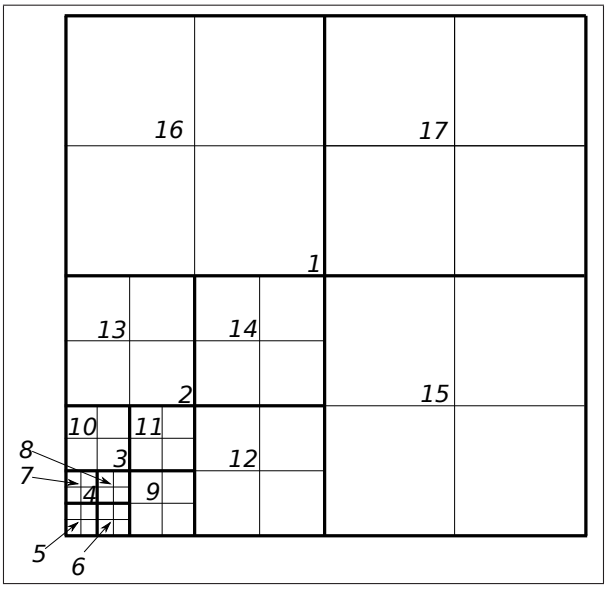

Figure 3.4: Sketch shows the resulting mesh if the adaptive mesh demonstrated in Figure 3.2 is required to refine block 4 and coarsen blocks $14, 15, 16$ and 17.

Due to these adaptive changes to the mesh, the Morton order used by PARAMESH re-enumerates the blocks, and shuffles the parallel distribution so it maintains a near-optimal workload to each MPI process and maximises the block locality. This is shown in Figure 3.5, where a quad-tree indicates the corresponding tree structure from the mesh shown in Figure 3.4.

PARAMESH manages the dynamic load balancing very aggressively, and seeks to balance the entire tree. In Figure 3.3, in order to balance the workload, PARAMESH permits more than one MPI process to share from a single branch in the tree structure. In Figure 3.5, due to the changes made, blocks are shuffled. The MPI process represented by shape $\circ$ receives all the new blocks which are from a new branch in the tree. In this way, the workload can be distributed near optimally and the block locality is maximised. The trade-off is a large number of data may be moved around between MPI processes every

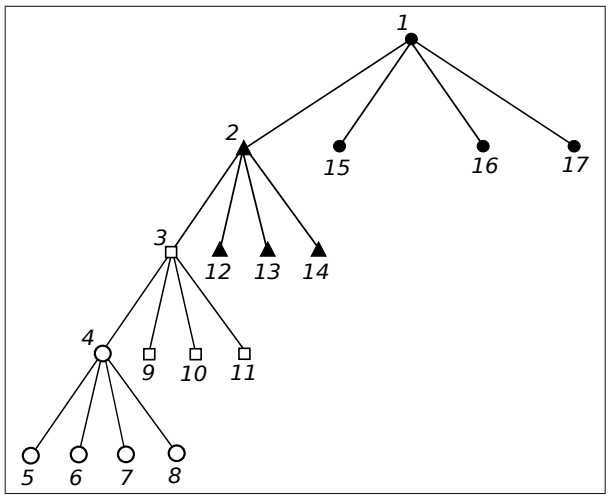

Figure 3.5: Sketch shows a quad-tree with indices of blocks from the corresponding mesh shown in Figure 3.4, which came from the previously defined mesh in Figure 3.2. The changes made to Figure 3.2 were refining block 4 and coarsening blocks $14, 15, 16$ and 17. Furthermore, four different shapes (i.e. $\square$, $\triangle$, $\bullet$ and $\circ$) are used to indicate a possible distribution for four MPI processes in a parallel environment.

time when AMR is carried out dynamically. The dynamic load balancing and the Morton order are automatically maintained within the subroutines of AMR in PARAMESH.

In order to dynamically adapt meshes during the computation, grid transfer operators are typically required. When refining a mesh block, values are required to be interpolated to these newly-generated blocks. On the other hand, restriction is needed before coarsening blocks. Similar to the description stated in Section 2.2.1, in order to obtain one block from coarsening, four neighbouring blocks are required in 2-D and eight neighbouring blocks are required in 3-D. This is previously described in Section 2.2.1. Refining and coarsening are discussed in detail, along with an interpolation and a restriction operator in Sections 2.2.1.1 and 2.2.1.2, respectively. These operators are presented in Equations (2.13) and (2.12), and may be applied to these cell-centred grid points. The preservation of mesh validity is also maintained in PARAMESH, so that between adjacent blocks, there is only one refinement level difference.

As mentioned before, PARAMESH supports other types of grid points as well. There are many grid transfer operators that are associated with these types of grid points. However, only cell-centred data is considered in this thesis. Therefore, grid transfer operators for other types of grid points are not discussed here.

The restriction only requires grid points but not guard cells. However, for the interpolation operator, when applied on cells which are near the block edges, values of guard cells are required. Therefore, in order to carry out a valid interpolation guard cells should

be updated prior to the interpolation operator. The subroutine which updates guard cells in PARAMESH is described in the following section. Note that this update for guard cells is used in many other places of a typical solver and so is of great importance within PARAMESH.

Before we proceed, it is worth noting there are alternative grid transfer operators for cell-centred grid points implemented in PARAMESH. For example, injection-type operators. These operators typically perform one-to-one transfer. Therefore, their orders of accuracy are lower than the described bilinear/trilinear interpolations and cell averaging restriction. There are also templates in PARAMESH for the user to define other transfer operators which consider many more grid points at the source of the transition. However, this requires multiple layers of guard cells. In the following section, guard cells and their update in PARAMESH are explained.

3.2.4 The Role of Guard Cells and Its Updating Subroutine

One of the issues which is associated with mesh partitioning (also called domain decomposition) in a parallel environment is that blocks are generally required to exchange data with their neighbouring blocks. The use of discretization schemes with local approximations (e.g. FDM and FEM) are typical examples. Instead of exchanging data a grid point at a time, guard cells which surround each mesh block are employed to store values of corresponding grid points from neighbouring blocks. This is previously demonstrated in Figure 2.14 and discussed in detail in Section 2.5.3. It is worth noting that because it is a cell-centred grid, some of these guard cells are used to apply boundary conditions (see Figure 2.2).

PARAMESH introduces the guard cells to all of its mesh blocks regardless of whether or not they are on the same MPI process. Typically, one layer of guard cells is used, and this already satisfies the use of the standard five-point stencil in 2-D that is presented in Equation (2.3) and the seven-point stencil in 3-D. It is also enough for the bilinear interpolation operator which is presented in Equation (2.12). If larger stencils are required, such as the nine-point stencil in 2-D which is shown in Figure 2.1, then either multiple updates or multiple layers of guard cells are needed. In PARAMESH, an important subroutine that updates these guard cells is called “guard cells update” (GCU).

The GCU by default performs a global synchronisation to all guard cells of all types of grid points from all refinement levels. This process is very expensive to be carried out in terms of efficiency. One GCU per iteration is sufficient for Jacobi iterative method, if a Gauss-Seidel method is needed, then many more GCUs are required in order to update

the "parents" blocks with the most up-to-date values (though this can be reduced through the use of a red-black ordering). Although the Gauss-Seidel method generally converges quicker than Jacobi, since GCU is very time-consuming, it may not be the most efficient choice. This is especially true when the scheme is being used as a smoother rather than a solver. The choice of smoother in the multigrid solver that is implemented in Campfire is described later in Section 3.3.4.

When a grid is adaptive (i.e. there are multiple refinement levels), an issue arises near the edges between two different refinement levels. There are approaches that directly deal with this issue by using some averaging operators on grid points from the fine refinement. However, in PARAMESH, GCU includes a restriction functionality which uses the approach that is shown in Equation (2.13). This restriction is carried out whenever GCU is called, and it restricts the values from current blocks to all their parent blocks. In this way, at the edges between two different refinement levels, the coarser block can directly interact with the "parent" block of the finer blocks.

There is another issue associated with guard cells. That is the choice of block size. As mentioned in Section 2.5.3, it should be as small as possible in order to obtain a flexible AMR, as well as a flexible distribution in parallel. On the other hand, the guard cells themselves do not contribute directly to the computation. So having a large number of guard cells only deteriorates the efficiency of MPI communication and also imposes a heavy burden on the memory storage. This is previously discussed in Section 2.5.3. The choice of block size is 8×8 for 2-D grids, of which there are a further 36% guard cells with just a single layer in that block. In 3-D grids, the choices is $8 \times 8 \times 8$ which yields 48.8% of guard cells. With a larger block size the proportion of guard cells is much reduced (e.g. less than 20% with a $32 \times 32 \times 32$ block in 3-D).

To summarise the GCU subroutine in PARAMESH, it firstly identifies the settings for guard cells. The "parents-children" relations between all blocks are re-established (as the tree structure shown in Figure 3.3), in case of changes made through AMR. Then the solution from cell-centred grid points on the "children" blocks are restricted to all "parents" blocks. After the restriction, for each block, the correct values of guard cells from corresponding neighbouring blocks are obtained through MPI and stored in temporary arrays. When all blocks are visited, the guard cells can then be updated using these values in the temporary arrays. Finally, boundary conditions are re-imposed to these guard cells which are used as boundary points on cell-centred grids. Algorithm 7 illustrates the GCU subroutine. MPI communications may be required in steps 2, 3 and 4.

Stencils and grid transfer operators which require multiple layers of guard cells are not considered in this thesis for three reasons. First of all, the requirement for additional

Algorithm 7 Guard cells update

1. Initialise settings for guard cells
2. Re-establish "parents-children" relations of all mesh blocks
3. Restrict solution on "children" blocks to all "parent" blocks
4. Obtain correct values of guard cells for all mesh blocks in temporary array
5. Update guard cells using values from the temporary array
6. Re-impose boundary conditions to the guard cells that are positioned outside the boundary.

layers of guard cells imposes a further challenge on the choice of block size, and corresponding memory usage. Secondly, the GCU subroutine in PARAMESH is the most time-consuming component, adding more guard cells or increasing the number of GCU calls makes the situation overwhelming. Finally, large stencils and transfer operators typically require the problem solution to be as smooth as possible, and high order solutions frequently exhibit oscillations near very steep fronts where the larger stencils are not desirable.

In the following section, the main procedures that are undertaken in Campfire are described.

3.3 Campfire

In this section, the main procedures and components that have been implemented in Campfire [91] are described. In Section 3.3.1, a major modification to the Morton order that is used in PARAMESH is introduced. Then procedures of AMR and adaptive time stepping in Campfire are summarised in Sections 3.3.2 and 3.3.3 respectively. Finally, the adaptive multigrid solver which is implemented in Campfire [91] is described in Section 3.3.4.

3.3.1 New Ordering and Partitioning Strategy for Data Structure in Campfire

Campfire inherited most of the data structures and mesh generation functions from PARAMESH. On the other hand, different from PARAMESH, Campfire is designed for a multigrid solver. Such a solver is required to perform computations on each individual grid before moving to another. Even when an adaptive grid is used, such as the one presented in Figure 3.2, computations are done sequentially from one refinement level to another.

The Morton order from PARAMESH such as shown in Figure 3.3 enumerates mesh

blocks in a depth-first manner, which does not always lead to a good partition for a multigrid solver. Therefore, a new ordering strategy is implemented in Campfire. It is called CEG. It enumerates mesh blocks sequentially from the coarsest grid to the finest grid. Let's considering the example of Morton order shown in Figure 3.3, if the CEG order is used instead, Figure 3.6 demonstrates the results.

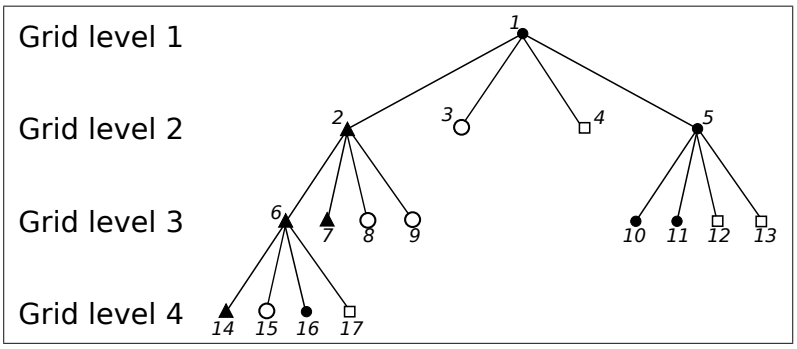

Figure 3.6: Sketch shows the CEG strategy that is used in Campfire which replaces the Morton order in PARAMESH. Four different shapes (i.e. □, △, • and ○) are used to indicate a possible distribution of Campfire for four MPI processes in a parallel environment. See Figure 3.3 for the corresponding example of Morton order.

When the AMR is carried out during computations in Campfire, ordering using the CEG order is dynamically maintained. As mentioned, multigrid runs from one grid to another, therefore, the partitioning strategy used by PARAMESH which partitions the whole tree is not optimal for multigrid. For example, in Figure 3.2 the finest refinement level is only distributed to two out of four MPI processes. The situation in Figure 3.4 is even worse, where only one MPI process is used for the problem on the finest refinement level. Therefore, the CEG order also includes a new partitioning strategy for the multigrid solver in Campfire. This strategy splits the workload on each grid independently and distributes to the MPI processes. This results a good balance on each level of grid but poor block locality as Campfire does not prioritize the preservation of the "parent-children" relation. For the dynamic load balancing, Campfire aggressively distributes these newly-generated blocks to maintain a good workload balance.

One trade-off from using the CEG order is that if the grid does not have enough mesh blocks, the parallel efficiency starts to deteriorate. As demonstrated in Figure 3.6, if more than four MPI processes are used, the only grid that can benefit is grid 3. Generally speaking, the problems in 2-D commonly have coarser grids that are not parallelizable for a large number of MPI processes. We will come back to this issue when illustrating 2-D models later on in this thesis.

3.3.2 Adaptive Mesh Refinement in Campfire

Having discussed a major modification to the Morton order in Campfire, in this section, the AMR in Campfire is summarised.

There are two main procedures for AMR: marking and adapting. This is previously described in Section 2.2.1. In Campfire, the marking procedure is done with the use of two logical arrays which are presented in Equation (3.2). This marking procedure is based upon the solution u^{τ} from the previous time step (or the given initial condition $u^{\tau=0}$ when the program begins). The superscript $\tau+1$ denotes the current time step and τ denotes the previous time step. For uniform grids, solutions from blocks which are on the finest grid are considered in the marking procedure. If an adaptive grid is used, such as the one presented in Figure 3.2, then only solutions from leaf blocks are considered. When this marking procedure is applied on each leaf block, two problem-specific, user-defined, criteria are typically required. One is for possible refining and another is for possible coarsening. These criteria are user-specified and may consider the solutions from all grid points within each block. Then logical values (i.e. "true" or "false") are given to the two logical arrays to indicate the decision. When marking for refining, a discrepancy check is included. If there is a circumstance where a block is marked for refining and coarsening at the same time, marking for coarsening is neglected. This marking procedure is presented in Algorithm 8.

The next procedure is to actually adapt these mesh blocks. This is done in another subroutine using these two logical arrays. It is further separated into: refining and coarsening. The point-wise refining and coarsening are described previously in Sections 2.2.1.1 and 2.2.1.2 respectively. Similar principles can be applied to mesh blocks.

Before the refining process starts, a GCU is required to be carried out. This is to ensure all guard cells are up-to-date, because some interpolation operators may require these values (e.g. the bilinear interpolation from Equation (2.3)). Only the leaf blocks can be refined. When a leaf block is marked "true" for refining, and its refinement level is lower than the maximum level, then four new blocks are considered to be generated. However, before memory spaces are assigned, if the current MPI process which holds the original "parent" block will imbalance the workload by taking these newly-generated blocks, then some of these blocks are shared between neighbouring MPI processes. Then appropriate memory spaces are assigned, and the indices of these blocks are created by the CEG ordering. The interpolation operator is then carried out on each new block. In the refining process, the discrepancy check that is described in the marking procedure is included, which neglects the decision to coarsen. The refining and coarsening processes need to preserve the mesh validity. This means all adjacent blocks may not have more than

Algorithm 8 Marking procedure of adaptive mesh refinement in Campfire

This algorithm calls to "Adaptive mesh refinement in Campfire" which is presented in Algorithm 9

1. Input: u^{τ} – solution from the previous time step or initial condition
2. Input: lb – number of blocks
3. Initialization
4. refine(all blocks) = false
5. derefine(all blocks) = false
6. for $clb = 1$ **to** lb **do**
7. **if** clb.leaf = true **and** clb.level != max level **then**
8. **if** $clb.u^{\tau}$.error $>$ problem-specific user-defined refining criterion **then**
9. refine(clb) = true
10. **end if**
11. **if** derefine(clb) = true **then**
12. derefine(clb) = false
13. **end if**
14. **else if** clb.leaf = true **and** refine(clb) = false **and** clb.level != min level **then**
15. **if** $clb.u^{\tau}$.error $<$ problem-specific user-defined coarsening criterion **then**
16. derefine(clb) = true
17. **end if**
18. **end if**
19. end for
20. Call "Adaptive mesh refinement in Campfire" with arrays refine and derefine

one level of refinement. If the situation happens, blocks with coarser mesh refinement level are automatically refined regardless of their criteria.

For coarsening, a group of four blocks which share the same "parent" block is required in 2-D. In 3-D, it is a group of eight blocks. Only the leaf blocks can be coarsened. When all leaf blocks in a group are marked for coarsening, and their refinement level is higher than the minimum level, firstly, values on these blocks are restricted to their "parent" block. Secondly, the indices of these blocks are taken out from the ordering of CEG. Finally, all the remaining blocks are re-ordered so the ordering of CEG can be maintained. This AMR process in 2-D is presented in Algorithm 9. Extensions of refining and coarsening to 3-D with cubic/cuboid blocks are straightforward.

Algorithm 9 Adaptive mesh refinement in Campfire

This algorithm calls to "Guard cells update" which is presented in Algorithm 7
1. Input: refine – logical array
2. Input: derefine – logical array
3. Call "Guard cells update"
4. for $clb = 1$ **to** lb **do**
5. **if** refine(clb) = true **then**
6. Four new blocks are taken into account
7. **if** current MPI process imbalances the workload with these new blocks **then**
8. Some blocks are allocated to neighbouring MPI processes
9. Create indices of blocks and ordering of CEG is maintained
10. Maintain ordering of CEG
11. **end if**
12. **if** Mesh validity is compromised **then**
13. refine(clb.neighbours) = true
14. Call "Adaptive mesh refinement in Campfire" with refine and derefine
15. **end if**
16. **end if**
17. end for
18. for $clb = 1$ **to** lb **do**
19. **if** derefine(clb) = true **and** derefine(clb.neighbours) = true **then**
20. **if** Mesh validity is not compromised **then**
21. Restrict solutions on these four blocks to their "parent" block
22. Deallocate memory and remove indices
23. Maintain ordering of CEG
24. **end if**
25. **end if**
26. end for
27. Call "Guard cells update"

Having described the spatial adaptive routines, in the following section, the adaptive

time stepping in Campfire is explained.

3.3.3 Adaptive Time Stepping in Campfire

Temporal adaptivity is previously described in Section 2.2.2. As mentioned before, in this thesis, a member of the family of BDF methods, specifically BDF2, is used extensively. Furthermore, the BDF2 method requires solutions from the past two time steps. Initially at time step 1, the only available solution is the given initial condition $u^{\tau=0}$. Therefore, BDF1 method has to be employed just for this time step.

One of the important components in adaptive time stepping is the indicator. Like the spatial adaptivity, it can be problem-specific, user-defined and/or solution-based. In Campfire, a general indicator is available, which depends upon the convergence rate of the multigrid solver. It is briefly mentioned in Section 2.4 that if a multigrid method is applied to a time-dependent problem, it solves the algebraic system arising from the local discretizations of PDEs at each implicit time step. Within each time step, this multigrid solver may be required to perform multiple V-cycles in order to obtain an acceptable approximation to the true solution. This convergence criterion typically comes from user-defined criteria (this is described in detail with each specific model). In practice, when a time step size δt is relatively small, the multigrid solver typically converges very quickly. The solver may converge with a reasonable number of V-cycles if δt is relatively large. In contrast, the solver struggles when δt is too large, and may even fail to converge. Thus, the number of V-cycles that the multigrid solver performed to solve the algebraic system (based upon an "acceptable residual") at the current time step can be a good indicator for the next time step size.

To sum up, in total there are five user-defined parameters for adaptive time stepping in Campfire. They are given as the following:

acceptable_residual : solution is acceptable if its residual is below this value;
max_V-cycle : the maximum number of V-cycles can be perfromed;
low_V-cycle : solver converges too easily;
high_V-cycle : solver struggles to converge;
max_δt : δt cannot be increased beyond this value.

The parameters acceptable_residual and max_δt generally are floating numbers, and the parameters that are associated with the number of V-cycles are integers.

The adaptive time stepping routine commonly has four conditions. First of all, if the

solver converges very quickly, then the next time step size may be increased. The amount of increasing is $\frac{10}{9}$ of the current δt^{τ} size. We described in Section 2.2.2, for the BDF2 method, the new time step size should not be increased by more than 191% of the current one. Our choice in Campfire satisfies this stability requirement. Secondly, if the solution at the current time step failed to converge (or converges too slowly to be accepted), then this time step is reset with a smaller time step size which is three quarters of the original one. Thirdly, if the solution slowly converges, but it is still acceptable, then the next time step uses a slightly smaller time step size which is $\frac{9}{10}$ of the previous one. The final condition is when the iteration converges with a reasonable convergence rate, then the next time step size remains the same.

The adaptive time stepping routine in Campfire is summarised in Algorithm 10.

Algorithm 10 Adaptive time stepping in Campfire

1. Input: No.V-cycles – the number of V-cycles that are used by the multigrid solver at this time step
2. Input : r – residual
3. if No.V-cycles $\leq$ low_V-cycle **and** $r \leq$ acceptable_residual **then**
4. $\quad \delta t^{\tau+1} = \frac{10}{9}\delta t^{\tau}$
5. $\quad$ **if** $\delta t^{\tau+1} \geq$ max_δt **then**
6. $\quad\quad \delta t^{\tau+1} =$ max_δt
7. $\quad$ **end if**
8. else if No.V-cycles $\geq$ max_V-cycle **or** $r >$ acceptable_residual **then**
9. $\quad$ Reset the current time step τ with $\delta t^{\tau} = \frac{3}{4}\delta t^{\tau}$
10. else if No.V-cycles $\geq$ high_V-cycle **then**
11. $\quad \delta t^{\tau+1} = \frac{9}{10}\delta t^{\tau}$
12. end if

3.3.4 Adaptive Multigrid Solver in Campfire

In this section, we describe the implementation of the multigrid solver in Campfire. It is based upon the nonlinear FAS multigrid method which is described in Section 2.4.4. If adaptive grids are used, then the MLAT technique is employed together with the nonlinear FAS multigrid which is explained in Section 2.4.5.

The tree data structure from PARAMESH (e.g. "unk" array) is used in Campfire, along with other supportive data. The CEG strategy for both ordering and partitioning is employed which replaces the original Morton order and partitioning in PARAMESH (see Section 3.3.1). Furthermore, to better suit the needs of the nonlinear FAS multigrid method, five "unk" arrays are associated with one variable. For example, the first variable

has the following data structure:

$\text{unk}(1,i,j,k,lb)$, stores the solution;
$\text{unk}(2,i,j,k,lb)$, stores values from RHS (or modified RHS);
$\text{unk}(3,i,j,k,lb)$, stores the residual;
$\text{unk}(4,i,j,k,lb)$, stores the solution from the previous time step;
$\text{unk}(5,i,j,k,lb)$, stores the solution from the one before previous time step.

For a system of PDEs, the second variable then starts with $\text{unk}(6,i,j,k,lb)$ and also has five arrays and this continues for all the variables.

For a nonlinear time-dependent problem, Campfire starts with an initialization, which allocates the required memory. Then all grid points are assigned with a given initial condition. For time-dependent systems, a loop which increments by a fixed δt (or $\delta t^{\tau+1}$ if adaptive time stepping is used) each time step starts. Before the solver is involved, an AMR is carried out to adapt the mesh according to the solution from the previous time step (or initial condition for the first step). Then a nonlinear FAS multigrid solver is employed to solve the algebraic system that is arising from the discretization of the PDEs at the current implicit time step.

There are two noteworthy changes in the implementation of the multigrid method in Campfire. Firstly, a local Gauss-Seidel version of the described nonlinear Jacobi iterative method which is presented in Equation (2.34) (or a Gauss-Seidel version of Equation (2.36) if there is a system of PDEs) is used as the smoother in the multigrid method. This smoother uses the most up-to-date values for the computation within a block. However, as mentioned previously, to achieve global Gauss-Seidel, a large number of GCUs are needed. Therefore, in order to maintain the efficiency, only one GCU is carried out per each iteration. In other words, within each block, the most up-to-date values are used in the computation. On the other hand, guard cells that are around the blocks are only updated once every iteration. This results in an iterative method which preforms Gauss-Seidel within blocks, but only Jacobi overall.

For completeness, it is worth noting there is a possible alternative method for the smoother, which is called Gauss-Seidel iteration with red-black ordering. This red-black Gauss-Seidel requires grid points to be separated into two groups: red and black. The use of the standard second-order stencil (e.g. five-point stencil from Equation (2.3)) on a grid point with one colour, only needs other points with the other colour for the computation. This implementation is achieved in Campfire with the block partition. However, in order to use a red-black Gauss-Seidel iteration, two GCUs are needed per iteration. One

GCU when computations are done on one colour of points, and another at the end of the iteration.

Secondly, this Gauss-Seidel within each block, and Jacobi overall method is used to solve the coarsest grid problem. There is no stopping criterion implemented for the coarsest grid solver in Campfire. Instead, a fixed number of iterations are performed. This is user specified and depends on the size of the coarsest grid and the accuracy required in this coarsest grid solver.

After one V-cycle finishes, by default, the infinity norm of the residual from all variables is computed. If the norm does not satisfy the problem-specific, user-defined, stopping criteria, another V-cycle is carried out. When the loop of the multigrid method finishes (either the solution converges or the maximum number of V-cycles is reached), an adaptive time stepping technique is applied (see Section 3.3.3). Then another time step starts with an adjusted δt (or resets the current time step with a smaller δt if non-convergence). These procedures are carried out until the end of the simulation. The structure of Campfire and its main procedures are summarised in Algorithm 11. It is assumed for this illustration that the program is running in a parallel environment with spatial and temporal adaptive techniques to solve a single nonlinear PDE which has the form of $A(u) = f$. Note many previously discussed algorithms are called.

Algorithm 11 Campfire

This algorithm calls to "Adaptive mesh refinement in Campfire" which is presented in Algorithm 9, "Adaptive time stepping in Campfire" which is presented in Algorithm 10, and "V-cycle MLAT nonlinear FAS multigrid method" which is presented in Algorithm 6.

1. Initialise with a pre-defined mesh structure and allocate memory spaces
   Set levels of refinement range from the finest Ω_f to the coarsest Ω_c
2. Impose a given initial guess
   $u^{\tau=0,h=\Omega_f,\ldots,\Omega_c}$
3. Impose an initial time stepping size $\delta t^{\tau=1}$
4. for $\tau = 1$ **to** ending time **do**
5. Call "Adaptive mesh refinement in Campfire" with solution u^{τ}
6. Call "V-cycle MLAT nonlinear FAS multigrid method"
7. Call "Adaptive time stepping in Campfire" with No.V-cycles and residual $r^{\tau+1}$
8. end for
9. Close Campfire

Campfire also includes a checkpoint system via the use of the software library HDF5. HDF5 saves all the data at the requested time into a checkpoint file, along with the current mesh structure. In addition, it works with parallel output. Thus, Campfire can be resumed from loading the checkpoint file. Information about this HDF5 software library can be

found in [108].

This described multigrid solver in Campfire was developed for a 3-D isothermal system of binary alloy solidification. This application is developed by Goodyer et al., and can be found in [91]. Comparing to the two-phase-field model of binary alloy solidification which is presented in Section 1.3.2.2, the model from [91] has only two dependent variables with the absence of the temperature variable. Goodyer et al. use the seven-point stencil from FDM and BDF2 as the discretization schemes, and cell-centred grid points. The main procedures of both spatial and temporal adaptivity are as explained in Sections 3.3.2 and 3.3.3 respectively.

In the following chapter, results which are obtained using Campfire (with some modifications) to solve the two-phase-field model of binary alloy solidification are presented. These results have been published in [27]. We use these to emphasize some of the contributions that are made during this work to both PARAMESH and Campfire, including efficiency improvement and new functionalities.

Chapter 4

Fully Coupled Phase-Field Solver for Model of Binary Alloy Solidification

In the previous chapter, the software tool, Campfire, and its mesh generator PARAMESH are described. This software tool is used for solving the 3-D model that is presented in [91]. Campfire has since been further improved and used to solve a more complex 3-D phase-field model of binary alloy solidification. The credit for solving this model mainly goes to Bollada [27], however a number of significant contributions developed as part of this work are described in this chapter.

This model of binary alloy solidification is previously presented in Section 1.3.2.2. Here, in Section 4.1, the model and the resulting solver that is implemented in Campfire are briefly described. Then, in the following sections, we introduce the contributions made to Campfire as part of this work, which are crucial in allowing Bollada to solve this solidification problem. More specifically, in Section 4.2, an improvement to the restart capacity of Campfire is introduced. Another improvement to the GCU routine, in terms of efficiency, is described in Section 4.3. Results that are obtained using the implementations we made are demonstrated in Section 4.4. Most of these results are generated by Bollada and are included in [27]. We refer back to Section 1.3.2.2 for the glossary of this model.

4.1 Introduction

As mentioned in Chapter 1, Goodyer et al. in [91] solve a 3-D binary alloy solidification phase-field isothermal model using Campfire. The model consists of only two dependent variables: a phase-field, $\phi(x,y,z,t)$, and a solute concentration, $U(x,y,z,t)$. The two-phase-field model for binary alloy solidification, as described in Section 1.3.2.2, is an extension of this that has an additional dependent variable: temperature, $\theta(x,y,t)$. It is worth noting that the evolution of the phase-field variable and its concentration happen much quicker than the evolution of the temperature. This 3-D model consists of Equations (1.22), (1.23) and (1.24) from Section 1.3.2.2, and is solved by Bollada et al. in [27] using Campfire.

The spatial discretization used for this 3-D model is the FDM on structured Cartesian grids with the cell-centred grid points. Furthermore, the scheme is based upon the central finite difference seven-point stencil (an equivalent five-point stencil for 2-D problems is presented in Equation (2.3)). The resulting continuous-in-time, initial value system of ODEs is then discretized using the adaptive, implicit BDF2 temporal discretization scheme which is shown in Equation (2.14). Then the described adaptive, nonlinear FAS multigrid solver from Campfire is employed for solving the discrete system of this 3-D model arising from each implicit time step. In [27], Bollada et al. use a point-wise Jacobi iteration as the smoother (see Section 2.3), which updates each variable one at a time. The same iterative method is used in the coarse grid solver with a fixed number of iterations. Within the coarse grid correction, restriction and interpolation are performed by the 3-D version of the described operators from Equations (2.13) and (2.12). The former performs the restriction with a cell averaging and the latter is a trilinear interpolation.

The strategies of spatial and temporal adaptivity are described previously in Sections 3.3.2 and 3.3.3 respectively. We include the details of the indicators that are used in this chapter. In the following section, a contribution to the software implementation of Campfire, employed in obtaining results in [27], is described.

4.2 Coarse-To-Fine Solution Prolongation in Campfire

In this section, we discuss one of the main contributions from the work in this thesis to the work published in [27]. It is a new functionality that is implemented into Campfire, which allows the program to prolongate the solution from the current finest grid to an even finer grid, and then to continue the computation. This is generally done by prolongating the solution from checkpoint files of using HDF5. It is called coarse-to-fine solution prolon-

gation (C2F). In Section 4.2.1, the reasons for needing this C2F functionality is described. In Section 4.2.2, we explain the procedures that are undertaken for this implementation. Some sample results from using C2F are included in Section 4.4.

4.2.1 The Motivation of New Functionality

The multigrid solver in Campfire has been run on the supercomputer HECToR [1], using up to one thousand cores for simulating binary alloy solidification. This simulation starts with a small seed, but as the solidification develops, dendrites start to form and grow rapidly. After the simulation progresses past the initial transient, a steady growth of the dendrites can be seen. This is indicated by the tip radius and tip velocity of the dendrites tending toward constant values. This steady growth is the main interest in [27], although the size of dendrites, their concentration and temperature may still be evolving.

This 3-D simulation is very computationally challenging and time consuming, even when one thousand cores are used. However, if the phase of steady growth is reached, we may prolongate this steady solution to an even finer grid as a good initial approximation to the solution on that fine grid. In this way, simulations may avoid starting from the very beginning. This gives an enormous efficiency boost and, as the results in [27] demonstrated, the accuracy from doing so is also acceptable.

As mentioned before, Campfire outputs checkpoint files through the use of HDF5. So the required implementation is to read in the checkpoint file and prolongate the solution one level finer. One further challenge is that when these checkpoint files from using HECToR were generated, there was no consideration for potentially more memory allocation. In the following section, the implementation of C2F is described.

4.2.2 Implementation of Coarse-To-Fine Solution Prolongation

Coarse-to-fine solution prolongation (C2F) is a newly implemented feature in Campfire. The motivation for this implementation is described in the previous section.

Given the structure of Campfire and its library PARAMESH, the concept of this implementation is straightforward. All that is needed is to have actual occurrence of mesh blocks from all refinement levels when the PARAMESH subroutine "amr_init" is called. In PARAMESH, this allows the potential memory spaces to be considered. When computation is carried out in Campfire, data from finer levels can be allocated into appropriate memory spaces. The AMR may be carried out freely afterward, even if the finest level of

[1] HECToR was situated at Edinburgh Parallel Computer Centre until April 2014, [109].

refinement has to be coarsened completely. During the computation, a global parameter "lrefine_max" may be modified to control the finest level of refinements that is permitted.

On the other hand, our challenge is for C2F to handle checkpoint files which have not initiated potential memory spaces for finer levels of refinements. In other words, when "amr_init" was called, there were no blocks on the even finer grid and none were anticipated. Therefore, no appropriate memory allocation is made. Even if the parameter "lrefine_max" is modified, without the proper memory management, it may result in segmentation faults. This complicates the implementation of the C2F, and carefully selected memory allocation before "amr_init" is needed. The "amr_init" routine only requires one mesh block at each refinement level to allocate the appropriate memory. One way to achieve this is when the solution from a checkpoint file is read into the program, a carefully selected refinement on the current finest grid is carried out. This generates blocks on the even finer grid. Then we re-initiate the memory allocation routine "amr_init" from PARAMESH. This way the new level of refinement can be recognised and potential memory spaces are considered appropriately. The computation can be carried out afterwards, and the value of "lrefine_max" may be increased whenever the prolongation is needed.

Another issue arises from refining the current finest grid as we read-in the checkpoint files. If the chosen blocks are on the edges between different levels of refinements then, to preserve the mesh validity, a large number of blocks may be refined. This is inefficient and unnecessary. Therefore, the blocks are chosen which are away from the refinement edges.

A few parameters of the C2F are added into Campfire for user editing, they are

C2F_start_level, which indicates the finest refinement level the checkpoint file provides;
C2F_desired_level, which indicates the finest refinement level that is needed;
C2F_turning_point, which indicates at which time step the prolongation is carried out.

Multiple prolongations are possible, but it is not used for the results presented here. In the following section, another improvement made to the routine of GCU in the software library PARAMESH is described.

4.3 Improvement to Guard Cells Update in PARAMESH

Having described the newly implemented functionality that increases Campfire's restart capacity, in this section, an improvement made to the routine of GCU in the software library PARAMESH is discussed. This routine of GCU is previously described in Section

3.2.4, and presented in Algorithm 7.

It is mentioned in Section 3.2.4 that the GCU routine is the most time-consuming process in Campfire. Due to the nature of the multigrid algorithm, there are potential improvements which can be made to the GCU in PARAMESH. One modification which has been done already by Goodyer, is adding an additional parameter to indicate the current level of grid. Then only guard cells which are on this level of grid are updated by GCU. This is an improvement to line 5 in Algorithm 7.

Another improvement that we contributed is limiting the number of restrictions done by GCU. First of all, it is worth noting that these restrictions are necessary for blocks that are at refinement edges. This is described previously in Section 3.2.4. On the other hand, in a multigrid algorithm, only two grids are interacting with each other at one time. So further restrictions to much coarser grids are completely unnecessary and time-consuming. Therefore, a modification to line 4 in Algorithm 7 is made to limit the restrictions to only one level coarser. This improvement has no impact on the computation in the multigrid solver at all, but greatly improves the efficiency. The improved GCU routine with the described modifications is summarised in Algorithm 12.

Algorithm 12 Improved guard cells update

1. Input: lvl – current level of grid
2. Initialise settings for guard cells
3. Re-establish "parents-children" relations of all mesh blocks
4. Restrict solution from blocks on grid lvl to blocks on grid $lvl-1$
5. Obtain correct values of guard cells for blocks on grid lvl and place in temporary array
6. Update guard cells using values from the temporary array
7. Re-impose boundary conditions on these guard cells that are positioned outside the boundary

All the works that are presented in this thesis benefit from this improvement. To give a clear demonstration, we take a middle stage 3-D simulation of the binary alloy solidification. Running the same simulation for 20 fixed time steps with both the original and the improved GCUs with 16 processors. The CPU times are presented in Table 4.1. These results suggest the improved GCU reduces the required CPU time by more than a factor of 2, and both solutions at the end of the simulations are identical.

In the following section, the results which are included in [27] are summarised.

Cases	CPU times (seconds)
Original implementation	67434.02
Improved implementation	25564.80

Table 4.1: Comparison between the original and improved GCUs with fixed 20 time steps and 16 processors.

4.4 Results

In this section, results from solving the model of binary alloy solidification are presented, and they are already included in [27]. These results are generated from using either the national supercomputer HECToR [109] or the high performance computing facility provided by the University of Leeds, named ARC2 [15].

Due to the nature of this problem, the dendrite is formed from a small seed and its growth is rapid. The computational domain is Ω which has Cartesian coordinates $(x,y,z) \in \Omega = (0,800) \times (0,800) \times (0,800)$, and we impose $dx = dy = dz$ for every mesh block. The initial condition for the phase-field variable ϕ with seed radius given by R is prescribed by

$$\phi(t=0) = -\tanh\left[0.6\left(\sqrt{x^2+y^2+z^2} - R\right)\right], \tag{4.1}$$

where x,y,z are Cartesian coordinates. The initial solute condition is $U = 0$, and the temperature profile is

$$\theta(t=0) = -\Omega_{undercooling} + \frac{1}{2}\Omega_{undercooling}(\phi + 1), \tag{4.2}$$

where $\Omega_{undercooling}$ is an undercooling parameter which sets the temperature of the liquid's initial and far boundary condition below its freezing point.

In Figure 4.1, we present a typical image showing the beginning of the formation of 6 dendrite arms. It is based on a run with an undercooling parameter of 0.525 and a Lewis number $L_e = 40$. The snapshot at $t = 102$ shows the $\phi = 0$ isosurface computed using the finest grid size $dx = 0.78$ [2]. A snapshot of the isosurface $\phi = 0$ at a later time (i.e. $t = 186$) is illustrated in Figure 4.2.

Within the solution, the interface is smooth but very steep. This is an ideal situation to use the adaptive mesh refinement, which gathers the most of the computation around the moving interface. To demonstrate this feature, we present a cross-section along the x-axis of a typical solution in Figure 4.3. The domain size is $(0,800) \times (0,800) \times (0,800)$,

[2]This grid, if refined everywhere, the resolution is $1024 \times 1024 \times 1024$, here we continue to use the notation from [27], which indicates the choices of dx instead of resolution.

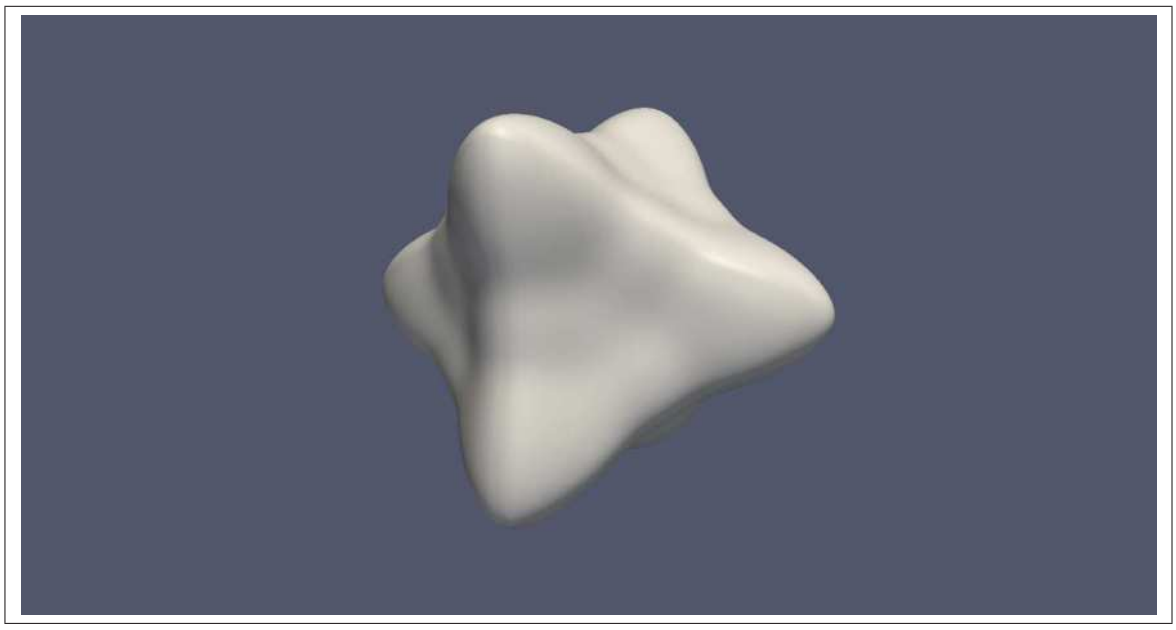

Figure 4.1: Dendrite at $t = 102$ with $L_e = 40$ and $dx = 0.78$. The silver shape is the contour of $\phi = 0$.

although only $(0, 600)$ is shown on this x-axis in this figure. The need for a very fine mesh around the phase interface is clear.

In Section 4.2.1, we explained that our motivation for C2F is because the tip radius will become steady as the simulation continues. This is illustrated in Figure 4.4 with simulations on three different grid hierarchies (in each case dx has a maximum value of 0.78, 0.39 and 0.195 respectively). These results (with $L_e = 40$ and $\Omega_{undercooling} = 0.325$) suggest that the latter two solutions give good agreement but that the solution using the coarsest grid hierarchy, with $dx = 0.78$, is less reliable.

In order to obtain the mesh convergence, results in Figure 4.4 were necessary to execute complete runs for each choice of dx from a small initial seed at $t = 0$, to a large time at which the tip radius is approximately constant. This is extremely computationally demanding for $dx = 0.195$ and may be very substantially improved through the described C2F technique. With this approach, once a solution is reached with a steady tip radius using grid size $dx = 0.39$, it can be prolongated to a finer grid which has $dx = 0.195$. A new run may be undertaken on this finer grid, to get a steady tip velocity much more efficiently than beginning again from the initial seed. We demonstrate this with a higher Lewis number (i.e. $L_e = 100$) in Figure 4.5. It may be observed that when the solution is interpolated around $t = 170$, a jump in the radius occurs. However, as the simulation continues, the tip radius converges relatively quickly. In this way, the simulation from $t = 0$ to $t = 170$ with grid size $dx = 0.195$ is not required, which provides an enormous

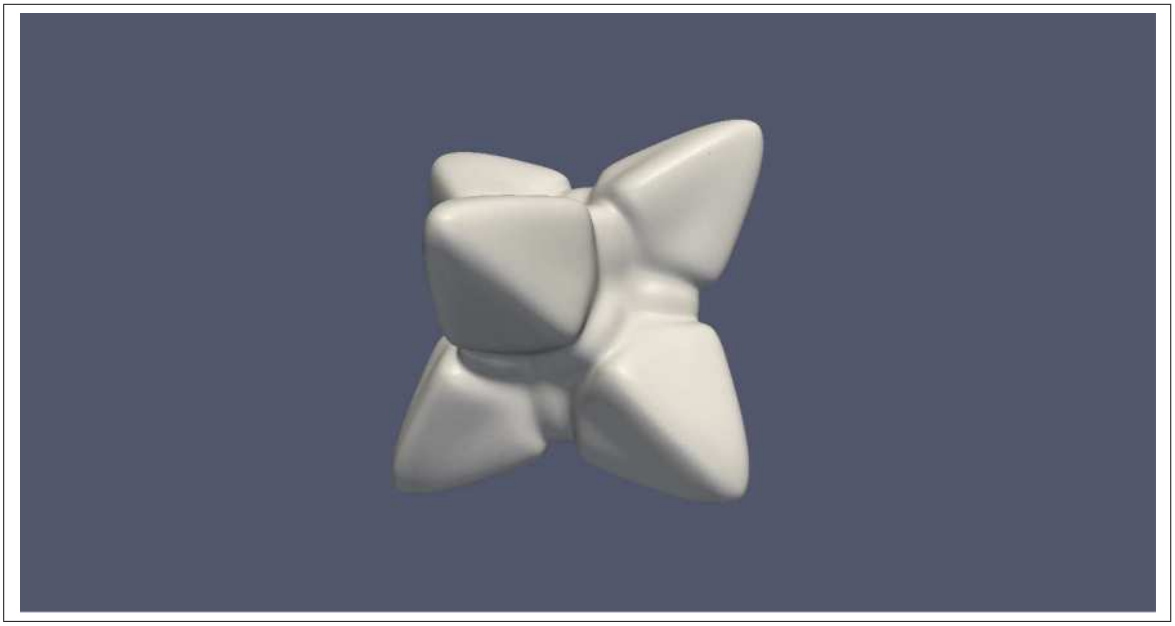

Figure 4.2: Dendrite at $t = 186$ with $L_e = 40$ and $dx = 0.78$. The silver shape is the contour of $\phi = 0$.

computational advantage from using the C2F.

The parallel performance from using 64 cores up to 1024 cores for a typical middle-late stage of the simulation are also discussed here. We have described parallel performance in Section 2.5.1.2 and the efficiency may be computed through Equation (2.59). We choose the simulation with grid size $dx = 0.39$ and undertake 10 fixed time steps, and the wall clock times are presented in Figure 4.6. The results suggest that the strong scalability of our implementation is not optimal, but that significant benefits are still obtained from parallel execution up to at least 1024 cores. The two sets of points included in this figure are for the cases where mesh adaptivity is switched off for the 10 time steps ("no remesh") or where the adaptivity is permitted ("remesh"). It is clear that the adaptivity itself is not responsible for the loss of parallel efficiency in this solver: this is primarily due to poor parallel performance on the coarse grids and in the grid transfer operations. We include a more detailed discussion in the next chapter (see Section 5.4.4) on the choice of the coarsest grid. Nevertheless, the execution time is reduced each time the number of cores is increased, as well as providing additional memory capacity to allow larger problems to be solved.

Having discussed results from solving the model of binary alloy solidification, in the following chapter, two 2-D thin film models are described.

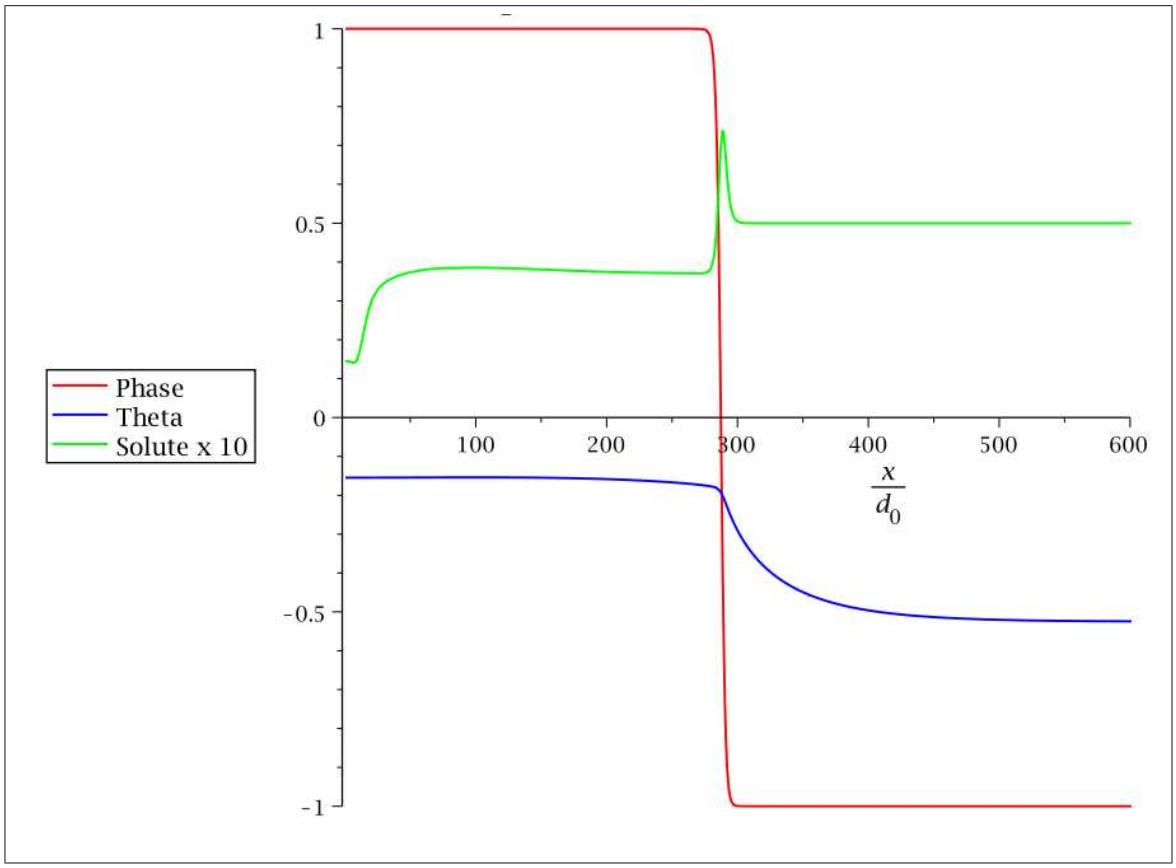

Figure 4.3: A cross-section along the x-axis of a typical solution with $L_e = 40$.

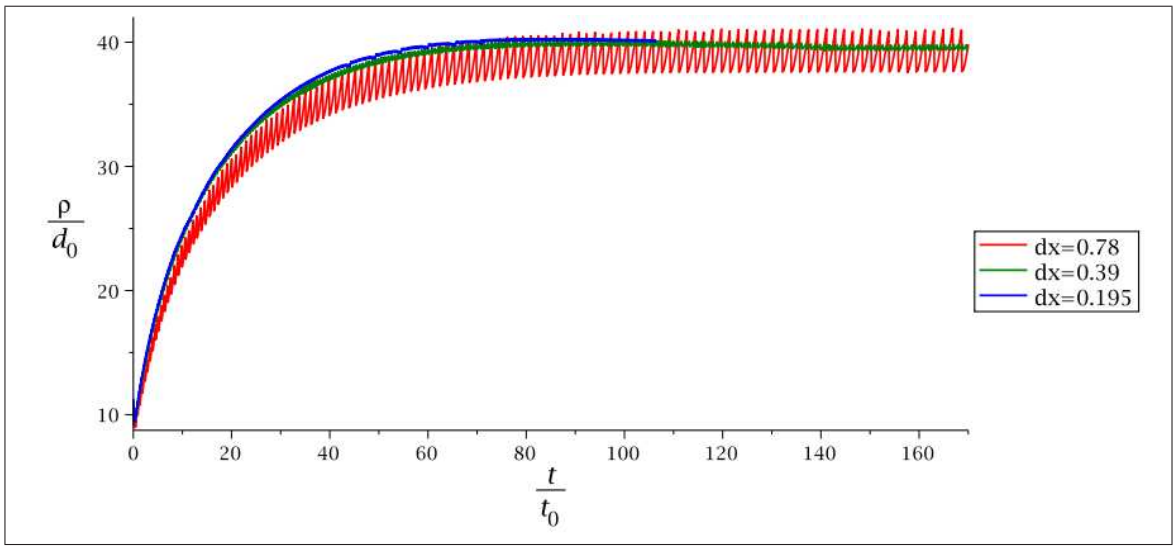

Figure 4.4: A plot of the evolution of tip radius on the y-axis with three different grid sizes.

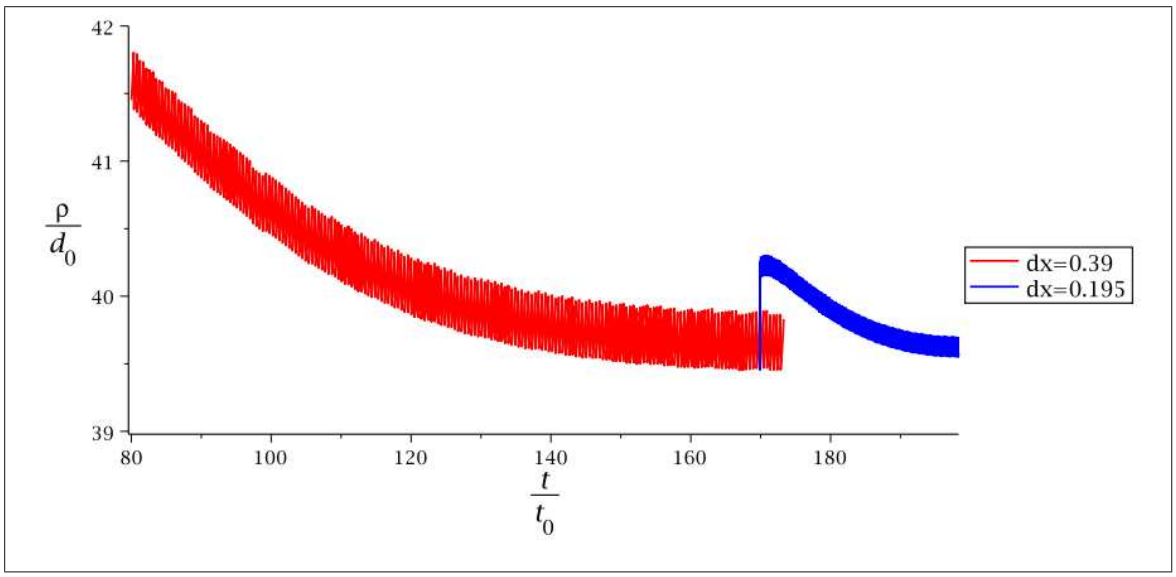

Figure 4.5: Effect of using the C2F technique on the evolving tip radius of the dendrite.

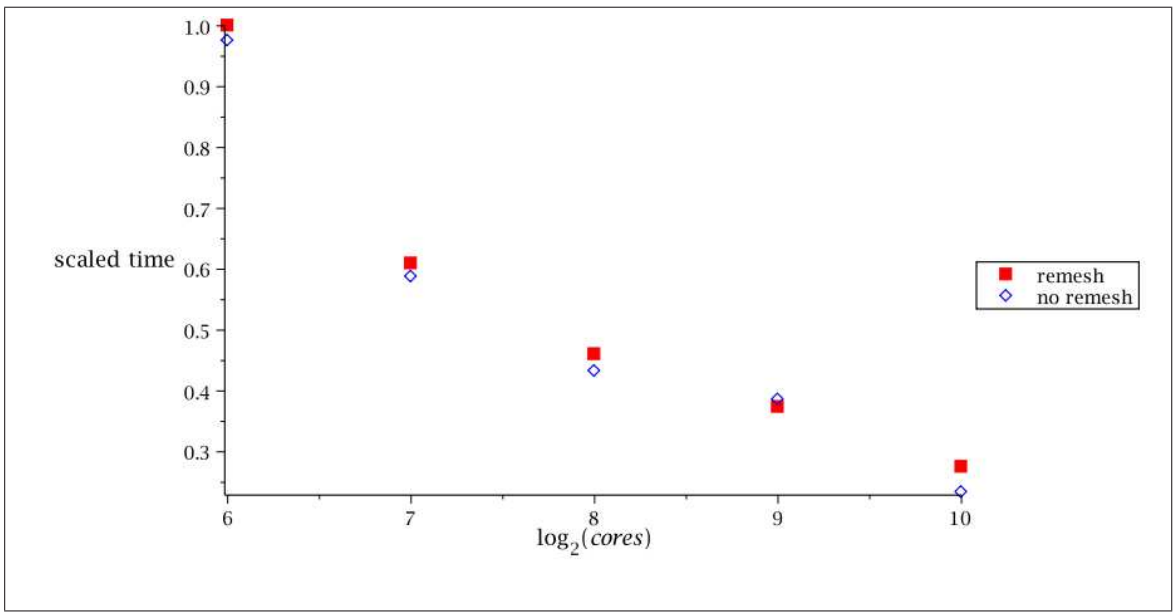

Figure 4.6: Plot of the wall-clock time for a middle-late stage simulation with grid size $dx = 0.39$, using different number of cores (64 to 1024).

Chapter 5

Parallel, Adaptive and Fully Implicit Time Stepping with FAS Multigrid on Models of Thin Film Flows

---

In this chapter, the models of thin film flows that are presented in Section 1.3.1 are solved by using our nonlinear multigrid solver. The first model is the droplet spreading model that is described by Gaskell et al. in [81], which is presented again in Section 5.1. The discretization schemes for this model are discussed in Section 5.2. Its implementations in Campfire are explained in Section 5.3. This required extension of the software is to allow non-time-dependent equations to be solved alongside a time-dependent PDE. Furthermore, Dirichlet boundary conditions are also implemented. Details are presented in Sections 5.3.1 and 5.3.2 respectively. In order to demonstrate that an overall second order convergence rate may be obtained the implementation of convergence tests is described in Section 5.3.3. Results obtained from using our modified solver are presented in Section 5.4. Validation against existing results from [81] is shown in Section 5.4.1, and convergence tests and multigrid performance are then demonstrated in Section 5.4.2. The use of adaptivity is explained in Section 5.4.3. Both the temporal adaptivity and the spatial adaptivity are individually assessed based upon their efficiency and accuracy. The use of parallel computing is finally included in Section 5.4.4. The droplet spreading model is concluded in Section 5.4.5, where a robust solver that combines multigrid, adaptivity and

parallelism is used to provide another set of convergence tests.

The second thin-film model that we consider is for fully-developed flows, as described by Gaskell et al. in [82] and Kalliadasis et al. in [125]. The model itself is presented in Section 5.5. Its discretization schemes and the resulting fully-discrete system at each time step are discussed in Section 5.6. Results obtained by using our multigrid solver are demonstrated in Section 5.7: validation against existing results is shown in Section 5.7.1; convergence tests are carried out and other numerical results are presented in Section 5.7.2; and the sections on fully-developed flows are concluded in Section 5.7.3.

We refer back to Section 1.3.1 for the glossaries for the models of thin film flows presented in this chapter. It is worth noting that we attempt to follow the notations that are generally used in the existing publications. Therefore, despite using h as the notation for distance between two adjacent grid points elsewhere in this thesis, here h is used as the notation for the variable of thin film thickness. The notation for distance between two adjacent grid points is dx and dy in this chapter, and for simplicity we ensure $dx = dy$ throughout.

5.1 Droplet Spreading Model Outline

The droplet spreading model with precursor film is previously described in Section 1.3.1.2. A sketch of the cross section of the droplet model is shown in Figure 1.2. The droplet spreading model is derived from the Navier-Stokes equations through the use of the lubrication approximation. This model has already been solved by Gaskell et al. in [81]. Here we present solutions of this model that are generated by using our parallel, adaptive multigrid solver in Campfire.

As mentioned in Section 1.3.1.2, this non-dimensional model consists of two dependent variables: $h(x,y,t)$ and $p(x,y,t)$. The former measures the droplet thickness, and the latter represents the pressure field of the droplet. For clarity, the equations of this model are presented here again. The thin film equation for the droplet thickness is given as

$$\frac{\partial h}{\partial t} = \frac{\partial}{\partial x}\left[\frac{h^3}{3}\left(\frac{\partial p}{\partial x} - \frac{B_o}{\varepsilon} sin\alpha\right)\right] + \frac{\partial}{\partial y}\left[\frac{h^3}{3}\left(\frac{\partial p}{\partial y}\right)\right], \tag{5.1}$$

where B_o is the Bond number which measures the relative importance of gravitational to surface tension forces. The pressure field of the droplet satisfies

$$p = -\triangle(h) - \Pi(h) + B_o h\cos\alpha, \tag{5.2}$$

where $\Pi(h)$ is the disjoining pressure term which has the following form:

$$\Pi(h) = \frac{(n-1)(m-1)(1-\cos\Theta_e)}{h^*(n-m)\varepsilon^2}\left[\left(\frac{h^*}{h}\right)^n - \left(\frac{h^*}{h}\right)^m\right], \tag{5.3}$$

where n and m are the exponents of interaction potential and Θ_e is the equilibrium contact angle. Note in Equation (1.20) from Section 1.3.1.2, a term $s(x,y)$ is included for possible topographies. However, this is not considered in this chapter.

The computational domain Ω is rectangular and has non-dimensional Cartesian coordinates $(x,y) \in \Omega = (0,1) \times (0,1)$. It is further assumed that the droplet is far away from the boundary. Therefore zero Neumann boundary conditions are applied for both h and p:

$$\frac{\partial h}{\partial \nu} = \frac{\partial p}{\partial \nu} = 0 \text{ on } \partial\Omega, \tag{5.4}$$

where ν denotes the outward-pointing normal to the boundary $\partial\Omega$.

The choices made for the initial condition $h(x,y,t=0)$ are presented later in this chapter, and once $h(x,y,t=0)$ is defined, the initial condition for p can be obtained from Equation (5.2). In the following section, discretization schemes that are used for the droplet spreading model are described.

5.2 Discretization Schemes for Droplet Spreading Model

The droplet spreading model is a nonlinear, coupled, parabolic system of PDEs. Due to the use of lubrication approximation, this model is only solved in a 2-D situation. For the spatial discretization scheme, the central FDM with the five-point stencil which is shown in Equation (2.3) is applied to both the thin film and the pressure equations (i.e. Equations (5.1) and (5.2)). The thin film equation for the droplet thickness shown in Equation (5.1) is time-dependent and thus requires a temporal discretization scheme. We choose the fully-implicit BDF2 method (see Equation (2.11) with $p = 2$). Furthermore, the BDF1 method (see Equation (2.9)) is used for the very first time step. For the purpose of demonstration, here we illustrate the BDF2 method with a fixed δt. If temporal adaptivity is required, it is straightforward to apply the adaptive BDF2 method from Equation (2.14).

