applied sciences

MDPI

Review

Mathematical Tools for Simulation of 3D Bioprinting Processes on High-Performance Computing Resources: The State of the Art

Luisa Carracciuolo * and Ugo D'Amora

The Institute of Polymers, Composites and Biomaterials (IPCB) of the National Research Council (CNR), 80078 Pozzuoli, NA, Italy; ugo.damora@cnr.it

* Correspondence: luisa.carracciuolo@cnr.it

Abstract: Three-dimensional (3D) bioprinting belongs to the wide family of additive manufacturing techniques and employs cell-laden biomaterials. In particular, these materials, named *"bioink"*, are based on cytocompatible hydrogel compositions. To be printable, a bioink must have certain characteristics before, during, and after the printing process. These characteristics include achievable structural resolution, shape fidelity, and cell survival. In previous centuries, scientists have created mathematical models to understand how physical systems function. Only recently, with the quick progress of computational capabilities, high-fidelity and high-efficiency *"computational simulation"* tools have been developed based on such models and used as a proxy for real-world learning. Computational science, or "in silico" experimentation, is the term for this novel strategy that supplements pure theory and experiment. Moreover, a certain level of complexity characterizes the architecture of contemporary powerful computational resources, known as *high-performance computing (HPC)* resources, also due to the great heterogeneity of its structure. Lately, scientists and engineers have begun to develop and use computational models more extensively to also better understand the bioprinting process, rather than solely relying on experimental research, due to the large number of possible combinations of geometrical parameters and material properties, as well as the abundance of available bioprinting methods. This requires a new effort in designing and implementing computational tools capable of efficiently and effectively exploiting the potential of new HPC computing systems available in the Exascale Era. The final goal of this work is to offer an overview of the models, methods, and techniques that can be used for "in silico" experimentation of the physicochemical processes underlying the process of 3D bioprinting of cell-laden materials thanks to the use of up-to-date HPC resources.

Keywords: computational simulation; 3D bioprinting; high-performance computing; cell-laden hydrogels

Citation: Carracciuolo, L.; D'Amora, U. Mathematical Tools for Simulation of 3D Bioprinting Processes on High-Performance Computing Resources: The State of the Art. *Appl. Sci.* **2024**, *14*, 6110. https://doi.org/10.3390/app14146110

Academic Editor: Juan A. Gómez-Pulido

Received: 23 May 2024
Revised: 5 July 2024
Accepted: 10 July 2024
Published: 13 July 2024

1. Introduction

In recent years, great interest has been devoted to three-dimensional (3D) bioprinting, which employs a numerically controlled dispensing system to deposit cells and biomaterial inks, named bioink, in a well-defined and controlled way. A wide number of papers have shown how beneficial it could be in the creation of smart patches for wound healing or thick tissue constructs, named scaffolds, for bone and cartilage tissue engineering [1–4]. Despite this, there are numerous difficulties with bioprinting, including the inability of cell-laden biomaterials to be printed, the severe circumstances that cells must endure while being printed, and the development of the bioprinted scaffold [5]. What you print in bioprinting is not necessarily what you obtain; natural cell rearrangements may alter the system, frequently for the better. Indeed, bioprinting is complicated at every stage of the process, from pre-processing to processing to post-processing.

Numerous geometrical characteristics and tunable material properties have been shown experimentally to have a clear and significant impact on cell viability and the

appropriate creation of the intended tissue structure [6]. Considering the great number of possible combinations of geometrical constraints and material properties, as well as the great number of bioprinting techniques that are currently available, it can be crucial to create and apply computer models to gain a deeper knowledge of the bioprinting process, rather than exclusively depending on experimental research [7]. Indeed, in silico experiments like computational research and digital fabrication are thought to be important sources of innovation and to have a significant impact on the emerging field of 3D bioprinting and tissue engineering. While producing more accurate and realistic virtual experiments is still a difficulty, this can be overcome with a variety of strategies, including statistical tools and methodologies coupled with first-principles-based simulations (The contribution of Bradley et al. [8], with the terms *"first principles models"*, intend to represent models derived from fundamental laws of physics, chemistry, thermodynamics, kinetics, and transport phenomena, such as mass balance and energy balances. These models can take many different forms, such as a system of ordinary or partial differential equations (ODE or PDE), a system of fundamental algebraic equations, or a mix of the two to create a general nonlinear algebraic partial differential system of equations. Although the term *"first principles models"* was primarily used to represent methods for systems with continuous dynamics, systems with discrete or non-continuous relationships, as well as stochastic systems, can be represented by them as well [8]. On the contrary, purely data-driven models are distinct from first principles models in that their parameters are *"fitted"* according to the available data. Empirical models are another name for data-driven models. The data may originate from historical records, physical experiments, samples from simulations or first-principles models, or any combination of the aforementioned sources [8]). It is relevant to mention that computational science (which is the basis for any "in silico" experimentation) is a highly interdisciplinary discipline that requires the contribution of experts from the field of interest, mathematicians, and computer scientists [9] who jointly define the descriptive mathematical model of the problem, and implement, using appropriate tools, algorithms, and software, its numerical model (see Section 2 for a more complete description of the computational science (CS) process).

Numerous research papers have examined the effects of various hydrogel-based materials on cellular survival, *"printability"* (see [10] for a definition of the term), and shape fidelity [11], due to their significant role in the 3D bioprinting process. In fact, printability and shape accuracy are the most critical factors in bioprinting [12] since the printed construct should mimic the exact structure and architecture of the natural tissue.

In ref. [13], in the context of bone tissue engineering, a proof-of-concept study that integrates computationally guided design with customized bioprinting of cells and biomaterial carriers is reported. This work intends to be the necessary premise to the definition of a protocol for the *"automatic"* determination of the characteristics of *"biological"* inks fundamental to the success of scaffolds created through 3D printing: the characteristics of these bioinks must guarantee the mechanical features necessary to the printing process and at the same time the survival of the cells, suitably embedded into the ink, at every stage of the scaffold's life.

Since each of the aforementioned stages is characterized by particular needs of the bioink [14], it was preferred to classify the above methods according to their applicability in each of the following phases:

Pre-printing

In the print cartridge, a cell-laden bioink can be considered a composite material. Therefore, even in the absence of different compounds such as rheological enhancers or additional biomaterials, high-cell-density suspensions behave as colloidal systems that exhibit printability. The impact of the cells embedding on the viscoelastic properties of the bioink is further complicated by the possibility that the cells are surrounded by a pericellular matrix, which could alter their mechanical characteristics, the hydrodynamic radius, and boundary conditions at the fluid interface [15].

It is worth noting that cells can interact with each other, during the pre-printing phase and during the whole process, adding a further level of complexity to the overall system. Furthermore, different cell sources can be also considered when dealing with interface tissues such osteochondral one, which involves bone and cartilage tissues. Nonetheless, a bioink must prevent cell sedimentation to preserve a uniform cell suspension [16]. The cells in the ink are no longer evenly distributed when they settle. This may result in clogged nozzles as well as an uneven cell distribution in the finished printed structure (i.e., more cells in the early printed layers than in the latter ones). Creating bigger and more complex scaffolds may exacerbate the concerns related to cell sedimentation because these structures usually demand longer printing times (i.e., printing full-scale tissues or organs can take hours or even days). In conclusion, since cells can flow when a force is applied, a bioink would avoid cell sedimentation while yet remaining printable [14].

Additionally, the volume occupied by the cells within a bioink varies according to their size and density. The hydrogel is excluded from the volume occupied by cells, which could affect the physicochemical and viscoelastic properties [15]. For example, cells may actually hinder the cross-linking process by limiting contact between reacting groups or acting as a physical barrier between various ink layers.

During Printing

This indicates that as a force is applied, their viscosity alters. Non-Newtonian fluids are primarily categorized into shear thickening (viscosity increases when shear rate increases) and shear thinning (viscosity reduces when shear rate increases) in response to this viscous tendency. Generally speaking, printability increases with material viscosity—at least to the point at which the internal pressures created can harm cells. Furthermore, some studies examine the impact of temperature on the viscosity of bioink. Nonetheless, they must be bioprinted at physiological conditions at 37° C to preserve the cells [11]. Additionally, as a result of cell migration and proliferation, the distribution of cells during printing may change, which may have an impact on the bioink's rheological characteristics. In fact, as cells interact with the matrix around them and with one other, traction forces are created that operate on the hydrogel macromolecules in a printed structure.

Post-Printing

Bioink needs to produce a 3D *"milieu"* that promotes cellular survival and function after printing. Because they offer an aqueous, cell-compatible environment that can mimic many of the mechanical and biochemical characteristics of the original tissue, hydrogels are frequently utilized in 3D cell culture. The simple passage of nutrients and waste products to and from encapsulated cells is made possible by their high water content and permeability. They can also provide a variety of signals to control cell phenotypic, differentiation, growth, and migration.

The goal of this scoping review is to provide an overview of the models, methods and techniques that can be used for "in silico" experimentation (or computational simulation) of the physico/chemical processes underlying the process of 3D printing of cell-laden materials [14,17] on high-performance computing (HPC) systems.

Although bioprinting for tissue engineering has made significant progress, additional research is required to determine the best bioink compositions and process parameters to create repeatable procedures that will result in the production of more functional tissue-engineered scaffolds. More knowledge is needed about the interplay between fluid flow, nutrient or oxygen distribution, cell–cell interaction, and flow shear stress in in vitro tissue morphogenesis. In this scenario, computational and mathematical modeling can be very helpful in explaining the intricate underlying mechanisms and facilitating the rapid and effective comparison of different conditions. Moreover, advances in "in silico" experimentation largely rely on concurrent advances in algorithms, software, and hardware needed to obtain

high-performance levels for computational models. So many challenges related to the effective use of advanced computing platforms of the Exascale Era should be faced.

To this aim, this review paper proposes itself as a guide for future research to capitalize on the potential of mathematical models to find improved answers to the unmet clinical needs of modern society. The review work intends to answer the following research question:

> *"What are the relevant mathematical tools for the simulation of the different phases of 3D bioprinting with a particular focus on the first principles-based (FP) models in macro and mesa-scales and how such tools should be modified/integrated to be compliant with the emerging computational resource in the Exascale Era?"*.

Literature searches were conducted in three electronic databases (PubMed, Google Scholar, and Web of Science). The following search keys were used for the literature search:

`(Sub-Search-1) AND (Sub-Search-2) AND (Sub-Search-3) AND (Sub-Search-4) AND (Sub-Search-5)`

where

`Sub-Search-1:` `(bioprinting OR (cell-laden hydrogels) OR bioinks)`
`Sub-Search-2:` `(simulation OR modeling OR (computational science))`
`Sub-Search-3:` `(macro or mesa scale FP models)`
`Sub-Search-4:` `(numerical tools for computational solution of FP-model problems)`
`Sub-Search-5:` `(Exascale models and algorithms).`

The review is therefore organized as follows: in Section 2 a description of how computational science (CS) allows us to perform "in silico" experiments will be provided; Section 3 will give an inventory of the mathematical tools used in the literature for the computational simulation of the processes active in each of the above-mentioned phases (a description of the literature that is the result of the search `[(Sub-Search-1) AND (Sub-Search-2)]`); Section 4 will report a description of the "state of the art" of HPC systems in the Exascale Era; Sections 5 and 6 will give a deeper insight into both models and numerical tools used in the literature (a description of the literature that is the result of the searches `(Sub-Search-3)` and `(Sub-Search-4)` respectively); Section 7 will describe the most advanced results in developing and implementing such mathematical tools on HPC systems in the Exascale Era (a description of the literature that is the result of the search `(Sub-Search-5)`); the final part of this paper (Section 8) will summarize the aims, results, potential usefulness of the work, and the open challenges and will highlight the future perspectives of the field in the emergent realm of biofabrication and process optimization thanks to approaches based on models that *"learn from data"*.

For quick access to the transversal contents and to the description of the mathematical tools that are the aim of the research question related to this work, we present Table 1.

Table 1. An overview of the mathematical tools suggested in this review work that we included as relevant in consideration of our literature search.

Transversal Contents			
Introduction on Computational Science (Section 2) **"State of the Art" of HPC Systems (Section 4)**			
Inventory of Mathematiacl Tools for Computational Simulation of Bioprinting Processes (Section 3)			
Models	**Numerical Tool**		**Exascale Numerical Tool**
	Discretization	**Algorithms**	
Fluid Flows Equations or coupled continuos equations (Two-Phase flow problem, transport and response of species, Section 5.1) Cellular Particle Dynamics (Section 5.2.2)	FEM, FDM, FVM (Section 6.1.1) Cellular Particle Dynamics Algorithm (Section 6.2.2)	• Time Steppers (point 1 of Section 6.1.2) • Non-linear problem solvers (point 2 of Section 6.1.2) • Linear problem solvers (point 3 of Section 6.1.2) • The Saddle Point Problem solvers (point 4 of Section 6.1.2)	• General consideration on models suitable for Exascale (Section 7.1) • Parallel Time Steppers (Section 7.2.1) • Composite Non Linear Solver (Section 7.2.2) • Parallel Iterative and Direct Linear Solver (Section 7.2.3) • Fast Multipole Methods and "Hierarchical" matrices (Section 7.2.4)
Monte Carlo approach (Section 5.2.1)		The Markov Chain Monte Carlo Algorithms (Section 6.2.1)	Parallel Monte Carlo based methods (Section 7.2.5)
The Cellular Automata Model (Section 5.2.3)		The Cellular Automata Model Algorithms (Section 6.2.3)	General considerations on stochastic-based algorithms (Section 7.2)

2. The Role of Computational Science (CS)

Mathematical models have traditionally been created by theorists to better understand how physical systems function. But in the last several decades, there has been a tremendous advancement in processing power that has allowed for the creation of high-fidelity simulation software based on such models, used as a proxy for learning about the real world. We call this new method CS, which is a supplement to pure theory and experiment. Deep knowledge of mathematical modeling, numerical analysis, software engineering, high-performance computing, statistics, and a thorough comprehension of the technological application domain being investigated are essential for successful CS research. This is therefore a highly interdisciplinary work that requires the cooperation of application scientists, mathematicians, statisticians, and computer scientists [18], all acting in an integrated and interdependent ecosystem.

Figure 1 represents the process on which CS is based:

1. Field experts and mathematicians jointly define the descriptive mathematical model of the problem.
2. Mathematicians and computer scientists jointly define and implement the descriptive numerical model of the problem. The numerical model is described by an algorithm that can have a high computational cost due to the great number of operations. Therefore, it may be desirable to exploit the parallelism made available by resources for HPC [19–25].
3. Field experts, mathematicians, and computer scientists jointly validate the correctness and accuracy of the numerical model and algorithm by executing its implementing software. Parallel algorithms and software are evaluated on the basis of their performance in terms of the number of performed operations per time unit, strong and weak scalability, and other useful metrics [26].

4. If necessary, on the basis of the observations collected in the previous steps, a new formulation of the mathematical model is constructed.

In the CS process, each step has an impact on all the other steps in a loop where single-field experts can contribute just a small part (see the right part of Figure 1). Research endeavors may yield the dissemination of precisely designed software, empowering members of the wider scientific community to conduct their research based on the CS process [27,28].

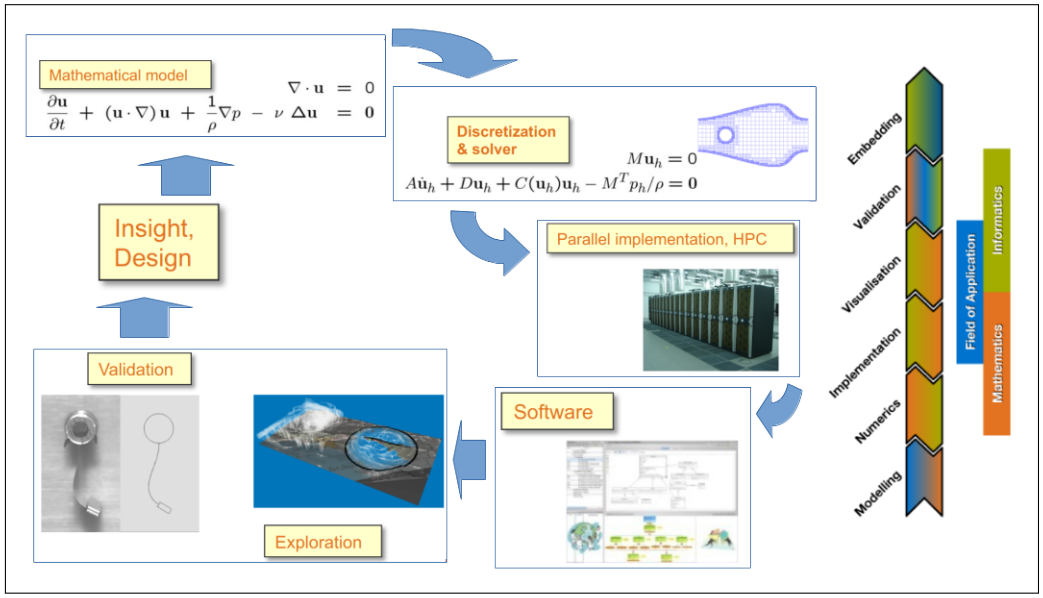


Figure 1. The process of Computational Science. Image realized on the basis of materials from The German Priority Programme "Software for Exascale Computing" (SPPEXA) [29].

3. An Inventory of the "In Silico" Experiments in 3D-Bioprinting of Cell-Laden Hydrogels

In past centuries, scientists have created mathematical models to understand how physical systems function. With the quick progress of computational capabilities, high-fidelity and high-efficiency *"computational simulation"* tools have been developed based on such models and used as a proxy for real-world learning.

Scientists and engineers in the context of material science have begun to develop and use computational models more extensively to better understand also the bioprinting process, rather than solely relying on experimental research, due to the large number of possible combinations of geometrical parameters and material properties, as well as the abundance of available bioprinting methods.

Figure 2 shows the logical structure graph of the relationships between 3D bioprinting computational simulation and concerned technologies and tools. In particular, bioprinting techniques could help find answers to some clinical needs of modern society, such as the regeneration of bone tissue using scaffolds or the "in vitro" study of the proliferation of tumor cells. Due to the enormous complexity of these techniques, experimental knowledge of the physical/chemical phenomena associated with them alone is not sufficient to fully understand their dynamics. Computational simulation is now widely recognized as a very powerful tool for broadening this understanding. Computational simulation obviously cannot exist without an effective investment in research on mathematical models and algorithms capable of guaranteeing effectiveness and efficiency in the solution of the involved numerical problems.

This section gives an inventory of the numerical approaches used in the literature for computational simulation of the processes active in each phase of the bioprinting process. Table 2 reports a purview of the papers described in the following that used the most

representative mathematical models based on the "first principles" approach to predict the different phases of the bioprinting process.

Table 2. Purview of the papers highlighting the most representative mathematical models based on the "first principles" approach used to describe the different phases of the bioprinting process.

	Model	Aim	Ref.
Pre Printing	Fluid Flows Equations	Evaluate the influence of a seeding loading regime on suction pressure and infiltration velocity of the cell suspension.	[30]
		Introduce a new methodology to simulate the cell-seeding process under perfusion conditions. The cells are treated as spherical particles dragged by the fluid media.	[31]
		Design a model describing the fusion of growing cell clusters, such as embryo cell aggregates or tumor cell spheroids.	[32]
During Printing	Fluid Flows Equations	Used to study the influence of needle geometries on cell clogging during printing.	[33]
		Determine how printing process parameters affect the flow velocity of cell-laden printing material and the survivability of cells within printed structures.	[34]
		Aid the design of an extruder system for the biofabrication of vascular networks with the goal of maintaining cell viability throughout the printing process.	[35]
	Monte Carlo approach	Used to study cellular adhesiveness and motility, interactions with concentration fields, and concentration-dependent, stochastic cell dynamics, driven by metabolite-dependent cell death.	[36]
	Immersed Boundary Method	Used to study cell deformation during the extrusion/ injection-based bioprinting processes.	[37]
	Two-Phase Level Set Methods	Used to study droplet formation.	[38,39]
		Computational simulations of bioink printing process in suspension baths.	[40]
		Used to assess the influence of bioink temperature on shear stress, pressure, and velocity.	[11]
	Models for transport and response of biological and chemical species	Optimize printing process parameters in order to achieve the required cell dispersion and maintain cell viability.	[41]
		Investigate the distribution of oxygen and glucose in bioprinted structures, as well as examine the reality of the diffusion and consumption phenomena and their impact on the growth and demise of cells.	[42]
	Surface Tension Model	Study the filament deposition and the final bioprinted shape during the extrusion/ injection-based bioprinting processes.	[43]

Table 2. *Cont.*

	Model	Aim	Ref.
Post Printing	Monte Carlo approach	Study cell self-assembly and cellular aggregate fusion of multicellular aggregate systems.	[44]
		Study early morphogenesis of 3D tissue structures formed through the post-printing fusion of the bioink particles,	[45]
	Fluid Flows Equations	Study post-printing cell rearrangement to forecast the geometry and stability of printed structures	[46]
	Cellular Particle Dynamics	Study the discrete bioink units' ability to self-assemble into the intended form after deposition and their deposition rate.	[47–49]
	Cellular Automata	Simulate post-printing cell behavior within the 3D bioprinted complex under a variety of bioprinting conditions.	[50]

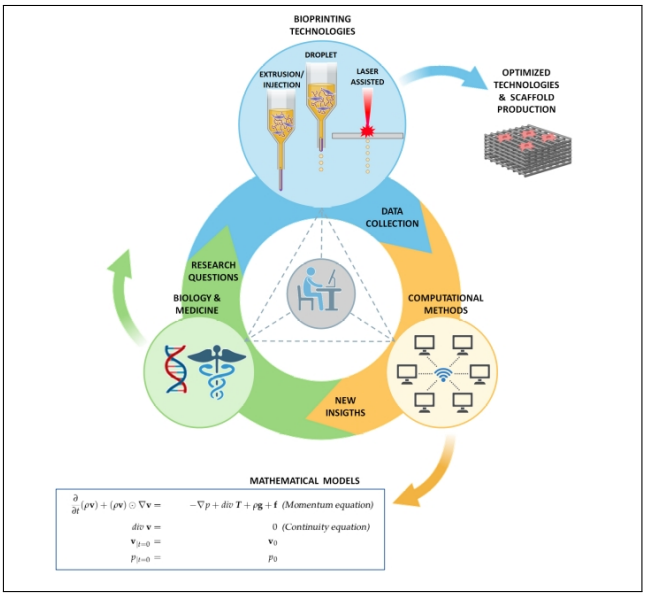


Figure 2. Graph of the logical structure of the relationships between 3D bioprinting computational simulation and concerned technologies and tools.

3.1. Pre-Printing

A crucial stage in biofabrication process is cell seeding. Using computational methodologies in conjunction with experimental activities, efficient cell seeding strategies are designed. For example, the relative impact of permeability and thickness, and coating on cell seeding efficiency and homogeneity was evaluated in [30] using a fully factorial experimental design, in conjunction with the Finite Elements Method (FEM) discretization of fluid flows equations, to predict the scaffold permeability effects on cell seeding effectiveness. For another example, the number of cells and the seeding time were optimized using the simulation of cellular interactions with biomaterials, cells, and dynamic medium in a situation where cell seeding takes place under perfusion conditions [31].

Furthermore, the theoretical framework, based on tissue fluidity, considers that cells tend to aggregate like droplets of an exceedingly viscous and incompressible fluid. In the mathematical model, cell clusters are assumed to have a size lower than 0.5 mm. Otherwise, gas and nutrients exchange are prevented, enabling the formation of necrotic areas [5]. A good agreement was found between continuum models of the coalescence of highly viscous liquid droplets and tissue cell cluster fusion studies. But there is a crucial

difference between very viscous inanimate fluids and cell populations: they do not follow the principle of volume conservation. Dechristé et al. designed a compressible viscoelastic fluid model describing the fusion of growing cell clusters, such as embryo cell aggregates or tumor cell spheroids [32]. The authors initially gave the whole partial differential system, which describes the fusing of expanding spheroid, based on the Stokes equation. The combined model included a viscous model representing the velocity field and a tumor growth model. It is interesting to note that both models produced similar results since the velocity field's divergence equals the rate at which volume changes. The authors reduced the complexity of the 3D model to a 0D model, which included an ordinary differential system, under the symmetry conditions. To show the consistency, the experiments were fitted to a numerical model. Similarly, Kuan et al. built a 2D continuum theory to model the aggregate formation in a system characterized by uniformly distributed cells, as well as the coalescence of aggregate doublets [51]. In particular, the authors explained the production of aggregates as an active-phase separation phenomenon, and the merging of aggregates as the coalescence of viscoelastic droplets, with the important timeframes associated with the active force's turnover. The theory offered a broad framework for investigating the nonequilibrium dynamics and rheology of dense cellular aggregates [52].

3.2. During Printing

By clarifying the intricate interactions between the many printing parameters and forecasting their ideal configurations for certain goals, mathematical model techniques have shown to be effective in streamlining the bioprinting process [53,54]. There may be several goals for the computational optimization of bioprinting procedures [55].

Maximizing printing fidelity—the degree of resemblance between the product models and the real bioprinted product—is the goal of some models. Alternative strategies aim to optimize biomimetic fidelity, which refers to the construct's biological, mechanical, and rheological resemblance to its in vivo equivalent. Alternative methods address the problem of simultaneously optimizing printing and the biomimetic fidelity in bioprinting. For instance, to assess the significance of different process parameters, such as ultraviolet (UV) intensity and UV exposure time, on cell viability in stereolithography-based 3D bioprinting, Xu et al. [56] successfully developed a predictive model using computational methods based on data to anticipate cell viability with good sensitivity. Several investigations have also attempted to create various methods to enhance the printability of bioinks. Using multiple regression analysis, Lee J. et al. showed how printability and the ink's mechanical characteristics relate to one another. Nevertheless, depending on the goal of the study and the data-collection procedure, this approach highlighted certain limitations even with the significant advancements made in the application of the models in the biological sciences [57]. The biofabrication of high-resolution, fragile cell transfer of printed human-induced pluripotent stem cells (iPSC) using a low-cost 3D printer for exact cell placement and stem cell development is based on computational analysis, which uses fluid flow equations and it allows to tune flow parameters to improve needle geometry [33]. Simulations for studying cell deformation during the extrusion/injection-based bioprinting processes are described in [37]. Those simulations are based on the Immersed Boundary Method (IBM) [58]. Some authors [38,39] used the Two-Phase Level Set Methods (TPLSMs) [59,60] to study droplet formation.

Sego et al. used computational methods to predict the spheroids' viability under different process parameters, during biofabrication prediction. The authors developed a discrete–continuous heuristic model that predicts metabolic effects over cells during spheroid-dependent construction combining field equations with a cellular Potts-type method, which is based on a Monte Carlo approach [36].

To determine how process parameters affect the flow velocity of cell-laden printing material and the survivability of cells within printed structures, researchers have explored the experimental effects of needle geometry and air pressure, and appropriately, models, based on equations for viscoelastic fluids, have been created to mimic these findings [34].

The models can be used to optimize process parameters in order to achieve the required cell dispersion and maintain cell viability. It has also been shown that computational modeling, based on models for the transport and response of biological and chemical species, can be used to produce well-designed tissue engineering structures that enhance bone and tissue regeneration by employing a particular spatial layout of cells in a matrix [41]. In particular, compared to uniform structures, a cell-gradient pattern of cell-laden hydrogels with different cell densities produced greater cell survivability after printing.

Using TPLSM and Carreau constitutive viscosity models, Prendergast et al. reported computational simulations of bioink printing in suspension baths in [40]. With IBM and the Surface Tension Model [61], a further publication [43] examined the filament deposition and the final bioprinted shape. The linear "Phan-Thien and Tanner (PTT)" model provided a clear explanation of the viscoelastic bioink's rheology [62]. Employing a TPLSM-based simulation, Gomez-Blanco et al. assessed the influence of bioink temperature on shear stress, pressure, and velocity [11].

Meanwhile, the design of an extruder system for the biofabrication of vascular networks was aided by multiphysics computational models, based on fluid dynamics equations, with the goal of maintaining cell viability throughout the printing process [35].

Recently, Gironi et al. developed a computational model, based on models for the transport and response of biological and chemical species, to investigate the distribution of oxygen and glucose in bioprinted structures, as well as to examine the reality of the diffusion and consumption phenomena and their impact on the growth and demise of cells. The model's initial validation was carried out on straightforward droplet-shaped structures. The investigation of constructions with and without channels, varying in size and geometry, was conducted using the model. In order to create vascularized bioprinted tissues, more intricate bioprinted construct configurations with bigger sizes and hierarchical vascular networks will be studied computationally and physically [42].

However, a summary of the different numerical models used to investigate cell viability and shape fidelity during the printing process can be found in [7]. The goal was to use these data to develop a mathematical model that could forecast the state of the cell resulting from the generated shear stresses. Additionally, this could aid in the optimization of the printing procedure and explore the effects of process-induced mechanical effects on cell vitality (e.g., refer to Nair et al. [63]).

3.3. Post-Printing

Computational methods may allow also one to target the overall design of the biofabrication process by forecasting the characteristics and quality of the end product based on the stimuli that are delivered and how they are arranged in space and time [64]. Furthermore, they may allow the evaluation of the cell maturation after the printing phase.

In particular, over time, cells embedded into the bioink proliferate and grow, tending to form a tissue. This represents an important aspect of the bioprinting process. Computational modeling can be used to simulate the fusing of cellular aggregates in biofabrication [7]. For instance, cell self-assembly and the cellular aggregate fusion of multicellular aggregate systems were studied in [44] utilizing kinetic Monte Carlo (KMC) methods [65]. Specifically, KMC was used to anticipate how post-printed scaffolds' morphological characteristics would alter as tissue morphogenesis progressed. Predicted equilibrium tissue configurations were extrapolated from the interfacial tensions among the many cell types that comprise the bioprinted tissue. This was accomplished by applying a discrete multicellular lattice model that explained how cells interacted using the differential adhesion theory [44].

Computational techniques can be utilized not just to mimic the bioprinting process but also to investigate the mechanisms behind the fusing of multicellular structures. For example, post-printing cell rearrangement was predicted using the Lattice Boltzmann approach, which is based on fluid dynamics equations, to forecast the geometry and stability of printed structures [46]. A number of shapes were tested: a faulty hexagon, a square lattice that fuses into a structure resembling tissue, a hexagonal configuration of

cylinders that fuses into a cylindrical tube, and a sequence of cylinders that fuses into a planar construct. New working hypotheses can be evaluated using computer simulations more quickly and cheaply than in a lab [46]. Jakab et al. mimicked, by using a Monte Carlo approach, the early morphogenesis of 3D tissue structures formed through the post-printing fusion of the bioink particles, in analogy with early structure-forming processes in the embryo that utilize the apparent liquid-like behavior of tissues composed of motile and adhesive cells [45].