The resulting fully-discrete system for the unknowns at time step $\tau + 1$ from the thin

film equation is given as

$$\begin{aligned}
&\frac{h_{i,j}^{\tau+1}-(\frac{4}{3}h_{i,j}^{\tau}-\frac{1}{3}h_{i,j}^{\tau-1})}{\delta t}= \\
&\frac{2}{3dx^2}\left\{\left(\frac{1}{2}\left[\frac{\left(h_{i+1,j}^{\tau+1}\right)^3}{3}+\frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right]\right)\left(p_{i+1,j}^{\tau+1}-p_{i,j}^{\tau+1}\right)\right. \\
&\quad-\left(\frac{1}{2}\left[\frac{\left(h_{i-1,j}^{\tau+1}\right)^3}{3}+\frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right]\right)\left(p_{i,j}^{\tau+1}-p_{i-1,j}^{\tau+1}\right) \\
&\quad+\left(\frac{1}{2}\left[\frac{\left(h_{i,j+1}^{\tau+1}\right)^3}{3}+\frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right]\right)\left(p_{i,j+1}^{\tau+1}-p_{i,j}^{\tau+1}\right) \\
&\quad\left.-\left(\frac{1}{2}\left[\frac{\left(h_{i,j-1}^{\tau+1}\right)^3}{3}+\frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right]\right)\left(p_{i,j}^{\tau+1}-p_{i,j-1}^{\tau+1}\right)\right\} \\
&-\frac{2B_o}{3\varepsilon}\sin\alpha(h_{i,j}^{\tau+1})^2\left(\frac{h_{i+1,j}^{\tau+1}-h_{i-1,j}^{\tau+1}}{2dx}\right).
\end{aligned} \tag{5.5}$$

The system from the pressure field equation is

$$\begin{aligned}
&p_{i,j}^{\tau+1}+\frac{1}{dx^2}\left\{h_{i+1,j}^{\tau+1}+h_{i-1,j}^{\tau+1}+h_{i,j+1}^{\tau+1}+h_{i,j-1}^{\tau+1}-4h_{i,j}^{\tau+1}\right\} \\
&\quad+\frac{(n-1)(m-1)(1-\cos\Theta_e)}{h^*(n-m)\varepsilon^2}\left[\left(\frac{h^*}{h_{i,j}^{\tau+1}}\right)^n-\left(\frac{h^*}{h_{i,j}^{\tau+1}}\right)^m\right] \\
&\quad-B_o h_{i,j}^{\tau+1}\cos\alpha=0.
\end{aligned} \tag{5.6}$$

Because it is fully-implicit, all forms in the above equations are unknowns at the new time step (i.e. $\tau+1$). Having obtained the fully-discrete system of droplet spreading model, in the following section, the implementation of the solver in Campfire is described.

5.3 Implementation of Droplet Spreading Model

This section shows how Campfire has been generalised to allow models with mixed parabolic and elliptic equations to be successfully implemented. Firstly, the implementation of non-time-dependent equations is described in Section 5.3.1. Then the Dirichlet boundary condition for our cell-centred multigrid solver is discussed in Section 5.3.2.

Finally, in order to demonstrate the overall second order convergence rate, our implementation of convergence tests is presented in Section 5.3.3.

5.3.1 Implementation of Non-Time-Dependent Equations in Campfire

Campfire when first developed by Green and Goodyer [93, 94] only considered systems of PDEs that were all time-dependent. Models that are solved in this thesis commonly have equations which are non-time-dependent, such as the pressure from Equation (5.2). Solving these equations was not originally considered in Campfire. One of the major issues comes from the modified RHS computation in the nonlinear FAS multigrid (see Equation (2.53)). Therefore, understanding the procedures of calculating the modified RHS term is crucial for implementing the non-time-dependent equations in Campfire.

Let's use the notation that is previously used in Section 2.1.2 for describing the temporal discretization schemes. Consider a time-dependent PDE, after applying a local spatial discretization scheme (e.g. FDM), a single initial value ODE is obtained and is given in Equation (2.7). This ODE is then discretised by a fully-implicit temporal discretization scheme. Here for the purpose of demonstration, we choose the BDF1 method (also known as the backward Euler method). This method is presented in Equation (2.9). However, in Campfire, it must be written as the following. Note we change the notation of function f to function g to avoid repetition from the multigrid method.

$$u^{\tau+1} - \delta t g(t^{\tau+1}, u^{\tau+1}) = u^{\tau}. \tag{5.7}$$

This form can be linked to the representations that are used for describing the nonlinear FAS multigrid in Section 2.4.4. For example, the left-hand side term in Equation (5.7) (i.e. $u^{\tau+1} - \delta t g(t^{\tau+1}, u^{\tau+1})$) is the corresponding term $A^f(u^f)$ in Equation (2.49). The RHS term u^{τ} is the corresponding term f^f in Equation (2.49). Then the term u^{τ} is stored separately to the "unk" array for storing values from the RHS (e.g. $\text{unk}(2, i, j, k, lb)$). This is previously described in Section 3.3.4.

The reason for having this particular format is that the modified RHS in Campfire is not calculated as $r^c + A^c(w^c)$ as suggested in Equation (2.53). Instead, it uses an accumulating approach, in which $[r^c - (f^c - A^c(w^c))]$ is firstly accumulated through two residual calculations before and after the restriction: r^c is the restricted residual from the fine grid, and is computed originally on the fine grid as $f^f - A^f(u^f)$; $f^c - A^c(w^c)$ is the residual term on the coarse grid using the restricted values. Since the RHS term f^f (which is u^{τ} initially) is separately stored and restricted to the coarse grid as f^c, it is then straightfor-

ward to add f^c to $[r^c - (f^c - A^c(w^c)]$ in order to give the correct values for the modified RHS.

There are two advantages for computing the modified RHS in this way. Firstly, it does not require extra user implementation but uses only the residual calculation. This may be very advantageous in the cases where users are not familiar with the nonlinear FAS multigrid method. Another advantage is that, since the "unk" array for storing values from the RHS (e.g. unk$(2,i,j,k,lb)$) is from the previous time step only, its values do not change during the computation made in the current time step. Therefore, it does not require guard cell update (GCU) or any other updates for this particular array. This in turn improves the performance of Campfire, as the GCU is expensive to carry out for parallel simulations.

For non-time-dependent equations, such as Equation (5.2), it is common to use the format that is presented in Equation (5.2), where the terms that depend upon p are arranged on the left-hand side of the equation, and the terms that do not depend upon p are moved to the RHS. This format is used in many publications for the descriptions of multigrid methods and the modified RHS (see [31, 216] for examples). However, in Campfire, non-time-dependent equations have to be arranged into the form shown in Equation (5.6). This is because the terms that do not depend upon p (but on h) may still change in their values during the computations made in the current time step, so may require GCU. The "unk" array for storing values from the RHS (e.g. unk$(2,i,j,k,lb)$) is therefore set to be 0 initially (but is generally non-zero for the coarse grid correction equations).

Having both time-dependent and non-time-dependent equations correctly implemented in Campfire, in the following section, implementations of the nonlinear FAS multigrid solver and the Dirichlet boundary condition for cell-centred finite differences are described.

5.3.2 Implementations of Multigrid Solver and Dirichlet Boundary Condition in Campfire for Droplet Spreading Model

In this section, implementation of the nonlinear FAS multigrid solver in Campfire is described, particularly the pre-, post-smoothers and the coarsest grid solver. We also discuss the implementation of the Dirichlet boundary condition. It is the first time this type of boundary condition has been implemented in Campfire. Although the droplet spreading model does not require this type of boundary condition, it is used in Section 5.4.2.1 for a modification of this model. This very common type of boundary condition is also required by some of the other models presented in this thesis.

In the original implementation of Campfire, it uses a point-wise nonlinear Jacobi iteration as the smoother. As mentioned previously in Section 3.3.4, it is difficult and very time-consuming to achieve a global Gauss-Seidel iteration in a parallel environment. However, the Jacobi iteration requires extra memory to store the values from the previous iteration (it typically uses some temporary memory allocations), and is generally slower when compared to the "local Gauss-Seidel, global Jacobi" iteration that is described in Section 3.3.4.

As well as using this "local Gauss-Seidel, global Jacobi" approach within each iteration, we also implement a block version of this smoother. Here, instead of using point-wise iteration all unknowns corresponding to the same grid point are updated simultaneously. An example of such a nonlinear block-Jacobi method is presented in Equation (2.36). It is straightforward to change this so it performs Gauss-Seidel within a mesh block, but only Jacobi updates at the edge of each block. It is worth noting that the Jacobian matrix C (for the droplet spreading model, this is a 2×2 matrix) in the nonlinear block-Jacobi method requires inverting. One of the simple ways to find the inverse of the Jacobian matrix is to use Cramer's rule and the determinant of the matrix C. Alternatively, Gaussian elimination (previously described in Section 2.3.1) can be employed to reduce the matrix C to an upper triangular form with the use of pivoting. Then an upper triangular solver can be applied to obtain the solution of the 2×2 system through the use of backward substitution.

To sum up, this nonlinear block method which performs "local Gauss-Seidel, global Jacobi" iteration is used as the pre- and post-smoothers, as well as the coarsest grid solver in the nonlinear FAS multigrid method that is implemented in Campfire.

The Dirichlet boundary condition is required in this thesis. Campfire originally did not support this boundary condition. Our modification allows the general Dirichlet boundary condition to be implemented in Campfire. This change is crucial for three models that are presented in this thesis. The boundary conditions are specified in a subroutine which is called "amr_1blk_bcset" in Campfire. Different boundaries are separately defined. In a 2-D situation, there are four boundaries. This changes to six boundaries in 3-D.

In this thesis, we work with cell-centred data. With these cell-centred grids, ghost cells are required to be positioned outside of the boundaries in order to provide the boundary conditions. This is previously described in Section 2.1.1.2 and the ghost cells are illustrated in Figure 2.2. Within Campfire, due to the use of guard cells, we do not need to separately define ghost cells. Those guard cells that are positioned outside of the domain boundaries can be used as the ghost cells. This is illustrated in Figure 2.14. On these guard cells, the Dirichlet boundary condition can be achieved as presented in Equation

(2.6). These values that help to impose the correct Dirichlet boundary condition are updated to these guard cells outside of the domain through the GCU (see line 7 in Algorithm 12).

In the following section, the convergence tests that are done for the model of fully-developed flows are described.

5.3.3 Convergence Test Based Upon Solution Restriction

The spatial and the temporal discretization schemes that are used in this thesis are the central FDM with the five-point stencil in 2-D (an equivalent seven-point stencil in 3-D) and the BDF2 method (although the BDF1 method has to be employed for the very first time step) respectively. These schemes are described previously in Chapter 2. Both the spatial and the temporal discretization schemes are second-order accurate. For the central FDM, this means the error is proportional to dx^2 (with the assumption that $dx = dy = dz$). For BDF2 method, the error is proportional to δt^2. Therefore, the combination of the two may result in an overall second-order convergence rate. This means by halving the time step size and doubling the number of grid points in each direction (e.g. $8 \times 8 \rightarrow 16 \times 16$), the error to the true solution should reduce by a factor of four.

On the other hand, as mentioned previously, the true solution generally is unavailable. Thus, approximate solutions from different runs with different grid hierarchies are used for the comparisons. In order to explain this approach, let's consider three example grid hierarchies using 2-D grids. They are $8 \times 8 - 16 \times 16$, $8 \times 8 - 16 \times 16 - 32 \times 32$ and $8 \times 8 - 16 \times 16 - 32 \times 32 - 64 \times 64$. Each grid is associated with a δt: $\delta t^{16\times16}$, $\delta t^{32\times32} = \frac{\delta t^{16\times16}}{2}$ and $\delta t^{64\times64} = \frac{\delta t^{32\times32}}{2}$, respectively.

The solutions are obtained by solving the same problem on these three finest grids separately, with their corresponding δt, and with the assumption that the ending time T is exactly the same for all runs. The solution from grid hierarchy $8 \times 8 - 16 \times 16$ remains the same, and solutions from other grid hierarchies are restricted by using a restriction operator (e.g. four-point averaging shown in Equation (2.13)). Unlike the vertex-centred grid points, with cell-centred grids there is no overlap between the cell-centres at different levels. Hence to make a comparison between two solutions we restrict the fine solution to the coarse grid and then compare. Thus, the solution which is restricted from grid hierarchy $8 \times 8 - 32 \times 32$ can be compared to the solution from grid hierarchy $8 \times 8 - 16 \times 16$. Similarly, the restricted solution from hierarchy $8 \times 8 - 64 \times 64$ can be compared to the original solution from hierarchy $8 \times 8 - 32 \times 32$.

The infinity norm and the discrete two norm are computed from these comparable

solutions on the coarser grid. The infinity norm is defined as the following:

$$||e||_\infty := max(|u_{i,j}^{restricted} - u_{i,j}|), \tag{5.8}$$

where $u^{restricted}$ is the restricted solution from the finer grid hierarchy, u is the original solution from the coarser grid hierarchy (note these solutions are obtained from separate runs) and $i, j = 1, \ldots, n$. Having defined the infinity norm, in the example used here, $||e||_\infty$ is calculated twice. The first time, solutions from $8 \times 8 - 16 \times 16$ and $8 \times 8 - 32 \times 32$ are compared. The second time, solutions from $8 \times 8 - 32 \times 32$ and $8 \times 8 - 64 \times 64$ are compared. The ratio between these two infinity norms is the measurement of the convergence rate. A ratio of around 4.0 indicates the second-order convergence rate (if only first-order convergence rate is achieved, then the ratio is around 2.0).

This ratio can also be calculated through the use of the two norms. The discrete two norm is given as

$$||e||_2 := \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(u_{i,j}^{restricted} - u_{i,j})^2}{n \times n}}. \tag{5.9}$$

In the notations for the infinity norm and the two norm above, e is used to represent the difference between two approximate solutions u. For four models that are solved in this thesis (two models of thin film flows, CHHS system of equations and model of tumour growth), second order convergence rate will be demonstrated (often for the first time to our knowledge). We consider all grid points from the finest grid or when adaptive grids are used, all the points from the finest refinement level possible. For the current droplet spreading model, the infinity norm and the two norm are computed for both variables h and p. The results are presented later in this chapter.

In the following section, results from solving the droplet spreading model using the described multigrid solver are discussed.

5.4 Results on Droplet Spreading Model

Results generated using our multigrid solver are presented in this section. Firstly, validation against existing results are described in Section 5.4.1. Then convergence tests and multigrid performance are discussed in Section 5.4.2. Results using adaptivity are illustrated in Section 5.4.3. Parallel computing is also included, and its results are described in Section 5.4.4. Finally, the robust solver combined with adaptivity and parallel computing and its results are presented in Section 5.4.5.

5.4.1 Validation

Results generated by using the parallel, adaptive multigrid solver from Campfire are compared to selected results presented in [81].

First of all, the values of parameters that are used in the droplet spreading model (Equations (5.1) to (5.3)) are presented in Table 5.1. These values were used by Gaskell et al. in [81] to generate their results presented in Figures 4 and 5.

Parameters	Values	Parameters	Values
B_o	0	ε	0.005
Θ_e	1.53°	h^*	0.04
n	3	m	2
α	0°		

Table 5.1: The parameters of the droplet spreading model that were used by Gaskell et al. in [81] for generating the results presented in their Figures 4 and 5.

The computational domain Ω has Cartesian coordinates $(x,y) \in \Omega = (0,1) \times (0,1)$. The initial condition for the variable of droplet thickness $h(x,y,t=0)$ is given as

$$h^{t=0}(r) = \max\left(5\left(1 - \frac{320}{9}r^2\right), h^*\right), \tag{5.10}$$

where $r^2 = x^2 + y^2$. Having obtained $h(x,y,t=0)$, the initial condition for pressure p on all internal grid points i,j $(i,j = 1,\ldots,n)$ may be defined as

$$\begin{aligned} p_{i,j}^{t=0} = {} & \frac{1}{dx^2}\left\{h_{i+1,j}^{t=0} + h_{i-1,j}^{t=0} + h_{i,j+1}^{t=0} + h_{i,j-1}^{t=0} - 4h_{i,j}^{t=0}\right\} \\ & + \frac{(n-1)(m-1)(1-\cos\Theta_e)}{h^*(n-m)\varepsilon^2}\left[\left(\frac{h^*}{h_{i,j}^{t=0}}\right)^n - \left(\frac{h^*}{h_{i,j}^{t=0}}\right)^m\right] \\ & - B_o h_{i,j}^{t=0}\cos\alpha. \end{aligned} \tag{5.11}$$

We choose Figure 5(b) from [81] to validate against. This figure shows the evolution of the maximum height of the droplet during simulations. All grids are uniform and the time duration is $[0, 10^{-5}]$. In Figure 5.1, the left-hand side shows a copy of Figure 5(b) from [81]. The right-hand side figure shows the results using our multigrid solver implemented in Campfire. The maximum height of the droplet is initially 5.0, as implied by the initial condition in Equation (5.10).

In order to generate these results, we use a 16×16 grid as the coarsest grid. There are 2 pre- and post-smoothers on each grid level, and 60 iterations of the smoother are

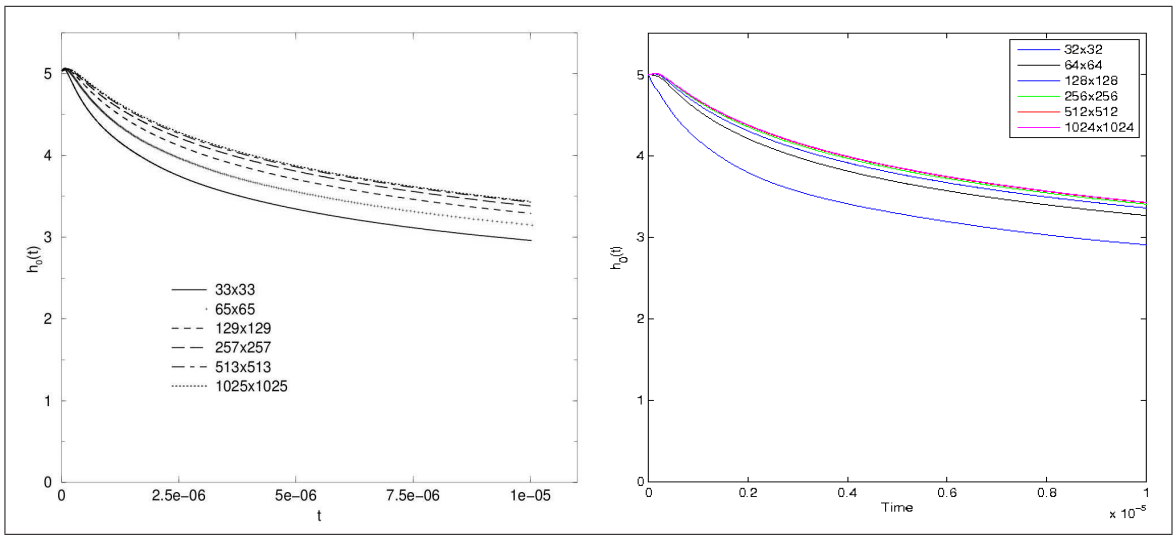

Figure 5.1: Figures show the evolution of the maximum height of the droplet during simulations, on the left-hand side is Figure 5(b) from [81] and on the right-hand side, we show results from our multigrid solver implemented in Campfire. Parameters used to generate these results are shown in Table 5.1. Legends in these figures indicate the finest resolutions of grids that are used for that particular simulation.

used for the coarsest grid solver. The time step size for grid hierarchy $16 \times 16 - 32 \times 32$ is $\delta t = 3.2 \times 10^{-7}$. Each time the finest grid is refined, the time step size is halved. This leads to use a time step size $\delta t = 10^{-8}$ for grid hierarchy $16 \times 16 - 1024 \times 1024$. Note that for some choices of the time step size δt, the size of the final step may be smaller so it finishes precisely at the end time $T = 1 \times 10^{-5}$.

There are two stopping criteria for the multigrid solver per time step based upon the infinity norm of residuals from the two variables, and at least one of them must be satisfied in order to continue the computation. The first is an absolute stopping criterion, which determines to stop the solver at the current time step if the infinity norm is smaller than 10^{-6}. The second is a relative stopping criterion, which takes the infinity norm after the first V-cycle in the current time step, and determines to stop if this infinity norm is reduced by a factor of 1×10^5 by subsequent V-cycles.

Figure 5.1 shows a good agreement with the results from grid hierarchies $16 \times 16 - 32 \times 32$, $16 \times 16 - 512 \times 512$ and $16 \times 16 - 1024 \times 1024$. For other grid hierarchies (i.e. $16 \times 16 - 64 \times 64$, $16 \times 16 - 128 \times 128$ and $16 \times 16 - 256 \times 256$), our results appear to be more accurate than the ones from [81]. This may be caused by the adaptive time stepping used in [81], based upon local error estimation, as opposed to our fixed time step size simulations.

There are other differences between our solver and the solver of Gaskell et al. [81].

Firstly, we use the BDF2 method as the temporal discretization scheme, but Gaskell et al. used the Crank-Nicolson method (see Equation (2.10)). Secondly, Gaskell et al. used a prediction-correction approach. Within a typical time step, instead of using the solution from the previous time step as the initial guess (which is our approach), Gaskell et al. included a predictor. This predictor uses the solution from the previous time step, and performs a fully explicit, second-order accurate "prediction" in order to provide a better initial guess for each time step (this predictor is illustrated in Equation (20) in [81]). This predictor also provides the error estimation which is then used to control the adaptive time stepping. Thirdly, Gaskell et al. employed a positivity preserving scheme as the spatial discretization scheme, which means that their discretization of the h^3 terms in Equation (5.1) is slightly different to that given in Equation (5.5), and used in our simulations. The possible advantage of using this scheme is the positivity of h may be preserved as the precursor film $h^* \to 0$. However, in our solver, the standard second-order five-point stencil from Equation (2.3) is applied. Finally, Gaskell et al. used vertex-centred grids instead of the cell-centred grids that we employed in this thesis. Despite these differences, the solutions from two solvers are highly agreeable when sufficient numbers of degrees of freedom are applied, such as the results from using grid hierarchies $16 \times 16 - 512 \times 512$ and $16 \times 16 - 1024 \times 1024$ shown in Figure 5.1. Further validation comes from analysing the multigrid convergence rates.

Since the solver from [81] also performs a nonlinear multigrid method with FAS, we validate the performance of our multigrid solver against the one used by Gaskell et al. More specifically, we validate the convergence rate of each multigrid V-cycle from a typical time step by the infinity norm of residuals. This is shown in Figure 4(b) from [81]. In Figure 5.2, the left-hand side shows the performance of the solver used in [81]. On the right-hand side is the performance of our multigrid solver from Campfire. For both solvers, a total number of 10 V-cycles within this particular time step are performed. From this figure, the results suggest that both solvers perform similarly. It is worth noting there is one significant difference. In the results from [81], the convergence rate deteriorates significantly from the 9^{th} V-cycle to the 10^{th} V-cycle. However, the results from using our multigrid solver remain robust in this situation. Overall we believe these tests provide excellent validation.

Having validated the implementation of our multigrid solver, in the following section, the convergence tests which demonstrate the overall second-order convergence rate are presented. Multigrid performance is also discussed.

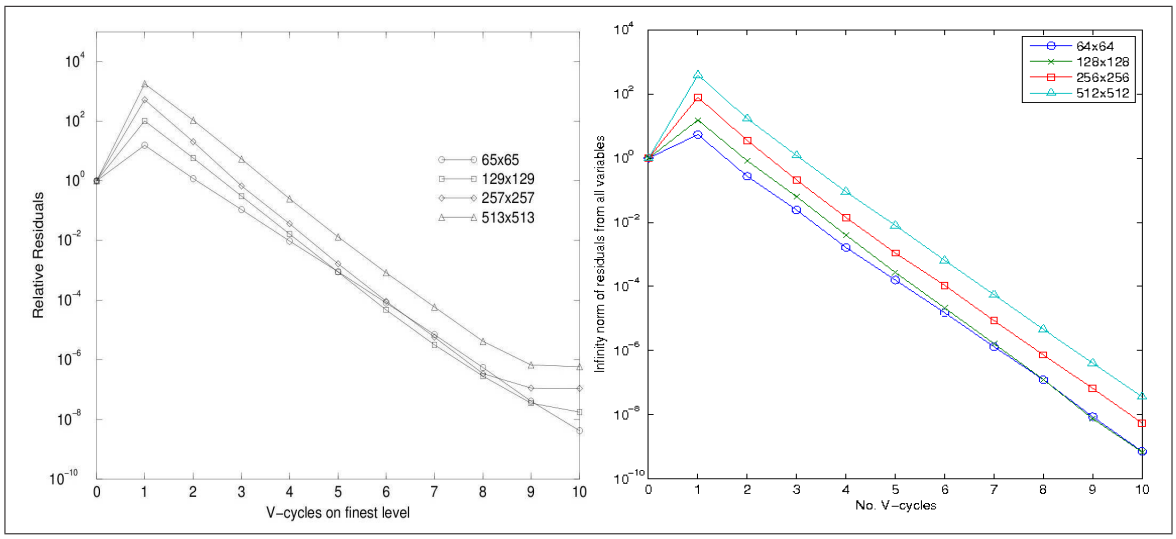

Figure 5.2: Figures show the convergence rate of a typical multigrid V-cycle from a single time step. On the left-hand side is Figure 4(b) from [81] and on the right-hand side are the results of our multigrid solver. Four different finest grid resolutions are used, such as shown in the legends. Parameters that are used to generate these results are shown in Table 5.1.

5.4.2 Convergence Tests and Multigrid Performance

The implementation of our convergence tests is described in Section 5.3.3. The results of these tests are presented in Section 5.4.2.1. The multigrid optimality is discussed in Section 5.4.2.2. Issues around the convergence tests are further investigated in Section 5.4.2.3, where additional results are presented. It is worth noting that all the work that is presented in this section was executed sequentially on one CPU. The workstation used to generate these results consists of 3.4GHz Dual Quad-Core Intel Xeon processors with 8GB memory.

5.4.2.1 Convergence Tests

In this section, convergence tests are carried out for the solutions at time $T = 1 \times 10^{-5}$. We choose four cases to demonstrate, and they have the resolutions on the finest grids of 128×128, 256×256, 512×512, 1024×1024 and 2048×2048. They all have the same coarsest grid (i.e. 16×16). Thus there are 4, 5, 6, 7 and 8 grids in the multigrid hierarchies for each different case. These cases are noted as levels 4, 5, 6, 7 and 8 respectively. It is noteworthy that level 8 is considered to be a large simulation in terms of the number of degrees of freedom and the number of grids, and is relatively expensive in terms of sequential computations. Here it is included for demonstrating the convergence rates, however, level 7 is generally the finest grid that is used later in this chapter.

The initial condition that is used by Gaskell et al. in [81] (see Equation (5.10)) is discontinuous in its first derivative and also very steep (i.e. initially the angle between the droplet and the substrate around the contact line is almost 90°). For the purpose of demonstrating the convergence of the solver we initially impose a much smoother initial condition for the variable h: it is

$$h^{t=0} = h^* + \sin(\pi x)\sin(\pi y), \tag{5.12}$$

where (x,y) are the Cartesian coordinates. When $h^{t=0}$ is available, the initial condition for pressure p can be obtained through Equation (5.11) in a straightforward manner. Since the initial condition is no longer flat in the vicinity of the boundaries, we also change the boundary conditions from those previously presented in Equation (5.4) to the following:

$$h = h^* \text{ and } p = 0 \text{ on } \partial\Omega. \tag{5.13}$$

The implementation of this Dirichlet boundary condition is described in Section 5.3.2. Furthermore, we also undertake simulations with and without the inclusion of the disjoining pressure term in Equation (5.2). Firstly this term $\Pi(h)$ is excluded from the pressure Equation (5.2). This can be done by setting $n = 1$ and $m \neq 1$. This problem is solved using uniform grids and fixed time stepping.

In Table 5.2, results of the convergence tests are shown, and they include both the infinity norm and the two norm of the differences between consecutive solutions. The convergence tests and these norms are previously explained in Section 5.3.3. Without the disjoining pressure term and with a smooth initial condition (see Equation (5.12)), the time-dependent variable h shows a near perfect second order convergence rate in this table. By quadrupling the number of grid points and halving the time step sizes, both the infinity norm and the two norm are reducing by a factor of 4. Since both the spatial and temporal discretization schemes are of second order accuracy, these results coincide with our expectation.

On the other hand, the convergence tests on the variable p only appear to be getting close to second order accurate using values of the norms (of the difference between consecutive solutions) when the resolutions of grids are sufficiently fine. Nevertheless, on these finest grids we are approaching second order convergence. Analysis of the convergence rates or discussions on the convergence theories are outside the scope of this thesis, and therefore we will not pursue it further. However, in Section 5.4.2.3, we present some of our additional observations that may be related to this issue.

Including the disjoining pressure term adds additional nonlinearity to the model. We

For variable h						
Levels	δt	Time steps	Infinity norm	Ratio	Two norm	Ratio
4	8×10^{-8}	125	-	-	-	-
5	4×10^{-8}	250	3.779×10^{-5}	-	1.880×10^{-5}	-
6	2×10^{-8}	500	9.449×10^{-6}	4.00	4.700×10^{-6}	4.00
7	1×10^{-8}	1000	2.362×10^{-6}	4.00	1.175×10^{-6}	4.00
8	5×10^{-9}	2000	5.905×10^{-7}	4.00	2.937×10^{-7}	4.00
For variable p						
4	8×10^{-8}	125	-	-	-	-
5	4×10^{-8}	250	7.457×10^{-3}	-	8.389×10^{-4}	-
6	2×10^{-8}	500	1.500×10^{-2}	0.50	1.213×10^{-3}	0.69
7	1×10^{-8}	1000	6.664×10^{-3}	2.25	4.124×10^{-4}	2.94
8	5×10^{-9}	2000	2.116×10^{-3}	3.15	1.113×10^{-4}	3.71

Table 5.2: Results show the differences in consecutive solutions measured in the stated norm, followed by the ratio of consecutive differences. These results are generated with a smooth initial condition and without the disjoining pressure term. Parameters are shown in Table 5.1, with the exceptions of $n = 1$ and $m \neq 1$.

repeat the tests done in Table 5.2 but now including the disjoining pressure term $\Pi(h)$ in the pressure Equation (5.2). Results of the convergence tests are presented in Table 5.3. In this table, results on variable h again indicate a clear second order convergence rate (though not quite so perfectly as in Table 5.2). On the other hand, results on variable p appear to be better than those presented in Table 5.2, especially when the resolutions of grids are not very fine. The values of the two norm of p shows a definite second order convergence, and its infinity norm tends to be second order. It may be seen that the values from both norms are relatively large compared to those in Table 5.2.

Note that later in this chapter we also consider convergence using the true initial condition shown in Equation (5.10) but before this we need to investigate adaptive methods. In the following section, however, multigrid performance is first discussed.

5.4.2.2 Multigrid Performance

Having presented some convergence tests, the performance of the multigrid solver is considered here. Previously, in Section 2.4, it is described that the convergence rate of multigrid methods should be independent from the size of the finest grid. This may already be seen from Figure 5.2 in the previous section. The convergence rate for four different grid hierarchies are more or less parallel to each other: despite the difference in the number of grid points, the convergence rates for these cases share nearly the same slope. Here

For variable h						
Levels	δt	Time steps	Infinity norm	Ratio	Two norm	Ratio
4	8×10^{-8}	125	-	-	-	-
5	4×10^{-8}	250	5.710×10^{-5}	-	2.095×10^{-5}	-
6	2×10^{-8}	500	1.970×10^{-5}	2.90	5.677×10^{-6}	3.69
7	1×10^{-8}	1000	5.549×10^{-6}	3.55	1.444×10^{-6}	3.93
8	5×10^{-9}	2000	1.430×10^{-6}	3.88	3.634×10^{-7}	3.97
For variable p						
4	8×10^{-8}	125	-	-	-	-
5	4×10^{-8}	250	$1.554 \times 10^{+1}$	-	2.111×10^{0}	-
6	2×10^{-8}	500	6.539×10^{0}	2.38	6.547×10^{-1}	3.22
7	1×10^{-8}	1000	2.131×10^{0}	3.07	1.755×10^{-1}	3.73
8	5×10^{-9}	2000	0.605×10^{0}	3.52	4.477×10^{-2}	3.92

Table 5.3: Results show the differences in consecutive solutions measured in the stated norm, followed by the ratio of consecutive differences. These results are generated with the use of a smooth initial condition and the disjoining pressure term in the pressure equation. Parameters are shown in Table 5.1.

we demonstrate the details of the multigrid solver that is used to obtain the results shown in Table 5.2, and it is presented in Table 5.4. The results from this table show that when we quadruple the number of grid points as well as double the number of time steps, the average number of V-cycles that is needed stays a constant. In addition, the CPU time for sequential executions is increased by a factor of 8 from one run to the next. Since the problem size is also increased by a factor of 8, this suggests that our multigrid solver converges with a linear complexity of $\mathcal{O}(n)$.

Levels	Finest grid	Total number of time steps	Average V-cycles per time step	CPU time (seconds)
4	128^2	125	4.0	25.6
5	256^2	250	4.0	200.9
6	512^2	500	4.0	1637.5
7	1024^2	1000	4.0	13121.6
8	2048^2	2000	4.0	109620.4

Table 5.4: Table shows the resolution of the finest grid, total number of time steps, average V-cycles required per time step and the CPU time for five difference hierarchies of uniform grids.

We further show a log-log plot in Figure 5.3, with the average CPU times per time step from all five test cases that are presented in Table 5.4, against the total number of

grid points on the corresponding finest grid. For comparison, a line with slope of 1 is also shown in Figure 5.3. Each time a finer grid is used, the number of grid points on the finest grid is quadrupled. The average time cost per time step is parallel with the line of slope of 1. This demonstrates that our multigrid solver has a linear complexity.

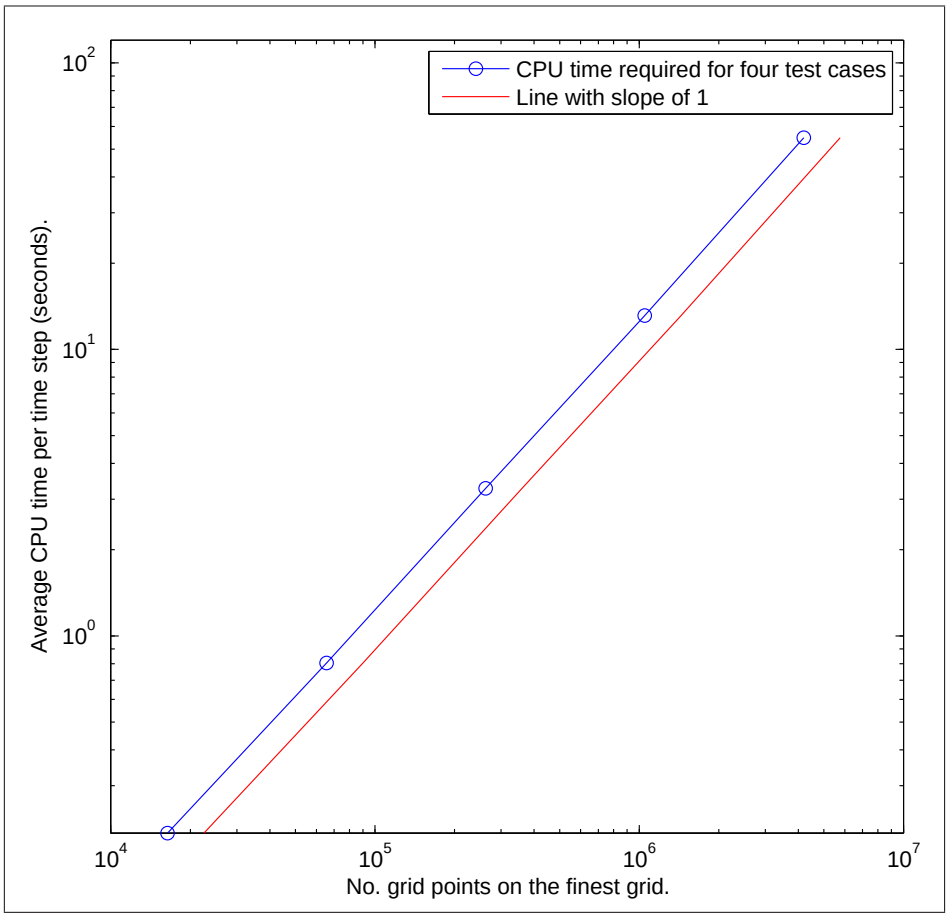

Figure 5.3: Figure shows a log-log plot, in x direction, the total number of grid points from the finest grid is shown, and the average CPU time per time step in seconds is presented in y direction. For comparison, a line with slope of 1 is also shown in the figure.

In the following section, issues around the results from the convergence tests are discussed further.

5.4.2.3 Discussion

The convergence tests presented in Section 5.4.2.1 (i.e. in Tables 5.2 and 5.3) suggest a second order convergence rate for the time-dependent variable h. However, there are issues around the convergence tests on the variable p, especially the results presented in Table 5.2, which show that p is quite slow to approach second order convergence. In

order to be confident that p really is approaching second order, we present additional results from a number of further tests.

First of all, we validate our multigrid solver and eliminate the possibilities that the issues are caused by incorrect implementation. In order to achieve this, the droplet spreading model is reformulated to a simple linear coupled system of two PDEs. More specifically, the original model that is shown in Equations (5.1) and (5.2) is reduce to the following:

$$\frac{\partial h}{\partial t} = \triangle p, \tag{5.14}$$

$$p = -\triangle h. \tag{5.15}$$

We use the same initial and boundary conditions for this coupled linear system of PDEs, as presented in Equations (5.12) and (5.13) respectively. Except for the precursor film thickness h^* which is used in the initial condition, parameters that are shown in Table 5.1 have no influence in this newly reformulated coupled linear system. Due to the existing convergence analyses and theories, the expectation of the convergence rate from solving this coupled linear system is that both variables are second order, providing second order spatial and temporal discretization schemes are applied.

The convergence tests previously carried out in Section 5.4.2.1 are repeated on this newly reformulated coupled linear system and results are presented in Table 5.5. The results presented in this table coincide with our expectation, that is for both variables, they have very clear second order convergence rates. These results also suggest that our solver is correctly implemented and therefore that the convergence issues for the pressure variable p may be caused by the nonlinearity in the original droplet spreading model.

To further investigate our hypothesis, two additional systems are proposed. The first one is given as the following:

$$\frac{\partial h}{\partial t} = \nabla \cdot (h \nabla p), \tag{5.16}$$

$$p = -\triangle h. \tag{5.17}$$

The convergence tests that are carried out in Table 5.5 previously are now repeated on this new model. Results are shown in Table 5.6. The convergence tests for variable h presented in this table indicate an optimal second order convergence rate. It may be seen that the convergence rates of pressure p deteriorate for the coarser runs. However, a second order convergence rate is approached eventually.

The second of the two additional models has an h^2 nonlinear term instead of h, and is given as the following:

$$\frac{\partial h}{\partial t} = \nabla \cdot \left(h^2 \nabla p\right), \tag{5.18}$$

For variable h						
Levels	δt	Time steps	Infinity norm	Ratio	Two norm	Ratio
4	8×10^{-8}	125	-	-	-	-
5	4×10^{-8}	250	3.779×10^{-5}	-	1.890×10^{-5}	-
6	2×10^{-8}	500	9.449×10^{-6}	3.99	4.724×10^{-6}	3.99
7	1×10^{-8}	1000	2.362×10^{-6}	3.99	1.181×10^{-6}	3.99
For variable p						
4	8×10^{-8}	125	-	-	-	-
5	4×10^{-8}	250	5.778×10^{-6}	-	2.890×10^{-6}	-
6	2×10^{-8}	500	1.445×10^{-6}	3.99	7.224×10^{-7}	3.99
7	1×10^{-8}	1000	3.614×10^{-7}	3.99	1.806×10^{-7}	3.99

Table 5.5: Results show the differences in consecutive solutions measured in the stated norm, followed by the ratio of consecutive differences from the reformulated coupled linear system which consists of Equations (5.14) and (5.15). These results are generated with the use of the smooth initial condition and the Dirichlet boundary condition that are used in Section 5.4.2.1.

For variable h						
Levels	δt	Time steps	Infinity norm	Ratio	Two norm	Ratio
4	8×10^{-8}	125	-	-	-	-
5	4×10^{-8}	250	3.786×10^{-5}	-	1.964×10^{-5}	-
6	2×10^{-8}	500	9.466×10^{-6}	4.00	4.921×10^{-6}	4.00
7	1×10^{-8}	1000	2.367×10^{-6}	4.00	1.231×10^{-6}	4.00
For variable p						
4	8×10^{-8}	125	-	-	-	-
5	4×10^{-8}	250	6.644×10^{-2}	-	8.762×10^{-3}	-
6	2×10^{-8}	500	2.055×10^{-2}	3.23	2.340×10^{-3}	3.74
7	1×10^{-8}	1000	5.710×10^{-3}	3.60	5.957×10^{-4}	3.93

Table 5.6: Results show the differences in consecutive solutions measured in the stated norm, followed by the ratio of consecutive differences from the reformulated coupled linear system which consists of Equations (5.16) and (5.17). These results are generated with the use of the smooth initial condition and the Dirichlet boundary condition that are used in Section 5.4.2.1.

$$p = -\triangle h. \tag{5.19}$$

We repeat the convergence tests done previously in Tables 5.5 and 5.6 on this second additional model. Results are shown in Table 5.7. Again, the convergence tests for variable h indicate an optimal second order convergence rate. For this model, the convergence rates for the pressure p are worse than those shown in Table 5.6 (but still better than those in Table 5.2). From these results, we believe the nonlinearity in this type of model is one of the causes for pressure variable p to require such a fine mesh resolution before a second order convergence is approached.

For variable h						
Levels	δt	Time steps	Infinity norm	Ratio	Two norm	Ratio
4	8×10^{-8}	125	-	-	-	-
5	4×10^{-8}	250	3.796×10^{-5}	-	1.879×10^{-5}	-
6	2×10^{-8}	500	9.492×10^{-6}	4.00	4.700×10^{-6}	4.00
7	1×10^{-8}	1000	2.373×10^{-6}	4.00	1.175×10^{-6}	4.00
For variable p						
4	8×10^{-8}	125	-	-	-	-
5	4×10^{-8}	250	6.041×10^{-2}	-	6.965×10^{-3}	-
6	2×10^{-8}	500	2.745×10^{-2}	2.20	2.403×10^{-3}	2.90
7	1×10^{-8}	1000	8.942×10^{-3}	3.07	6.586×10^{-4}	3.65

Table 5.7: Results show the differences in consecutive solutions measured in the stated norm, followed by the ratio of consecutive differences from the reformulated coupled linear system which consists of Equations (5.18) and (5.19). These results are generated with the use of the smooth initial condition and the Dirichlet boundary condition that are used in Section 5.4.2.1.

So far the results in this section are generated using uniform grids, fixed time step size and a smooth initial condition. In the following sections, the use of spatial and temporal adaptivity is described, with the original initial condition that is used in [81] (i.e. Equation (5.10)).

5.4.3 Adaptivity

Adaptivity is previously described in Section 2.2. Its implementation in Campfire is discussed in Sections 3.3.2 and 3.3.3. Both temporal and spatial adaptivity are applied to the droplet spreading model here. However, they are individually assessed. The temporal adaptivity is described in Section 5.4.3.1 and the spatial adaptivity is discussed in Section 5.4.3.2.

5.4.3.1 Temporal Adaptivity

In this section, temporal adaptivity is applied to the full droplet spreading model. Its implementation in Campfire is discussed in Section 3.3.3. In this section, we use the initial conditions for h and p which are presented in Equations (5.10) and (5.11) respectively, and represent a droplet with a height of 5 and radius $\frac{8\sqrt{5}}{3}$. The boundary conditions are presented in Equation (5.4). The parameters that are used are presented in Table 5.1.

The temporal adaptivity is achieved by using the adaptive BDF2 method that is presented in Equation (2.14). For the purpose of demonstration, an aggressive adaptive approach is applied. This is achieved by setting the parameters "low_V-cycle" to 6 and "high_V-cycle" to 7. This setting forces Campfire to always seek a different time step size. We present results of two different grid hierarchies, they are levels 6 and 7 with 512×512 and 1024×1024 as the finest grids. The equivalent simulations using fixed time step sizes are already presented in the right-hand side of Figure 5.1. For the adaptive time stepping we use the same initial step size as in the non-adaptive cases: initial time step sizes for these two cases are 2×10^{-8} and 1×10^{-8} respectively. The resulting evolution of the time step sizes is shown in Figure 5.4. It may be seen from this figure that our adaptive time stepping approach is indeed aggressive (the sudden reductions of time step sizes at the end are simply to ensure both simulations are ended at the same time $T = 10^{-5}$). Let's contrast with the equivalent simulations (from the right-hand side of Figure 5.1) that are undertaken using fixed time step sizes. For the level 5 simulation (with 512×512 as the finest grid) 500 time steps are required with a step size of 2×10^{-8}. For the level 6 simulation (with 1024×1024 as the finest grid) 1000 time steps are required with a step size of 1×10^{-8}. The use of the adaptive time stepping approach reduces the number of time steps required to 39 and 45 respectively.

The detailed comparison between the use of fixed time step size and the adaptive time stepping is presented in Table 5.8. From this table, as well as the number of time steps required to reach $T = 1 \times 10^{-5}$ being shown for adaptive time stepping versus fixed time steps, the execution times are also shown. On level 6 adaptive time stepping takes just 9.6% of the time with use of the fixed time step size. This percentage becomes 5.2% for the simulation on level 7. This is despite the increased number of average of V-cycles required per time step in the adaptive case. Note however that the number of V-cycles needed is still independent of grid sizes.

Two questions are worth asking. Firstly, are our choices of the time step size too small for the fixed time step approach? Additional tests show that for the level 6 case, increasing the initial time step size by a factor of 5 causes the multigrid solver to converge more slowly as, within each time step, about three more V-cycles are needed. The computation

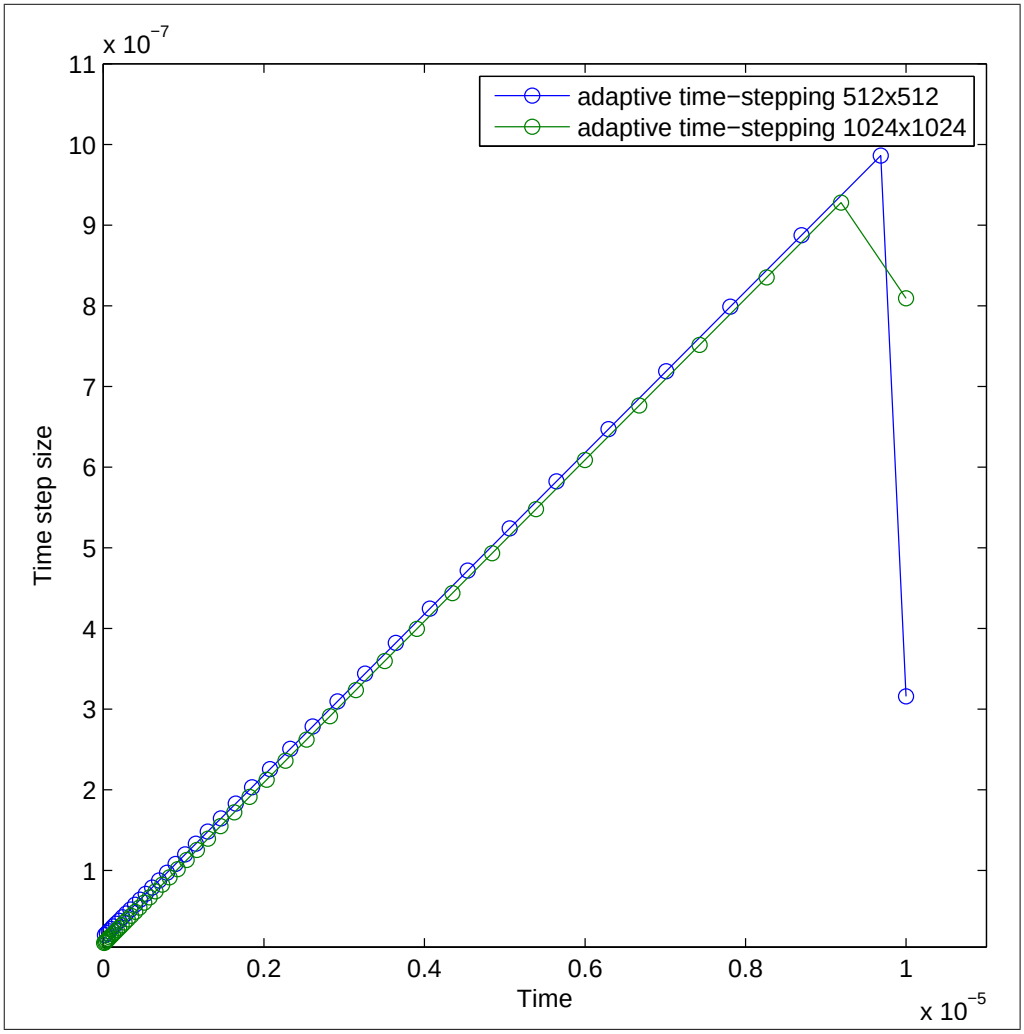

Figure 5.4: Evolution of the time step sizes from two test cases for the droplet spreading problem on 512×512 and 1024×1024 finest grids. Results are obtained by using the adaptive BDF2 method shown in Equation (2.14).

fails to converge if the initial time step size is increased by a factor of 10. The second question is: when using the adaptive time stepping, how accurate are these solutions? In order to answer this question, we take the approach which is used in Figure 5.1, that is measuring the maximum height of the droplet. The comparison is shown in Figure 5.5. From this figure, it can be seen that by using adaptive time stepping, the evolutions of the height of the droplet are very close to the ones using the original approach with fixed time step sizes.

Levels	No. TSs fixed δt	Avg. V-cycle per TS	CPU time (seconds)	No. TSs ATS	Avg. V-cycle per TS	CPU time (seconds)
6	500	5.0	2095.3	39	5.9	201.5
7	1000	5.0	16721.3	45	5.8	874.4

Table 5.8: Comparisons between the use of fixed time step size and the adaptive time stepping for two test cases. The total number of time steps, the average V-cycles required per time step and the CPU time are used for the comparisons. Due to the limit of space, abbreviations are used, where TS means "time step", Avg. means average and ATS is short for adaptive time stepping.

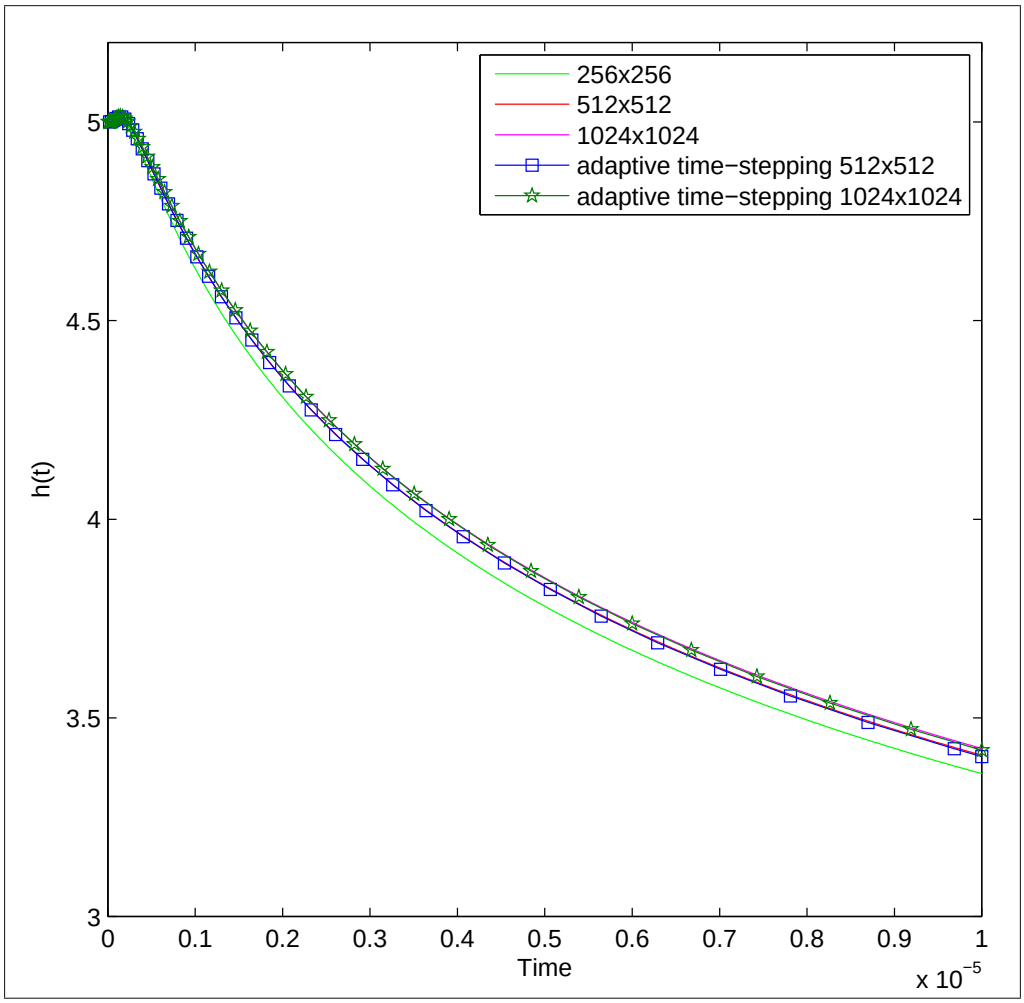

Figure 5.5: Figure shows the evolutions of the maximum height of the droplet. We measure the maximum height for comparison and demonstration of the accuracy of the adaptive time stepping. Results with the finest grids 256×256, 512×512 and 1024×1024 have been previously presented in Figure 5.1, and they are obtained using the fixed time step size. For the two cases of adaptive time stepping, we use squares and stars to indicate the values of time step size within each time step.

To give a further indication of how accurate the solutions are at the end of simulation, a zoom-in feature is shown in Figure 5.6. The results shown in this figure indicate that our adaptive time stepping approach deteriorates the accuracy by only a very small amount. More specifically, for level 6 (with the finest grid resolution of 512×512), the values of the maximum heights of the droplet are 3.406 from the use of fixed time step size, and 3.403 from the use of adaptive time stepping. For level 7 (with the finest grid resolution of 1024×1024), the value is 3.423 from the use of fixed time step, and is 3.419 from the use of adaptive time stepping.

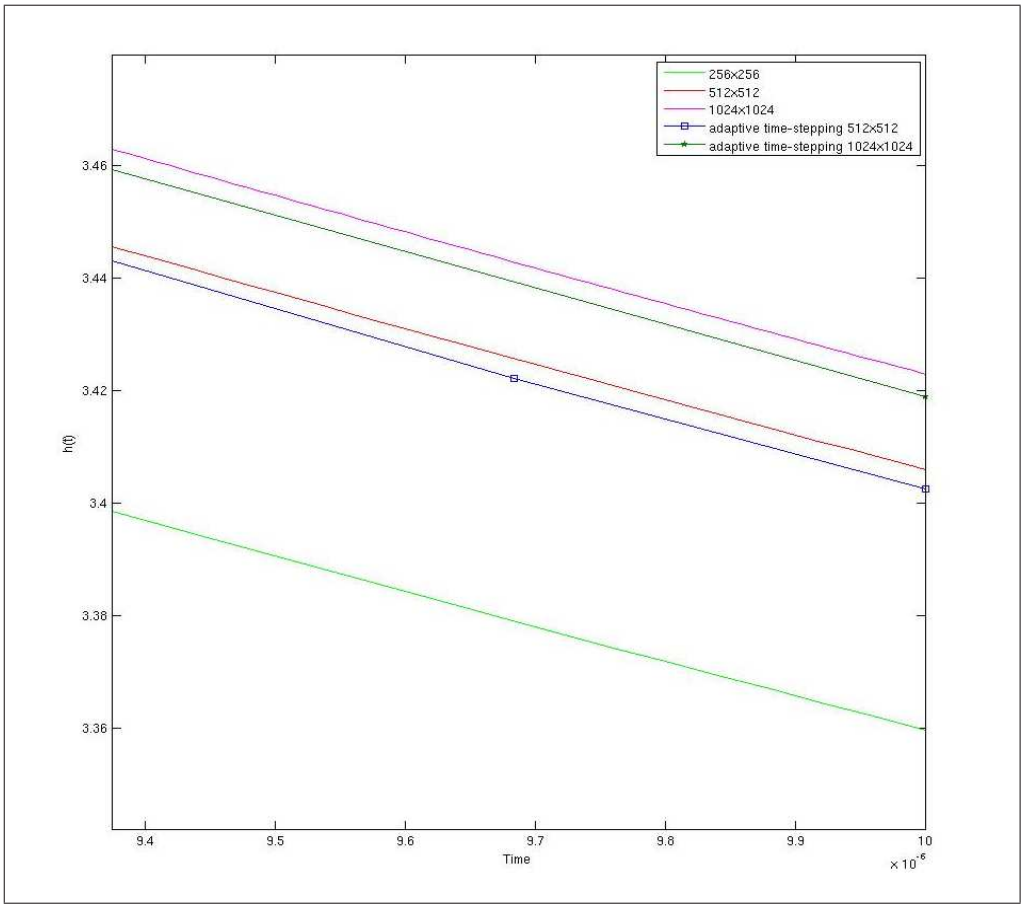

Figure 5.6: Figure shows a zoom-in feature for the end of the graphs that are originally presented in Figure 5.5. We magnify the figure at the end of simulation (i.e. $T = 1 \times 10^{-5}$) and use the measurement of the maximum height of droplet to indicate the accuracy of the simulation.

The adaptive time stepping approach, as presented here uses the number of V-cycles to control the step size. This implementation is previously described in Section 3.3.3. However, another common approach is to control step size based upon an indicator that uses a local error estimate. Gaskell et al. [81] used such an approach. Within their prediction-correction approach, the difference between the solutions from the use of the predictor

and that from the implicit corrector was used to obtain the error estimate which was, in turn, used to adjust the time step size. Although our approach is different, the implementation of adaptive time stepping in Campfire is capable of using other indicators, such as the one based upon this local error estimate.

For completeness, we extend the simulation with the adaptive time stepping approach presented in Figure 5.4, and show the evolution of time step sizes for a much larger time duration (i.e. $T = [0, 0.02]$ by which time the droplet is approaching the domain boundary). This is illustrated in Figure 5.7. Note that the sudden drop in the time step size at the end of simulation is so that the simulation ends precisely at the final time (i.e. $T = 0.02$). Results shown in this figure demonstrate the fast evolution of time step sizes from our aggressive, adaptive time stepping approach. In addition, it eventually settles towards a constant value as the droplet becomes smooth and highly diffuse.

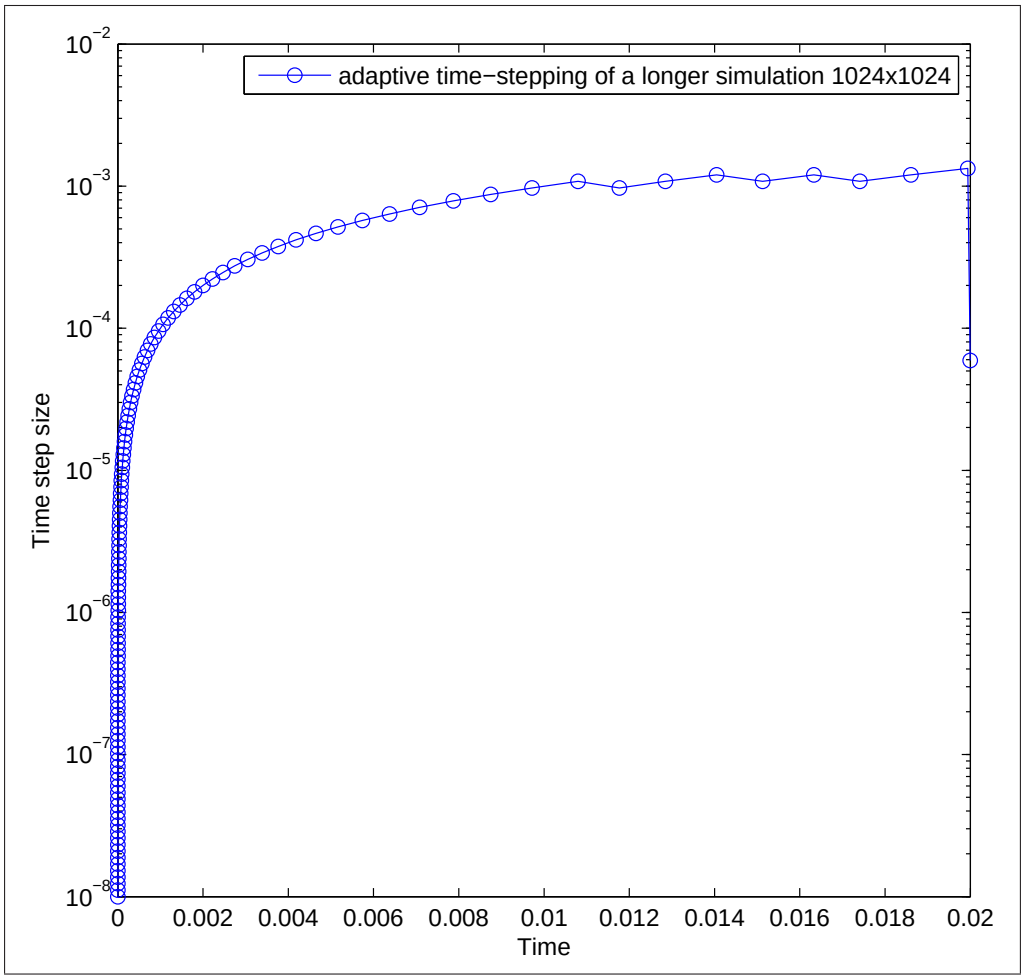

Figure 5.7: Figure shows the evolution of time step sizes from a much longer simulation using the adaptive time stepping approach demonstrated previously in Figure 5.4.

In the following section, the impact of using spatial adaptivity on the droplet spreading

model is discussed.

5.4.3.2 Spatial Adaptivity

In this section, spatial adaptivity (also called adaptive mesh refinement (AMR)) is applied to the droplet spreading model. The spatial adaptivity is previously described in Section 2.2.1. Its implementation in Campfire is discussed in Section 3.3.2. As in the previous subsection, the problem we consider here is with the original non-smooth initial condition which is presented in Equation (5.10). Such a problem features a distinct radial moving contact line which must be accurately resolved.

Here we choose three test cases for the purpose of demonstration. They are maximum levels of 5, 6 and 7 (level 4 has too few grid points to make a useful assessment). For simplicity, we continue to use the maximum level as our notation, however, it is worth noting that when associated with AMR, the level number does not mean a uniform grid, but instead means the highest level of local mesh refinement. More specifically, for each of the test cases, their highest levels of mesh refinement are equivalent to uniform grid resolutions of 256×256, 512×512 and 1024×1024. We further define the minimal level of refinement to be level 4 which has the equivalent grid resolution of 128×128.

From Algorithm 8 and the description in Section 3.3.2, the quality of the AMR is controlled by the problem-specific refinement and coarsening criteria. For this droplet spreading model, our adaptive refinement strategy is based upon a discrete approximation to the second derivative of the solution of variable h (i.e. $|\nabla^2 h|$). Within each mesh block, on every grid point, (i, j), the adaptive assessment is computed via:

$$\text{adaptive assessment}_{i,j} = |h_{i+1,j} + h_{i-1,j} + h_{i,j+1} + h_{i,j-1} - 4h_{i,j}|. \tag{5.20}$$

Then the maximum value of adaptive assessment is selected to represent this mesh block and compared against the user-defined refinement/coarsening criteria. The specific choices for these criteria must be obtained and evaluated from practice. In this case we define the refining criterion to be 0.01 and the coarsening criterion to be 0.001. Therefore, if adaptive assessment is greater than 0.01 (i.e. $dx^2|\nabla^2 h| > 0.01$), this mesh block is marked for refining, or if it is less than 0.001 (i.e. $dx^2|\nabla^2 h| < 0.001$), then this mesh block is marked for coarsening. This set of criteria is aggressive, so most of the computation is around the moving contact line. The stopping criteria for the multigrid solver are the same as described in Section 5.4.1.

In order to evaluate the AMR, we first use the approach with fixed time step size, and the end time $T = 1 \times 10^{-5}$. In Table 5.9, details of the three different test cases

are presented. For comparison, we also include the CPU time for these test cases when uniform grids (and fixed time step size) are employed. From this table, the efficiency gained from using AMR is demonstrated. For instance, on level 7 with AMR, the CPU time is only 19.3% of the one with using uniform grids. The average V-cycle per time step increases by half a cycle on level 7 with AMR.

Levels	δt	Time steps	Avg. V-cycles per time step	CPU time (seconds)	CPU time (seconds) from uniform grids
5	4×10^{-8}	250	5.0	175.1	303.2
6	2×10^{-8}	500	5.0	645.6	2345.7
7	1×10^{-8}	1000	5.5	3578.9	18521.1

Table 5.9: Table shows the details of three test cases using the aggressive AMR. CPU times from only using uniform grids are also included for comparison.

Having presented the CPU time, we further compare the number of grid points on the finest grids used in the uniform cases to the number of leaf grid points that are used in the adaptive cases. The concepts of leaf grid points and leaf blocks are previously described in Section 3.3.2. For adaptive cases, refining and coarsening are carried out dynamically, thus these numbers of leaf points are the maximum numbers that occurred in the simulations. In Table 5.10, this comparison is summarised. From this table, the computational workload saved by using the AMR compared to the use of uniform grids is seen to be substantial. For a particular case, on level 7 with AMR, the number of leaf points is less than 1.0% of the number of points on the finest grid from using uniform grids.

Levels	Total No. leaf grid points	Total No. grid points from uniform grids	Ratio between AMR and uniform grids
5	$2,048$	$65,536$	0.0313
6	$6,400$	$262,144$	0.0244
7	$10,240$	$1,048,576$	0.0098

Table 5.10: Comparison of the maximum number of leaf grid points used in adaptive test cases and the total number of grid points in uniform test cases. A ratio between the number of leaf points with AMR and the number of grid points with uniform grids is also presented.

We demonstrate that the use of AMR significantly improves the efficiency of the computation. This leads to the inevitable question: how accurate are the solutions from using the AMR? In order to determine the accuracy of our solution, we track the maximum

height of the droplet during the simulation with the AMR and compare against results from using the uniform grids. In Figure 5.8, the evolutions of the maximum height of the droplet from the three test cases (i.e. levels 5, 6, and 7) with the use of aggressive AMR are presented. Results from using uniform grids (and fixed time step size) are also presented (as previously presented in Figure 5.1). From this figure, we see that the use of the aggressive AMR produces quite similar results to the ones from using uniform grids.

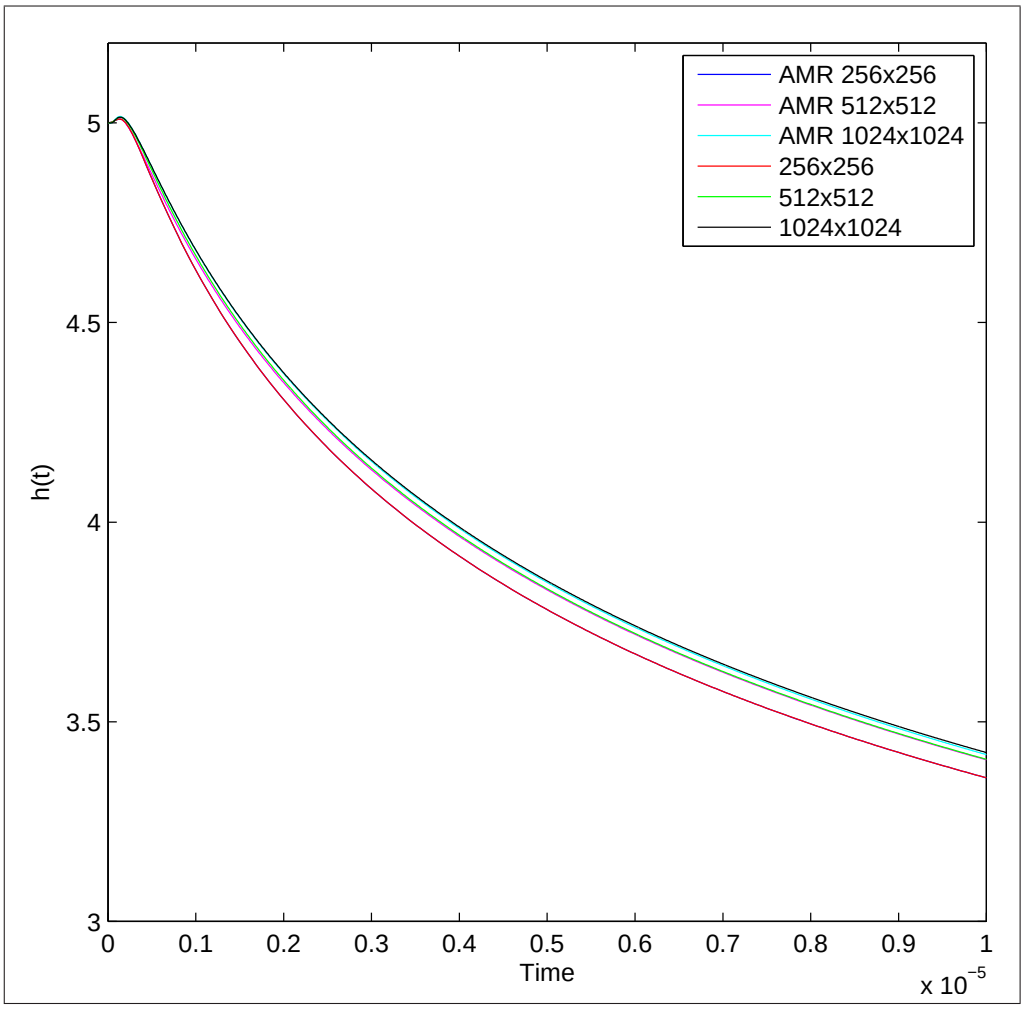

Figure 5.8: The evolution of the maximum height of the droplet from using the aggressive AMR and on uniform grids.

However, to give a further indication, a zoom-in feature is shown in Figure 5.9 which focusses on the solution at the end of simulation (i.e. $T = 1 \times 10^{-5}$). Results presented in this figure show the solutions from AMR are slightly less accurate than the ones from using uniform grids as the maximum level is increased. This is generally expected given the enormous reduction in degrees of freedom however the accuracy of the solutions is acceptable. By "acceptable", we mean that the accuracy of the level 7 solution with AMR,

although less than the one from level 7 with uniform grid, is much better than the one from level 6 with uniform grid (i.e. the maximum height of droplet is much closer to the one from level 7 with uniform grid). It may also be seen that the solution of the level 5 with AMR is almost identical to the one from level 5 with uniform grid. The values at the end of the simulations from level 7 are 3.423 from uniform grids, and 3.418 from the use of AMR. For level 6, the values are 3.406 from uniform grids, and 3.405 from the use of AMR. For level 5, the values are 3.35974 from uniform grids, and 3.35971 from the use of AMR.

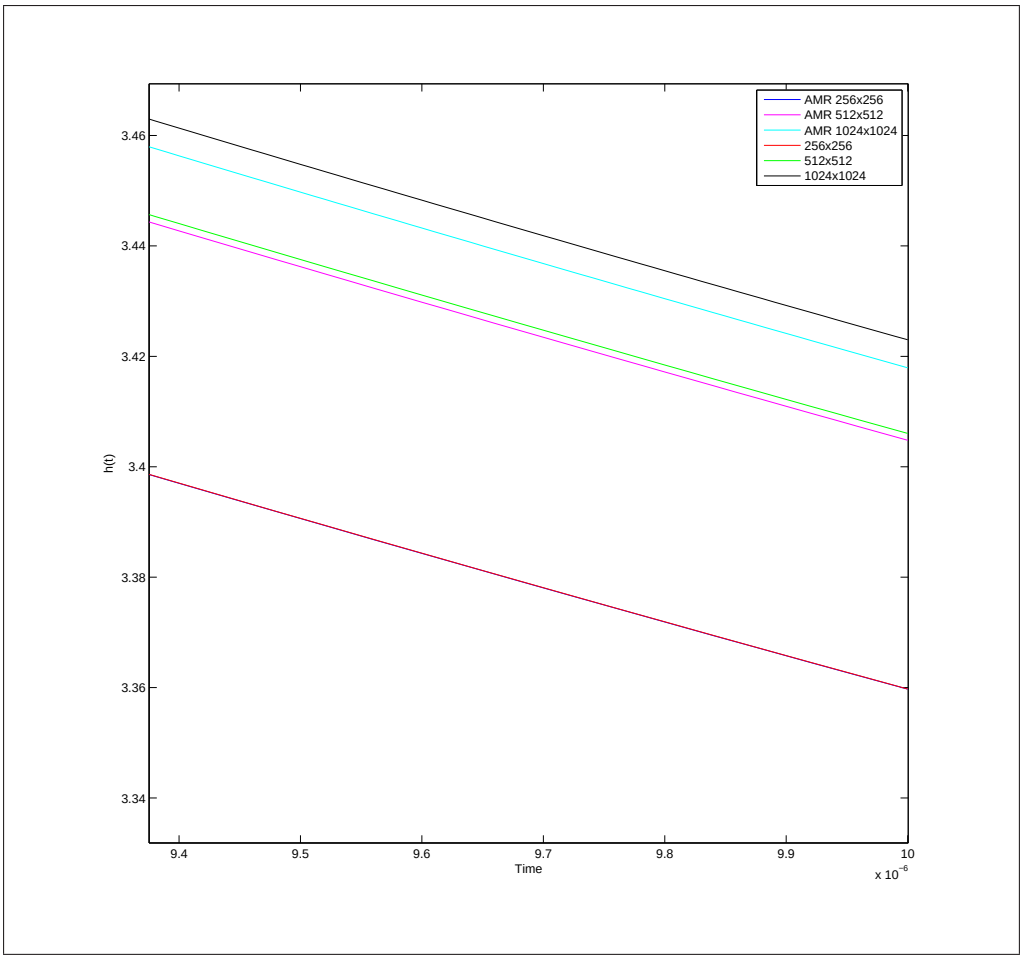

Figure 5.9: The maximum height of the droplet at $T = 1 \times 10^{-5}$ from using the aggressive AMR and uniform grids. This is a zoom-in of Figure 5.8 at the end of the simulation.

Having described an aggressive AMR approach, it is also possible to use a more conservative AMR approach. We demonstrate this use of conservative AMR with the level 7 test case. This conservative AMR is achieved by decreasing both the refining and the coarsening criteria by a factor of 10. We present the total number of grid points and the CPU time for this conservative AMR, in comparison with those obtained from using the

aggressive AMR and on uniform grids. This is summarised in Table 5.11. From this table, we see that the use of the conservative AMR leads to an increase of the total number of leaf points and the CPU time. Compared to the aggressive AMR, the number of grid points is increased by a factor of 1.68, and the CPU time is increased by a factor of 1.44. However, both values are still small compared to the corresponding ones with uniform grids.

Cases	Total No. grid points on the finest grid (or leaf points)	CPU time (seconds)
Uniform	$1,048,576$	18521.1
Aggressive AMR	$168,480$	3578.9
Conservative AMR	$292,896$	5137.6

Table 5.11: The total number of grid points (or leaf points) and the CPU time on the level 7 test case from using three different approaches: simulations with aggressive and conservative AMR and with uniform grids.

In Figure 5.10, we illustrate the evolutions of the maximum height of the droplet from these three approaches that are shown in Table 5.11. We also present in Figure 5.11 a zoom-in feature which focuses on the maximum height of the droplet at the end of simulation (i.e. $T = 1 \times 10^{-5}$). From these two figures, we see that using the conservative AMR approach provides a much more accurate solution than the one with the aggressive AMR. More specifically, values at the end of simulations shown in Figure 5.11 are 3.4230 from uniform grids, 3.4179 from the use of aggressive AMR, and 3.4225 from the use of conservative AMR.

The AMR implemented in Campfire, which is previously explained in Section 3.3.2, aims to dynamically adapt the mesh according to the evolution of the solution. We have shown results using both aggressive and conservative AMR approaches, however we have not yet illustrated the resulting meshes. Here we present snapshots of the evolution of the mesh refinement in both approaches during the simulations. These are shown in Figure 5.12 where a top-down view is presented. Different colours are used to identify different levels of mesh refinement. The red colour represents the finest mesh refinement (i.e. with resolution equivalent to 1024×1024). The yellow colour represents the mesh refinement with resolution equivalent to 512×512. The light blue colour indicates a coarser refinement which has a resolution equivalent to 256×256. The dark blue colour, which is further away from the centre of the graphs, has the coarsest mesh refinement, which is a resolution equivalent to 128×128. Results shown in this figure demonstrated the dynamic evolution of AMR during the simulations. It also suggests that if the interest

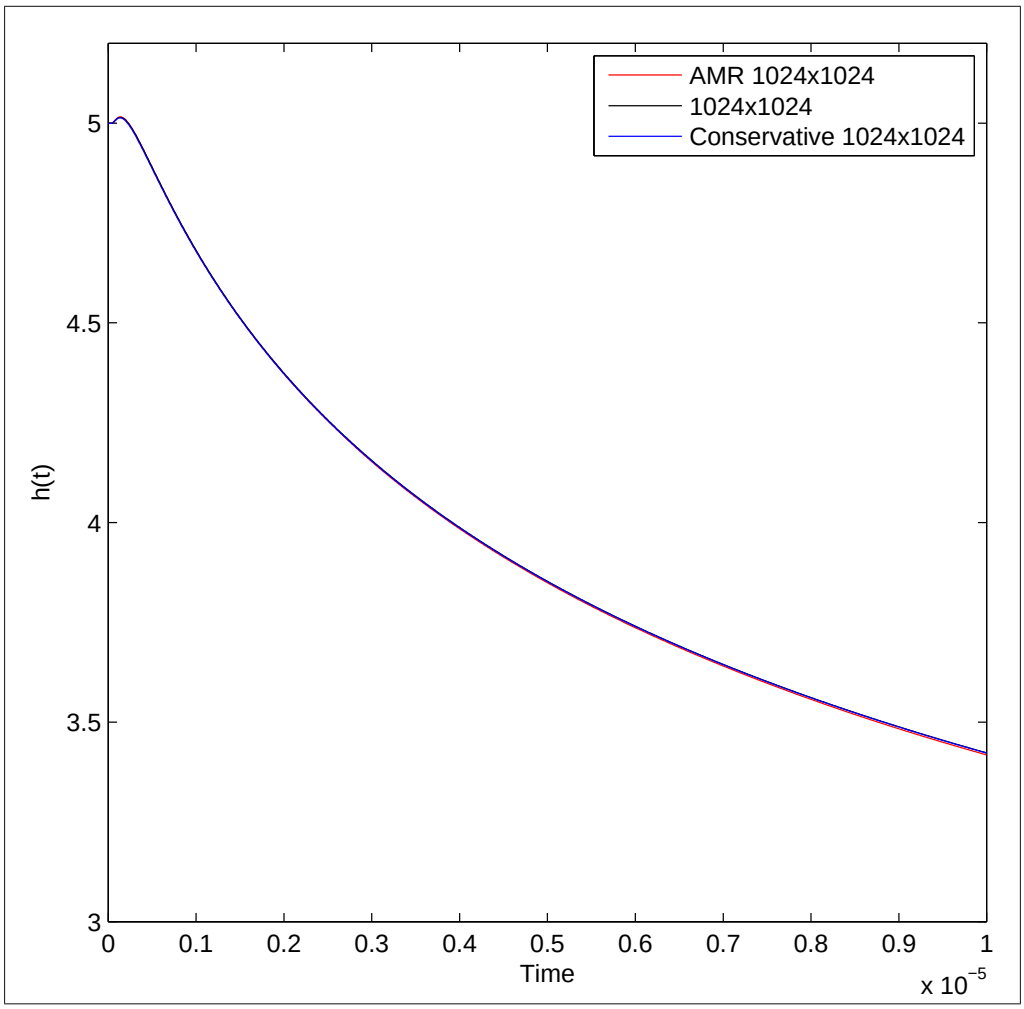

Figure 5.10: The evolution of the maximum height of the droplet using the three approaches which are presented as cases in Table 5.11.

is only at the moving contact line (or the area of the droplet), then the aggressive AMR approach has already successfully captured this region of interest. By using the aggressive AMR, a large amount of computations may be saved within the droplet. On the other hand, if the height and the shape of the droplet are also of interest, then the conservative AMR approach may be employed for a more accurate simulation, where the finest mesh refinement capture the whole droplet.

So far the simulations described have all been undertaken on a single workstation, and the program has been executed as a sequential code using only one CPU. The performance may be further improved by using parallel computing and typical results are described in the following section.

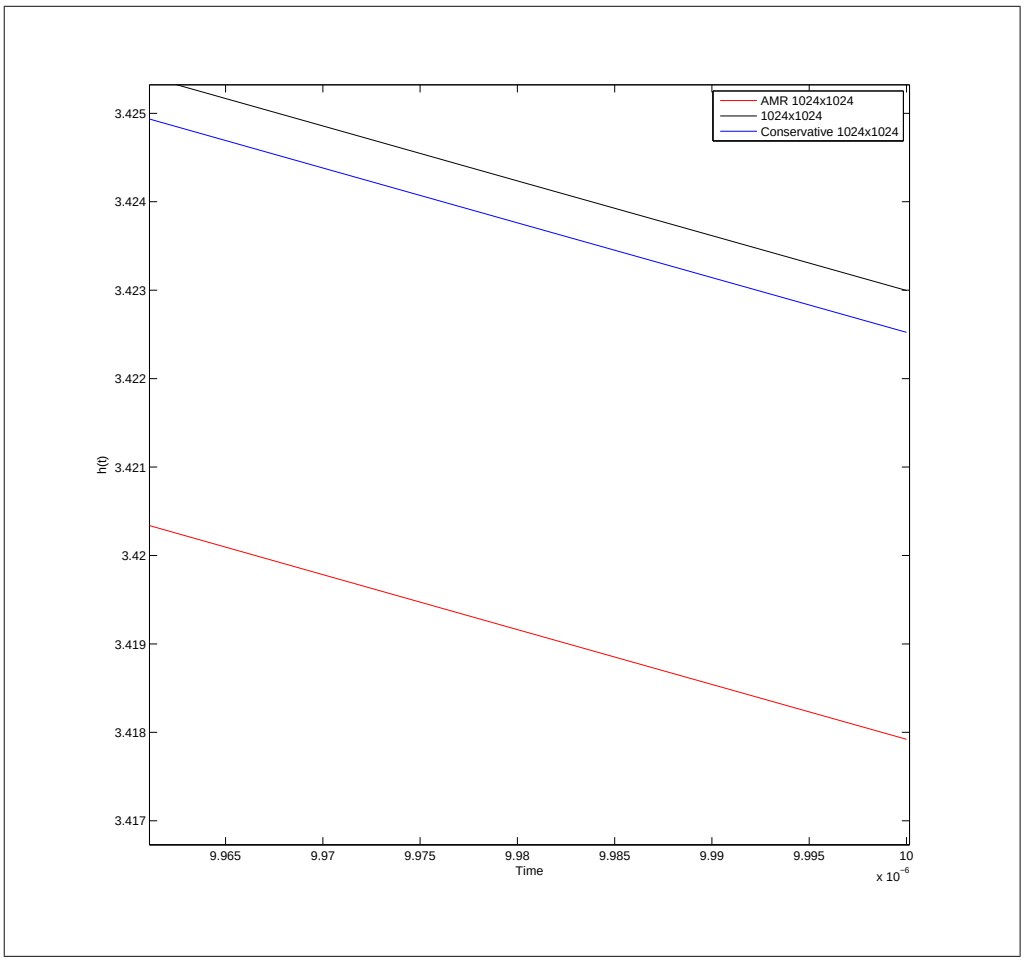

Figure 5.11: The zoom-in feature of Figure 5.10 at the end of its simulation. Solutions of three approaches are presented, as previously shown as cases in Table 5.11.

5.4.4 Parallel

In this section, results of using parallel computing are discussed. More specifically, the performance of our code in a parallel environment is described. Execution takes place on the high performance computing facility provided by the University of Leeds, named ARC2. Its capacity is 3040 cores in total (although the maximum practiced limit for each user is a few hundred cores), consisting of 8-core Intel E5-2670 2.6GHz processors (and commonly 2GB memory space associated with each core).

Unlike the problem in 3-D shown in the last chapter, which used more than 1000 cores, 2-D problems such as that considered here are more difficult to scale. This is caused by the lack of workload on coarser grids, and is previously discussed in Section 2.5.5. Therefore, in order to demonstrate the use of parallel computing on this 2-D droplet spreading model, only up to 64 cores from ARC2 are employed. We choose our test case

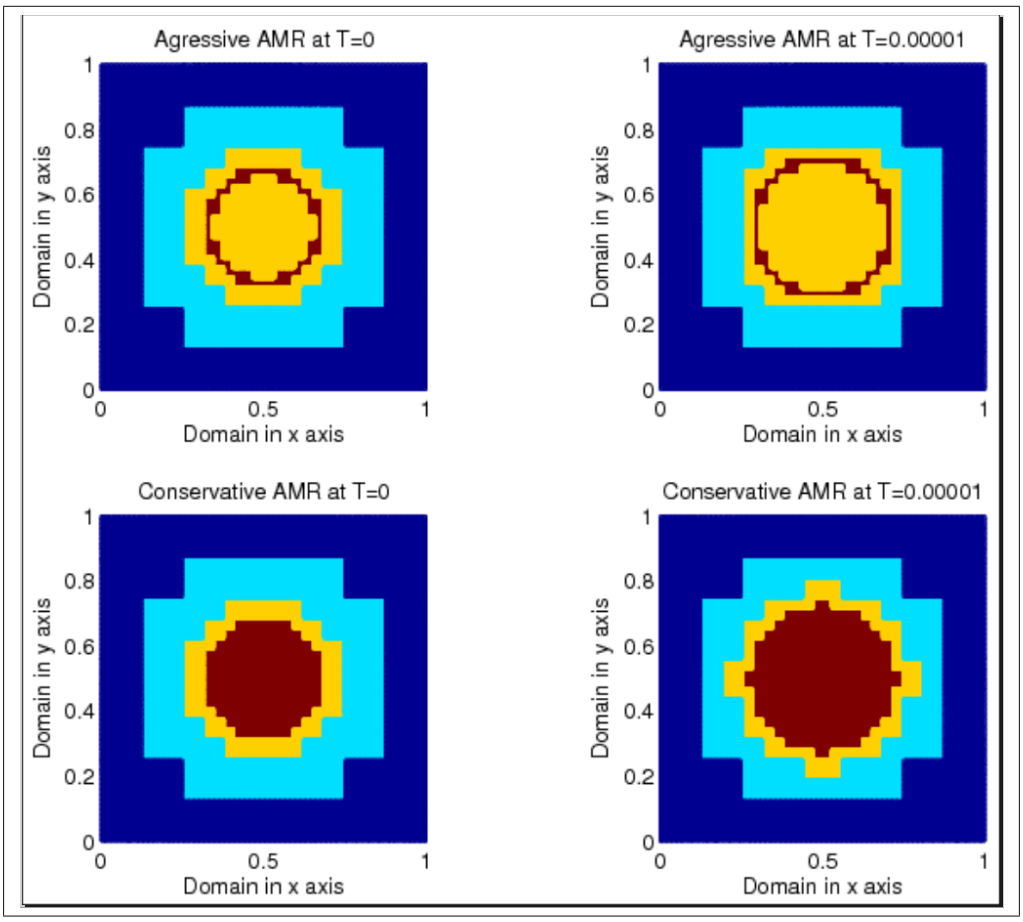

Figure 5.12: Figure shows two snapshots of the evolution of AMR during each adaptive simulation. Top-left is the aggressive AMR at $t = 0$ and top-right shows the aggressive AMR at $t = 1 \times 10^{-5}$. Bottom-left is the conservative AMR at $t = 0$ and bottom-right shows the conservative AMR at $t = 1 \times 10^{-5}$. Each colour represents a different level of mesh refinement: "red" - 1024×1024, "yellow" - 512×512, "light blue" - 256×256 and "dark blue" - 128×128.

for parallel computing to be 100 time steps with fixed time step size $\delta t = 1 \times 10^{-8}$ from simulations using the level 7 grid hierarchy. The coarsest grid is 16×16 and the block size is 8×8 (which means there are only 4 blocks on the coarsest grid). There are 4 pre- and post-smoothers for the multigrid solver and 4 iterations within the coarsest grid solver. We fix the number of V-cycles per time step to be 5.

First of all, consider uniform grids without AMR or adaptive time stepping. Results are presented in Table 5.12. From this table, we see continuous reductions in the execution time up to 32 cores. The use of 64 cores is less effective than the use of 32 cores, and so is not justified at this mesh level.

The measures of parallel performance are previously discussed in Section 2.5.1.2. Let's consider the parallel efficiency, which is previously presented in Equation (2.59).

No. cores	1	2	4	8	16	32	64
CPU time (seconds)	3282.6	1687.1	843.5	774.1	490.6	348.6	368.3

Table 5.12: The CPU times of using different numbers of cores in parallel on level 7, with uniform grids, which has the grid hierarchy $16 \times 16 - 1024 \times 1024$, and mesh block size is 8×8.

This is illustrated in Figure 5.13. From this figure, the parallel efficiency appears to be near optimal when using 2 and 4 cores. It deteriorates rapidly when 8 cores are employed, which may be caused by the fact that there is not enough workload on the coarser grids. Secondly, we present the speed-up which is previously explained in Equation (2.56). This is shown in Figure 5.14. Results from this figure shows a drop of speed-up from 32 to 64 cores. This reflects these CPU times presented in Table 5.12.

To further investigate this issue, an additional set of tests are done. These tests use the grid with resolution of 32×32 as the coarsest grid, instead of 16×16. The CPU times of these tests with seven different numbers of cores are presented in Table 5.13. Results from this table show a continuous reduction in time required up to 64 cores, and faster times than those presented in each case in Table 5.12. On the other hand, the overall pattern of the results suggests a similar qualitative behaviour of parallel efficiency and speed-up to those presented in Figures 5.13 and 5.14 respectively.

No. cores	1	2	4	8	16	32	64
CPU Time (seconds)	3264.1	1625.1	840.1	687.3	439.5	351.2	301.7

Table 5.13: The CPU times of using different numbers of cores in parallel on level 7 but with a finer coarsest grid (i.e. 32×32), i.e. a grid hierarchy of $32 \times 32 - 1024 \times 1024$, and mesh block size of 8×8.

So far these parallel tests show results of so-called strong scaling. This type of scaling analyses the same workload spread across different numbers of cores. As the number of cores doubles, the workload per core is therefore halved each time. There is another test for scalability, that is termed weak scaling, which keeps a constant workload assigned to each core.

For tests of weak scaling, we use a grid with resolution of 64×64 as the coarsest grid. The finest grid with resolution of 1024×1024 in this setting has 5 levels of grids in its grid hierarchy. The weak scaling tests we present in Table 5.14 consist of four cases (i.e. levels 5, 6, 7 and 8). They have the finest grids with resolutions of 1024×1024, 2048×2048, 4096×4096 and 8192×8192 respectively. Fixed time step sizes are used in all four simulations, and on level 5 (1024×1024) the $\delta t = 1 \times 10^{-8}$. The δt is halved each time

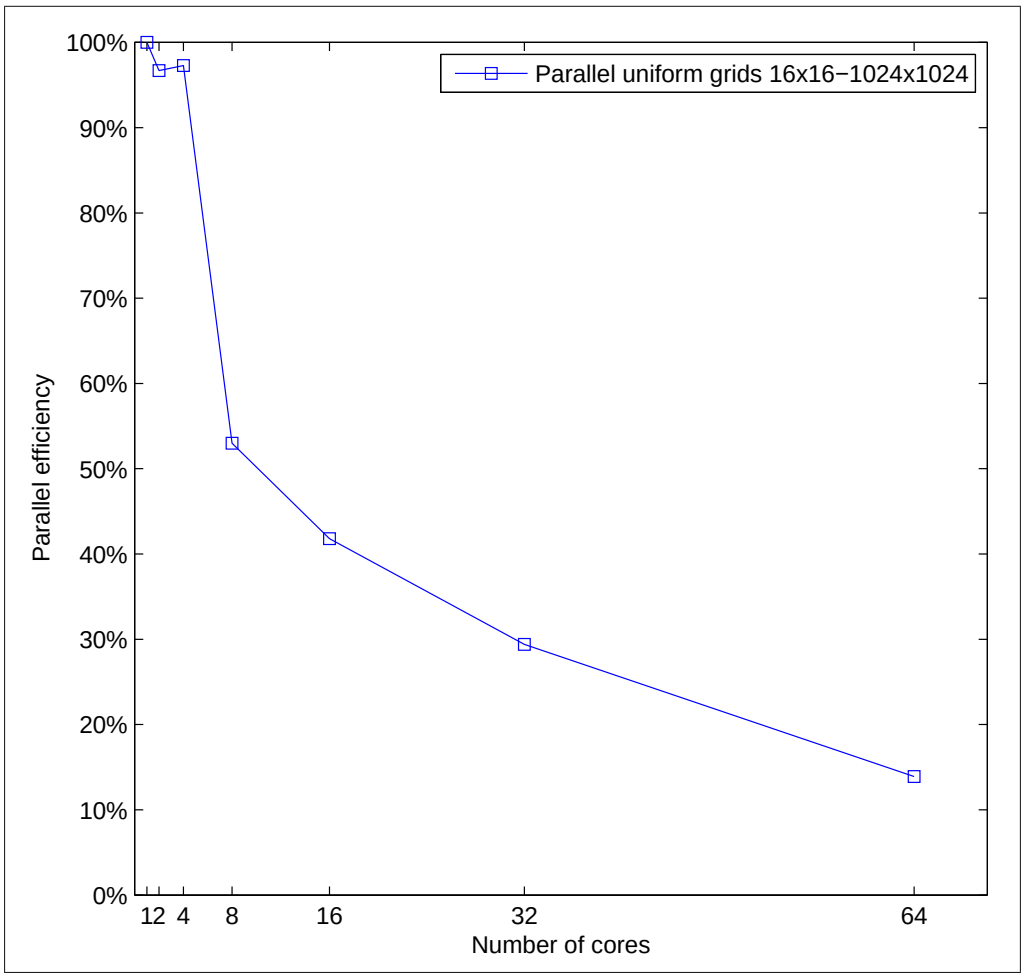

Figure 5.13: The parallel efficiency (see Equation (2.59)) from using different numbers of cores. This test is done on level 7 with uniform grids, which has the grid hierarchy $16 \times 16 - 1024 \times 1024$, and timings are presented in Table 5.12.

we refine the grid resolution to ensure the stability of the simulation. The number of V-cycles within each time step is fixed to 5. In an ideal world, all the run times in Table 5.14 should be the same. On the other hand, results in this table indicate that for a 2-D problem such as this the weak scaling is quite poor, primarily because the communications become a dominating factor on the coarsest grid. We are unable to increase the size of the coarsest grid as the problem size grows because this loses multigrid efficiency or requires increased work for an exact nonlinear solve.

Having presented the multigrid performance in a parallel environment we conclude that we should use parallelism to enhance capability more than to enhance performance. In this spirit we combine all techniques that are previously demonstrated in order to provide the capability to investigate further the convergence tests with the original steep initial condition. This is undertaken in the next section, along with a general discussion

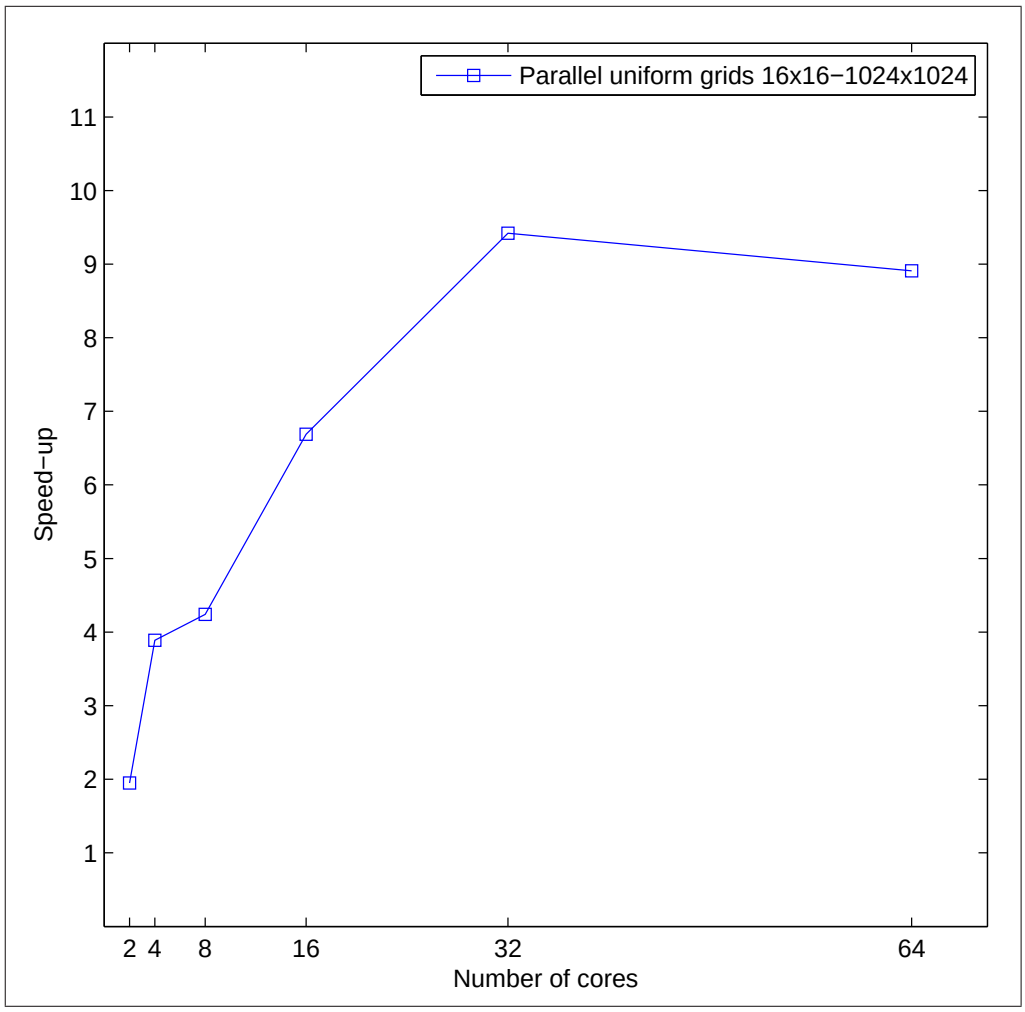

Figure 5.14: The speed-up (see Equation (2.56)) which compares the CPU times from using different numbers of cores with the CPU time of the sequential case. This test is done on level 7 with uniform grids, which has the grid hierarchy $16 \times 16 - 1024 \times 1024$, and timings are presented in Table 5.12.

of the application of our software to this problem.

5.4.5 Discussion

Previously in this chapter temporal adaptivity, spatial adaptivity and parallel computing have been individually applied with our multigrid solver from Campfire. The results from each technique are separately verified. Here we combine all these techniques, and the resulting robust solver provides us the opportunity to investigate convergence tests with the original steep initial condition (presented in Equation (5.10)) and its corresponding boundary condition, shown in Equation (5.4).

As mentioned in Section 5.4.2.1, the original initial condition that is used by Gaskell

Levels	No. Cores	CPU time (seconds)
5	1	3299.0
6	4	4151.2
7	16	7328.4
8	64	22500.7

Table 5.14: Table shows results of the described weak scaling. Note the coarsest grid is changed to 64×64 for all tests in this table.

et al. in [81] (see Equation (5.10)) is discontinuous in its first derivative and initially very steep. Our convergence tests shown in Section 5.4.2.1 have used a smoother initial condition instead. For completeness, convergence tests with the use of this steep initial condition, uniform grids and fixed time step size are first applied, and the results and discussions are presented. It is expected that using this initial condition shown in Equation (5.10) may not provide a second order convergence rate at an early time such as $T = 1 \times 10^{-5}$ due to the discontinuity in the first derivative. On the other hand, as the droplet evolves, its features are soon smoothed, especially around the moving contact line. We therefore also consider taking a long simulation in the expectation that an overall second order convergence rate may be obtained.

The results from convergence tests at time $T = 1.2 \times 10^{-2}$ are presented in Table 5.15. Simulations from only levels 4, 5 and 6 are shown (with a 16×16 coarsest grid for all three cases), the reason for the absence of level 7 being the excessive work with fixed δt and uniform refinement. Results in this table demonstrate an overall convergence rate that is already close to second order. Results from this set of uniform grid tests also demonstrates that for longer simulations, the approach with the use of uniform grids and fixed time step sizes is not viable. Considering the level 6 simulation, by using 16 cores on ARC2, the average time per time step is around 1.4 seconds. The whole simulation takes about 233.3 hours. If the level 7 simulation were to be undertaken, according to the performance of our multigrid solver, the estimated time for such a simulation is more than two and half months using 16 cores.

As directly noted, this provides a strong motivation for using the spatial and the temporal adaptivity together and gain additional efficiency from using the parallel computing. Four different test cases are chosen and their highest levels of mesh refinement are equivalent to the grid resolutions of 512×512, 1024×1024, 2048×2048 and 4096×4096 respectively. A 16×16 coarsest grid is used for all cases, therefore, they are noted as levels 6, 7, 8 and 9 respectively. The conservative AMR approach which is described previously is employed here. The end time is chosen to be $T = 2.0 \times 10^{-2}$, which is a little

For variable h						
Levels	δt	Time steps	Infinity norm	Ratio	Two norm	Ratio
4	8×10^{-8}	150,000	-	-	-	-
5	4×10^{-8}	300,000	1.466×10^{-3}	-	5.068×10^{-4}	-
6	2×10^{-8}	600,000	4.116×10^{-4}	3.56	1.413×10^{-4}	3.59
For variable p						
4	8×10^{-8}	150,000	-	-	-	-
5	4×10^{-8}	300,000	1.542×10^{0}	-	5.384×10^{-1}	-
6	2×10^{-8}	600,000	4.312×10^{-1}	3.58	1.472×10^{-1}	3.66

Table 5.15: Results show the differences in consecutive solutions measured in the stated norm, followed by the ratio of consecutive differences from the droplet spreading model with the original initial condition which is presented in Equation (5.10). These results are generated with solutions at $T = 1.2 \times 10^{-2}$ using uniform grids and fixed time step sizes. Computations are carried out using 16 cores on ARC2.

longer than the simulations shown in Table 5.15. The convergence tests are carried out, and results are summarised in Table 5.16. Results shown in this table clearly demonstrate second order convergence for both variables h and p. These results are generated using the ARC2 facility, and individually occupied 16 cores throughout each simulation.

To sum up, the droplet spreading model is presented in Section 5.1, and its fully-discrete system, through the use of the five-point stencil and the BDF2 method, is presented in Section 5.2. This model has already been solved by Gaskell et al. in [81] which enables us to validate the parallel and adaptive multigrid solver we have developed. Note that [81] does not support AMR as our solver does. Section 5.3 provides an overview of the new features added to Campfire to support this capability. A number of these newly implemented features are used throughout this thesis for subsequent problems. Due to the clear and comprehensive presentation made in [81], we are able to replicate the results in [81] and validate the implementation of our multigrid solver. Results of the validation are presented in Section 5.4.1. Then results of the convergence tests and the multigrid performance are discussed in Section 5.4.2. To further enhance the efficiency, the temporal adaptivity and the spatial adaptivity are discussed. Results are separately generated and verified in Section 5.4.3. The use of parallel computing is also included and is discussed in Section 5.4.4. In the following section, the model of fully-developed flows is described.

For variable h						
Levels	Starting δt	Time steps	Infinity norm	Ratio	Two norm	Ratio
6	1×10^{-8}	1418	-	-	-	-
7	5×10^{-9}	2011	2.480×10^{-5}	-	1.117×10^{-5}	-
8	2.5×10^{-9}	25184	6.174×10^{-6}	4.02	2.737×10^{-6}	4.08
9	1.25×10^{-9}	472368	1.544×10^{-6}	3.99	6.864×10^{-7}	3.99
For variable p						
6	1×10^{-8}	1418	-	-	-	-
7	5×10^{-9}	2011	5.321×10^{-2}	-	2.007×10^{-2}	-
8	2.5×10^{-9}	25184	1.324×10^{-2}	4.02	4.873×10^{-3}	4.12
9	1.25×10^{-9}	472368	3.309×10^{-3}	3.99	1.208×10^{-3}	4.04

Table 5.16: Results show the differences in consecutive solutions measured in the stated norm, followed by the ratio of consecutive differences from the droplet spreading model with the original initial condition which is presented in Equation (5.10). These results are generated with solutions at $T = 2.0 \times 10^{-2}$ using spatial adaptivity and temporal adaptivity. Computations are carried out using 16 cores on ARC2.

5.5 Fully-Developed Flows Model Outline

The model of fully-developed flows is previously discussed in Section 1.3.1.1. In the same way as the droplet spreading model, it is derived from the Navier-Stokes equations (see Equations (1.12) and (1.13)) through the use of the lubrication approximation. This model has already been proposed and solved by Gaskell et al. in [82] and Kalliadasis et al. in [125]. Here it is solved by the described parallel, adaptive multigrid solver with the use of fully-implicit time stepping.

This model of fully-developed flows consists of two dependent variables: $h(x,y,t)$ and $p(x,y,t)$. The former measures the thin film flow thickness, and the latter represents the pressure. For clarity, this model is also presented here, where the thin film equation is given as

$$\frac{\partial h}{\partial t} = \frac{\partial}{\partial x}\left[\frac{h^3}{3}\left(\frac{\partial p}{\partial x} - 2\right)\right] + \frac{\partial}{\partial y}\left[\frac{h^3}{3}\left(\frac{\partial p}{\partial y}\right)\right]. \tag{5.21}$$

Throughout this fully-developed flow, the pressure field satisfies

$$p = -6\triangle(h+s) + 2\sqrt[3]{6}N(h+s), \tag{5.22}$$

where $N = Ca^{1/3}\cot\alpha$ measures the relative importance of the normal component of gravity [25]. See [82], for example, for more details of the derivation.

In Equation (5.22) $s(x,y)$ is the topography of the substrate: an example shown previ-

ously in Figure 1.1 consists of three half-cylinder-like obstacles. It may be defined otherwise, for example, local or span-wise peaks and trenches. The topographies may also be modified to simulate a physical phenomenon of falling film with turbulence wires, such as the example shown in Figure 1.1 with three half-cylinder-like obstacles and $\alpha = 90°$, which is described by Raach and Somasundaram in [183]. There are several different choices of topographies that are presented in this chapter. Details of each of them are described later, and all topographies are further assumed to be non-time-dependent. The computational domain Ω is rectangular. For boundary conditions, it is generally assumed that there exists zero flux at boundaries. Therefore zero Neumann boundary conditions are applied, with the exception of the upstream boundary, which represents a source of flow. For instance, a constant Dirichlet boundary condition on h may be used for the upstream boundary. Thus the boundary conditions are defined as the following. Note $x = 0$ represents the upstream, $x = 1$ represents the downstream, $y = 0$ and 1 represent either side of the flow and g is a given function.

$$\begin{aligned} &h(x=0,y) = g, \ \ \frac{\partial p}{\partial x}|_{x=0} = 0, \ \ \frac{\partial h}{\partial x}|_{x=1} = \frac{\partial p}{\partial x}|_{x=1} = 0, \\ &\frac{\partial p}{\partial y}|_{y=0} = \frac{\partial p}{\partial y}|_{y=1} = \frac{\partial h}{\partial y}|_{y=0} = \frac{\partial h}{\partial y}|_{y=1} = 0. \end{aligned} \tag{5.23}$$

It is worth noting that this model of the fully-developed flows is presented as a non-time-dependent system in [82] and [125]. The thin film Equation (5.21) is modified into a time-dependent PDE in the model solved here. This is suggested by Gaskell et al. [82] as a viable approach. In addition, based upon the boundary conditions presented in Equation (5.23), the time-dependent system will eventually reach a solution of steady state. Another reason to use this time-dependent system is so we may study the situation where the incoming flow from the upstream direction is time-dependent (e.g. a flow that has waves).

This is on-going research, and will be discussed again later in this thesis. The initial condition $h(x,y,t = 0)$ is presented later in this chapter, and once $h(x,y,t = 0)$ is defined, the initial condition for p may be obtained. In the following section, discretization schemes that are used for the model of fully-developed flows are described.

5.6 Discretization Schemes for the Model of Fully-Developed Flows

The presented model of fully-developed flows (Equations (5.21) and (5.22)) is a nonlinear, coupled system of equations. However, only the thin film equation (Equation (5.21))

is time-dependent, and thus requires a temporal discretization scheme. Furthermore, an implicit temporal discretization scheme is applied. Due to the simplification made through the use of the lubrication approximation, this model is only solved in a 2-D situation. In this section, the choices of discretization schemes for this 2-D model are described and the resulting algebraic system is presented.

For the spatial discretization scheme, the central FDM with the five-point stencil which is shown in Equation (2.3) is applied. The resulting system from the thin film equation is then discretized by the fully implicit BDF2 method. Note the BDF1 scheme has to be employed for the very first time step, and the reason is previously described in Section 3.3.3. For the purpose of demonstration, here we present the BDF2 method with a fixed δt. It is straightforward if the adaptive time stepping of the BDF2 method which is presented previously in Equation (2.14) is to be used.

When the five-point stencil and BDF2 method with fixed δt are applied to Equation (5.21) and five-point stencil is applied to Equation (5.22), the resulting fully-discrete system of equations at each time step is very similar to Equations (5.5) and (5.6), considered in the first half of this chapter, but without the precursor film and the disjoining pressure term. The thin film equation becomes:

$$
\begin{aligned}
&\frac{h_{i,j}^{\tau+1} - (\frac{4}{3}h_{i,j}^{\tau} - \frac{1}{3}h_{i,j}^{\tau-1})}{\delta t} = \\
&\frac{2}{3dx^2}\Bigg\{ p_{i+1,j}^{\tau+1}\left[\frac{1}{2}\left(\frac{\left(h_{i+1,j}^{\tau+1}\right)^3}{3} + \frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right)\right] + p_{i-1,j}^{\tau+1}\left[\frac{1}{2}\left(\frac{\left(h_{i-1,j}^{\tau+1}\right)^3}{3} + \frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right)\right] \\
&\quad - p_{i,j}^{\tau+1}\left[\frac{1}{2}\left(\frac{\left(h_{i+1,j}^{\tau+1}\right)^3}{3} + \frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right) + \frac{1}{2}\left(\frac{\left(h_{i-1,j}^{\tau+1}\right)^3}{3} + \frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right)\right] \\
&\quad - 2dx\left[\frac{1}{2}\left(\frac{\left(h_{i+1,j}^{\tau+1}\right)^3}{3} + \frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right) - \frac{1}{2}\left(\frac{\left(h_{i-1,j}^{\tau+1}\right)^3}{3} + \frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right)\right] \\
&\quad + p_{i,j+1}^{\tau+1}\left[\frac{1}{2}\left(\frac{\left(h_{i,j+1}^{\tau+1}\right)^3}{3} + \frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right)\right] + p_{i,j-1}^{\tau+1}\left[\frac{1}{2}\left(\frac{\left(h_{i,j-1}^{\tau+1}\right)^3}{3} + \frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right)\right] \\
&\quad - p_{i,j}^{\tau+1}\left[\frac{1}{2}\left(\frac{\left(h_{i,j+1}^{\tau+1}\right)^3}{3} + \frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right) + \frac{1}{2}\left(\frac{\left(h_{i,j-1}^{\tau+1}\right)^3}{3} + \frac{\left(h_{i,j}^{\tau+1}\right)^3}{3}\right)\right]\Bigg\},
\end{aligned}
\tag{5.24}
$$

and the pressure field equation becomes

$$p_{i,j}^{\tau+1} + \frac{6}{dx^2}\Bigg\{ \left(h_{i+1,j}^{\tau+1} + s_{i+1,j}\right) + \left(h_{i-1,j}^{\tau+1} + s_{i-1,j}\right) \left(h_{i,j+1}^{\tau+1} + s_{i,j+1}\right) + \left(h_{i,j-1}^{\tau+1} + s_{i,j-1}\right) -4\left(h_{i,j}^{\tau+1} + s_{i,j}\right) \Bigg\} - 2\sqrt[3]{6}N\left(h_{i,j}^{\tau+1} + s_{i,j}\right) = 0. \tag{5.25}$$

The same multigrid solver that is previously described in Section 5.3, and that has been validated for the droplet spreading model, is used here for the model of fully-developed flows as well. We do not repeat the description of the solver therefore. In the following section, results of simulations of fully-developed flows by using this described parallel, adaptive multigrid solver from Campfire are discussed.

5.7 Results for the Model of Fully-Developed Flows

Results generated by using our multigrid solver are presented here. This solver is previously used for solving the droplet spreading model, and is described in Section 5.3 along with other implementations required by the model of fully-developed flows. Firstly, validation against existing results is discussed in Section 5.7.1. Then convergence tests for these simulations are carried out, and their results are presented in Section 5.7.2. A general discussion is made in Section 5.7.3 to conclude the model of fully-developed flows.

5.7.1 Validation

In this section, our solutions are validated against existing published results, as presented in [125]. It is worth noting that the model used by Kalliadasis et al. in [125] consists of a 1-D fourth order non-time-dependent PDE [1], instead of the equivalent coupled system with two second order equations used here. In Section 1.1, this relation between a fourth order PDE and two coupled second order equations is described (see Equations (1.10) and (1.11)). In the same way, it may be seen that the single fourth order PDE in [125] is equivalent to the presented coupled system of second order equations.

Kalliadasis et al. demonstrated six test cases with different widths of peaks and trenches in Figs. 10 and 11 in [125]. They are peaks with widths 1, 5 and 10. For clarity, they are denoted as peaks 1, 2 and 3 respectively in this chapter. Additionally,

[1] The interested reader is directed to Equation (4) in [125] for this single fourth order PDE.

there are trenches with widths 1, 5 and 10. Similarly they are denoted as trenches 1, 2 and 3 respectively. In this section, we validate all 6 cases with our results. It is worth noting that the results from [125] are 1-D. Thus the 2-D results we present here are using spanwise peaks and trenches, so the "cross-section" features of our results can be compared to the 1-D results shown in [125]. For ease of presentation we present our results as the view of the whole solution from a horizontal view point, rather than remove and select parts of it. Furthermore, because they are spanwise topographies, narrow domains may be used to generate these essentially 1-D results.

First of all, we demonstrate the validity of using such a narrow domain. This is done by running a specific testing case with a full square domain and a narrow rectangular domain. This narrow domain has the same length in the x direction but only a quarter length in the y direction. We demonstrate this with the feature of topography of trench 2 in Figure 5.15. Trench 2 is chosen for this validation, however other cases can equally provide comparisons. Figure 5.15 (a) shows the full domain feature (i.e. domain size is 20×20) of topography of trench 2, and Figure 5.15 (b) is the narrow domain feature (i.e. domain size is 20×5). The implementation for the full domain in Campfire uses sixteen 8×8 blocks to form a square coarsest grid with 4 rows and 4 columns of blocks (blocks on finer levels are refined from those on the coarsest grid). The coarsest grid of the narrow domain is implemented with four 8×8 blocks with 1 row and 4 columns. It is worth noting in our settings $dx = dy$. There are four different level of refinements in the shown features in Figure 5.15. For the full domain, they are mesh refinements with possible finest resolutions of 32×32, 64×64, 128×128 and 256×256. For the narrow domain, they are mesh refinements with possible finest resolutions of 32×8, 64×16, 128×32 and 256×64. Finer grids are considered later for validations against results in [125].

The presented topography in Figure 5.15 is defined by Kalliadasis et al. in [125] and subsequently defined by Gaskell et al. in a similar way in [82]. The topography is defined via arctangent functions, and is given as

$$\begin{aligned} s(x,y) = s_0 &\left[\tan^{-1}\left(\frac{x^* - l_t/2}{\gamma l_t}\right) + \tan^{-1}\left(\frac{-x^* - l_t/2}{\gamma l_t}\right)\right] \\ &\times \left[\tan^{-1}\left(\frac{y^* - w_t/2}{\gamma l_t}\right) + \tan^{-1}\left(\frac{-y^* - w_t/2}{\gamma l_t}\right)\right], \end{aligned} \tag{5.26}$$

where l_t is the length of the topography [2], w_t is the width of the topography (for the spanwise features presented here, $w_t =$ domain width), $s_0 = 1$ is the topography height/depth

[2] This is denoted W in [125], and equals to 1, 5 and 10 for corresponding topography features.

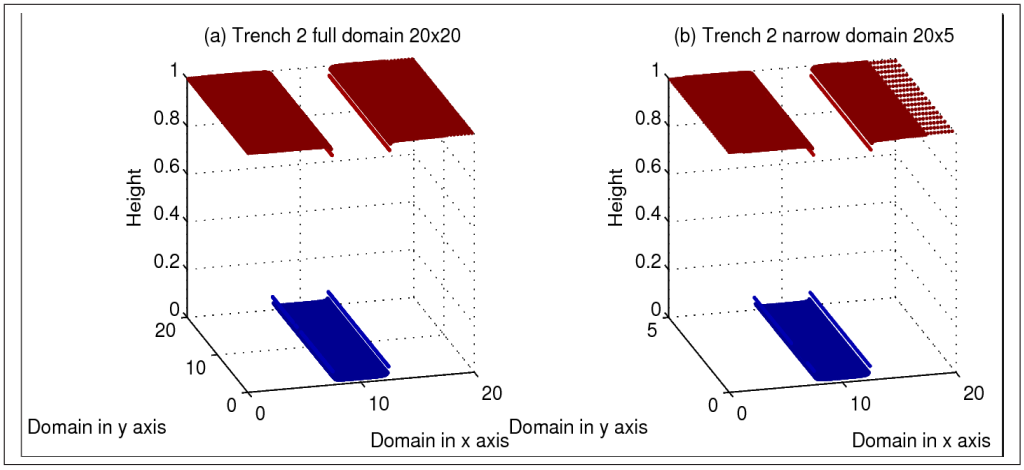

Figure 5.15: Figure (a) shows the topography of trench 2 with a full square 20×20 domain; Figure (b) shows the topography of trench 2 with a narrow domain of size 20×5.

[3] and $\gamma = 0.001$ is a parameter which defines the steepness of the topography [4]. The coordinate system, (x^*, y^*), for defining the topography is introduced by Gaskell et al., where (x_t, y_t) is the centre of the topography: $(x^*, y^*) = (x - x_t, y - y_t)$, with (x, y) the Cartesian coordinates.

Due to the use of a time-dependent system, initial conditions are required for variables h and p. The initial condition for h is given as

$$h^{t=0} = 1 - s, \tag{5.27}$$

which is a flat free surface. The corresponding initial pressure $p(x, y, t = 0)$ is defined via Equation (5.22). The boundary condition that is presented in Equation (5.23) is employed here with $g = 1$. Then within each time step, the coupled system is solved by our multigrid solver. This solver performs 2 pre- and post-smoothers on each level of the grid, except the coarsest grid where a fixed number of 60 iterations are carried out. The stopping criteria are based upon the infinity norm of residuals from both variables, as described in Section 5.4.1 for solving the droplet spreading model. Here an absolute and a relative stopping criteria are applied, and at least one of them must be satisfied in order to successfully conclude computation within each time step. The former terminates the current time step when the norm is smaller than 10^{-9} and the latter determines to start the next time step if the norm from the first V-cycle is reduced by a factor of 10^{-6}. The converged steady-state solution is said to be reached if, within a time step, only one multigrid V-cycle reduces the infinity norm to smaller than 10^{-9}. Having described the topography, the converged

[3] This is denoted D in [125].
[4] This is denoted δ in [125].

steady state solutions that are obtained using the two settings shown in Figure 5.15 are presented in Figure 5.16 viewed from horizontal view point. Two solutions in this Figure are almost identical. The conservative adaptive mesh refinement (AMR) that is applied to the droplet spreading model is used here to speed up the simulations. We also include the adaptive time stepping for additional efficiency in order to quickly reach the steady state. This simulation, and others that are presented later in this section, also benefited from using the ARC2 facility, and 16 cores are employed for each individual simulation. These techniques have already been described previously with the droplet spreading model, and they are used here in a very similar way. Therefore, they are not described repeatedly. Even with these techniques, however a significant run time is required to fully converge the flows to the steady state.

To numerically compare these two solutions presented in Figure 5.16, the infinity norm and the two norm of the differences between the solutions are considered. It is worth noting that there is no solution restriction required since two solutions are from the same grid hierarchy. The solutions on each grid point are paired and compared with their corresponding solutions on the other domain, thus only a quarter of the solution from full domain is compared to the whole size of the solution from the narrow domain. These numerical values are shown in Table 5.17. Results in this table indicate the solutions generated from full and narrow domains are almost identical and thus validate the use of the narrow domain is suitable for this type of problem.

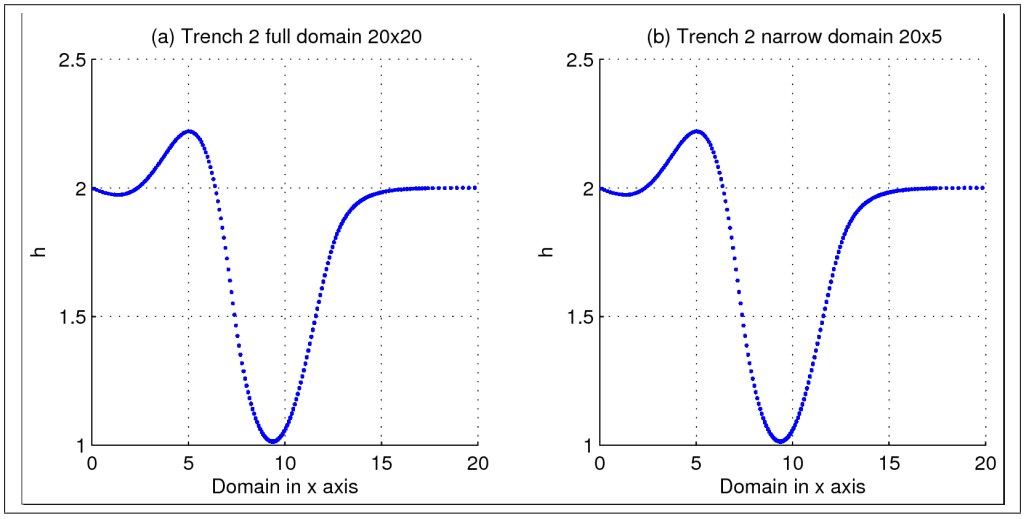

Figure 5.16: Figure (a) shows the film thickness over trench 2 with a full square 20×20 domain; Figure (b) shows the film thickness over trench 2 with a narrow domain of size 20×5.

To validate our solutions against those presented in [125], finer grids (with narrow

For variable h	
Infinity norm	9.504×10^{-14}
Two norm	2.479×10^{-14}
For variable p	
Infinity norm	4.821×10^{-13}
Two norm	8.061×10^{-14}

Table 5.17: Values of the infinity norm and the two norm of the differences of the solutions generated using a full square domain and a narrow rectangular domain.

rectangular domains) are employed. Some of our results shown here are from simulations with 7 levels of grids, they are 32×8, 64×16, 128×32, 256×64, 512×128, 1024×256 and 2048×512. Others are from simulations with 6 levels of grids. Since our interest for the model of fully-developed flows is in its converged steady-state solutions, the C2F solution prolongation technique described in Section 4.2 is used here. When the converged steady state solution is obtained from using the grid hierarchy $32 \times 8 - 256 \times 64$, for example, this solution is prolongated to a finer grid hierarchy as the new initial condition. This is repeated until the solution is obtained from the grid hierarchy $32 \times 8 - 2048 \times 512$ or $32 \times 8 - 1024 \times 256$. Parameters used in the model to generate the subsequent results are presented in Table 5.18. It is worth noting that with $\alpha = 90^\circ$, N in the pressure Equation (5.22) becomes zero. Therefore, the term $2\sqrt[3]{6}N(h+s)$ disappears.

Parameters	Values	Parameters	Values
s_0	1	γ	0.001
w_t	domain width	l_t	1, 5 and 10
α	90°		

Table 5.18: Values of parameters used in the simulations of fully-developed flows.

A series of 6 figures are presented here in a sequence: peak 1, 2, 3, trench 1, 2 and 3. Copies of the corresponding figures from [125] (fig. 10 - 11) are shown as the top figure (also denoted as figure (a)) in each case. Our solution from each case is then presented as the bottom figure (also denoted as figure (b)). The validations are illustrated in Figures 5.17 to 5.22. The comparisons from peak 1, 3, trench 1 and 3 provide strong validation. Our solutions from peak 2 and trench 2 are slightly different from those in [125]. However, they still provide a quantitatively good validation. We believe these differences are caused by using two slightly different models.

In the following section, results of the convergence tests from these six steady-state test cases are described.

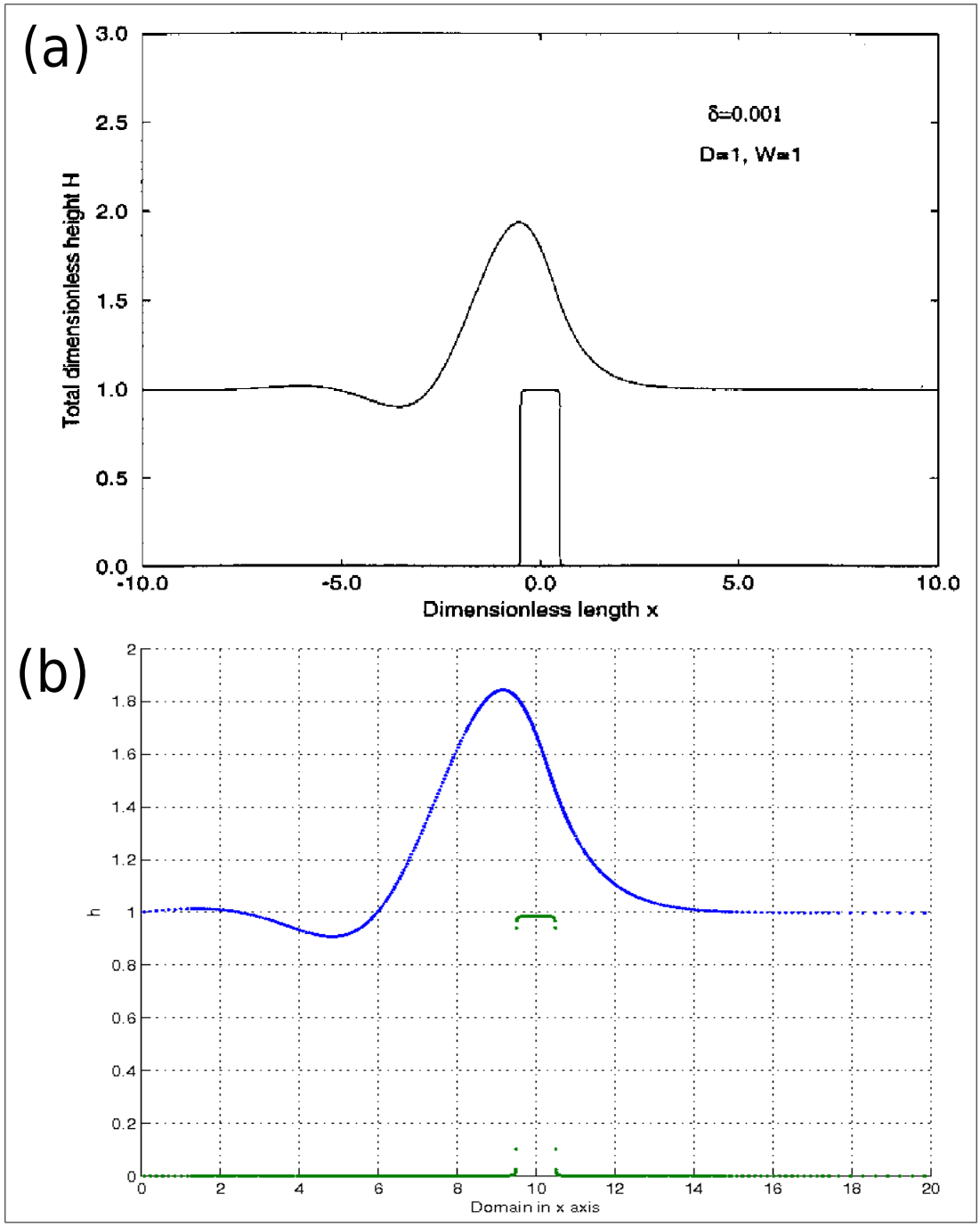

Figure 5.17: Figure (a) shows the peak 1 testing case from [125]. Figure (b) shows our corresponding results.