In ref. [47–49], the predictive power of the computational method, named Cellular Particle Dynamics (CPDs) [65,66], was demonstrated in cases of simple printed constructs prepared with spherical multicellular bioink units. The discrete bioink units' ability to self-assemble into the intended form after deposition and their deposition rate both affect the success of a bioprinting procedure. In particular, the creation of a tissue after printing is mainly guided by basic principles of biological organization. The development of biological structures in bioprinting, where mini-tissues are used as bioink, is characterized by their discrete fusion into multicellular spheroids or cylinders, which resembles the merging of liquid droplets. More specifically, the fusion of discrete bioink particles can be described by equations applicable to very viscous liquids.

Phase field theories were used to simulate the fusing of cellular aggregates in biofabrication in a different mathematical model, in which the surrounding hydrogel and cells were considered as a binary fluid combination consisting of two immiscible fluids [67]. In particular, the surrounding hydrogel ink and the cellular aggregates were modeled by considering a viscous fluid and a spheroid of complex fluids, respectively. A mean-field potential was then created, using a higher-order spectra model, by combining the long-range, attractive interactions between cells with the short-range, repulsive interactions resulting from the immiscibility. A two-phase model that can be used to study how active matter interacts with the surrounding viscoelastic medium was described in [68]. Moreover, an Oldroyd-B model was used to characterize the fluid rheology [62].

The development of a cellular automata model (CA) [69] to simulate post-printing cell behavior within the 3D bioprinted complex was first described in [50]. Preliminary "in vitro" studies were used to establish the criteria for cell proliferation, viability, and cluster formation in the CA model. In addition to quantitatively capturing the post-printing "in vitro" behavior of cells in the 3D scaffold, the CA model was able to predict and elucidate the behavior of the cells under a variety of bioprinting conditions. It was able to replicate, for example, the way that cells move and proliferate inside the pores. Furthermore, the in-silico data demonstrated how cell proliferation is directly influenced by the initial cell density in the bioink as well as the number of starting cells.

4. A "State of the Art" of HPC Systems in the Exascale Era

Progress in the deployment of computing platforms has constituted both a pull and a push factor for the advancement of scientific knowledge and engineering design. Historically, the HPC systems era started in the 1980s, when vector supercomputing dominated high-performance computing, as embodied in systems designed by Cray Research Inc. [70,71]. The 1990s saw the rise of massively parallel processing (MPP) in distributed memory systems and shared memory multiprocessors (SMPs) [72]. In turn, clusters of commodity (Intel/AMD x86) [73] and purpose-built processors (such as IBM's BlueGene) [74] dominated the beginning of this Millenium.

Today, these clusters (which are made of hundreds of nodes and millions of processors/cores) are enriched by computational accelerators in the form of coprocessors, such as General Purpose Graphical Processing Units (GP-GPUs); they are based on high-speed, low-latency interconnects (such as Infiniband) (see Figure 3 for a representation of modern HPC systems) [71].

Parallel computers (in all their different architectural evolutions) made it possible to lay the foundations for overcoming the limits that, over the years, have slowed down the possibility of enhancing processor computational power [75]. Just a few years ago,

teraflops (10^{12} floating point operations/second) was the state-of-the-art of HPC technology. Today, those same values can be obtained by a simple personal computer with just one accelerator, so advanced computing is now defined by multiple petaflop ((10^{15}) floating operations/second) supercomputing systems. The exponential increase in advanced computing capability is outlined by the results of the well-known high-performance LINPACK (HPL) benchmark [76] which is used to compile the Top 500 list of the world's fastest computers [77]. Such a list shows that the Exascale (10^{18} operations per second) Era is by now the current affairs in the long journey of performance increases that lasts for more than half a century. Indeed, in the second part of the 2022 year, the "Frontier" supercomputer installed at the US Oak Ridge National Laboratory broke the glass ceiling of the Exascale limit [78].

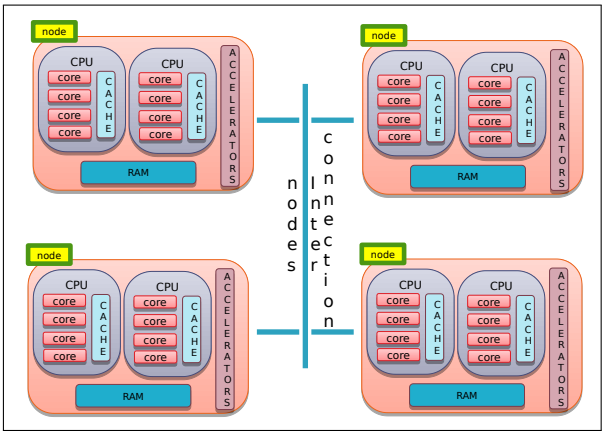


Figure 3. The hierarchical architecture of modern HPC systems. Several processing units (core/CPU) are aggregated in a CPU/node and share some memory devices. Access to "remote" memory devices on other nodes is performed thanks to an interconnection network. Memory is organized into levels, where access speed is inversely proportional to memory size and directly proportional to the memory "distance" from the processing units. Accelerators are a very particular type of processing unit with thousands of cores.

Historically, HPC advances largely relied on concurrent advances (in a sort of co-design approach) in algorithms, software, architecture, and hardware that led to higher performance levels for computational models. However, there are many challenges related to the effective use of advanced computing platforms of the Exascale Era, including, but not limited to, massive parallelism and high complexity in programming such heterogeneous computing facilities.

A series of studies identified research challenges in developing the generation of advanced computing systems for the Exascale objective [71,79,80]. From a computational scientist's point of view, particular relevance has challenges related to

- Application programming support, which, in response to the complexity and scale of advanced computing hardware, makes available programming languages, numerical libraries, and programming models, eventually combined in hybrid and hierarchical approaches, exploit the multi-node, multi-core, and accelerators-based architecture of up-to-date HPC resources.
- New mathematical models and algorithms that can give solutions to the need for increasing amounts of data locality and the need to obtain much higher levels of concurrency and that can guarantee a high level of *"scalability"* [81] and *"granularity"* [82]. Then, it is now the time to develop new algorithms that are more energy-aware, more resilient, and characterized by reduced synchronization and communication needs.

Therefore, such scientists, supported by an appropriate programming environment, should spend effort developing models, re-designing algorithms, and re-implementing applications to fully exploit the power of Exascale architectures.

In addition to the technical ones, opportunities offered by the Exascale Era also brought challenges related to organizational, cultural, and economic issues. Among them, the issues related to the specific need to educate new students and workers to ensure a sufficiently large and capable workforce to manage *"the Exascale transition"* are worth noting. Despite these challenges, all the scientific communities cannot avoid facing them since every advance in computing technology has driven industry innovation and economic growth, spurring research in every domain of science. Managing the innumerable technical, political, and economic challenges will not be easy. Still, these challenges could be more addressable if they were faced in coordinated planning across different academic, industry, and social contexts. All the above considerations are the basis of the multiple efforts spent by different communities all around the world to reach and consolidate the objectives related to the effective and efficient use of up-to-date advanced computing platforms during the Exascale Era (i.e., see the various projects funded all around the world [80,83–85] or the initiatives from the civil society [86]).

Such efforts reveal the strategic role of HPC and SC for the present and the future. Indeed, the HPC community is already starting to question itself on the future of post-Exascale computing. For example, the organizers of both the International Conference on High-Performance Computing 2024 and the International Conference on Parallel Processing 2023 asked Kathy Yelick, Vice Chancellor for Research at the University of California Berkeley, to deliver the opening keynote talk, titled *"Beyond Exascale Computing"* [87,88]. The Yelick talk summarizes the results from two reports [89,90] both on Scientific Computing in the Post-Exascale Era. About her statements about the future of SC, one cannot help but agree that we have *" ... to continue to rethink applications ..."* and thus the mathematical models and the algorithms.

5. Mathematical Models

CS may be used to describe the dynamic behavior of physicochemical systems using simulations and models. Mathematical models can be created at the three conventional scales: microscopic, macroscopic, and kinetic [91].

On the microscopic scale, the state of the system is made up of all of its component states and is solely a function of time. The dynamics are modeled using a large system of ordinary differential equations, each representing a variable at the microscopic level. At the macroscopic level, continuum models for systems whose state variable can be a vector, are created. This kind of variable, which depends on both space and time, is provided by locally averaged data. Partial differential equations derived from equilibrium or conservation rules characterize the dynamics within the elementary volume of the space of microscopic states. At the mesoscopic scale, kinetic theory models can be also defined for systems composed of numerous interacting units, each of whose unique states is referred to as a microscopic state. The dynamics are described by a differential system created by conservation relations within the elementary volume of the space of microscopic states [91]. Furthermore, mathematical models at different scales can also be combined to form a hybrid technique called a multiscale approach [92].

Although the idea of multiscale modeling impacts many domains, material modeling has only benefited greatly from it in recent decades. Since the physics at a finer scale is better understood than at a coarser size, multiscale modeling refers to formulating equations, parameters, or simulation techniques that characterize behavior at a particular length scale starting from information obtained at the finer scale [92]. Many of these techniques have been created, employing different strategies to bridge various lengths and time spans. For a summary of multiscale computational methods and representative examples, the reader may refer to Table 1 of the work by Fish et al. [92]. As examples, we provide below two indicative samples taken from that table:

Upscaling methods based on the same mathematical models: On different scales, the same (or similar) physics models are applied. The interscale coupling could be based

on different techniques such as asymptotic theory, wavelets, or macrohomogeneity conditions [92].

Resolved-scale methods based on a Multigrid approach: Separate physics models and discretization schemes are used in different spatial domains that can be simulated concurrently. Coupling strategies are based on interscale operators (restriction and prolongation). The solution algorithms are similar to iterative Multigrid matrix solvers [92].

The extensive treatment of the "multiscale approaches" goes beyond the scope of this work but could be the objective of future developments. In any case, for those wishing to delve deeper into the topic, they can refer to the interesting review by Fish et al. [92].

According to what has already been tested and described in the literature cited in Section 3, we will only focus on models used at both macro and mesa scales, neglecting models that work at micro-scales.

5.1. Models at the Macro-Scale

The proposed models are all based on Fluid Flows Equations that are described in Section 5.1.1. Such equations could be coupled with other ones to describe more complex problems such as *"The two-phase flow problem"* (see Section 5.1.2) or the simulation of *Transport and response of biological and chemical species* (see Section 5.1.3).

In Table 3 we provide a summary of *"Pros and Cons"* of the described models.

Table 3. Summary of *"Pros and Cons"* of the models at the macro-scale.

Model	*"Pros"*	*"Cons"*
The Navier–Stokes equations for Viscoelastic Fluids (Section 5.1.1)	The most comprehensive way to describe complex fluid dynamics.	The definition of model parameters could be very hard.
The Level Set Method (LSM) for the two-phase flow problem (Section 5.1.2)	LSM is based on a front-capturing approach that uses a scalar function on a stationary grid to capture the interface evolution. The ability to compute phenomena like bubble break-up and coalescence is made possible by the implicit interface capture.	The main difficulty in using this approach is keeping a clear boundary between the different fluids.
The Immersed Boundary Method (IBM) for the two-phase flow problem (Section 5.1.2)	IBM is based on a combination of front-capturing and front-tracking approaches. It can be useful to describe fluid–structure interaction when the structure is intended to be an immersed body.	IBM could inherit drawbacks of front-tracking approaches in describing interfaces with complex geometries. Moreover, the correct definition of the distribution function d could be crucial.
The continuum surface force (CSF) model for the two-phase flow problem (Section 5.1.2)	The probably most famous approach used to compute the surface tension force needed by the application of front capturing methods. The model considers surface tension force as a continuous, 3D effect across an interface, rather than as a boundary value condition on the interface.	The definition of the transition region and of the "color" function $\tilde{c}$.

Table 3. *Cont.*

Model	*"Pros"*	*"Cons"*
Transport and response of biological and chemical species (Section 5.1.3)	Models used to study how the tissue reacts to the chemical and mechanical environment in the culture medium. They can be considered a strict framework of conservation laws for mixes of interacting continua that describe the transfer of mass and momentum between different component phases.	The definition of model parameters could be very hard.

5.1.1. The Navier–Stokes equations for Viscoelastic Fluids

A fluid is said to be Newtonian if the local rate of change in its deformation over time is linearly proportional to the viscous stresses arising from its flow. On the other hand, non-Newtonian fluids defy this principle, and, for the most part, their viscosity depends on the present or past shear rate. Non-Newtonian fluids include blood, proteins, polymers, suspensions, emulsions, chemical reactants, and most other fluids found in both nature and industry. These fluids typically exhibit a wide range of remarkable properties, such as viscoelasticity, shear-thinning, and shear-thickening properties [93,94]. Shear-thinning (also called pseudoplastic) fluids are characterized by an apparent viscosity which decreases by increasing the shear rate. Conversely, shear-thickening fluids are those whose apparent viscosity rises as the shear rate decreases [95]. Finally, it should be noted that viscoelastic fluids exhibit time-dependent strain, a characteristic in which the fluids' strain gradually approaches the equilibrium value when stress is applied.

Navier–Stokes equations are used to describe the conservation of mass and momentum of incompressible fluid, varying in temporal domain Θ, in a spatial domain $\Omega \subseteq \Re^n$, $n = 1, 2, 3$ with boundary $\delta\Omega$ as the following problem:

Problem 1. *The Navier–Stokes equations defined in* $\Omega \times \Theta$

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + (\rho \mathbf{v}) \odot \nabla \mathbf{v} = -\nabla p + div\, \boldsymbol{T} + \rho \mathbf{g} + \mathbf{f} \quad \textit{(Momentum equation)} \tag{1}$$

$$div\, \mathbf{v} = 0 \quad \textit{(Continuity equation)} \tag{2}$$

$$\mathbf{v}_{|t=0} = \mathbf{v}_0 \tag{3}$$

$$p_{|t=0} = p_0 \tag{4}$$

where $\mathbf{v}$ *is the velocity field,* ρ *is the density of the fluid,* p *is the pressure field,* $\mathbf{g}$ *is the gravitational acceleration,* $\mathbf{f}$ *is an additional force field, and* $\boldsymbol{T}$ *is the stress response to the deformation of the fluid.*

An example for $\mathbf{f}$ is the *"Surface Tension"* force [96] in a two-phase problem [59] (see Section 5.1.2).

The notations $\nabla \mathbf{u}$, div $\mathbf{u}$, ∇f and $\mathbf{u} \odot \nabla \mathbf{u}$ respectively are used to representing:

$$\nabla \mathbf{u} = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \cdots & \frac{\partial u_n}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial u_1}{\partial x_n} & \cdots & \frac{\partial u_n}{\partial x_n} \end{bmatrix} \tag{5}$$

$$\text{div } \mathbf{u} = \sum_{i=1,\ldots,n} \frac{\partial u_i}{\partial x_i} \tag{6}$$

$$\operatorname{div}\mathbf{T} = \begin{bmatrix} \sum_{j=1,\ldots,n} \frac{\partial t_{1,j}}{\partial x_j} \\ \vdots \\ \sum_{j=1,\ldots,n} \frac{\partial t_{n,j}}{\partial x_j} \end{bmatrix} \tag{7}$$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{8}$$

$$\mathbf{u} \odot \nabla \mathbf{u} = \begin{bmatrix} \sum_{j=1,\ldots,n} u_j \frac{\partial u_1}{\partial x_j} \\ \vdots \\ \sum_{j=1,\ldots,n} u_j \frac{\partial u_n}{\partial x_j} \end{bmatrix} \tag{9}$$

where $f(x,t) \in \Re$, $\mathbf{u}(x,t) = [u_i(x,t)]^T_{i=1,\ldots,n} \in \Re^n$, $\mathbf{T}(x,t) = \left[t_{i,j}(x,t)\right]_{i,j=1,\ldots,n} \in \Re^{n\times n}$ and $x = (x_1,\ldots,x_n) \in \Re^n$

The solvent-polymer stress splitting technique is used in this formulation to decompose the stress response into two terms [93]: $\boldsymbol{T}_s$, which represents the solvent's instantaneous response, and $\boldsymbol{T}_p$, which represents the polymeric contribution to the stress response which accounts for the polymers' memory effects [97]:

$$\boldsymbol{T} = \boldsymbol{T}_s + \boldsymbol{T}_p \tag{10}$$

where the solvent part, $\boldsymbol{T}_s$, is defined as below:

$$\boldsymbol{T}_s = 2\mu_s(\dot{\gamma})\boldsymbol{D} \tag{11}$$

where

$$\boldsymbol{D} = \frac{1}{2}\left[\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right], \tag{12}$$

where $\dot{\gamma}$ is the shear rate defined by [98]

$$\dot{\gamma} = \sqrt{2}\|\boldsymbol{D}\|, \tag{13}$$

and where $\|\cdot\|$ denotes the Hilbert–Schmidt norm of a matrix $\boldsymbol{A}$ defined as

$$\|\boldsymbol{A}\| = \sqrt{\sum_{i,j=1,\ldots,n} a_{ij}^2}$$

The following viscosity models μ_s for generalized Newtonian fluids could be considered [93,99]:

Newtonian viscosity model

$$\mu_s^{(0)} = \mu_\infty^{(0)} = \mu_s,$$

Power-Law model

$$\mu_s^{(1)} = \mu_\infty^{(1)}\dot{\gamma}^{n-1},$$

Cross model

$$\mu_s^{(2)} = \mu_\infty^{(2)} + \left(\mu_0^{(2)} - \mu_\infty^{(2)}\right)\left[1 + (\lambda\dot{\gamma})^b\right]^{-1},$$

Modified Cross model

$$\mu_s^{(3)} = \mu_\infty^{(3)} + \left(\mu_0^{(3)} - \mu_\infty^{(3)}\right)\left[1 + (\lambda\dot{\gamma})^b\right]^{-a},$$

Carreau model

$$\mu_s^{(4)} = \mu_\infty^{(4)} + \left(\mu_0^{(4)} - \mu_\infty^{(4)}\right)\left[1 + (\lambda\dot{\gamma})^2\right]^{\frac{n-1}{2}},$$

Carreau–Yasuda model

$$\mu_s^{(5)} = \mu_\infty^{(5)} + \left(\mu_0^{(5)} - \mu_\infty^{(5)}\right)\left[1 + (\lambda\dot{\gamma})^m\right]^{\frac{n-1}{m}},$$

where $\mu_0^{(\theta)}$ and $\mu_\infty^{(\theta)}$ are the asymptotic viscosity values at zero and infinite shear rates (where $\theta = 0, 1, 2, 4, 5$). The parameters of the non-Newtonian viscosity models are a, b, m, and n, and the relaxation time is denoted by the symbol λ. The Power-Law model [93] may explain shear-thinning behavior for $n < 1$, shear-thickening behavior for $n > 1$, and Newtonian behavior for $m = 1$.

Several constitutive rheological models are available in the literature with polymeric stresses $\boldsymbol{T}_p$ [93,97], which are typically functions A number of constitutive rheological models with polymeric stresses $\boldsymbol{T}_p$ are available in the literature [93,97]. These stresses are usually functions $\boldsymbol{f}_S(\cdot)$ of the conformation tensor $\boldsymbol{C}$,

$$\boldsymbol{T}_p = \frac{\mu_p \boldsymbol{f}_S(\boldsymbol{C})}{\lambda} \tag{14}$$

where μ_p the polymeric viscosity.

According to [97], the conformation tensor $\boldsymbol{C} \in \Re^{n\times n}$ can be viewed as an internal state variable that describes the molecular deformation of the polymer chains. Assuming symmetry and positive definiteness, the conformation tensor C behaves in accordance with the following equation:

$$\overset{\triangledown}{\boldsymbol{C}} = -\frac{\boldsymbol{f}_R(\boldsymbol{C})}{\lambda} \tag{15}$$

where the relaxation function, which varies for each specific constitutive model, is $\boldsymbol{f}_R(\cdot)$. The operator *upper-convected derivative*, described in [97], is indicated by $\triangledown$. With the symbol $\bullet$, we intend to represent the matrix–matrix operation.

$$\overset{\triangledown}{\boldsymbol{C}} = \frac{\partial \boldsymbol{C}}{\partial t} + \nabla \bullet (\mathbf{v}\boldsymbol{C}) - \boldsymbol{C} \bullet \nabla\mathbf{v} - \nabla\mathbf{v}^T \bullet \boldsymbol{C} \tag{16}$$

where

$$\nabla \bullet (\mathbf{v}\boldsymbol{C}) = \begin{bmatrix} \sum_{k=1,\ldots,n} v_k \frac{\partial c_{1,1}}{\partial x_k} & \cdots & \sum_{k=1,\ldots,n} v_k \frac{\partial c_{1,n}}{\partial x_k} \\ \vdots & \ddots & \vdots \\ \sum_{k=1,\ldots,n} v_k \frac{\partial c_{n,1}}{\partial x_k} & \cdots & \sum_{k=1,\ldots,n} v_k \frac{\partial c_{n,n}}{\partial x_k} \end{bmatrix} \tag{17}$$

Table 4 collects the expressions of the strain $\boldsymbol{f}_S(\boldsymbol{C})$ and relaxation functions $\boldsymbol{f}_R(\boldsymbol{C})$ for some constitutive models [97].

Table 4. The relaxation and strain functions for some constitutive models are denoted by $\boldsymbol{f}_S(\boldsymbol{C})$ and $\boldsymbol{f}_R(\boldsymbol{C})$. L is the ratio of the length of a fully expanded polymer dumbbell to its equilibrium length, and $\mathrm{tr}(\boldsymbol{C})$ denotes the trace of the tensor $\boldsymbol{C}$.

	Oldroyd B	**FENE-P**	**FENE-CR**	**Linear PTT**
$\boldsymbol{f}_R(\boldsymbol{C})$	$\boldsymbol{C} - \boldsymbol{I}$	$\frac{\boldsymbol{C}}{1-\mathrm{tr}(\boldsymbol{C})/L^2} - \boldsymbol{I}$	$\frac{\boldsymbol{C}-\boldsymbol{I}}{1-\mathrm{tr}(\boldsymbol{C})/L^2}$	$[1 + \epsilon\mathrm{tr}(\boldsymbol{C} - \boldsymbol{I})](\boldsymbol{C} - \boldsymbol{I})$
$\boldsymbol{f}_S(\boldsymbol{C})$	$\boldsymbol{C} - \boldsymbol{I}$	$\frac{\boldsymbol{C}}{1-\mathrm{tr}(\boldsymbol{C})/L^2} - \boldsymbol{I}$	$\frac{\boldsymbol{C}-\boldsymbol{I}}{1-\mathrm{tr}(\boldsymbol{C})/L^2}$	$\boldsymbol{C} - \boldsymbol{I}$

Classically, in the case of the Oldroyd-B model, it is usual to skip the use of C by combining Equations (1) and (15). Then, the constitutive equations in terms of the viscoelastic stress tensor, T_p is written as

$$\lambda T_p + \overset{\triangledown}{T}_p = 2\mu_p \boldsymbol{D}. \tag{18}$$

The summation of μ_s and μ_p of the fluid is defined as total viscosity and denoted as $\mu_0 = \mu_s + \mu_p$. The ratio of μ_s to μ_p is defined as viscosity ratio of viscoelastic fluid also known as the retardation ratio and denoted as $\beta = \frac{\mu_s}{\mu_p} \leq 1$.

Different boundary conditions can occur on the boundary $\partial\Omega$ of Ω depending on the different physical conditions. However, it is important to consider that while the Neumann condition specifies the gradient of the velocity perpendicular to the boundary, the Dirichlet boundary condition expresses the value of the velocity on the boundary. The decomposition of the velocity $\mathbf{v}$ in its normal component $v_N := \mathbf{v} \cdot \mathbf{n}$ and its tangential part $v_T := \mathbf{v} - v_N \mathbf{n}$ (where $\mathbf{n}$ is the vector normal to boundary $\partial\Omega$) allows the description of five different types of boundary conditions [98] as reported in the Table 5.

Table 5. Boundary conditions that are possible on the boundary $\partial\Omega$ of Ω depending on the different physical conditions.

No-slip	The boundary $\partial\Omega$ is considered as a rigid wall where no fluid comes out of the wall. On the boundary, the following equation is valid [98]: $$\mathbf{v} = 0$$
Free-slip	No fluid comes out of the wall, but in contrast to the no-slip condition, no friction acts along the wall. Therefore, on the boundary, the following equation is valid [98] $$v_N = 0 \text{ and } \boldsymbol{\tau} \cdot \mathbf{T} \cdot \mathbf{n} = 0 \text{ for each } \boldsymbol{\tau} \perp \mathbf{n}$$ With the symbol $\cdot$, we intend to represent the matrix-vector operation and the vector dot product operation depending on the type of the operands.
Inflow	In this case, all velocity components are stated [98], i.e., $$\mathbf{v} = \mathbf{v}_{in} \text{ where } \mathbf{v}_{in} \text{ is given}$$
Outflow	In order to prescribe boundary conditions at the outlet, many technics exist. Practicable importance can be attributed to the so-called "do-nothing" method, which is described by the following equation $$\nabla \mathbf{v} \cdot \mathbf{n} - p\mathbf{n} = 0$$ with a prescribed normalized pressure p [98].
Periodic	In the case of a one-directional periodical problem with periodic length π, the velocities and the pressure have to be equal on both sides of the periodical direction. If we assume that the problem is periodic in the x_i direction, it follows that [98] $$\begin{aligned} \mathbf{v}(x_1, \ldots, x_i, \ldots, x_n, t) &= \mathbf{v}(x_1, \ldots, x_i + \pi, \ldots, x_n, t) \\ \partial_{x_1}\mathbf{v}(x_1, \ldots, x_i, \ldots, x_n, t) &= \partial_{x_1}\mathbf{v}(x_1, \ldots, x_i + \pi, \ldots, x_n, t) \\ p(x_1, \ldots, x_i, \ldots, x_n, t) &= p(x_1, \ldots, x_i + \pi, \ldots, x_n, t) \end{aligned}$$

If the velocity is stated on the whole boundary of the domain (i.e., when pure Dirichlet boundary conditions are considered), the additional condition

$$\int_{\partial\Omega} \mathbf{v} \cdot \mathbf{n} \, dA = 0$$

must be satisfied because the velocity field is divergence-free [98].

5.1.2. The Two-Phase Flow Problem

Two-phase flow describes the combined behavior of two different fluids (or phases) that are immiscible at a molecular level and both occupy domain $\Omega \in \Re^n$. For example, this method was used to describe a water droplet in the air or the description of cellular aggregates in the surrounding hydrogel. Within each phase, all physical quantities are taken to be continuous; however, at the interface, at least one physical quantity is discontinuous. Modeling and simulating multiphase fluid flows have been a big challenge over the years; we will describe some of the related models.

We denote the domain occupied by the two liquids with Ω_1 and Ω_2, respectively, such that $\Omega = \Omega_1 \cup \Omega_2$ and such that they are separated by an interface $\Gamma = \Omega_1 \cap \Omega_2$. In general, the phase index $i = 1, 2$ is used for all quantities of the liquid phase i. If the fluids are in motion, the domains $\Omega_i(t), i = 1, 2$, as well as the interface $\Gamma(t)$, are unknown at the time instant t getting a so-called *free boundary problem*.

When it comes to the numerical treatment of free boundary problems, we can often identify two different kinds of issues [98].

1. The representation and evolution in time of the interface.
2. The way the imposed surface boundary conditions are treated.

The first issue above will be treated at the following point 1, whereas point 2 deals with the second issue above.

1. ***Representation and time evolution of the interface.*** To represent the two-phase flows with shifting inter-phase boundaries, various methodologies have been developed in response to the problem's complexity, both numerically and physically. In principle, the methods can be divided into two classes: the so-called *front-tracking (FT) methods* and *front-capturing (FC) methods*.
   In these methods, two-phase flow is treated as a single flow with the density and viscosity smoothly varying across the moving interface which is captured in a Eulerian framework (FC) or in a Lagrangian, framework (FT), with the terms Eulerian and Lagrangian frameworks, meaning, meaning the following [100]:

   Lagrangian framework: In classical field theories, the Lagrangian specification of the flow field is a way of looking at fluid motion where the observer follows an individual fluid parcel as it moves through space and time.

   Eulerian framework: The Eulerian specification of the flow field is a way of looking at fluid motion that focuses on specific locations in the space through which the fluid flows as time passes.

   The front-tracking method is based on a formulation in which a separate unstructured grid (surface grid) with nodes converging at a local velocity is used to represent the interface. One benefit of the technique is that interfacial conditions can be easily included because the interface is precisely established by the surface grid's position. Moreover, a good approximation of the curvature allows for the consideration of surface tension force. The method's drawback is that it requires the use of extra algorithms to compute flows with substantial interface deformations such bubble break-up and coalescence since the surface grid will be severely distorted.
   In contrast, the front-capturing approach uses a scalar function on a stationary grid to capture the interface as part of its implementation of the interface evolution. The ability to compute phenomena like bubble break-up and coalescence is made possible by the implicit interface capture provided by front-capturing techniques. The two main difficulties in using this approach are keeping a clear boundary between the different fluids and accurately calculating the surface tension forces [98].
   The *Level Set* approach was developed as a result of the front-capturing approach [59,60]. A different method, known as the Immersed Boundary (IB) method [58,101–104], was developed for flows across challenging geometries. It can be thought of as a combination of front-capturing and front-tracking procedures.

Level Set Method The idea on which the level set method is based is quite simple. Given an interface Γ in $\Re^n$ of codimension one, bounding an open region Ω, the LS method intends to analyze and compute its subsequent motion under a velocity field v.

This velocity depends on the position, time, the geometry of the interface (e.g., its normal or its mean curvature) and the external physics.

The idea is to define a smooth function $\phi(x,t)$ representing the interface as the set where $\phi(x,t) = 0$.