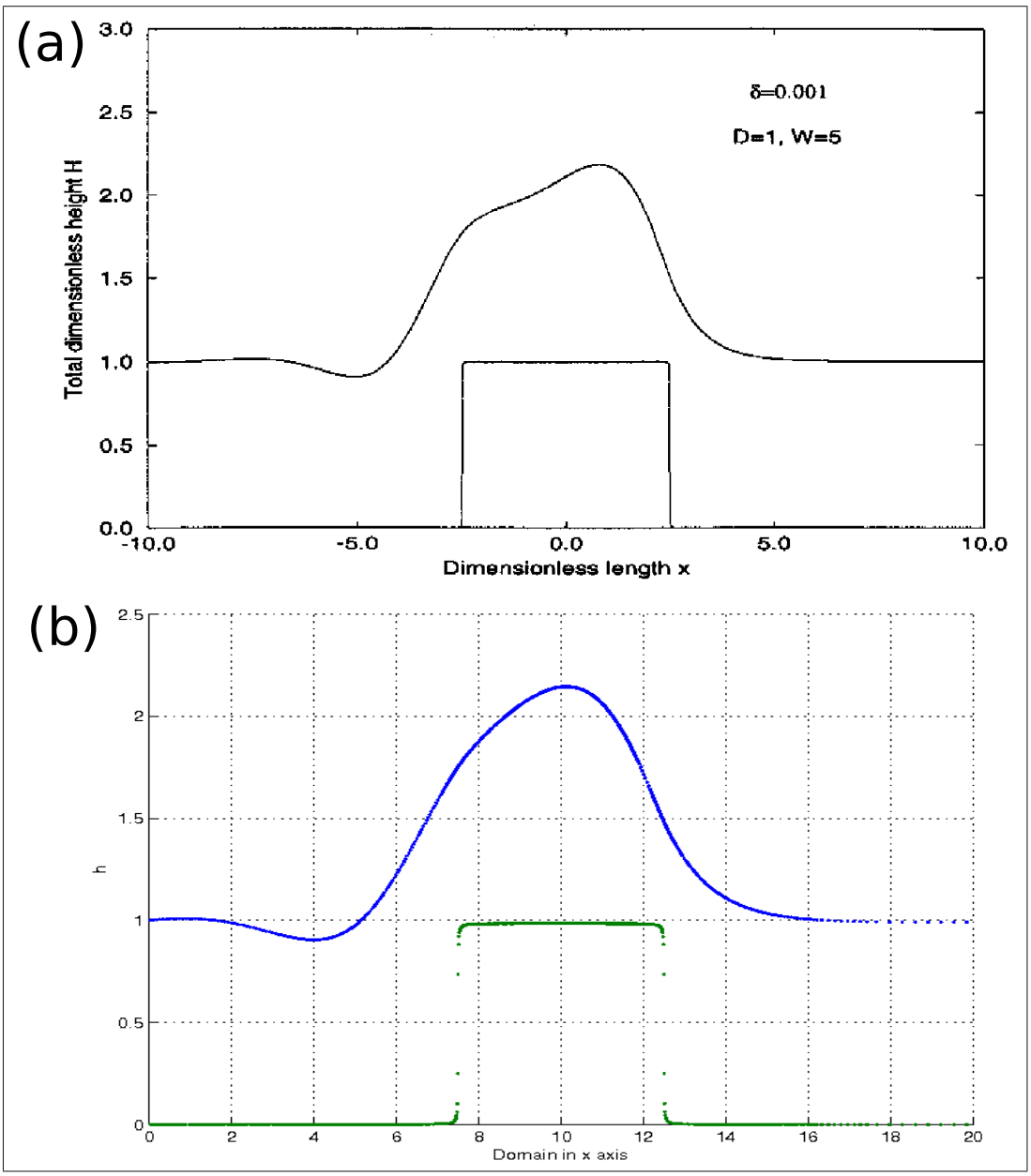

Figure 5.18: Figure (a) shows the peak 2 testing case from [125]. Figure (b) shows our corresponding results.

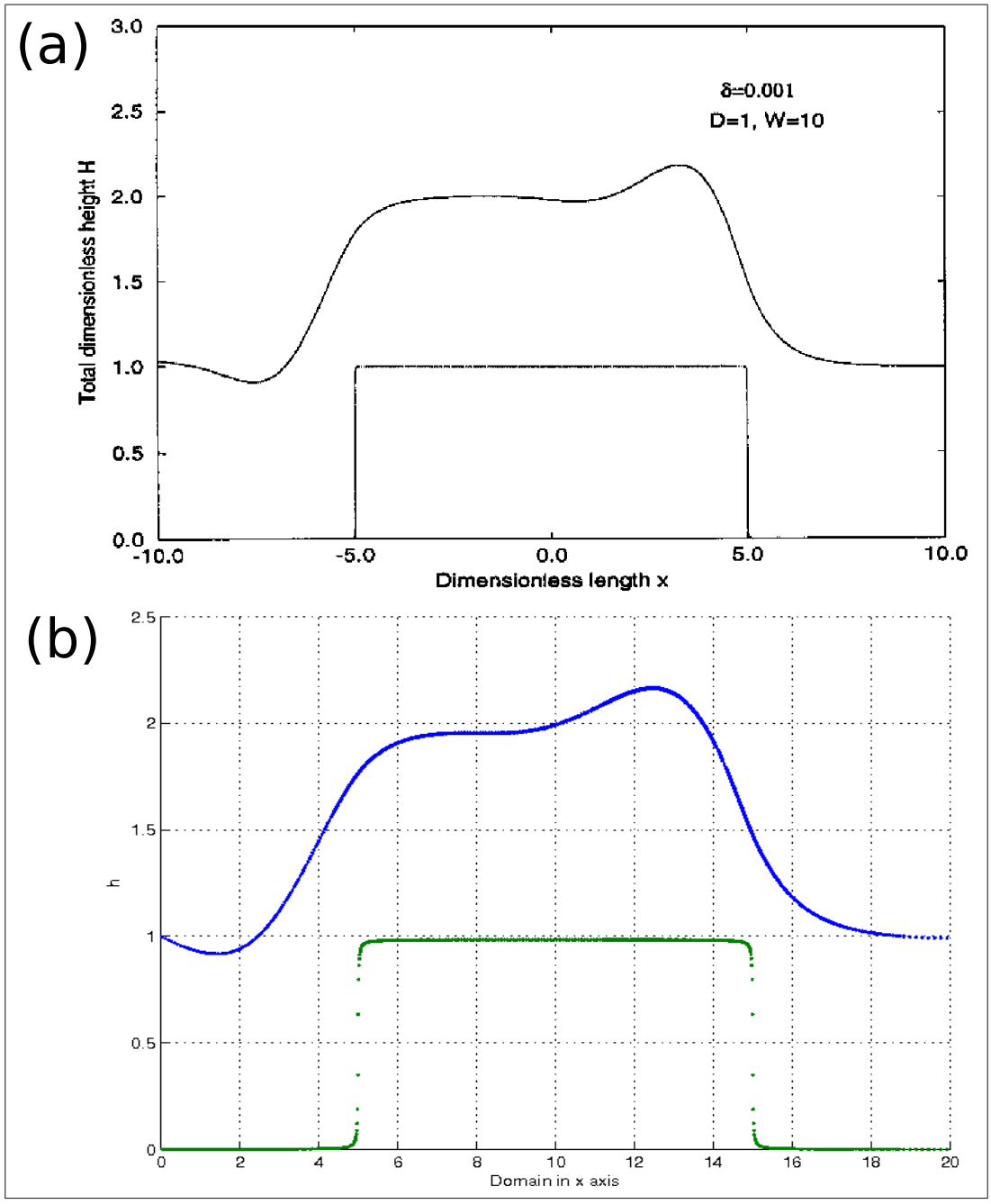

Figure 5.19: Figure (a) shows the peak 3 testing case from [125]. Figure (b) shows our corresponding results.

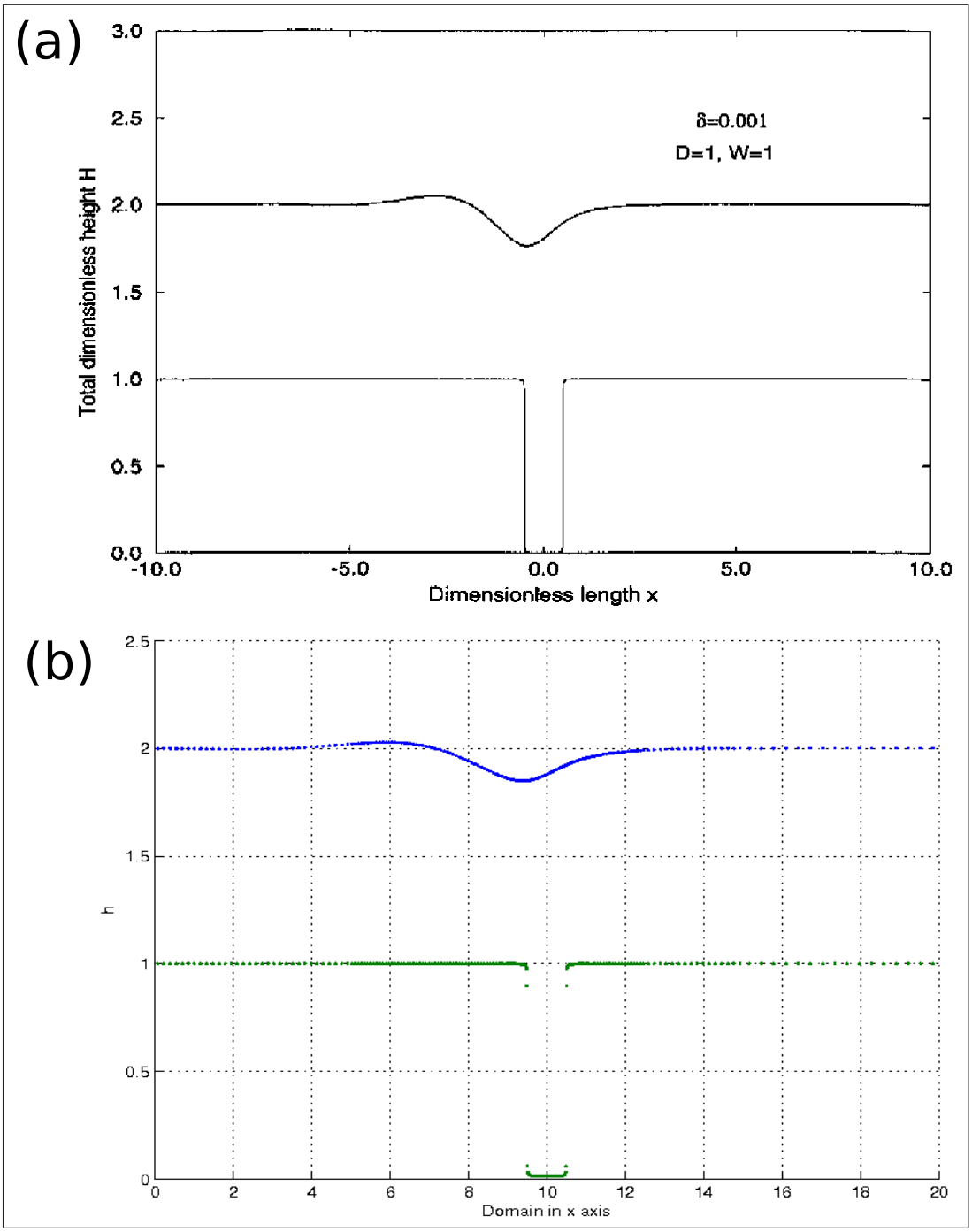

Figure 5.20: Figure (a) shows the trench 1 testing case from [125]. Figure (b) shows our corresponding results.

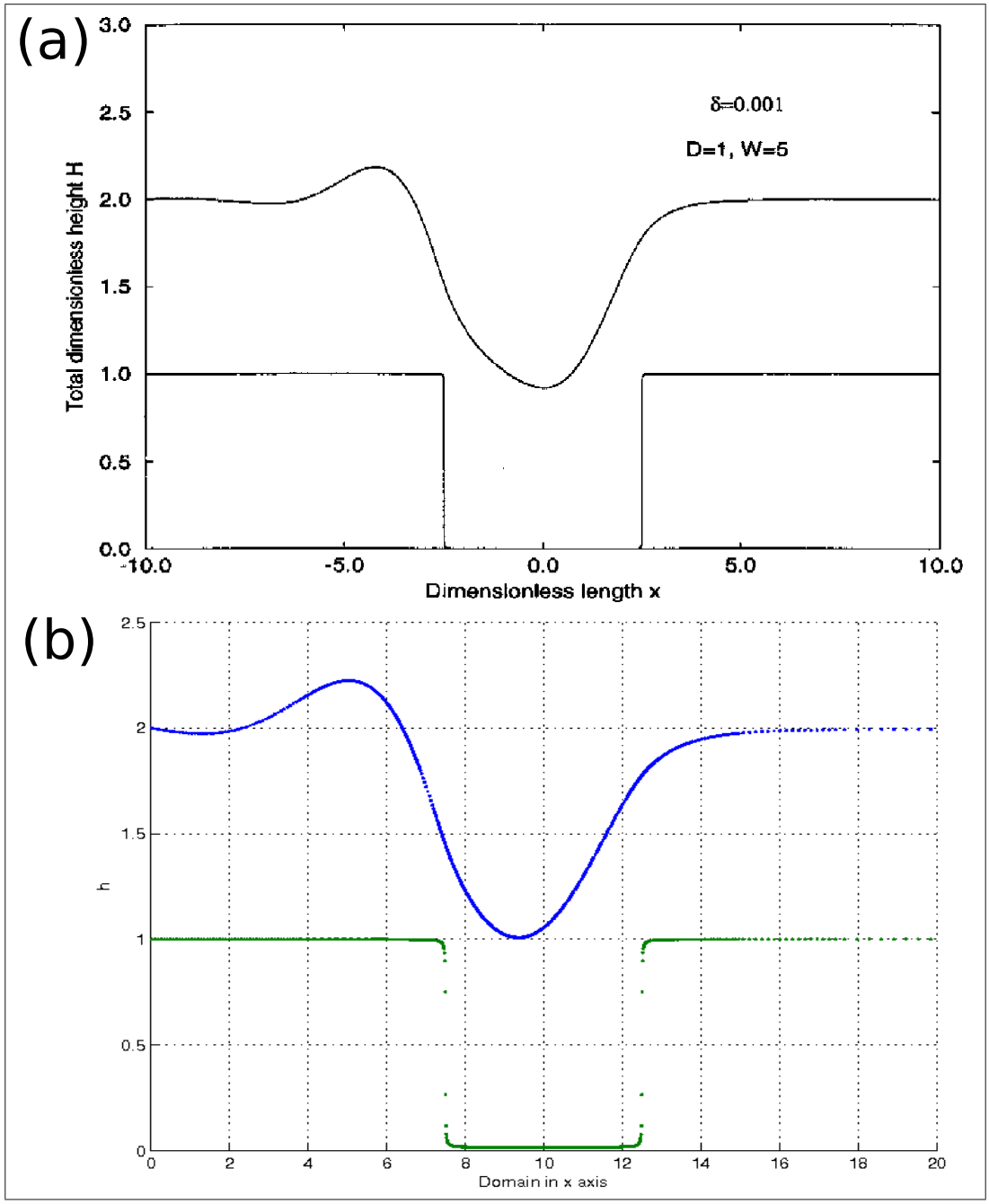

Figure 5.21: Figure (a) shows the trench 2 testing case from [125]. Figure (b) shows our corresponding results.

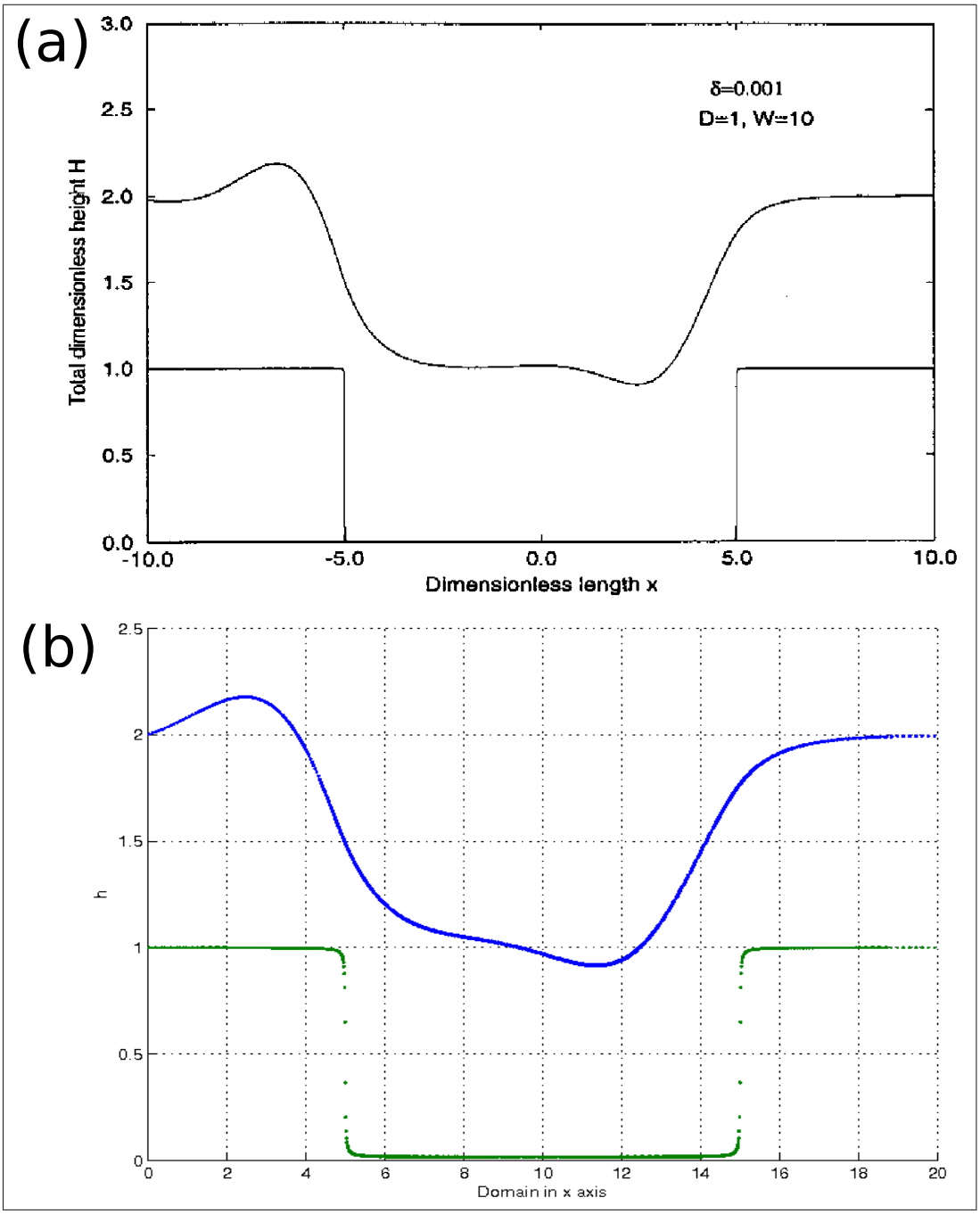

Figure 5.22: Figure (a) shows the trench 3 testing case from [125]. Figure (b) shows our corresponding results.

5.7.2 Convergence Tests

In order to present the convergence rate, the simulations from our multigrid solver that are presented in Figure 5.20 are selected. Four different grid hierarchies are employed here and all of them share the same coarsest grid 32×8. The coarsest grid hierarchy consists of 4 grids and the finest grid hierarchy consists of 7 grids. These convergence tests are based upon solution restriction (to compute the difference between solutions at consecutive levels) as previously described in Section 5.3.3. We use the C2F approach described in the previous section, which interpolates the converged steady state solution to a finer grid as its initial condition. The results are presented in Table 5.19 and indicate an overall second order convergence rate.

For variable h					
Levels	Starting δt	Infinity norm	Ratio	Two norm	Ratio
4	1×10^{-2}	-	-	-	-
5	5×10^{-3}	0.128×10^{0}	-	2.681×10^{-2}	-
6	2.5×10^{-3}	2.529×10^{-2}	5.07	4.015×10^{-3}	6.68
7	1.25×10^{-3}	6.351×10^{-3}	3.98	9.901×10^{-4}	4.05
For variable p					
4	1×10^{-2}	-	-	-	-
5	5×10^{-3}	0.140×10^{0}	-	7.290×10^{-2}	-
6	2.5×10^{-3}	1.663×10^{-2}	8.40	1.023×10^{-2}	7.61
7	1.25×10^{-3}	4.052×10^{-3}	4.10	2.609×10^{-3}	3.92

Table 5.19: Results show the differences in consecutive solutions measured in the stated norm, followed by the ratio of consecutive differences from the converged steady state solutions of the model of fully-developed flows with the case of trench 1.

In the following section, we conclude this chapter with a general discussion.

5.7.3 Discussion

To sum up the model of fully-developed flows, the mathematical model is presented in Section 5.5, based upon [82, 125]. Here it is solved as a time-dependent system using the described multigrid solver, which was comprehensively tested and validated for the droplet spreading model in the first half of the chapter. The discretization schemes and the resulting fully discrete system are described in Section 5.6. The results obtained from using our multigrid solver are presented in Section 5.7. Validation is considered first in Section 5.7.1, against results provided by Kalliadasis et al. in [125]. Overall we believe our results provide good validation. Then convergence tests are carried out. Results in

Section 5.7.2 suggest an overall second order convergence rate is obtained. This time-dependent system also provides us with the opportunity to study the variable incoming flows. However, this is on-going research and will be discussed later in this thesis. In the following chapter, a phase-field model of the Cahn-Hilliard-Hele-Shaw system of equations is described.

Chapter 6

Fully and Semi-Implicit Time Stepping with Parallel, FAS Multigrid on Cahn-Hilliard-Hele-Shaw System of Equations

---

In this chapter, a two-phase-field model that is presented in Section 1.3.2.3 is solved by using multigrid solvers. This model is known as the Cahn-Hilliard-Hele-Shaw (CHHS) system of equations, and as stated by Wise et al. in [226], it is used as a stepping stone toward the multi-phase-field model of tumour growth discussed in the following chapter. By following this route, the CHHS system of equations provides a good opportunity to test our implementations in Campfire for this Cahn-Hilliard type of system. By using the fully-implicit BDF2 method, we also demonstrate an overall second order convergence rate for solving the CHHS system, whereas only a first order convergence rate is obtained in [225].

In Section 6.1, the CHHS system of equations is presented. Two different temporal discretization schemes are employed in this chapter and are described in Section 6.2. The first one is the semi-implicit scheme derived from using a convex splitting approach of Wise [225]. The second one is the fully-implicit BDF2 method that is previously described in Section 2.1.2.2. In Section 6.3, the implementations are described. This in-

cludes the choice of the spatial discretization scheme. Two discrete systems are obtained in Section 6.3.1 from using the same spatial discretization scheme but the two different temporal discretization schemes. Based upon these two temporal schemes, two multigrid solvers are implemented in Campfire and their implementations are explained in Section 6.3.2. The first one is our multigrid solve with the use of the BDF2 method. The second one aims to replicate the solver used in [225], where the semi-implicit scheme is employed. Applying these two multigrid solvers, results for this CHHS system of equations are presented in Section 6.4. Firstly, we validate our implementation against the results from [225]. Then convergence tests and multigrid performances are discussed. It is worth noting the CHHS system is only solved in a 2-D situation. This is sufficient to demonstrate the CHHS type of system is within the capacity of our implementation. This chapter is concluded by a discussion of the comparison of red-black Gauss-Seidel and local Gauss-Seidel, global Jacobi iterations made in Section 6.4.3. We refer back to Section 1.3.2.3 for the glossary of this model.

6.1 Model Outline

The CHHS-type systems of equations are previously described in publications such as [144, 145]. They are commonly used to model binary fluid flows. The CHHS system of equations solved here is a simplification from them, and is presented previously in Section 1.3.2.3. Wise states in [225], "this CHHS phase-field model takes into account the chemical diffusivity from multiple components, and may be used to study a fluid mixture". As mentioned before, the multi-phase-field model of tumour growth presented in [226], is based upon this CHHS system. Wise et al. stated in [226] that this CHHS system of equations had been used as a stepping stone toward the model of tumour growth. It is our intention to follow Wise's route and use this system as a testing case for the implementation of our multigrid solver in Campfire for these types of system (see also Chapter 7).

In order to solve this system of equations, Wise in [225] derived a temporal discretization scheme that is called semi-implicit convex splitting, and proved its unconditional energy stability and unconditional unique solvability. On the other hand, this particular semi-implicit scheme used by Wise in [225] is only first order accurate. Furthermore, we intend to show the second order accurate BDF2 method is also suitable for solving this model. Combining with the second order five-point stencil, we present the overall second order convergence rate from our multigrid solver. Only an overall first order convergence rate is achieved by Wise in [225].

For clarity, the Cahn-Hilliard-Hele-Shaw system of equations is presented here again as the following:

$$\frac{\partial \phi}{\partial t} = \triangle \mu - \nabla \cdot (\phi \boldsymbol{u}), \tag{6.1}$$

$$\mu = \phi^3 - \phi - \varepsilon^2 \triangle \phi, \tag{6.2}$$

$$\boldsymbol{u} = -\nabla p - \gamma \phi \nabla \mu, \tag{6.3}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{6.4}$$

where $\phi(x,y,t) = \pm 1$ represents pure fluids, $\mu(x,y,t)$ denotes chemical potential, $\boldsymbol{u}(x,y,t)$ is the advective velocity and $p(x,y,t)$ is a pressure. Finally, ε is the phase-field interface thickness, and $\gamma > 0$ (note if $\gamma = 0$, this system reduces to the Cahn-Hilliard equation [42]).

In practice, instead of solving for the velocity $\boldsymbol{u}(x,y,t)$ by calculating pressure $p(x,y,t)$, we substitute Equation (6.3) into Equations (6.4) and (6.1), and solve for the pressure $p(x,y,t)$. Then this results in the three coupled equations:

$$\frac{\partial \phi}{\partial t} = \nabla \cdot (M(\phi) \nabla \mu) - \nabla \cdot (\phi \nabla p), \tag{6.5}$$

$$\mu = \phi^3 - \phi - \varepsilon^2 \triangle \phi, \tag{6.6}$$

$$-\triangle p = \gamma \nabla \cdot (\phi \nabla \mu), \tag{6.7}$$

where $M(\phi) = 1 + \gamma \phi^2$ arises when the velocity is dispensed with. The computational domain Ω is rectangular and has Cartesian coordinates $(x,y) \in \Omega = (0,3.2) \times (0,3.2)$ (also noted as $L_x = L_y = 3.2$ in [225]). Wise imposes a zero Neumann boundary condition on all variables around the domain Ω, that is

$$\frac{\partial \phi}{\partial \nu} = \frac{\partial \mu}{\partial \nu} = \frac{\partial p}{\partial \nu} = 0 \quad \text{on} \quad \partial \Omega, \tag{6.8}$$

where ν denotes the outward-pointing normal to the boundary $\partial \Omega$.

For completeness, initial conditions are needed for all three dependent variables. After defining $\phi(x,y,t=0)$, the second variable $\mu(x,y,t=0)$ is straightforward, since $\mu(x,y,t)$ is an assignment of $\phi(x,y,t)$ based upon Equation (6.6). However, since pressure $p(x,y,t)$ only appears in gradient form, a solver is needed to obtain an unique initial condition for the pressure, i.e. $p(x,y,t=0)$. We explain this later in this chapter.

It is worth noting the CHHS system of equations (Equations (6.5) to (6.7)) combined with the imposed boundary condition (Equation (6.8)) has no unique solutions. This is due to the pressure variable p appearing only in gradient form in this system. We employ

an approach which is described in [225], and this is described later in Section 6.3.2.1. In the following section, two temporal discretization schemes that are used separately to solve this CHHS system of equations are described.

6.2 Temporal Discretization Schemes

Two temporal discretization schemes are presented here for the CHHS system. They are used separately to solve this system. The first one is from [225], where Wise described a time-discrete scheme that is derived from a convex splitting approach, termed as "semi-implicit, semi-discrete scheme". However, for clarity, it is denoted simply as the semi-implicit scheme here. The second scheme is the fully-implicit BDF2 method, which is described in Section 2.1.2.2 and applied to the models of thin film flows in Chapter 5. In the following sections, both the semi-implicit scheme and the fully-implicit BDF2 method are described.

6.2.1 Semi-Implicit Convex Splitting Scheme

The semi-implicit scheme used in [225] is proven by Wise to have unconditional energy stability and unconditional unique solvability. This temporal discretization scheme is derived from using a convex splitting approach. Similar to the family of BDF methods (see Section 2.1.2.2), there are many schemes within the convex splitting family and these do not necessarily have to be only semi-implicit [71]. For instance, there is a fully-implicit two-step scheme from Hu et al. [117]. Other applications using convex splitting schemes can be found in [143, 221, 224].

The semi-implicit scheme that is used here is from [225] and is applied to Equations (6.5) to (6.7). The resulting system is given as the following:

$$\phi^{\tau+1} - \phi^{\tau} = \delta t \nabla \cdot \left(M\left(\phi^{\tau}\right) \nabla \mu^{\tau+1} \right) - \delta t \nabla \cdot \left(\phi^{\tau} \nabla p^{\tau+1} \right), \tag{6.9}$$

$$\mu^{\tau+1} = \left(\phi^{\tau+1}\right)^3 - \phi^{\tau} - \varepsilon^2 \triangle \phi^{\tau+1}, \tag{6.10}$$

$$-\triangle p^{\tau+1} = \gamma \nabla \cdot \left(\phi^{\tau} \nabla \mu^{\tau+1} \right), \tag{6.11}$$

where a superscript $\tau+1$ denotes solution from the current time-step, τ denotes solution from the previous time-step and function $M(\phi) = 1 + \gamma\phi^2$.

6.2.2 Fully-Implicit BDF2 Method

The fully-implicit BDF2 method is explained in Section 2.1.2.2, and applied to the models of thin film flows in Chapter 5. Here the BDF2 method is applied to the CHHS system of equations with a fixed δt, it is given as the following:

$$\phi^{\tau+1} - \left(\frac{4}{3}\phi^{\tau} - \frac{1}{3}\phi^{\tau-1}\right) = \frac{2\delta t}{3}\nabla\cdot\left(M(\phi^{\tau+1})\nabla\mu^{\tau+1}\right) - \frac{2\delta t}{3}\nabla\cdot\left(\phi^{\tau+1}\nabla p^{\tau+1}\right), \quad (6.12)$$

$$\mu^{\tau+1} = \left(\phi^{\tau+1}\right)^3 - \phi^{\tau+1} - \varepsilon^2\triangle\phi^{\tau+1}, \quad (6.13)$$

$$-\triangle p^{\tau+1} = \gamma\nabla\cdot\left(\phi^{\tau+1}\nabla\mu^{\tau+1}\right), \quad (6.14)$$

where function $M(\phi) = 1+\gamma\phi^2$, a superscript $\tau+1$ denotes solution from the current time-step, τ denotes solution from the previous time-step and $\tau-1$ indicates solution from the one before the previous time-step. Although the BDF2 method is presented with a fixed δt here, if needed, it is straightforward to use the adaptive approach shown in Equation (2.14). It is worth noting that for the very first time-step, BDF1 method (also named as backward Euler method) has to be used. In the following section, the spatial discretization scheme is described and the fully-discrete systems are obtained.

6.3 Implementation

In this section, implementations of the solvers for CHHS system of equations are described. Two systems of nonlinear algebraic equations are generated from using two different temporal discretization schemes (presented in the previous section). The central FDM with the five-point stencil which is shown in Equation (2.3) is applied in each case. Having obtained the discrete systems in Section 6.3.1, implementations of two multigrid solvers are discussed in Section 6.3.2. The first solver is our multigrid solver for the BDF2 system, and is described in Section 6.3.2.1. The second one aims at replicating the multigrid solver used in [225], and it is called the Wise solver. The semi-implicit temporal discretization scheme is employed as part of this solver and is discussed in Section 6.3.2.2.

6.3.1 Spatial Discretization Scheme

The central FDM with the five-point stencil which is shown in Equation (2.3) is used as the spatial discretization scheme here. Combining with the two temporal discretization schemes presented earlier, two discrete systems may be obtained.

Firstly the five-point stencil is combined with the BDF2 method, and the resulting discrete system consists of the following three equations. Note for the CHHS system equations, the notation for the equal distance between the adjacent grid points is h.

$$\begin{aligned}
&\frac{\phi_{i,j}^{\tau+1}-\left(\frac{4}{3}\phi_{i,j}^{\tau}-\frac{1}{3}\phi_{i,j}^{\tau-1}\right)}{\delta t}= \\
&\frac{2}{3h}\left\{\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau+1}+\phi_{i+1,j}^{\tau+1}}{2}\right)^2\right]\frac{\mu_{i+1,j}^{\tau+1}-\mu_{i,j}^{\tau+1}}{h}\right. \\
&\quad-\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau+1}+\phi_{i-1,j}^{\tau+1}}{2}\right)^2\right]\frac{\mu_{i,j}^{\tau+1}-\mu_{i-1,j}^{\tau+1}}{h} \\
&\quad+\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau+1}+\phi_{i,j+1}^{\tau+1}}{2}\right)^2\right]\frac{\mu_{i,j+1}^{\tau+1}-\mu_{i,j}^{\tau+1}}{h} \\
&\quad\left.-\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau+1}+\phi_{i,j-1}^{\tau+1}}{2}\right)^2\right]\frac{\mu_{i,j}^{\tau+1}-\mu_{i,j-1}^{\tau+1}}{h}\right\} \\
&-\frac{2}{3h}\left\{\frac{\phi_{i,j}^{\tau+1}+\phi_{i+1,j}^{\tau+1}}{2}\frac{p_{i+1,j}^{\tau+1}-p_{i,j}^{\tau+1}}{h}-\frac{\phi_{i,j}^{\tau+1}+\phi_{i-1,j}^{\tau+1}}{2}\frac{p_{i,j}^{\tau+1}-p_{i-1,j}^{\tau+1}}{h}\right. \\
&\quad\left.+\frac{\phi_{i,j}^{\tau+1}+\phi_{i,j+1}^{\tau+1}}{2}\frac{p_{i,j+1}^{\tau+1}-p_{i,j}^{\tau+1}}{h}-\frac{\phi_{i,j}^{\tau+1}+\phi_{i,j-1}^{\tau+1}}{2}\frac{p_{i,j}^{\tau+1}-p_{i,j-1}^{\tau+1}}{h}\right\},
\end{aligned} \tag{6.15}$$

$$\mu_{i,j}^{\tau+1}=\left(\phi_{i,j}^{\tau+1}\right)^3-\phi_{i,j}^{\tau+1}-\varepsilon^2\left(\frac{\phi_{i+1,j}^{\tau+1}+\phi_{i-1,j}^{\tau+1}+\phi_{i,j+1}^{\tau+1}+\phi_{i,j-1}^{\tau+1}-4\phi_{i,j}^{\tau+1}}{h^2}\right), \tag{6.16}$$

$$\begin{aligned}
&-\left(\frac{p_{i+1,j}^{\tau+1}+p_{i-1,j}^{\tau+1}+p_{i,j+1}^{\tau+1}+p_{i,j-1}^{\tau+1}-4p_{i,j}^{\tau+1}}{h^2}\right)= \\
&\frac{\gamma}{h}\left\{\frac{\phi_{i,j}^{\tau+1}+\phi_{i+1,j}^{\tau+1}}{2}\frac{\mu_{i+1,j}^{\tau+1}-\mu_{i,j}^{\tau+1}}{h}-\frac{\phi_{i,j}^{\tau+1}+\phi_{i-1,j}^{\tau+1}}{2}\frac{\mu_{i,j}^{\tau+1}-\mu_{i-1,j}^{\tau+1}}{h}\right. \\
&\quad\left.+\frac{\phi_{i,j}^{\tau+1}+\phi_{i,j+1}^{\tau+1}}{2}\frac{\mu_{i,j+1}^{\tau+1}-\mu_{i,j}^{\tau+1}}{h}-\frac{\phi_{i,j}^{\tau+1}+\phi_{i,j-1}^{\tau+1}}{2}\frac{\mu_{i,j}^{\tau+1}-\mu_{i,j-1}^{\tau+1}}{h}\right\}.
\end{aligned} \tag{6.17}$$

Secondly the five-point stencil is combined with the semi-implicit scheme, the result-

ing discrete system which consists of three equations as follows.

$$\begin{aligned}
\frac{\phi_{i,j}^{\tau+1}-\phi_{i,j}^{\tau}}{\delta t} = & \\
& \frac{1}{h}\Bigg\{\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i+1,j}^{\tau}}{2}\right)^2\right]\frac{\mu_{i+1,j}^{\tau+1}-\mu_{i,j}^{\tau+1}}{h} \\
& -\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i-1,j}^{\tau}}{2}\right)^2\right]\frac{\mu_{i,j}^{\tau+1}-\mu_{i-1,j}^{\tau+1}}{h} \\
& +\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i,j+1}^{\tau}}{2}\right)^2\right]\frac{\mu_{i,j+1}^{\tau+1}-\mu_{i,j}^{\tau+1}}{h} \\
& -\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i,j-1}^{\tau}}{2}\right)^2\right]\frac{\mu_{i,j}^{\tau+1}-\mu_{i,j-1}^{\tau+1}}{h}\Bigg\} \\
& -\frac{1}{h}\Bigg\{\left(\frac{\phi_{i,j}^{\tau}+\phi_{i+1,j}^{\tau}}{2}\right)\left(\frac{p_{i+1,j}^{\tau+1}-p_{i,j}^{\tau+1}}{h}\right)-\left(\frac{\phi_{i,j}^{\tau}+\phi_{i-1,j}^{\tau}}{2}\right)\left(\frac{p_{i,j}^{\tau+1}-p_{i-1,j}^{\tau+1}}{h}\right) \\
& +\left(\frac{\phi_{i,j}^{\tau}+\phi_{i,j+1}^{\tau}}{2}\right)\left(\frac{p_{i,j+1}^{\tau+1}-p_{i,j}^{\tau+1}}{h}\right)-\left(\frac{\phi_{i,j}^{\tau}+\phi_{i,j-1}^{\tau}}{2}\right)\left(\frac{p_{i,j}^{\tau+1}-p_{i,j-1}^{\tau+1}}{h}\right)\Bigg\},
\end{aligned} \tag{6.18}$$

$$\mu_{i,j}^{\tau+1} = \left(\phi_{i,j}^{\tau+1}\right)^3 - \phi_{i,j}^{\tau} - \varepsilon^2\left(\frac{\phi_{i+1,j}^{\tau+1}+\phi_{i-1,j}^{\tau+1}+\phi_{i,j+1}^{\tau+1}+\phi_{i,j-1}^{\tau+1}-4\phi_{i,j}^{\tau+1}}{h^2}\right), \tag{6.19}$$

$$\begin{aligned}
& -\left(\frac{p_{i+1,j}^{\tau+1}+p_{i-1,j}^{\tau+1}+p_{i,j+1}^{\tau+1}+p_{i,j-1}^{\tau+1}-4p_{i,j}^{\tau+1}}{h^2}\right) = \\
& \frac{\gamma}{h}\Bigg(\frac{\phi_{i,j}^{\tau}+\phi_{i+1,j}^{\tau}}{2}\frac{\mu_{i+1,j}^{\tau+1}-\mu_{i,j}^{\tau+1}}{h}-\frac{\phi_{i,j}^{\tau}+\phi_{i-1,j}^{\tau}}{2}\frac{\mu_{i,j}^{\tau+1}-\mu_{i-1,j}^{\tau+1}}{h} \\
& +\frac{\phi_{i,j}^{\tau}+\phi_{i,j+1}^{\tau}}{2}\frac{\mu_{i,j+1}^{\tau+1}-\mu_{i,j}^{\tau+1}}{h}-\frac{\phi_{i,j}^{\tau}+\phi_{i,j-1}^{\tau}}{2}\frac{\mu_{i,j}^{\tau+1}-\mu_{i,j-1}^{\tau+1}}{h}\Bigg).
\end{aligned} \tag{6.20}$$

Having presented both discrete systems, in the following section, implementations of two multigrid solvers in Campfire are described. Implementations of the initial condition and an additional process which ensures the CHHS system has a unique solution are explained.

6.3.2 Implementations of Multigrid Solvers

Two multigrid solvers are described for solving the systems described in the previous section. The first one is combined with the fully-implicit BDF2 method and is described in Section 6.3.2.1. The second one uses the semi-implicit scheme proposed by Wise in [225], and also combined with Wise's version of an iterative solver. This solver is discussed in Section 6.3.2.2.

6.3.2.1 Implementation of Our Multigrid Solver

The implementation of our multigrid solver in Campfire is previously described in detail in Section 3.3. In Chapter 5, this multigrid solver is employed with the described modifications (see Section 5.3) to solve the models of thin film flows. The multigrid solver that is applied to this CHHS system of equations shares many common features with the one used in Chapter 5.

Firstly, there are two equations in the CHHS system which are non-time-dependent. They are Equations (6.6) and (6.7), the former is the equation for the variable of chemical potential μ, and the latter is the pressure equation for variable p. As shown in Equation (6.6), the chemical potential μ is similar to the pressure term p in the models of thin film flows that are presented in the previous chapter. The implementation used here for μ is previously described in Section 5.3.1. On the other hand, the pressure Equation (6.7) is not a simple assignment but actually a PDE to be discretised and solved, and this is described later in this section.

In order to demonstrate that the use of the five-point stencil and the BDF2 method provides an overall second order convergence rate, the convergence tests presented in this chapter inherit the implementation that is previously described in Section 5.3.3, which is based upon solution restriction. Both the infinity norm and the two norm are employed for all variables here, and details are presented later in this chapter.

Having described some common implementations shared with the previous multigrid solver, which is used for the models of thin film flows, there are a number of specific implementation details which are needed for solving the CHHS system.

Firstly, a block version of a nonlinear iterative method is employed in our multigrid solver as the smoother and the coarsest grid solver for this CHHS system. An example of such a nonlinear block-Jacobi method is presented in Equation (2.36). However, by following the suggestion made in [225], a red-black Gauss-Seidel iteration is employed in our implementation in Campfire for this CHHS system [1]. The red-black Gauss-Seidel

[1] For completeness, the red-black Gauss-Seidel iteration has already been tried on the models of thin film

iteration is previously mentioned in Section 3.3.4. The comparison between the red-black Gauss-Seidel and the local Gauss-Seidel, global Jacobi is further discussed in Section 6.4.3. Furthermore, by using the block version of the nonlinear iterative method (see Equation (2.36)), the local Jacobian matrix C of this iteration becomes a 3×3 matrix on each grid point. The inverse of this matrix is calculated block-by-block by Gaussian elimination and an upper triangular solver with the use of backward substitution.

Different from the pressure Equations (5.2) and (5.22) in the models of thin film flows, the pressure Equation (6.7) in the CHHS system is not a simple assignment of the time-dependent variable. Despite this difference, the implementation of non-time-dependent PDEs that is previously discussed in Section 5.3.1 may still be used to handle the pressure equation in this CHHS system. More specifically, the pressure equation (Equation (6.17)), when being implemented in Campfire, should be the form of the discretization of

$$-\triangle p - \gamma \nabla \cdot (\phi \nabla \mu) = 0. \tag{6.21}$$

The rest of implementation is straightforward as described in Section 5.3.1.

Thirdly, an averaging process is required for the pressure variable p. This is because the pressure term only appears in its gradient form in the whole CHHS system. Together with the use of zero Neumann boundary condition, the pressure field may shift up and down during our computation and/or by the grid transfer operators in the multigrid method. To prevent this, we introduce an averaging process for the pressure variable p which ensures the discrete integral of p equals to zero (i.e. $\int p = 0$), and it is given as the following:

$$p_{i,j}^{\text{averaged}} = p_{i,j} - \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} p_{i,j}}{n \times n}. \tag{6.22}$$

This averaging process is denoted $(p||\mathbf{1}) = 0$ in [225]. In our implementation in Campfire, this process is carried out after every iteration of the smoother and the coarsest grid solver, as well as after every grid transfer operation (i.e. restriction and interpolation).

Then it is worth noting that for obtaining the initial condition of pressure variable p, simple assignments are not enough, and a solver is required. In order to demonstrate this solver, let's assume the initial conditions for $\phi(x,y,t=0)$ and $\mu(x,y,t=0)$ are obtained (μ is a function of ϕ, thus obtaining its initial condition is straightforward once $\phi(x,y,,t=0)$ is specified). Here an iterative solver is applied, and values of $p_{i,j}^{l+1}$ are updated as in Equation (6.23). Note for clarity, this iterative solver is presented in Jacobi form, however,

flows presented in the previous chapter, and there are noticeable improvements in the convergence rates. However, these improvements are relatively too small to have an impact on the number of V-cycles required in each time step.

in the actual implementation, we use the local Gauss-Seidel, global Jacobi iteration (see Section 3.3).

$$
\begin{aligned}
p_{i,j}^{l+1,t=0} = \frac{\gamma h^2}{4} \Bigg\{ & \left(\frac{\phi_{i+1,j}^{l,t=0} + \phi_{i,j}^{l,t=0}}{2} \right) \left(\frac{\mu_{i+1,j}^{l,t=0} - \mu_{i,j}^{l,t=0}}{h^2} \right) \\
& - \left(\frac{\phi_{i-1,j}^{l,t=0} + \phi_{i,j}^{l,t=0}}{2} \right) \left(\frac{\mu_{i,j}^{l,t=0} - \mu_{i-1,j}^{l,t=0}}{h^2} \right) \\
& + \left(\frac{\phi_{i,j+1}^{l,t=0} + \phi_{i,j}^{l,t=0}}{2} \right) \left(\frac{\mu_{i,j+1}^{l,t=0} - \mu_{i,j}^{l,t=0}}{h^2} \right) \\
& - \left(\frac{\phi_{i,j-1}^{l,t=0} + \phi_{i,j}^{l,t=0}}{2} \right) \left(\frac{\mu_{i,j}^{l,t=0} - \mu_{i,j-1}^{l,t=0}}{h^2} \right) \Bigg\} \\
& + \frac{1}{4} \left(p_{i+1,j}^{l,t=0} + p_{i-1,j}^{l,t=0} + p_{i,j+1}^{l,t=0} + p_{i,j-1}^{l,t=0} \right),
\end{aligned}
\tag{6.23}
$$

where superscript $l+1$ indicates the current iteration and l denotes the previous iteration. The stopping criterion for this solver is based upon the residual of the pressure equation, and terminates when the infinity norm of residual $r_{p(x,y,t=0)} \leq 1 \times 10^{-11}$. For completeness, it is worth noting that our multigrid solver in Campfire is capable to obtain this initial condition shown above. However, due to the capability of re-starting from checkpoint files, the initial condition is only required to be computed once. Thus this simple iterative solver is used here.

Finally, for the stopping criterion at each time step, we follow the suggestion made in [225], that a scaled 2-norm is used as a single stopping criterion for our multigrid solver. This scaled 2-norm takes values from residuals of all variables into account, and is given as the following:

$$
||\boldsymbol{R}(\boldsymbol{\phi})||_{2,\star} := \sqrt{\frac{h^2}{3L_xL_y} \sum_{k=1}^{3} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(R_{i,j}^{(k)}(\boldsymbol{\phi}) \right)^2},
\tag{6.24}
$$

where $\boldsymbol{R}(\boldsymbol{\phi})$ is the $3 \times n \times n$ residual array that counts all three variables, $R_{i,j}^{(k)}(\boldsymbol{\phi})$ are its components, n is the number of grid points in each axis direction, and $k = 1,2,3$ denotes each of the three variables in the CHHS system. Furthermore, (i,j) are the Cartesian coordinates for all internal grid points $(i,j = 1,\ldots,n)$, and L_x, L_y are the lengths of the domain in x and y directions, respectively.

In the following section, the implementation of the Wise solver is described.

6.3.2.2 Implementation of Wise's Multigrid Solver

Having described our multigrid solver, another multigrid solver which is intended to replicate the one used in [225] is also implemented in Campfire. This solver is denoted as the Wise solver, and shares many common features from the one that has just been described, however, there are a few minor differences. These differences are summarised here.

Firstly, the temporal discretization scheme implemented in Wise solver is the semi-implicit scheme as described in Section 6.2.1. Secondly, the smoother (and the coarsest grid solver) used in the Wise solver is slightly different from those described earlier. Although it also updates all variables together at one grid point, the cubic term in the equation of chemical potential of μ (i.e. Equation (6.6)) is linearised by a local Newton approximation. Equations (6.18), (6.19) and (6.20) are rearranged into the forms of Equations (6.25) - (6.27). Note for simplicity (and descriptive purposes only), the superscripts $l+1$ and l are used to indicate the current and previous lexicographic Gauss-Seidel sweep, respectively. However, it is the red-black Gauss-Seidel iteration that is used in the implementation. Moreover, these variables that are marked with superscripts $l+1$ and l

are already assumed to be from the current time step $\tau+1$.

$$
\begin{aligned}
&\phi_{i,j}^{l+1}+\frac{\delta t}{h^2}\left\{\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i+1,j}^{\tau}}{2}\right)^2\right]+\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i-1,j}^{\tau}}{2}\right)^2\right]\right.\\
&\qquad\left.\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i,j+1}^{\tau}}{2}\right)^2\right]+\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i,j-1}^{\tau}}{2}\right)^2\right]\right\}\mu_{i,j}^{l+1}\\
&\quad+\frac{\delta t}{h^2}\left\{\frac{\phi_{i,j}^{\tau}+\phi_{i+1,j}^{\tau}}{2}+\frac{\phi_{i,j}^{\tau}+\phi_{i-1,j}^{\tau}}{2}+\frac{\phi_{i,j}^{\tau}+\phi_{i,j+1}^{\tau}}{2}+\frac{\phi_{i,j}^{\tau}+\phi_{i,j-1}^{\tau}}{2}\right\}p_{i,j}^{l+1}\\
&\qquad=\\
&\qquad S_{i,j}^{(1)}\left(\boldsymbol{\phi}^{\boldsymbol{\tau}}\right)\\
&\quad+\frac{\delta t}{h^2}\left\{\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i+1,j}^{\tau}}{2}\right)^2\right]\mu_{i+1,j}^{l}+\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i-1,j}^{\tau}}{2}\right)^2\right]\mu_{i-1,j}^{l+1}\right.\\
&\qquad\left.+\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i,j+1}^{\tau}}{2}\right)^2\right]\mu_{i,j+1}^{l}+\left[1+\gamma\left(\frac{\phi_{i,j}^{\tau}+\phi_{i,j-1}^{\tau}}{2}\right)^2\right]\mu_{i,j-1}^{l+1}\right\}\\
&\quad+\frac{\delta t}{h^2}\left\{\frac{\phi_{i,j}^{\tau}+\phi_{i+1,j}^{\tau}}{2}p_{i+1,j}^{l}+\frac{\phi_{i,j}^{\tau}+\phi_{i-1,j}^{\tau}}{2}p_{i-1,j}^{l+1}\right.\\
&\qquad\left.+\frac{\phi_{i,j}^{\tau}+\phi_{i,j+1}^{\tau}}{2}p_{i,j+1}^{l}+\frac{\phi_{i,j}^{\tau}+\phi_{i,j-1}^{\tau}}{2}p_{i,j-1}^{l+1}\right\},
\end{aligned}
\tag{6.25}
$$

$$
\begin{aligned}
&\left[-3\left(\phi_{i,j}^{l}\right)^2-\frac{4\varepsilon^2}{h^2}\right]\phi_{i,j}^{l+1}+\mu_{i,j}^{l+1}=\\
&S_{i,j}^{(2)}\left(\boldsymbol{\phi}^{\boldsymbol{\tau}}\right)-2\left(\phi_{i,j}^{l}\right)^3-\frac{\varepsilon^2}{h^2}\left(\phi_{i+1,j}^{l}+\phi_{i-1,j}^{l+1}+\phi_{i,j+1}^{l}+\phi_{i,j-1}^{l+1}\right),
\end{aligned}
\tag{6.26}
$$

$$
\begin{aligned}
&-\frac{\gamma}{h^2}\left\{\frac{\phi^{\tau}_{i,j}+\phi^{\tau}_{i+1,j}}{2}+\frac{\phi^{\tau}_{i,j}+\phi^{\tau}_{i-1,j}}{2}\right.\\
&\quad\left.+\frac{\phi^{\tau}_{i,j}+\phi^{\tau}_{i,j+1}}{2}+\frac{\phi^{\tau}_{i,j}+\phi^{\tau}_{i,j-1}}{2}\right\}\mu^{l+1}_{i,j}-\frac{4}{h^2}p^{l+1}_{i,j}=\\
&\quad S^{(3)}_{i,j}\left(\boldsymbol{\phi^{\tau}}\right)-\frac{1}{h^2}\left(p^{l}_{i+1,j}+p^{l+1}_{i-1,j}+p^{l}_{i,j+1}+p^{l+1}_{i,j-1}\right)\\
&-\frac{\gamma}{h^2}\left\{\frac{\phi^{\tau}_{i,j}+\phi^{\tau}_{i+1,j}}{2}\mu^{l}_{i+1,j}+\frac{\phi^{\tau}_{i,j}+\phi^{\tau}_{i-1,j}}{2}\mu^{l+1}_{i-1,j}\right.\\
&\quad\left.+\frac{\phi^{\tau}_{i,j}+\phi^{\tau}_{i,j+1}}{2}\mu^{l}_{i,j+1}+\frac{\phi^{\tau}_{i,j}+\phi^{\tau}_{i,j-1}}{2}\mu^{l+1}_{i,j-1}\right\},
\end{aligned}
\tag{6.27}
$$

where $S^{(1)}_{i,j}=\phi^{\tau}_{i,j}$, $S^{(2)}_{i,j}=-\phi^{\tau}_{i,j}$ and $S^{(3)}_{i,j}=0$ on the finest grid, but on coarser grids, they are substituted by the corresponding modified RHS of the FAS multigrid method. Then Cramer's Rule is applied on this 3×3 linear system to obtain $\phi^{l+1}_{i,j}$, $\mu^{l+1}_{i,j}$ and $p^{l+1}_{i,j}$. This process is used as pre-, post-smoother and the coarsest grid solver in the Wise solver. It is worth noting that the averaging process for the pressure variable p shown in Equation (6.22) is carried out in the Wise solver, after every iteration of the smoother and the coarsest grid solver, as well as after every grid transfer operation. The initial condition for pressure p is also obtained through the use of the simple solver shown in Equation (6.23).

Having described two multigrid solvers, in the following section, results from using these two solvers to solve this CHHS system of equations are discussed.

6.4 Results on Cahn-Hilliard-Hele-Shaw System of Equations

Applying the two multigrid solvers described in previous sections, results on this CHHS system of equations are presented in this section. Firstly, the Wise solver is used to validate results from our implementation against the ones from [225] in Section 6.4.1. Then convergence tests and multigrid performance are discussed in Section 6.4.2. A final discussion is made in Section 6.4.3 to conclude this chapter. It is worth noting that to generate these results, the parallel computer cluster ARC2 is used, and generally up to 16 cores are employed for each task.

6.4.1 Validation

The CHHS system of equations is solved by Wise and published in [225]. In this section, we validate our results which are generated by using the described Wise solver implemented in Campfire by comparing with the results presented in [225].

The computational domain Ω has Cartesian coordinates $(x,y) \in \Omega = (0,3.2) \times (0,3.2)$. The initial condition for the phase-field variable $\phi(x,y,t=0)$ is given as

$$\phi_{i,j}^{t=0} = \frac{\left[1-\cos\left(\frac{4x_i\pi}{L_x}\right)\right] \times \left[1-\cos\left(\frac{2y_j\pi}{L_y}\right)\right]}{2} - 1, \tag{6.28}$$

where $i,j = 1,\ldots,n$, n is the number of internal grid points in each direction, and $L_x = L_y = 3.2$ are the lengths of domain in the x and the y directions, respectively.

Having obtained $\phi(x,y,t=0)$, the initial condition for the variable of chemical potential μ on all internal grid points (i,j) may be defined as

$$\mu_{i,j}^{t=0} = \left(\phi_{i,j}^{t=0}\right)^3 - \phi_{i,j}^{t=0} - \frac{\varepsilon^2}{h^2}\left(\phi_{i+1,j}^{t=0} + \phi_{i-1,j}^{t=0} + \phi_{i,j+1}^{t=0} + \phi_{i,j-1}^{t=0} - 4\phi_{i,j}^{t=0}\right). \tag{6.29}$$

The process to obtain the initial condition for the pressure variable p is already described, and $p(x,y,t=0)$ is obtained through the use of iteration (6.23).

Having obtained the correct initial conditions for all variables, we validate the Wise solver from our implementation against the results presented in Table 1 in [225]. There are five different grid hierarchies used, having finest grids with the resolutions of 32×32, 64×64, 128×128, 256×256 and 512×512. All of them have the 16×16 grid as the coarsest grid in their grid hierarchies. Therefore, they are denoted levels 2, 3, 4, 5 and 6 respectively. The results shown in this table include the number of V-cycles required at the 20^{th} time step from all different grid hierarchies to reduce the scaled 2-norm of the residuals below a fixed stopping criterion that is 1×10^{-8}. A fixed time step size $\delta t = 1 \times 10^{-3}$ is applied to all 20 time steps for all levels. The effects of two parameters in the CHHS system (i.e. ε and γ) on the convergence of the employed multigrid solver are considered. Additionally, the effects from different numbers of pre- and post-smoothes applied in the multigrid solver are also tested and the parameter which indicates the number of iterations of both smoothers is denoted as λ.

Our results are presented in Table 6.1, and in total there are five different combinations of these parameters. Thus it leads to five different test cases. For clarity, we present Wise's results (i.e. the number of V-cycles) shown from Table 1 in [225] with brackets in Table 6.1 here.

	Test 1	Test 2	Test 3	Test 4	Test 5
ε	$2\times10-1$	$2\times10-1$	$1\times10-1$	$5\times10-2$	$5\times10-2$
γ	2.0	2.0	2.0	4.0	8.0
λ	1	2	2	2	2
level 2	8 (7)	6 (5)	6 (4)	6 (4)	6 (5)
level 3	9 (8)	6 (5)	7 (5)	7 (5)	7 (5)
level 4	9 (8)	6 (5)	7 (5)	7 (5)	8 (6)
level 5	9 (9)	7 (5)	7 (5)	7 (5)	8 (6)
level 6	9 (9)	7 (5)	7 (5)	7 (5)	8 (6)

Table 6.1: Table shows the number of V-cycles required at the 20^{th} time step from five different grid hierarchies of the Wise solver. The scaled 2-norm of the residuals are measured against a fixed stopping criterion that is 1×10^{-8}. A fixed time step size $\delta t = 1\times10^{-3}$ is chosen for all tests. Wise's results that are published in [225] are presented inside of brackets here for comparison.

Results presented in Table 6.1 show a quantitatively good comparison to the results from [225]. However, it may be seen that the Wise solver from our implementation in Campfire commonly requires 1 or 2 extra V-cycles. Lack of details in [225] means that we cannot be sure how these differences arise. For example, the initial conditions for variables μ and p are not described in [225], and the exact implementation of the averaging process for the pressure field is also not described by Wise. Despite the extra V-cycles, the effects caused by modifying values of the parameters in the Wise solver from our implementation show a good agreement to these from [225]. Further validation comes from the convergence rates.

The reductions in the scaled 2-norm of the residuals for the cases of Tests 1 and 2 are illustrated in Fig. 1 in [225]. We validate the results of the Wise solver from our implementation against these presented in [225]. This comparison is shown in Figure 6.1, where the left-hand side is a copy of Fig. 1 from [225], the right-hand side figure shows the results from our implementation. This figure clearly demonstrates the same rates of convergence and explains the extra V-cycles that are required by our implementation, as indicated in Table 6.1. In particular, the main differences between the two solvers are the values of the scaled 2-norm after the very first V-cycle. Possible causes are already discussed, but overall we believe these tests provide excellent validation.

In the following section, convergence tests and multigrid performances from the Wise solver and our multigrid solver are described.

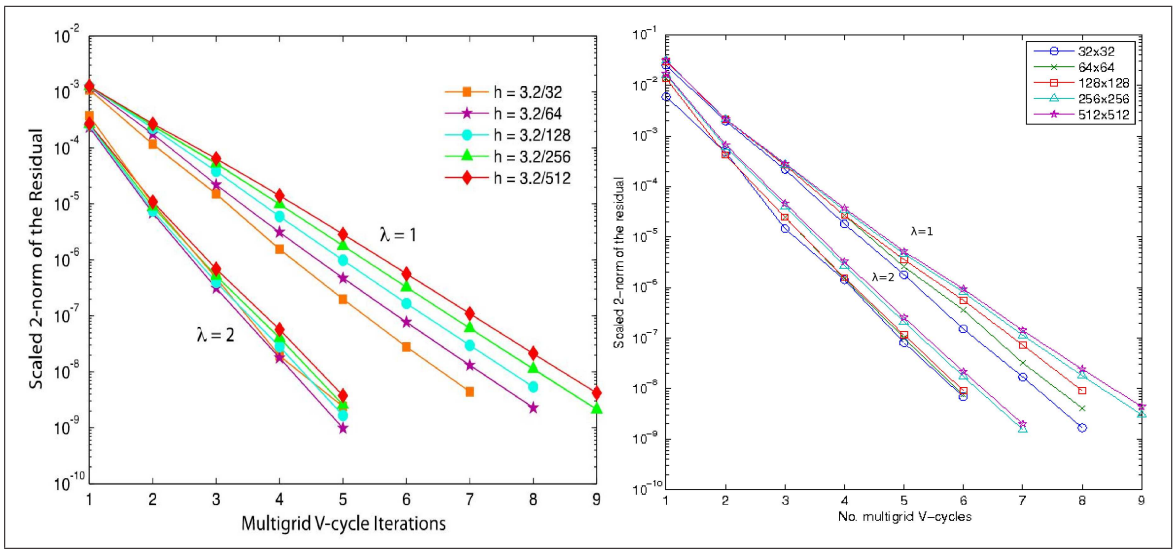

Figure 6.1: Figures show the reductions in the scaled 2-norm of the residuals for cases of Tests 1 and 2 presented in Table 6.1. The left-hand side is a copy of Fig. 1 from [225], and the right-hand side figure shows our results.

6.4.2 Convergence Tests and Multigrid Performance

Having validated our implementation of the Wise solver against [225], convergence tests based upon solution restriction are undertaken here for both multigrid solvers. Results are presented in Section 6.4.2.1. The performance of our multigrid solver with the BDF2 method is also discussed in Section 6.4.2.2.

6.4.2.1 Convergence Tests

In this section, convergence tests and multigrid performance are discussed. The implementation of convergence tests is based upon solution restriction, and this is described previously in Section 5.3.3. Here the infinity norm and the two norm of residuals from all variables are separately measured.