The level set function $\phi(x,t)$ has the following properties:

$$\begin{aligned} \phi(x,t) &> 0 \qquad \text{for } x \in \Omega_1 \\ \phi(x,t) &< 0 \qquad \text{for } x \in \Omega_2 \\ \phi(x,t) &= 0 \qquad \text{for } x \in \Gamma(t) \end{aligned}$$

Thus, the interface is to be captured for all times t, by defining the set of points $\Gamma(t)$ for which the ϕ values (levels) obey the following equation

$$\frac{\partial \phi}{\partial t} + \boldsymbol{v} \cdot \nabla \phi = 0. \tag{19}$$

Coupling the level set method with problems related to two-phase Navier–Stokes incompressible flow needs the reformulation of Problem 1 as follows

Problem 2. *Equations of two-phase Navier–Stokes incompressible flow defined in* $\Omega \times \Theta$ *using the level set method*

$$\begin{aligned} \frac{\partial}{\partial t}(\rho \mathbf{v}) + (\rho \mathbf{v}) \odot \nabla \mathbf{v} &= -\nabla p + div\ \boldsymbol{T} + \rho \mathbf{g} + \delta(\phi)\sigma\kappa\mathbf{N} && (20)\\ div\ \mathbf{v} &= 0 && (21)\\ \frac{\partial \phi}{\partial t} + \boldsymbol{v} \cdot \nabla \phi &= 0 && (22)\\ \mathbf{v}_{|t=0} &= \mathbf{v}_0 && (23)\\ p_{|t=0} &= p_0 && (24)\\ \phi_{|t=0} &= \phi_0 && (25) \end{aligned}$$

where $\mathbf{v}$ *is the velocity field,* p *is the pressure field,* $\mathbf{g}$ *is the gravitational acceleration,* $\rho(\phi)$ *and* $\boldsymbol{T}(\phi)$ *are the piecewise constant fluid densities and stress tensors described by the Heaviside function,* σ *is the surface tension coefficient,* κ *is the curvature of the interface,* $\mathbf{N}$ *is the unit normal vector outward to* $\Gamma(t)$*, and* $\delta(\phi)$ *is a delta function.*

The following equations hold [59]: With the symbol $|u|$*, we denote the modulus of the vector* $u \in \Re^n$*, that is,*

$$|u| = \sum_{i=1,\ldots,n} u_i^2$$

$$\begin{aligned} \boldsymbol{T}(\phi(x,t)) &= \boldsymbol{T}_1 H_1(x,t) + \boldsymbol{T}_2(1 - H_1(x,t)) && (26)\\ \rho(\phi(x,t)) &= \rho_1 H_1(x,t) + \rho_2(1 - H_1(x,t)) && (27)\\ H_1(x,t) &= \begin{cases} 1 & for\ x \in \Omega_1(t) \\ 0 & for\ x \notin \Omega_1(t) \end{cases} && (28)\\ \delta(\phi(x,t)) &= \begin{cases} 1 & for\ x \in \Gamma(t) \\ 0 & for\ x \notin \Gamma(t) \end{cases} && (29)\\ \mathbf{N} &= -\frac{\nabla \phi}{|\nabla \phi|} && (30)\\ \kappa &= -div\left(\frac{\nabla \phi}{|\nabla \phi|}\right) && (31) \end{aligned}$$

Note that the term $\sigma\kappa\mathbf{N}$ added to Equation (20) is used to model the surface tension forces [98].

The function ϕ can have different format [105–107], which helps to avoid some disadvantages of the LS method related to the discrete solution of transport equations, which may be affected by numerical error leading to loss or gain of mass.

In standard level set methods, the level set function ϕ is defined to be a signed distance function

$$\phi_d(x,t) = \begin{cases} d(x,t) & \text{for } x \in \Omega_1 \\ -d(x,t) & \text{for } x \notin \Omega_1 \end{cases} \tag{32}$$

where $d(x,t) = \min_{x_\Gamma \in \Gamma(t)} |x - x_\Gamma|$.

To improve numerical robustness, *"smoothed out"* Heaviside functions (needed to represent density and viscosity discontinuities over the interface) are often used. For example [106],

$$H_{sm}(\phi) \quad = \quad \phi \tag{33}$$

Immersed Boundary Method

The Immersed Boundary Method (IBM) can be useful to describe fluid–structure interaction when the structure is intended to be an Immersed Body. In the IBM, the fluid is represented in a Eulerian coordinate system, and the configuration of the Immersed Body is described by a Lagrangian approach [101].

Consider the simulation of incompressible flow around the body in Figure 4a, which is described by Problem 1 where $\Omega = \Omega_f \cup \Omega_b$. Ω_f denotes the surrounding fluid domain occupied by the solid body domain Ω_b with the boundary denoted by Γ_b.

In an IB method, the boundary condition would be imposed indirectly through a specification of Problem 1. In general, the specification takes the form of a source term (or forcing function) in the governing equations, which reproduces the effect of boundary [102].

The original IBM, created by Peskin [58], is typically appropriate for flows with immersed elastic limits. It was designed for the combined modeling of blood flow and muscle contraction in a beating heart. A collection of massless points moves with the local fluid velocity v, and these points serve as a Lagrangian tracker of the location of the Immersed Body, represented by a set of elastic fibers.

Thus, the coordinate X of the Lagrangian point is governed by the equation

$$\frac{\delta X}{\delta t} = \boldsymbol{v}(X,t) \tag{34}$$

The effect of the Immersed Body on the surrounding fluid is essentially captured by transmitting the fiber's stress $\boldsymbol{F}$ to the fluid through a localized forcing term $\boldsymbol{f}^{IBM}(x,t)$ in the momentum equations, which is given by

$$\boldsymbol{f}^{IBM}(x,t) = \int_{\Omega_b} \boldsymbol{F}(X,t)\delta(|x-X|)dX, \tag{35}$$

where $\delta(\cdot)$ is the Dirac delta function. The Dirac delta function in $\Re^n$ is formally defined by the following:

$$\delta(x) = \begin{cases} +\infty & x = 0 \\ 0 & x \neq 0 \end{cases}$$

and is also constrained to satisfy the identity

$$\int_{x\in\Re^n} \delta(x)dx = 1.$$

The function $\boldsymbol{F}$ can be written as

$$\boldsymbol{F}(X,t) = -\frac{\wp E}{\wp X}, \tag{36}$$

where $E[X]$ is a given functional called *"the elastic potential energy of the material"* in configuration X and where the notation $\frac{\wp E}{\wp X}$ is shorthand for the *"Fréchet derivative"* of E with respect to X. Thanks to the use of the Dirac delta function, Equation (34) can be rewritten as

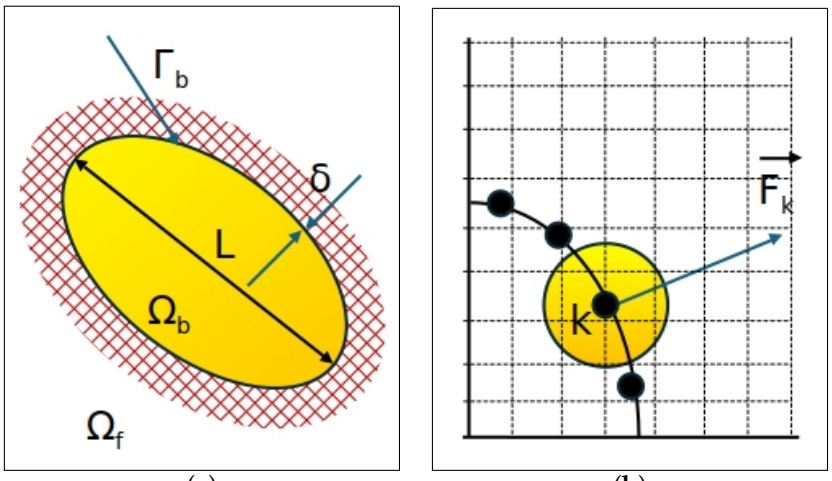
$$\frac{\delta X}{\delta t} = \int_{\Omega_b} \boldsymbol{v}(x,t)\delta(|x-X|)dx. \tag{37}$$

The forcing term is thought to be distributed over a band of cells surrounding each Lagrangian point (see Figure 4b), as the fiber locations typically do not coincide with the nodal points of the Eulerian grid. This distributed force is then imposed on the momentum equations of the surrounding nodes. As a result, a smoother distribution function, represented by d in this case, effectively replaces the delta function and may be applied to a discrete mesh. The d function allows for the rewriting of Equations (37) and (35) as follows:

$$\frac{\delta X}{\delta t} = \int_{\Omega_b} \boldsymbol{v}(x,t)d(|x-X|)dx, \tag{38}$$

$$\boldsymbol{f}^{IBM}(x,t) = \int_{\Omega_b} \boldsymbol{F}(X,t)d(|x-X|)dX. \tag{39}$$

The choice of the distribution function d is a key ingredient in this method. Several different distribution functions were employed in the past (i.e., see [102] for a list of employed d).



(**a**) (**b**)

Figure 4. A generic body around which flow is to be simulated. The body occupies the volume Ω_b with boundary Γ_b (**a**). Transfer of forcing $\boldsymbol{F}$ from Lagrangian boundary point X to surrounding fluid nodes (**b**).

Coupling the IB equations with problems related to the Navier–Stokes incompressible flow requires the reformulation of the Problem 1 as follows:

Problem 3. *Equations of the Navier–Stokes incompressible flow defined in* $\Omega_f \times \Theta$ *surrounding an Immersed Body* Ω_b*:*

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + (\rho \mathbf{v}) \odot \nabla \mathbf{v} = -\nabla p + div\ \boldsymbol{T} + \rho \mathbf{g} + \boldsymbol{f}^{IBM} \quad (40)$$

$$div\ \mathbf{v} = 0 \quad (41)$$

$$\mathbf{v}_{|t=0} = \mathbf{v}_0 \quad (42)$$

$$p_{|t=0} = p_0 \quad (43)$$

$$\frac{\delta X}{\delta t} = \int_{\Omega_b} \boldsymbol{v}(x,t) d(|x - X|) dx \quad (44)$$

$$\boldsymbol{f}^{IBM}(x,t) = \int_{\Omega_b} \boldsymbol{F}(X,t) d(|x - X|) dX \quad (45)$$

$$\boldsymbol{F}(X,t) = -\frac{\wp E}{\wp X}, \quad (46)$$

where ρ *is the density of the fluid,* $\mathbf{v}$ *is the velocity field,* p *is the pressure field,* $\mathbf{g}$ *is the gravitational acceleration,* $\boldsymbol{T}$ *is the stress response to the deformation of the fluid,* X *is the Lagrangian coordinate of the generic point of the body, and* E *is the elastic potential energy of the body material.*

Equations (38) and (39) are interaction equations. They convert Lagrangian variables to Eulerian variables (Equation (38)) and vice versa (Equation (39)).

2. ***Surface boundary conditions treatment***

Since the second derivative of a discontinuous function determines the curvature κ, using front-capturing methods to compute the surface tension force requires particular considerations. Various methods were devised to precisely calculate the surface tension force. The most well-known method is presumably the *continuum surface force (CSF)* model [61]. The model considers surface tension force as a continuous, 3D effect across an interface, rather than as a boundary value condition on the interface. The continuum method eliminates the need for interface reconstruction. A model of a diffuse interface is examined, in which the surface tension force is converted into a volume force that distributes across several cell layers.

We note (see Equation (20)) that surface tension $\boldsymbol{F}_{sa}$ per interfacial area at a point $\boldsymbol{x}_\Gamma$ on the interface Γ is given by

$$\boldsymbol{F}_{sa}(\boldsymbol{x}_\Gamma) = \rho \kappa(\boldsymbol{x}_\Gamma) \mathbf{N}(\boldsymbol{x}_\Gamma) \quad (47)$$

where ρ is the surface tension coefficient, κ is the curvature of the interface, and $\mathbf{N}$ is the unit normal vector outward to $\Gamma(t)$.

Suppose that the interface Γ where the fluid changes from fluid 1 to fluid 2 discontinuously is replaced by a continuous transition. Applying a pressure jump at an interface brought on by surface tension is no longer appropriate. Instead, as can be shown in Figure 5, surface tension should be considered active everywhere in the transition area.

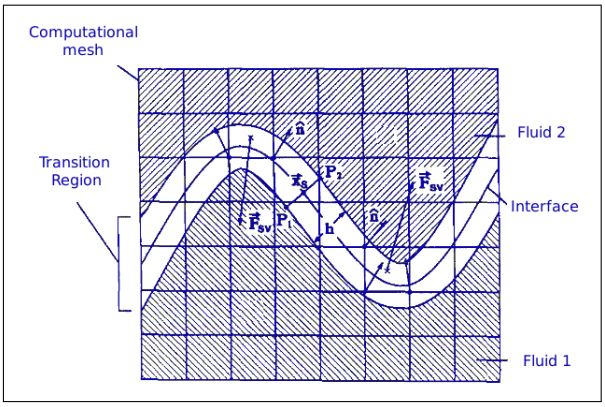


Figure 5. The transition region (unshaded) separating two fluids used in the CSF formulation: h is the width of the region. Credits: image inspired by the work of Brackbill et al. [61].

As in [61], we choose a volume force $\boldsymbol{F}_{sv}(\boldsymbol{x})$ at any point x of transition region as

$$\boldsymbol{F}_{sv}(\boldsymbol{x}) = \begin{cases} \rho\kappa(\boldsymbol{x})\nabla\tilde{c}(\boldsymbol{x}) & \text{for } |\mathbf{N}(x_\Gamma)(\boldsymbol{x}-\boldsymbol{x}_\Gamma)| < h \\ 0 & \text{for } |\mathbf{N}(x_\Gamma)(\boldsymbol{x}-\boldsymbol{x}_\Gamma)| \geq h \end{cases} \tag{48}$$

where $\tilde{c}(\boldsymbol{x})$ is a smooth *"color function"* that approaches the *"characteristic function"*, $c(\boldsymbol{x})$, as the width h of the region approaches to zero,

$$\lim_{h\to 0} \tilde{c}(\boldsymbol{x}) = c(\boldsymbol{x}), \tag{49}$$

and where the *"characteristic function"* can be defined as

$$c(\boldsymbol{x}) = \begin{cases} 1 & \text{for } x \in \Omega_1 \\ 0 & \text{for } x \in \Omega_2 \\ \frac{1}{2} & \text{for } x \in \Gamma \end{cases}, \tag{50}$$

where Ω_1 and Ω_2 represent the domains occupied by fluids 1 and 2, respectively. We note that for the curvature κ of the interface Γ and for the unit normal vector $\mathbf{N}$ outward to Γ, the following equations are valid:

$$\mathbf{N} = -\frac{\nabla\tilde{c}}{|\nabla\tilde{c}|} \tag{51}$$

$$\kappa = -\mathrm{div}\left(\frac{\nabla\tilde{c}}{|\nabla\tilde{c}|}\right) \tag{52}$$

The function $\tilde{c}(\boldsymbol{x})$ can be defined as a convolution of the function $c(\boldsymbol{x})$ with an interpolation function $\mathcal{J}$ according to Brackbill et al. [61]:

$$\tilde{c}(\boldsymbol{x}) = \frac{1}{h^3}\int_V c(\boldsymbol{y})\mathcal{J}(y-x)dy \tag{53}$$

where $\mathcal{J}$ has to satisfy the following conditions

(a) $\int_V \mathcal{J}(x)dx = h^3$,
(b) $\mathcal{J}(x) = 0$ for $|x| \geq h/2$,
(c) $\mathcal{J}(x)$ is differentiable and decreases monotonically with increasing $|x|$.

Volume force $\boldsymbol{F}_{sv}(\boldsymbol{x})$ will result in the same total force as $\boldsymbol{F}_{sa}(x_\Gamma)$, but spread over the finite interface width, i.e.,

$$\lim_{h\to 0}\int_{\Delta V} \boldsymbol{F}_{sv}(\boldsymbol{x})dx^n = \int_{\Delta A} \boldsymbol{F}_{sa}(\boldsymbol{x}_\Gamma)dA \tag{54}$$

Coupling the *continuum surface force (CSF)* model with problems related to two-phase Navier–Stokes incompressible flow requires the reformulation of the Problem 1 as follows:

Problem 4. *Equations of two-phase Navier–Stokes incompressible flow defined in* $\Omega \times \Theta$ *using the CSF model:*

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + (\rho \mathbf{v}) \odot \nabla \mathbf{v} = -\nabla p + div\ \boldsymbol{T} + \rho \mathbf{g} + \boldsymbol{F}_{sv}(\boldsymbol{x}) \quad (55)$$

$$div\ \mathbf{v} = 0 \quad (56)$$

$$\mathbf{v}_{|t=0} = \mathbf{v}_0 \quad (57)$$

$$p_{|t=0} = p_0 \quad (58)$$

where $\mathbf{v}$ *is the velocity field,* p *is the pressure field,* $\mathbf{g}$ *is the gravitational acceleration, and* $\rho(x)$ *and* $\boldsymbol{T}(x)$ *are the piecewise constant fluid densities and stress tensors described by the Heaviside function.*
The following equations hold [61]:

$$\boldsymbol{T}(x) = \boldsymbol{T}_1 H_1(x) + \boldsymbol{T}_2(1 - H_1(x)) \quad (59)$$

$$\rho(x) = \rho_1 H_1(x) + \rho_2(1 - H_1(x)) \quad (60)$$

$$H_1(x) = \begin{cases} 1 & for\ x \in \Omega_1(t) \\ 0 & for\ x \notin \Omega_1(t) \end{cases} \quad (61)$$

$$(62)$$

Note that the term $\boldsymbol{F}_{sv}(\boldsymbol{x})$ *(as defined in Equation 48) is added to Equation (55) to model the surface tension forces [61].*

5.1.3. Transport and Response of Biological and Chemical Species

This subsection will describe models that are used to study how the tissue reacts to the chemical and mechanical environment in the culture medium. Although there are not many studies on this topic, this aspect is crucial for models that seek to create better culture protocols [64]. Creating a strict framework of conservation laws for mixes of interacting continua has been the subject of countless investigations [108]. According to this theory, the characteristics of composite biological tissues are assessed in relation to the volume percentage of the constituent parts (i.e., extracellular matrix, culture medium, and cells). A system of conservation equations that control the transfer of mass and momentum between the component phases is the foundation of the multi-phase models.

For a system comprising incompressible phases, if inertial effects and body forces are negligible, then the equations governing the i-th phase may be expressed as

$$\frac{\partial}{\partial t}(\rho_i \theta_i) + \rho_i \mathrm{div}(\theta_i \boldsymbol{v}) - D_i \nabla^2 \theta_i = S_i, \quad (63)$$

$$\mathrm{div}(\theta_i \boldsymbol{T}_i) + \sum_{j \neq i} \boldsymbol{F}_{ij} = \mathbf{0} \quad (64)$$

where ρ_i, θ_i, $\boldsymbol{v}$, D_i, and $\boldsymbol{T}_i$ are, respectively, the density, the volume fraction (a value in $[0\ 1]$ interval), the velocity, the diffusive coefficient, and the stress tensor characterizing the i-th phase, where S_i is the net rate of mass transfer into the i-th phase and $\boldsymbol{F}_{ij}$ denotes the force acting on phase i as a result of interactions with phase j.

The sum of all phases adds up to 1 ($\sum_i \theta_i = 1$) following the conservation of mass. The biological component can also be modeled as a single-phase material, for instance, cells only, where Equation (63) can be used to estimate θ_c as the cell density; in such a case, the conservation of mass law above does not apply; for example, Equation (64) does not apply.

Depending on the type of biological phase, diffusive and convective transport are taken into account in Equation (63). For example, cells are typically thought of as a

single phase that diffuses slowly and does not undergo convection. The force balance between the various phases is captured by the equation for the conservation of momentum (see Equation (64)). Interactions between cell populations, between the cells and the scaffold, and between the cells and the media are included in Equation (64). More complex processes, including tissue evolution through cell proliferation or ECM deposition (in biology, the extracellular matrix (ECM), also called the intercellular matrix (ICM), is a network consisting of extracellular macromolecules and minerals that provide structural and biochemical support to surrounding cells [109]), can be considered by using the term S_i. The introduction of functions controlling cell proliferation or ECM formation is possible. Proliferation and death terms are typically included in cell growth functions, and they depend on the available cell volume fraction θ_c and the concentration of nutrients like oxygen. Indeed, it is widely known that the migration, differentiation, and proliferation of cells are influenced by growth factors, waste products like lactate, and nutrients like oxygen (called *"chemical species"*).

Mass conservation is used to describe the distribution of any solute, assuming Fickian diffusion and setting the initial and boundary conditions.

$$\frac{\partial C_i}{\partial t} + \mathrm{div}(C_i \boldsymbol{v}) - D_i \nabla^2 C_i \quad = \quad M_i, \tag{65}$$

where $\boldsymbol{v}$ is the media flow velocity, D_i is the appropriate coefficient of diffusion, M_i is a term indicating the metabolic reaction, and C_i is the concentration of the i-th species (in, for example, oxygen).

See [64,108] for more details about models related to the transport of chemical species and the response of biological species (cells and tissues) to the chemical and mechanical environment.

5.2. Models at the Meso Scale

The proposed models are based on a stochastic approach (the Monte Carlo-based method) on a deterministic particle-dynamics-based approach (the Cellular Particle Dynamics), and on a potential hybrid stochastic–deterministic approach (the cellular automata method). In Table 6, we provide a summary of the *"Pros and Cons"* of the described models.

Table 6. Summary of *"Pros and Cons"* of the models at the meso-scale.

Model	*"Pros"*	*"Cons"*
Monte Carlo-based methods (MCMs) (Section 5.2.1)	MCMs are a broad category of computing methods for assessing sums and integrals or estimates of energy or other macro-parameters in a physical or biological system. They are universally appreciated for their power and easy implementation beyond their versatility and flexibility.	They can only be used to assess equilibrium states.
The cellular automata method (Section 5.2.3)	An approach where the behaviors of the species under study are determined by rules which can be probabilistic or deterministic. CA could be particularly significant in managing the behaviors of systems where the traditional, differential-equation-based methodologies are either useless or difficult to adopt due to the numerous complex interconnections between the system constituents.	They can only be used to assess equilibrium states. Either synchronous or asynchronous rule applications lead to CA development but serious practical implementations limit the use of CA to an asynchronous rule application.
The cellular particle dynamics (CPD) (Section 5.2.2)	CPD is based on the idea of constructing a cell from a cloud of sub-cellular entities called elements. All the elements, through their relative positions, and in response to cell internal and external forces, determine the dynamics of all the cells.	The definition of model parameters could be very difficult.

5.2.1. Monte Carlo-Based Methods

The collection of algorithms, based on Markov Chain Monte Carlo (MCMC) methods, are universally appreciated for their power and easy implementation beyond their versatility and flexibility. Such relevance inspired Dongarra et al. [110] to call the original Metropolis algorithm [111], which is the first implementation of an MCMC method, one of the ten most important algorithms of the twentieth century. For the story of the evolution of MCMC methods, we suggest reading the paper by Matthew Richey [112] that is also relevant for the accessible and synthetic introduction to MCMC methods. MCMC is a broad computing method for assessing sums and integrals or estimates of energy or other macro-parameters in a physical or biological system [113].

The original MCMC method, the so-called *"Metropolis algorithm"*, was inspired by nature and arose in physics. Its creators were aware that they had defined

> *...a general method, suitable for fast electronic computing machines, of calculating the properties of any substance which may be considered as composed of interacting individual molecules. [111]*

The original Metropolis algorithm proposed an approach to solve the basic problem of averaging in classical equilibrium statistical mechanics. Such an approach was intended to compute averages of any physical observable F using an arithmetic mean of the values of F obtained in different sampled states of a system composed of N particles. The sampling schema described in [111] was based on an approach that later was discovered to be a Marchov Chain stochastic process [112]. Let us briefly describe some of the fundamental ideas behind the Marchov Chain.

Definition 1. Markov Chains
Given a finite state space $S = \{1, 2, ..., N\}$*, a Markov Chain is a stochastic process defined by a sequence of random variables,* $X_i \in S$, $i \in \mathbb{N}$ *such that*

$$Prob(X_{k+1} = x_{k+1} | X_1 = x_1, \ldots, X_k = x_k) = Prob(X_{k+1} = x_{k+1} | X_k = x_k). \quad (66)$$

Equation (66) expresses the fact that the probability of being in a particular state at the $(k+1)$-th step only depends on the state at the k-th step. When Markov Chains are considered for which this dependence is independent of k (that is, time-homogeneous Markov Chains), a $N \times N$ transition matrix $\boldsymbol{P} = (\boldsymbol{p}_{i,j})$ can be defined by

$$\boldsymbol{p}_{i,j} = Prob(X_{k+1} = j | X_k = i). \quad (67)$$

Note that

1. $\sum_{i=1}^{N} \boldsymbol{p}_{i,j} = 1$.
2. The (i, j)-entry of the K-th power of $\boldsymbol{P}$ gives the probability of transitioning from state i to state j in K steps.

A Markov Chain is

Irreducible if, for all states i and j, there exists K such that $\left(\boldsymbol{P}^K\right)_{i,j} \neq 0$.

A-periodic if, for all states i and j, the Greatest Common Divisor of the set $\left\{K : \left(\boldsymbol{P}^K\right)_{i,j} > 0\right\}$ is equal to 1.

An irreducible, a-periodic Markov Chain must have a unique distribution π on the state space S with the property that

$$\pi = \pi \boldsymbol{P}. \quad (68)$$

where $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$ and where π_i is the probability of state i.

If Equation (68) is valid, the Markov Chain is said to be *"stable on the distribution"* π, or that π is the stable distribution for the Markov Chain. The following two considerations are valid:

- If π is the stable distribution for an irreducible, a-periodic Markov Chain, then the Markov Chain can be used to sample from π.
- Samples from π can be used to approximate the properties of π. For example, suppose f is any real-valued function on the state space S, and suppose that $s_1, s_2, \ldots, s_M$ is a sample from π; then, the ergodic theorem [114] ensures that

$$\lim_{M \to \infty} \frac{1}{M} \sum_{i=1}^{M} f(s_i) = E_\pi[f], \quad (69)$$

where $E_\pi[f]$ is the expected value $E_\pi[f] = \sum_{i=1}^{N} f(i)\pi_i$.

So, if an irreducible, a-periodic Markov Chain can be built that is stable on π, that chain can be used to generate states from S that can be exploited to approximate, by Equation (69), the expected value $E_\pi[f]$ of any real-valued function on the state space S. The original Metropolis algorithm builds a Markov Chain that is stable on a particular type of distribution $\pi^{Boltzmann}$ that is the *"Boltzmann distribution"*.

Indeed, in the context of statistical mechanics, which is the branch of physics concerned with the average behavior of large systems of interacting particles, the Metropolis algorithm was introduced to study the properties of $\pi^{Boltzmann}$. In statistical mechanics, the state of the particles is described by a configuration ω taken from the configuration space Ω. The physics of a configuration space is described by an energy function $E : \omega \in \Omega \leftarrow E(\omega) \in \Re^+$. Then, the value $E(\omega)$ is said to be the energy of the configuration ω. The random organization of molecules in a limited space is governed by the fundamental principle of statistical physics

assuming that Nature seeks low-energy configurations. For any $\omega \in \Omega$, its Boltzmann probability, $\pi^{Boltzmann}(\omega)$, is

$$\pi^{Boltzmann}(\omega) = \frac{BoltzmannWeight(\omega)}{Z}, \tag{70}$$

where T is the temperature and k is the Boltzmann's constant,

$$BoltzmannWeight(\omega) = e^{-\frac{E(\omega)}{kT}} \tag{71}$$

and where Z is known as *"Partition Function"*, defined as

$$Z = \sum_{\omega'} e^{-\frac{E(\omega')}{kT}}. \tag{72}$$

The relationship between energy and probability leads to expressions for many interesting physical quantities. For example, the total energy of the system, $\langle E \rangle$, is the expected value of the energy function $E(\omega)$ and is defined by

$$\langle E \rangle = \sum_{\omega \in \Omega} E(\omega)\pi^{Boltzmann}(\omega). \tag{73}$$

The brilliant idea behind the Metropolis algorithm is that it creates an easily computed Markov Chain that is stable on the Boltzmann distribution. The making of such a chain requires only the evaluation of the ratio between the values of Boltzmann probabilities of two consecutive states (i.e., see algorithm described in Section 6.2.1 where $g(\cdot) = \pi^{Boltzmann}(\cdot)$) that is

$$\frac{\pi^{Boltzmann}(\omega)}{\pi^{Boltzmann}(\omega^*)} = \frac{BoltzmannWeight(\omega)}{BoltzmannWeight(\omega^*)} = e^{-\frac{E(\omega)-E(\omega^*)}{kT}}$$

and not the full probabilities Z, avoiding the evaluation of the partition function which is analytically and computationally intractable in any realistic setting. All the steps in the Markov Chain can be computed easily, if the value of $E(\omega)$ is easily computable, and, most importantly, the value of $\Delta E = E(\omega) - E(\omega^*)$. In many settings, E is extremely simple to compute; often, it is independent of $|\Omega|$ (the energy could reflect just the influence that neighboring particles exert on each other).

Finally, thanks to the use of the Metropolis algorithm, a sample $\{\omega_i\}_{i=1,\ldots,M}$ of Ω can be generated that can be used to compute an approximation (see Equation (69)) $\tilde{E}$ of $\langle E \rangle$ by the following

$$\tilde{E} = \frac{1}{M}\sum_{i=1}^{M} E(\omega_i). \tag{74}$$

Moreover, since naturally physical systems *"seek"* the state of minimal value for their energy, the succession of states $\{\omega_i\}_{i=1,\ldots,M}$ *"converges"* toward the equilibrium state characterizes the minimum energy state.

The generalization of the Metropolis algorithm is due to Hastings [115]. To generalize the Metropolis method for a given distribution π, a Markov Chain $P = \left(p_{i,j}\right)_{i,j=1,\ldots,N}$ should be defined with π as its stationary distribution.

Hastings assumes that $p_{i,j}$ has the form

$$p_{i,j} = \begin{cases} q_{i,j}\alpha_{i,j} & \text{if } i \neq j \\ 1 - \sum_{k \neq i} p_{i,k} & \text{if } i = j \end{cases}, \tag{75}$$

where $Q = \left(q_{i,j}\right)_{i,j=1,\ldots,N}$ is the proposed transition matrix of an arbitrary Markov Chain on a finite state space S, and $\alpha_{i,j}$ is given by

$$\alpha_{i,j} = \frac{s_{i,j}}{1 + \frac{\pi_i}{\pi_j}\frac{q_{i,j}}{q_{j,i}}}, \tag{76}$$

where $s_{i,j}$ is a symmetric function of i and j and is chosen so that $0 \leq \alpha_{i,j} \leq 1$ for all $i, j = 1, \ldots, N$. For such choices for $s_{i,j}$, the matrix P satisfies the following reversibility condition

$$\pi_i p_{i,j} = \pi_j p_{j,i}. \tag{77}$$

From Equation (77), it follows that

$$\sum_{i=1,\ldots,N} \pi_i p_{i,j} = \pi_j, \text{for all } j = 1, \ldots, N, \tag{78}$$

and hence that π is a stationary distribution of P.

The following two examples for $S = \left(s_{i,j}\right)_{i,j=1,\ldots,N}$ are given:

$$\begin{aligned} S^{(M)} &= \left(s_{i,j}^{(M)}\right)_{i,j=1,\ldots,N}, \quad \text{where } s_{i,j}^{(M)} = \begin{cases} 1 + \frac{\pi_i}{\pi_j}\frac{q_{i,j}}{q_{j,i}} & \left(\text{if } \frac{\pi_i}{\pi_j}\frac{q_{i,j}}{q_{j,i}} \geq 1\right) \\ 1 + \frac{\pi_j}{\pi_j}\frac{q_{j,i}}{q_{i,j}} & \left(\text{if } \frac{\pi_i}{\pi_j}\frac{q_{i,j}}{q_{j,i}} < 1\right) \end{cases} \\ S^{(B)} &= \left(s_{i,j}^{(B)}\right)_{i,j=1,\ldots,N}, \quad \text{where } s_{i,j}^{(B)} = 1. \end{aligned} \tag{79}$$

With $q_{i,j} = q_{j,i}$ and S defined as $S^{(M)}$, a method devised by Metropolis et al. is obtained. If $q_{i,j} = q_{j,i}$ and S is defined as $S^{(B)}$, another classical MCMC method from Barker [116] is recalled.