For the purpose of demonstration, convergence tests are done on the solutions at $T = 4 \times 10^{-1}$. The parameters in the CHHS systems are set to be $L_x = L_y = 3.2$, $\varepsilon = 2 \times 10^{-1}$ and $\gamma = 2.0$. In the multigrid solvers, $\lambda = 2$. Note these settings were used by Wise in [225] where he conducted his convergence tests to demonstrate that the semi-implicit scheme gives an overall first order convergence rate. One of the differences between Wise's convergence tests and ours, is that Wise used a refinement path (changes in the time step sizes) which is $\delta t = 0.4h^2$. In our convergence tests, we intend to halve the time step sizes as we refine the grids (as we will show in the following subsection our BDF2 solver is second order). Initially, the time step size in the level 2 case is $\delta t = 1 \times 10^{-2}$,

which then requires 40 time steps to reach $T = 4 \times 10^{-1}$. We use four different grid hierarchies for the convergence tests, and they have the finest grids with the resolutions of 64×64, 128×128, 256×256 and 512×512, respectively. They are in turn denoted as levels 3, 4, 5 and 6, with a grid of 16×16 as the coarsest grid.

Firstly, we present results of our convergence tests using our implementation of the Wise solver (with the semi-implicit scheme). All three different variables are separately tested, and results are shown in Table 6.2. From this table, we see that the ratios of the differences in consecutive solutions for variables ϕ and p are close to 2 for both the infinity norm and the two norm. The corresponding ratios for the variable μ are clearly smaller than 2. Results shown in Table 6.2 indicate the overall convergence rate of using the semi-implicit temporal discretization scheme and the five-point stencil from Wise solver is only first order.

For variable ϕ						
Levels	δt	Time steps	Infinity norm	Ratio	Two norm	Ratio
3	1×10^{-2}	40	-	-	-	-
4	5×10^{-3}	80	6.687×10^{-2}	-	1.714×10^{-2}	-
5	2.5×10^{-3}	160	3.759×10^{-2}	1.78	9.505×10^{-3}	1.80
6	1.25×10^{-3}	320	1.995×10^{-2}	1.88	5.023×10^{-3}	1.89
For variable μ						
3	1×10^{-2}	40	-	-	-	-
4	5×10^{-3}	80	1.401×10^{-2}	-	5.283×10^{-3}	-
5	2.5×10^{-3}	160	7.953×10^{-3}	1.76	3.025×10^{-3}	1.75
6	1.25×10^{-3}	320	6.893×10^{-3}	1.15	2.020×10^{-3}	1.50
For variable p						
3	1×10^{-2}	40	-	-	-	-
4	5×10^{-3}	80	7.672×10^{-3}	-	4.819×10^{-3}	-
5	2.5×10^{-3}	160	4.294×10^{-3}	1.79	2.673×10^{-3}	1.80
6	1.25×10^{-3}	320	2.551×10^{-3}	1.68	1.384×10^{-3}	1.93

Table 6.2: Results show the differences in consecutive solutions measured in the stated norms, followed by the ratio of consecutive differences. These results are generated with the use of Wise solver and the semi-implicit temporal discretization scheme.

The second set of convergence tests are carried out using our multigrid solver with the fully-implicit BDF2 method and the described block version of the red-black Gauss-Seidel smoother. Results of this set of convergence tests are shown in Table 6.3. This table shows near optimal, second order, convergence rates for all three variables. It may also be seen that values of the infinity norm and the two norm (of the difference between consecutive solutions) obtained using our multigrid solver with the BDF2 method, and

shown in Table 6.3, compared against Table 6.2, are $100-1000$ times smaller at these mesh resolutions.

For variable ϕ						
Levels	δt	Time steps	Infinity norm	Ratio	Two norm	Ratio
3	1×10^{-2}	40	-	-	-	-
4	5×10^{-3}	80	1.074×10^{-3}	-	3.885×10^{-4}	-
5	2.5×10^{-3}	160	2.718×10^{-4}	3.95	9.781×10^{-5}	3.97
6	1.25×10^{-3}	320	6.905×10^{-5}	3.93	2.468×10^{-5}	3.96
For variable μ						
3	1×10^{-2}	40	-	-	-	-
4	5×10^{-3}	80	1.141×10^{-4}	-	6.347×10^{-5}	-
5	2.5×10^{-3}	160	2.901×10^{-5}	3.93	1.610×10^{-5}	3.96
6	1.25×10^{-3}	320	7.389×10^{-6}	3.93	4.092×10^{-6}	3.93
For variable p						
3	1×10^{-2}	40	-	-	-	-
4	5×10^{-3}	80	9.110×10^{-5}	-	5.316×10^{-5}	-
5	2.5×10^{-3}	160	2.328×10^{-5}	3.91	1.370×10^{-5}	3.88
6	1.25×10^{-3}	320	5.933×10^{-6}	3.92	3.522×10^{-6}	3.89

Table 6.3: Table shows results from the convergence tests using the infinity norm and the two norm on all three variables ϕ, μ and p from the CHHS system of equations. These results are generated with the use of our multigrid solver and the fully-implicit BDF2 method. Parameters used are described in the text.

In the following section, multigrid performance of our solver with the BDF2 method is evaluated.

6.4.2.2 Multigrid Performance

Previously in Section 6.4.1, the multigrid performance is already presented for our implementation of Wise's solver in Figure 6.1. That shows that the convergence rate (the reduction of the scaled 2-norm of residuals) of the multigrid solver is independent from the grid size. For completeness, in order to evaluate our multigrid solver with the BDF2 method, the simulations with $\lambda = 2$ in the right-hand side of Figure 6.1 are repeated.

Results shown in Figure 6.2 indicate that our multigrid solver, with the use of BDF2 method, shares similar convergence rate (in the 20^{th} time step) as those shown previously on the right-hand side of Figure 6.1. In two cases: 256×256 and 512×512, they require one less V-cycle than the Wise solver to reach the same stopping criterion. It may also be seen that these convergence rates from using different grid hierarchies are independent from the grid size. This is the expected multigrid performance.

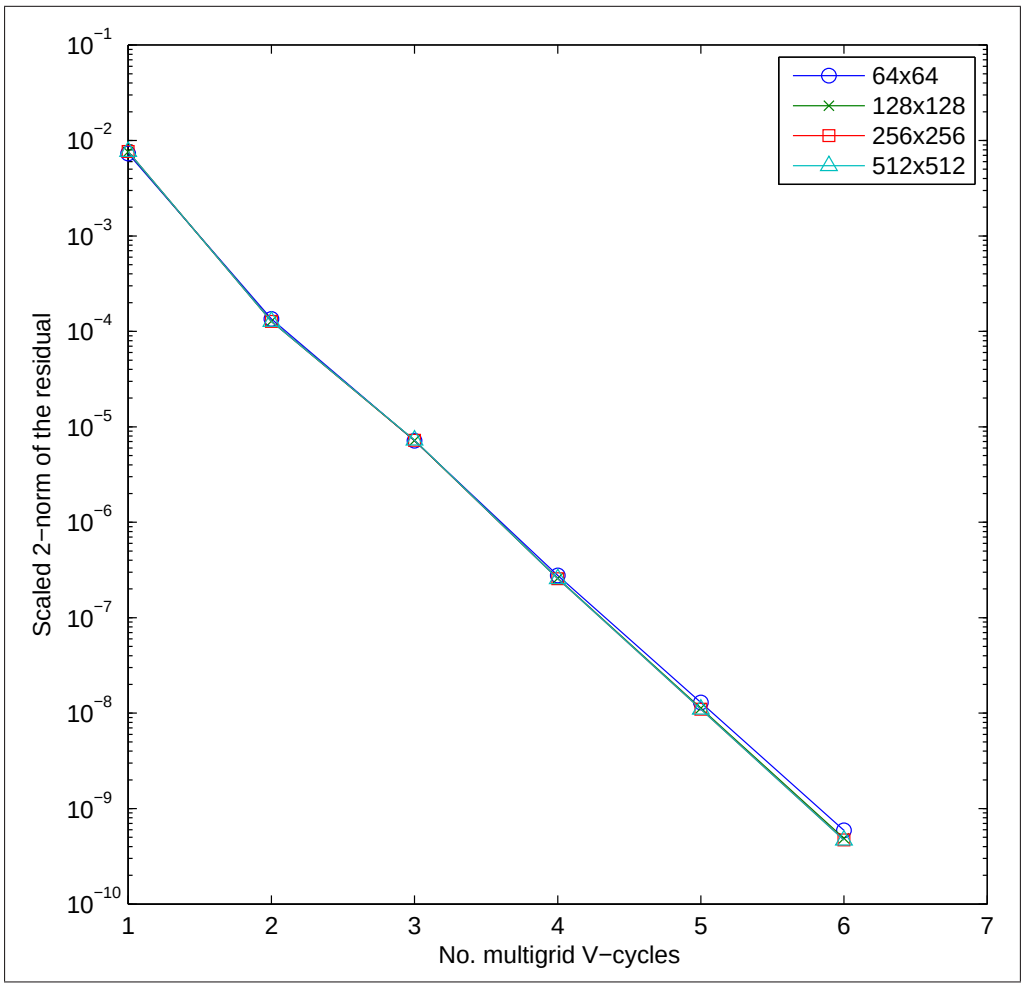

Figure 6.2: Figures show the reductions in the scaled 2-norm of the residuals with $\lambda = 2$. These results are generated by using our multigrid solver with the BDF2 method, and they are repeated simulations which have been previously presented on the right-hand side in Figure 6.1.

In order to demonstrate the linear complexity (i.e. $\mathscr{O}(n)$) of our multigrid solver with the BDF2 method, the simulations shown in Table 6.3 are repeated. These simulations are executed as sequential code and CPU times for all four different grid hierarchies are obtained and the average numbers of V-cycles per time step are summarised. All the works that are presented here are generated from a single workstation which consists of 3.4GHz Dual Quad-Core Intel Xeon processors with 8GB memory. Results are presented in Table 6.4.

Results shown in Table 6.4 suggest a linear increase in the CPU times. This is measured as we quadruple the number of grid points and double the number of time steps, the CPU times are increased by a factor of 8 each case. It also may be seen that the average number of V-cycles required per time step stays a constant. Hence this suggests our

Levels	Finest grid	Total number of time steps	Average V-cycles per time step	CPU time (seconds)
3	64^2	40	5.4	47.1
4	128^2	80	5.3	279.3
5	256^2	160	5.2	2184.2
6	512^2	320	5.0	18400.7

Table 6.4: Table shows the results generated using our multigrid solver with the BDF2 method. CPU times are included which are obtained from using a single CPU. The same simulations are previously presented in Table 6.3 to demonstrate the convergence tests.

multigrid solver performs optimally.

A log-log plot is further used with the average CPU times per time step from all four test cases that are presented in Table 6.4 against the total number of grid points on the corresponding finest grid. This plot is shown in Figure 6.3. Results of this figure show a linear increase in the CPU times as we quadruple the number of grid points and double the number of time steps. Timing results appear slightly sub-linear from the first two cases, however, the overall increase in the larger CPU times is parallel with the line that has the slope of 1.

The following section concludes this chapter with a summary and discussion of the results obtained using our approach for the solution of the CHHS system of equations.

6.4.3 Discussion

In this chapter, two multigrid solvers are implemented in Campfire to solve the CHHS system of equations. This two-phase-field model is described by Wise in [225], and also presented in Section 6.1. The first solver (referred to here as the Wise solver) is intended to replicate the one used in [225], where a first-order semi-implicit temporal scheme is derived using a convex splitting approach. Such a scheme is shown in Section 6.2.1. We validated this solver against results given in [225], and they are quantitatively agreeable. Then through the use of our convergence tests, it is proven that this scheme, together with a second order five-point stencil, is able to provide an overall first order convergence rate.

On the other hand, the second solver (also called our multigrid solve with the BDF2 method) employs a fully-implicit second order BDF2 method as the temporal discretization scheme. Together with a second order five-point stencil, the overall convergence rate is indeed second order. Meanwhile, we also demonstrate the multigrid performance from this solver.

As mentioned previously, the use of the Gauss-Seidel iteration with the red-black

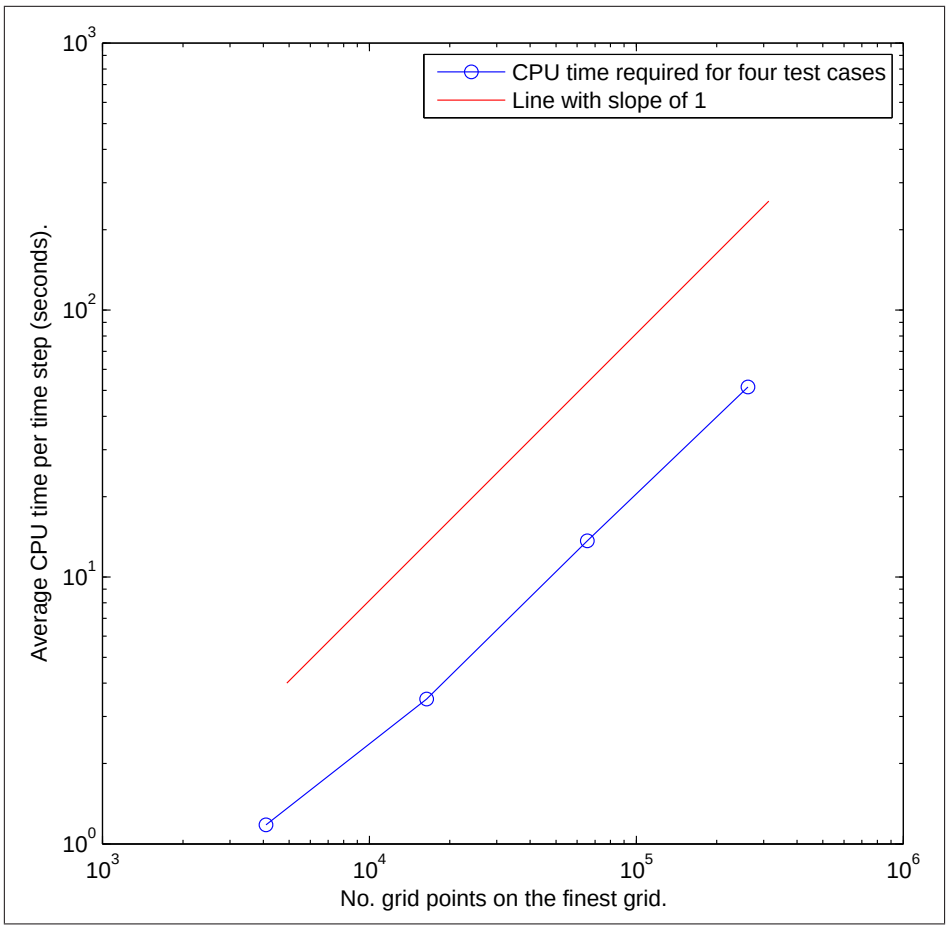

Figure 6.3: Figure shows a log-log plot, in x direction, the total number of grid points from the finest grid is shown, and the average CPU time per time step in seconds is presented in y direction. For comparison, a line with slope of 1 is also shown in the figure.

ordering as the multigrid smoothers and the coarsest grid solver appears to lead to better multigrid convergence rates. This is demonstrated using our multigrid solver with the fully-implicit BDF2 method. The simulations shown in Figure 6.2 are repeated separately by using the red-black Gauss-Seidel iteration and the local Gauss-Seidel, global Jacobi iteration on levels 5 and 6. The resulting convergence rates within the 20^{th} time step are illustrated in Figure 6.4. The figure shows that the red-black Gauss-Seidel approach does indeed provide better convergence rates compared to the local Gauss-Seidel, global Jacobi approach. In the simulations used in Figure 6.4, which consist of 20 time steps, the total number of V-cycles needed by the red-black Gauss-Seidel is less than the numbers required by the local Gauss-Seidel, global Jacobi approach. However, as discussed in Section 3.3.4, the red-black Gauss-Seidel requires two guard cell updates (GCU) per iteration in its implementation in Campfire, whereas the local Gauss-Seidel, global Jacobi only requires one GCU per iteration. Although many improvements have already been

made to increase the efficiency of this time-dominant subroutine (see Section 4.3), the local Gauss-Seidel, global Jacobi is still more efficient. Summaries of the simulations in Figure 6.4 are presented in Table 6.5. Results in this show that the local Gauss-Seidel, global Jacobi iteration is slightly faster for this CHHS system, although its convergence rates are lower than the ones of the red-black Gauss-Seidel (shown in Figure 6.4).

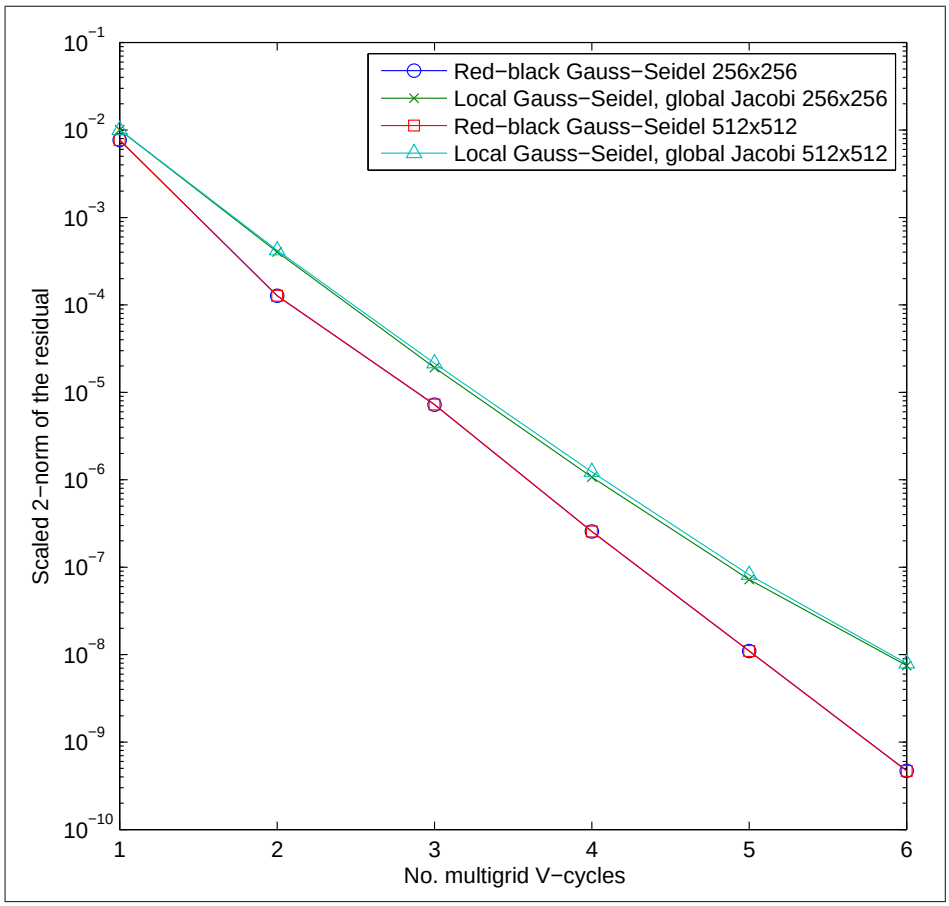

Figure 6.4: Figure shows the convergence rates using the scaled 2-norm of the residuals. These results are generated separably using the red-black Gauss-Seidel and the local Gauss-Seidel, global Jacobi iterations on levels 5 and 6.

To sum up, the model developed in this chapter is used as a stepping stone toward the multi-phase-field model of tumour growth by Wise et al. in [226] that follows. We intend to follow this route, and the results suggest that our implementation in Campfire is able to solve the Cahn-Hilliard-Hele-Shaw type of model. Results also suggest the use of BDF2 method is able to obtain an overall second order convergence rate from this type of model. This provides us with confidence in order to proceed into the next chapter, where a much more complicated model of tumour growth is presented. For the CHHS system of equations, there are no further discussions on adaptivity and parallel computing. Both

Levels	No. time steps	Avg. V-cycle per time step	Total No. V-cycle	CPU times (seconds) per time step
For red-black Gauss-Seidel iteration				
5	20	6.0	120	13.25
6	20	6.0	120	51.10
For local Gauss-Seidel, global Jacobi iteration				
5	20	6.5	130	11.47
6	20	6.6	132	43.81

Table 6.5: Tables the performances of our multigrid solver with two different iterations used for the simulations shown in Figure 6.4.

techniques have already been demonstrated in detail in the previous chapter.

Chapter 7

Parallel, Adaptive, Phase-Field Simulations with Fully-Implicit Time Stepping and FAS Multigrid on Model of Tumour Growth

---

Having described the CHHS system of equations in the previous chapter, here a multi-phase-field model of tumour growth, which is related to the CHHS system, is solved using our parallel, adaptive, multigrid solver. To begin with, a brief literature review on tumour modeling is given in Section 7.1, where different types of tumour growth models are discussed. In this thesis, we are most interested in continuum models which are further described in Section 7.1.2. The multi-phase-field model of tumour growth that is studied here is from Wise et al. [226], and this model is presented in Section 7.2. In [226], a FAS multigrid solver has also been used, and we discuss the differences in the implementation between their solver and ours in Section 7.3. Furthermore, the discrete system arising from this model through applying the finite difference method (see Section 2.1.1.1) and the BDF2 method (see Section 2.1.2.2) is presented in Section 7.3.2, along with a number of novel features of our implementation. Results obtained using our solver are illustrated in Section 7.4. We start, in Section 7.4.1, with a discussion on validating our results against those shown in [226]. Through our presented evidence, we conclude that due to

the sensitivity of certain quantitative outputs from this model to its initial conditions, and a corresponding lack of detailed information from [226], a quantitative validation is only of partial success. Qualitative agreement is also considered thereafter. In Section 7.4.2, through our convergence tests, an overall second order convergence rate is demonstrated: something that was not achieved by the authors of [226] for this multi-phase-field model of tumour growth. In Section 7.4.3, results of 3-D simulations are presented for a range of input parameters for the model. From our experiments and results, we notice a relatively poor multigrid convergence associated with the pressure variable, and in Section 7.5, this issue is described, along with a general discussion which concludes this chapter.

We refer back to Section 1.3.2.4 for the glossary of the multi-phase-field model of tumour growth.

7.1 Brief Literature Review on Tumour Modeling

In order to help the reader understand the process of tumour growth, we generally summarise based upon the work of [12, 14, 37, 155, 165].

Initially a tumour starts with a small number of cells as a cluster within the host tissue. Normally in mathematical models, the initial spot of the tumour is assumed to be away from blood vessels. At this stage, the nutrient supply to the tumour is only through diffusion from the surrounding healthy tissue. It is also generally believed the adhesion forces between tumour cells are much stronger than between healthy cells, so the shape of tumours is commonly compact spheres or ellipses. Due to the supply of nutrient through diffusion being very limited, the compact tumour is typically partitioned into three layers of cells. Cells at the periphery are the proliferating cells (i.e. these cells can divide), these cells are normally well supplied by nutrient. This cell birth process is termed mitosis. Because of the cell proliferation and adhesion, a pressure occurs and pushes towards the centre of the tumour. Cells in the middle layer are the dormant cells. The concentration of nutrient among these cells is very limited, so that cells are alive, however are prohibited from proliferating. Dead cells occur when the nutrient concentration drops below a minimum threshold. This initially happens at the centre of tumour, and as the tumour continues to grow, dead cells accumulate into a necrotic core. This accumulation process of dead cells is termed necrosis. Moreover, through experiments and clinical observations, it is believed there is volume loss in the necrotic core, due to the decomposition of dead cells that can diffuse through the tumour. This is accompanied with other cell loss mechanisms in other regions, such as apoptosis. Apoptosis is the process of programmed cell death, where a cell exceeds its natural lifespan. Therefore, when the solid tumour grows

to a certain size (a few millimeters in diameter), an equilibrium state may occur, that is the birth rate of new cells at the periphery is approximately equal to the rate of volume loss. This phase is termed avascular tumour growth.

The above equilibrium state is uncommon in practice. This is because most tumours during the avascular tumour growth, will release a chemical factor which can induce the nearby blood vessels to grow branches towards the tumour and eventually penetrate it. This process of inducing blood vessels is termed angiogenesis. The induced cells which are originally in the blood vessels are termed endothelial cells, following the migration of these cells, small branches of blood vessels can be formed. When the blood vessels reach the tumour, it means the end of the avascular tumour growth phase, and the next phase is termed vascularized tumour growth. During this phase, the tumour is provided with sufficient nutrient (via the blood vessels) to allow it to grow rapidly. In addition, tumour cells can be transported to other places through blood vessels to develop secondary tumours. This spreading process is termed metastasis. By metastasis, the large number of tumours can lead to physical obstruction or organ malfunction, and eventually kill the host. Therefore, metastasis is believed to be the predominant cause of mortality. Generally by the time a tumour reaches a clinically detectable size, it is already in the vascular growth phase.

7.1.1 Early Mathematical Models of Tumour Growth

Over the last few decades, the understanding of tumourigenesis (the birth and growth of tumours) has developed dramatically and the contribution of mathematical tumour modelling cannot be neglected. The biological experiments and clinical observations have been complemented by the mathematical models that have been developed. Even simple tumour models and simulations can help to support or deny the hypotheses made from observations. Therefore, the two, mathematical modelling and experimental work, when interwoven together, expand our knowledge on tumour growth and eventually contribute to the therapies [14]. For example, the review paper of Byrne et al. [37] gives a number of examples in detail to illustrate how theory can drive experiments and vice versa.

On the other hand, notwithstanding the advances in scientific and medical research, the process of tumourigenesis and its further growth still remains elusive. The procedures of tumour growth itself and its interactions with the host are complex [155]. The disease is initialised by its avascular growth, and Mayneord [163] in 1932 developed a mathematical model based on an observation which illustrated that the volume of tumour tissue starts to grow exponentially. The paper [14] by Araujo and McElwain describes

the reducing thickness of tissue at the periphery layer. This was later demonstrated by Folkman and Hochberg in [77] (experimental paper). Gimbrone in [86] (experimental paper) concluded that the neovascularisation of a tumour plays an imperative role in further continued growth.

Based on the discovery of a solid necrotic core, in 1955 Thomlinson and Gray [215] developed a mathematical model to describe the diffusion and consumption of nutrient, identifying that necrosis is caused when the nutrient drops below a threshold. Tumour cells in this stage can be separated into three types: proliferating cells with sufficient nutrient, dormant cells with less nutrient, which are prohibited from proliferating, and dead cells [14]. Similarly, Burton developed a diffusion model in [35] in 1966 that restricts the nutrient supply to be merely on the tumour surface.

A few years later, Greenspan [95] proposed a diffusion model that simulates avascular tumour growth. Greenspan simulated his results in 1-D in [96]. Later on, with a modified model, Greenspan [97] studied non-symmetric tumour growth in the avascular phase and its stability.

In 1987, Adam developed a reaction-diffusion model with a 1-D second order parabolic PDE governing equation in his series of papers [1–3]. This model was used to validate other earlier models and their results, which included Greenspan's hypotheses of the source of inhibitor.

During the avascular growth phase, a tumour merely grows to a few millimetres in diameter. Further growth is then limited by volume loss in the necrotic region (coupled with other cell loss mechanisms in other regions, more details are given by Durand in [68]). In this stage nutrient supplied through diffusion from healthy tissue can only keep the solid tumour in an equilibrium state [14, 155]. This ends when a chemical, termed tumour angiogenesis factor (TAF), is released by the solid tumour. Such a chemical factor was found by Folkman in [76] (experimental paper) in 1976. A review paper of tumour-induced angiogenesis [12] by Anderson and Chaplain summarised a number of mathematical models of others for angiogenesis.

With a better understanding of tumour growth, tumour modelling research has increased dramatically since the 1990s [14]. A large number of more advanced techniques became available and, with inspiration from other scientific fields, tumour modelling has diverged into three different routes based upon the representation of the tumour tissue [155]. The first is continuum modelling, where the tumours are treated as a collection of tissue. In this tissue-scale type of modelling, the focus is on the concentrations of tumour, and commonly the growth process is governed by partial differential equations [155]. Fast numerical methods (e.g. the multigrid method) can be applied. These types of model are

generally used to compute relatively large time-scale simulations where the shapes of the tumour can be properly formed. This modelling is the main focus of this chapter. Nevertheless, the other two types of modeling present their strengths in different respects. The second approach is discrete modeling, which represents each and every tumour cell individually. This cluster of cells then grows according to a set of specific biophysical rules. However, the need for computational power increases significantly with the number of cells that are modelled [155]. Thus the time-scale of simulations from this type of model is generally limited, and commonly used to determine cell behaviour rather than the total shape of tumours. The final type of modelling is logically driven from the previous two, termed hybrid modelling. This approach intends to combine both continuum and discrete modelling, with research aiming at obtaining the best features of both [155]. Usually, advances made in continuum modelling can be of benefit in hybrid models too.

7.1.2 Continuum Models

Continuum modelling, as stated previously, uses a set of PDEs to model the morphology of tumours. Greenspan's model in [95], mentioned above, is one of the earliest continuum models. These early models usually consist of an ordinary differential equation coupled to one or more reaction-diffusion equations [14]. In this section we describe some more advanced models. Most of these are further categorised as multiphase modelling, which treats the tumour mass (solid tumour) as a multi-component material [155]. Although a large number of models possess this feature, we focus here on the work from two groups that appear to be the most relevant to this research, when considering applications of multigrid methods. They are the group of Lowengrub from University of California at Irvine and the group of H.M. Byrne, now at Oxford University. A few models from others are also included in passing in the following discussion.

In [38], Byrne and Chaplain describe a mathematical model which contains two pairs of coupled reaction-diffusion equations. These two pairs of coupled equations are solved numerically by use of finite difference approximations. The Crank-Nicolson method is used as the time-stepping scheme, and Simpson's rule is employed for the resulting system of equations. In [39], Byrne and Chaplain extended their model, outlined above, to include the necrotic core inside the solid tumour.

Byrne and Preziosi developed another model in [36]. This model treats the tumour as a two-phase material. Within this model, mass exchange between phases plays a fundamentally important role. This two-phase model is later demonstrated in [40], by Byrne et al., to have, when appropriate asymptotic limits are provided, very similar features to the

earlier work from Greenspan [97] and others.

In [118], Hubbard and Byrne develop a four-phase model to simulate vascular tumour growth. In this model, healthy cells, tumour cells, blood vessels, extracellular material and diffusible nutrient can be distinguished. The representation of tumour growth is derived from the principles of conservation of mass and momentum for the various volume fractions. Numerically, the hyperbolic PDEs of mass balance are discretised through a conservative, upwind, finite volume scheme; whereas the generalised Stokes equations for momentum balance employ a finite element scheme. Additionally, the discretization of nutrient reaction-diffusion equations is also based upon a finite element scheme, solved by a Newton iteration. Model simulation is presented on a 2-D unstructured triangular mesh.

The mathematical models of Lowengrub's group are constructed from a different point of view. An early mathematical model that this group used, which is summarised in their review paper [155] and appendices in [52], consists mainly of calculations for cell velocity (through obtaining pressure) and concentration of nutrient. In [52], Cristini et al. applied these two coupled systems and used boundary-integral simulations to perform a fully nonlinear vascularised tumour growth simulation.

Zhang et al. develop a mathematical model in [229] that simulates a full nonlinear model of 2-D tumour growth with angiogenesis. Cell velocity and concentration of nutrient in this model are governed by a Darcy-Stokes law and reaction-diffusion equations, respectively. The chemical factor TAF is described by a reaction-diffusion equation. The spatial discretization method is an adaptive FEM with a level set approach for tracking the tumour boundary. This model uses the midpoint method for equations that may be solved explicitly in time. Based on the work of Zhang et al. [229], Macklin and Lowengrub [157] used a 2-D regular Cartesian mesh and a special level set method to track the tumour boundary.

Li et al. extended this tumour model into 3-D in [150]. The model from Cristini et al. [52] is still used to describe the cell velocity and nutrient diffusion. A new adaptive boundary integral method is proposed to incorporate the 3-D domain. The surface of the tumour is discretized into a mesh of flat triangles. An explicit second order Runge-Kutta time stepping method is used as the temporal discretization scheme and the linear algebraic systems are numerically solved using the Generalized Minimum Residual (GMRES) iterative method.

Macklin et al. in [159] and [158] numerically describe the model from Macklin and Lowengrub [157] in detail. It consists of a number of nonlinear, elliptic and parabolic PDEs. The ghost fluid method is introduced around the tumour/host interface. Away from

the tumour boundary a centred FDM is used to discretize the elliptic and parabolic PDEs. The model that is re-proposed in [159] is solved on a 2-D square domain, a nonlinear adaptive Gauss-Seidel-type iterative method (NAGSI) is used.

Pham et al. summarise three main constitutive laws of treating incompressible fluid in [182], they are Darcy's law, Stokes' law and the combined Darcy-Stokes law. Additionally, further evaluations are carried out for the stability analyses against experimental observations. There are a number of tumour models using these fluid constitutive laws and the one solved here obeys Darcy's law.

In [223], Wise et al. developed a nonlinear (FAS) multigrid method to solve a sixth-order, nonlinear, strongly anisotropic Cahn-Hilliard parabolic equation. Then, Cristini et al. developed a tumour mathematical model in [51] whose governing equation is of Cahn-Hilliard type, which is a fourth-order nonlinear parabolic PDE with a reaction-diffusion equation for the nutrient. An adaptive Cartesian block-structured mesh refinement scheme was applied on a 2-D domain. A centred FDM and an implicit scheme in time were used as spatial and temporal discretization schemes respectively.

7.1.3 Discrete and Hybrid Models

In this section, we briefly describe some of the discrete and hybrid models of tumour growth. The discrete modelling, as described previously, is based upon explicitly tracking and monitoring each individual cell (or representative cells). These cells are updated according to a set of biophysical rules. Generally, discrete modelling is subdivided into two categories: lattice-based (cellular automata) and lattice-free. [155] reviews a number of methods and models that commonly appear in the literature. In [8], Ambrosi and Preziosi also demonstrate a number of discrete models.

Hybrid modelling is an approach that combines both continuum and discrete modelling. We mentioned an example from Zhang et al. [229] previously. In that model, a continuum approach is used to describe the solid tumour growth, whereas a discrete model is applied to simulate the angiogenesis. The model of angiogenesis was presented by Anderson and Chaplain in detail. Anderson and Chaplain in [12] used a reaction-diffusion equation to show the diffusion of TAF. Nevertheless, the migration of the endothelial cells was modelled using the biased random walk approach. Moreover, in [229], Zhang et al. added a feature to Anderson and Chaplain's discrete model, that allowed for the possibility that the entire capillary may be convected by the external cell velocity using a kinematic condition. In [165], McDougall et al. also develop a hybrid model for the process of angiogenesis in the manner of [12, 229]. However, this was with a number of

more complex parameters introduced.

Based on previous studies, Frieboes et al. in [78] proposed a discrete model for angiogenesis; this model combines with a continuum model which is described by Wise et al. in the first paper, [227], of this series. This complex hybrid model allows Frieboes et al. to investigate multispecies tumour invasion, and the morphological instabilities that may underlie invasive phenotypes.

A more advanced hybrid model is proposed by Kim et al. in [132], which uses a continuum approach to describe the quiescent and necrotic layers of the tumour and extracellular matrix, and a discrete model illustrates the activities of growth and proliferation at the periphery of the tumour.

7.2 Model Outline

The multi-phase-field model of tumour growth that is considered in this chapter is solved by Wise et al. in [226]. This model is previously presented in Section 1.3.2.4. For clarity, it is summarised here again.

There are in total four phase-field variables which represent volume fractions in this model, and they are summarised in the glossary at the beginning of this chapter. In addition, there are three assumptions amongst these volume fractions. Firstly, it is assumed the extracellular fluid volume fraction is everywhere constant, $\phi_W(x,y,z,t) = \phi_{W,0} = \text{constant}$. After this assumption, the tumour model only consists of multiple solid cell fractions. Secondly, cells are assumed to be close-packed, and this leads to the sum of the healthy cell volume fraction ϕ_H, the viable tumour cell volume fraction ϕ_V and the dead tumour cell volume fraction ϕ_D to be equal to 1 (i.e. $\phi_H + \phi_V + \phi_D = 1$). Thirdly, it is further assumed that inside the tumour there are only two types of cells: viable and dead. This indicates the total tumour cell volume fraction ϕ_T is the sum of ϕ_V and ϕ_D (i.e. $\phi_T = \phi_V + \phi_D$). Based upon these three assumptions, there are only two phase-field variables that are required to be solved, and they are ϕ_T and ϕ_D. Once the solutions of these two variables are obtained, other variables may be derived from the assumptions made.

The component ϕ_T is assumed to obey the following Cahn-Hilliard-type advection-reaction-diffusion equations:

$$\frac{\partial \phi_T}{\partial t} = M\nabla\cdot(\phi_T\nabla\mu) + S_T - \nabla\cdot(\underline{u}_S\phi_T), \tag{7.1}$$

$$\mu = f'(\phi_T) - \varepsilon^2\nabla^2\phi_T, \tag{7.2}$$

where $M > 0$ is the mobility constant, $f(\phi_T) = \phi_T^2(1-\phi_T)^2/4$ is the quartic double-well potential, $\underline{u}_S$ is the tissue velocity (and is substituted out of the equation later), and $\varepsilon > 0$ is an interface thickness parameter between healthy and tumour tissue. S_T is the net source of tumour cells which depends on ϕ_T, ϕ_V and ϕ_D, and it is given as

$$S_T = nG(\phi_T)\phi_V - \lambda_L\phi_D, \tag{7.3}$$

where n is the concentration of nutrient which is specified later, $\phi_V = \phi_T - \phi_D$, and $\lambda_L \geq 0$ is the rate of tumour cell proliferation. $G(\phi_T)$ is a continuous cut-off function and it is defined by Wise et al. in [226] as the following:

$$G(\phi_T) = \begin{cases} 1 & \text{if } \frac{3\varepsilon}{2} \leq \phi_T \\ \frac{\phi_T}{\varepsilon} - \frac{1}{2} & \text{if } \frac{\varepsilon}{2} \leq \phi_T < \frac{3\varepsilon}{2} \\ 0 & \text{if } \phi_T < \frac{\varepsilon}{2}. \end{cases} \tag{7.4}$$

A similar dynamical equation for solving the volume fraction of dead tumour cells ϕ_D is used:

$$\frac{\partial\phi_D}{\partial t} = M\nabla\cdot(\phi_D\nabla\mu) + S_D - \nabla\cdot(\underline{u}_S\phi_D), \tag{7.5}$$

where S_D is the net source of dead tumour cells, which depends on ϕ_V and ϕ_D. This source term is defined by Wise et al. in [226] as the following:

$$S_D = (\lambda_A + \lambda_N\mathscr{H}(n_N - n))\,\phi_V - \lambda_L\phi_D, \tag{7.6}$$

where λ_A is the death rate of tumour cells from apoptosis, λ_N is the death rate of tumour cells from necrosis, n_N is the necrotic limit, and $\mathscr{H}$ is an Heaviside function. This Heaviside function is given as

$$\mathscr{H}(n_N - n) = \begin{cases} 1 & \text{if } n_N - n \geq 0 \\ 0 & \text{if } n_N - n < 0. \end{cases} \tag{7.7}$$

The tissue velocity $\underline{u}_S$ is assumed to obey Darcy's law, and is defined as

$$\underline{u}_S = -\kappa(\phi_T,\phi_D)(\nabla p - \frac{\gamma}{\varepsilon}\mu\nabla\phi_T), \tag{7.8}$$

where $\kappa > 0$ is the tissue motility function and $\gamma \geq 0$ is a measure of the excess adhesion. An additional assumption made by Wise et al. is that there is no proliferation or death of

the host tissue, thus the velocity is constrained to satisfy

$$\nabla \cdot \underline{u}_S = S_T. \tag{7.9}$$

Instead of solving for the tissue velocity, Equations (7.8) and (7.9) are combined together, and a resulting Poisson-like equation for the cell pressure p can be constructed:

$$-\nabla \cdot (\kappa(\phi_T, \phi_D)\nabla p) = S_T - \nabla \cdot (\kappa(\phi_T, \phi_D)\frac{\gamma}{\varepsilon}\mu\nabla\phi_T). \tag{7.10}$$

A quasi-steady equation is given for the nutrient concentration through diffusion:

$$0 = \nabla \cdot (D(\phi_T)\nabla n) + T_c(\phi_T, n) - n(\phi_T - \phi_D), \tag{7.11}$$

where $D(\phi_T) = D_H(1 - Q(\phi_T)) + Q(\phi_T)$ is the diffusion coefficient, D_H is the nutrient diffusivity in the healthy tissue, $Q(\phi_T)$ is an interpolation function, and $T_c(\phi_T, n) = (v_P^H(1 - Q(\phi_T)) + v_P^T Q(\phi_T))(n_C - n)$ is the nutrient capillary source term. Furthermore, $v_P^H \geq 0$ and $v_P^T \geq 0$ are constants specifying the degree of pre-existing uniform vascularization, $n_C \geq 0$ is the nutrient level in capillaries and the interpolation function, $Q(\phi_T)$, is defined by Wise et al. in [226] as

$$Q(\phi_T) = \begin{cases} 1 & \text{if } 1 \leq \phi_T \\ 3\phi_T^2 - 2\phi_T^3 & \text{if } 0 < \phi_T < 1 \\ 0 & \text{if } \phi_T \leq 0. \end{cases} \tag{7.12}$$

To sum up, this multi-phase-field model of tumour growth consists of a coupled system of five equations, and they are Equations (7.1), (7.2), (7.5), (7.10) and (7.11). There are five dependent variables in total in this system: two phase-field variables, ϕ_T and ϕ_D; three supplementary variables, μ, p and n. These PDEs are valid throughout a domain Ω, there are no internal boundary conditions for the solid tumour, the necrotic core or other variables (this is the advantage of the phase-field approach, see Section 1.3.2.1 for a description of phase-field model). Therefore, only one set of outer boundary conditions is required and this set is the following mixture of Neumann and Dirichlet boundary conditions:

$$\mu = p = 0, \ n = 1, \ \frac{\partial \phi_T}{\partial \nu} = \frac{\partial \phi_D}{\partial \nu} = 0 \text{ on } \partial\Omega, \tag{7.13}$$

where ν denotes the outward-pointing normal to the boundary $\partial\Omega$.

Initial conditions are required, and a specific set for $\phi_T(x, y, z, t = 0)$ and $\phi_D(x, y, z, t = 0)$ is defined later. After both of these are given, $\mu(x, y, z, t = 0)$, $p(x, y, z, t = 0)$ and

$n(x,y,z,t=0)$ can be determined from Equations (7.2), (7.10) and (7.11) respectively. Since μ is an assignment of the Laplacian of ϕ_T plus an explicit source, as shown in Equation (7.2), the initial condition for $\mu(x,y,z,t=0)$ is straightforward. On the other hand, initial conditions for pressure p and nutrient n require an elliptic solver. This is discussed later. In the following section, implementation of our discretization and solver for this multi-phase-field model of tumour growth is described.

7.3 Implementation

In this section, we discuss the differences between the discretizations and multigrid solver used in [226] and our discretizations and multigrid solver. In Section 7.3.1, we provide details of the implementation from [226]. Through the understanding of their solver, our solver is improved with a number of techniques. These techniques, and the discrete system from applying the FDM and the BDF2 method to Equations (7.1), (7.2), (7.5), (7.10) and (7.11), are discussed in Section 7.3.2.

7.3.1 Discussion on the Implementation of Multigrid Solver Used by Wise et al.

Here we do not wish to repeat in full all of the details of the solver developed by Wise et al. However, the implementation of their solver, as described in [226] is summarised here, with the most significant features highlighted.

First of all, the FDM with the use of a seven-point stencil in 3-D (or five-point stencil in 2-D) is used as the spatial discretization scheme. In addition, the advection terms that occur in the system (i.e. $\nabla\cdot(\underline{u}_S\phi_T)$ and $\nabla\cdot(\underline{u}_S\phi_D)$ in Equations (7.1) and (7.5) respectively) are discretized via a third-order WENO reconstruction method [1]. The choice of temporal discretization scheme for the two time-dependent PDEs is based upon the second order Crank-Nicolson method (see Equation (2.10)).

The resulting fully-discrete system at each time step is then solved by using a multigrid solver. It is the nonlinear multigrid method with the FAS (see Section 2.4.4) that is implemented in their solver. Cartesian grids are employed to cover the domain with the use of cell-centred grid points. Additionally, through the use of fast adaptive composite grid method (see [31] for example), the solver is combined with dynamic adaptive mesh refinement (AMR).

[1] The interested reader is directed to [121] for more information on this WENO method.

Within their multigrid solver, the choice of smoother and the coarsest grid solver is a block version of the red-black Gauss-Seidel iteration, which updates all variables on one grid point at a time. However, these five updates for the five variables are not performed simultaneously. Due to their choices of discretization schemes [2], some of the terms are not evaluated implicitly. The result is that Equations (7.1) and (7.2) are strongly coupled, and are updated together at each grid point by performing a 2×2 Cramer's rule for $\phi_{T\,i,j,k}^{l+1}$ and $\mu_{i,j,k}^{l+1}$ (the superscript $l+1$ indicates the solution at the next iteration). Then the discrete form of Equation (7.5), which is weakly coupled with the first two equations, is solved subsequently to update $\phi_{D\,i,j,k}^{l+1}$. The discrete forms of the last two Equations, (7.10) and (7.11), are decoupled from the others, and so variables $p_{i,j,k}^{l+1}$ and $n_{i,j,k}^{l+1}$ are updated by simple division afterwards. The grid transfer operators within their multigrid solver use a cell averaging restriction and a bilinear interpolation. The former is presented in Equation (2.13) and the latter is indicated in Equation (2.12).

Due to the assumptions previously made for this multi-phase-field model, the range of values for the phase-field variable (i.e. ϕ_T) should be between 0 and 1, where 0 represents healthy cells, and 1 represents tumour cells. To prevent these values becoming negative at the numerical level, Wise et al. in [226] introduced a modification to the mobility constant M. For the phase-field variables ϕ_T and ϕ_D, M_T and M_D are now two mobility functions. These two functions are given by

$$M_T(\phi_T) = M \times \left(\max\{\phi_T, 0\} + 1.0 \times 10^{-10}\right), \tag{7.14}$$

$$M_D(\phi_T, \phi_D) = M \times \left(\max\left\{\min\left\{\phi_T, \phi_D\right\}, 0\right\} + 1.0 \times 10^{-10}\right). \tag{7.15}$$

Despite using the second order Crank-Nicolson method as the temporal discretization scheme and the second order central FDM with the 2-D five-point stencil (and the third order WENO method for these advection terms) as the spatial discretization scheme, results in [226] suggest that only an overall first order convergence rate is obtained when second order is hoped for. Furthermore, this overall first order convergence rate is observed with (by setting $\gamma \neq 0$) and without (by setting $\gamma = 0$) the excess surface adhesion term (which includes the advection). Wise et al. stated a few possible explanations. Firstly, it may be caused by not using small enough time step size and/or fine enough resolution of the grids. Secondly, non-smooth functions in the model may be the root causes for the reduction of the order of convergence. More specifically, they identified that the definitions of the cut-off function $G(\phi_T)$, the interpolation function in nutrient capillary source term

[2] The interested reader is directed to Equations (47) - (51) and (60) - (64) in [226] for the complete discrete system used by Wise et al.

$Q(\phi_T)$, and the two mobility functions $M_T(\phi_T)$ and $M_D(\phi_T, \phi_D)$ are not smooth enough for an overall second order convergence rate to be guaranteed.

Having discussed the implementation of their multigrid solver, as well as learning possible reasons for the reduction in the order of convergence, in the following section, the implementation of our multigrid solver in Campfire is described.

7.3.2 Implementation of Our Multigrid Solver in Campfire

In this section, the detailed implementations of our discretization and multigrid solver in Campfire are discussed. This includes a novel implementation, using a penalty term, for this multi-phase-field model of tumour growth. This implementation, combined with many changes made to those described in the previous section, will be shown to yield an overall second order convergence rate (see Section 7.4.2).

First of all, the fully-discrete systems for the five equations is presented. The choice of spatial discretization scheme is the FDM with seven-point stencil (an equivalent 2-D representation of five-point stencil is shown in Equation (2.3)). For time-dependent PDEs (i.e. Equations (7.1) and (7.5)), the BDF2 method with fixed δt is used for the presentation here. Note that the generalizations to an adaptive time-stepping is implemented in practice and that the BDF1 method has to be applied at least for the very first time step. It is also worth noting that since Wise et al. always select $\kappa(\phi_T, \phi_D) \equiv 1$ in [226], and did not give detail of any alternative forms of this κ function, here we also assume $\kappa(\phi_T, \phi_D) \equiv 1$ in

what follows. The discrete system from Equation (7.1) is

$$
\begin{aligned}
&\phi_{T\,i,j,k}^{\tau+1} - \left(\frac{4}{3}\phi_{T\,i,j,k}^{\tau} - \frac{1}{3}\phi_{T\,i,j,k}^{\tau-1}\right) - \\
&\frac{2\delta t}{3h^2}\Bigg[\left(\frac{M\phi_{T\,i+1,j,k}^{\tau+1} + M\phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i+1,j,k}^{\tau+1} - \mu_{i,j,k}^{\tau+1}\right) - \left(\frac{M\phi_{T\,i-1,j,k}^{\tau+1} + M\phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i,j,k}^{\tau+1} - \mu_{i-1,j,k}^{\tau+1}\right) \\
&\quad + \left(\frac{M\phi_{T\,i,j+1,k}^{\tau+1} + M\phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i,j+1,k}^{\tau+1} - \mu_{i,j,k}^{\tau+1}\right) - \left(\frac{M\phi_{T\,i,j-1,k}^{\tau+1} + M\phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i,j,k}^{\tau+1} - \mu_{i,j-1,k}^{\tau+1}\right) \\
&\quad + \left(\frac{M\phi_{T\,i,j,k+1}^{\tau+1} + M\phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i,j,k+1}^{\tau+1} - \mu_{i,j,k}^{\tau+1}\right) - \left(\frac{M\phi_{T\,i,j,k-1}^{\tau+1} + M\phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i,j,k}^{\tau+1} - \mu_{i,j,k-1}^{\tau+1}\right)\Bigg] \\
&-\frac{2\delta t}{3}\left[n_{i,j,k}^{\tau+1} G\left(\phi_{T\,i,j,k}^{\tau+1}\right)\phi_{V\,i,j,k}^{\tau+1} - \lambda_L \phi_{D\,i,j,k}^{\tau+1}\right] - \\
&\frac{2\delta t}{3h^2}\Bigg[\left(\frac{\phi_{T\,i+1,j,k}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i+1,j,k}^{\tau+1} - p_{i,j,k}^{\tau+1}\right) - \left(\frac{\phi_{T\,i-1,j,k}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i,j,k}^{\tau+1} - p_{i-1,j,k}^{\tau+1}\right) \\
&\quad + \left(\frac{\phi_{T\,i,j+1,k}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i,j+1,k}^{\tau+1} - p_{i,j,k}^{\tau+1}\right) - \left(\frac{\phi_{T\,i,j-1,k}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i,j,k}^{\tau+1} - p_{i,j-1,k}^{\tau+1}\right) \\
&\quad + \left(\frac{\phi_{T\,i,j,k+1}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i,j,k+1}^{\tau+1} - p_{i,j,k}^{\tau+1}\right) - \left(\frac{\phi_{T\,i,j,k-1}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i,j,k}^{\tau+1} - p_{i,j,k-1}^{\tau+1}\right)\Bigg] + \\
&\frac{2\delta t\gamma}{3\varepsilon h^2}\Bigg[\left(\frac{\phi_{T\,i+1,j,k}^{\tau+1}\mu_{i+1,j,k}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i+1,j,k}^{\tau+1} - \phi_{T\,i,j,k}^{\tau+1}\right) \\
&\quad - \left(\frac{\phi_{T\,i-1,j,k}^{\tau+1}\mu_{i-1,j,k}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i,j,k}^{\tau+1} - \phi_{T\,i-1,j,k}^{\tau+1}\right) \\
&\quad + \left(\frac{\phi_{T\,i,j+1,k}^{\tau+1}\mu_{i,j+1,k}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i,j+1,k}^{\tau+1} - \phi_{T\,i,j,k}^{\tau+1}\right) \\
&\quad - \left(\frac{\phi_{T\,i,j-1,k}^{\tau+1}\mu_{i,j-1,k}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i,j,k}^{\tau+1} - \phi_{T\,i,j-1,k}^{\tau+1}\right) \\
&\quad + \left(\frac{\phi_{T\,i,j,k+1}^{\tau+1}\mu_{i,j,k+1}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i,j,k+1}^{\tau+1} - \phi_{T\,i,j,k}^{\tau+1}\right) \\
&\quad - \left(\frac{\phi_{T\,i,j,k-1}^{\tau+1}\mu_{i,j,k-1}^{\tau+1} + \phi_{T\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i,j,k}^{\tau+1} - \phi_{T\,i,j,k-1}^{\tau+1}\right)\Bigg] \\
&+\frac{2\delta t}{3\delta}\min\left(\phi_{T\,i,j,k}^{\tau+1}, 0\right) + \frac{2\delta t}{3\delta}\max\left(\phi_{T\,i,j,k}^{\tau+1} - 1, 0\right) = 0.
\end{aligned}
\tag{7.16}
$$

This discretization is based very much on the techniques already demonstrated in the

proceeding chapters. The only feature not previously discussed is the use of the penalty terms $\frac{1}{\delta}\min\left(\phi_{T\,i,j,k}^{\tau+1},0\right)$ and $\frac{1}{\delta}\max\left(\phi_{T\,i,j,k}^{\tau+1}-1,0\right)$. These terms have no impact when $0 \leq \phi_{T\,i,j,k}^{\tau+1} \leq 1$ but create a large correction to the system whenever ϕ_T tries to take a value outside of this interval. The larger the choice of the penalty parameter δ the larger this correction becomes - forcing the values of ϕ_T to be close to this range but at the expense of adding to the nonlinearity of the resulting system. The default value of δ selected in this chapter is 10^{-4}.

The discrete system from Equation (7.2) is simply:

$$\mu_{i,j,k}^{\tau+1} - \frac{2\phi_{T\,i,j,k}^{\tau+1} + 4\left(\phi_{T\,i,j,k}^{\tau+1}\right)^3 - 6\left(\phi_{T\,i,j,k}^{\tau+1}\right)^2}{4} + \frac{\varepsilon^2}{h^2}\left(\phi_{T\,i+1,j,k}^{\tau+1} + \phi_{T\,i-1,j,k}^{\tau+1} + \phi_{T\,i,j+1,k}^{\tau+1} + \phi_{T\,i,j-1,k}^{\tau+1} + \phi_{T\,i,j,k+1}^{\tau+1} + \phi_{T\,i,j,k-1}^{\tau+1} - 6\phi_{T\,i,j,k}^{\tau+1}\right) = 0. \tag{7.17}$$

The discrete system from Equation (7.5) is

$$
\begin{aligned}
&\phi_{D\,i,j,k}^{\tau+1} - \left(\frac{4}{3}\phi_{D\,i,j,k}^{\tau} - \frac{1}{3}\phi_{D\,i,j,k}^{\tau-1}\right) - \\
&\frac{2\delta t}{3h^2}\Bigg[\left(\frac{M\phi_{D\,i+1,j,k}^{\tau+1} + M\phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i+1,j,k}^{\tau+1} - \mu_{i,j,k}^{\tau+1}\right) - \left(\frac{M\phi_{D\,i-1,j,k}^{\tau+1} + M\phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i,j,k}^{\tau+1} - \mu_{i-1,j,k}^{\tau+1}\right) \\
&\quad + \left(\frac{M\phi_{D\,i,j+1,k}^{\tau+1} + M\phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i,j+1,k}^{\tau+1} - \mu_{i,j,k}^{\tau+1}\right) - \left(\frac{M\phi_{D\,i,j-1,k}^{\tau+1} + M\phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i,j,k}^{\tau+1} - \mu_{i,j-1,k}^{\tau+1}\right) \\
&\quad + \left(\frac{M\phi_{D\,i,j,k+1}^{\tau+1} + M\phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i,j,k+1}^{\tau+1} - \mu_{i,j,k}^{\tau+1}\right) - \left(\frac{M\phi_{D\,i,j,k-1}^{\tau+1} + M\phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(\mu_{i,j,k}^{\tau+1} - \mu_{i,j,k-1}^{\tau+1}\right)\Bigg] - \\
&-\frac{2\delta t}{3}\left[\left(\lambda_A + \lambda_N \mathscr{H}^{\text{smooth}}\left(n_N - n_{i,j,k}^{\tau+1}\right)\right)\phi_{V\,i,j,k}^{\tau+1} - \lambda_L \phi_{D\,i,j,k}^{\tau+1}\right] - \\
&\frac{2\delta t}{3h^2}\Bigg[\left(\frac{\phi_{D\,i+1,j,k}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i+1,j,k}^{\tau+1} - p_{i,j,k}^{\tau+1}\right) - \left(\frac{\phi_{D\,i-1,j,k}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i,j,k}^{\tau+1} - p_{i-1,j,k}^{\tau+1}\right) \\
&\quad + \left(\frac{\phi_{D\,i,j+1,k}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i,j+1,k}^{\tau+1} - p_{i,j,k}^{\tau+1}\right) - \left(\frac{\phi_{D\,i,j-1,k}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i,j,k}^{\tau+1} - p_{i,j-1,k}^{\tau+1}\right) \\
&\quad + \left(\frac{\phi_{D\,i,j,k+1}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i,j,k+1}^{\tau+1} - p_{i,j,k}^{\tau+1}\right) - \left(\frac{\phi_{D\,i,j,k-1}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}}{2}\right)\left(p_{i,j,k}^{\tau+1} - p_{i,j,k-1}^{\tau+1}\right)\Bigg] + \\
&\frac{2\delta t\gamma}{3\varepsilon h^2}\Bigg[\left(\frac{\phi_{D\,i+1,j,k}^{\tau+1}\mu_{i+1,j,k}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i+1,j,k}^{\tau+1} - \phi_{T\,i,j,k}^{\tau+1}\right) \\
&\quad - \left(\frac{\phi_{D\,i-1,j,k}^{\tau+1}\mu_{i-1,j,k}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i,j,k}^{\tau+1} - \phi_{T\,i-1,j,k}^{\tau+1}\right) \\
&\quad + \left(\frac{\phi_{D\,i,j+1,k}^{\tau+1}\mu_{i,j+1,k}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i,j+1,k}^{\tau+1} - \phi_{T\,i,j,k}^{\tau+1}\right) \\
&\quad - \left(\frac{\phi_{D\,i,j-1,k}^{\tau+1}\mu_{i,j-1,k}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i,j,k}^{\tau+1} - \phi_{T\,i,j-1,k}^{\tau+1}\right) \\
&\quad + \left(\frac{\phi_{D\,i,j,k+1}^{\tau+1}\mu_{i,j,k+1}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i,j,k+1}^{\tau+1} - \phi_{T\,i,j,k}^{\tau+1}\right) \\
&\quad - \left(\frac{\phi_{D\,i,j,k-1}^{\tau+1}\mu_{i,j,k-1}^{\tau+1} + \phi_{D\,i,j,k}^{\tau+1}\mu_{i,j,k}^{\tau+1}}{2}\right)\left(\phi_{T\,i,j,k}^{\tau+1} - \phi_{T\,i,j,k-1}^{\tau+1}\right)\Bigg] \\
&+\frac{2\delta t}{3\delta}\min\left(\phi_{D\,i,j,k}^{\tau+1}, 0\right) + \frac{2\delta t}{3\delta}\max\left(\phi_{D\,i,j,k}^{\tau+1} - 1, 0\right) = 0.
\end{aligned}
\tag{7.18}
$$

Again we have made use of penalty terms to penalize any deviation of ϕ_D from the in-

terval $[0,1]$. Furthermore, we have smoothed the source term S_D (see Equation (7.6)) by replacing $\mathscr{H}$ with $\mathscr{H}^{\text{smooth}}$, a smoothed Heaviside function, that is defined later in this section.

The discrete system from Equation (7.10) is given by:

$$
\begin{aligned}
&-\frac{1}{h^2}\left(p^{\tau+1}_{i+1,j,k}+p^{\tau+1}_{i-1,j,k}+p^{\tau+1}_{i,j+1,k}+p^{\tau+1}_{i,j-1,k}+p^{\tau+1}_{i,j,k+1}+p^{\tau+1}_{i,j,k-1}-6p^{\tau+1}_{i,j,k}\right)\\
&+\frac{\gamma}{\varepsilon h^2}\left[\left(\frac{\mu^{\tau+1}_{i+1,j,k}+\mu^{\tau+1}_{i,j,k}}{2}\right)\left(\phi^{\tau+1}_{T\,i+1,j,k}-\phi^{\tau+1}_{T\,i,j,k}\right)-\left(\frac{\mu^{\tau+1}_{i-1,j,k}+\mu^{\tau+1}_{i,j,k}}{2}\right)\left(\phi^{\tau+1}_{T\,i,j,k}-\phi^{\tau+1}_{T\,i-1,j,k}\right)\right.\\
&\quad+\left(\frac{\mu^{\tau+1}_{i,j+1,k}+\mu^{\tau+1}_{i,j,k}}{2}\right)\left(\phi^{\tau+1}_{T\,i,j+1,k}-\phi^{\tau+1}_{T\,i,j,k}\right)-\left(\frac{\mu^{\tau+1}_{i,j-1,k}+\mu^{\tau+1}_{i,j,k}}{2}\right)\left(\phi^{\tau+1}_{T\,i,j,k}-\phi^{\tau+1}_{T\,i,j-1,k}\right)\\
&\quad\left.+\left(\frac{\mu^{\tau+1}_{i,j,k+1}+\mu^{\tau+1}_{i,j,k}}{2}\right)\left(\phi^{\tau+1}_{T\,i,j,k+1}-\phi^{\tau+1}_{T\,i,j,k}\right)-\left(\frac{\mu^{\tau+1}_{i,j,k-1}+\mu^{\tau+1}_{i,j,k}}{2}\right)\left(\phi^{\tau+1}_{T\,i,j,k}-\phi^{\tau+1}_{T\,i,j,k-1}\right)\right]\\
&\quad-\left[n^{\tau+1}_{i,j,k}G\left(\phi^{\tau+1}_{T\,i,j,k}\right)\phi^{\tau+1}_{V\,i,j,k}-\lambda_L\phi^{\tau+1}_{D\,i,j,k}\right]=0.
\end{aligned}
\tag{7.19}
$$

Finally, the discrete system from Equation (7.11) is

$$\begin{aligned}
&\frac{1}{h^2}\Bigg[\frac{D_H\left(1-Q\left(\phi_{T\,i+1,j,k}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i+1,j,k}^{\tau+1}\right)+D_H\left(1-Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)}{2}\\
&\quad\left(n_{i+1,j,k}^{\tau+1}-n_{i,j,k}^{\tau+1}\right)\\
&-\frac{D_H\left(1-Q\left(\phi_{T\,i-1,j,k}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i-1,j,k}^{\tau+1}\right)+D_H\left(1-Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)}{2}\\
&\quad\left(n_{i,j,k}^{\tau+1}-n_{i-1,j,k}^{\tau+1}\right)\\
&+\frac{D_H\left(1-Q\left(\phi_{T\,i,j+1,k}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i,j+1,k}^{\tau+1}\right)+D_H\left(1-Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)}{2}\\
&\quad\left(n_{i,j+1,k}^{\tau+1}-n_{i,j,k}^{\tau+1}\right)\\
&-\frac{D_H\left(1-Q\left(\phi_{T\,i,j-1,k}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i,j-1,k}^{\tau+1}\right)+D_H\left(1-Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)}{2}\\
&\quad\left(n_{i,j,k}^{\tau+1}-n_{i,j-1,k}^{\tau+1}\right)\\
&+\frac{D_H\left(1-Q\left(\phi_{T\,i,j,k+1}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i,j,k+1}^{\tau+1}\right)+D_H\left(1-Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)}{2}\\
&\quad\left(n_{i,j,k+1}^{\tau+1}-n_{i,j,k}^{\tau+1}\right)\\
&-\frac{D_H\left(1-Q\left(\phi_{T\,i,j,k-1}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i,j,k-1}^{\tau+1}\right)+D_H\left(1-Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)\right)+Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)}{2}\\
&\quad\left(n_{i,j,k}^{\tau+1}-n_{i,j,k-1}^{\tau+1}\right)\Bigg]\\
&+\left\{V_P^H\left[1-Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)\right]+V_P^T Q\left(\phi_{T\,i,j,k}^{\tau+1}\right)\right\}\left(n_C-n_{i,j,k}^{\tau+1}\right)\\
&-n_{i,j,k}^{\tau+1}\phi_{V\,i,j,k}^{\tau+1}=0.
\end{aligned}\tag{7.20}$$

Unlike the implementation in [226] that is described in the previous section, where M was replaced by two mobility functions, here M is a constant. The use of penalty terms (i.e. $\frac{1}{\delta}\min(\phi,0)$ and $\frac{1}{\delta}\max(\phi-1,0)$) already provides the functionality to regulate values of phase-field variables close to the range of 0 to 1. It is worth noting that for the advection terms, the central difference (seven-point) 3-D stencil is applied and treated similarly as the others terms. This is known not to be stable unless h is sufficiently small and/or sufficient diffusion is present, however we observe no adverse effects for our simulations. To reflect the suggestions made by Wise et al. in [226] (described in the previous

section), and through our own experiments, we identified the non-smooth Heaviside function to be a potential cause that a higher rate of convergence is not obtained. Therefore, a smoothed Heaviside function $\mathscr{H}^{\text{smooth}}$ is employed to replace the original Heaviside function shown in Equation (7.7). This is defined as

$$\mathscr{H}^{\text{smooth}}(n_N - n) = \begin{cases} 1 & \text{if } n_N - n \geq \varepsilon^{\text{smooth}} \\ -\frac{1}{4(\varepsilon^{\text{smooth}})^3}(n_N - n)^3 + \frac{3}{4\varepsilon^{\text{smooth}}}(n_N - n) + \frac{1}{2} & \text{if } -\varepsilon^{\text{smooth}} \leq n_N - n \leq \varepsilon^{\text{smooth}} \\ 0 & \text{if } n_N - n < -\varepsilon^{\text{smooth}}, \end{cases} \tag{7.21}$$

where $\varepsilon^{\text{smooth}}$ controls the steepness of the smooth transition between 0 and 1.

Within our multigrid solver, for the smoothers and the coarsest grid solver, the nonlinear, block version of "local Gauss-Seidel, global Jacobi" iteration is applied (as a reminder, this is a Gauss-Seidel approach within blocks of the mesh but we only use "old" values from neighbouring blocks to update values at the edge of blocks - based upon a single guard cell update prior to each sweep). A Jacobi version of this iteration is already shown previously in Equation (2.36). The implementation of the non-time-dependent equations in Campfire was previously described in Sections 5.3.1 and 6.3.2, so here it is not repeated. It is worth noting that in our nonlinear, block version of "local Gauss-Seidel, global Jacobi" iteration, for this multi-phase-field model of tumour growth, the Jacobian matrix becomes a 5×5 matrix. The inverse of this matrix for each cell is calculated by Gaussian elimination and an upper triangular solver with the use of backward substitution. The grid transfer operations are, for restriction, the simple cell averaging operator (see Equation (2.13) for an 2-D example, and the 3-D version is straightforward); and for interpolation, bilinear (or 3-D trilinear) interpolation (see Equation (2.12) for a 2-D example) is employed. It is worth noting that, although this multi-phase-field model of tumour growth is of Cahn-Hilliard-type and strongly related to the CHHS system of equations that is described in the previous chapter, the averaging process is not required to define a unique pressure due to the use of the Dirichlet boundary condition shown in Equation (7.13) for the pressure term. However, a related issue is discussed later, in Section 7.5.

Having summarised the implementation of our multigrid solver, here we also explain the process for obtaining the initial conditions for all variables. Assuming $\phi_T(x,y,z,t=0)$ is prescribed (its initial condition in 2-D is partially given in [226] and is described later), the initial condition for $\mu(x,y,z,t=0)$ is straightforward since μ is a function of ϕ_T as shown in Equation (7.2). The initial condition for ϕ_D is generally taken as $\phi_D(x,y,z,t=0)=0$, which assumes that initially there are no dead tumour cells. Like the pressure variable in the CHHS system of equations that is described in the previous

chapter, the pressure and nutrient variables p and n here require an application of a solver to obtain their consistent initial conditions. A simple iterative solver may be used, however, for grids with finer resolutions, the computational cost may increase significantly. Therefore, a multigrid solver is used just to obtain the initial conditions for $p(x,y,z,t=0)$ and $n(x,y,z,t=0)$. This solves for the steady state solution of n firstly since Equation (7.11) is not dependent on p. The initial guess for n is $n^{l=0}=1$, and superscript l indicates the number of iterations. The smoothers and the coarsest grid solver in this multigrid solver use a simple "local Gauss-Seidel, global Jacobi" iteration, which comes from re-arranging Equation (7.20) to the following form:

$$f_n n_{i,j,k}^{l+1} = g_n, \tag{7.22}$$

where f_n and g_n are two functions that are not dependent upon $n_{i,j,k}^{l+1}$. The update routine is a simple division (i.e. $n_{i,j,k}^{l+1} = g_n/f_n$). The stopping criterion is dependent upon the infinity norm of residuals of n, and it terminates the solver for n when $||r_{n(x,y,z,t=0)}||_\infty \leq 1\times 10^{-9}$. Then this multigrid solver is applied to obtain the initial solution of p, with the initial guess $p^{l=0}=0$. The smoothers and the coarsest grid solver depend upon a simple division at each nodal update as the pressure Equation (7.19) may be re-arranged into

$$f_p p_{i,j,k}^{l+1} = g_p, \tag{7.23}$$

where f_p and g_p are two functions that are not dependent upon $p_{i,j,k}^{l+1}$. The stopping criterion is dependent upon the infinity norm of residual of p, and it terminates the solver for p when $||r_{p(x,y,z,t=0)}||_\infty \leq 1\times 10^{-9}$.

In the following section, results obtained using the described multigrid solver are described.

7.4 Results

In this section, we present the results from our multigrid solver. Firstly, we use 2-D results for the validation in Section 7.4.1. We also include a discussion which suggests that quantitative validation may not be straightforward for this model, due to the sensitivity of the solution to the initial conditions. On the other hand, our multigrid solvers have already been shown to provide excellent validations for other models in previous chapters, which therefore provides confidence in the results generated for this tumour growth model. In Section 7.4.2, convergence tests are used to demonstrate, for the first time, that an overall second order convergence rate can be obtained for this model of tumour growth (Wise et

al. were unable to achieve this in [226]). 3-D results are presented in Section 7.4.3, where we investigate a number of runs using different input parameters to the model. It is worth noting the simulations presented in this section are generated from using the computer cluster ARC2. However, since there already are discussions about the parallel scaling in Chapters 4 and 5, the parallel issues are not mentioned here.

7.4.1 Validation

In this section, results generated from the described multigrid solver are validated against those in [226] [3]. Only 2-D solutions are considered for the validation provided here.

First of all, the values of the parameters that are used in the multi-phase-field model of tumour growth are presented in Table 7.1. These values shown in Table 7.1 are the same as used in [226]. It is worth noting that when $\gamma = 0$, (where γ is the value of excess adhesion force at the diffuse tumour/host-tissue interface), several terms in Equations (7.16), (7.18) and (7.19) disappear. In [226], Wise et al. state that when $\gamma \neq 0$ they observe a deterioration in their multigrid convergence. Furthermore, for their discretization they claim that this also prevents a higher order convergence rate from being obtained. We discuss the multigrid convergence later in Section 7.5, and describe the influence of the γ terms to the convergence rate in Section 7.4.2.

Parameters	Values	Parameters	Values
M	10.0	ε	0.1
λ_L	1.0	λ_A	0.0
λ_N	3.0	γ	$-0.1/0.0/0.1$
n_N	0.4	D_H	1.0
v_P^H	0.5	v_P^T	0.0
δ	0.0001	$\varepsilon^{\text{Heaviside}}$	0.2
n_C	1.0		

Table 7.1: The parameters of the multi-phase-field model of tumour growth that were used by Wise et al. in [226].

The 2-D computational domain Ω has Cartesian coordinates $(x,y) \in \Omega = (0,40) \times (0,40)$. The initial condition for the phase-field variable ϕ_T used in [226] is not stated however they do state that the contour $\phi_T = 0.5$ is the curve

$$\frac{(x-20)^2}{1.1} + (y-20)^2 \leq 2^2. \tag{7.24}$$

[3] Some of the results from Wise et al. are also published in [227], however, they are very similar and only those in [226] are considered here.

In order to approximate this we initially choose $\phi_T = 1$ inside the ellipse (7.24) and $\phi_T = 0$ outside. We then smooth ϕ_T by applying a number of sweeps of a simple Jacobi iteration. At each smoothing sweep, this Jacobi iteration is

$$\phi_{T\,i,j}^{l+1,t=0} = \frac{1}{4}\left(\phi_{T\,i+1,j}^{l,t=0} + \phi_{T\,i-1,j}^{l,t=0} + \phi_{T\,i,j+1}^{l,t=0} + \phi_{T\,i,j-1}^{l,t=0}\right), \tag{7.25}$$

where the superscript $l+1$ denotes the current iteration whereas l denotes the previous iteration. We find that if the initial condition for ϕ_T is not sufficiently smooth then this impacts on the maximum δt for the early time steps. For a 128×128 finest mesh we typically take 10 smoothing sweeps, and we increase this number of sweeps (by a factor of 4) for each additional level of mesh refinement (at lease when undertaking convergence tests, so as to get consistent initial data).

Having described the initial conditions, the results presented here for the purpose of validation are derived from the grid hierarchy which consists of 7 mesh refinements: $8 \times 8 - 512 \times 512$. The finest mesh refinement is the same as Wise et al. used in [226], however, the choice of the coarsest grid is much coarser in our solver: 8×8, compares to 64×64 which is used in [226]. The reason for this is that a coarser coarsest grid may solve the coarsest problem "exactly" with much less work. This has a strong influence on our solver, with poorer multigrid convergence when the coarsest problem is poorly solved. We will discuss this issue in detail later in this chapter.

First of all, we present results generated using parameters shown in Table 7.1, with $\gamma = 0.1$. We present a copy of the corresponding results shown in [226] in the left column of Figure 7.1, and our results in the right column of the same figure. It is worth noting that the solutions plotted in [226] are showing ϕ_V instead of ϕ_T (for example), which is the volume fraction of viable tumour tissue and is calculated using $\phi_T - \phi_D$ through the assumption (i.e. $\phi_T = \phi_V + \phi_D$) made in Section 7.2. We follow this approach by presenting the solutions of ϕ_V in this section. It seems that our results show a slightly "thicker" shape of the tumour. However, both results show a similar evolution for the tumour and a "ribbon" effect indicating the presence of a necrotic core. Both results are generated using AMR: in [226], Wise et al. state that their AMR seeks to capture the tumour/host interface which has a steep gradient; our choice for the AMR strategy is a conservative one, that takes into account the gradients of ϕ_T, ϕ_D, p and n. Like the one presented in Equation (5.20), but for four variables. The refining criterion is 0.0005 and the coarsening criterion is 0.00001. Due to the profile of p, which covers a much larger area than the initial seed of ϕ_T, we further impose that no mesh blocks can be coarsened within the central area of the domain. We have also enabled the adaptive time-stepping.

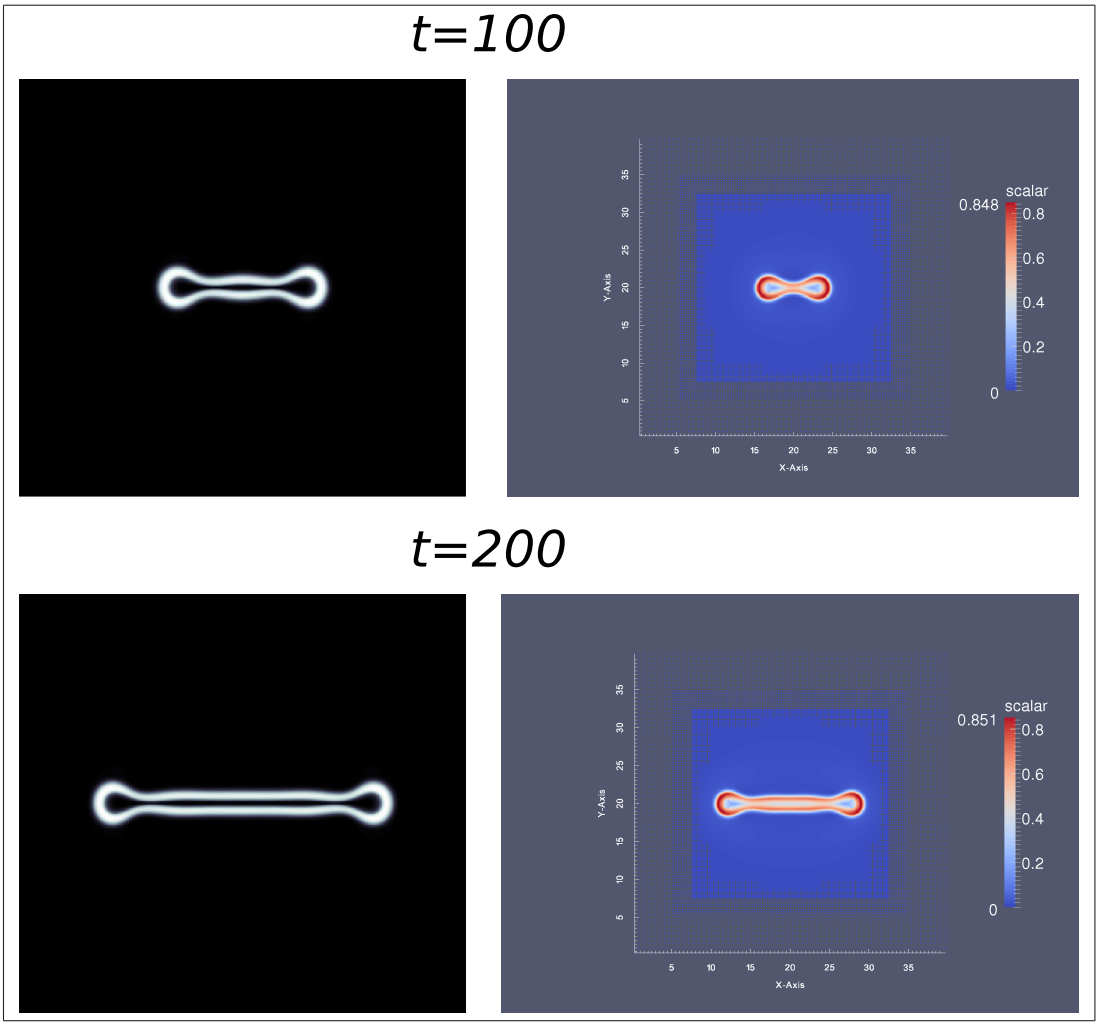

Figure 7.1: Solutions of ϕ_V from solving the described model of tumour growth using the parameters in Table 7.1, with $\gamma = 0.1$. The two figures in the left column are the corresponding results from [226], the white colour represents the value close1, the black colour shows the value close 0 and the enclosed black regions indicate necrotic tumour tissue. The other two figures in the right column are from our solver, the red colour shows a value that is close to 1, and the blue colour shows a value which is close to 0.

Secondly, we present the results from using $\gamma = 0.0$ in Figure 7.2. It is clear that our results are rather different from the ones presented in [226]. The possible reasons for these differences will be addressed below. For completeness, we also present the results from using $\gamma = -0.1$ in Figure 7.3, which also suggests the two solutions evolve differently.

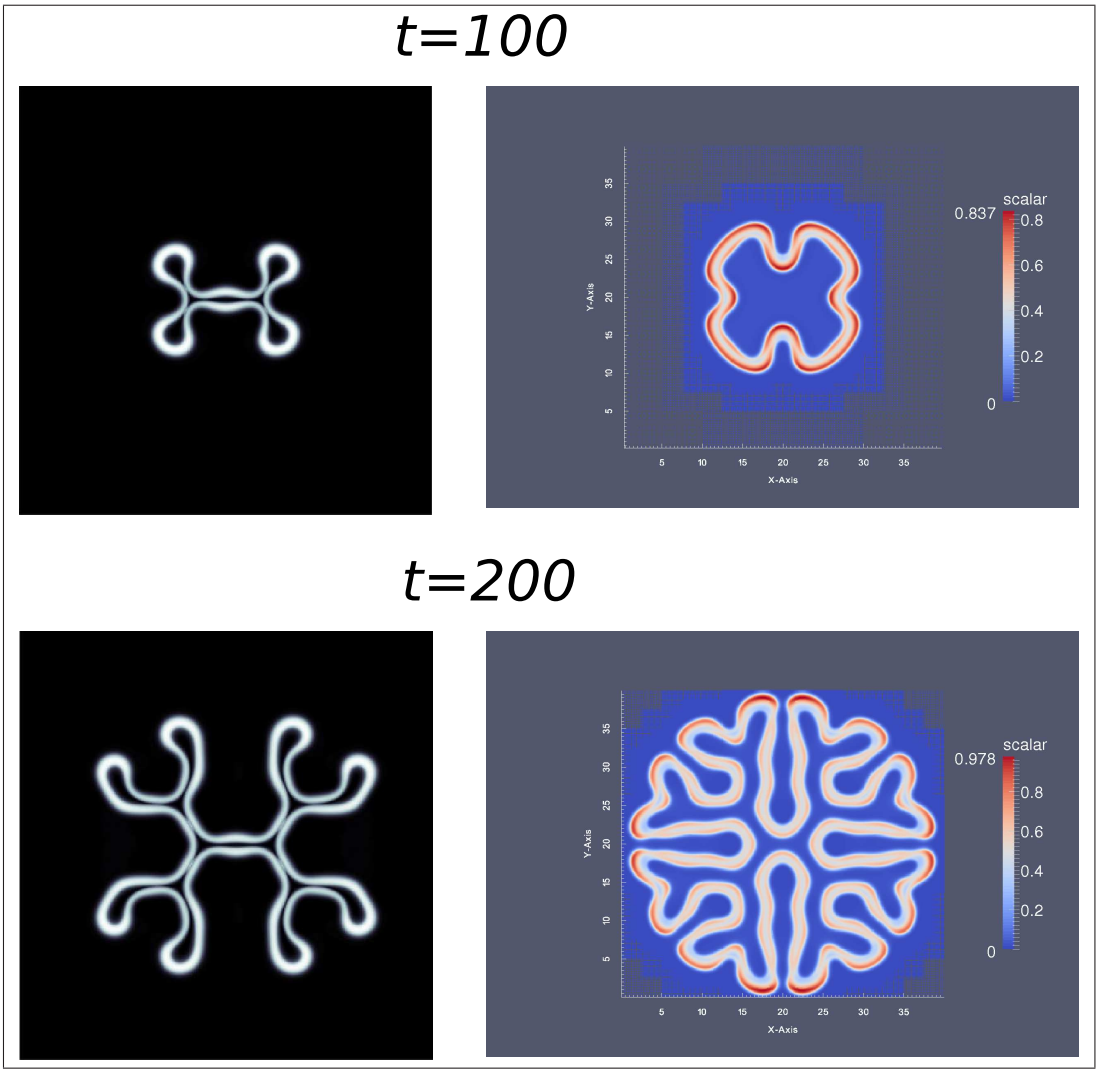

Figure 7.2: Solutions of ϕ_V from solving the described model of tumour growth using the parameters in Table 7.1, with $\gamma = 0.0$. The two figures in the left column are the corresponding results from [226], the white colour represents the value close to 1, the black colour shows the value close to 0 and the enclosed black regions indicate necrotic tumour tissue. The other two figures in the right column are from our solver, the red colour shows a value that is close to 1, and the blue colour shows a value which is close to 0.

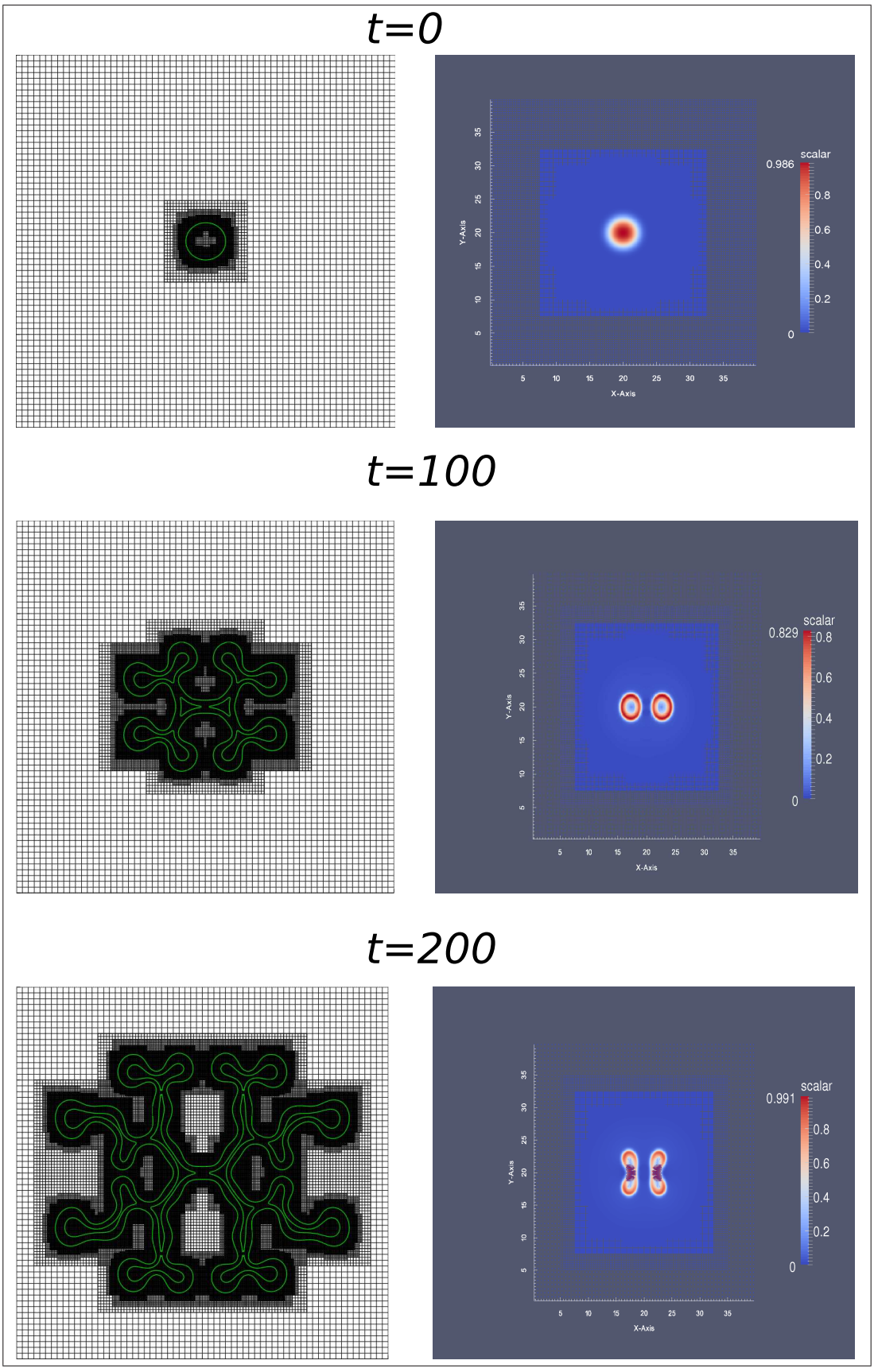

Figure 7.3: Solutions of ϕ_V from solving the described model of tumour growth using the parameters in Table 7.1, with $\gamma = -0.1$. The two figures in the left column are the corresponding results from [226], the green colour represents the value close to 0.5, the black colour (and white background) shows the value close to 0. The other two figures in the right column are from our solver, the red colour shows a value that is close to 1, and the blue colour shows a value which is close to 0.

As noted above, full and precise initial conditions are not stated in [226]. In the remainder of this section we show that the evolution of the tumour is in fact very sensitive to the precise choice of initial condition for this model. In order to demonstrate this, we choose the case with $\gamma = 0.0$ (which is presented in Figure 7.2), and we reduced the number of sweeps applied on the finest grid for smoothing the initial condition of ϕ_T to just 20 (down from 160). The results with reduced sweeps are presented in the left column of Figure 7.4, and for comparison, we presented the corresponding results from using the original number of sweeps (i.e. 160) in the right column of this figure. It seems that from the results in this figure, the smoothness of the initial condition has a strong influence on the evolution of the tumour.

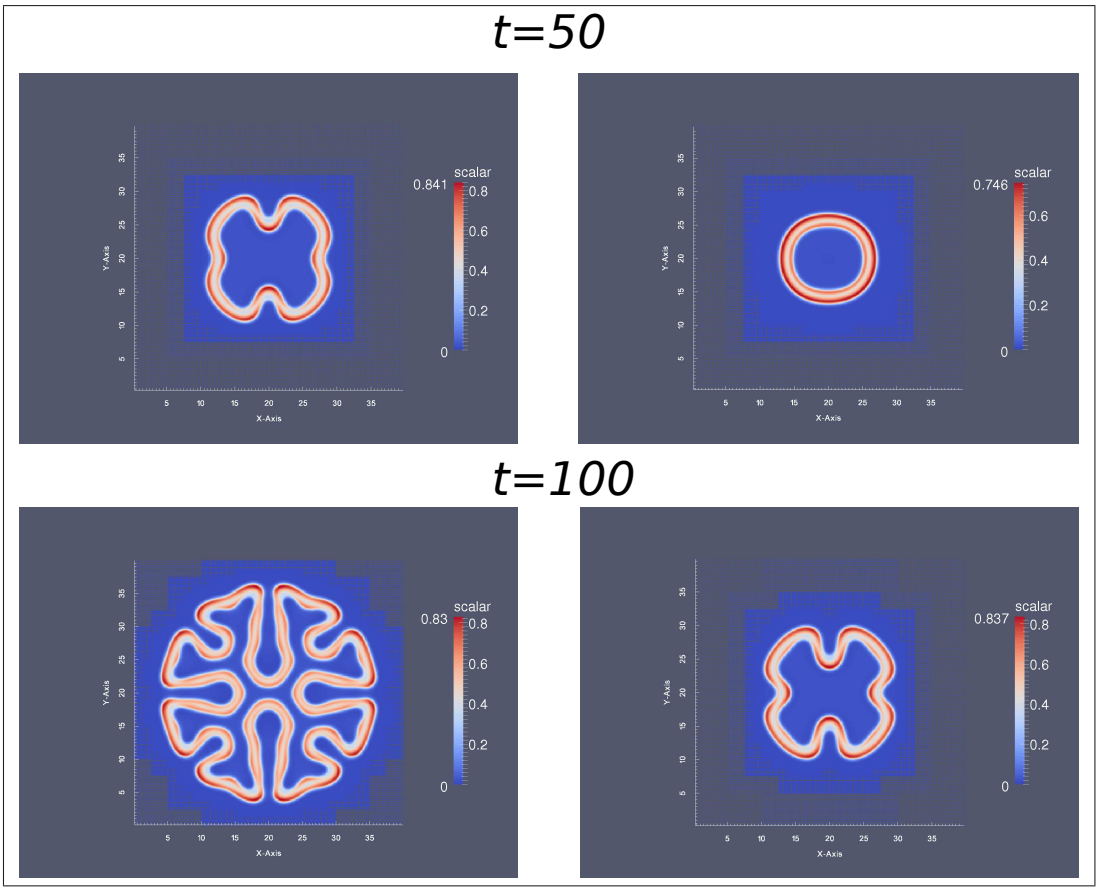

Figure 7.4: Figure shows the solutions of ϕ_V, which is the evidence that the presented model of tumour growth is sensitive to the initial conditions. The two figures in the left column are obtained (with $\gamma = 0.0$) through an initially discontinuous ϕ_T smoothed with only 20 sweeps; the two figures in the right column are the corresponding ones, using the original initial condition which has 160 smoothing sweeps.

Results in [226] only show the solutions of the phase-field variables, for completeness, here we also illustrate our computed results for the other variables. We choose the case

with $\gamma = 0.0$ and the smoother (i.e. 160 number of sweeps) initial condition for ϕ_T, and detailed results are presented in the following figures. It is worth noting that, due to the changes in the values of variables during the simulations, and for clarity, we re-scale each of the figures so the highest value of the current solution is represented by red and the lowest value of the current solution is shown as blue. The detailed results for the variable ϕ_T are shown in Figure 7.5; results for the variable μ are in Figure 7.6; solutions of the variable ϕ_D are in Figure 7.7; results for the pressure variable p are demonstrated in Figure 7.8; and finally results on the nutrient n are shown in Figure 7.9.

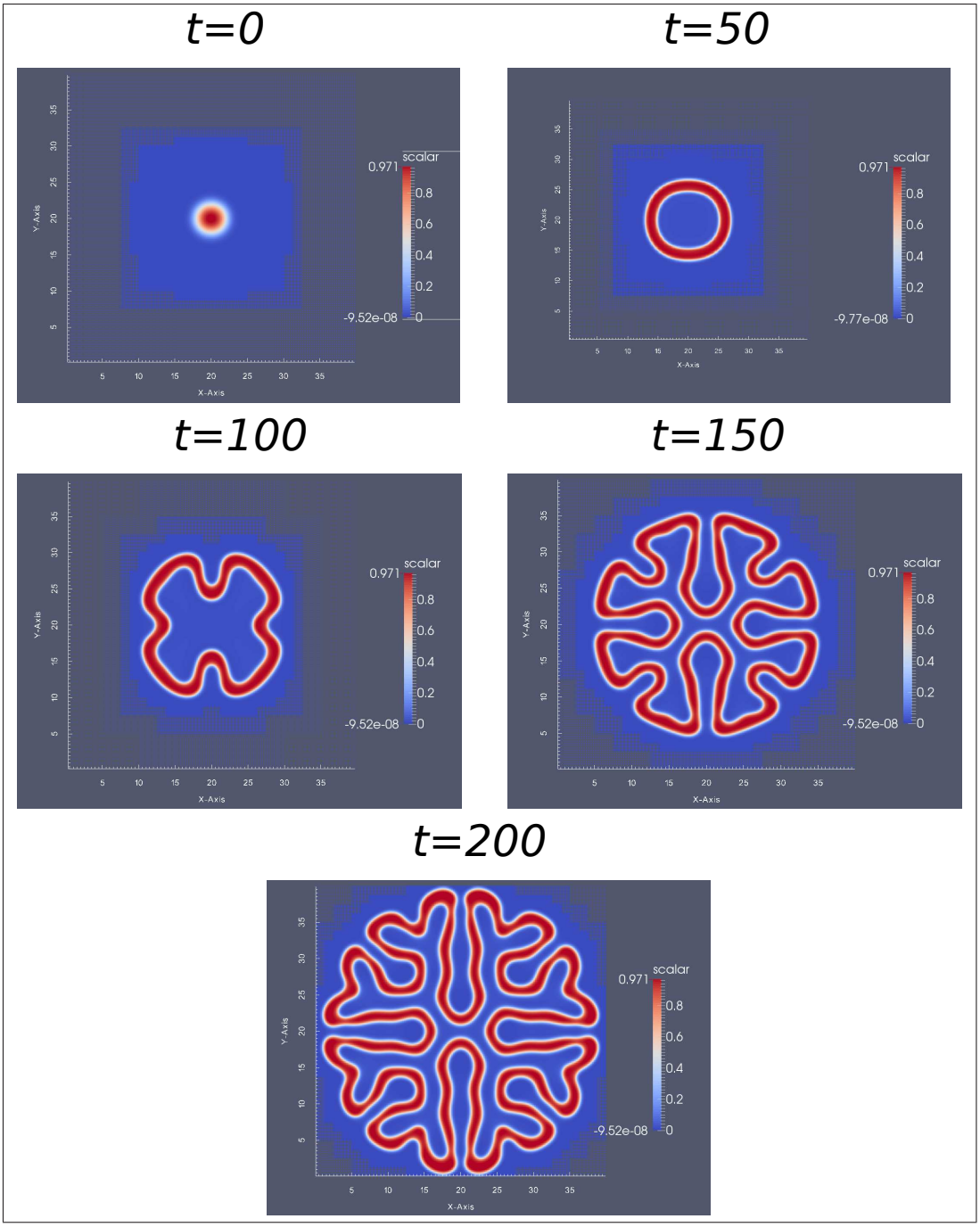

Figure 7.5: Results for variable ϕ_T from the described model of tumour growth with $\gamma = 0.0$ and the smoothest initial condition for ϕ_T.

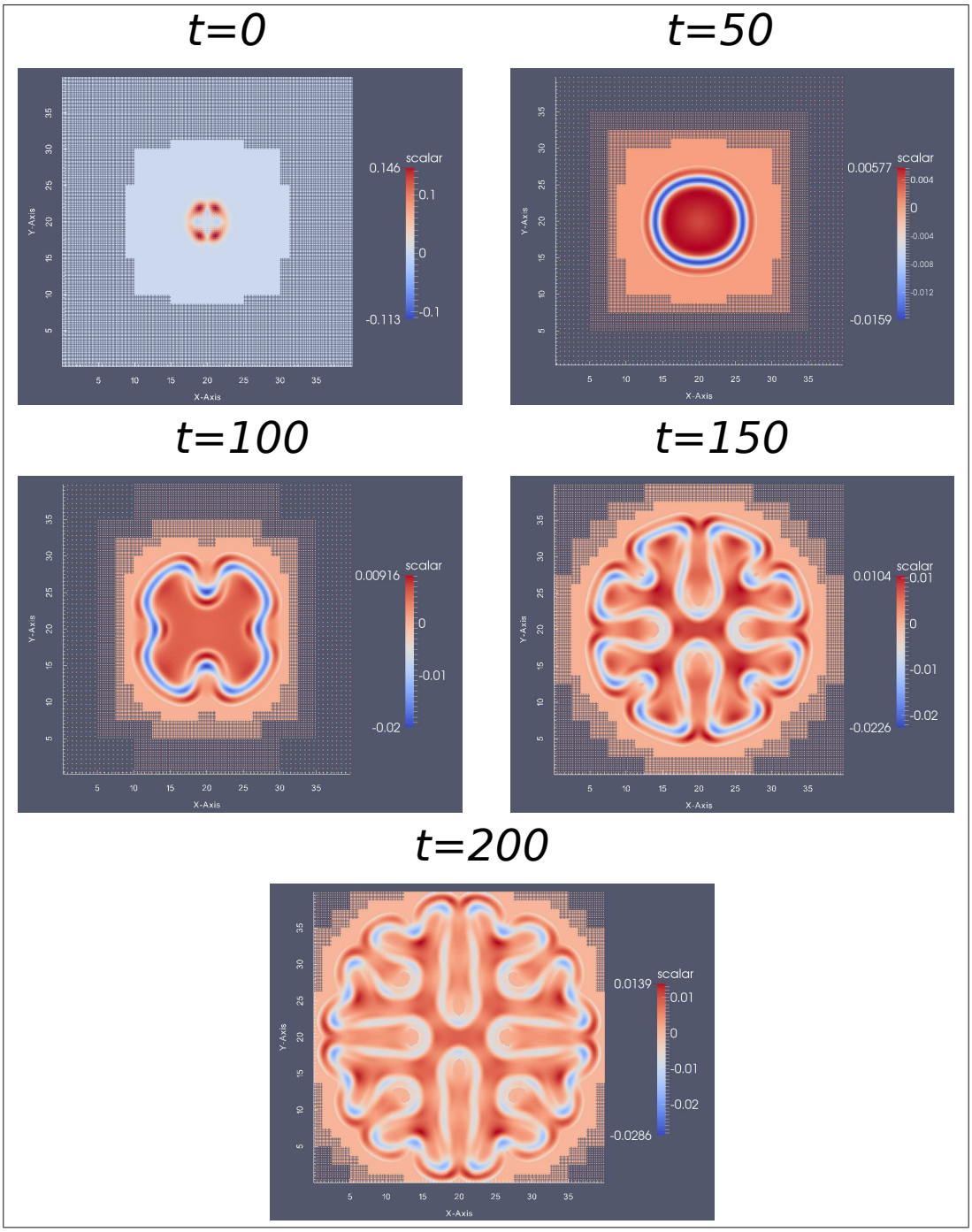

Figure 7.6: Results for variable μ from the described model of tumour growth with $\gamma = 0.0$ and the smoothest initial condition for ϕ_T.

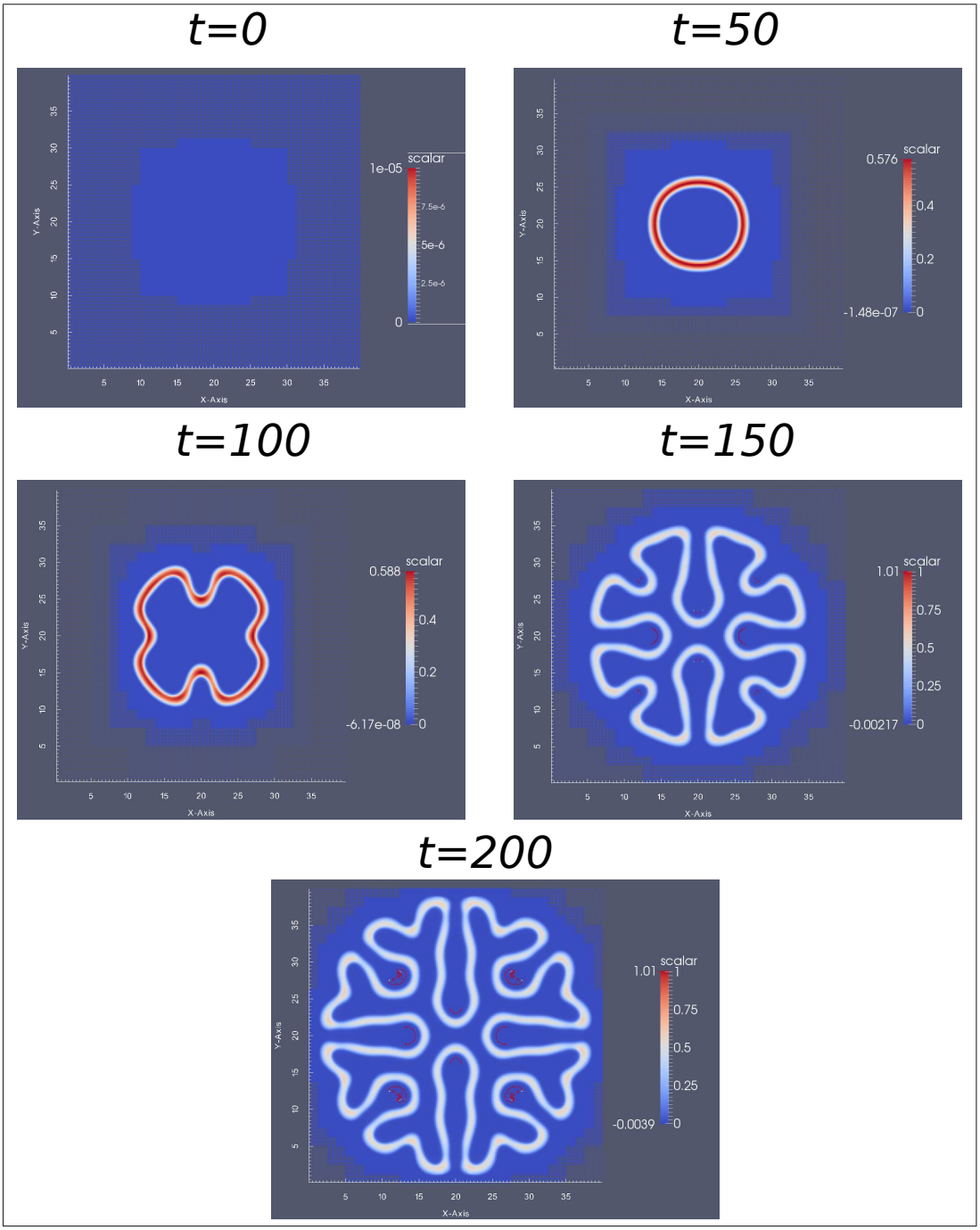

Figure 7.7: Results for variable ϕ_D from the described model of tumour growth with $\gamma = 0.0$ and the smoothest initial condition for ϕ_T.

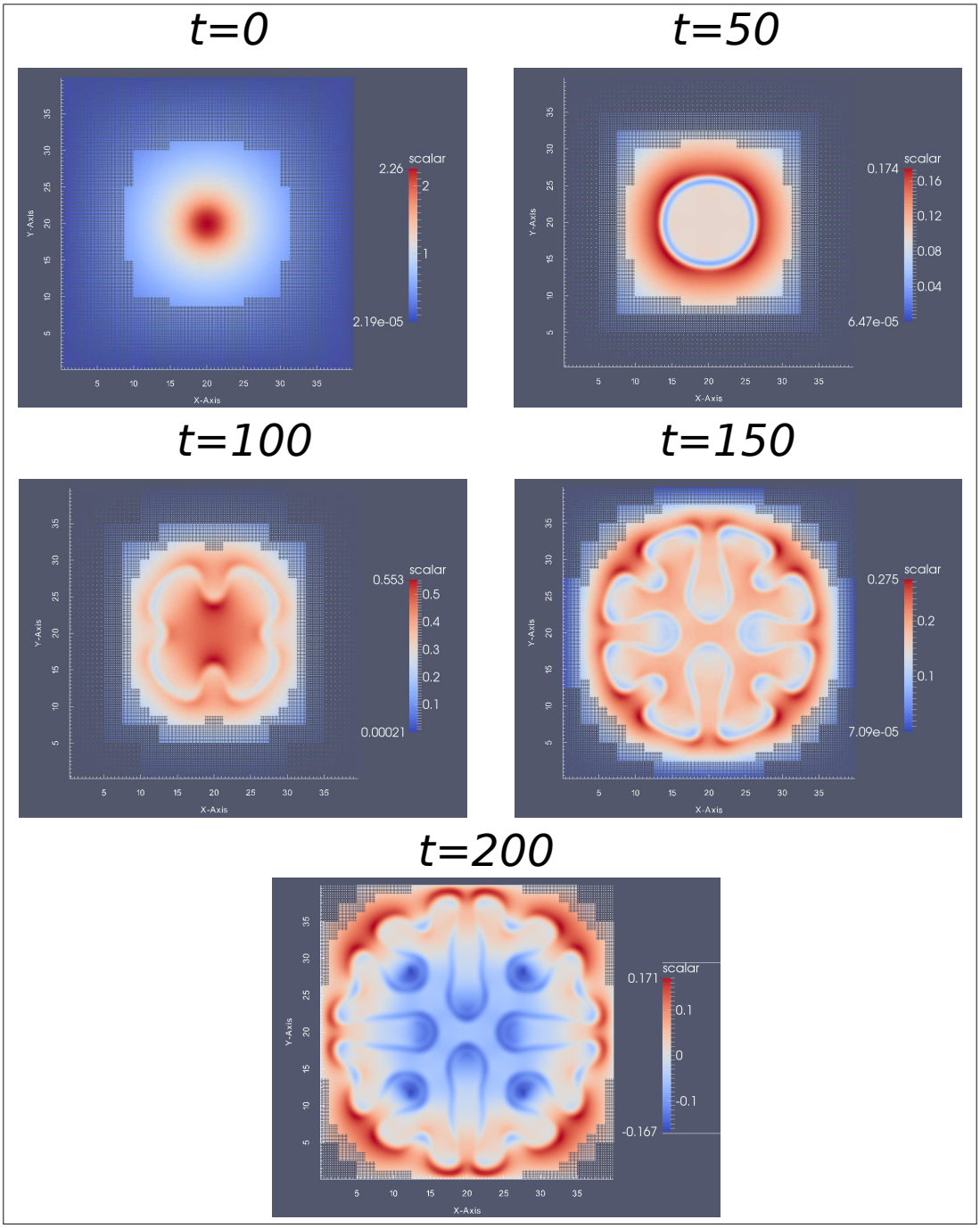

Figure 7.8: Results for variable p from the described model of tumour growth with $\gamma = 0.0$ and the smoothest initial condition for ϕ_T.

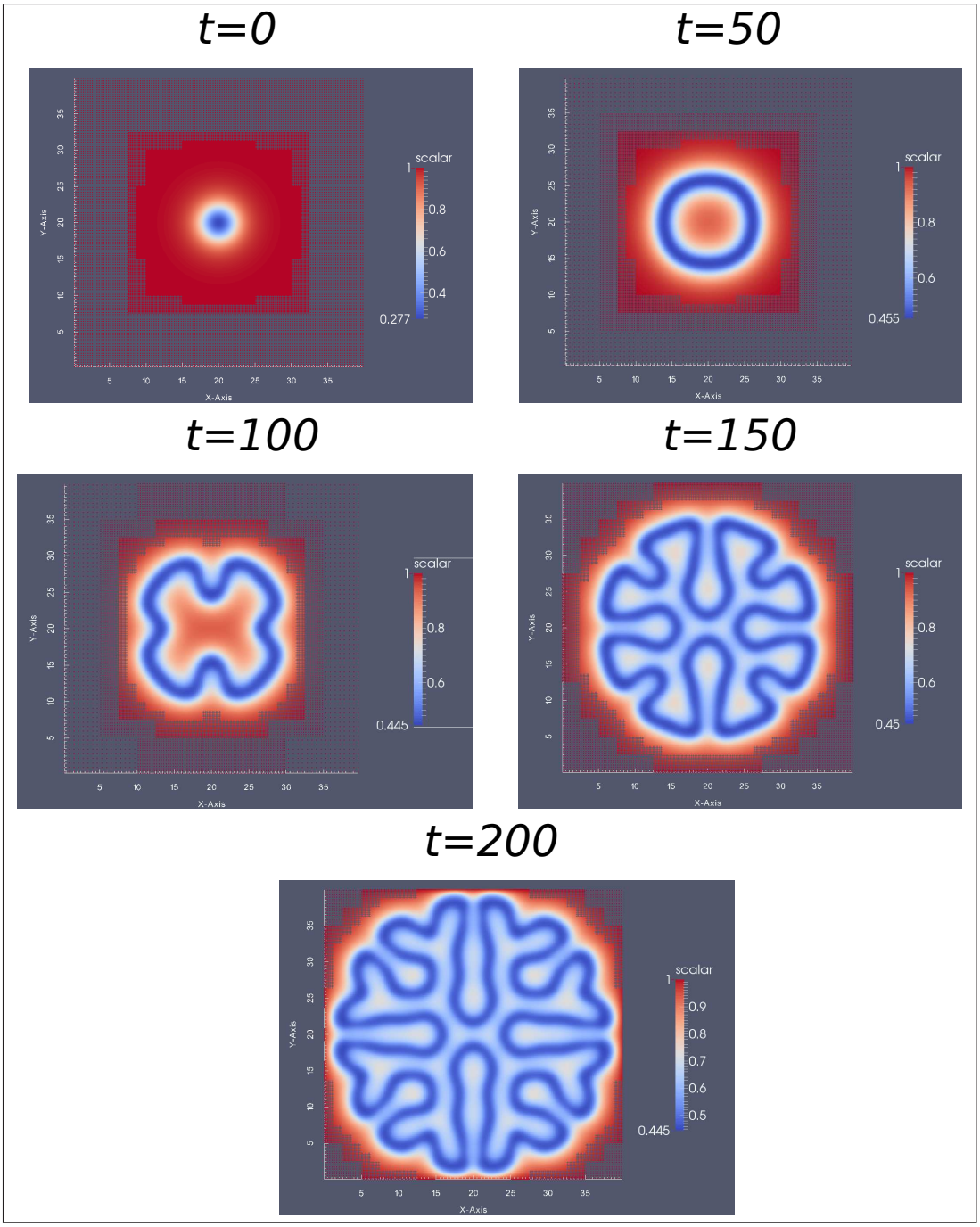

Figure 7.9: Results for variable n from the described model of tumour growth with $\gamma = 0.0$ and the smoothest initial condition for ϕ_T.

Although unable to provide the quantitative comparison against [226], we have shown qualitative similarities (at least for $\gamma = 0$ and $\gamma = 0.1$) and such a comparison is very sensitive to the choice of initial data. Furthermore, in the following section, convergence tests from the solutions of this model of tumour growth are described, which will show the expected second order convergence (unlike [226]).

7.4.2 Convergence Tests

We undertake convergence tests based upon solution restrictions as previously described in Section 5.3.3. These are based upon a sequence of 2-D solutions from the presented model of tumour growth and the computation of the norm of the difference between two solutions computed on consecutive grids. We choose four grid hierarchies, each with the same coarsest grid: 8×8. The finest mesh refinements for the four cases, if refined everywhere, have the grid resolutions: 128×128, 256×256, 512×512 and 1024×1024. They are denoted as levels 5, 6, 7 and 8, respectively. Like the results demonstrated in the previous section, here we also use the AMR with the same conservative strategy. The adaptive time-stepping is disabled for generating results for these convergence tests. Thus only the BDF2 method with a fixed time step size is used here, where δt is halved each time the maximum level is increased.

Unlike the usual way of applying the BDF1 method once for the very first time step, here we include multiple BDF1 steps to stabilise the simulation from the initial condition. Let's consider an example which seeks to use a time step size $\delta t = 2 \times 10^{-3}$ for the BDF2 method. In the beginning, we use $\delta t = 4 \times 10^{-4}$ (which is one fiftieth of 2×10^{-3}) for the BDF1 method and take 50 time steps with this δt and the BDF1 method. Then we switch to the BDF2 method and continue as normal. This use of multiple BDF1 steps is illustrated in Figure 7.10, and it allows a larger time step size to be taken throughout rest of the simulation, than would otherwise be possible. Note that when adaptive time-stepping is used this technique is not required: we simply use it here to allow fixed δt convergence tests to overcome initial restrictions on δt. This approach with using multiple BDF1 steps are used for levels 7 and 8.

The computed solutions from all grid hierarchies are taken at $T = 10$. First of all, the solutions with $\gamma = 0.0$ are considered, and we double the tumour/host tissue interface thickness (i.e. $\varepsilon = 0.2$). The wider interface makes it easier to observe an overall second order convergence rate for slightly larger h. Other parameters are as illustrated in Table 7.1. The results of the convergence tests with $\gamma = 0.0$ and $\varepsilon = 0.2$ are presented in Table 7.2. These results suggest an overall second order convergence rate is obtained for all five

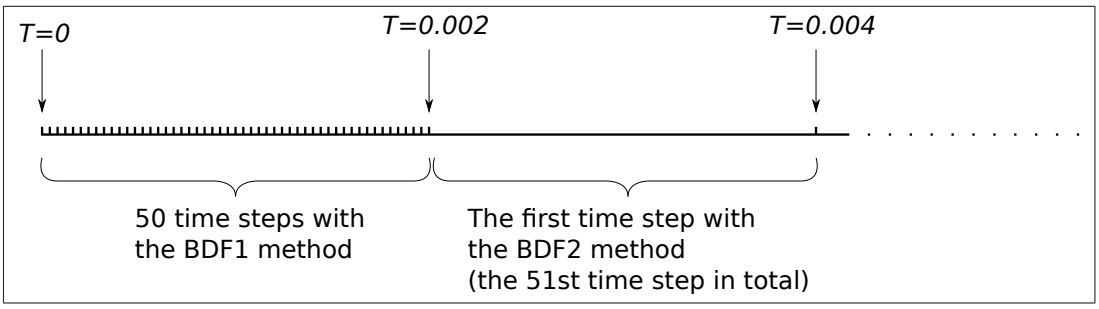

Figure 7.10: Sketch shows our approach with using multiple BDF1 methods to allow larger time step size to be taken.

variables.

For variable ϕ_T						
Levels	δt for BDF2 method	Time steps	Infinity norm	Ratio	Two norm	Ratio
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	0.258×10^{0}	-	2.529×10^{-2}	-
7	2×10^{-3}	5050	3.670×10^{-2}	7.02	4.077×10^{-3}	6.20
8	1×10^{-3}	10050	8.466×10^{-3}	4.34	9.811×10^{-4}	4.16
For variable μ						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	9.276×10^{-3}	-	1.145×10^{-3}	-
7	2×10^{-3}	5050	1.463×10^{-3}	6.34	1.834×10^{-4}	6.24
8	1×10^{-3}	10050	3.907×10^{-4}	3.74	4.582×10^{-5}	4.00
For variable ϕ_D						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	0.157×10^{0}	-	1.416×10^{-2}	-
7	2×10^{-3}	5050	2.656×10^{-2}	5.93	2.436×10^{-3}	5.81
8	1×10^{-3}	10050	6.238×10^{-3}	4.26	5.977×10^{-4}	4.08
For variable p						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	6.176×10^{-2}	-	9.961×10^{-3}	-
7	2×10^{-3}	5050	6.521×10^{-3}	9.45	1.226×10^{-3}	8.12
8	1×10^{-3}	10050	1.452×10^{-3}	4.49	2.891×10^{-4}	4.24
For variable n						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	6.378×10^{-2}	-	9.145×10^{-3}	-
7	2×10^{-3}	5050	9.340×10^{-3}	6.83	1.435×10^{-3}	6.37
8	1×10^{-3}	10050	2.157×10^{-3}	4.33	3.420×10^{-4}	4.20

Table 7.2: Results show the differences in consecutive solutions measured in the stated norm, followed by the ratio of consecutive differences from the described model of tumour growth with $\gamma = 0.0$ and $\varepsilon = 0.2$.

From Table 7.2, it may be seen that the infinity norms of the estimated errors for the phase-field variables (i.e. ϕ_T and ϕ_D) are relatively large. This is caused by the nature of the solution. If the two solutions are shifted slightly, the infinity norm of their difference may become very large, even though the two norm of this difference may be relatively small. This is illustrated in Figure 7.11.

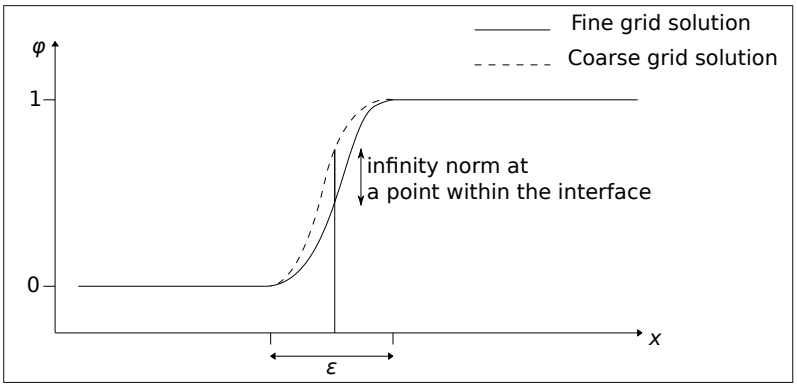

Figure 7.11: Sketch shows a possible situation where the infinity norm of the difference between two estimated solutions becomes relatively large from a slight shift of one solution relative to the other.

Having seen second order convergence rate with interface thickness $\varepsilon = 0.2$, now we apply our convergence tests to the solutions generated from using $\varepsilon = 0.1$ and $\gamma = 0.0$. Other settings are the same as shown in Table 7.2. The results of these convergence tests are presented in Table 7.3. These again show an overall second order convergence rate from all five variables is obtained, whereas in [226] Wise et al. can only obtain an overall first order convergence (despite their expectation of second order for this case with $\gamma = 0$).

For variable ϕ_T						
Levels	δt for BDF2 method	Time steps	Infinity norm	Ratio	Two norm	Ratio
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	0.719×10^{0}	-	4.969×10^{-2}	-
7	2×10^{-3}	5050	6.228×10^{-2}	11.5	4.876×10^{-3}	10.2
8	1×10^{-3}	10050	1.249×10^{-2}	4.99	1.142×10^{-3}	4.27
For variable μ						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	1.367×10^{-2}	-	1.279×10^{-3}	-
7	2×10^{-3}	5050	1.205×10^{-3}	11.3	1.103×10^{-4}	11.6
8	1×10^{-3}	10050	3.241×10^{-4}	3.72	2.888×10^{-5}	3.82
For variable ϕ_D						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	0.245×10^{0}	-	1.923×10^{-2}	-
7	2×10^{-3}	5050	1.663×10^{-2}	14.7	1.976×10^{-3}	14.7
8	1×10^{-3}	10050	4.303×10^{-3}	3.86	4.837×10^{-4}	4.08
For variable p						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	4.918×10^{-2}	-	1.203×10^{-2}	-
7	2×10^{-3}	5050	5.940×10^{-3}	8.28	1.726×10^{-3}	6.97
8	1×10^{-3}	10050	1.469×10^{-3}	4.04	4.487×10^{-4}	3.85
For variable n						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	0.102×10^{-0}	-	1.012×10^{-2}	-
7	2×10^{-3}	5050	7.385×10^{-3}	13.8	1.003×10^{-3}	10.1
8	1×10^{-3}	10050	1.508×10^{-3}	4.90	2.365×10^{-4}	4.24

Table 7.3: Results show the differences in consecutive solutions measured in the stated norm, followed by the ratio of consecutive differences from the described model of tumour growth with $\gamma = 0.0$ and $\varepsilon = 0.1$.

In [226], Wise et al. go on to state that "first order result should be expected because of the treatment of the excess surface adhesion term (i.e. when $\gamma \neq 0$)". Having taken their advice to smooth out the Heaviside function and used a second order approximation to all terms (combined with using the penalty terms in the phase-field equations), we now show convergence tests that are generated with $\gamma = 0.1$ and $\varepsilon = 0.2$. We also use an extra level of refinement, which, if refined everywhere, corresponds to the finest grid resolution of 2048×2048. We denote this as level 9. The results of these convergence tests are presented in Table 7.4. These again appear to show second order convergence - the only deviation from this being the infinity norm of consecutive ϕ_D solutions. As noted above, this is not a particularly appropriate norm and convergence in the two norm certainly shows second order.

For variable ϕ_T						
Levels	δt for BDF2 method	Time steps	Infinity norm	Ratio	Two norm	Ratio
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	9.118×10^{-2}	-	7.836×10^{-3}	-
7	2×10^{-3}	5050	1.322×10^{-2}	6.90	1.131×10^{-3}	6.93
8	1×10^{-3}	10050	2.579×10^{-3}	5.13	2.367×10^{-4}	4.78
9	5×10^{-4}	20050	5.691×10^{-4}	4.53	5.833×10^{-5}	4.06
For variable μ						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	2.627×10^{-3}	-	3.904×10^{-4}	-
7	2×10^{-3}	5050	3.890×10^{-4}	6.75	6.308×10^{-5}	6.19
8	1×10^{-3}	10050	4.252×10^{-4}	0.91	2.099×10^{-5}	3.01
9	5×10^{-4}	20050	1.236×10^{-4}	3.44	5.265×10^{-6}	3.99
For variable ϕ_D						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	9.115×10^{-2}	-	4.941×10^{-3}	-
7	2×10^{-3}	5050	0.108×10^{0}	0.84	2.110×10^{-3}	2.34
8	1×10^{-3}	10050	9.688×10^{-2}	1.12	1.389×10^{-3}	1.52
9	5×10^{-4}	20050	5.518×10^{-2}	1.76	4.356×10^{-4}	3.10
For variable p						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	1.850×10^{-2}	-	3.812×10^{-3}	-
7	2×10^{-3}	5050	3.956×10^{-3}	4.68	8.480×10^{-4}	4.50
8	1×10^{-3}	10050	9.043×10^{-4}	4.38	2.047×10^{-4}	4.14
9	5×10^{-4}	20050	2.136×10^{-4}	4.24	4.940×10^{-5}	4.14
For variable n						
5	8×10^{-3}	1250	-	-	-	-
6	4×10^{-3}	2500	2.221×10^{-0}	-	2.863×10^{-3}	-
7	2×10^{-3}	5050	3.290×10^{-3}	6.75	4.317×10^{-4}	6.63
8	1×10^{-3}	10050	6.319×10^{-3}	5.21	8.874×10^{-5}	4.86
9	5×10^{-4}	20050	1.661×10^{-2}	3.49	2.188×10^{-5}	4.06

Table 7.4: Results show the differences in consecutive solutions measured in the stated norm, followed by the ratio of consecutive differences from the described model of tumour growth with $\gamma = 0.1$ and $\varepsilon = 0.2$.

Through the presented results from our convergence tests, we conclude that using the finite difference five-point stencil in 2-D, the BDF2 method, our multigrid solver with AMR, the smoothed Heaviside function and the penalty terms, we are able to obtain an overall second order convergence rate for the described model of tumour growth for the first time. In the following section, results of 3-D simulations conducted on the model of tumour growth are presented.

7.4.3 Simulations in 3-D

The 3-D system of the model of tumour growth as presented in Section 7.3.2 is solved using our 3-D multigrid solver. The results are illustrated in this section. First of all, we conduct our simulation using the parameters shown in Table 7.1 with $\gamma = 0.0$. We use the finest grid as suggested by Wise et al. in [226], and if it is refined everywhere, the resolution is $256 \times 256 \times 256$. For the coarsest grid, we choose $16 \times 16 \times 16$ which is a grid coarser than the one used in [226]. Therefore, we denote this grid hierarchy as level 5.

In Figure 7.12, we show the shapes of the tumour and the growing process, by displaying the solutions of ϕ_T where the values of the variable are in a range from 0.5 to 1.0. This includes the interface of the healthy and tumorous cells as well as the tumour itself. The described initial condition in Equation (7.24) is for 2-D simulations and there is no description for the 3-D initial condition used in [226]. Therefore, we impose a 3-D initial condition with three ellipses, and the $\phi_T = 0.5$ isosurface of these ellipses are defined as

$$\begin{aligned} \frac{(x-19)^2}{1.1} + (y-19)^2 + (z-19)^2 &\leq 2^2, \\ (x-20)^2 + \frac{(y-20)^2}{1.1} + (z-20)^2 &\leq 2^2, \\ (x-21)^2 + (y-19)^2 + \frac{(z-19)^2}{1.1} &\leq 2^2. \end{aligned} \tag{7.26}$$

When $\phi_T(x,y,z,t=0)$ is in the range of any of these three ellipses, $\phi_T(x,y,z,t=0) = 1$, otherwise $\phi_T(x,y,z,t=0) = 0$. We then use 200 sweeps of the 3-D version of the Jacobi iteration shown in Equation (7.25) to obtain a smooth initial condition. In this figure, solutions at $t = 0$, 50 and 100 are presented. Note that the colour map is rescaled according to the presented solution on each subsequent feature in the figure, so the lowest, presented values in the solutions have the colour of blue, and the highest values in the presented solutions have the colour of red. This colouring approach is applied to all figures (and their subsequent features) illustrated in this section. It is worth noting that these are

scatter plots by the point-wise data values.