In Section 6.2.1, the algorithm implementing a Metropolis–Hastings method is described.

5.2.2. Cellular Particle Dynamics

In the previous subsection, MCMC methods were introduced. While it is incredibly helpful for various statistical physics and biology problems, there are some situations that might not even be amenable to stochastic approaches [49,117].

CPD is based, as in the Subcellular Element Model (SEM) approach [118], on the idea to *"construct a cell from a cloud of $N_{CellularElements}$ sub-cellular entities"* called elements [49,118]. The elements, belonging to a single cell, all together define the same cell and, through their relative positions and in response to the cell's internal and external forces, determine the cell's shape (see Figure 6).

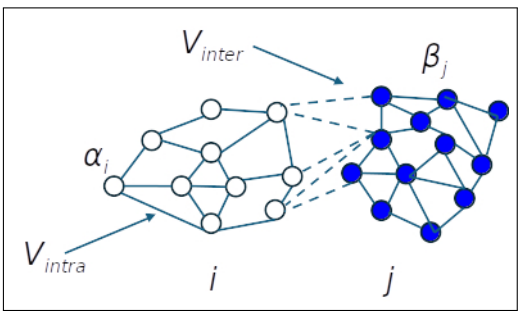


Figure 6. A schematic figure showing two cells constructed from elements, with intra-cellular potentials binding elements of a given cell to each other, and inter-cellular potentials binding adjacent elements from neighboring cells.

The SEM approach (that takes place in a Lagrangian framework) is based on a stochastic assumption using Langevin equations [119]. The stochastic variables are the position vectors $\boldsymbol{y}_{\alpha_i}$ of each element, where the notation α_i denotes the α-th element of the i-th cell, and where $\boldsymbol{y}_{\alpha_i} = \left(y^1_{\alpha_i}, \ldots, y^N_{\alpha_i}\right) \in \Re^N$.

In the simplest case, in which the inertial terms in the equation of motion are discarded and in which the movement of elements is due to stochastic forces and biomechanical interactions, the following equation is valid for the generic element α_i:

$$\mu \frac{\partial \boldsymbol{y}_{\alpha_i}}{\partial t} = \boldsymbol{\eta}_{\alpha_i} \underbrace{-\nabla_{\boldsymbol{y}_{\alpha_i}} \left[\sum_{\beta_i \neq \alpha_i} V_{intra}\left(\left|\boldsymbol{y}_{\alpha_i} - \boldsymbol{y}_{\beta_i}\right|\right) \right]}_{\text{Internal forces } \boldsymbol{F}^{Intra}_{\alpha_i}} \underbrace{-\nabla_{\boldsymbol{y}_{\alpha_i}} \left[\sum_{j \neq i} \sum_{\beta_j} V_{inter}\left(\left|\boldsymbol{y}_{\alpha_i} - \boldsymbol{y}_{\beta_j}\right|\right) \right]}_{\text{External forces } \boldsymbol{F}^{Inter}_{\alpha_i}} \tag{80}$$

where μ is the friction coefficient, $\boldsymbol{\eta}_{\alpha_i} = \left(\eta^1_{\alpha_i}, \ldots, \eta^N_{\alpha_i}\right) \in \Re^N$ is a stochastic force modeled as a Gaussian white noise with zero mean and variance $\left\langle \eta^n_{\alpha_i}(t), \eta^m_{\beta_j}(t') \right\rangle$ with

$$\left\langle \eta^n_{\alpha_i}(t), \eta^m_{\beta_j}(t') \right\rangle = 2D\delta_{n,m}\delta_{i,j}\delta_{\alpha_i,\beta_j}\delta_{t,t'}, \tag{81}$$

where D is the self-diffusion coefficient. The Equation (81) indicates that stochastic forces are uncorrelated in all the possible variables: between cells, between elements within a cell, between spatial dimensions, and between two different instants. The vectors $\boldsymbol{F}^{Intra}_{\alpha_i} = \left(F^{Intra}{}^1_{\alpha_i}, \ldots, F^{Intra}{}^N_{\alpha_i}\right) \in \Re^N$ and $\boldsymbol{F}^{Inter}_{\alpha_i} = \left(F^{Inter}{}^1_{\alpha_i}, \ldots, F^{Inter}{}^N_{\alpha_i}\right) \in \Re^N$ represent the internal and external biomechanical forces, respectively, acting on the generic element α_i.

The functions $V_{intra}\left(d_{\alpha_i,\beta_j}\right)$ and $V_{inter}\left(d_{\alpha_i,\beta_j}\right)$, which depend on the distance

$$d_{\alpha_i,\beta_j} = \left|\boldsymbol{y}_{\alpha_i} - \boldsymbol{y}_{\beta_j}\right| = \left(\sum_{n=1}^{N} \left(y^n_{\alpha_i} - y^n_{\beta_j}\right)^2 \right)^{\frac{1}{2}}$$

between two elements, represent the intra- and inter-cellular interaction potential between two elements in the same and different cells, respectively. In [49], Flenner et al. propose the following expression for functions $V_{intra}(d)$ and $V_{inter}(d)$

$$V_{intra}(d) = V_{LJ}(d, \epsilon, \rho) + V_c(d, k, \gamma), \tag{82}$$

$$V_{inter}(d) = V_{LJ}(d, \epsilon, \rho), \tag{83}$$

where $V_{LJ}(d, \epsilon, \rho)$ is the Lennard–Jones (LJ) potential energy

$$V_{LJ}(d, \epsilon, \rho) = 4\epsilon \left[\left(\frac{\rho}{d}\right)^{12} - \left(\frac{\rho}{d}\right)^{6} \right], \tag{84}$$

and where $V_c(d, k, \gamma)$ represents a confining potential energy of the form

$$V_c(d, k, \gamma) = \begin{cases} \frac{k}{2}(d - \gamma)^2, & \text{for } d > \beta \\ 0 & \text{for } d < \beta \end{cases}. \tag{85}$$

By tuning the values of k and γ in the confining potential $V_c(d, k, \gamma)$, the stiffness and the size of a cell can be controlled, respectively, while the LJ parameters ϵ and ρ represent the energy required to separate the elements and the size (diameter) of the elements, respectively.

5.2.3. Cellular Automata Model

The previous sections discussed a number of standard categories of physical models. These models govern the evolution of physical systems formalized in terms of coupled sets

of ordinary or partial differential equations. Most conventional models are deterministic, which means that the force fields that are employed and the initial conditions of the simulations fully determine the results of running simulations. This section offers an alternative approach where the behaviors of the species under study are determined by rules, which can be probabilistic or deterministic, as opposed to forces and energy.

The foundation of this approach is the use of cellular automata (CA) [120] postulated for the first time by the mathematical physicist John von Neumann [121] and the mathematician Stanislaw Ulam [122], more than 70 years ago.

Since then, different research teams have proven that CA is particularly significant in managing the behaviors of systems where the traditional, differential-equation-based methodologies are either useless or difficult to adopt due to the numerous complex interconnections between the system constituents.

According to the mathematician Stephen Wolfram, the CA model can be defined as follows:

> *CA are simple mathematical idealizations of natural systems. They consist of a lattice of discrete identical sites, each site taking on a finite set of ... values. The values of the sites evolve in discrete time steps according to ... rules that specify the value of each site in terms of the values of neighboring sites. CA may thus be considered as discrete idealizations of the partial differential equations often used to describe natural systems* [123].

According to Kier et al. [120], a model made up of the following parts is referred to as a *cellular automaton* (singular):

- A grid of individual cells.
- A group of ingredients.
- A collection of local rules dictating how the constituents behave.
- Identified starting conditions.

A simulation can be run after the model's aforementioned elements have been defined. The system in the simulation changes through a sequence of discrete time steps, or iterations, when all of the system's components are subjected to the model's rules, and the system's configuration is updated as a result.

Either synchronous or asynchronous rule application might lead to CA development. In the first scenario, every automaton cell changes on each iteration; in the second scenario, the algorithm merely processes a subset of the cells or even just one cell. One way to model the complex behavior of macroscopic systems, which arises from the interaction of all components on a small scale, is through the synchronous application of transition rules. However, there are serious practical implementation limitations associated with this approach, such as the potential for conflicts in certain situations (e.g., assigning two ingredients to move to the same empty cell).

Due to this latter factor, the majority of automata currently in development can only be implemented asynchronously [124] (see algorithm described in Section 6.2.2 for the asynchronous version of the CA simulation). The above general CA framework can be used to construct a wide range of various models.

The components of this CA model are then described in more detail.

The grid, the cells, and the ingredients: A cellular automaton consists of a regular grid of N_{Cells} cells. The grid can be in any finite number of dimensions N. Every grid cell can typically exist in a limited number N_{States} of "states", which specify the cell's occupancy. The cell may be empty or hold a specified ingredient, which, if it is present, may be a type of particle, a specific molecule, or some other relevant thing for the topic under consideration. An illustration of a bi-dimensional CA grid with 5×5 cells each, whereby some cells are occupied by ingredient A (red cells) and other cells by ingredient B (blue cells), can be found in Figure 7a.

Movements and other actions on the grid are controlled by rules that are only dependent on the characteristics of the cells that are closest to the ingredient. The

neighborhood of a cell is its immediate surroundings. The *"Von Neumann neighborhood"* is the most often employed neighborhood in two-dimensional CA investigations (see Figure 7d, where the blue cell neighborhood is pictured by the four red cells).

Another common neighborhood is the *"Moore neighborhood"*, pictured by the red cells surrounding the blue cell in Figure 7c. Another useful neighborhood is the *"extended Von Neumann neighborhood"*, shown in Figure 7d by the red and green cells surrounding the blue cell.

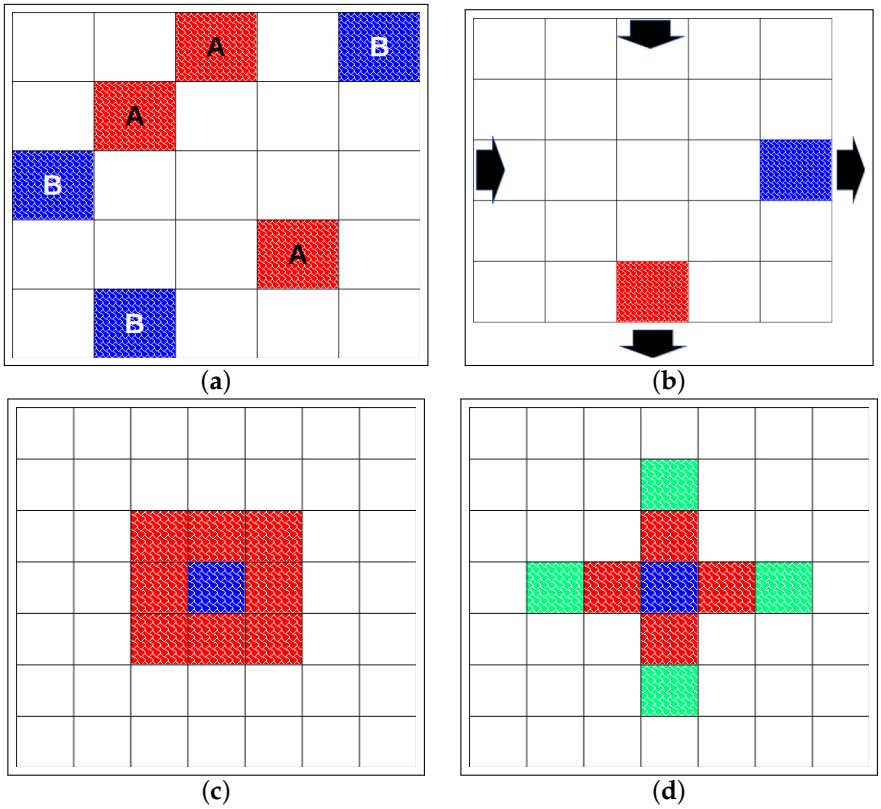


Figure 7. (**a**) A grid of cellular automata in two dimensions. Two sets of occupied cells with various constituents, *A* and *B*, are displayed. The empty cells are left blank. (**b**) Movement of cell constituents at the grid boundaries on a torus's surface. It is possible for one of the ingredients in the blue cell to move off the grid to the right and then return to the grid's left border. It is possible for another component of red cells to shift from the bottom to the top of the grid. (**c**,**d**) Representation (see the red cells) of the Moore (**c**) and von Neumann (**d**) neighborhood, respectively, for the blue cell. Green cells in Figure (**d**) represent the extended von Neumann neighborhood for the blue cell.

Every grid cell's value will be impacted by how its neighborhoods are managed, including cells that are on the edge of the CA grid. Keeping the values in those cells constant is one approach that might be taken. Using distinct neighborhood definitions for these cells is an additional strategy. They might have fewer neighbors, but it is also possible to define more elaborate strategies for the cells that are closest to the edges.

These cells are typically handled in a toroidal manner; for instance, in a bi-dimensional grid, one cell "goes off the top" and "enters" at the corresponding position on the bottom, and one cell "enters on" the right when it "goes off" the left (this is sometimes referred to as periodic boundary conditions in the field of partial differential equations). To illustrate this, consider taping the rectangle's left and right sides to create a tube, and then the top and bottom edges to create a torus (see Figure 7b).

The rules: The behaviors of the ingredients on the grid, and consequently the evolutions of the CA systems, are governed by a variety of rules. According to what is described in [69], a list of potential rule types is provided below.

Movement rules: Movement rules define the condition that, during an iteration, the ingredients move to neighborhood cells. These rules take several forms [120]:

- The *"breaking"* probability, $P_B(AB) \in [0,\ 1]$, defines the condition that two adjacent ingredients A and B may remain linked to each other.
- The *"joining"* parameter, $J(AB) \in \Re^+$, defines the condition that, if two ingredients A and B are separated by an empty cell, ingredient A moves toward or away from ingredient B.
- The *"free-moving"* probability $P_m(A) \in [0,\ 1]$ of an ingredient A defines the ingredient's propensity to move on the grid more rapidly or slowly.

Transition rules: Transition rules define the condition that, during an iteration, an ingredient will transform to some other species. These rules take several forms [120]:

- The *"simple first-order transition"* probability $P_T(AB) \in [0,\ 1]$ defines the condition that an ingredient of species A will change to species B.
- The *"reaction"* probability $P_R(AB \to CD) \in [0,\ 1]$ defines the condition that ingredients A and B will transform into ingredients C and D, respectively, in case they *"encounter"* each other during their movements in the grid.

Other types of rules may also be considered [69,120].

The critical characteristics of all these rules are their local nature, focusing solely on the ingredient in question and potentially any nearby ingredients.

All the probabilities listed above are enforced using a random-number generator in the CA algorithm. For example, suppose we use a random-number generator which generates numbers in the interval $[0,\ 1]$. Suppose that one of the above rule probabilities $P_\cdot(A)$ is set to $P_\cdot(A) = p$ and that the random-number generator generates a number r; the ingredient A can make the "move" prescribed by the rule if $r < p$ and "cannot move" otherwise.

The initial conditions: The remaining conditions of the simulation must be determined after the grid type, size, and governing rules have been established, the latter by giving particular values to the previously mentioned parameters. These consist of (1) the types and quantities of the beginning ingredients; (2) the setup of the system's initial state; (3) the number of simulation runs that are to be performed; and (4) the duration of the runs or the number of iterations they should contain.

About points (3) and (4) above, it needs to be emphasized that when the CA rules are stochastic, i.e., probabilistic, each simulation run is, in effect, an independent "experiment". This implies that the outcomes of different runs could theoretically be different. A single ingredient's activity is typically totally unpredictable. But for the majority of cases, we will look at which *collective outcome*—from a run with a lot of ingredients or from a lot of runs with few ingredients—tends to show a similar pattern. As a result, two further simulation-related details must be determined: the number of independent runs that must be completed and their duration (in iterations). These numbers will be heavily influenced by the type of simulation that will be run. In certain situations, it will be preferable to let the runs continue for a sufficient amount of time in order to reach an equilibrium or steady-state condition [120].

6. Numerical Methods and Algorithms

6.1. Models at the Macro-Scale

This section will describe the methods and techniques that can be used to discretize the continuous problems introduced in Section 5.1 and then compute the solution of the

discretized problem as an approximation of the continuous one. The themes can be explored further by reading [125–127].

6.1.1. Discretization in the Space-Time Domain

Numerical techniques called the finite-element, finite-difference, and finite-volume methods (FEM, FDM, and FVM, respectively) are used to discretize Partial Differential Equations (PDEs) into the space domain.

Nevertheless, they differ from one another in different ways, and each of them has both advantages and disadvantages. Understanding these distinctions could help to select the method most appropriate for the purposes. In the following, a description of the ideas on the basis of each of those methods is given. See [127–129] for other discussions about finite elements versus finite volumes versus finite differences.

Finite Elements A strong computational method for solving differential and integral equations that come up in many applied scientific and engineering domains is the Finite Element Method (FEM) [125].

The fundamental idea behind the FEM is to consider a given domain as a collection of basic geometric shapes called finite elements, for which the approximation functions required to solve a differential equation can be generated systematically. Indeed, the solution u of a differential equation can be approximated, on each element e, by a linear combination of unknown parameters c_j and appropriately selected functions ϕ_j:

$$u^h(x) = \sum_{i=1}^{l} c_i \phi_i(x), \text{ on } \Omega^e. \tag{86}$$

For a given differential equation, it is possible to develop different finite element models, depending on the choice of a particular type of approximation method (e.g., Galerkin, weak-form Galerkin, least-squares, subdomain, collocation, and so on). The finite element model is a set of algebraic relations among the unknown parameters c_j of the approximation: so, solving the problem to find an approximation u^h of u requires the solution of a system of algebraic equations with the unknown parameters c_j.

The major steps in the finite element formulation and analysis of a typical problem are as follows.

1. Discretization of the domain into a set of selected finite elements. This is accomplished by subdividing the given domain $\bar{\Omega} = \Omega \cup \Gamma$ into a set of subdomains $\bar{\Omega}^e = \Omega^e \cup \Gamma^e$ where $\Gamma = \partial\Omega$ and $\Gamma^e = \partial\Omega^e$, called finite elements (see Figure 8a). The phrase "finite element" often refers to both the geometry of the element and degree (or order) of approximation: The form of the element Ω^e can be either triangle or quadrilateral, and the degree of interpolation over it can take on various forms, including linear and quadratic. The finite element mesh of the domain Ω is the non-overlapping total (or assembly) of all elements used to represent the actual domain, and it is represented by Ω_h. In general, Ω_h may not equal the actual domain Ω because of its potentially complex geometry. Nodal points, also known as nodes, are typically taken at appropriate positions in the structure, usually to simplify the element. They are used to characterize the individual elements and subsequently the complete mesh structure (see Figure 8b).
2. Construction of a statement, often a weighted-integral in a weak-form statement according to Weighted Residual Methods (WRMs) [130], which is equivalent (in some sense) to the differential equation to be analyzed over a typical element. A general class of techniques, called the Weighted Residual Method, was created to obtain an approximate solution to the problem of the form

$$L(u(x)) = f(x), \text{ on } \Omega \tag{87}$$

where $L(u)$ is a general linear differential operator. If u^h is just an approximation of the true solution function u of (87), then an error or residual will exist such that

$$R(x) = L\left(u^h(x)\right) - f(x) \neq 0 \tag{88}$$

The idea at the basis of WRM is to force the residual to zero in some average sense over the domain Ω, namely,

$$\int_\Omega R(x) w_i(x) dx = 0,\ i = 1, \ldots, l, \tag{89}$$

where $w_i(x)$ values are the so-called *weight functions*. Depending upon the nature of the weight function, different types of Weighted Residual Methods can be used. The Point Collocation Method, Subdomain Collocation Method, Least Square Method, and Galerkin Method are a few of the common ones. The trial functions themselves are selected as the weight functions in the Galerkin variant of the Weighted Residual Method. So, in the Galerkin method, we set $w_i = \phi_i, \forall i = 1, \ldots, l$ and the weak-form Galerkin model, for each element e is the following set of algebraic relations obtained from (89) and (86):

$$\sum_{j=1}^{l} c_j \left[\int_\Omega L\left(\phi_j(x)\right) \phi_i(x) dx \right] = \int_\Omega f(x) \phi_i(x) dx,\ \ i = 1, \ldots, l. \tag{90}$$

or, in matrix form,

$$^e\boldsymbol{M}\,^e\boldsymbol{c} = {}^e\boldsymbol{f}, \tag{91}$$

where

$$\begin{aligned} ^e\boldsymbol{M} &= \left[\int_\Omega L\left(\phi_j(x)\right) \phi_i(x) dx \right]_{i,j=1,\ldots,n}, \\ ^e\boldsymbol{c} &= \left[c_1, \ldots, c_l\right]^T, \\ ^e\boldsymbol{f} &= \left[\int_\Omega f(x) \phi_i(x) dx \right]^T_{i=1,\ldots,l}, \end{aligned}$$

are, respectively, an $l \times l$ matrix and two vectors of length l. We assume that the vectors $^e\boldsymbol{f}$ also include both the "essential" and the "natural" boundary conditions (see [125] for details). The drawback of this model for second- and higher-order differential equations is that the approximation functions should be differentiable as many times as the actual solution $u(x)$.

The weak form in Equation (87) requires that the approximation chosen for $u(x)$ should be at least linear in x. In addition, it requires that $u(x)$ be made continuous across the elements. The approximation functions $\phi_i(x)_{i=1,\ldots,l}$, in each element e, are chosen to have the so-called *interpolation property*, that is,

$$\phi_i\left({}^e x_j\right) = \delta_{ij}, \forall i, j = 1, \ldots, l, \tag{92}$$

where $\left\{{}^e x_j\right\}_{j=1,\ldots,l}$ are the nodal points of the element e. The scalar δ_{ij} is defined as follows :

$$\delta_{ij} = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j \end{cases}$$

Then, the values of c_j in Equation (86) coincide with the values that u^h takes on at the same nodal points:

$$c_j = u^h({}^e x_i).$$

Therefore, solving the equation (91) allows one to identify the values that the approximation u^h takes on in the nodal points of each element e. Functions that

satisfy the interpolation property (92) are known as the Lagrange interpolation functions. The shape and number of nodes of the element are determined by the number l of linearly independent terms in the representation of u^h. Not all geometric shapes qualify as finite element domains. It turns out that triangle- and quadrilateral-like shapes, with an appropriate number of nodes, qualify as elements.

3. Finite element assembly to produce the global system of algebraic equations of the form

$$ {}^{g}\boldsymbol{M}\,{}^{g}\boldsymbol{c} = {}^{g}\boldsymbol{f}, \tag{93} $$

where ${}^{g}\boldsymbol{M}$, ${}^{g}\boldsymbol{c}$, and ${}^{g}\boldsymbol{f}$ are, respectively, am $N_{nodes} \times N_{nodes}$ matrix and two vectors of length N_{nodes} and where N_{nodes} denotes the number of the nodes (the global grid node) of all the $N_{elements}$ elements composing the finite element mesh. Therefore, the assembly procedure represents the realization of the global displacement boundary conditions as well as the inter-element compatibility of the approximation functions and the displacement field. Indeed, each global grid node is shared by different finite elements which all contribute to the solution of the unknown values ${}^{g}\boldsymbol{c}$ on the involved nodes. Let us denote with

$$ \{{}^{g}x_I\}_{I=1,\ldots,N_{nodes}} $$

the set of such global grid nodes; then, a transformation $\tau(i,e)$ exists that *"maps"* the *"local"* index i of the i-th node of the element e into the *"global"* index I of the same node (see Figure 8c). Thanks to the validity of the *"superposition"* property in FEM context due to the linear nature of the problem, for the global matrix ${}^{g}\boldsymbol{M}$ and the global vector ${}^{g}\boldsymbol{f}$, the following equalities are valid:

$$ {}^{g}\boldsymbol{M}_{IJ} = \sum_{e=1}^{N_{nodes}} \left(\sum_{\substack{i:\ \tau(i,e)=I \\ j:\ \tau(j,e)=J}} {}^{e}\boldsymbol{M}_{ij} \right) \tag{94} $$

$$ {}^{g}\boldsymbol{f}_{I} = \sum_{e=1}^{N_{nodes}} \left(\sum_{i:\ \tau(i,e)=I} {}^{e}\boldsymbol{f}_{i} \right) \tag{95} $$

If u is a function defined both in space and time and the linear differential operator $L(u)$ has the following form:

$$ L(u(x)) = \frac{\partial u}{\partial t} + E(u(x)), \tag{96} $$

where $E(u(x))$ does not contain terms with differentiation with respect to the time variable t, then Equation (93) can be rewritten as:

$$ {}^{g}\bar{M}\,{}^{g}\dot{\boldsymbol{u}} + {}^{g}\bar{K}({}^{g}\boldsymbol{u})\,{}^{g}\boldsymbol{u} = {}^{g}\boldsymbol{f}, \tag{97} $$

where $\boldsymbol{u}$ is a vector of the values assumed by the approximation u^h in all the nodal points of the finite elements mesh and where the symbol $\dot{u}$ denotes the partial derivative of the function u respect to time t: $\dot{u} = \frac{\partial u}{\partial t}$. The nonlinear Equation (97) represents an approximation to the original system of partial differential equations, which is discrete in space and continuous in time.

See [131–134] for applications of FEM to some viscoelastic problems. See [125,135,136] for details about the Finite Element Method, including those details related to how the integrals used by these methods are computed.

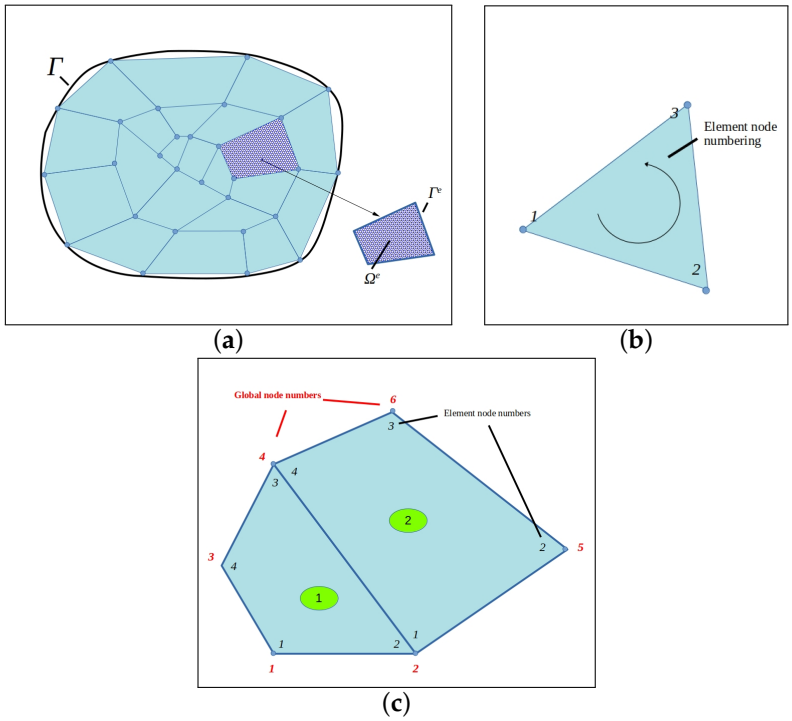


Figure 8. Finite element discretization of a domain (**a**). Linear triangular finite elements (**b**). Global–local correspondence of nodes for the assembly of elements (**c**).

Finite Differences [127] The basic idea of FDM is to compute an approximation u^h of the solution u of (87) by mean an approximation of all the differential operators defining $L(u(x))$. As for FEM, the major steps in the finite element formulation and analysis of a typical problem are:

1. Discretization of the space domain into a set of nodes $\{x_I = [x_1^I, \ldots, x_n^I]\}_{I=1,\ldots,N_{nodes}}$. This is accomplished by subdividing the given domain $\bar{\Omega} = \Omega \cup \Gamma$ by mean of a grid of nodes which represent the finite difference mesh (see Figure 9a).
2. Construction of approximation statements, based on a finite difference, for all the differential operators present in $L(u(x))$. Thanks to the use of Taylor expansion, the validity of the following relations can be demonstrated [137]:

$$\begin{aligned} \frac{\partial^n u}{\partial {x_i}^n}(x) &= \frac{{}_i\Delta_h^n[u](x)}{h^n} + O(h) \\ &= \frac{{}_i\nabla_h^n[u](x)}{h^n} + O(h) \\ &= \frac{{}_i\kappa_h^n[u](x)}{h^n} + O\left(h^2\right), \end{aligned} \tag{98}$$

where ${}_i\Delta_h^n[u](x)$, ${}_i\nabla_h^n[u](x)$ and ${}_i\kappa_h^n[u](x)$ are, respectively, named the n-th order forward, backward, and central differences with respect to x_i and are given by, respectively,

Forward difference

$${}_i\Delta_h^n[u](x) = \sum_{j=0}^{n} (-1)^{n-j} \binom{n}{j} u(x + jh\delta_j) \tag{99}$$

Backward difference

$${}_i\nabla_h^n[u](x) = \sum_{j=0}^{n} (-1)^{j} \binom{n}{j} u(x - jh\delta_j) \tag{100}$$

Central difference

$$ {}_i\kappa_h^n[u](x) = \sum_{j=0}^{n} (-1)^j \binom{n}{j} u\left(x + \left(\frac{n}{2} - j\right) h\delta_j\right) \tag{101}$$

where $\delta_j = \left[\delta_{ij}\right]_{i=1,\ldots,n}$.
By substituting one of the above approximations to each space derivative operator in $L(u(x))$, we obtain the so-called space approximated operator $L_h(u(x))$, which can be written generally as a nonlinear algebraic operator as follows:

$$L_h(u(x)) = \sum_{j}^{m} a_j(u) u\left(x + \alpha j h \delta_j\right) \tag{102}$$

where α and m depend on the type of approximation chosen.
The set of the following N_{nodes} algebraic equations

$$L_h(u(x_I)) = f(x_I), \;\; I = 1, \ldots, N_{nodes} \tag{103}$$

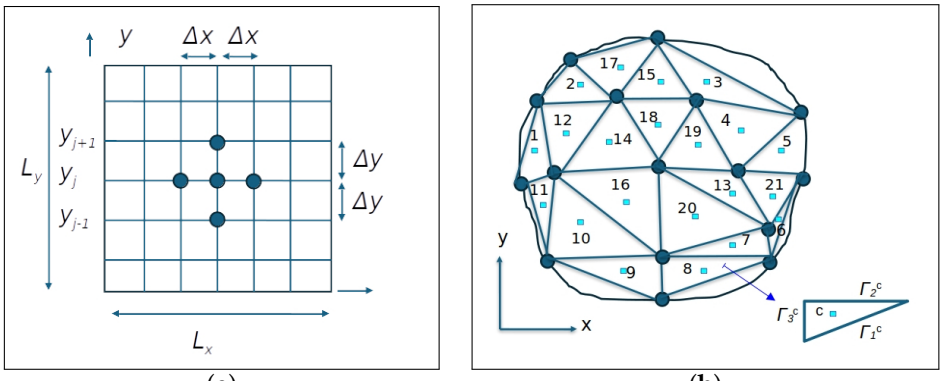
which are obtained by evaluating the equation $L_h(u(x)) = f(x)$ in each node of finite difference mesh can be written, after the inclusion of boundary conditions, in matrix form as

$$\boldsymbol{M}\boldsymbol{u} = \boldsymbol{z}, \tag{104}$$

where $\boldsymbol{M}$, $\boldsymbol{u}$, and $\boldsymbol{z}$ are, respectively, a matrix and two vectors whose generic elements are

$$\begin{aligned} \boldsymbol{M}_{i,j} &= a_j(u(x_i)) \\ \boldsymbol{z}_i &= z(x_i) \\ \boldsymbol{u}_i &= u_h(x_i) \end{aligned}$$

where $i, j = 1, \ldots, N_{nodes}$. Therefore, solving Equation (104) allows us to identify the values at which the approximation u_h takes on the nodal points of the finite element mesh. It is obvious that the accuracy of u_h depends on the type of approximation chosen for the derivative operators that are present in $L(u(x))$.



(**a**) (**b**)

Figure 9. (**a**) Finite difference discretization of a 2D rectangular domain $\Omega = [0,\ L_x] \times [0,\ L_y]$ by a grid of $N_{nodes} = N_x \times N_y$ nodes (distance between two adjacent nodes in the $x-$ and $y-$ directions are, respectively, Δx and Δy). If $h = \Delta x = \Delta y$, nodes are part of a uniform grid mesh. (**b**) Example of a finite volume discretization of a 2D domain. The cell centers x^c of each subdomain Ω^c are denoted by squares and enumerated according to the total number of the domains. The $\Gamma^c_{i=1,2,3}$ denotes the faces of each domain Ω^c.

Finite Volumes [127,138] A space discretization technique called the finite volume approach works well for numerical modeling of several kinds of conservation laws that are represented by PDEs of the form (87) where the linear differential operator $L(u)$ can be expressed as

$$L(u(x)) = div\ \boldsymbol{F}(u(x)). \tag{105}$$

As for FEM and FDM, the major steps in the finite element formulation and analysis of a typical problem are

1. Discretization of the domain into a set of selected subdomains. This is accomplished by subdividing the given domain $\bar{\Omega} = \Omega \cup \Gamma$ into a set of N_{cells} subdomains $\bar{\Omega}^c = \Omega^c \cup \Gamma^c$ where $\Gamma = \partial\Omega$ and $\Gamma^c = \partial\Omega^c$, called control volumes or cells (see Figure 9b), forming the finite volume mesh. Planar surfaces in 3D or straight edges in 2D define the boundaries of the control volumes. Consequently, flat faces or straight edges are used to approximate the bounding surface, if it is curved. Cell faces, or just faces, are the names given to these bounding discrete surfaces.
2. Construction of approximation statements needed to identify the approximation function u^h for the actual solution u of (105). Following the approach described for the FEM framework and based on Weighted Residual Methods, from (89), where the *weight functions* $w_i(x)$ are chosen to be constant and equal to identity, it follows that

$$\int_{\Omega^c} div\ \boldsymbol{F}(u(x))dx = \int_{\Omega^c} f(x)dx \tag{106}$$

for each cell Ω^c of the finite volume mesh. By using the Divergence Theorem, Equation (106) can be rewritten as

$$\oint_{\Gamma^c} \boldsymbol{F}(u(x))\boldsymbol{n}_{\Gamma^c} d\gamma = \int_{\Omega^c} f(x)dx. \tag{107}$$

Thanks to the use of mean value theorem, in each subdomain Ω^c, Equation (107) can be re-written as

$$\oint_{\Gamma^c} \boldsymbol{F}(u(x))\boldsymbol{n}_{\Gamma^c} d\gamma = |\Omega^c|\bar{f}_{\Omega^c}, \tag{108}$$

where $\bar{f}_{\Omega^c}$ are the mean values respectively of functions f in subdomain Ω^c (which can be considered as values of f in cell center x^c) and where the symbol $|\cdot|$ is used to represent the volume/area/length of respectively a 3D/2D/1D space domain. One can substitute the integral on the left side of Equation (108) with a summation over the faces enclosing the c-th cell, resulting in

$$\sum_{f=1,\ldots,N_f^c} \oint_{\Gamma_f^c} \boldsymbol{F}(u(x))\boldsymbol{n}_{\Gamma_f^c} d\gamma = |\Omega^c|\bar{f}_{\Omega^c}, \tag{109}$$

where N_f^c is the number of faces of the c-th cell and Γ_f^c represents the f-th face of the c-th cell. Details about the forms for the restriction of u^h on each Ω^c are available in the literature (i.e., see [139,140]), but, for the sake of semplicity, assume, as in [139], that, on each cell, u^h is a constant function whose value is $\bar{u^c}$; then, Equation (109) can be rewritten as

$$\sum_{f=1,\ldots,N_f^c} \oint_{\Gamma_f^c} \boldsymbol{F}^h(\bar{u^c})\boldsymbol{n}_{\Gamma_f^c} d\gamma = |\Omega^c|\bar{f}_{\Omega^c}, \tag{110}$$

where F^h is an approximation of F that can be defined using a variety of approaches. For this purpose, Taylor series expansions have historically been employed [141]. The end result of approximating F by F^h in terms of cell

center values (that is, the constant values $\bar{u^c}$), followed by substitution into Equation (110) and by the boundary conditions incorporation, is a set of discrete linear algebraic equations of the form

$$\mathcal{F}\boldsymbol{u} = \boldsymbol{h} \tag{111}$$

where $\mathcal{F}$, $\boldsymbol{u}$, and $\boldsymbol{h}$ are, respectively, an $N_{cells} \times N_{cells}$ matrix and two N_{cells} vectors. The solution u of (111) allows us to calculate the values of $u_h(x^c)$ for all the N_{cells} representing the finite volume mesh.

For further information, we recommend reading [138,139,141]

The same considerations about partial differential equations that are discrete in space and continuous in time, which were already made for finite elements, can be repeated in the case of finite difference and finite volume discretizations. An attempt at comparison between the different discretization methods, according to [127–129], can lead to the considerations listed in Table 7. Table 7 also contains, by way of example, references to the literature already introduced in Section 3 and regarding the use of discretization methods (where such information is available).

Table 7. Comparison, from the numerical point of view, between space discretization methods in terms of their *"Pros and Cons"*. The table contains, by way of example, references to the literature already introduced in Section 3 and regarding the use of discretization methods (where such information is available).

Discretization Methods	*"Pros"*	*"Cons"*	Refs
Finite Element Method (FEM)	• Possibility to increase the accuracy of the solution around a "critical" part of the domain: – By "increasing the order of the elements", approximating the physics fields with higher order polynomials, – By adaptive mesh refinement. The finer the mesh, the more accurate approximation can be obtained. • Irregular domain geometries can be considered.	The mathematics needs are advanced.	[11,30,33,35]
Finite Difference Method (FDM)	• Increased "approximation order" and accuracy are made simple on each dimension basis. • Space domain with box-shaped geometry can be discretized by a regular grid, useful for very large-scale simulations.	Managing curved boundaries or material discontinuities might be challenging.	[46]
Finite Volume Method (FVM)	• Only the cell borders' flux evaluation needs to be completed. • Helpful for nonlinear problems (transport problems). • Refining the mesh can improve local accuracy (around a corner of interest).	The functions that approximate the solution cannot be easily made of higher order.	[31]

It is noteworthy that each of the space discretization methods described above can be re-formulated via the other ones (i.e., see [127]).

6.1.2. Solution of the *"Discrete in Space"* Model

1. **The Time Integration algorithm** The proposed space discretization methods allow the construction of similar discrete problems since they all lead to a generally non-linear system of algebraic equations in the following form:

$$\boldsymbol{M}\dot{\boldsymbol{u}} + \boldsymbol{K}(\boldsymbol{u})\boldsymbol{u} = \boldsymbol{f} \tag{112}$$

which represents an ordinary differential equation (ODE) with respect to the time variable. Depending on the type of the approximation of the time derivative present in the left term of Equation (112) and the discretization of the temporal domain Θ, different methods are available to solve the problem described by Equation (112), which is discrete in space and continuous in time.
Assuming that Θ is the time interval $\Theta = [0,\ T]$ and that it is discretized by a set of $N_T + 1$ equally spaced points $\{t_n\}_{n=0,\ldots,N_T}$, where $t_n = \Delta t * n$ and $\Delta t = \frac{T}{N_T}$, then there exist different methods—called *"Time Integration Schemes (TIS)"*—to compute the approximation u_{n+1} of u at the time t_{n+1} as a function of just the approximation of u and $\dot{u}$ at the time t_n (explicit methods) or as a function of u and $\dot{u}$ at the times t_n and t_{n+1} (implicit methods).

$$\begin{aligned} u_{n+1} &= f(u_n, \dot{u}_n,) && \text{explicit schema} \\ u_{n+1} &= f(u_n, \dot{u}_n, u_{n+1}) && \text{implicit schema} \end{aligned}$$

See [142] for a classification of such schemes.
Among the rich set of available TIS, a one-parameter family of methods, called the α-family, is commonly used [143]. In this family of methods, a weighted average of time derivative $\dot{u}$ of a dependent variable u is approximated at two consecutive time steps by a linear interpolation of the values of the same variable u at the same two steps:

$$(1-\alpha)\dot{u}_n + \alpha\dot{u}_{n+1} = \frac{u_{n+1} - u_n}{\Delta t}. \tag{113}$$

By substituting Equation (113) into Equation (112), the following relation is obtained

$$\hat{\boldsymbol{K}}_{n+1}(\boldsymbol{u}_{n+1})\boldsymbol{u}_{n+1} = \hat{\boldsymbol{f}}_{n,n+1}, \tag{114}$$

where

$$\begin{aligned} \hat{\boldsymbol{K}}_{n+1}(\boldsymbol{u}_{n+1}) &= \boldsymbol{M} + a_1\boldsymbol{K}_{n+1} && (115) \\ \hat{\boldsymbol{f}}_{n,n+1} &= \Delta t[\alpha\boldsymbol{f}_{n+1} + (1-\alpha)\boldsymbol{f}_n] + \bar{\boldsymbol{K}}_n\boldsymbol{u}_n && (116) \\ \bar{\boldsymbol{K}}_n &= \boldsymbol{M} - a_2\boldsymbol{K}_n && (117) \end{aligned}$$

and where

$$\begin{aligned} \boldsymbol{K}_i &= \boldsymbol{K}(\boldsymbol{u}_i),\ \forall i = 0,\ldots,N_T, && (118) \\ \boldsymbol{f}_i &= \boldsymbol{f}(t_i),\ \forall i = 0,\ldots,N_T, && (119) \\ a_1 &= \alpha\Delta t, && (120) \\ a_2 &= (1-\alpha)\Delta t. && (121) \end{aligned}$$

Since at time $t = 0$ (i.e., $n = 0$), the right-hand side is computed using the initial values defined in the time boundary conditions, and since the vector $\boldsymbol{f}$ is always known, for both times t_n and t_{n+1}, then the approximation u_{n+1} of u at time t_{n+1} (when $n = 0,\ldots,N_T - 1$) can be computed by Algorithm 1.
For different values of the parameter α, several well-known time approximation schemes are obtained:

$\alpha = 0$, **the forward difference Euler explicit scheme**. The problem (114) is linear, the schema is conditionally stable, and its order of accuracy (see Definition 2 for a definition of the *"order of accuracy"* terms) is $O(\Delta t)$;

$\alpha = 0.5$, **the Crank–Nicolson implicit scheme**. The problem (114) is nonlinear, the schema is unconditionally stable, and its order of accuracy is $O\left((\Delta t)^2\right)$;

$\alpha = \frac{2}{3}$, **the Galerkin implicit scheme**. The problem (114) is nonlinear, the schema is unconditionally stable, and its order of accuracy is $O\left((\Delta t)^2\right)$;

$\alpha = 1$, **the backward difference Euler implicit scheme**. The problem (114) is nonlinear,the schema is unconditionally stable, and its order of accuracy is $O(\Delta t)$.

Algorithm 1 The algorithm implementing the α-family methods for the approximated solution of the discrete in space and continuous in time problem described by Equation (112).

1: **procedure** TIMEINTEGRATOR($\boldsymbol{M}$, $\boldsymbol{K}_0$, $\boldsymbol{K}_1$, $\boldsymbol{u}_0$, $\boldsymbol{u}_1$, $\boldsymbol{f}_0$, $\boldsymbol{f}_1$, α, T, N_T, $\boldsymbol{u}_{N_T}$)
2: **Input:** $\boldsymbol{M}$, $\boldsymbol{K}_0$, $\boldsymbol{K}_1$, $\boldsymbol{u}_0$, $\boldsymbol{u}_1$, $\boldsymbol{f}_0$, $\boldsymbol{f}_1$, T, N_T
3: **Output:** $\boldsymbol{u}_{N_T}$
4: **Compute** Δt, a_1 and a_2
5: **for** $n = 1$ to $N_T - 1$ **do**
6: $\hat{\boldsymbol{K}}_n(\boldsymbol{u}_n) \leftarrow$ **Compute** $\hat{\boldsymbol{K}}_n(\boldsymbol{u}_n) = \boldsymbol{M} + a_1\boldsymbol{K}_n$
7: $\bar{\boldsymbol{K}}_{n-1} \leftarrow$ **Compute** $\bar{\boldsymbol{K}}_{n-1} = \boldsymbol{M} - a_2\boldsymbol{K}_{n-1}$
8: $\hat{\boldsymbol{f}}_{n-1,n} \leftarrow$ **Compute** $\hat{\boldsymbol{f}}_{n-1,n} = \Delta t[\alpha \boldsymbol{f}_n + (1-\alpha)\boldsymbol{f}_{n-1}] + \bar{\boldsymbol{K}}_{n-1}\boldsymbol{u}_{n-1}$
9: $\boldsymbol{u}_{n+1} \leftarrow$ **Solve** $\hat{\boldsymbol{K}}_n(\boldsymbol{u}_n)\boldsymbol{u}_{n+1} = \hat{\boldsymbol{f}}_{n-1,n}$
10: $\boldsymbol{K}_{n+1} \leftarrow$ **Compute** $\boldsymbol{K}_{n+1} = \boldsymbol{K}(\boldsymbol{u}_{n+1})$
11: $\boldsymbol{f}_{n+1} \leftarrow$ **Compute** $\boldsymbol{f}_{n+1} = \boldsymbol{f}(t_{n+1})$
12: **end for**
13: **end procedure**

For $\alpha \geq 0.5$, the scheme is stable, and for $\alpha < 0.5$, the scheme is stable only if the time step meets the following restrictions (i.e., conditionally stable schemes) (see Reddy [136]):

$$\Delta t < \frac{2}{(1-2\alpha)\lambda_{max}}, \ \alpha < \frac{1}{2} \tag{122}$$

where the greatest eigenvalue of the eigenvalue problem related to the matrix Equation (114) is denoted by λ_{max}:

$$\left|\hat{\boldsymbol{K}}_{n+1}(\boldsymbol{u}_{n+1}) - \lambda \boldsymbol{I}\right| = 0.$$

See [144] for further definitions and descriptions of the *"stability"* and *"accuracy"* terms. Generally, Algorithm 1 requires the solution of a nonlinear problem (see line 9) that can be solved by methods described in the next point 2. If the problem degenerates into a linear one (i.e., when $a_1 = 0$) its solution can be obtained by methods described at next point 3.

2. **The Non-linear problem solver** Let us consider the following nonlinear algebraic system:

$$\boldsymbol{G}(\boldsymbol{u}) = \boldsymbol{0}, \tag{123}$$

where $\boldsymbol{G} = [G_i(\boldsymbol{u})]_{i=1,\ldots,M}^T$ and $\boldsymbol{u} = [u_i]_{i=1,\ldots,M}^T$ are vectors in $\Re^M$. Assuming the existence of a solution $\bar{\boldsymbol{u}}$ for the system (123), an approximation of that solution can be obtained by different methods. The *"Newton–Raphson methods"*, and their derivatives, should be cited as the most commonly used [145]. Such methods iteratively compute an approximation $\boldsymbol{u}_{n+1}$ of $\bar{\boldsymbol{u}}$, and they are based on a Taylor series development of the left term of (123) at an already known state $\boldsymbol{u}_n$

$$\boldsymbol{G}(\boldsymbol{u}_n + \Delta u) \approx \boldsymbol{G}(\boldsymbol{u}_n) + D[\boldsymbol{G}(\boldsymbol{u}_n)]\Delta \boldsymbol{u}, \tag{124}$$

where $D[\boldsymbol{G}(\boldsymbol{u}_n)]$ is the Jacobi matrix of $\boldsymbol{G}$ at $\boldsymbol{u}_n$, i.e.,

$$\boldsymbol{J}_n = D[\boldsymbol{G}(\boldsymbol{u}_n)] = \left[\frac{\partial G_i(\boldsymbol{u}_n)}{\partial u_j}\right]_{i,j=1,\ldots,M}.$$

Let $\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta u$; then

$$\boldsymbol{G}(\boldsymbol{u}_{n+1}) \approx \mathbf{0} \iff \boldsymbol{G}(\boldsymbol{u}_n) + \boldsymbol{J}_n \Delta \mathbf{u} \approx \mathbf{0}. \tag{125}$$

Therefore, an approximation $\boldsymbol{u}_N$ of $\bar{\boldsymbol{u}}$ can be computed using the iterative algorithm represented in Algorithm 2. The rate of convergence of the Newton-Raphson scheme is characterized by the following inequality:

$$\|\boldsymbol{u}_{n+1} - \bar{\boldsymbol{u}}\| \leq C\|\boldsymbol{u}_n - \bar{\boldsymbol{u}}\|^2.$$

Algorithm 2 The algorithm implementing the *"Newton–Raphson methods"* for the approximated solution of Equation (123).

1: **procedure** NEWTONSOLVER($\boldsymbol{G}$, $\boldsymbol{u}_0$, N, $\boldsymbol{u}_N$)
2: **Input:** $\boldsymbol{G}$, $\boldsymbol{u}_0$, N
3: **Output:** $\boldsymbol{u}_N$
4: **for** $n = 0$ to $N - 1$ **do**
5: $\boldsymbol{G}(\boldsymbol{u}_n) \leftarrow$ **Compute** $\boldsymbol{G}(\boldsymbol{u}_n)$
6: $\boldsymbol{J}_n \leftarrow$ **Compute** $\boldsymbol{J}_n$
7: $\Delta \boldsymbol{u}_n \leftarrow$ **Solve** $\boldsymbol{J}_n \Delta \boldsymbol{u}_n = -\boldsymbol{G}(\boldsymbol{u}_n)$
8: $\boldsymbol{u}_{n+1} \leftarrow$ **Compute** $\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta \boldsymbol{u}_n$
9: **end for**
10: **end procedure**

Depending on the approach used to approximate/compute the Jacobi-matrix (or tangent matrix) $\boldsymbol{J}_n$ of $\boldsymbol{G}(\boldsymbol{u}_n)$, different derivations of the scheme can be obtained: Table 8 lists some of them. Also, when the application of the BFGS-update is considered, Algorithm 2 can be used. Only the equation system at the line 7 of Algorithm 2 has to be reformulated by introducing the BFGS-update of the inverse of the secant matrix. See Algorithm 3 for details on the BFGS reformulation of Algorithm 2.
Algorithms 2 and 3 require the solution of a linear problem (see, respectively, lines 7 and 6) that can be solved by methods described in the next point 3.
Details about convergence and other numerical issues are available in the literature (see [145]).

Table 8. The Newton–Raphson-like schemes obtained from the original formulation by using different approaches to approximate/compute the Jacobi-matrix of $\boldsymbol{G}$.

Discrete Newton	The matrix $\boldsymbol{J}_n$ is approximated by $\boldsymbol{J}_n^{DN}$ where $$\boldsymbol{J}_n^{DN} = \left[{}^n\boldsymbol{j}_k^{DN} \right]_{k=1,\ldots,M},$$ and where $${}^n\boldsymbol{j}_k^{DN} = \frac{1}{h_k}(\boldsymbol{G}(\boldsymbol{u}_n + \epsilon_k \boldsymbol{\delta}_k) - \boldsymbol{G}(\boldsymbol{u}_n)).$$ ${}^n\boldsymbol{j}_k^{DN}$ should be considered an approximation of the k-th column of $\boldsymbol{J}_n$ obtained by a difference finite method with step size ϵ_k. The value of ϵ_k must be selected so that the tangent matrix approximation is as accurate as feasible. Its value should be very tiny.
Modified Newton	The matrix ${\boldsymbol{J}^{MN}}_n$ substitutes $\boldsymbol{J}_n$. It is considered to be equal to $\boldsymbol{J}_0$ for all steps n, i.e., $${\boldsymbol{J}^{MN}}_n = \boldsymbol{J}_0, \ \forall n = 1, \ldots, N$$
Quasi-Newton	The tangent matrix $\boldsymbol{J}_n$ is approximated by the *"secant matrix"* $\boldsymbol{J}_n^{QN}$ for which the following relation is valid: $$\boldsymbol{J}_n^{QN}(\boldsymbol{u}_n - \boldsymbol{u}_{n-1}) = -(\boldsymbol{G}(\boldsymbol{u}_n) - \boldsymbol{G}(\boldsymbol{u}_{n-1})).$$ Letting $\boldsymbol{K}_n^{QN} = \left(\boldsymbol{J}_n^{QN}\right)^{-1}$, then $\boldsymbol{K}_n^{QN}$ can be considered an approximation of the inverse of the tangent matrix $\boldsymbol{J}_n$ for which the following relation is valid : $$\boldsymbol{K}_n^{QN}\boldsymbol{g}_n = \boldsymbol{w}_n$$ where $\boldsymbol{w}_n = \boldsymbol{u}_n - \boldsymbol{u}_{n-1}$ and $\boldsymbol{g}_n = -(\boldsymbol{G}(\boldsymbol{u}_n) - \boldsymbol{G}(\boldsymbol{u}_{n-1}))$. A number of update algorithms exist for the explicit determination of $\boldsymbol{K}_n^{QN}$. Here, the BFGS method will be presented since it has been observed to be successfully applied in the solution of problems from Finite Element Methods. It was based on the following update for $\boldsymbol{K}_n^{QN}$: $$\boldsymbol{K}_n^{QN} = \left(\mathbf{1} + \boldsymbol{a}_n \boldsymbol{b}_n^T\right)\boldsymbol{K}_{n-1}^{QN}\left(\mathbf{1} + \boldsymbol{b}_n \boldsymbol{a}_n^T\right),$$ where $$a_n = \frac{1}{\boldsymbol{g}_n^T \boldsymbol{w}_n}\boldsymbol{w}_n,$$ $$b_n = -\left\{ \boldsymbol{g}_n - \left[-\frac{\boldsymbol{w}_n^T \boldsymbol{g}_n}{\boldsymbol{w}_n^T \boldsymbol{G}(\boldsymbol{u}_{n-1})} \right]^{-\frac{1}{2}} \boldsymbol{G}(\boldsymbol{u}_{n-1}) \right\}.$$

Algorithm 3 The algorithm implementing the *"BFGS"* reformulation of the Newton–Raphson method.

procedure BFGSNEWTONSOLVER($\boldsymbol{G}$, $\boldsymbol{u}_0$, N, $\boldsymbol{u}_N$)
Input: $\boldsymbol{G}$, $\boldsymbol{u}_0$, N
Output: $\boldsymbol{u}_N$
  $\boldsymbol{g}_1 \leftarrow$ **Compute** $\boldsymbol{g}_1 = -\boldsymbol{G}(\boldsymbol{u}_0)$
  $\boldsymbol{J}_0^{QN} \leftarrow$ **Compute** $\boldsymbol{J}_0^{QN}$
  $\boldsymbol{K}_0^{QN} \leftarrow$ **Solve** $\boldsymbol{J}_0^{QN}\boldsymbol{K}_0^{QN} = \boldsymbol{I}$
  $\boldsymbol{w}_1 \leftarrow$ **Compute** $\boldsymbol{w}_1 = \boldsymbol{K}_0^{QN}\boldsymbol{g}_1$
  for $n = 1$ to $N-1$ **do**
    $\boldsymbol{u}_n \leftarrow$ **Compute** $\boldsymbol{u}_n = \boldsymbol{u}_{n-1} + \boldsymbol{w}_n$
    $\boldsymbol{G}_n \leftarrow$ **Compute** $\boldsymbol{G}_n = \boldsymbol{G}(\boldsymbol{u}_{n-1})$
    $\boldsymbol{a}_n \leftarrow$ **Compute** $\boldsymbol{a}_n = \frac{1}{\boldsymbol{g}_n^T\boldsymbol{w}_n}\boldsymbol{w}_n$
    $\boldsymbol{b}_n \leftarrow$ **Compute** $\boldsymbol{b}_n = -\left\{\boldsymbol{g}_n - \left[-\frac{\boldsymbol{w}_n^T\boldsymbol{g}_n}{\boldsymbol{w}_n^T\boldsymbol{G}_n}\right]^{-\frac{1}{2}}\boldsymbol{G}_n\right\}$.
    $\boldsymbol{K}_n^{QN} \leftarrow$ **Compute** $\boldsymbol{K}_n^{QN} = (\boldsymbol{1} + \boldsymbol{a}_n\boldsymbol{b}_n^T)\boldsymbol{K}_{n-1}^{QN}(\boldsymbol{1} + \boldsymbol{b}_n\boldsymbol{a}_n^T)$,
    $\boldsymbol{w}_{n+1} \leftarrow$ **Compute** $\boldsymbol{w}_{n+1} = \boldsymbol{K}_n^{QN}\boldsymbol{g}_{n+1}$
  end for
  $\boldsymbol{u}_N \leftarrow$ **Compute** $\boldsymbol{u}_N = \boldsymbol{u}_{N-1} + \boldsymbol{w}_N$
end procedure

3. **The linear problem solver**: Let us consider the linear problem of the form

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}, \tag{126}$$

where $\boldsymbol{A} = \left[a_{ij}\right]_{i,j=1,\ldots,M}^T$, $\boldsymbol{y} = \left[y_i\right]_{i=1,\ldots,M}^T$ and $\boldsymbol{x} = \left[x_i\right]_{i=1,\ldots,M}^T$ are, respectively, a matrix and two vectors in $\Re^{M\times M}$ and $\Re^M$. Since the matrices coming from the discretization of problems of interest for this review are sparse if not even structured—for example, see matrices generated for the solution of Navier–Stokes equations for viscoelastic fluids in Section 6.1.3, which have a lot of zero elements (or have zero blocks)—particular attention will be spent in this work to describe methods to be used for the solution of (126) when $\boldsymbol{A}$ is sparse (or structured) (see [146] for a more precise definition of the sparsity concept).
There are three fundamental classes of methods that are used to solve Equation (126):

Direct methods The direct solution of (126) is generally based on a technique called *"LU decomposition"* [147]. This technique consists of factoring the matrix $\boldsymbol{A}$ as in Equation (127) (by algorithms whose computational complexity is generally on the order of M^3), and it is based on the concept that triangular systems of equations are "easy" to solve:

$$\boldsymbol{P}\boldsymbol{A}\boldsymbol{Q}^T = \boldsymbol{L}\boldsymbol{U}, \tag{127}$$

where $\boldsymbol{L}$ and $\boldsymbol{U}$ are, respectively, lower triangular and upper triangular matrices and where $\boldsymbol{P}$ and $\boldsymbol{Q}$ are permutation matrices. A permutation matrix is used to represent row or column interchanges in matrix $\boldsymbol{A}$, and it is obtained from the identity matrix $\boldsymbol{I}$ by applying on it the same sequence of row or column interchanges. The row or column permutations on $\boldsymbol{A}$ are often required for both numerical and sparsity issues [146].
The *LU* decomposition facilitates the solution of (126) effectively, particularly when solving several systems that share the same matrix $\boldsymbol{A}$. In fact, if a matrix $\boldsymbol{A}$ already has a $\boldsymbol{LU}$ decomposition available, the linear system in (126) can be solved by (1) the solution of $\boldsymbol{L}\boldsymbol{z} = \boldsymbol{P}\boldsymbol{y}$ (forward substitution), followed by (2) the solution of $\boldsymbol{U}\boldsymbol{w} = \boldsymbol{z}$ (backward substitution) where $\boldsymbol{w} = \boldsymbol{Q}^T\boldsymbol{x}$. Then, the values

of all the unknowns x_i, $\forall i = 1, \ldots, M$ can be obtained by permuting the rows of $\boldsymbol{w}$ based on the permutation matrix $\boldsymbol{Q}$. The execution of both a forward and a backward substitution has a general computational complexity of $O(2M^2)$.

If the matrix $\boldsymbol{A}$ is symmetric and positive definite (that is, if i$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} > 0$ for all nonzero vectors x), then the $\boldsymbol{LU}$ decomposition of $\boldsymbol{A}$ becomes $\boldsymbol{PAP}^T = \boldsymbol{LL}^T$ and is called *"Cholesky"* factorization.

Numerical issues related to the stability and accuracy of the presented direct solver can be found in [146].

With the aim to present and discuss methods useful in computing the *LU* decomposition of sparse matrices, we propose the *"Frontal methods"*. Such methods have their origin in the solution of finite-element problems, but they are not restricted to this application. They are of great interest in their own right since the extension of these ideas to multiple fronts, regardless of the origin of the problem, will offer a great opportunity for parallelism [146,148,149].

In the context of *"Frontal methods"* the matrix $\boldsymbol{A}$ can be considered as the sum of submatrices of varying sizes, as happens for the matrices that are the result of the assembly process underlying the discretization using finite elements. That is, each entry is computed as the sum of one or more values:

$$\boldsymbol{A}_{ij} = \sum_{l=1,\ldots,L} \boldsymbol{A}_{ij}^{[l]}, \ \forall i, j = 1, \ldots, M. \tag{128}$$

Entry $\boldsymbol{A}_{ij}$ in (128) is said to be fully summed (or fully assembled) if all the operations (128) have been performed, and the index of the unknown k is said to be fully summed if all the $\boldsymbol{A}$ entries in its row (i.e., $\boldsymbol{A}_{kj}$, $\forall j = 1, \ldots, M$) and its column (i.e., $\boldsymbol{A}_{ik}$, $\forall i = 1, \ldots, M$) are fully summed.