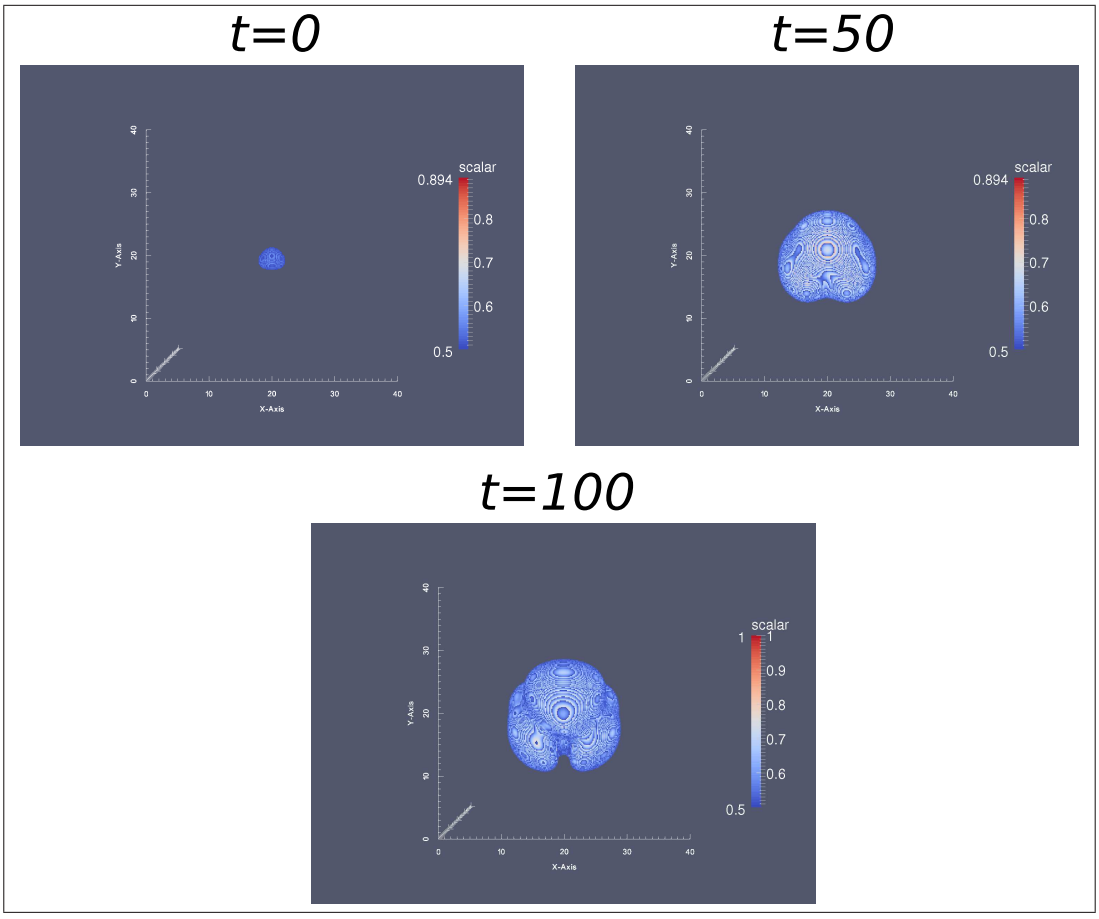

Figure 7.12: 3-D results for variable ϕ_T with $\gamma = 0.0$. In this figure, the solutions of ϕ_T which have values in a range from 0.5 to 1.0 are displayed. Top-left feature is the initial condition of ϕ_T, top-right feature the shape of tumour at $t = 50$ and the bottom feature is the shape at $t = 100$.

We now undertake further investigation of the solution ϕ_T at $t = 100$ shown in Figure 7.12. Three figures of cross-sections in x, y and z planes respectively are illustrated in Figure 7.13. These cross-sections are undertaken at the Cartesian coordinates $x = 20$, $y = 20$ and $z = 20$ respectively.

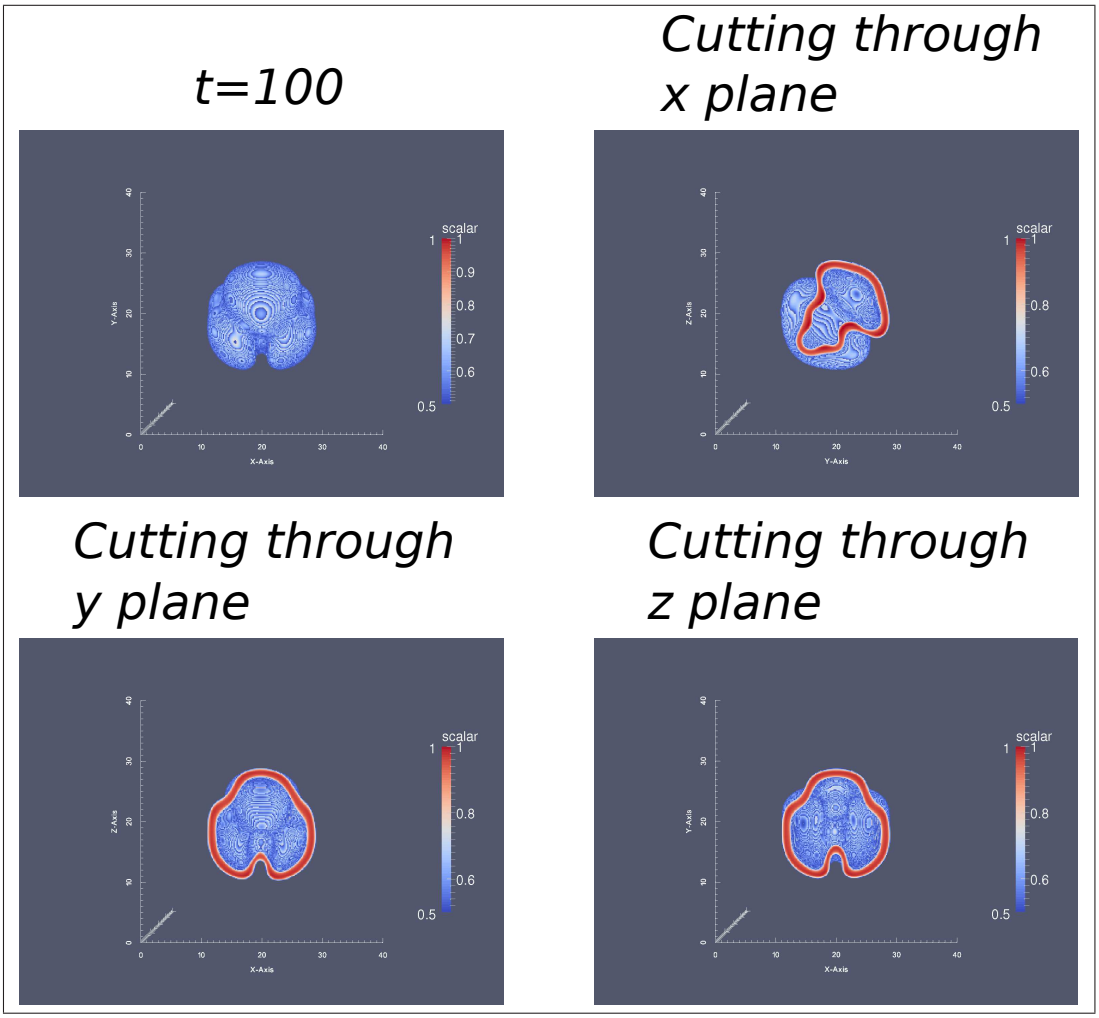

Figure 7.13: 3-D results and images of cross-sections for variable ϕ_T with $\gamma = 0.0$. In this figure, the solution of ϕ_T at $t = 100$ is displayed again in the top-left feature (it is previously presented in Figure 7.12); the cross-section through the plane $x = 20$ is presented in the top-right feature; the cross-section through the plane $y = 20$ is in the bottom-left feature; and finally the cross-section through the plane $z = 20$ is displayed in the bottom-right feature.

We intend to investigate further the solutions from using different input parameters. However, 3-D simulations with level 5 grid hierarchy are very computationally demanding, especially for these post $t = 100$. Due to the nature of the problem, we may have to refine the mesh almost everywhere in order to capture the growing tumour. Thus, we

decide to conduct further 3-D simulations on a grid hierarchy with 4 levels, for which the finest grid, if refined everywhere, has a grid resolution: $128 \times 128 \times 128$. First of all, we need to verify that using a coarser grid hierarchy has limited effects to the outcome of the simulation. Therefore, the simulation presented in Figure 7.12 is repeated with this coarser grid hierarchy and we compare both solutions at $t = 100$ in Figure 7.14. From the results presented in this figure, we notice that the solution from the level 4 simulation is slightly advanced in growth compared to the one from the level 5, however, this feature has also been observed with 2-D simulations. Nevertheless, solutions from using level 4 are relatively close to these produced from level 5, thus we decide that these results are acceptable.

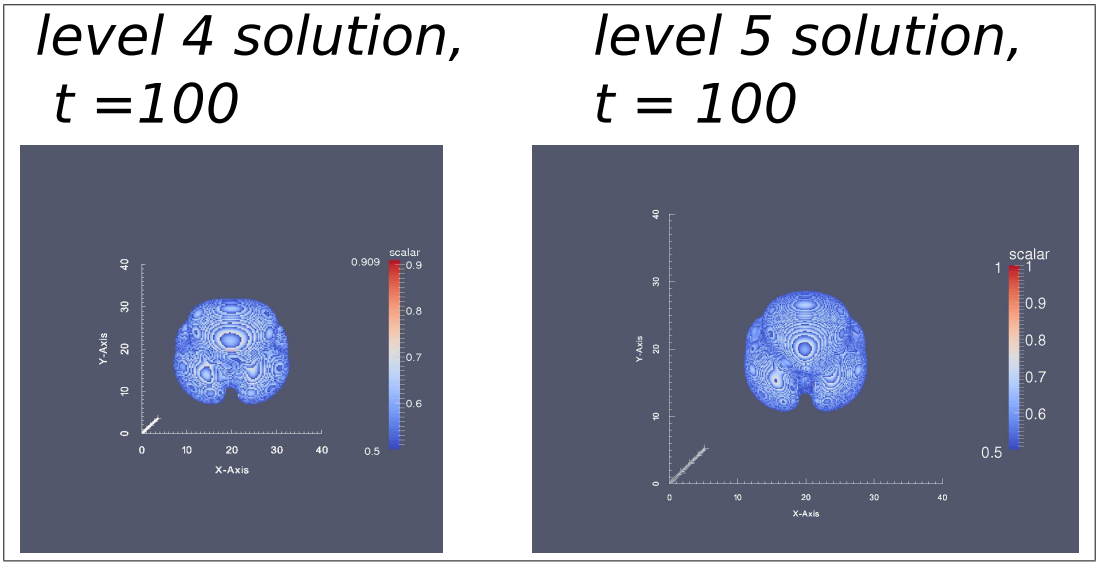

Figure 7.14: 3-D results variable ϕ_T with $\gamma = 0.0$ at $t = 100$ from two different grid hierarchies: the one on the left-hand side is from a grid hierarchy with 4 levels; the one on the right-hand side is from the described level 5 grid hierarchy which has been presented in Figure 7.12.

With a coarser grid hierarchy, we present the solutions of ϕ_T at $t = 50$, 100, 150 and 200 in Figure 7.15 (the solution at $t = 100$ is already presented on the left-hand side of Figure 7.14).

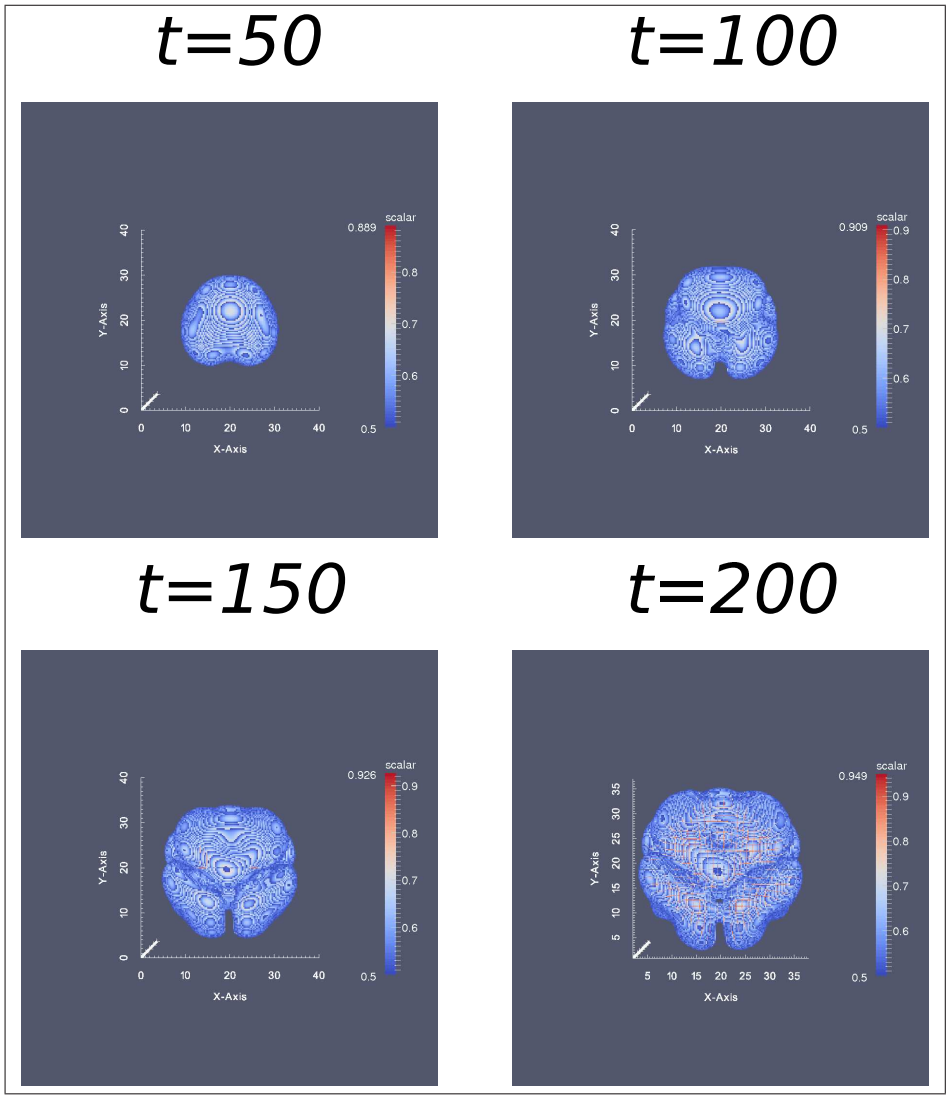

Figure 7.15: 3-D results of variable ϕ_T with $\gamma = 0.0$ at $t = 50$ (at top left), 100 (at top right), 150 (at bottom left) and 200 (at bottom right) from the level 4 grid hierarchy. The solution at $t = 100$ is already presented in Figure 7.14.

We provide further investigations on the solution of ϕ_T at $t = 200$ by presenting cross-sections with respect to each axis at its middle point. This is illustrated in Figure 7.16.

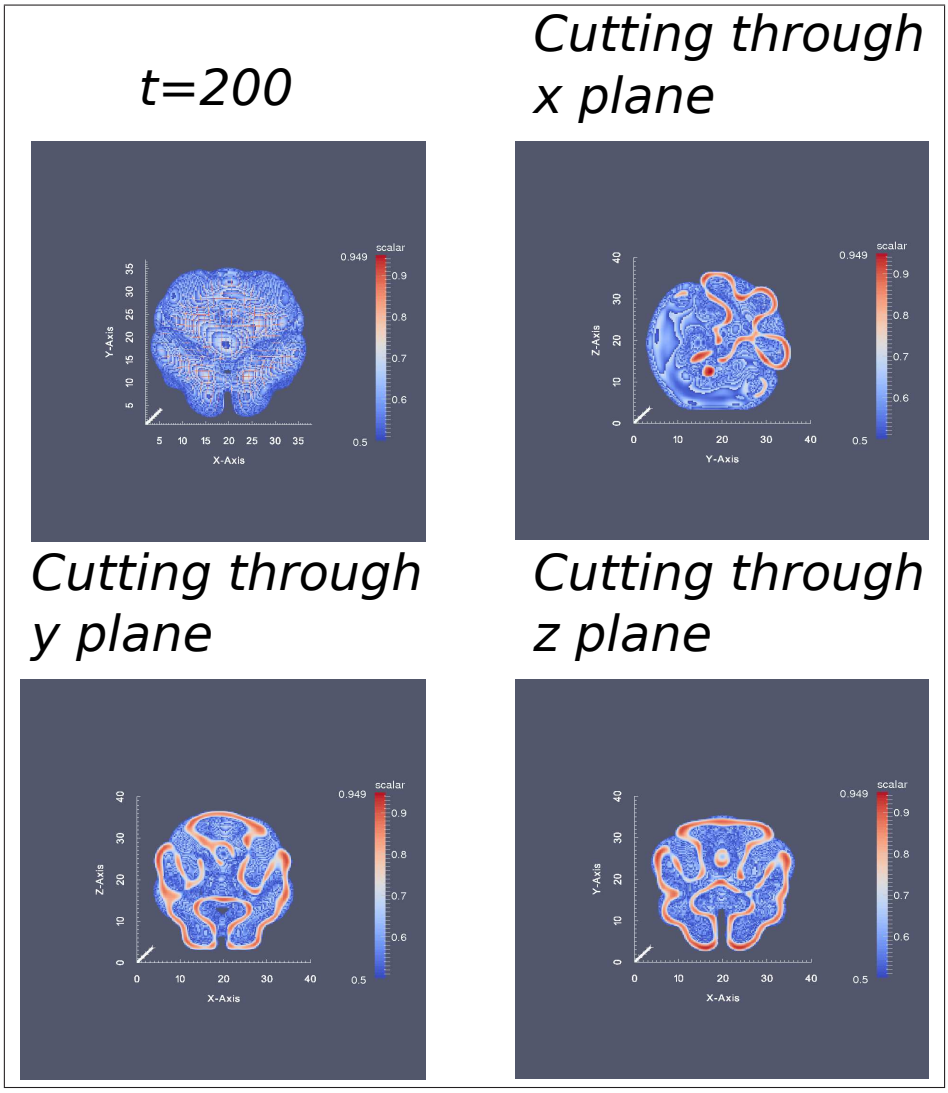

Figure 7.16: 3-D results and images of cross-sections for variable ϕ_T with $\gamma = 0.0$ from level 4. In this figure, the solution of ϕ_T at $t = 200$ is displayed again in the top-left feature (it is previously presented in Figure 7.15); the cross-section through the plane $x = 20$ is presented in the top-right feature; the cross-section through the plane $y = 20$ is in the bottom-left feature; and finally the cross-section through the plane $z = 20$ is displayed in the bottom-right feature.

For completeness, we present the solutions at $t = 50$, 100, 150 and 200 for all the other four variables (i.e. μ, ϕ_D, p and n). It is worth noting that the variable μ, p and n are most conveniently presented as cross-sections which cut through the plane $x = 20$. In Figure 7.17, solutions of μ are illustrated; in Figure 7.20, solutions of the pressure variable p are displayed; and in Figure 7.21, solutions of nutrient n are presented. On the other hand, we present the solutions of ϕ_D in Figure 7.18, and it is followed by a display of cross-sections features in Figure 7.19. From the 2-D results presented in the previous section, we discover that the values of the solutions of ϕ_D (especially around the interface) may not be in the range of 0 to 1, thus in these figures of ϕ_D, we employ a software tool (i.e. Paraview) to determine the middle value of the current solution and use it as the threshold (i.e. any values lower than this threshold are made invisible).

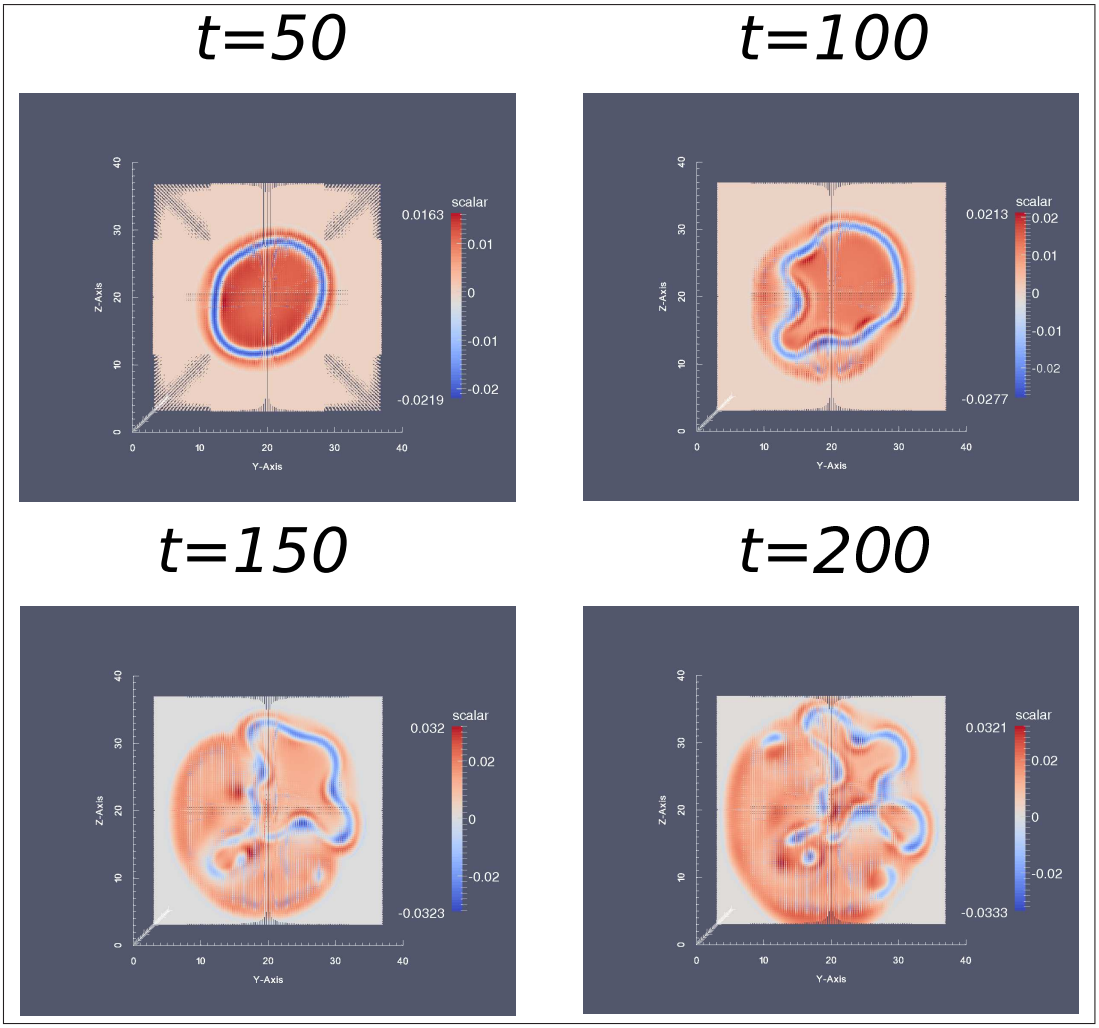

Figure 7.17: 3-D results of variable μ with $\gamma = 0.0$ at $t = 50$ (at top left), 100 (at top right), 150 (at bottom left) and 200 (at bottom right) from the level 4 grid hierarchy. The shown features are cross-sections through the plane $x = 20$.

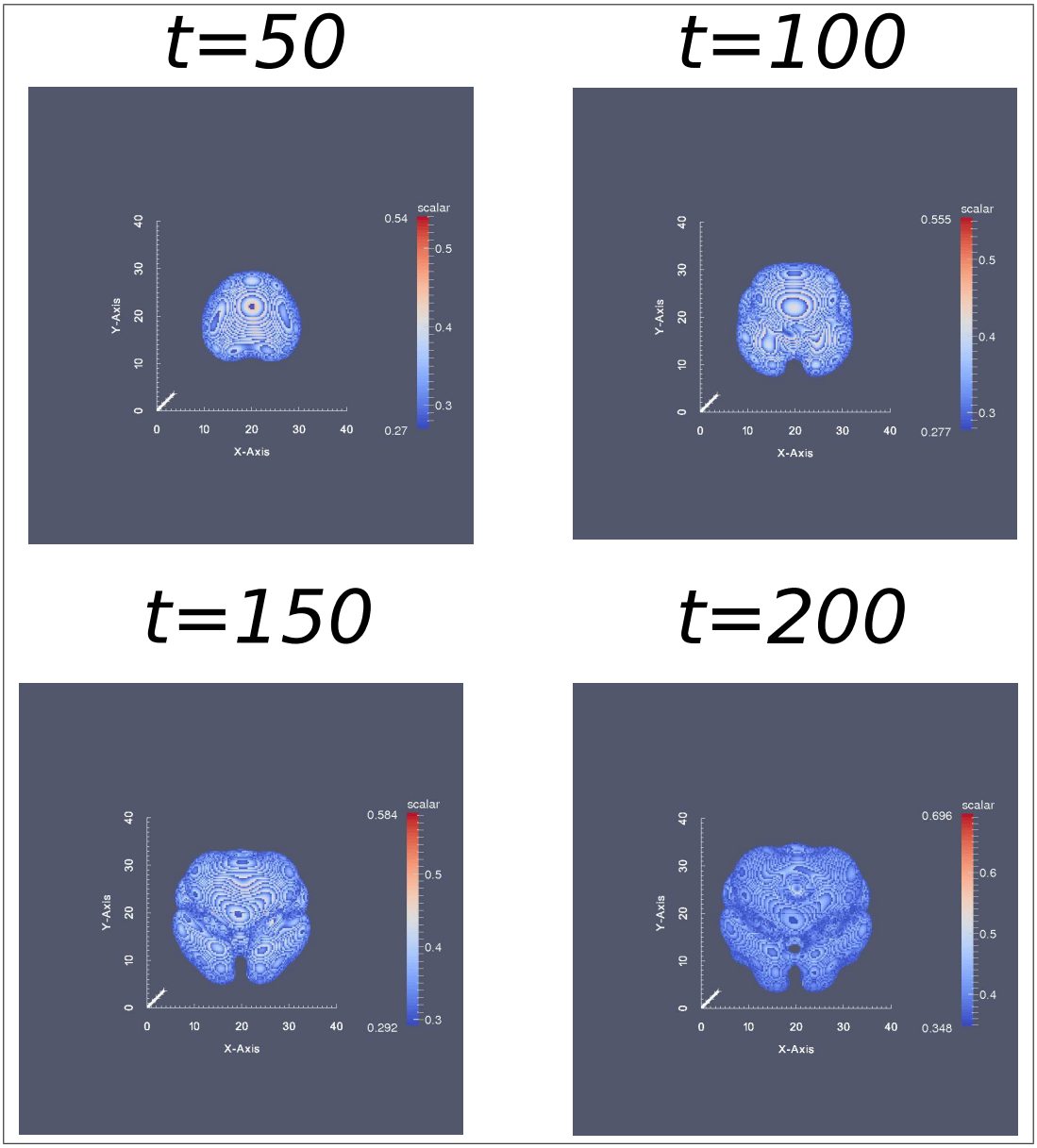

Figure 7.18: 3-D results of variable ϕ_D with $\gamma = 0.0$ at $t = 50$ (at top left), 100 (at top right), 150 (at bottom left) and 200 (at bottom right) from the level 4 grid hierarchy. The choice of the lowest value shown in these features is the middle value of the current solution.

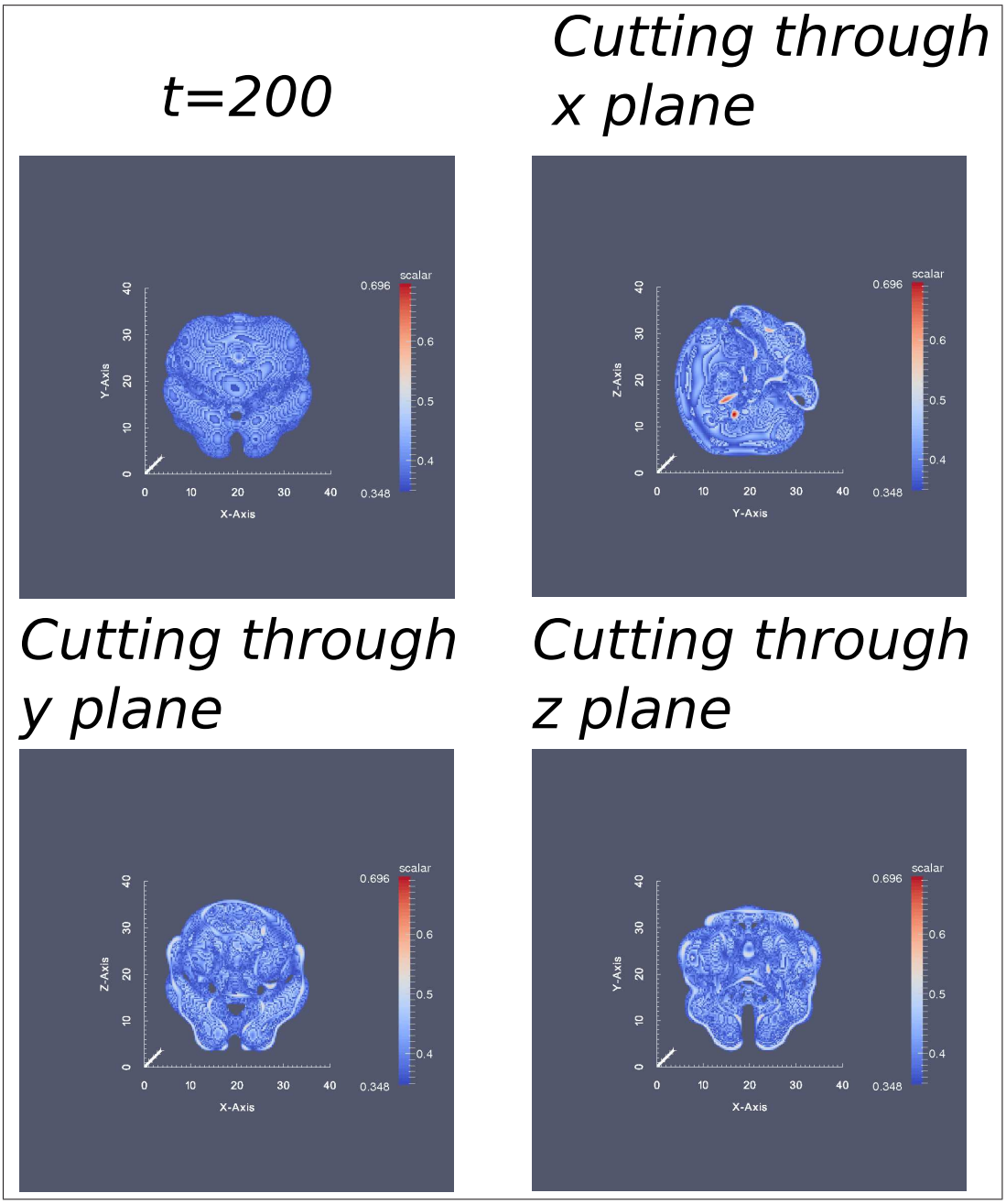

Figure 7.19: 3-D results and images of cross-sections for variable ϕ_D with $\gamma = 0.0$ from level 4. In this figure, the solution of ϕ_D at $t = 200$ is displayed again in the top-left feature (it is previously presented in Figure 7.18); the cross-section through the plane $x = 20$ is presented in the top-right feature; the cross-section through the plane $y = 20$ is in the bottom-left feature; and finally the cross-section through the plane $z = 20$ is displayed in the bottom-right feature.

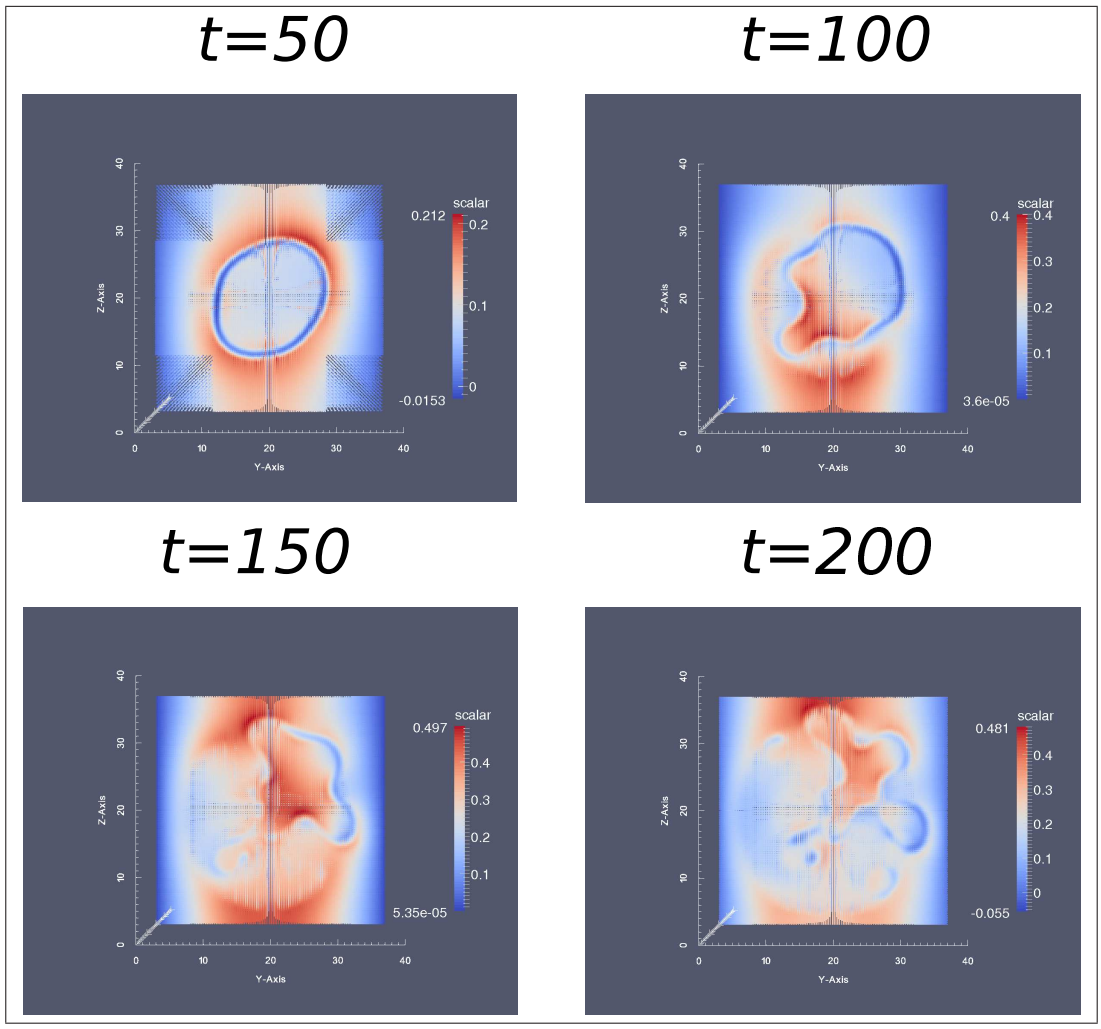

Figure 7.20: 3-D results of variable p with $\gamma = 0.0$ at $t = 50$ (at top left), 100 (at top right), 150 (at bottom left) and 200 (at bottom right) from the level 4 grid hierarchy. The choice of the lowest value shown in these features is the middle value of the current solution.

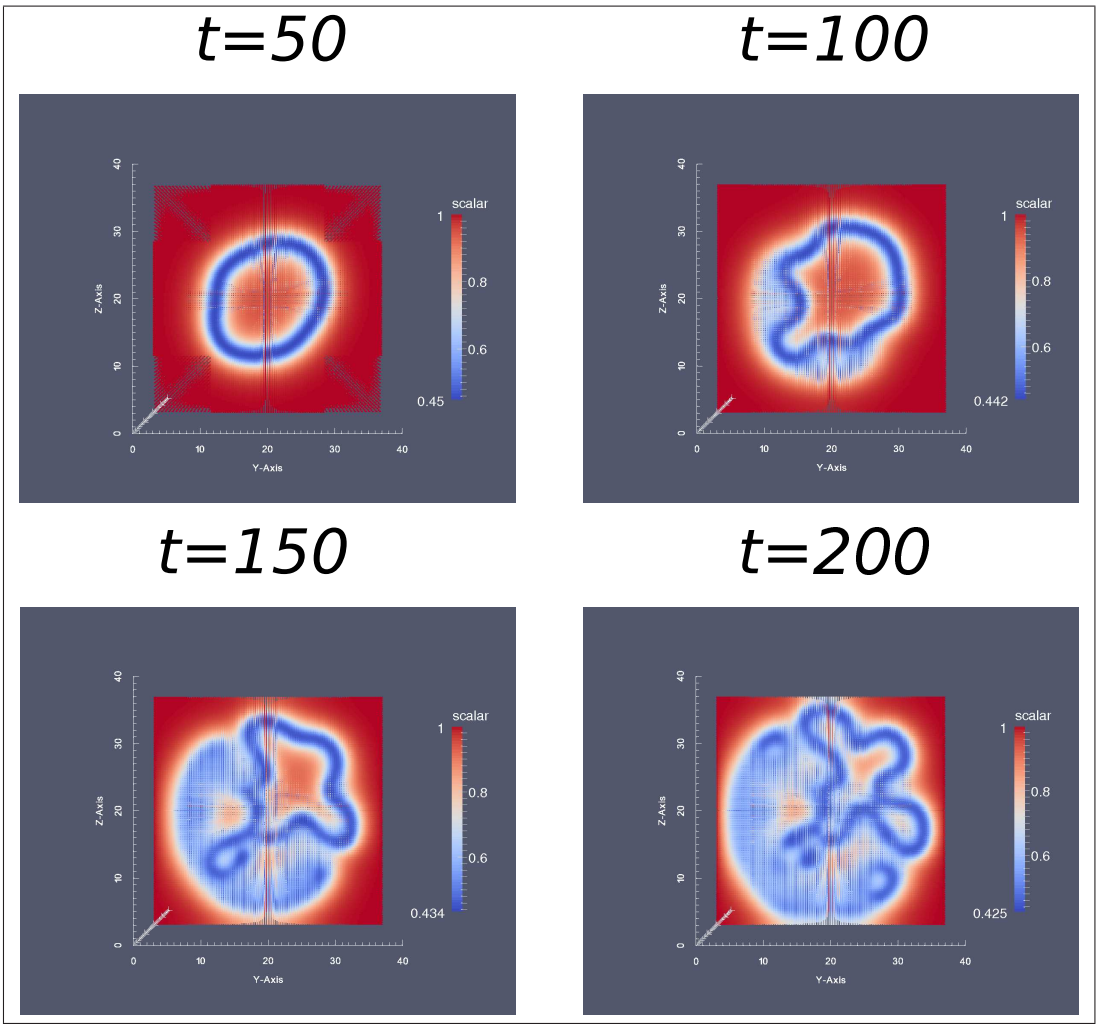

Figure 7.21: 3-D results of variable n with $\gamma = 0.0$ at $t = 50$ (at top left), 100 (at top right), 150 (at bottom left) and 200 (at bottom right) from the level 4 grid hierarchy. The choice of the lowest value shown in these features is the middle value of the current solution.

Having presented solutions and features in cross-section from simulations using the parameters shown in Table 7.1 with $\gamma = 0$, here we also present additional results with different input parameters. In order to illustrate the influences of these modified parameters, we only alter one parameter per test. It is worth noting for the following 3-D results, only solutions of ϕ_T are presented. Two approaches are taken for each test, firstly we present solutions at $t = 50$, 100, 150 and 200 (or as close as possible); then cross-sections with respect to each axis at the final solution are illustrated. These results should be compared to the base case illustrated in Figures 7.15 and 7.16.

The first test is to increase D_H which is the nutrient diffusivity in the healthy tissue to 3 (up from 1). The results are presented in Figures 7.22 and 7.23. The results illustrated in these figures suggest by increasing D_H, the speed of tumour growth is also increased. The interface of healthy and tumour cells at $t = 200$ has just touched the (artificial) domain boundary.

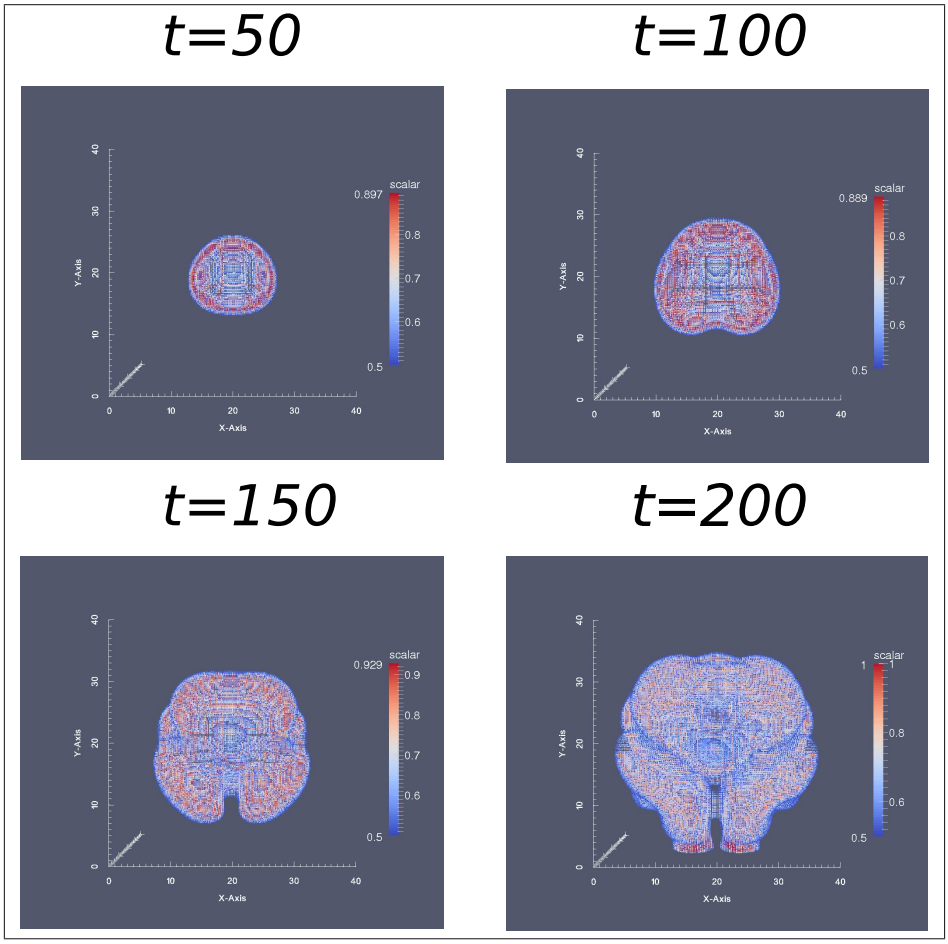

Figure 7.22: 3-D results of variable ϕ_T with $\gamma = 0.0$ and $D_H = 3$ at $t = 50$ (at top left), 100 (at top right), 150 (at bottom left) and 200 (at bottom right) from the level 4 grid hierarchy.

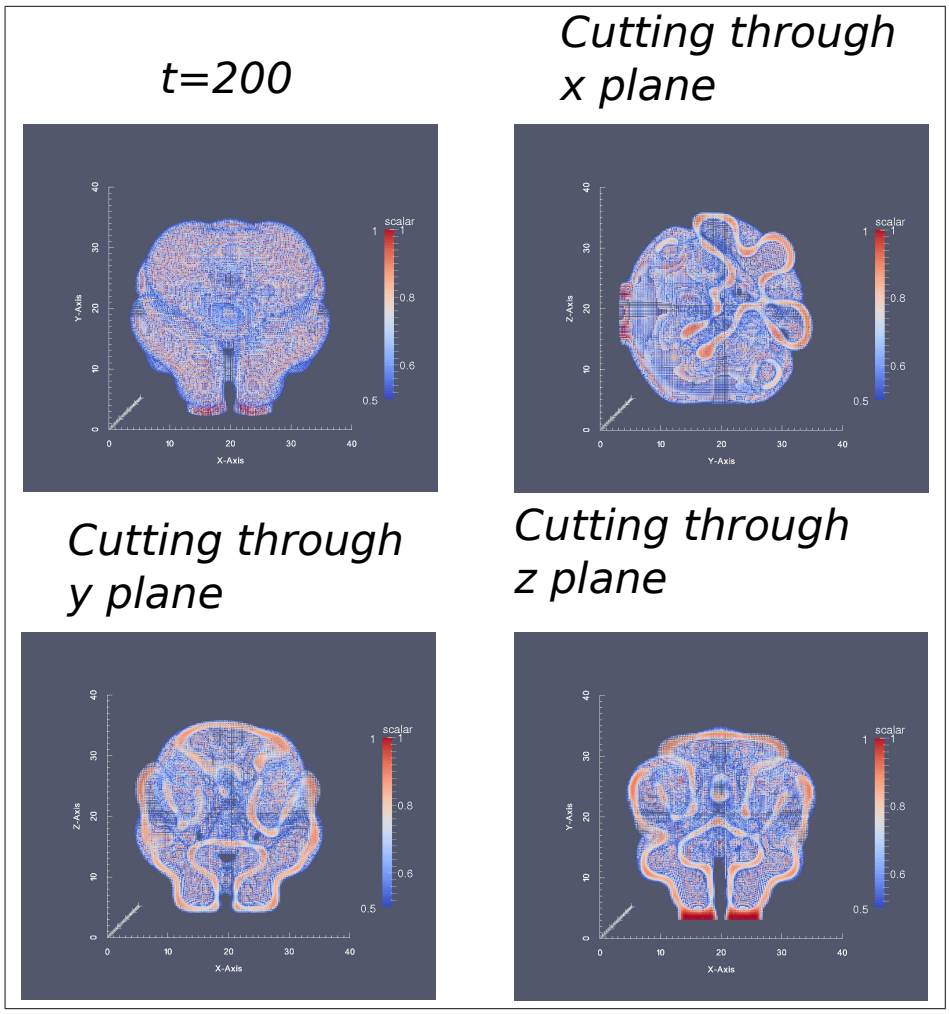

Figure 7.23: 3-D results and images of cross-sections for variable ϕ_T with $\gamma = 0.0$ and $D_H = 3$ from level 4. In this figure, the solution of ϕ_T at $t = 200$ is displayed again in the top-left feature (it is previously presented in Figure 7.22); the cross-section through the plane $x = 20$ is presented in the top-right feature; the cross-section through the plane $y = 20$ is in the bottom-left feature; and finally the cross-section through the plane $z = 20$ is displayed in the bottom-right feature.

The second test is to increase λ_L, which is the birth rate of tumour cells, to 3 (up from 1). The results are presented in Figures 7.24 and 7.25. These results suggest increasing λ_L from 1 to 3 does not significantly increase the speed of tumour growth, however, it alters the shape of the tumour slightly.

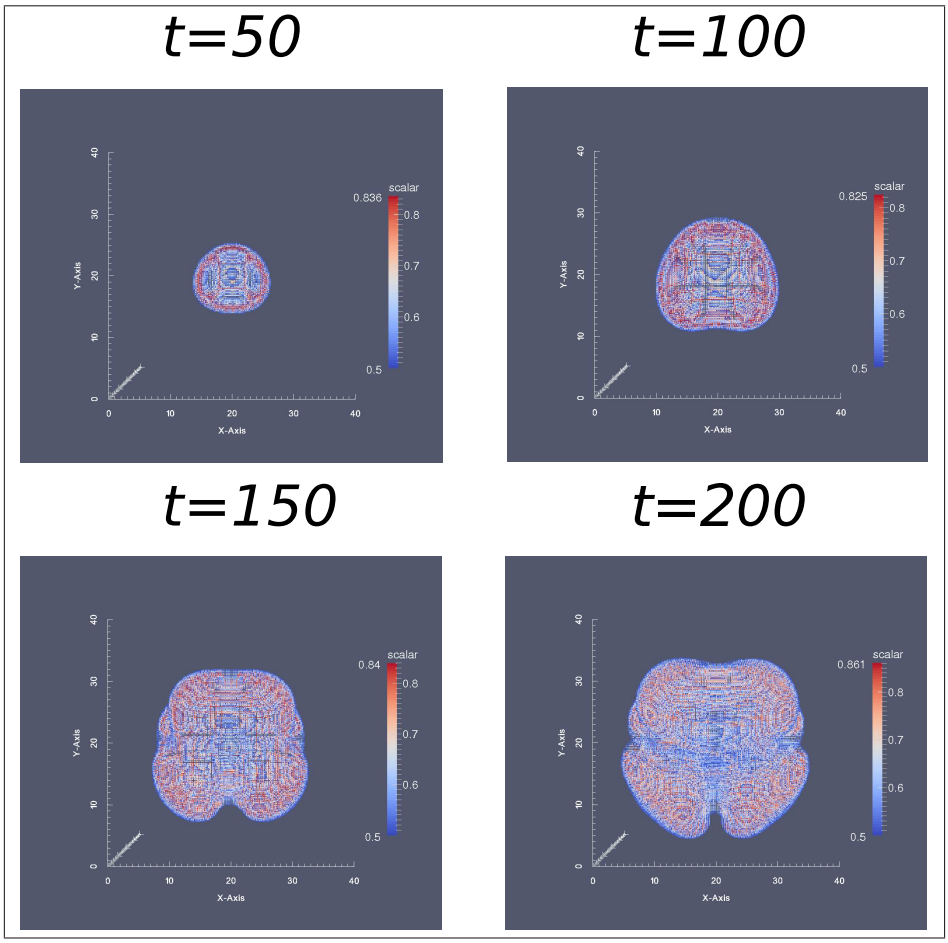

Figure 7.24: 3-D results of variable ϕ_T with $\gamma = 0.0$ and $\lambda_L = 3$ at $t = 50$ (at top left), 100 (at top right), 150 (at bottom left) and 200 (at bottom right) from the level 4 grid hierarchy.

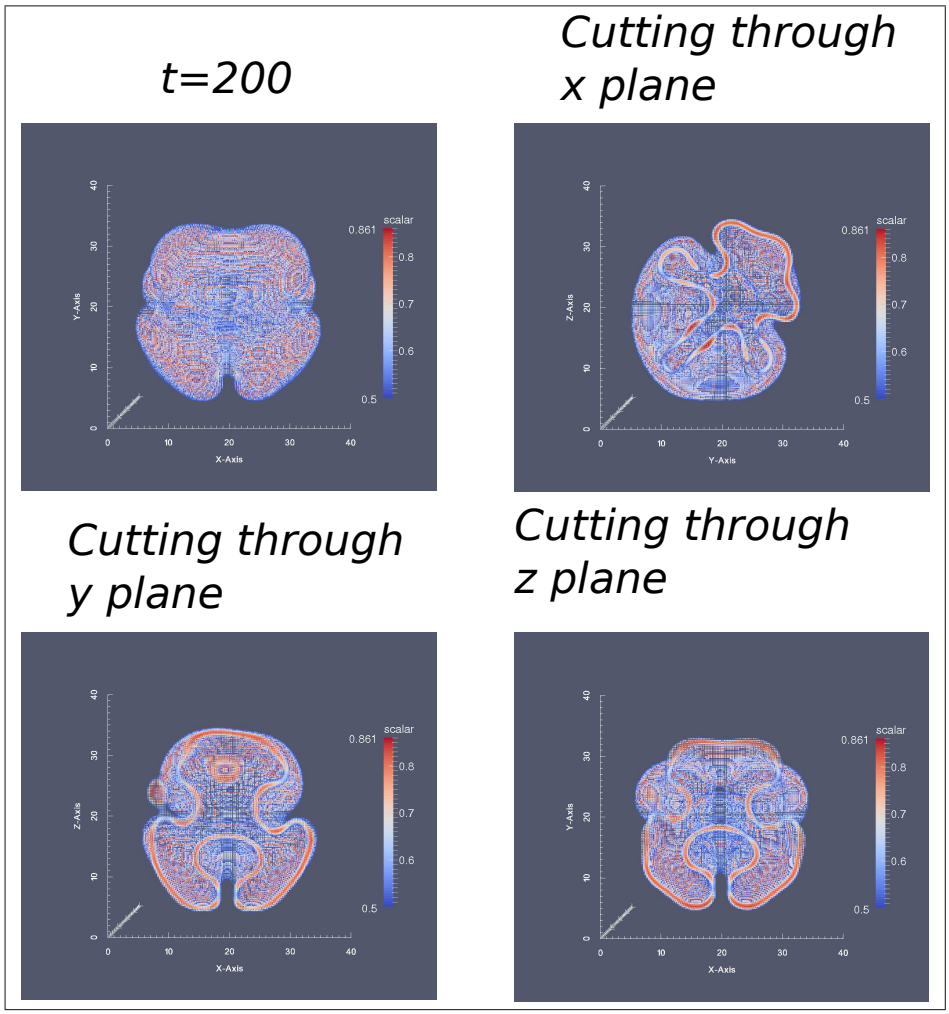

Figure 7.25: 3-D results and images of cross-sections for variable ϕ_T with $\gamma = 0.0$ and $\lambda_L = 3$ from level 4. In this figure, the solution of ϕ_T at $t = 200$ is displayed again in the top-left feature (it is previously presented in Figure 7.24); the cross-section through the plane $x = 20$ is presented in the top-right feature; the cross-section through the plane $y = 20$ plane is in the bottom-left feature; and finally the cross-section through the plane $z = 20$ is displayed in the bottom-right feature.

The third test is to increase λ_N which is the death rate of tumour cells through necrosis to 5 (up from 3). The results are presented in Figures 7.26 and 7.27. From these solutions, it seems increasing λ_N positively affects the speed of tumour growth, and the interface of the solution of ϕ_T at $t = 200$ has reached several domain boundaries.

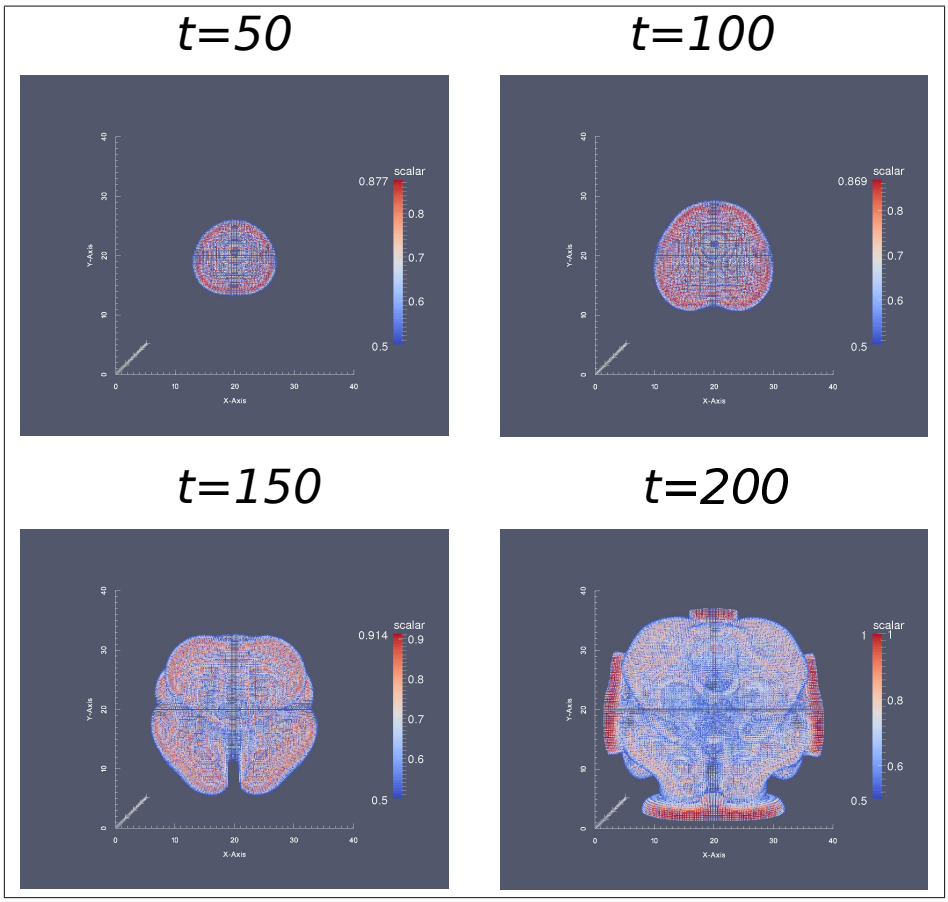

Figure 7.26: 3-D results of variable ϕ_T with $\gamma = 0.0$ and $\lambda_N = 5$ at $t = 50$ (at top left), 100 (at top right), 150 (at bottom left) and 200 (at bottom right) from the level 4 grid hierarchy.

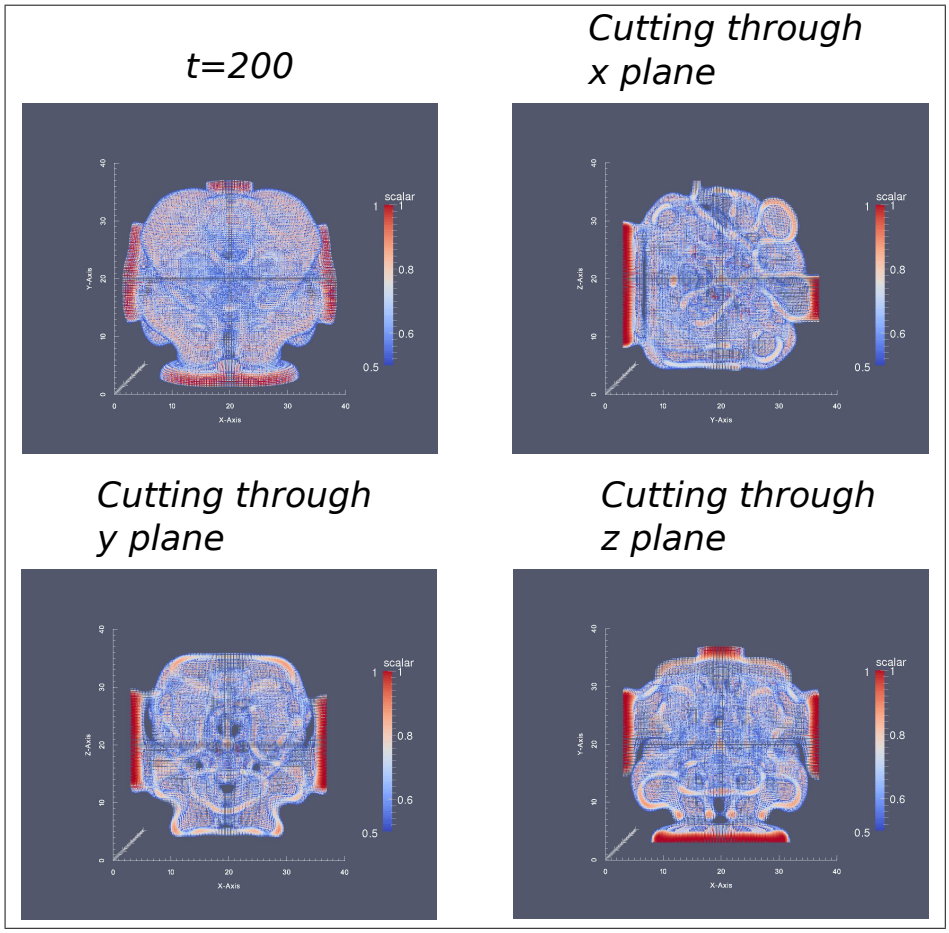

Figure 7.27: 3-D results and images of cross-sections for variable ϕ_T with $\gamma = 0.0$ and $\lambda_N = 5$ from level 4. In this figure, the solution of ϕ_T at $t = 200$ is displayed again in the top-left feature (it is previously presented in Figure 7.26); the cross-section through the plane $x = 20$ is presented in the top-right feature; the cross-section through the plane $y = 20$ is in the bottom-left feature; and finally the cross-section through the plane $z = 20$ is displayed in the bottom-right feature.

The fourth test is to decrease M which is the mobility constant to 5 (down from 10). The results are presented in Figures 7.28 and 7.29. The speed of tumour growth is significantly affected by this change, so the solution at $t = 200$ is not presented. In addition, the solution at $t = 150$ suggests the initial out-layer of healthy/tumour interface has exited through the boundary. Therefore, the cross-sections are performed on the solution at $t = 100$.

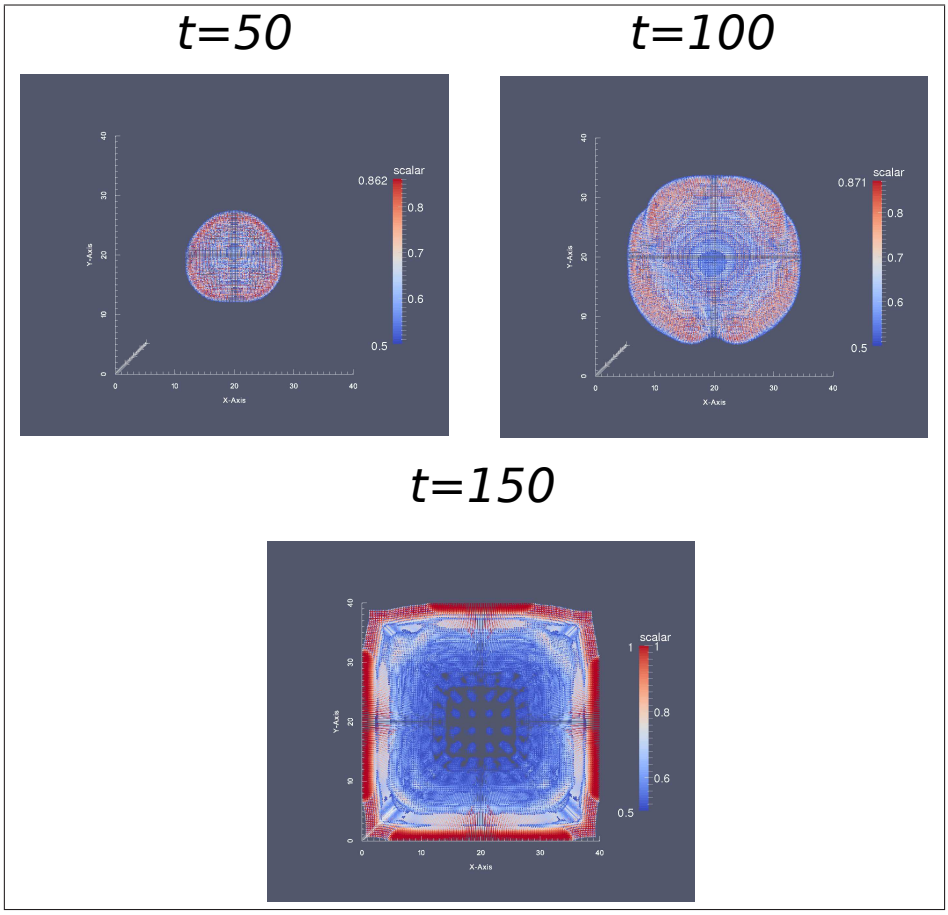

Figure 7.28: 3-D results of variable ϕ_T with $\gamma = 0.0$ and $M = 5$ at $t = 50$ (at top left), 100 (at top right) and 150 (at bottom left) from the level 4 grid hierarchy.