The fundamental notion of frontal techniques is to limit decomposition operations to a frontal matrix $\boldsymbol{F}$, on which Level 3 BLAS [150] is used to execute dense matrix operations. (The BLAS (Basic Linear Algebra Subprograms) are routines that provide optimized standard building blocks for performing primary vector and matrix operations. BLAS routines can be classified depending on the type of operands: Level 1: operations involving just vector operands; Level 2: operations between vectors and matrices; and Level 3: operations involving just matrix operands.) In the frontal scheme, the factorization proceeds as a sequence of partial factorization on frontal matrices $\boldsymbol{F}^{[l]}$, which can be represented, up to some row or column permutations, by a 2×2 blocks structure:

$$\boldsymbol{F}^{[l]} = \begin{bmatrix} \boldsymbol{F}_{11}^{[l]} & \boldsymbol{F}_{12}^{[l]} \\ \boldsymbol{F}_{21}^{[l]} & \boldsymbol{F}_{22}^{[l]} \end{bmatrix}. \tag{129}$$

where $\boldsymbol{F}_{11}^{[l]}$ is related to the unknown indices that are fully summed.

The *LU* decomposition by Frontal methods can be described by Algorithm 4.

A variation of frontal solvers is the *Multifrontal method*. It could be considered an improvement of the frontal solver, which, based on an appropriate order or coupling of the addends in the summation in (128), can lead to the use of several independent fronts paving the way, naturally, to the development of parallel algorithms. Efficient parallel and sequential decomposition algorithms based on Frontal and Multifrotal methods should be able to perform an analysis of the structure of matrix $\boldsymbol{A}$ to define all the needed reorganizations of its row and columns (i.e., by the definition of appropriate permutations) to better exploit sparsity, preserve *"good"* numerical properties, and define the order of front assembly. See [146] for details about techniques to be used in such analyses.

Algorithm 4 The algorithm implementing the *"Frontal method"*.

procedure FRONTALDECOMPOSITION($\left\{\boldsymbol{A}^{[l]}\right\}_{l=1,\ldots,L}$, L)
Input: $\left\{\boldsymbol{A}^{[l]}\right\}_{l=1,\ldots,L}$, L
Output: $\boldsymbol{L}$, $\boldsymbol{U}$
    $\boldsymbol{S}^{[l-1]} \leftarrow \boldsymbol{0}$
    for $l = 1$ to L **do**
        $\boldsymbol{F}^{[l]} \leftarrow$ **Compute** $\boldsymbol{S}^{[l-1]} + \boldsymbol{A}^{[l]}$
        $\boldsymbol{L}_{11}^{[l]}, \boldsymbol{U}_{11}^{[l]} \leftarrow$ **Factorize** $\boldsymbol{F}_{11}^{[l]} = \boldsymbol{L}_{11}^{[l]}\boldsymbol{U}_{11}^{[l]}$
        $\boldsymbol{U}_{12}^{[l]} \leftarrow$ **Solve** $\boldsymbol{L}_{11}^{[l]}\boldsymbol{U}_{12}^{[l]} = \boldsymbol{F}_{12}^{[l]}$
        $\boldsymbol{L}_{21}^{[l]} \leftarrow$ **Solve** $\boldsymbol{L}_{21}^{[l]}\boldsymbol{U}_{11}^{[l]} = \boldsymbol{F}_{21}^{[l]}$
        $\boldsymbol{S}^{[l]} \leftarrow$ **Compute** $\boldsymbol{S}^{[l]} = \boldsymbol{F}_{22}^{[l]} - \boldsymbol{L}_{21}^{[l]}\boldsymbol{U}_{12}^{[l]}$
    end for
    $\boldsymbol{L} \leftarrow$ **Form** $\boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_{11}^{[1]} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ & \boldsymbol{L}_{11}^{[2]} & & \boldsymbol{0} \\ & & \ddots & \vdots \\ \boldsymbol{L}_{21}^{[1]} & \boldsymbol{L}_{21}^{[2]} & \cdots & \boldsymbol{L}_{11}^{[L]} \end{bmatrix}$
3ex
    $\boldsymbol{U} \leftarrow$ **Form** $\boldsymbol{U} = \begin{bmatrix} \boldsymbol{U}_{11}^{[1]} & & & \boldsymbol{U}_{12}^{[1]} \\ \boldsymbol{0} & \boldsymbol{U}_{11}^{[2]} & & \boldsymbol{U}_{12}^{[2]} \\ \vdots & & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & & \boldsymbol{U}_{11}^{[L]} \end{bmatrix}$
end procedure

Iterative methods: Direct methods could exploit the sparse linear system structure as much as possible to avoid computations with zero elements and zero-element storage. However, these methods are often too expensive for large systems, except where the matrix $\boldsymbol{A}$ has a special structure. For many problems, the direct solution methods will not lead to solutions in a reasonable amount of time. So, researchers have long tried to iteratively approximate the solution $\boldsymbol{x}$ starting from a *"good"* initial guess $\boldsymbol{x}_0$ for the solution [151].

The Krylov subspace iteration methods (KSMs) are among the most important iterative techniques for solving linear systems because they have significantly altered how users approach huge, sparse, non-symmetric matrix problems. The techniques in question were recognized as one of the top ten algorithms that had the biggest impact on the advancement and application of science and engineering during the 1900s [110], and they seem to have also confirmed their importance in the current century due to the effort spent to improve their effectiveness and efficiency in up-to-date computing architecture.

Like the majority of feasible iterative methods now in use for resolving large linear equation systems, KSM makes use of a projection process. A projection process is the standard method for taking an approximation from a subspace to the solution of a linear system [152].

Let $\mathcal{K}_N$ and $\mathcal{L}_N$ be two N-dimensional subspaces of $\Re^M$. Finding an approximate solution $\boldsymbol{x}_N$ to (126) requires a projection technique onto the subspace $\mathcal{K}_N$ and orthogonal to $\mathcal{L}_N$. This is carried out by imposing the conditions that $\boldsymbol{x}_N$ belongs to $\mathcal{K}_N$ and that the new residual vector is orthogonal to $\mathcal{L}_N$, where $\boldsymbol{x}_0$ is the initial guess to the solution; that is,

$$\text{Find } \boldsymbol{x}_N \in \boldsymbol{x}_0 + \mathcal{K}_N, \text{ such that } \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_N \perp \mathcal{L}_N.$$

Note that if $\boldsymbol{x}_N$ is written in the form $\boldsymbol{x}_N = \boldsymbol{x}_0 + \boldsymbol{\delta}$, and the initial residual vector $\boldsymbol{r}_0$ is defined as $\boldsymbol{r}_0 = \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_0$, then the above condition becames,

$$\text{Find } \boldsymbol{\delta} \in \mathcal{K}_N, \text{ such that } \boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{\delta} \perp \mathcal{L}_N. \tag{130}$$

In other words, the approximate solution can be defined as

$$\text{Find } \boldsymbol{\delta} \in \mathcal{K}_N \text{ such that} \qquad \langle \boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{\delta}, \boldsymbol{w} \rangle = 0, \qquad \forall \boldsymbol{w} \in \mathcal{L}_N \tag{131}$$

$$\text{Compute} \qquad \boldsymbol{x}_N = \boldsymbol{x}_0 + \boldsymbol{\delta}, \tag{132}$$

where the symbol $\langle \cdot, \cdot \rangle$ denotes the Euclidean inner product. In the most general form, this is a basic projection step. A series of these projections is used in the majority of common techniques. A new pair of subspaces, $\mathcal{K}_{N+1}$ and $\mathcal{L}_{N+1}$, are used in a new projection step to compute $\boldsymbol{x}_{N+1}$. An initial guess, $\boldsymbol{x}_N$, is equal to the most recent solution estimate obtained from the previous projection step. Many well-known techniques in scientific computing, such as the more complex KSM procedure or the simpler Gauss–Seidel step, have a unifying framework that is provided by projection methods.

A Krylov subspace method is a method for which the subspace $\mathcal{K}_N = \mathcal{K}_N(\boldsymbol{A}, \boldsymbol{r}_0)$ is the Krylov subspace [152]

$$\mathcal{K}_N(\boldsymbol{A}, \boldsymbol{r}_0) = span\left\{ \boldsymbol{A}^i \boldsymbol{r}_0 \right\}_{i=0,\ldots,N-1}. \tag{133}$$

Variations in the choices of subspace $\mathcal{L}_N$ give rise to different Krylov subspace methods. The most popular methods arise from two general options for $\mathcal{L}_N$.

The first is simply $\mathcal{L}_N = \mathcal{K}_N$ and the minimum residual variation $\mathcal{L}_N = A\mathcal{K}_N$. The most representative among the KSMs is related to the choice $\mathcal{L}_N = A\mathcal{K}_N$, and it will be described in this work: the *"Generalized Minimum Residual Method (GMRES)"*. The second class of methods is based on defining $\mathcal{L}_N$ to be a Krylov subspace method associated with $\boldsymbol{A}^T$, namely, $\mathcal{L}_N = \mathcal{K}_N(\boldsymbol{A}^T, \boldsymbol{r}_0)$ (see [152] for details).

The GMRES method can be used to solve sparse linear systems whose matrix $\boldsymbol{A}$ is general (i.e., not necessarily symmetric or positive definite). Let $\boldsymbol{V}_N$ matrices whose columns are an orthonormal basis for space $\mathcal{K}_N = \mathcal{K}_N(\boldsymbol{A}, \boldsymbol{r}_0)$, then $\boldsymbol{\delta}$ can be expressed as as $\boldsymbol{\delta} = \boldsymbol{V}_N \boldsymbol{\vartheta}$ where $\boldsymbol{\vartheta} \in \mathcal{K}_N$. Furthermore, let $\mathcal{L}_N = A\mathcal{K}_N$ and define

$$\begin{aligned} J_N(\boldsymbol{\vartheta}) = \|\boldsymbol{r}_N\|_2 &= \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_N\|_2, \\ &= \|\boldsymbol{y} - \boldsymbol{A}(\boldsymbol{x}_0 + \boldsymbol{V}_N\boldsymbol{\vartheta})\|_2, \\ &= \|\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{V}_N\boldsymbol{\vartheta}\|_2, \end{aligned} \tag{134}$$

where $\|\cdot\|_2$ is the spectral norm, that is, the matrix norm induced by the Euclidean vector's inner product.

Then, condition (130) (or equivalently the condition (132)) is satisfied by $\tilde{\boldsymbol{x}}_N = \boldsymbol{x}_0 + \boldsymbol{V}_N\tilde{\boldsymbol{\vartheta}}$ if, and only if, $\tilde{\boldsymbol{\vartheta}}$ is the solution to the following minimum problem (see Proposition 5.3 in Saad [152]):

$$\tilde{\boldsymbol{\vartheta}} = argmin_{\boldsymbol{\vartheta} \in \mathcal{K}_N} J_N(\boldsymbol{\vartheta}). \tag{135}$$

A schema for the GMRES iterative algorithm, useful to compute the approximation $\boldsymbol{x}_N$ of the solution for problem (126), is represented in Algorithm 5.

Algorithm 5 The GMRES iterative algorithm.

1: **procedure** GMRES($\boldsymbol{A}$, $\boldsymbol{y}$, $\boldsymbol{x}_0$, N, $\boldsymbol{x}_N$)
2: **Input:** $\boldsymbol{A}$, $\boldsymbol{x}_0$, N
3: **Output:** $\boldsymbol{x}_N$
4: $\boldsymbol{r}_0 \leftarrow$ **Compute** $\boldsymbol{r}_0 = \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{y}$
5: $\boldsymbol{V}_0 \leftarrow$ **Assign** $[\boldsymbol{r}_0]$
6: **for** $n = 1$ to N **do**
7: $\boldsymbol{V}_n \leftarrow$ **Compute an orthonormal basis** $[\boldsymbol{V}_{n-1}, v_n]$ for $\mathcal{K}_n(\boldsymbol{A}, \boldsymbol{r}_0)$
8: $\boldsymbol{\vartheta}_n \leftarrow$ **Compute** $\tilde{\boldsymbol{\vartheta}}_n = argmin_{\boldsymbol{\vartheta} \in \mathcal{K}_N} J_n(\boldsymbol{\vartheta})$
9: $\boldsymbol{u}_n \leftarrow$ **Compute** $\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \boldsymbol{V}_n \tilde{\boldsymbol{\vartheta}}_n$
10: **end for**
11: **end procedure**

Different methods can be used to perform steps at line 7 of Algorithm 5. The most common ones are related with the Gram–Schmidt or Householder procedures, which both compute, at the step n, both the matrix $\boldsymbol{V}_n$ and the $(n+1) \times n$ upper Hessenberg matrix (see [153] for the definition of upper Hessenberg matrix) $\bar{\boldsymbol{H}}_n$, for which the following relation is valid [152]:

$$\boldsymbol{A}\boldsymbol{V}_n = \boldsymbol{V}_{n+1}\bar{\boldsymbol{H}}_n. \tag{136}$$

Thanks to Equation (136) (see Proposition 6.5 in Saad [152]), the function $J_n(\boldsymbol{\vartheta})$ can be rewritten as

$$J_n(\boldsymbol{\vartheta}) = \|\boldsymbol{c} - \bar{\boldsymbol{H}}_n \boldsymbol{\vartheta}\|_2, \tag{137}$$

where $\boldsymbol{c}$ is the constant $n+1$ vector defined as $\boldsymbol{c} = [\|\boldsymbol{r}_0\|_2, 0, \ldots, 0, 0]^T$. The solution to the minimum problem at line 8 of Algorithm 5 is easy and inexpensive to compute since it requires the solution of an $(n+1) \times n$ least-squares problem where n is typically small and where the coefficient matrix $\bar{\boldsymbol{H}}_n$ is an upper triangular matrix.

Although KS methods are well founded theoretically, they are all likely to experience slow convergence when dealing with problems arising from applications where the numerical features of the coefficient matrix $\boldsymbol{A}$ are not *"good"*. The matrix's *"condition number"* is the primary parameter that describes its numerical "goodness". The condition number of a square matrix $\boldsymbol{A}$ is defined as

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|_2 \left\|\boldsymbol{A}^{-1}\right\|_2$$

In numerical analysis, the condition number of a matrix $\boldsymbol{A}$ is a way of describing how well or badly a linear system with $\boldsymbol{A}$ as a coefficient matrix could be numerically solved in an effective way: if $\kappa(\boldsymbol{A})$ is small, the problem is well-conditioned; otherwise, the problem is rather ill-conditioned and it can hardly be solved numerically in an effective way [147].

In these situations, preconditioning is essential to the effectiveness of Krylov subspace algorithms. Preconditioners, in general, are modifications to initial linear systems that make them *"easier"* to solve.

Identifying a preconditioning matrix $\boldsymbol{M}$ is the first stage in the preconditioning process. The matrix $\boldsymbol{M}$ can be defined in a variety of ways, but it must meet a few minimal requirements. The most crucial one is that linear system $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{b}$ must be cheap to solve because the preconditioned algorithms will need to solve linear systems with coefficient matrix $\boldsymbol{M}$ at each stage. Furthermore, $\boldsymbol{M}$ should be nonsingular and *"close"* to matrix $\boldsymbol{A}$ in some sense (i.e., $\boldsymbol{M}^{-1}\boldsymbol{A} \approx \boldsymbol{I}$) [152].

For example, if the preconditioner $\boldsymbol{M}$ is applied from the left, the modification leads to the preconditioned linear system

$$\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{b}, \tag{138}$$

whose coefficients matrix $\boldsymbol{M}^{-1}\boldsymbol{A}$ should have better numerical features than $\boldsymbol{A}$ because it is *"near"* the identity matrix $\boldsymbol{I}$. An incomplete factorization of the original matrix $\boldsymbol{A}$ is one of the easiest ways to define a preconditioner. This involves taking into account the formula $\boldsymbol{A} = \boldsymbol{LU} - \boldsymbol{R}$, in which $\boldsymbol{R}$ represents the residual or factorization error and $\boldsymbol{L}$ and $\boldsymbol{U}$ represent the lower and upper portions of $\boldsymbol{A}$, respectively, with the same nonzero structure. Calculating this incomplete factorization, often known as $ILU(0)$, is not too difficult or expensive.

More sophisticated preconditioners are available, such as the preconditioners based on Domain Decomposition Methods and those based on Multigrid Methods (i.e., see [152] for the basic panoramic of such tools).

It is worth mentioning that if the matrix $\boldsymbol{A}$ is symmetric and positive definite, then specialized algorithms exist in the context of KSM: the Conjugate Gradient (CG) method is the most famous among those [152].

Hierarchical/Multilevel methods: When solving linear systems resulting from discretized partial differential equations (PDEs), preconditioned Krylov subspace algorithms tend to converge more slowly as the systems get bigger. There is a significant loss of efficiency because of this decline in the convergence rate. The convergence rates that the Multigrid method class can achieve, on the other hand, are theoretically independent of the mesh size. Discretized elliptic PDEs are the primary focus of Multigrid methods, which distinguish them significantly from the preconditioned Krylov subspace approach. Later, the technique was expanded in many ways to address nonlinear PDE problems as well as problems not described by PDEs [152]. The most important contribution in this sense is related to the Algebraic Multigrid (AMG) [154]. The main ingredients of multilevel methods to solve (126) are as follows.

(a) A hierarchy

$$\boldsymbol{A}_h \boldsymbol{x}^h = \boldsymbol{y}^h, \; h = 0, \ldots, H \tag{139}$$

of H problems along with restriction $\boldsymbol{I}_{h+1}^h$ and prolongation $\boldsymbol{I}_h^{h+1}$ operators to move between levels, where

$$\boldsymbol{A}_H = \boldsymbol{A}, \boldsymbol{x}^H = \boldsymbol{x}, \text{ and } \boldsymbol{y}^H = \boldsymbol{y}, \tag{140}$$

and where

$$\boldsymbol{A}_h = \boldsymbol{I}_{h+1}^h \boldsymbol{A}_{h+1} \boldsymbol{I}_h^{h+1}, \tag{141}$$

$$\boldsymbol{x}^h = \boldsymbol{I}_{h+1}^h \boldsymbol{x}^{h+1}, \tag{142}$$

$$\boldsymbol{y}^h = \boldsymbol{I}_{h+1}^h \boldsymbol{y}^{h+1}, \tag{143}$$

with

$$\boldsymbol{I}_{h+1}^h = \left(\boldsymbol{I}_h^{h+1}\right)^T$$

for each $h = 0, \ldots, H-1$;

(b) A relaxation operator, $S\left(\boldsymbol{A}_h, \boldsymbol{f}^h, \boldsymbol{f}_0^h, \boldsymbol{b}^h\right)$, for *"smoothing"*. $S\left(\boldsymbol{A}_h, \boldsymbol{f}^h, \boldsymbol{f}_0^h, \boldsymbol{b}^h\right)$ is in general a tool to compute iteratively an approximation for the solution of the linear system $\boldsymbol{A}_h \boldsymbol{f}_h = \boldsymbol{b}_h$ from the initial guess $\boldsymbol{f}_0^h$.

Algorithm 6 describes the recursive implementation schema of AMG methods based on the ingredients listed above, where $\boldsymbol{x}_0^h$ is the initial guess for solution $\boldsymbol{x}^h$.

Algorithm 6 Implementation schema of AMG methods.

1: **function** $\boldsymbol{x}^h$ =AMG(H, $\boldsymbol{A}_h$, $\boldsymbol{x}_0^h$, $\boldsymbol{y}^h$, ν_1, ν_2, γ)
2: **Input:** H, $\boldsymbol{A}_h$, $\boldsymbol{x}_0^h$, $\boldsymbol{y}^h$, ν_1, ν_2, γ
3: **Output:** $\boldsymbol{x}^h$
4: $\boldsymbol{x}^h \leftarrow$ **Pre-smooth** ν_1 times $S^{\nu_1}\left(\boldsymbol{A}_h, \boldsymbol{x}^h, \boldsymbol{x}_0^h, \boldsymbol{y}^h\right)$
5: $\boldsymbol{r}^h \leftarrow$ **Get residual** $\boldsymbol{r}^h = \boldsymbol{y}^h - \boldsymbol{A}_h\boldsymbol{x}^h$
6: $\boldsymbol{r}^{h-1} \leftarrow$ **Restrict residual** $\boldsymbol{r}^{h-1} = \boldsymbol{I}_h^{h-1}\boldsymbol{r}^h$
7: $\boldsymbol{A}_{h-1} \leftarrow$ **Restrict matrix** $\boldsymbol{A}_{h-1} = \boldsymbol{I}_h^{h-1}\boldsymbol{A}_h\boldsymbol{I}_{h-1}^h$
8: **if** $h = 1$ **then**
9: $\delta^0 \leftarrow$ **Solve Coarse System** $\boldsymbol{A}_0\delta^0 = \boldsymbol{r}^0$
10: **else**
11: $\delta^{h-1} = \boldsymbol{0}$
12: **for** $i = 1$ to γ **do**
13: $\delta^{h-1} \leftarrow$ **Recursive Solve** $\delta^{h-1} = \text{AMG}\left(H-1, \boldsymbol{A}_{h-1}, \delta^{h-1}, \boldsymbol{r}^{h-1}, \nu_1, \nu_2, \gamma\right)$
14: **end for**
15: **end if**
16: $\boldsymbol{x}^h \leftarrow$ **Correct by prolongation** $\boldsymbol{x}^h = \boldsymbol{x}^h + \boldsymbol{I}_{h-1}^h\delta^{h-1}$
17: $\boldsymbol{x}^h \leftarrow$ **Post-smooth** ν_2 times $S^{\nu_2}\left(\boldsymbol{A}_h, \boldsymbol{x}^h, \boldsymbol{x}^h, \boldsymbol{y}^h\right)$
18: **return** $\boldsymbol{x}^h$
19: **end function**

The Multigrid schema is defined by the parameter γ, which controls the number of times AMG is iterated in line 13 of Algorithm 6. The V-cycle Multigrid is obtained for the situation $\gamma = 1$. The W-cycle Multigrid is the case where $\gamma = 2$. The illustrations in Figure 10 show how complicated the ensuing inter-grid up and down moves can be. Seldom is the case $\gamma = 3$ used [152].

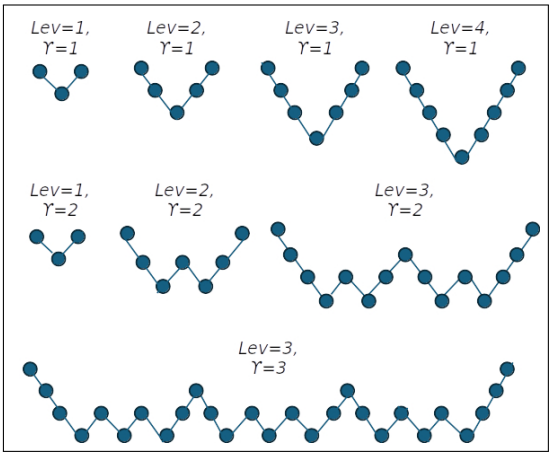


Figure 10. Illustrations of different V and W cycles.

Details about restriction, prolongation, and relaxation operators can be found in Stuben [154].

4. **The Saddle Point Problem solvers** Let us consider the block 2×2 linear systems of the form

$$\begin{bmatrix} A & B_1^T \\ B_2 & -C \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{144}$$

where $A \in \Re^{n\times n}$, $B_1, B_2 \in \Re^{m\times n}$ and $C \in \Re^{m\times m}$. If one or more of the following requirements are met by the constituent blocks A, B_1, B_2, and C, then the linear system (144) defines a (generalized) *"Saddle Point problem"* [155]:

C1 A is symmetric ($A = A^T$);

C2 the symmetric part of A, $H = \frac{1}{2}(A + A^T)$, is positive semidefinite;
C3 $B_1 = B_2 = B$;
C4 C is symmetric and positive semidefinite;
C5 $C = O$.

In addition to the standard differentiation between direct and iterative techniques, generalized Saddle Point problem-solving algorithms can be broadly classified into two groups: *segregated* and *coupled* methods [155].

Segregated methods x_1 and x_2, the two unknown vectors, are computed independently via segregated procedures. This method entails solving two linear systems (referred to as reduced systems), one for each of x_i, that are less in size than $n + m$. Each reduced system in segregated techniques can be addressed using an iterative or direct approach. The Schur complement reduction method, which is based on a block LU factorization of the block 2×2 matrix in Equation (144) (also known as the global matrix $\mathcal{A}$), is one of the primary examples of the segregated approach. Indeed, if A is nonsingular, the Saddle Point matrix admits the following block triangular factorization:

$$\mathcal{A} = \begin{bmatrix} A & B_1^T \\ B_2 & -C \end{bmatrix} = \begin{bmatrix} I & 0 \\ B_2 A^{-1} & I \end{bmatrix} \begin{bmatrix} A & B_1^T \\ 0 & S \end{bmatrix}, \tag{145}$$

where $S = -\left(C + B_2 A^{-1} B_1^T\right)$ is the Schur complement of A in $\mathcal{A}$. It follows that if $\mathcal{A}$ is nonsingular, S is also nonsingular. Using Equation (145), the linear system (144) can be transformed into

$$\begin{bmatrix} I & 0 \\ -B_2 A^{-1} & I \end{bmatrix} \begin{bmatrix} A & B_1^T \\ B_2 & -C \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} I & 0 \\ -B_2 A^{-1} & I \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \tag{146}$$

or equivalently,

$$\begin{bmatrix} A & B_1^T \\ 0 & S \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 - B_2 A^{-1} y_1 \end{bmatrix}. \tag{147}$$

Algorithm 7 The algorithm of the segregated approach to the solution of a generalized Saddle Point system.

1: **procedure** SADDLEPOINTSEGREGATEDMETHOD(A, B_1, B_2, C, y_1, y_2, x_1, x_2)
2: **Input:** A, B_1, B_2, C, y_1, y_2
3: **Output:** x_1, x_2
4: $w \leftarrow$ **Solve** $Aw = y_1$
5: $v \leftarrow$ **Compute** $v = y_2 - B_2 w$
6: $x_2 \leftarrow$ **Solve** $Sx_2 = v$
7: $z \leftarrow$ **Compute** $z = y_1 - B_1 x_2$
8: $x_1 \leftarrow$ **Solve** $Ax_1 = z$
9: **end procedure**

Then, the solution of the linear system (144) by the segregated method can be performed using Algorithm 7. Solution of linear systems at the lines 4, 6 and 8 of Algorithm 7 can be solved either directly or iteratively. This approach is attractive if the order m of the reduced system at line 6 of Algorithm 7 is small and if linear systems with coefficient matrix A can be solved efficiently. Since the solution of two linear systems with coefficient matrix A is required, it should be convenient to perform just one LU factorization of A to be used twice. The two main drawbacks are that A must be nonsingular and that the Schur complement S can be full, making it prohibitively expensive to factor or compute. When creating S, numerical instability could potentially be an issue, particularly if A is not well conditioned [155]. If S is too costly to form or factor, a Schur complement

reduction can still be used by solving related linear systems using iterative techniques like KSM, which only require S in the form of matrix-vector products that involve the matrices B_1^T, B_2, and C, as well as by solving a linear system with matrix A. These techniques do not require access to individual entries of S. It is possible that the Schur complement system is not well conditioned, in which case preconditioning will be necessary.

Coupled methods Coupled methods deal with the system (144) as a whole, computing x_1 and x_2 (or approximations to them) simultaneously. Among these approaches are direct solvers based on triangular factorizations of the global matrix $\mathcal{A}$, as well as iterative algorithms such as KSM applied to the system as a whole (144), usually preconditioned in some way.

For direct methods based on triangular factorizations of $\mathcal{A}$, we give a brief overview limited to the symmetric case (that is, where $A = A^T$, $B_1 = B_2 = B$ and $C = C^T$), since no specialized direct solver exists for a nonsymmetric Saddle Point problem. Furthermore, we assume that A is positive definite and B has full rank, then the Saddle Point matrix $\mathcal{A}$ admits the following factorization,

$$\mathcal{A} = \mathcal{L}\mathcal{D}\mathcal{L}^T \tag{148}$$

where $\mathcal{D}$ is a diagonal matrix and $\mathcal{L}$ is a unit lower triangular matrix. To be more precise, A is positive definite, so its decomposition is $A = L_A D_A L_A^T$, where L_A is the unit lower triangular and D_A is the diagonal (also positive definite); additionally, the Schur complement $S = -(C + BA^{-1}B^T)$ is negative definite, so its decomposition is $S = -L_S D_S L_S^T$. Thus, we are able to write

$$\mathcal{A} = \begin{bmatrix} A & B^T \\ B & -C \end{bmatrix} = \begin{bmatrix} L_A & 0 \\ L_B & L_S \end{bmatrix} \begin{bmatrix} D_A & 0 \\ 0 & -D_S \end{bmatrix} \begin{bmatrix} L_A^T & L_B^T \\ 0 & L_S^T \end{bmatrix} = \mathcal{L}\mathcal{D}\mathcal{L}^T, \tag{149}$$

where $L_B = BL_A^{-T}D_A^{-1}$. Note that $BA^{-1}B^T = L_B D_A L_B^T$. However, in practice, with the original ordering, the factors will be fairly dense, and sparsity preservation requires the employment of symmetric permutations $\mathcal{Q}$. Not every sparsity-preserving permutation is suitable, though. It is demonstrated that there are permutation matrices $\mathcal{Q}$ such that the factorization of $\mathcal{L}\mathcal{D}\mathcal{L}^T$, where $\mathcal{D}$ is a diagonal matrix, is not available for $\mathcal{Q}\mathcal{A}\mathcal{Q}^T$. Moreover, some permutations might result in issues with numerical instability [155].

Iterative methods are preferred for solving the linear system (144) using coupled methods, as sparse $\mathcal{Q}\mathcal{A}\mathcal{Q}^T$ factorization methods are not completely foolproof, especially for Saddle Point systems arising from PDE problems.

About the usage of iterative algorithms, we wrote some words about the use of preconditioned KSM for iterative solution of the entire linear system (144). Various preconditioning strategies designed for (generalized) Saddle Point systems are explained. To develop high-quality preconditioners for Saddle Point problems, one must take advantage of the problem's block structure and possess a comprehensive understanding of the origin and structure of each block [155].

We consider block diagonal and block triangular preconditioners for KSM applied to the coupled system $\mathcal{A}x = y$ as in (144). The basic block diagonal preconditioner is given by

$$\mathcal{P}_d = \begin{bmatrix} \hat{A} & 0 \\ 0 & -\hat{S} \end{bmatrix} \tag{150}$$

where both $\hat{A}$ and $\hat{s}$ are approximations of A and S, respectively. Several different approximations have been considered in the literature. A fairly general framework considers a splitting of A into

$$A = D - E, \tag{151}$$

where D is invertible and easy to invert. Then $\hat{A} = D$ and $\hat{S} = -\left(C + B_2 D^{-1} B_1^T\right)$ are chosen.

The basic block (upper) triangular preconditioner has the form

$$\mathcal{P}_t^{upper} = \begin{bmatrix} \hat{A} & B_1^T \\ 0 & \hat{S} \end{bmatrix}, \tag{152}$$

and, on the other hand, the form of a (lower) triangular preconditioner is

$$\mathcal{P}_t^{lower} = \begin{bmatrix} \hat{A} & 0 \\ B_2 & \hat{S} \end{bmatrix}, \tag{153}$$

where, as before, both $\hat{A}$ and $\hat{s}$ are approximations of A and S, respectively.