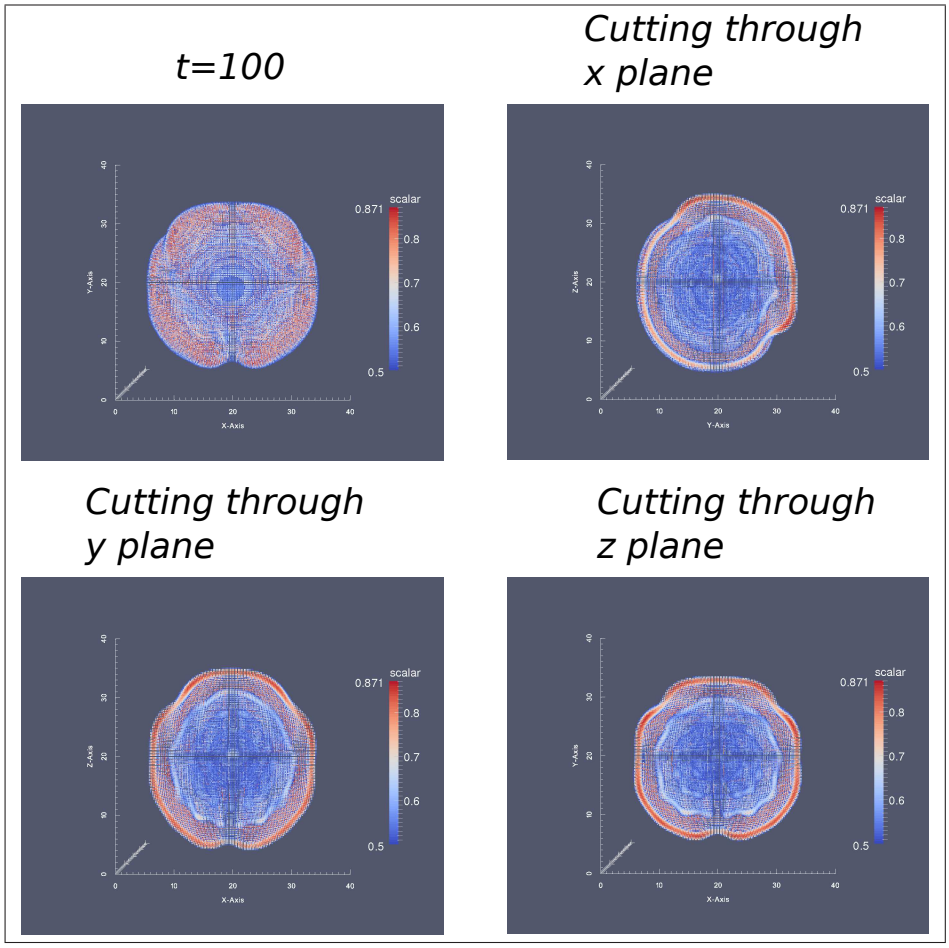

Figure 7.29: 3-D results and images of cross-sections for variable ϕ_T with $\gamma = 0.0$ and $M = 5$ from level 4. In this figure, the solution of ϕ_T at $t = 100$ is displayed again in the top-left feature (it is previously presented in Figure 7.28); the cross-section through the plane $x = 20$ is presented in the top-right feature; the cross-section through the plane $y = 20$ is in the bottom-left feature; and finally the cross-section through the plane $z = 20$ is displayed in the bottom-right feature.

The final test is to increase M to 15 (the original choice is 10 as presented in Table 7.1 and the choice presented in the previous test is 5). The results are presented in Figures 7.30 and 7.31, and show a more isotropic shape.

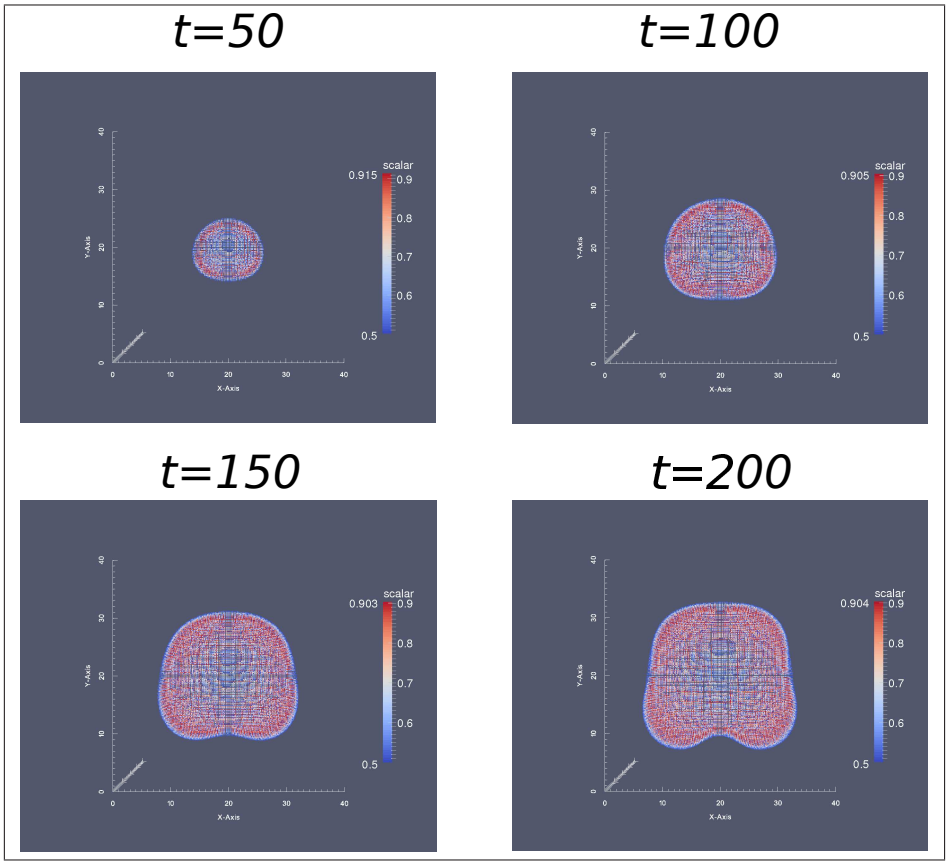

Figure 7.30: 3-D results of variable ϕ_T with $\gamma = 0.0$ and $M = 15$ at $t = 50$ (at top left), 100 (at top right), 150 (at bottom left) and 200 (at bottom right) from the level 4 grid hierarchy.

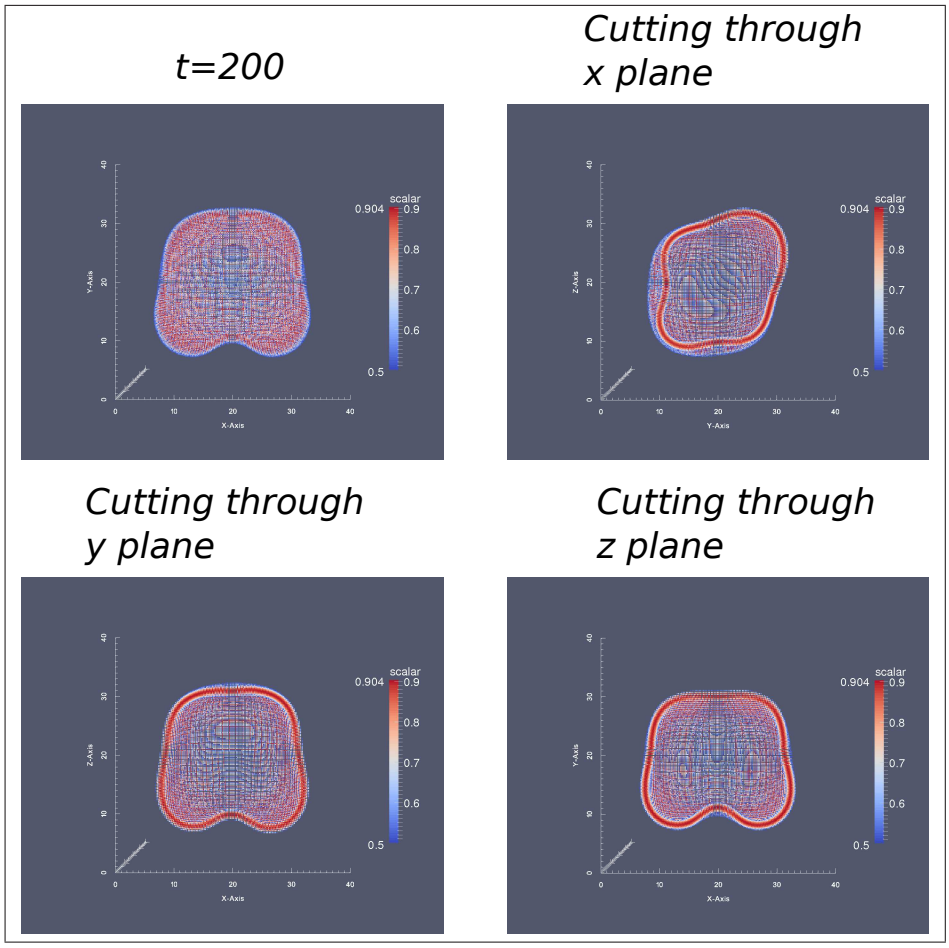

Figure 7.31: 3-D results and images of cross-sections for variable ϕ_T with $\gamma = 0.0$ and $M = 15$ from level 4. In this figure, the solution of ϕ_T at $t = 200$ is displayed again in the top-left feature (it is previously presented in Figure 7.30); the cross-section through the plane $x = 20$ is presented in the top-right feature; the cross-section through the plane $y = 20$ is in the bottom-left feature; and finally the cross-section through the plane $z = 20$ is displayed in the bottom-right feature.

Having presented a number of 3-D results, in the following section, we conclude this chapter with a discussion on the multigrid convergence issue that is mentioned previously.

7.5 Discussion

In this section, we conclude our discussion of this multi-phase-field model of tumour growth. First of all, the difficulties of obtaining an optimal multigrid convergence from solving the pressure equation are discussed. It was noted in [226] that "extensive numerical tests indicated very poor multigrid convergence when the term $\nabla \cdot (\kappa(\phi_T, \phi_D)\, \mu \nabla \phi_T)$ was treated implicitly in the pressure equation". Our experiments support this observation since the pressure equation shows poor multigrid convergence in our fully-implicit implementation too. Here we illustrate this issue with the case of $\gamma = 0.0$ and $\varepsilon = 0.2$.

In order to demonstrate this issue, we firstly show that, within our multigrid solver, the infinity norm of residuals from four variables: ϕ_T, μ, ϕ_D and n, has an optimal multigrid convergence. This is shown in Figure 7.32, using results obtained from a typical time step using the BDF2 method. Four grid hierarchies are used to generated these results, as shown in Table 7.2. We implement two stopping criteria, and at least one of them must be satisfied in order to continue to the next time step. The first one is a relative stopping criterion, which takes the infinity norm after the first V-cycle in the current time step, and flags that we have converged if the infinity norm is reduced by a factor of 10^{-9} by subsequent V-cycles. The second one is an absolute stopping criterion, which flags convergence of the solver at the current time step if the infinity norm is smaller than 10^{-11}. Results shown in this figure suggest these four variables (excluding the pressure variable p) have an optimal multigrid convergence, and the number of V-cycles required to reduce the residuals is independent of the grid sizes.

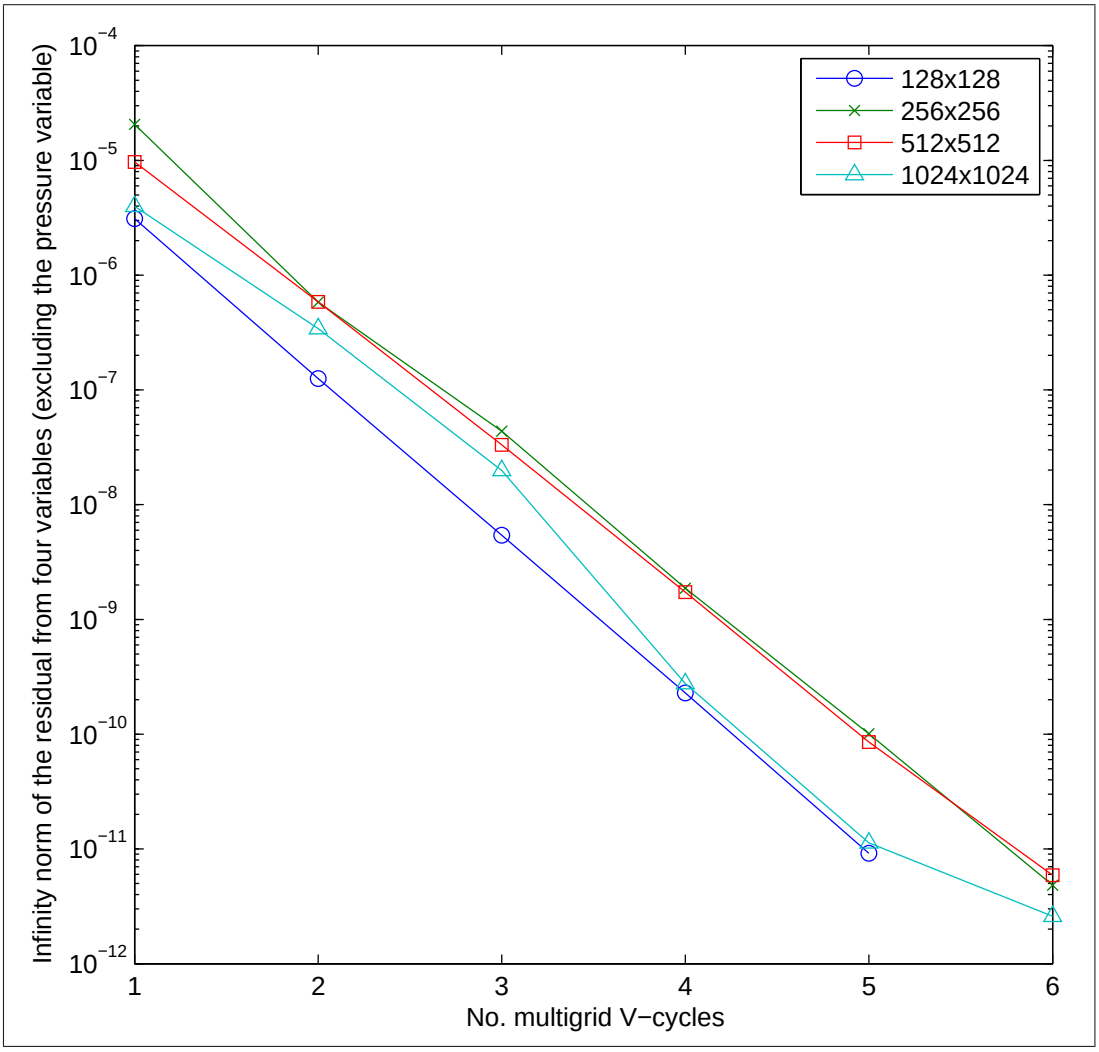

Figure 7.32: The convergence rate of a typical multigrid V-cycle from a typical time step. The vertical axis shows the values of $\max\{||\phi_T||_\infty, ||\mu||_\infty, ||\phi_D||_\infty, ||n||_\infty\}$ after each V-cycle.

On the other hand, the infinity norm from the pressure variable p does not show an optimal multigrid convergence, and this is illustrated in Figure 7.33. Results from this figure suggest that the reductions in the infinity norm is not independent from grid sizes. As mentioned previously, a related problem has already been identified by Wise et al. in [226]. The precise causes of this sub-optimal performance is not known, however it may be related to inexact solution of the coarsest grid problems or to the choice of boundary conditions for p (noting that this behaviour was not observed in the CHHS results of the previous chapter, where different pressure boundary condition were presented). Note that the effect of this poor convergence of p is not particularly detrimental to the performance of the solver overall since the second order results illustrated in Section 7.4.2 were obtained using a stopping criterion that weights the norm of p much less than the norms of the other variables for each sequence of V-cycles (at each time step).

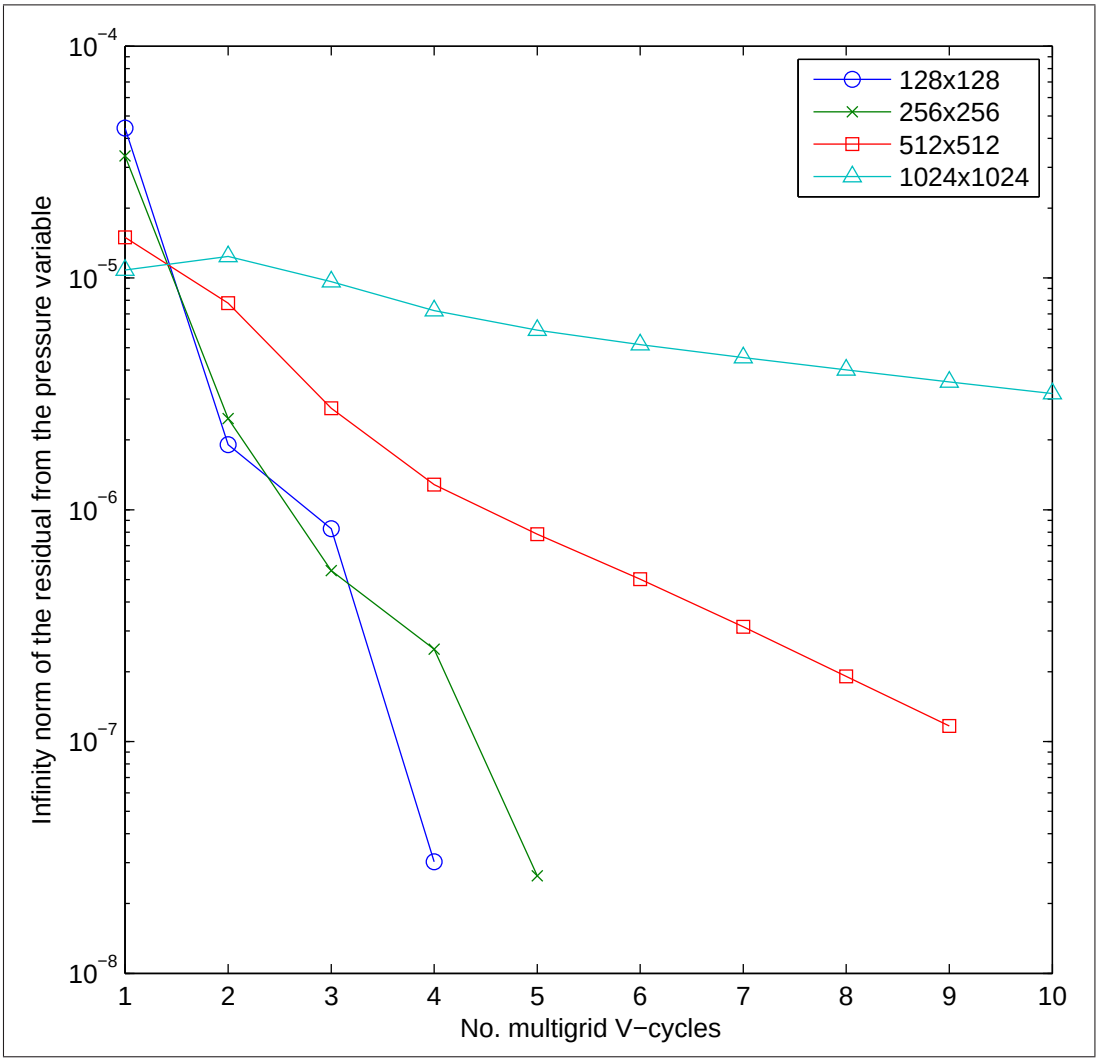

Figure 7.33: The convergence rate of a typical multigrid V-cycle from a typical time step. The shown results are from solving the pressure equation.

To sum up, in Section 7.1, a brief literature review on tumour modeling is given, where different types of models of tumour growth are described. One of these is from Wise et al. [226], and this model is presented in Section 7.2. In order to understand and solve this model, implementations of the solver used by Wise et al. and our solver are described in detail in Section 7.3. Furthermore, in Section 7.3.2, we show our full discrete system, which includes the use of penalty terms and smoothed cut-off functions. Results from solving this model are presented in Section 7.4. Firstly we attempt to validate our results against those presented in [226]. Due to the sensitivity of this model to its initial condition, we are unable to provide quantitative validations for all cases. However, our implementation of the multigrid solvers and other techniques has already been validated in previous chapters, and we therefore have confidence in our solver. Furthermore, through our convergence tests we are able to demonstrate that an overall second order convergence rate can be obtained, which previously was not achieved in [226]. 3-D results from this model are also obtained and illustrated in Section 7.4.3, where we illustrate the effects of altering different input parameters. In Section 7.5, we discuss the issue of poor multigrid convergence associated with the pressure variable. In the following chapter, we conclude the work presented in this thesis, and describe possible future work.

Chapter 8

Conclusions

The main objective of this thesis is to develop an efficient, accurate and reliable numerical solver for general parabolic systems of PDEs. In Chapter 1, a general introduction to nonlinear parabolic systems is presented, along with descriptions of five different mathematical models which are considered in this thesis. These models range from fluid dynamics, solidification to tumour growth. In order to solve these systems of PDEs, discretization methods are required. In Chapter 2, a number of these methods are introduced in Section 2.1. Within these descriptions, we emphasize the focus in this thesis which is using finite difference methods (see Section 2.1.1.1), cell-centred Cartesian grids with Neumann or Dirichlet boundary conditions (see Section 2.1.1.2) and backward differentiation formulae (see Section 2.1.2.2). The resulting algebraic systems can then be solved through a number of solution methods. Within them, the multigrid methods are the focus of this thesis. We describe in detail a number of variations. To further improve the efficiency, parallel computing may be employed. The main application of parallelization presented in this thesis comes from mesh partitioning via domain decomposition.

Having given the descriptions of the multigrid solution methods in theory, the actual implementation is then explained in Chapter 3, which consists of a software library, PARAMESH, and a multigrid solver, Campfire. Chapter 4 is where we discuss the first of five models: the model of binary alloy solidification. These results are already summarised in [27], where we show our multigrid solver has successfully scaled up to 1000 cores. In addition, in Chapter 4, we discuss several improvements made to Campfire and PARAMESH. The models considered subsequently all benefit from these enhancements.

Two models of Thin film flows are presented in Chapter 5. The droplet spreading model is explained in Section 5.1. We validate our results against those presented by Gaskell et al. in [81], and are able to conclude that our multigrid solver is efficient and accurate. A number of tests are also conducted which suggest the use of adaptivity and parallelism are both effective. In Chapter 5, we also include another variation of the long-wave model of thin film flows: fully-developed flows. For this mode, we obtain the same second order convergence rate as previously, and are able to validate our results against those in [125].

The Cahn-Hilliard-Hele-Shaw system of equations are presented in Section 6.1, Chapter 6. This model is from [225], and has been used as a stepping stone toward the model of tumour growth that follows. We are not only able to solve this model with our multigrid solver, but also able to replicate the solver used by Wise in [225]. A second order convergence rate is demonstrated through the use of our settings, while a first order convergence rate is replicated from our implementation of Wise's solver.

The multi-phase-field model of tumour growth is discussed in Chapter 7. First of all, a brief literature review on tumour modeling is summarised in Section 7.1. The model of tumour growth that is studied in this thesis is from [226], and is based upon a derivation from the CHHS system of equations. From carefully learning the work of Wise et al., we identified the root causes of why a second order convergence rate is not obtained. Improvements are then implemented and, for the first time, a second order convergence rate is obtained for this type of model of tumour growth. We also illustrate a number of solutions from 3-D simulations and the issue of slow multigrid convergence for the pressure variable is highlighted.

In the following section, possible future developments are summarised to conclude this thesis.

8.1 Future Developments

Here we suggest a number of possible extension to this research:

- one of the possible extension for this research is to use the Newton multigrid instead of the nonlinear multigrid with FAS. Brabazon et al. in [28] show that Newton multigrid on a family of nonlinear parabolic equations is able to obtain a better multigrid convergence rate than the nonlinear multigrid with FAS. The time cost per each time step for the Newton multigrid is also much less. The Newton multigrid method is described in detail in Section 2.4.3. This Newton multigrid method is not implemented in Campfire. With uniform grids, this should be straightforward

in Campfire, although the difficulties may rise from the parallelization of the global Jacobian matrix.

- We believe the use of Newton multigrid may significantly improve the parallel scaling. By using the Newton multigrid, the problem on the coarsest grid becomes linear, in fact the problem solved in the multigrid solver is linear, thus we can afford much finer coarsest grid since linear problems are easier to solve "exactly" with a relatively small number of iterations (or even a sparse direct method). In Section 5.4.4, we identified one of the possible inefficiencies in parallel multigrid is the coarsest grid. In the nonlinear multigrid method with FAS, this coarsest grid has to be very coarse so a nonlinear problem can be solved "exactly". Using a finer coarsest grid, from the results presented in Section 5.4.4, suggests a better parallel efficiency, however, a trade-off occurs when the nonlinear problem requires too many iterations on the finer coarsest grid.

- It is also possible to consider the application of Newton-Krylov methods (e.g. GMRES) with multigrid as the preconditioner. If the above two steps can be achieved, the resulting application can be used as the preconditioner. Furthermore, if one uses linear multigrid as a preconditioner for an iterative solver of the Newton linearization an exact coarse grid solution may not be necessary at all. Thus, providing the possibilities of an even finer coarsest grid.

- To combine the Newton multigrid with adaptivity is still elusive, from our best understanding, the MLAT approach for the FAS multigrid may not be valid or straightforward in terms of software implementation in parallel. One possible solution is to use the adaptive composite grid method which is described in general in Section 2.4.5.

- Another possible extension comes from the fully-developed flow, this is briefly mentioned in Section 5.7.3. By introducing a time-dependent Dirichlet boundary condition at the upstream, we may simulate a flow with regular or variable waves on top of one or more topographies. Then we may compare these results with the solutions of the Navier-Stokes equations.

- There are also other alternative tumour models, for example, Hawkins-Daarud et al. [106] introduced a tumour model which was quite similar to the one solved here. Thus, one may also be interested in implementing this model in Campfire. It would be possible to gain second order convergence rate from this model of Hawkins-Daarud et al.

- The final extension suggested here is to investigate the issue with the pressure equation and its convergence from the model of tumour growth presented by Wise et al. in [226]. One possible solution may be to numerically analyse the model to gain additional necessary understanding. It was believed by the authors that since it obeys the Darcy's law, it ought to be difficult to converge.

Bibliography

[1] J.A. Adam. A simplified mathematical model of tumor growth. *Mathematical Biosciences*, 81:229–244, 1986.

[2] J.A. Adam. A mathematical model of tumor growth II effects of geometry and spatial nonuniformity on stability. *Mathematical Biosciences*, 86:183–211, 1987.

[3] J.A. Adam. A mathematical model of tumor growth III comparison with experiment. *Mathematical Biosciences*, 86:213–227, 1987.

[4] *AFEPack*, accessed on 12th Apr 2014. http://dsec.pku.edu.cn/r̃li/software.php.

[5] S. Ahmed, C.E. Goodyer, and P.K. Jimack. An efficient preconditioned iterative solution of fully-coupled elastohydrodynamic lubrication problems. *Applied Numerical Mathematics*, 62:649–663, 2012.

[6] R.E. Alcouffe, Achi Brandt, J.E. Dendy, and J.W. Painter. The multi-grid method for the diffusion equation with strongly discontinuous coefficients. *SIAM Journal on Scientific Computing*, 2:430–454, 1981.

[7] R.F. Almgren. Variational algorithms and pattern formation in dendritic solidification. *Journal of Computational Physiscs*, 106, 1993.

[8] D. Ambrosi and L. Preziosi. On the closure of mass balance models for tumor growth. *Mathematical Models and Methods in Applied Sciences*, 12:737–754, 2002.

[9] G.M. Amdahl. Validity of the single-processor approach to achieving large-scale computing capabilities. *AFIPS '67 (Spring) Proceedings*, 30:483–485, 1967.

[10] P.R. Amestoy, I.S. Duff, J.Y. L'Excellent, Y. Robert, F.H. Rouet, and B. Ucar. On computing inverse entries of a sparse matrix in an out-of-core environment. *SCIDAC 2009: Scientific Discovery Through Advanced Computing*, 34:A1975–A1999, 2012.

[11] P. Amodio and F. Mazzia. A parallel Gauss-Seidel method for block tridiagonal linear systems. *SIAM Journal on Scientific Computing*, 16:1451–1461, 1995.

[12] A.R.A. Anderson and M.A.J. Chaplain. Continuous and discrete mathematical models of tumor-induced angiogenesis. *Bulletin of Mathematical Biology*, 60:857–900, 1998.

[13] D.M. Anderson, G.B. McFadden, and A.A. Wheeler. Diffuse-interface methods in fluid mechanics. *Annual Review of Fluid Mechanics*, 30:139–165, 1998.

[14] R.P. Araujo and D.L.S. McElwain. A history of the study of solid tumour growth: The contribution of mathematical modelling. *Bulletin of Mathematical Biology*, 66:1039–1091, 2004.

[15] *ARC2*, accessed on 14th Sep 2014. http://it.leeds.ac.uk/info/124.

[16] R. De Asmundis, D. di Serafino, F. Riccio, and G. Toraldo. On spectral properties of steepest descent methods. *IMA Journal of Numerical Analysis*, 33:1416–1435, 2013.

[17] S.A. Babkin. *The practice of parallel programming*. CreateSpace Independent Publishing Platform, 2010.

[18] M.J. Baines, M.E. Hubbard, and P.K. Jimack. A moving mesh finite element algorithm for the adaptive solution of time-dependent partial differential equations with moving boundaries. *Applied Numerical Mathematics*, 54:450–469, 2005.

[19] M.J. Baines, M.E. hubbard, P.K. Jimack, and A.C. Jones. Scale-invariant moving finite elements for nonlinear partial differential equations in two dimensions. *Applied Numerical Mathematics*, 56:230–252, 2006.

[20] W.D. Bascom, R.L. Cottington, and C.R. Singleterry. Dynamic surface phenomena in the spontaneous spreading of oils on solids. *Advances in Chemistry*, 43:355–379, 1964.

[21] P. Bastian, W. Hackbusch, and G. Wittum. Additive and multiplicative multi-grid - a comparison. *Computing*, 60:345–364, 1998.

[22] M. Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of Computational Physics*, 182:418–477, 2002.

[23] M.J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.

[24] A. Bertozzi. The mathematics of moving contact lines in thin liquid films. *Notices of the AMS*, 45:689–697, 1998.

[25] A. Bertozzi and M.P. Brenner. Linear stability and transient growth in driven contact lines. *Physics of Fluids*, 9:530–539, 1997.

[26] M. Berzins, J. Schmidt, Q. Meng, and A. Humphrey. Past, present and future scalability of the Uintah software. In *Proceedings of Blue Waters Workshop*, 2012.

[27] P. Bollada, C. Goodyer, P. Jimack, A. Mullis, and F. Yang. Thermalsolute phase field three dimensional simulation of binary alloy solidification. *Journal of Computational Physics*, Submitted August 2014.

[28] K.J. Brabazon, M.E. Hubbard, and P.K. Jimack. Nonlinear multigrid methods for second order differential operators with nonlinear diffusion coefficient. *Computers and Mathematics with Applications*, Submmited April 2014.

[29] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31:333–390, 1977.

[30] E.L. Briggs, D.J. Sullivan, and J. Bernholc. Real-space multigrid-based approach to large-scale electronic structure calculations. *Physical Review B*, 45:14362–14375, 1996.

[31] W.L. Briggs, V.E. Henson, and S.F. McCormick. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, 2000.

[32] D.J. Browne and J.D. Hunt. A fixed grid front-tracking model of the growth of a columnar front and an equiaxed grain during solidification of an alloy. *Numerical Heat Transfer*, 45:395–419, 2004.

[33] A. Bunse-Gerstner and R. Stover. On a conjugate gradient-type method for solving complex symmetric linear systems. *Linear Algebra and its Applications*, 287:105–123, 1999.

[34] R.L. Burden and J.D. Faires. *Numerical Analysis*. Wadsworth Group, Brooks/Cole, 2001.

[35] A.C. Burton. Rate of growth of solid tumours as a problem of diffusion. *Growth*, 30:157–176, 1966.

[36] H. Byrne. Modelling solid tumour growth using the theory of mixtures. *Mathematical Medicine and Biology*, 20:341–366, 2003.

[37] H.M. Byrne, T. Alarcon, M.R. Owen, S.D. Webb, and P.K. Maini. Modelling aspects of cancer dynamics: a review. *Philosophical Transactions of the Royal Society A*, 364:1563–1578, 2006.

[38] H.M. Byrne and M.A.J. Chaplain. Growth of nonnecrotic tumors in the presence and absence of inhibitors. *Mathematical Biosciences*, 130:151–181, 1995.

[39] H.M. Byrne and M.A.J. Chaplain. Growth of necrotic tumors in the presence and absence of inhibitors. *Mathematical Biosciences*, 135:187–216, 1996.

[40] H.M. Byrne and J.R. King. A two-phase model of solid tumour growth. *Applied Mathematics Letters*, 16:567–573, 2003.

[41] G. Caginalp. Stefan and Hele-Shaw type models as asymptotic limits of the phase-field equations. *Physical Review A*, 39:5887–5896, 1989.

[42] J.W. Cahn. On spinodal decomposition. *Acta Metall*, 9:795–801, 1961.

[43] J.W. Cahn and J. Hilliard. Free energy of a nonuniform system I: inter-facial free energy. *Journal of Chemical Physics*, 28:258, 1958.

[44] J. Cao, G.F. Zhou, C. Wang, and X.Z. Dong. A hybrid adaptive finite difference method powered by a posteriori error estimation technique. *Journal of Computational and Applied Mathematics*, 259:117–128, 2014.

[45] B. Carpentieri, Y.F. Jing, T.Z. Huang, W.C. Pi, and X.Q. Sheng. Combining the CORS and BiCORSTAB iterative methods with MLFMA and SAI preconditioning for solving large linear systems in electromagnetics. *Applied Computational Electromagnetics Society Journal*, 27:102–111, 2012.

[46] B. Chapman, G. Jost, and R. Van De Pas. *Using OpenMP: portable shared memory parallel programming*. MIT Press, 2007.

[47] L.Q. Chen. Phase-field models for microstructure evolution. *Annual Review of Materials Research*, 32:113–140, 2002.

[48] S. Chen, B. Merriman, S. Osher, and P. Smereka. A simple level set method for solving Stefan problems. *Journal of Computational Physics*, 135, 1997.

[49] E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices. *Journal of Computational and Applied Mathematics*, 86:387–414, 1997.

[50] J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43:50–67, 1947.

[51] V. Cristini, X.R. Li, J.S. Lowengrub, and S.M. Wise. Nonlinear simulations of solid tumor growth using a mixture model: invasion and branching. *Journal of Mathematical Biology*, 58:723–763, 2009.

[52] V. Cristini, J. Lowengrub, and Q. Nie. Nonlinear simulation of tumor growth. *Journal of Mathematical Biology*, 46:191–224, 2003.

[53] J.T. Cui. Multigrid methods for two-dimensional Maxwell's equations on graded meshes. *Journal of Computational and Applied Mathematics*, 255:231–247, 2014.

[54] E. Cuthill. Several strategies for reducing the bandwidth of matrices. *in Sparse Matrices and their Applications, edited by D.J. Rose and R.A. Willoughby, Plenum Press, New York*, pages 157–166, 1972.

[55] C. Dawson and V. Aizinger. A discontinuous Galerkin method for three-dimensional shallow water equations. *Journal of Scientific Computing*, 22:245–267, 2004.

[56] P.G. de Gennes. Wetting: statics and dynamics. *Reviews of Modern Physics*, 57:827–863, 1985.

[57] *DEAL.II*, accessed on 16th Apr 2014. http://www.dealii.org/.

[58] M. Decre and J. Baret. Gravity-driven flows of viscous liquids over two-dimensional topographies. *Journal of Fluid Mechanics*, 487:147–166, 2003.

[59] K. Dekker and J.G. Verwer. *Stability of Runge-Kutta methods for stiff nonlinear differential equations*. Elsevier Science, Amsterdam, 1984.

[60] L.J. Deng, T.Z. Huang, X.L. Zhao, L. Zhao, and S. Wang. An economical aggregation algorithm for algebraic multigrid (AMG). *Journal of Computational Analysis and Applications*, 16:181–198, 2014.

[61] P. Deuflhard. *Newton methods for nonlinear problems*. Springer, Berlin, 2004.

[62] R. Diestel. *Graph theory*. Springer, 2005.

[63] J.A. Diez, L. Kondic, and A. Bertozzi. Global models for moving contact lines. *Physical Review E*, 63:011208:1–011208:13, 2000.

[64] J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. van der Vorst. *Numerical linear algebra for high-performance computers*. SIAM, 1998.

[65] P. DuChateau and D. Zachmann. *Applied Partial Differential Equations*. Harper & Row, Publishers, Inc., 1989.

[66] I.S. Duff and J.A. Scott. MA42 - a new frontal code for solving sparse unsymmetric systems. *Technical Report RAL 93-064, Rutherford Appleton Laboratory*, 1993.

[67] *The Distributed and Unified Numerics Environment (DUNE)*, accessed on 12th Apr 2014. http://www.dune-project.org.

[68] R.E. Durand. Cell cycle kinetics in an in vitro tumor model. *Cell Tissue Kinet*, 9:430–412, 1976.

[69] E. Emmerich. Stability and error of the variable two-step BDF for semilinear parabolic problems. *Journal of Applied Mathematics and Computing*, 19:33–55, 2005.

[70] C. Engelmann. Scaling to a million cores and beyond: using light-weight simulation to understand the challenges ahead on the road to exascale. *Future Generation Computer Systems*, 30:49–65, 2014.

[71] D. Eyre. Unconditionally gradient stable time marching the Cahn-Hilliard equation. *Computational and Mathematical Models of Microstructural Evolution*, 53:1686–1712, 1998.

[72] C.A. Felippa. Solution of linear equations with skyline-stored symmetric matrix. *Computers and Structures*, 5:13–29, 1975.

[73] C.K. Filelis-Papadopoulos, G.A. Gravvanis, and E.A. Lipitakis. On the numerical modeling of convection-diffusion problems by finite element multigrid preconditioning methods. *Advances in Engineering Software*, 68:56–69, 2014.

[74] M.J. Flynn. Very high speed computing systems. *Proceedings of the IEEE*, 12:1901–1909, 1966.

[75] M.J. Flynn and K.W. Rudd. Parallel architectures. *ACM Computing Surveys*, 28:67–70, 1996.

[76] J. Folkman. Tumor angiogenesis factor. *Cancer Research*, 34:2109–2113, 1974.

[77] J. Folkman and M. Hochberg. Self-regulation of growth in three dimensions. *Journal of Experimental Medicine*, 138:745–753, 1973.

[78] H.B. Frieboes, F. Jin, Y.L. Chuang, S.M. Wise, J.S. Lowengrub, and V. Cristini. Three-dimensional multispecies nonlinear tumor growth - II. tumor invasion and angiogenesis. *Journal of Theoretical Biology*, 264:1254–1278, 2010.

[79] V.K. Garg. *Concurrent and distributed computing in Java*. John Wiley & Sons, Inc., 2004.

[80] P.H. Gaskell, P.K. Jimack, Y.Y. Koh, and H.M. Thompson. Development and application of a parallel multigrid solver for the simulation of spreading droplets. *International Journal for Numerical Methods in Fluids*, 56:979–989, 2008.

[81] P.H. Gaskell, P.K. Jimack, M.Sellier, and H.M. Thompson. Efficient and accurate time adaptive multigrid simulations of droplet spreading. *International Journal for Numerical Methods in Fluids*, 45:1161–1186, 2004.

[82] P.H. Gaskell, P.K. Jimack, M. Sellier, H.M. Thompson, and M.C.T. Wilson. Gravity-driven flow of continuous thin liquid films on non-porous substrates with topography. *Journal of Fluid Mechanics*, 509:253–280, 2004.

[83] C.W. Gear. *Numerical initial value problems in ordinary differential equations*. Prentice-Hall, New Jersey, 1971.

[84] A. George and J.W. Liu. *Computer solution of large sparse positive definite systems*. Prentice-Hall Series in Computational Mathematics, 1981.

[85] U. Ghia, K.N. Ghia, and C.T. Shin. High-Resolutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, 48:387–411, 1982.

[86] M.A. Gimbrone, R.H. Aster, R.S. Cortran, J. Corkery, J.H. Jandl, and J. Folkman. Preservation of vascular integrity in organs perfused in vitro with a platelet-rich medium. *Nature*, 221:33–36, 1969.

[87] B. Gmeiner, H. Kostler, M. Sturmer, and U. Rude. Parallel multigrid on hierarchical hybrid grids: a performance study on current high performance computing clusters. *Concurrency and Computing: Practice and Experience*, 26:217–240, 2012.

[88] G.H. Golub and C.F. Van Loan. *Matrix Computations, third edition*. The Johns Hopkins University Press, 1996.

[89] G.H. Golub and C.F. Van Loan. *Matrix Computations, 4th edition*. The Johns Hopkins University Press, 2013.

[90] C.E. Goodyer and M. Berzins. Adaptive timestepping for elastohydrodynamic lubrication solvers. *SIAM Journal on Scientific Computing*, 28:626–650, 2006.

[91] C.E. Goodyer, P.K. Jimack, A.M. Mullis, H.B. Dong, and Y. Xie. On the fully implicit solution of a phase-field model for binary alloy solidification in three dimensions. *Advances in Applied Mathematics and Mechanics*, 4:665–684, 2012.

[92] S. Gottlieb, C.W. Shu, and E. Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Review*, 43:89–112, 2001.

[93] J.R. Green, P.K. Jimack, A.M. Mullis, and J. Rosam. Towards a three-dimensional parallel, adaptive, multilevel solver for the solution of nonlinear, time-dependent, phase-change problems. *Parallel, Distributed and Grid Computing for Engineering*, pages 251–274, 2009.

[94] J.R. Green, P.K. Jimack, A.M. Mullis, and J. Rosam. An adaptive, multilevel scheme for the implicit solution of three-dimensional phase-field equations. *Numerical Metthods for PDEs*, 27:106–120, 2011.

[95] H.P. Greenspan. Models for the growth of a solid tumor by diffusion. *Studies in Applied Mathematics*, 51:317–340, 1972.

[96] H.P. Greenspan. On the self-inhibited growth of cell cultures. *Growth*, 38:81–95, 1974.

[97] H.P. Greenspan. On the growth and stability of cell cultures and solid tumors. *Journal of Theoretical Biology*, 56:229–242, 1976.

[98] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI portable parallel programming with the message-passing interface*. The MIT press, 1999.

[99] D.H. Guo and W. Gropp. Applications of the streamed storage format for sparse matrix operations. *International Journal of High Performance Computing Applications*, 28:3–12, 2014.

[100] Z. Guo, J. Mi, S. Xiong, and P.S. Grant. Phase field study of the tip operating state of a freely growing dendrite against convection using a novel parallel multigrid approach. *Journal of Computational Physics*, 257:278–297, 2014.

[101] J. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31:532–533, 1988.

[102] W. Hackbusch. *Multi-grid methods and applications*. Springer, Berlin, 1985.

[103] W. Hackbusch. *Iterative solution of large sparse systems of equations*. Springer-Verlag, New York, 1994.

[104] L.A. Hageman and D.M. Young. *Applied iterative methods*. Academic Press, New York, 1981.

[105] E. Hairer, S.P. Norsett, and G. Wanner. *Solving ordinary differential equations I*. Springer Series in Computational Mathematics, Springer, Berlin, 1993.

[106] A. Hawkins-Daarud, K.G. van der Zee, and J.T. Oden. Numerical simulation of a thermodynamically consistent four-species tumor growth model. *International Journal for Numerical Methods in Biomedical Engineering*, 28:3–24, 2012.

[107] M. Hayes, S. O'Brien, and J.H. Lammers. Green's function for steady flow over a small two-dimensional topography. *Physics of Fluids*, 12:2845–2858, 2000.

[108] *HDF5*, accessed on 17th Apr 2014. http://www.hdfgroup.org/HDF5/.

[109] *HECToR*, accessed on 28th Apr 2014. http://www.hector.ac.uk/.

[110] R.P. Heikes, D.A. Randall, and C.S. Konor. Optimized icosahedral grids: performance of finite-difference operators and multigrid solver. *Monthly Weather Review*, 141:4450–4469, 2013.

[111] R. Hempel and A. Schuller. The GMD communications subroutine library for grid-oriented problems. *Arbeitspapiere der GMD*, 589, 1991.

[112] M.R. Hestenes. *Conjugate direction methods in optimization*. Springer-Verlag, Berlin, 1980.

[113] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[114] J.S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer, 2008.

[115] L.M. Hocking. Rival contact-angle models and the spreading of drops. *Journal of Fluid Mechanics*, 239:671–681, 1992.

[116] T.Y. Hou, Z. Li, S. Osher, and H. Zhao. A hybrid method for moving interface problems with application to the Hele-Shaw flow. *Journal of Computational Physics*, 134:236–259, 1997.

[117] Z. Hu, S.M. Wise, C. Wang, and J.S. Lowengrub. Stable and efficient finite-difference nonlinear-multigrid schemes for the phase field crystal equation. *Journal of Computational Physics*, 228:5323–5339, 2009.

[118] M.E. Hubbard and H.M. Byrne. Multiphase modelling of vascular tumour growth in two spatial dimensions. *not published*, in press.

[119] W. Hundsdorfer and J.G. Verwer. *Numerical solution of time-dependent advection-diffusion-reaction equations, volume 33 of Springer Series in Computational Mathematics*. Springer Verlag, 2003.

[120] J.M. Hyman, R.J. Knapp, and J.C. Scovel. High order finite volume approximations of differential operators on nonuniform grids. *Physica D.*, 60:112–138, 1992.

[121] G.S. Jiang and C.W. Shu. Efficient implementation of weighted eno schemes. *Journal of Computational Physics*, 126:202–228, 1996.

[122] A.C. Jones and P.K. Jimack. An adaptive multigrid tool for elliptic and parabolic systems. *International Journal for Numerical Methods in Fluids*, 47:1123–1128, 2005.

[123] J.Rosam, P.K. Jimack, and A. Mullis. A fully implicit, fully adaptive time and space discretisation method for phase-field simulation of binary alloy solidification. *Journal of Computational Physics*, 225:1271–1287, 2007.

[124] D. Juric and G. Tryggvason. A front-tracking method for dendritic solidification. *Journal of Computational Physics*, 123:127–148, 1996.

[125] S. Kalliadasis, C. Bielarz, and G.M. Homsy. Steady free-surface thin film flows over topography. *Physics of Fluids*, 12:1889–1898, 2000.

[126] C. Kamath and A. Sameh. A projection method for solving non-symmetric linear systems on multiprocessors. *Parallel Computing*, 9:291–312, 1989.

[127] A. Karma and W.J. Rappel. Phase-field method for computationally efficient modeling of solidification with arbitrary interface kinetics. *Physical Review Letters*, 77:4050–4053, 1996.

[128] A. Karma and W.J. Rappel. Quantitative phase-field modeling of dendritic growth in two and three dimensions. *Physical Review E*, 57:4323–4349, 1998.

[129] G. Karypis and V. Kumar. *MeTiS: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0*. University of Minnesota, Minneapolis, 1998.

[130] D. Kay and R. Welford. A multigrid finite element solver for the Cahn-Hilliard equation. *Journal of Computational Physics*, 212:288–304, 2006.

[131] P. Kegel, M. Steuwer, and S. Gorlatch. dOpenCL: Towards uniform programming of distributed heterogeneous multi-/many-core systems. *Journal of Parallel and Distributed Computing*, 73:1639–1648, 2013.

[132] Y.J. Kim, M.A. Stolarska, and H.G. Othmer. A hybrid model for tumor spheroid growth in vitro I: Theoretical development and early results. *Mathematical Models and Methods in Applied Sciences*, 17:1773–1798, 2007.

[133] Y.T. Kim, N. Goldenfeld, and J. Dantzig. Computation of dendritic microstructures using a level set methods. *Physical Review E*, 62, 2000.

[134] D.B. Kirk and W.W. Hwu. *Programming massively parallel processors, second edition: a hands-on approach*. Morgan Kaufmann, 2012.

[135] J. Kleinberg and E. Tardos. *Algorithm design*. Pearson/Addison-Wesley, 2006.

[136] A. Klimke and B. Wohlmuth. Algorithm 847: spinterp: piecewise multilinear hierarchical sparse grid interpolation in MATLAB. *ACM Transactions on Mathematical Software*, 31:561–579, 2005.

[137] D.A. Knoll and P.R. McHugh. Newton-Krylov methods applied to a system of convection-diffusion-reaction equations. *Computer Physics Communications*, 88:141–160, 1995.

[138] R. Kobayashi. Modeling and numerical simulations of dendritic crystal growth. *Physica D*, 63:410–423, 1993.

[139] J. Kouatchou. Comparison of time and spatial collocation methods for the heat equation. *Journal of Computational and Applied Mathematics*, 150:129–141, 2003.

[140] Y.L. Lai, W.W. Lin, and D. Pierce. Conjugate gradient and minimal residual methods for solving symmetric indefinite systems. *Journal of Computational and Applied Mathematics*, 84:243–256, 1997.

[141] J.D. Lambert. *Numerical methods for ordinary differential system: initial value problem*. John Wiley & Sons Ltd, West Sussex, 1991.

[142] J.S. Langer. Directions in condensed matter physics (ed. G. Grinstein and G. Mazenko). *World Scientific Publishing: Singapore*, pages 164–186, 1986.

[143] B.V. Lee and A. Rutenberg. Fast and accurate coarsening simulation with an unconditionally stable time step. *Physical Review E*, 68:703, 2003.

[144] H. Lee, J.S. Lowengrub, and J. Goodman. Modeling pinchoff and reconnection in a Hele-Shaw cell. I. the models and their calibration. *Physics of Fluids*, 14:492–513, 2002.

[145] H. Lee, J.S. Lowengrub, and J. Goodman. Modeling pinchoff and reconnection in a Hele-Shaw cell. II. analysis and simulation in the nonlinear regime. *Physics of Fluids*, 14:514–545, 2002.

[146] V.W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A.D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. In *Proceedings of 37th Annual International Symposium on Computer Architecture*, pages 451–460, 2010.

[147] Y.C. Lee, H.M. Thompson, and P.H. Gaskell. An efficient adaptive multigrid algorithm for predicting thin film flow on surfaces containing localised topographic features. *Computers & Fluids*, 36:838–855, 2007.

[148] M.D. Lelah and A. Marmur. Spreading kinetics of drops on glass. *Journal of Colloid and Interface Science*, 82:518–525, 1981.

[149] J. Li, Y.F. Jiang, C.W. Yang, Q.Y. Huang, and M. Rice. Visualizing 3D/4D environmental data using many-core graphics processing units (GPUs) and multi-core central processing units (CPUs). *Computers and Geosciences*, 59:78–89, 2013.

[150] X.R. Li, V. Cristini, Q. Nie, and J.S. Lowengrub. Nonlinear three-dimensional simulation of solid tumor growth. *Discrete and Continuous Dynamical Systems - Series B*, 7:581–604, 2007.

[151] X.S. Li, M. Shao, I. Yamazaki, and E.G. Ng. Factorization-based sparse solvers and preconditioners. *SCIDAC 2009: Scientific Discovery Through Advanced Computing*, 180, 2009.

[152] J. Liesen and P. Tichy. The worst-case GMRES for normal matrices. *BIT*, 44:79–98, 2004.

[153] P.T. Lin. Improving multigrid performance for unstructured mesh drift-diffusion simulatioins on 147,000 cores. *International Journal for Numerical Methods in Engineering*, 91:971–989, 2012.

[154] J.W.H. Liu. The multifrontal method for sparse matrix solution: theory and practice. *SIAM Review*, 34:82–109, 1992.

[155] J.S. Lowengrub, H.B. Frieboes, Y-L. Chuang F. Jin, X. Li, P. Macklin, S.M. Wise, and V. Cristini. Nonlinear modelling of cancer: Bridging the gap between cells and tumours. *Nonlinearity*, 23:R1–R91, 2010.

[156] D.G. Luenberger. *Introduction to linear and nonlinear programming*. Addison-Wesley, New York, 1973.

[157] P. Macklin and J. Lowengrub. Nonlinear simulation of the effect of microenvironment on tumor growth. *Journal of Theoretical Biology*, 245:677–704, 2007.

[158] P. Macklin and J.S. Lowengrub. A new ghost cell/level set method for moving boundary problems: application to tumor growth. *Journal of Scientific Computing*, 35:266–299, 2008.

[159] P. Macklin, S. McDougall, A.R.A. Anderson, M.A.J. Chaplain, V. Cristini, and J. Lowengrub. Multiscale modelling and nonlinear simulation of vascular tumour growth. *Journal of Mathematical Biology*, 58:765–798, 2009.

[160] P. MacNeice, K.M. Olson, C. Mobarry, R. deFainchtein, , and C. Packer. PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, 126:330–354, 2000.

[161] A.P. Mahmoudzadeh and N.H. Kashou. Evaluation of interpolation effects on upsampling and accuracy of cost functions-based optimized automatic image registration. *International Journal of Biomedical Imaging*, 2013, 2013.

[162] H. Majidian. Numerical approximation of highly oscillatory integrals on semi-finite intervals by steepest descent methods. *Numerical Algorithms*, 63:537–548, 2013.

[163] W.V. Mayneord. On a law of growth of Jensen's rat sarcoma. *American Journal of Cancer*, 16:841–846, 1932.

[164] S. McCormick. *Multilevel adaptive methods for partial differential equations, volume 6 of Frontiers in Applied Mathematics*. SIAM, Philadelphia, 1989.

[165] S.R. McDougall, A.R.A. Anderson, and M.A.J. Chaplain. Mathematical modelling of dynamic adaptive tumour-induced angiogenesis: Clinical implications and therapeutic targeting strategies. *Journal of Theoretical Biology*, 241:564–589, 2006.

[166] *MPI official website*, accessed on 24nd Jan 2014. http://www.mcs.anl.gov/research/projects/mpi/.

[167] N. Moelans, B. Blanpain, and P. Wollants. An introduction to phase-field modeling of micro structure evolution. *Computer Coupling of Phase Diagrams and Thermochemistry*, 32:268–294, 2008.

[168] P.K. Moore and L. Petzold. A stepsize control strategy for stiff systems of ordinary differential-equations. *Applied Numerical Mathematics*, 15, 1994.

[169] *MUMPS website*, accessed on 8th Apr 2014. http://mumps.enseeiht.fr/index.php.

[170] A.E. Naiman, I.M. Babuka, and H.C. Elman. A note on conjugate gradient convergence. *Numerische Mathematik*, 76:209–230, 1997.

[171] K. Olson. PARAMESH: A parallel adaptive grid tool. in Parallel Computational Fluid Dynamics 2005: Theory and Applications: Proceedings of the Parallel CFD Conference, College Park, MD, U.S.A., 2006.

[172] K. Olson and P. Macneice. An overview of the PARAMESH AMR software and some of its applications. in Adaptive Mesh Refinement-Theory and Applications, Proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, Series: Lecture Notes in Computational Science and Engineering, 2005.

[173] S.E. Orchard. On surface levelling in viscous liquids and gels. *Applied Scientific Research*, 11:451, 1962.

[174] A. Oron, S.H. Davis, and S.G. Bankoff. Long-scale evolution of thin liquid films. *Reviews of Modern Physics*, 69:931–980, 1997.

[175] J.M. Ortega and W.C. Rheinbolt. *Iterative solution of non-linear equations in several variables*. Academic Press, 1970.

[176] S. Osher and R.P. Fedkiw. Level set methods: an overview and some recent results. *Journal of Computational Physics*, 169, 2001.

[177] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. Lefohn, and T.J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26:80–113, 2007.

[178] P. Pacheco. *An introduction to parallel programming*. Morgan Kaufmann, 2011.

[179] *PARAMESH source code*, accessed on 12th Apr 2014. http://sourceforge.net/projects/paramesh/.

[180] *PARAMESH website*, accessed on 12th Apr 2014. http://www.physics.drexel.edu/ olson/parameshdoc/Users_manual/amr.html.

[181] F. Pellegrini. *Scotch and libscotch 5.0 user's guide*. Technical report, LaBRI, University Bordeaux, 2007.

[182] K. Pham, H.B. Frieboes, V. Cristini, and J. Lowengrub. Predictions of tumour morphological stability and evaluation against experimental observations. *Journal of the Royal Society Interface*, 8:16–29, 2011.

[183] H. Raach and S. Somasundaram. Numerical investigations on heat transfer in falling films around turbulence wires. *5th European Thermal-Sciences Conference, The Netherlands*, 2008.

[184] J.C. Ramirez, C. Beckermann, A. Karma, and H.J. Diepers. Phase-field modeling of binary alloy solidification with coupled heat and solute diffusion. *Physical Review E*, 69, 2004.

[185] Singiresu S. Rao. *The Finite Element Method In Engineering*. Elsevier Butterworth-Heinemann, 2005.

[186] T. Rauber and G. Runger. *Parallel programming: for multicore and cluster systems*. Springer, 2010.

[187] J.N. Reddy. *An Introduction to the Finite Element Method*. McGraw-Hill Education, 2005.

[188] K. Rektorys. *Solving Ordinary and Partial Boundary Value Problems*. CRC Press LLC, 1999.

[189] J. Rosam. *A Fully implicit, Fully adaptive multigrid method for multiscale phase-field modelling*. PhD thesis, School of Computing, University of Leeds, 2007.

[190] J. Rosam, P.K. Jimack, and A.M. Mullis. An adaptive, fully implicit multigrid phase-field model for the quantitative simulation of non-isothermal binary alloy solidification. *Acta Materialia*, 56:4559–4569, 2008.

[191] J. Rosam, P.K. Jimack, and A.M. Mullis. Quantitative phase-field modelling of solidification at high Lewis number. *Physical Review E*, 79, 2009.

[192] Y. Saad. *Iterative methods for sparse linear systems*. Society for industrial and Applied Mathematics, 2003.

[193] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM Journal on Scientific Computing*, 7:856–869, 1986.

[194] L.W. Schwartz. Hysteretic effects in droplet motion on heterogeneous substrates: direct numerical simulation. *Langmuir*, 14:3440–3453, 1998.

[195] L.W. Schwartz and R.R. Eley. Simulation of droplet motion on low-energy and heterogeneous surfaces. *Journal of Colloid and Interface Science*, 202:173–188, 1998.

[196] M. Sellier, Y.C. Lee, H.M. Thompson, and P.H. Gaskell. Thin film flow on surfaces containing arbitrary occlusions. *Computers & Fluids*, 38:171–182, 2009.

[197] B. Seny, J. Lambrechts, T. Toulorge, V. Legat, and J.F. Remacle. An efficient parallel implementation of explicit multirate Runge-Kutta schemes for discontinuous Galerkin compuatations. *Journal of Computational Physics*, 256:135–160, 2014.

[198] J.A. Sethian. *Level set methods and fast marching methods*. Cambridge University Press, 1999.

[199] X.H. Shi, H.J. Bao, and K. Zhou. Out-of-Core multigrid slover for streaming meshes. *ACM Transactions on Graphics*, 28:173–180, 2009.

[200] J. Shin, S. Kim, D. Lee, and J. Kim. A parallel multigrid method of the Cahn-Hilliard equation. *Computational Materials Science*, 71:89–96, 2013.

[201] A. Shinozaki and Y. Oono. Spinodal decomposition in a Hele-Shaw cell. *Physical Review*, 45:R2161–R2164, 1992.

[202] S.McCormick and J. Thomas. The fast adaptive composite grid (FAC) method for elliptic equations. *Mathematics of Computation*, 46:439–456, 1986.

[203] M. Smelyanskiy, D. Holmes, J. Chhugani, A. Larson, D. Carmean, D. Hanson, P. Dubey, K. Augustine, K. Kim, A. Kyker, V.W. Lee, A.D. Nguyen, L.Seiler, and R.A. Robb. Mapping high-fidelity volume rendering for medical imaging to CPU, GPU and many-core architectures. *Computer Graphics Forum*, 26:80–113, 2007.

[204] C.A. Smethurst, D.J. Silvester, and M.D. Mihajlovic. Unstructured finite element method for the solution of the Boussinesq problem in three dimensions. *International Journal for Numerical Methods in Fluids*, 73:791–812, 2013.

[205] G.D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford University Press, 1985.

[206] C.R. Snow. *Concurrent Programming*. Cambridge University Press, 1992.

[207] R.V. Southwell. *Relaxation Methods in Theoretical Physics*. Clarendon Press, 1946.

[208] G. Strang and G.J. Fix. *An Analysis Of The Finite Element Method*. Prentice-Hall, Inc., 1973.

[209] *TOP500 Supercomputer sites*, accessed on 22nd Jan 2014. http://www.top500.org.

[210] *SuperLU website*, accessed on 8th Apr 2014. http://crd-legacy.lbl.gov/ xiaoye/SuperLU.

[211] L.H. Tanner. The spreading of silicone oil drops on horizontal surfaces. *Journal of Physics D: Applied Physics*, 12:1473–1485, 1979.

[212] R. Tenchev, T. Gough, O.G. Harlen, P.K. Jimack, D.H. Klein, and M.A. Walkley. Three-dimensional finite element analysis of the flow of polymer melts. *Journal of Non-Newtonian Fluids*, 23:38–56, 2011.

[213] R. Tenchev, O.G. Harlen, P.K. Jimack, and M.A. Walkley. Finite element modelling of two- and three-dimensional viscoelastic polymer flows. *Trends in Engineering Computational Technology, Eds: M. Papadrakakis and B.H.V. Topping (Saxe-Coburg Publications, UK)*, pages 81–101, 2008.

[214] V. Thomee. From finite differences to finite elements, a short history of numerical analysis of partial differential equations. *Journal of Computational and Applied Mathematics*, 128:1–54, 2001.

[215] R.H. Thomlinson and L.H. Gray. The histological structure of some human lung cancers and the possible implications of radiotherapy. *British Journal of Cancer*, 9:539–549, 1955.

[216] U. Trottenberg, C.Oosterlee, and A.Schuller. *Multigrid*. Academic Press, 2001.

[217] D.A. Venditti and D.L. Darmofal. Grid adaptation for functional outputs: applications to two-dimensional inviscid flows. *Journal of Computational Physics*, 176:40–69, 2002.

[218] H.K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics, the finite volume method*. Longman, 1995.

[219] C. Walshaw. *The parallel JOSTLE library user guide: Version 3.0*. 2000.

[220] C. Walshaw and M. Cross. Parallel optimisation algorithms for multilevel mesh partitioning. *Parallel Computing*, 26:1635–1660, 2000.

[221] C. Wang, X. Wang, and S. Wise. Unconditionally stable schemes for equations of thin film epitaxy. *Discrete and Continuous Dynamical Systems*, 28:405–423, 2010.

[222] B. Wilkinson and M. Allen. *Parallel programming techniques and applications using networked workstations and parallel computers*. Pearson Prentice Hall, 2005.

[223] S. Wise, J. Kim, and J. Lowengrub. Solving the regularized, strongly anisotropic Cahn-Hilliard equation by an adaptive nonlinear multigrid method. *Journal of Computational Physics*, 226:414–446, 2007.

[224] S. Wise, C. Wang, and J. Lowengrub. An energy stable and convergent finite-difference scheme for the phase field crystal equation. *SIAM Journal on Numerical Analysis*, 47:2269–2288, 2009.

[225] S.M. Wise. Unconditionally stable finite difference, nonlinear multigrid simulation of the Cahn-Hilliard-Hele-Shaw system of equations. *Journal of Scientific Computing*, 44:38–68, 2010.

[226] S.M. Wise, J.S. Lowengrub, and V. Cristini. An adaptive multigrid algorithm for simulating solid tumor growth using mixture models. *Mathematical and Computer Modelling*, 53:1–20, 2011.

[227] S.M. Wise, J.S. Lowengrub, H.B. Frieboes, and V. Cristini. Three-dimensional multispecies nonlinear tumor growth - I. model and numerical method. *Journal of Theoretical Biology*, 253:524–543, 2008.

[228] D.M. Young. *Iterative solution of large linear systems*. Academic Press, New York, 1971.

[229] X. Zhang, S.M. Wise, and V. Cristini. Nonlinear simulation of tumor necrosis, neovascularization and tissue invasion via an adaptive finite-element/level-set method. *Bulletin of Mathematical Biology*, 67:211–259, 2005.

[230] L. Zheng, H. Zhang, T. Gerya, M. Knepley, D.A. Yuen, and Y.L. Shi. Implementation of a multigrid solver on a GPU for Stokes equations with strongly variable viscosity based on Matlab and CUDA. *International Journal of High Performance Computing Applications*, 28:50–60, 2014.

[231] O.C. Zienkiewicz and R.L. Taylor. *The finite element method. Volume 1 the basis*. Butterworth-Heinemann, 2000.