For the coupled solution of a linear system (144) using KSM, the availability of reasonable approximations for the block A and the Schur complement S is necessary for the development of excellent block diagonal and block triangular preconditioners. Such approximations are difficult to build and heavily rely on the specific problem at hand.

For other details about the treatment of Saddle Point problems, we suggest reading the monograph by Benzi et al. [155].

6.1.3. The Discrete Model of The Navier–Stokes Equations for Viscoelastic Fluids

Considering the high flexibility of FEM (see considerations listed in Table 7), we propose the description of FEM discretization of the vscoelastic fluid model.

According to [125], and for the sake of simplicity, we choose to use an Oldroyd-B model for T_P and a Newtonian viscosity model for T_s. So, a viscoelastic fluid's behavior can be described by the following system of equations:

$$\begin{aligned}
\operatorname{div} \mathbf{v} &= 0 && (154)\\
\frac{\partial}{\partial t}(\rho \mathbf{v}) + (\rho \mathbf{v}) \odot \nabla \mathbf{v} &= -\nabla p + \operatorname{div}\left(2\mu_s \boldsymbol{D}\right) + \operatorname{div} \boldsymbol{T}_p + \rho \mathbf{g} + \mathbf{f} && (155)\\
\lambda T_p + \overset{\triangledown}{T}_p &= 2\mu_p \boldsymbol{D}. && (156)\\
\mathbf{v}_{|t=0} &= \mathbf{v}^0 && (157)\\
p_{|t=0} &= p^0 && (158)\\
T_{p_{|t=0}} &= {T_p}^0 && (159)
\end{aligned}$$

The finite element form of the above formulation is often termed the MIX (mixed method) formulation, and it is the basis for many other advanced formulations.

The system of equations (154)–(156) can also be written in Cartesian component form in a Eulerian reference frame $x = (x_1, \ldots, x_n)$ as follows:

Continuity equation

$$\sum_{i=1,\ldots,n} \frac{\partial v_i}{\partial x_i} = 0 \tag{160}$$

Momentum equation

$$\rho\left(\frac{\partial v_i}{\partial t}+\sum_{j=1,\ldots,n} v_j\frac{\partial v_i}{\partial x_j}\right) = -\frac{\partial p}{\partial x_i}+\sum_{j=1,\ldots,n}\frac{\partial}{\partial x_j}(2\mu_s D_{ij})+\sum_{j=1,\ldots,n}\frac{\partial}{\partial x_j}(T_p)_{ij}+h_i \quad (161)$$

Constitutive equation

$$\lambda T_{p_{ij}}+\overset{\triangledown}{T}_{p_{ij}} = 2\mu_p D_{ij}. \quad (162)$$

for all $i=1,\ldots,n$ where $\mathbf{h}=\rho\mathbf{g}+\mathbf{f}$, $D_{ij}=\frac{1}{2}\left(\frac{\partial v_i}{\partial x_j}+\frac{\partial v_j}{\partial x_i}\right)$. The equations in (160)–(162) provide $n+1+n^2$ relations for the unknowns $[v_i(x,t)]_{1,\ldots,n}$, $p(x,t)$, $\left[(T_p)_{ij}(x,t)\right]_{i,j=1,\ldots,n}$.

The application of the finite element procedure to the partial differential equations in (160)–(162) follows the standard format. The velocity components v_i^e, pressure p^e, and extra-stress components $\left(T_p^e\right)_{ij}$ are approximated, in each element e, by expansions of the form

$$v_i^e = \sum_{m=1}^{M}\Psi_m v_i\left({}^e x_m^{\Psi},t\right) = \boldsymbol{\Psi}^{Te}\boldsymbol{v}_i \quad (163)$$

$$p^e = \sum_{n=1}^{N}\Phi_n p\left({}^e x_n^{\Phi},t\right) = \boldsymbol{\Phi}^{Te}\boldsymbol{p} \quad (164)$$

$$T_{ij}^e = \sum_{k=1}^{K}\Pi_k T_{ij}\left({}^e x_k^{\Pi},t\right) = \boldsymbol{\Pi}^{Te}\boldsymbol{T}_{ij} \quad (165)$$

where

- $\boldsymbol{\Psi}$, $\boldsymbol{\Phi}$, and $\boldsymbol{\Pi}$ are vectors of basis functions (the symbol $(\cdot)^T$ denotes the transposition operation);
- M, N, and K indicate the number of nodal points ${}^e x_m^{\Psi}$, ${}^e x_m^{\Phi}$ and ${}^e x_m^{\Pi}$ at which the various unknowns are defined in each element e;
- ${}^e\boldsymbol{v}_i$, ${}^e\boldsymbol{p}$ and ${}^e\boldsymbol{T}_{ij}$ are vectors of unknowns into the nodal points in each element e, i.e.,

$${}^e\boldsymbol{v}_i = \left[v_i\left({}^e x_m^{\Psi},t\right)\right]^T_{m=1,\ldots,M}, \quad (166)$$

$${}^e\boldsymbol{p} = \left[p\left({}^e x_n^{\Phi},t\right)\right]^T_{n=1,\ldots,N}, \quad (167)$$

$${}^e\boldsymbol{T}_{ij} = \left[T_{ij}\left({}^e x_k^{\Pi},t\right)\right]^T_{k=1,\ldots,K}; \quad (168)$$

and where the underscript p for the extra stress has been omitted for clarity.

We recall that

$$\overset{\triangledown}{T}_{ij}=\frac{\partial T_{ij}}{\partial t}+\sum_{k=1,\ldots,n} v_k\frac{\partial T_{ij}}{\partial x_k}-\sum_{k=1,\ldots,n} T_{ik}\frac{\partial v_j}{\partial x_k}-\sum_{k=1,\ldots,n} T_{kj}\frac{\partial v_i}{\partial x_k} \quad (169)$$

Then, by using the finite element approximations (163)–(165) in standard weighted-integral statements (i.e., weak forms) of the system in (160)–(162), the following system of finite element equations can be obtained.

Continuity equation

$$\sum_{i=1,\ldots,n}\left[\int_{\Omega_e}\boldsymbol{\Phi}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_i}dx\right]{}^e\boldsymbol{v}_i=0 \quad (170)$$

Momentum equation

$$\begin{aligned}\left[\int_{\Omega_e}\rho\boldsymbol{\Psi}\boldsymbol{\Psi}^T dx\right]{}^e\dot{\boldsymbol{v}}_i &+\left[\sum_{j=1,\ldots,n}\left[\int_{\Omega_e}\rho\boldsymbol{\Psi}\boldsymbol{\Psi}^T{}^e\boldsymbol{v}_j\frac{\partial\boldsymbol{\Psi}^T}{\partial x_j}dx\right]\right]{}^e\boldsymbol{v}_i\\ &+\left[\int_{\Omega_e}\frac{\partial\boldsymbol{\Psi}}{\partial x_i}\boldsymbol{\Phi}^T dx\right]{}^e\boldsymbol{p}\\ &-\left[\sum_{j=1,\ldots,n}\left[\int_{\Omega_e}\mu_s\frac{\partial\boldsymbol{\Psi}}{\partial x_j}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_j}dx\right]\right]{}^e\boldsymbol{v}_i\\ &-\left[\sum_{j=1,\ldots,n}\left[\int_{\Omega_e}\mu_s\frac{\partial\boldsymbol{\Psi}}{\partial x_j}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_i}dx\right]{}^e\boldsymbol{v}_j\right]\\ &-\left[\sum_{j=1,\ldots,n}\left[\int_{\Omega_e}\frac{\partial\boldsymbol{\Psi}}{\partial x_j}\boldsymbol{\Pi}^T dx\right]{}^e\boldsymbol{T}_{ij}\right]\\ &=\int_{\Omega_e}\boldsymbol{\Psi}{}^e\boldsymbol{h}_i dx\end{aligned}\tag{171}$$

Constitutive equation

$$\begin{aligned}\left[\int_{\Omega_e}\boldsymbol{\Pi}\boldsymbol{\Pi}^T dx\right]{}^e\dot{\boldsymbol{T}}_{ij} &+\left[\int_{\Omega_e}\lambda\boldsymbol{\Pi}\boldsymbol{\Pi}^T dx\right]{}^e\boldsymbol{T}_{ij}\\ &-\left[\int_{\Omega_e}\mu_p\boldsymbol{\Pi}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_j}dx\right]{}^e\boldsymbol{v}_i-\left[\int_{\Omega_e}\mu_p\boldsymbol{\Pi}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_i}dx\right]{}^e\boldsymbol{v}_j\\ &+\left[\sum_{k=1}^{n}\left[\int_{\Omega_e}\boldsymbol{\Psi}^T{}^e\boldsymbol{v}_k\frac{\partial\boldsymbol{\Pi}^T}{\partial x_k}dx\right]\right]{}^e\boldsymbol{T}_{ij}\\ &-\left[\sum_{k=1}^{n}\left[\int_{\Omega_e}\boldsymbol{\Pi}^T{}^e\boldsymbol{T}_{ik}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_k}dx\right]\right]{}^e\boldsymbol{v}_j\\ &-\left[\sum_{k=1}^{n}\left[\int_{\Omega_e}\boldsymbol{\Pi}^T{}^e\boldsymbol{T}_{kj}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_k}dx\right]\right]{}^e\boldsymbol{v}_i\end{aligned}\tag{172}$$

where ${}^e\mathbf{n}=\left[{}^e n_i\right]_{i=1,\ldots,n}$ is the vector normal to boundary $\partial\Omega_e$.

The algebraic relations system (170)–(172), in each element e, can be written in nonlinear matrix form as follows:

$$\begin{bmatrix}{}^e\boldsymbol{M}_{\Psi\Psi} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & {}^e\boldsymbol{M}_{\Pi\Pi\Pi}\end{bmatrix}\begin{bmatrix}{}^e\dot{\boldsymbol{V}}\\ {}^e\dot{\boldsymbol{p}}\\ {}^e\dot{\boldsymbol{T}}\end{bmatrix}+\begin{bmatrix}{}^e\boldsymbol{K}_{\Psi\Psi}({}^e\boldsymbol{V}) & {}^e\boldsymbol{K}_{\Psi\Phi} & {}^e\boldsymbol{K}_{\Psi\Pi}\\ {}^e\boldsymbol{K}_{\Phi\Psi} & \mathbf{0} & \mathbf{0}\\ {}^e\boldsymbol{K}_{\Pi\Psi} & \mathbf{0} & {}^e\boldsymbol{K}_{\Pi\Pi\Pi}+{}^e\boldsymbol{K}_{\Psi\Pi}({}^e\boldsymbol{V})\end{bmatrix}\begin{bmatrix}{}^e\boldsymbol{V}\\ {}^e\boldsymbol{p}\\ {}^e\boldsymbol{T}\end{bmatrix}=\begin{bmatrix}{}^e\boldsymbol{H}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}\tag{173}$$

where

$${}^e\boldsymbol{M}_{\Psi\Psi}=\boldsymbol{I}\otimes{}^e\boldsymbol{M}_{\Psi},\qquad {}^e\boldsymbol{M}_{\Pi\Pi\Pi}=\boldsymbol{I}\otimes(\boldsymbol{I}\otimes{}^e\boldsymbol{M}_{\Pi}),\qquad {}^e\boldsymbol{K}_{\Phi\Psi}=\left[{}^e\boldsymbol{K}^1_{\Phi\Psi}\ \cdots\ {}^e\boldsymbol{K}^n_{\Phi\Psi}\right],$$

$${}^e\boldsymbol{K}_{\Psi\Phi}=\left[{}^e\boldsymbol{K}^1_{\Psi\Phi}\ \cdots\ {}^e\boldsymbol{K}^n_{\Psi\Phi}\right]^T,\qquad {}^e\boldsymbol{K}_{\Pi\Pi\Pi}=\boldsymbol{I}\otimes(\lambda\boldsymbol{I}\otimes{}^e\boldsymbol{M}_{\Pi}),$$

$${}^e\boldsymbol{K}_{\Psi\Pi}=\boldsymbol{I}\otimes\left[{}^e\boldsymbol{K}^1_{\Psi\Pi}\ \cdots\ {}^e\boldsymbol{K}^n_{\Psi\Pi}\right],\qquad {}^e\boldsymbol{K}^i_{\Phi\Psi}=\left[\int_{\Omega_e}\boldsymbol{\Phi}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_i}dx\right],$$

$${}^e\boldsymbol{T}=\left[{}^e\boldsymbol{T}^T_{11}\ \cdots\ {}^e\boldsymbol{T}^T_{1n}\cdots{}^e\boldsymbol{T}^T_{n1}\cdots{}^e\boldsymbol{T}^T_{nn}\right]^T,\qquad {}^e\boldsymbol{V}=\left[{}^e\boldsymbol{v}^T_1\ \cdots\ {}^e\boldsymbol{v}^T_n\right]^T,$$

$${}^e\bar{\boldsymbol{K}}_{\Psi\Psi}=-\left[\sum_{j=1,\ldots,n}\left[\int_{\Omega_e}\mu_s\frac{\partial\boldsymbol{\Psi}}{\partial x_j}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_j}dx\right]\right],\qquad {}^e\boldsymbol{M}_{\Pi}=\left[\int_{\Omega_e}\boldsymbol{\Pi}\boldsymbol{\Pi}^T dx\right],$$

$${}^e\boldsymbol{K}^{i*}_{\Pi\Psi}(T)=\left[\sum_{k=1}^{n}\left[\int_{\Omega_e}\boldsymbol{\Pi}^T{}^e\boldsymbol{T}_{ik}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_k}dx\right]\right],\qquad {}^e\boldsymbol{K}^j_{\Psi\Pi}=-\left[\int_{\Omega_e}\frac{\partial\boldsymbol{\Psi}}{\partial x_j}\boldsymbol{\Pi}^T dx\right],$$

$${}^e\boldsymbol{K}_{\Psi\Pi}({}^e\boldsymbol{V})=\boldsymbol{I}\otimes\left(\boldsymbol{I}\otimes\left[\sum_{k=1}^{n}\left[\int_{\Omega_e}\boldsymbol{\Psi}^T{}^e\boldsymbol{v}_k\frac{\partial\boldsymbol{\Pi}^T}{\partial x_k}dx\right]\right]\right),\qquad {}^e\boldsymbol{K}^j_{\Pi\Psi}=\left[\int_{\Omega_e}\mu_p\boldsymbol{\Pi}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_i}dx\right],$$

$${}^e\boldsymbol{K}^{*j}_{\Pi\Psi}(T)=\left[\sum_{k=1}^{n}\left[\int_{\Omega_e}\boldsymbol{\Pi}^T{}^e\boldsymbol{T}_{kj}\frac{\partial\boldsymbol{\Psi}^T}{\partial x_k}dx\right]\right],\qquad {}^e\boldsymbol{K}^i_{\Psi\Phi}=\left[\int_{\Omega_e}\frac{\partial\boldsymbol{\Psi}}{\partial x_i}\boldsymbol{\Phi}^T dx\right],$$

$$
{}^{e}\bar{\boldsymbol{K}}^{i}_{\Psi\Psi}({}^{e}\boldsymbol{V}) = \left[\sum_{j=1,\ldots,n}\left[\int_{\Omega_e}\rho\boldsymbol{\Psi}\boldsymbol{\Psi}^{T\,e}v_j\frac{\partial\boldsymbol{\Psi}^{T}}{\partial x_j}dx\right]\right], \qquad {}^{e}\boldsymbol{M}_{\Psi} = \left[\int_{\Omega_e}\rho\boldsymbol{\Psi}\boldsymbol{\Psi}^{T}dx\right],
$$

$$
\begin{aligned}
{}^{e}\boldsymbol{H} &= \left[\left(\int_{\Omega_e}\boldsymbol{\Psi}^{T\,e}\boldsymbol{h}_1dx\right)\cdots\left(\int_{\Omega_e}\boldsymbol{\Psi}^{T\,e}\boldsymbol{h}_ndx\right)\right]^{T},\\
{}^{e}\boldsymbol{K}_{\Pi\Psi} &= -\left(\boldsymbol{I}\otimes diag\left({}^{e}\boldsymbol{K}^{j}_{\Pi\Psi}\right)_{j=1,\ldots,n}\right)-\left(diag\left({}^{e}\boldsymbol{K}^{i}_{\Pi\Psi}\right)_{i=1,\ldots,n}\otimes I\right)\\
&\quad -\left(\boldsymbol{I}\otimes diag\left({}^{e}\boldsymbol{K}^{*j}_{\Pi\Psi}(T)\right)_{j=1,\ldots,n}\right)-\left(diag\left({}^{e}\boldsymbol{K}^{i*}_{\Pi\Psi}(T)\right)_{i=1,\ldots,n}\otimes I\right),\\
{}^{e}\boldsymbol{K}_{\Psi\Psi}({}^{e}\boldsymbol{V}) &= \left(diag\left({}^{e}\bar{\boldsymbol{K}}^{i}_{\Psi\Psi}({}^{e}\boldsymbol{V})\right)_{i=1,\ldots,n}+(\boldsymbol{I}\otimes{}^{e}\bar{\boldsymbol{K}}_{\Psi\Psi})\right)+{}^{e}\bar{\bar{\boldsymbol{K}}}_{\Psi\Psi},\\
{}^{e}\bar{\bar{\boldsymbol{K}}}_{\Psi\Psi} &= -\begin{bmatrix}\left[\int_{\Omega_e}\mu_s\frac{\partial\boldsymbol{\Psi}}{\partial x_1}\frac{\partial\boldsymbol{\Psi}^{T}}{\partial x_1}dx\right] & \cdots & \left[\int_{\Omega_e}\mu_s\frac{\partial\boldsymbol{\Psi}}{\partial x_n}\frac{\partial\boldsymbol{\Psi}^{T}}{\partial x_1}dx\right]\\ \vdots & \ddots & \vdots\\ \left[\int_{\Omega_e}\mu_s\frac{\partial\boldsymbol{\Psi}}{\partial x_1}\frac{\partial\boldsymbol{\Psi}^{T}}{\partial x_n}dx\right] & \cdots & \left[\int_{\Omega_e}\mu_s\frac{\partial\boldsymbol{\Psi}}{\partial x_n}\frac{\partial\boldsymbol{\Psi}^{T}}{\partial x_n}dx\right]\end{bmatrix},
\end{aligned}
$$

and where $\boldsymbol{I}$ is the identity matrix of dimension $n\times n$. The operator $\otimes$ represents the Kronecker products of two matrices: if $A=\left(a_{ij}\right)_{i,j=1,\ldots,n}$ and $B=\left(b_{ij}\right)_{i,j=1,\ldots,m}$ are respectively matrices of dimensions $n\times n$ and $m\times m$, $A\otimes B$ is a matrix of dimensions $mn\times mn$, where

$$
A\otimes B=\begin{bmatrix}a_{11}B & \ldots & a_{1n}B\\ \vdots & \ldots & \vdots\\ a_{n1}B & \ldots & a_{nn}B\end{bmatrix}.
$$

The operator $\mathrm{diag}(B_i)_{i=1,\ldots,n}$ represents the matrix of dimensions $mn\times mn$

$$
diag(B_i)_{i=1,\ldots,n}=\begin{bmatrix}B_1 & \ldots & 0\\ \vdots & \ldots & \vdots\\ 0 & \ldots & B_n\end{bmatrix}.
$$

where B_i is a matrix of dimension $m\times m$ for all $i=1,\ldots,n$. .

The assembly process of all the elements of the finite element mesh generates an algebraic system of the type in Equation (112) where the matrices and vectors have the structure described in Equation (173), that is,

$$
{}^{NSVF}\boldsymbol{M}\,{}^{NSVF}\dot{\boldsymbol{u}}+{}^{NSVF}\boldsymbol{K}\left({}^{NSVF}\boldsymbol{u}\right)\boldsymbol{u}={}^{NSVF}\boldsymbol{f}, \tag{174}
$$

where

$$
{}^{NSVF}\boldsymbol{M} = \begin{bmatrix}\boldsymbol{M}_V & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{M}_T\end{bmatrix}, \tag{175}
$$

$$
{}^{NSVF}\boldsymbol{K}(\boldsymbol{u}) = \begin{bmatrix}\boldsymbol{K}_{V,V}(\boldsymbol{u}) & \boldsymbol{K}_{V,p} & \boldsymbol{K}_{V,T}\\ \boldsymbol{K}_{p,V} & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{K}_{T,V} & \boldsymbol{0} & \boldsymbol{K}_{T,T}(\boldsymbol{u})\end{bmatrix}, \tag{176}
$$

$$
{}^{NSVF}\boldsymbol{u}=\begin{bmatrix}\boldsymbol{V}\\ \boldsymbol{p}\\ \boldsymbol{T}\end{bmatrix}, \quad {}^{NSVF}\boldsymbol{f}=\begin{bmatrix}\boldsymbol{H}\\ \boldsymbol{0}\\ \boldsymbol{0}\end{bmatrix}. \tag{177}
$$

During a Time Integration Scheme, the solver of a nonlinear problem always requires the solution to linear systems (see the line 6 of Algorithm 3) whose coefficient matrix $\boldsymbol{A}$

(see Equation (115)) has the form of a Saddle Point problem. Indeed, the block structure of matrix $\boldsymbol{A}$ can be reassembled as

$$\boldsymbol{A} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{M}_V + a_1 \boldsymbol{K}_{V,V}(\boldsymbol{u}_0) & \boldsymbol{K}_{V,p} \\ \boldsymbol{K}_{p,V} & \boldsymbol{0} \end{bmatrix} & \begin{bmatrix} \boldsymbol{K}_{V,T} \\ \boldsymbol{0} \end{bmatrix} \\ \begin{bmatrix} \boldsymbol{K}_{T,V} & \boldsymbol{0} \end{bmatrix} & \boldsymbol{M}_T + a_1 \boldsymbol{K}_{T,T}(\boldsymbol{u}_0) \end{bmatrix}, \tag{178}$$

or equivalently

$$\boldsymbol{A} = \begin{bmatrix} {}^{NSVF}\boldsymbol{K}_{11} & {}^{NSVF}\boldsymbol{K}_{12} \\ {}^{NSVF}\boldsymbol{K}_{21} & {}^{NSVF}\boldsymbol{K}_{22} \end{bmatrix}, \tag{179}$$

where the matrix ${}^{NSVF}\boldsymbol{K}_{11}$ is, in turn, a Saddle Point problem where its upper-left block $\boldsymbol{M}_V + a_1 \boldsymbol{K}_{V,V}(\boldsymbol{u}_0)$ is a symmetric matrix, where its left-bottom and upper-right blocks are each the transpose of the other (i.e., $\boldsymbol{K}_{p,V} = \boldsymbol{K}_{V,p}^T$), and where its bottom-right one is a zero block. Such a system could then be solved by an opportune combination of the algorithms described in Section 6.1.2 on the basis of numerical and structural properties of matrices ${}^{NSVF}\boldsymbol{M}$ and ${}^{NSVF}\boldsymbol{K}(\boldsymbol{u})$. The number of system unknowns is $N_{nodes} \times \left(n + n^2 + 1\right)$, but, in consideration of the symmetric nature of the matrix $\boldsymbol{T}_p$, just $\frac{n(n+1)}{2}$ elements of $\boldsymbol{T}_p$ have to be computed, so the real number of system unknowns should be $N_{nodes} \times \left[(n+1) + \frac{n(n+1)}{2}\right] = \frac{N_{nodes}}{2}(n+1)(n+2)$.

We note that the solution of Problems 2 and 3 related to Two-Phase Fluids requires the solution of the equations describing Problem 1 coupled with other equations describing, respectively, interface motion (see Equation (22)) and Immersed Body motion (see Equation (44)).

The discretization of such equations adds row and column blocks, related, respectively, with values chosen to approximate ϕ and X on the discretization mesh, to the algebraic system in Equation (174). The resulting system, which preserves its block sparse structure, can then be solved using algorithms described in Subsection 6.1.2.

The same considerations could be made about the equations (63)–(64) and (65) if they are coupled with the equations describing Problem 1 (or Problems 2 and 3).

6.2. Models at the Meso Scale

6.2.1. The Markov Chain Monte Carlo Algorithms

Let Ω be a finite configuration space of a physical system and assume that a suitable proposal transition method $\mathcal{M}(\omega)$ from the state $\omega \in \Omega$ to the state $\omega^* \in \Omega$ has been selected: in general, $\mathcal{M}$ is defined by a transition probabilities $P(\omega^*|\omega)$. Let π be a given distribution for which a function $g : \Omega \leftarrow \Omega$ exists such that

$$\frac{g(\omega)}{g(\omega^*)} = \frac{\pi(\omega)}{\pi(\omega^*)}.$$

If ω_0 is the initial state, the Metropolis–Hastings algorithm generates the set of M states $\{\omega_i\}_{i=1,\ldots,M} \subset \Omega$ that asymptotically reaches the stationary π distribution, and it is depicted in Algorithm 8.

Algorithm 8 The MCMC Metropolis–Hastings algorithm.

procedure MCMCMETROPOLISHASTINGS(Ω, ω_0, M)
Input: Ω, ω_0, M
Output: $\{\omega_i\}_{i=1,\dots,M}$
  for $n = 1$ to M **do**
    $\omega^* \leftarrow$ **Select** $\omega^* = \mathcal{M}(\omega_{i-1})$
    $\xi \leftarrow$ **Generate a uniform random number** $\xi \in [0,1]$
    $\Xi \leftarrow$ **Compute** $\Xi = \min\left(1, \frac{g(\omega^*)}{g(\omega_{i-1})}\frac{P(\omega_{i-1}|\omega^*)}{P(\omega^*|\omega_{i-1})}\right)$ ▷ Acceptance probability Ξ
    if $\xi \leq \Xi$ **then** ▷ Allow the new state
      $\omega_i \leftarrow \omega^*$
    else ▷ Reject the inew state
      $\omega_i \leftarrow \omega_{i-1}$
    end if
  end for
end procedure

6.2.2. The Cellular Particle Dynamics Algorithms

Let $\mathcal{C}$ ne a finite set of N_{Cells} cells, and assume that each cell is composed of $N_{CellularElement}$ elements. Let $\boldsymbol{y}_{\alpha_i} = \left(y^1_{\alpha_i}, \dots, y^N_{\alpha_i}\right) \in \Re^N$, the position vectors of each element α_i where $\alpha = 1, \dots, N_{CellularElement}$ and $i = 1, \dots, N_{Cells}$.

For the generic element α_i, Equation (80) can be rewritten as

$$\mu\frac{\partial \boldsymbol{y}_{\alpha_i}}{\partial t} = \boldsymbol{\eta}_{\alpha_i} - \sum_{\beta_i \neq \alpha_i} \nabla_{\boldsymbol{y}_{\alpha_i}}\left[V_{intra}\left(\left|\boldsymbol{y}_{\alpha_i} - \boldsymbol{y}_{\beta_i}\right|\right)\right] - \sum_{j\neq i}\sum_{\beta_j} \nabla_{\boldsymbol{y}_{\alpha_i}}\left[V_{inter}\left(\left|\boldsymbol{y}_{\alpha_i} - \boldsymbol{y}_{\beta_j}\right|\right)\right] \tag{180}$$

Thanks to the properties of the gradient ∇_y of a composite real function $f(g(y))$, we have

$$\nabla_y f(g(y)) = \frac{df}{dg}(g)\nabla_y g(y).$$

Then, for each component n and for each element α_i, Equation (180) can rewritten as

$$\mu\frac{\partial y^n_{\alpha_i}}{\partial t} = \eta^n_{\alpha_i} - \sum_{\beta_i \neq \alpha_i}\left[\frac{y^n_{\alpha_i} - y^n_{\beta_i}}{d_{\alpha_i,\beta_i}}\frac{d\, V_{intra}}{d\, d_{\alpha_i,\beta_i}}\left(d_{\alpha_i,\beta_i}\right)\right] - \sum_{j\neq i}\sum_{\beta_j}\left[\frac{y^n_{\alpha_i} - y^n_{\beta_j}}{d_{\alpha_i,\beta_j}}\frac{d\, V_{intra}}{d\, d_{\alpha_i,\beta_j}}\left(d_{\alpha_i,\beta_j}\right)\right] \tag{181}$$

The set of all $N \times N_{CellularElement} \times N_{Cells}$ equations of type (181) represents a generally non-linear system of algebraic equations in the form represented by Equation (112) where the vector of unknowns $\boldsymbol{y}$ is given by

$$\boldsymbol{y} = \left[y^n_{\alpha_i}\right]_{\substack{\alpha=1,\dots,N_{CellularElement}\\ i=1,\dots,N_{Cells}\\ n=1,\dots,N.}}$$

This system could then be solved by an opportune combination of the algorithms described in Section 6.1.2 on the basis of numerical and structural properties of the matrices involved.

6.2.3. The Cellular Automata Model Algorithms

Let $\mathcal{A}$ bea Cellular Automata defined by

1. A N-dimensional grid $\mathcal{G}$ of $N_{Cells} = \prod_{n=1}^{N} N^n_{Cells}$ cells $\mathcal{C} = \{\mathfrak{c}^n\}_{n=1,\dots,N_{Cells}}$ where, for each of them, the set of neighborhood $\mathcal{N}^n$ is defined;
2. A set $\mathcal{I}$ of $N_{ingredients}$ ingredients, i.e., $\mathcal{I} = \left\{\mathfrak{i}^l\right\}_{l=1,\dots,N_{ingredients}}$;
3. A set $\mathcal{R}$ of N_{rules} rules, i.e., $\mathcal{R} = \{\mathfrak{r}^n(\mathcal{I})\}_{n=1,\dots,N_{rules}}$.

Given the number M of the iteration defining the length of each run and the number N_{run} defining the number of the runs to be executed, the $M \times N_{run}$ states $\{\mathcal{G}_{i,j}\}_{i=1,\ldots,M;j=1,\ldots,N_{run}}$ of asynchronous simulation based on the automata $\mathcal{A}$ can be computed by Algorithm 9.

Algorithm 9 The asynchronous cellular automata algorithm.

procedure ASYNCHRONOUSCELLULARAUTOMATA($\mathcal{A}$, $\mathcal{G}_0$, N, M)
Input: $\mathcal{A}$, $\mathcal{G}_0$, N, M
Output: $\{\mathcal{G}_{i,j}\}_{i=1,\ldots,M;j=1,\ldots,N_{run}}$
  for $j = 1$ to N_{run} **do**
    Generate an initial condition state $\mathcal{G}_{0,j}$ for the grid $\mathcal{G}$;
    for $i = 1$ to M **do**
      for $n = 1$ to N_{Cells} **do**
        if (cell $\mathfrak{c}^n_{i-1,j}$ of the grid $\mathcal{G}_{i-1,j}$ is occupaied by some ingredient $\mathfrak{i}^l \in \mathcal{I}$) **then**
          for $m = 1$ to N_{rules} **do**
            Apply rule $\mathfrak{r}^m(\mathcal{I})$ to $\mathfrak{c}^n_{i-1,j}$ to **define** $\mathfrak{c}^n_{i,j}$, and its neighborhood $\mathcal{N}^n_{i,j}$, on grid $\mathcal{G}_{i,j}$
          end for
        end if
      end for
    end for
  end for
end procedure

7. Algorithms for HPC Systems in the Exascale Era

The extremes of temporal and spatial scales, as well as the intricate nonlinear interactions of several physical processes, are two recurring themes in computer science. Although it is still beyond the capabilities of present petascale systems, high-fidelity simulation of this kind of physical system may become affordable on approaching Exascale computer resources. Exploiting the capabilities of impending Exascale architectures will need significant new model creation, algorithm reform, and reimplementation of research application code. A few of these problems have already been documented [156–158].

If solvers are a mainstay of scientific computing and justify a great investment in CS research for Exascale resources, the discretization and mathematical formulation of the problem are also critical stages in the simulation process that provide new difficulties for both linear and nonlinear solvers.

In computational materials sciences, Exascale computing is needed to predict and understand the behavior of new and heterogeneous materials at disparate lengths and time scales. It is worth noting that to describe such systems, characterized by a significant structural disorder and chemical complexity, a multiscale approach should be pursued. To this aim, creating links between continuum and microscale models is essential for bridging different length scales. Providing precise up-scaling interaction for coarse-grained approaches and down-scaling interactions to treat nanoscale and electronic environments is how this is achieved [156]. For example, linking a microscopic model to an atomistic or continuum model using a microscopic simulation to fit or estimate parameters contained in a higher-level model is among the CS challenges. To this aim, a system of closely coupled nonlinear differential/integral equations should be solved. Additionally, gathering data from several parallel instances of microscopic simulations might be necessary. For solving both coupling equations and nonlinear equations employed in a microscopic model, it is extremely desirable to apply iterative solver acceleration approaches that can leverage a physics-motivated preconditioner. Time-evolution systems that are both stable and efficient are required to explain the material's dynamic behavior [156].

7.1. Models for Exascale Computing

As described in Section 5, mathematical models often involve coupled physical phenomena being inherently multiphysics-based. In addition, there are many potential levels of description from the atomic up to macroscopic scales.

Although it is severely constrained by the computer resources at hand, the degree of fine-grained physical accuracy in models is contingent upon the significance of the fine-scale processes' characteristics on the macroscopic time and length scales. In theory, classical molecular models may try to perform simulations at this size. Even with Exascale resources, such a model has far too many unknowns to compute at macroscopic (engineering) scales. To this aim, the approach is to divide and conquer the scales by creating simplified (or coarse-grained) models able to correctly describe a particular set of dynamics. In such a way, it can also be possible to control the complexity and scale disparity of first-principles-based models. Naturally, using this method produces multiscale or scale-bridging models [156,159–161] in which a coupled hierarchy of models is considered.

These models avoid using a brute-force approach that would lead to an infeasible solution by making many attempts to bridge disparate time and length scales. By definition, these scale-bridging algorithms employ the split of scales to generate optimal formulations at different levels of issue description. This clearly leads to a hierarchical or layered issue description, which is useful in conjunction with the expected future Exascale computer systems' hierarchical nature [156] (see Section 4).

Two common mathematical representations are utilized, as explained in Section 5: (1) continuum models, which use a Eulerian approach to numerically represent functions on a continuous domain by means of a finite-dimensional representation linked to a mesh, and (2) particle models, which follow the dynamics of individual or groups of discrete entities in space in a Lagrangian fashion. The particle models can explain a wider range of (nonequilibrium) behavior, but they are noisy and can become costly at macroscopic scales. Despite their lack of noise and better-understood convergence properties, continuum models may become too expensive to use at small scales. Particle-based models (see Section 5.2 for examples of such approaches) are suitable for Exascale computing applications in some cases.

It is possible to process particles individually until they synchronize at a later time-step level. This allows for arbitrary fine-grained parallelism and the exploitation of asynchrony. Particle models can efficiently use single precision due to their statistical character, where the accuracy impact of a lower precision is significantly smaller than the associated statistical noise. It is also easier to manage particles' fault tolerance due to their statistical character.

Exascale computing generally offers a chance to reconsider and enhance the way that models in multiphysics/multiscale models should be integrated across scales and physical processes. An effective substitute for operator splitting has been made possible by the development of nonlinear implicit solvers. The shift toward more closely coupled techniques presents a great possibility for adaptation to the Exascale, with its memory bandwidth stringencies, despite some unsolved mathematical issues [156,162]

This article makes multiple references to the phrases *"coupled"* (also known as *"splitted"* or *"segregated"*), and since coupling exists at least on an algorithmic and physical level, some definitions are needed. According to Keyes et al. [162], *"the terms "strong (versus weak) coupling" of physical models mean that physical components of the modeled system are coupled by a strong (versus weak) interaction. Correspondingly, the term "tight (versus loose) coupling of numerical models means that algorithmic components are coupled by a tight (versus loose) interaction"*.

In the management of multiscale multiphysics problems, the strategy described in Keyes et al. can therefore be considered appropriate

> *"Because we strongly advocate examining coupling strength before pursuing a decoupled or split strategy for solving a multiphysics problem, we propose "coupled until proven decoupled" as a perspective worthy of 21st-century simulation purposes and resources."* [162]

Description of multiphysics modeling, associated implementation issues such as coupling or decoupling approaches, and example applications are available in Keyes et al. [162].

A contemporary multiscale technique, the multiscale high-order/low-order (HOLO) strategy [161], is appropriate for issues where a microscopic description must be connected to a macroscopic one. The resulting technique will be referred to as HOLO if the macroscopic system is called the low-order (LO) description and the microscopic one the high-order (HO) description. Descriptions of some applications of the HOLO method are available in Chacon et al. [161].

7.2. Parallel Solvers for Exascale Computing

To create and implement effective solvers on Exascale systems, it will be necessary to overcome many obstacles. The architectural features of these computers, which include additional memory levels, more concurrency with less memory per core, and heterogeneous cores, will affect the performance of all the proposed algorithms. As explained in [157], in fact, the following considerations should guide the development and use of new Exascale solvers.

- Extreme-scale computers might consist of millions of nodes with thousands of light cores or hundreds of thousands of nodes with more aggressive cores, according to technological advancements. In any case, solvers will need to enable billion-way concurrence with small thread states [157].
- Extreme-scale systems will probably encounter a high number of hardware malfunctions since they will be constructed on a vast number of components, highlighting a resilient and non-deterministic behavior. Applications should therefore anticipate a high frequency of soft mistakes (such as data value changes due to logic latch faults) and hard interruptions (device failure). Current fault-tolerance methods (those based on traditional, full checkpoint/restart) may cause significant overhead and be impractical for these systems. Solvers must therefore be equipped with methods for locating and resolving software errors and interruptions to reduce any detrimental consequences on an application [157].
- Compared to memory technologies, the cost of devices designed for floating-point operations is decreasing more quickly. Recent system trends indicate that the memory per node is already modestly increasing while memory per core is dropping. Therefore, to take advantage of parallelism, solvers will need to reduce synchronization, increase computation on local data, and shift their focus from the typical scenario of weak scaling, which involves using more resources to solve problems of larger sizes, to one that favors strong scaling, which involves using more resources to solve a problem of fixed size in order to reduce the time to solution (see [163] for a definition of weak vs. strong scaling) [157].
- Power consumption will be a major issue for future HPC systems. Although low-energy accelerators and other hardware design elements can lower the needed power, more steps must be taken to further minimize energy consumption. An approach that could be used is algorithmic study [157].

Several methods and procedures [156,157] have been proposed in response to these factors to help achieve algorithms that make use of the opportunities provided by Exascale computing platforms. A summary of some of these strategies and techniques is provided below.

Communication- and synchronization-avoiding algorithms The expected costs of computation at the Exascale (also in terms of energy consumption) will be more conditioned by movements from/to the memory or between different memories (i.e., *"a communication"*) than by the execution of floating point operations. So, when solving huge problems on parallel architectures with extreme levels of concurrency, the most significant concern becomes the cost related to communication and synchronization overheads. This is especially the case for preconditioned Krylov methods

(see Section 6.1.2). New versions of such algorithms *"minimizing communication"* should be investigated to address the critical issue of such communication costs (see Section 7.2.3). Likewise, such popular algorithms execute tasks requiring synchronization points, for example, the computation of BLAS operators and the preconditioner applications. Then, additive algorithmic variants (with no synchronization among components) may be more attractive compared to their multiplicative counterparts [164]. Such considerations are taken into account also by *"Time integrators"* in the Exascale context (see Section 7.2.1). Furthermore, the linear solution process, which is at the base of the solution of almost all described problems in the previous sections, requires constantly moving the sparse matrix entries from the main memory into the CPU, without register or cache reuse. Therefore, methods that demand less memory bandwidth need to be researched. There are promising linear techniques, such as coefficient-matrix-free representations [165], that totally avoid sparse matrices, or Fast Multipole Methods (see Section 7.2.4) [157].

Mixed-Precision Arithmetic Algorithms Using a combination of single- and double-precision arithmetic in a calculation is known as mixed-precision arithmetic. Single-precision arithmetic is typically faster than double-precision arithmetic, but some computations may be so susceptible to roundoff errors that the entire computation is typically not possible to complete with just a single precision, so mixed-precision arithmetic can be used. In order to quantify the situations in which using mixed-precision arithmetic is advantageous for a given class of algorithms, further research is required, with a particular focus on the numerical behavior of this technique. In addition to speeding up problem-solving, mixed-precision arithmetic might also need less memory. When working with mixed-precision arithmetic, it is crucial to understand how round-off errors spread and affect computed value accuracy [157]. Section 7.2.3 provides examples of mixed-precision techniques for solving linear problems.

Fault-tolerant and resilient algorithms Allowing for nondeterminism in both data and operations is a beneficial—and perhaps essential—feature that makes fault-tolerant and robust algorithms possible. Algorithms based on pseudorandom number generators offer a multitude of possibilities at the extreme scale. Monte Carlo-like methods, or more in general, stochastic-based algorithms such as Cellular Automata, can identify resilient implementations by using stochastic replications, which can improve concurrency by several orders of magnitude by employing stochastic replications to find robust implementations [157] (see Section 7.2.5). Regarding fault-tolerant algorithms, a possible solution to the problem of global synchronous checkpoint/restart on extreme-scale systems is the use of localized checkpoints and asynchronous recovery. Recent work is focused on algorithm-based fault tolerance (ABFT): for example, much of the ABFT work on linear solvers is based on the use of checksum [157,166].

7.2.1. *"Implicit-Explicit"* and *"Parallel in Time"* Solvers

Different problem components frequently develop on various temporal scales. Thus, in real-world applications, it is frequently not possible to integrate PDE discretizations in time efficiently using a single time-stepping technique. By utilizing the benefits of several strategies and mitigating the drawbacks of employing a single set strategy, partitioned methods get around these challenges. A class of partitioned time integration methods called implicit–explicit (IMEX) methods, commonly referred to as semi-implicit schemes, expand explicit and implicit schemes that accommodate variable redundant calculations in favor of more scalable algorithms.

Proposed IMEX methods are based on an approach that uses extrapolation methods [167] for the solution of ODE [168,169]. Extrapolation techniques are a subset of numerical analysis that are associated with convergence acceleration techniques, the application of which has been researched and used in numerous contexts for many years. Sequences and series are frequently dealt with in numerical analysis, applied mathematics, and engineering. They are created using approximation techniques based on parameters,

iterative processes, and so on. In actuality, certain series or sequences frequently converge so slowly that it seriously hinders their usefulness. In such a context, extrapolation methods can play a role. In the specific context of the ODE solution, extrapolation methods can help to improve the rate of convergence of a *"time integrator"* method.

Definition 2. *The extrapolation method for ordinary differential equations:*
The aim is to compute an approximated solution to the the following ODE problem

$$\dot{u}(t) = F(t, u(t)) \tag{182}$$

where $u(t_0) = u_0$, $u(t) \in \Re^d$ *and where* $t \in \Theta = [t_0\ t_n]$. *As described in Section 6.1.2 about the Time Integration Algorithms, to compute an approximation of the solution in the considered time domain* Θ, *a discretization* Θ_{N_n} *of* Θ *should be considered. For this purpose, consider* Θ_{N_n} *as the set made up of the* $N_n + 1$ *equally spaced points* $\{t_k\}_{k=0,\ldots,N_n}$ *where* $t_k = t_0 + \Delta_n * k, k = 1, \ldots,\ N_n$ *and where* $\Delta_n = \frac{t_n}{N_n}$.

The approximation u_{Δ_n} *in* Θ_{N_n} *is constructed recursively with the operator* $\Psi_{\Delta_n}^{k,k-1}$ *(a so-called "one step method") with step size* Δ_n *and order of accuracy* p. *The rate at which a numerical approximation of a differential equation converges to the actual solution is measured in numerical analysis by the order of accuracy. Consider a differential equation's exact solution,* u, *and its numerical approximation,* u_h, *where* h *is a parameter that describes the approximation, and the numerical solution* u_h *is said to be pth-order accurate if the error* $E(h) := \|u - u_h\|$ *is limited above by a value proportional to* h *to the pth power:*

$$\begin{aligned} E(h) := \|u - u_h\| &\leq Ch^p \\ u_{\Delta_n}(t_0) &= u_0, \qquad (183) \\ u_{\Delta_n}(t_k) &= \Psi_{\Delta_n}^{k,k-1} u_{\Delta_n}(t_{k-1}). \qquad (184) \end{aligned}$$

Let $\{n_j\}_{j=1,\ldots,M}$ *be a set of positive integers representing a sequence such that* $n < n_j < n_{j+1}$, *and let*

$$T_{l,j} = \prod_{k=1}^{l} \Psi_{\Delta_{n_j}}^{k,k-1} u_{\Delta_{n_j}}(t_0),\ 0 < l \leq n_j. \tag{185}$$

Given the M *couples* $\left(\Delta_{n_j}, T_{n_j,j}\right)$ *and an integer value* $\omega \geq 1$, *there exists a unique polynomial* $\pi \in \Pi$ *where*

$$\Pi = \left\{ \Delta \mapsto a_0 + \sum_{l=0}^{M-1} a_{p+l} \Delta^{p+\omega l} :\ a_l \in \Re^d \right\},$$

such that interpolates all the above M *couples, that is,* $\pi\left(\Delta_{n_j}\right) = T_{n_j,j}, \forall j = 1, \ldots, M$ *[169]. The value* $T_{n,0} = u_{\Delta_n}(t_n)$ *can then be obtained by evaluating* π *in* 0, *that is,* $T_{n,0} = \pi(0)$. *The method described above, used to compute* $T_{n,0}$, *can be considered a "*one step method*", and it can be proved [169] that, under some specific conditions, it is a method of order* $p + \omega M$.

The dynamics of processes that have multiple physics and multiscale components can be categorized into relatively fast and slow terms. The informal expressions *"stiff"* and *"nonstiff"* are commonly associated with the fast and slow evolution, respectively. So we are concerned with solving the following problem:

$$\dot{u}(t) = f(t, u(t)) + g(t, u(t)) = F(t, u(t)) \tag{186}$$

where the functions f and g represent the nonstiff and the stiff components of the problem, respectively, and where $u(t_0) = u_0$, $u(t) \in \Re^d$ and where $t \in \Theta = [t_0\ T]$.

In Constantinescu et al. [170,171], the following base "one step methods" (with step size h) $\left\{ {}^{(\bullet)}\Psi_h^{n,n-1} i \right\}_{\bullet=}$[W-IMEX,Pure-IMEX,Split-IMEX] are used to solve (186) by an approach based on extrapolation methods:

$$u^{n+1} = u^n + \left[I - h\frac{\partial g}{\partial u}(t_n, u^n)\right]^{-1}[hF(t_n, u^n)] \quad \text{[W-IMEX]} \quad (187)$$

$$u^{n+1} = u^n + hf(t_n, u^n) + \left[I - h\frac{\partial g}{\partial u}(t_n, u^n)\right]^{-1}[hg(t_n, u^n)] \quad \text{[Pure-IMEX]} \quad (188)$$

$$u^{n+1} = u^* + \left[I - h\frac{\partial g}{\partial u}(t_n, u^n)\right]^{-1}[hg(t_n, u^*)];\ u^* = u^n + hf(t_n, u^n) \quad \text{[Split-IMEX]} \quad (189)$$

The computation of each $T_{n_j,j}$ during the implementation of an extrapolation method (see Definition 2) can be executed independently from the others, so the wall set of tasks to compute all the $T_{n_j,j}$ values can be executed concurrently. Then, extrapolation methods, as well as improving the convergence order of a *"one step method"*, can be considered as a first approach to implement "parallelism" in a naturally serial process as the ODE solution by time steppers. Indeed, current methods for transient simulations are almost exclusively sequential in time. Time progresses by employing one or more solutions from past timesteps to compute the next, regardless of how spatial domains are discretized. As a result, the number of timesteps multiplied by the speed at which one timestep can be computed determines the time to solution for transient applications.

To overcome such limitations on concurrency levels exploited by traditional time integrators, parallel-time algorithms have been studied for many years [172]. Based on Gander et al., who very recently created a unified framework for Parallel-In-Time algorithms [173], we describe them as the iterative updating of the approximation for the solution of the *"all-at-once global problem"* (see Equation (192) in Definition 3). In particular, following the approach described in Gander et al., the Definition 3 for the Parallel-In-Time method is given as follows,

Definition 3. *The Parallel-In-Time (*PInt*) iterative solution of time-dependent ODE*
The aim is to compute an approximate solution of the following ODE problem

$$\dot{u}(t) = A(t, u(t)) \quad (190)$$

where $u(t_0) = u_0$, $u(t) \in \Re^d$ *and where* $t \in \Theta = [t_0\ T]$. *As described in Section 6.1.2 about the time integration algorithms, to compute an approximation of the solution in the considered time domain* Θ, *a discretization* Θ_N *of* Θ *should be considered. For this purpose, consider* Θ_N *as the set made up of the* $N+1$ *equally spaced points* $\{t_n\}_{n=0,\ldots,N}$ *where* $t_n = t_0 + \Delta * n, n = 1, \ldots, N$ *and where* $\Delta = \frac{T}{N}$.

Let a time block (or simply block) denote the discretization of a time subinterval $[t_n, t_{n+1}]$ *using* $M > 0$ *grid points,*

$$\tau_{n,m} = t_n + \Delta_m \tau_m, m \in \{1, ..., M\},$$

where $\tau_m \in [0, 1]$ *denotes normalized grid points in time used for all N time subintervals.*

Let a block variable be a vector

$$\mathbf{u}_n = [u_{n,1}, \ldots, u_{n,M}]^T$$

where $u_{n,m}$ *is an approximation of* $u(\tau_{n,m})$ *on the time block for the time subinterval* $[t_n, t_{n+1}]$.

Finally, let us denote as block operators the two functions $\phi : \Re^M \to \Re^M$ *and* $\chi : \Re^M \to \Re^M$ *for which the block variables of a numerical solution of (190) satisfy*

$$\begin{aligned} \phi(\mathbf{u}_1) &= \chi(u_0\mathbf{1}), \\ \phi(\mathbf{u}_{n+1}) &= \chi(\mathbf{u}_n),\ n = 1, \ldots, N-1 \end{aligned} \quad (191)$$

where $\mathbf{1} = [1, \ldots, 1]^T$, *where the time integration operator* ϕ *is bijective and where* χ *is a transmission operator. The time propagator* ψ *updating* $\mathbf{u}_n$ *to* $\mathbf{u}_{n+1}$ *is given by*

$$\psi = \phi^{-1}\chi.$$

The numerical approximation (191) of (190) can be described as the following all-at-once global problem

$$\begin{pmatrix} \phi & & & \\ -\chi & \phi & & \\ & \ddots & \ddots & \\ & & -\chi & \phi \end{pmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_N \end{bmatrix} = \begin{bmatrix} \chi(u_0\mathbf{1}) \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \tag{192}$$

Iterative PinT algorithms solve (192) by updating a vector $\mathbf{u}^k = \left[u_1^k, \ldots, u_N^k\right]^T$ *to* $\mathbf{u}^{k+1}$ *until some stopping criterion is satisfied. If the global iteration can be written as a local update for each block variable separately, the local update formula is called a block iteration.*

Let a primary block iteration be an updating formula of the form

$$\mathbf{u}_{n+1}^{k+1} = \mathbf{B}_1^0\left(\mathbf{u}_{n+1}^k\right) + \mathbf{B}_0^1\left(\mathbf{u}_n^{k+1}\right) + \mathbf{B}_0^0\left(\mathbf{u}_n^k\right),\ \mathbf{u}_0^k = u_0\mathbf{1},\ \forall k \in \mathbb{N}, \tag{193}$$

where $n \geq 0$*, and where* $\mathbf{B}_1^0$*,* $\mathbf{B}_0^1$*,* $\mathbf{B}_0^0$ *are operators that satisfy the consistency condition*

$$\left(\mathbf{B}_1^0 - \mathbf{I}\right)\psi + \mathbf{B}_0^1 + \mathbf{B}_0^0 = 0. \tag{194}$$

The solution of the all-at-once global problem *(192) by an iterative method based on a primary block iteration of the type (193) represents the Parallel-In-Time approach for the solution of time-dependent ODE (190).*

In Table 9, a summary of the the most popular PInt methods, and their block iteration operators, are reported (see Gander et al. [173] for details).

Table 9. Summary of the the most popular *PInt* methods and their block iteration operators, where $\omega > 0$ is a relaxation parameter, $\mathbf{T}_C^F$ and $\mathbf{T}_F^C$ are transfer operators from *"coarse to fine"* and *"fine to coarse"* time blocks, respectively, and where ϕ_C is the time integration operator on a coarse time block.

Methods	$\mathbf{B}_1^0\left(\mathbf{u}_{n+1}^k\right)$	$\mathbf{B}_0^1\left(\mathbf{u}_n^{k+1}\right)$	$\mathbf{B}_0^0\left(\mathbf{u}_n^k\right)$
DBJ	$\mathbf{I} - \omega\mathbf{I}$	$\omega\phi^{-1}\chi$	–
ABJ	$\mathbf{I} - \tilde{\phi}^{-1}\phi$	$\tilde{\phi}^{-1}\chi$	–
ABGS	$\mathbf{I} - \tilde{\phi}^{-1}\phi$	–	$\tilde{\phi}^{-1}\chi$
Parareal	–	$(\phi^{-1} - \tilde{\phi}^{-1})\chi$	$\tilde{\phi}^{-1}\chi$
TMG	$(1-\omega)\left(\mathbf{I} - \mathbf{T}_C^F\phi_C^{-1}\mathbf{T}_F^C\phi\right)$	$\omega\left(\phi^{-1} - \mathbf{T}_C^F\phi_C^{-1}\mathbf{T}_F^C\right)\chi$	$\mathbf{T}_C^F\phi_C^{-1}\mathbf{T}_F^C\chi$
TMG_c	–	$\left(\phi^{-1} - \mathbf{T}_C^F\tilde{\phi}_C^{-1}\mathbf{T}_F^C\right)\chi$	$\mathbf{T}_C^F\tilde{\phi}_C^{-1}\mathbf{T}_F^C\chi$
TMG_f	$\left(\mathbf{I} - \mathbf{T}_C^F\phi_C^{-1}\mathbf{T}_F^C\phi\right)(\mathbf{I} - \tilde{\phi}^{-1}\phi)$	$\left(\tilde{\phi}^{-1} - \mathbf{T}_C^F\phi_C^{-1}\mathbf{T}_F^C\phi\tilde{\phi}^{-1}\right)\chi$	$\mathbf{T}_C^F\phi_C^{-1}\mathbf{T}_F^C\chi$
PFASST	$\left(\mathbf{I} - \mathbf{T}_C^F\tilde{\phi}_C^{-1}\mathbf{T}_F^C\phi\right)(\mathbf{I} - \tilde{\phi}^{-1}\phi)$	$\left(\tilde{\phi}^{-1} - \mathbf{T}_C^F\tilde{\phi}_C^{-1}\mathbf{T}_F^C\phi\tilde{\phi}^{-1}\right)\chi$	$\mathbf{T}_C^F\tilde{\phi}_C^{-1}\mathbf{T}_F^C\chi$

Algorithm 10 describes an implementation of the *PInt* approach. We observe that Algorithm 10 lends itself well to parallelization: indeed, if we imagine dividing the N vectors $\mathbf{u}_n$ among P concurrent tasks, then

- The initial approximation vectors $\mathbf{u}_n^0$ (see lines 5–7 of Algorithm 10) can be computed concurrently and independently of each other in an *"embarrassingly parallel"* approach (no communications are needed among tasks);
- The initial vectors $\mathbf{u}_0^k$ (see lines 8–10 of Algorithm 10) can be computed by all the P tasks concurrently and independently of each other;
- In each iteration of the iterative method that is needed to refine the approximation of the vectors $\mathbf{u}_n$ (see lines 11–17 of Algorithm 10), the computation of vectors $\mathbf{z}_n^k$ can be performed concurrently and locally to the assigned task p. To complete the computation of $\mathbf{u}_{n+1}^{k+1}$, task p should eventually send the vector $\mathbf{w}_n^k$ to its right neighbor task $p+1$ and should receive the vector $\mathbf{z}_{n-1}^k$ from its left neighbor task $p-1$.

Algorithm 10 The algorithm of the Parallel-In-Time iterative solution of ODE.

procedure PINT(ϕ, χ, u_0, $\mathbf{B}_0^0$, $\mathbf{B}_1^0$, $\mathbf{B}_0^1$, N, K, $\left\{\mathbf{u}_n^K\right\}_{n=1,\ldots,N}$)
Input: ϕ, χ, u_0, $\mathbf{B}_0^0$, $\mathbf{B}_1^0$, $\mathbf{B}_0^1$, N, K
Output: $\left\{\mathbf{u}_n^K\right\}_{n=1,\ldots,N}$
  ▷ Compute initial approximation vectors $\mathbf{u}_n^0$ by a one step method applied M times
  for $n = 1$ to N **do**
    Compute $\mathbf{u}_n^0 = \left[u_{n,1}^0, \ldots, u_{n,M}^0\right]^T$
  end for
  for $k = 1$ to K **do** ▷ Compute vectors $\mathbf{u}_0^k$
    Compute $\mathbf{u}_0^k = u_0\mathbf{1}$
  end for
  for $k = 0$ to $K - 1$ **do** ▷ Compute approximation vectors $\mathbf{u}_n^k$ by iterating on k
    for $n = 0$ to $N - 1$ **do**
      $\mathbf{z}_n^k \leftarrow$ **Compute** $\mathbf{B}_0^1\left(\mathbf{u}_n^{k+1}\right) + \mathbf{B}_0^0\left(\mathbf{u}_n^k\right)$
      $\mathbf{w}_n^k \leftarrow$ **Compute** $\mathbf{B}_1^0\left(\mathbf{u}_{n+1}^k\right)$
      $\mathbf{u}_{n+1}^{k+1} \leftarrow$ **Compute** $\mathbf{u}_{n+1}^{k+1} = \mathbf{z}_n^k + \mathbf{w}_n^k$
    end for
  end for
end procedure

7.2.2. Composite Nonlinear Solver

An key part of contemporary simulations is large-scale algebraic solvers for nonlinear problems. Newton's techniques begin with the nonlinear operator's global linearization. After linearization, a huge sparse linear system is produced, and the matrix can be implicitly represented by several *"matrix-free"* techniques or explicitly represented by storing the nonzero coefficients [165]. As a stand-alone solver, Newton's approach has a number of shortcomings. Memory bandwidth and communication limitations are caused by the repetitive linearization construction and solution [174].

According to Brune et al. [174] we consider a collection of solvers for nonlinear equations of the type

$$\boldsymbol{F}(\boldsymbol{u}) = \boldsymbol{b}. \tag{195}$$

A small set of algorithmic building blocks is shown, which, through composite combinations or nonlinear preconditioning, yield a wide range of solvers with varying convergence and performance characteristics. This is an easily understood comparison to the situation of linear solvers. Implementations of this kind of solvers are available in the PETSc software library and may be brought to bear on many relevant applications.

The history of the *"Portable, Extensible Toolkit for Scientific computing (PETSc)"* [27,175] began in the early 1990s and has always focused on the largest-scale parallel systems available at the time. PETSc had the original goal to develop efficient algebraic solvers and provide the ability to use any appropriate solver for any suitable set of PDEs and can be considered one of the most important and widely used mathematical software libraries. New advances in PETSc software library are now available to accomplish the needs of the emerging extreme-scale architecture [176].

Let us express the action of a stationary nonlinear solver that employs the solution technique M as follows: $\boldsymbol{u}_{i+1} = M(\boldsymbol{F}, \boldsymbol{u}_i, \boldsymbol{b})$. A sequence or series of two (or more) solution techniques M and N, which both provide an approximate solution to (195), make up a nonlinear composite combination. On the other hand, by using inner technique N, nonlinear preconditioning can be expressed as a modification of the function $\boldsymbol{F}$. After that, an outer solver M receives the modified function and uses it to solve the preconditioned system. In general, composite and preconditioning combinations of different solution methods can be considered as an effective acceleration strategy to be used in approximate solutions of Problem (195).

A compact representation for nonlinear composite combinations and preconditioning is described in Table 10. A list of solution methods M that can be considered as building blocks for different combinations are listed in Brune et al. [174]. In these works, we describe just the Newton–Krylov solution methods (*" NEWT-K"*) since they are *"... often the workhorse of simulations requiring solution of nonlinear equations ..."* [174].

In Algorithm 11, the Newton–Krylov solution methods are described: details about the cited (see line 7 in Algorithm 11) "line search" algorithm (or other *"globalization strategies"*) can be found in Brune et al. [174] or in Pawlowski et al. [177]. Newton-Krylov's basic method is very sensitive to the initial iterate and may diverge if it is not sufficiently close to the solution. Generally, globalization strategies seek directions $\Delta \boldsymbol{u}$ and step lengths λ in such a way that such convergence problems can be reduced [177].

An extremely important class of methods comprises nonlinear variants of domain decomposition and multilevel algorithms. The convergence of the decomposition solvers is not guaranteed. Although they may converge, since the decomposition solvers provide additional opportunities for parallelism, the logical extension is to employ them in conjunction with the global solvers as nonlinear preconditioners or accelerators. Among these kinds of methods, we introduce the *"Nonlinear Additive Schwarz Method (NASM)"* [178] for its properties concerning computational "granularity," which is desirable in a parallel computing context.

Table 10. Nonlinear composite combinations and preconditioning given outer and inner solver M and N.

Composite		
Type	**Statement**	**Abbreviation**
Additive	$\mathbf{u}_{i+1} = \mathbf{u}_i + \alpha_M(M(\mathbf{F}, \mathbf{u}_i, \mathbf{b}) - \mathbf{u}_i) + \alpha_N(N(\mathbf{F}, \mathbf{u}_i, \mathbf{b}) - \mathbf{u}_i)$	$M + N$
Multiplicative	$\mathbf{u}_{i+1} = M(\mathbf{F}, N(\mathbf{F}, \mathbf{u}_i, \mathbf{b}), \mathbf{b})$	$M * N$
Preconditioning		
Type	**Statement**	**Abbreviation**
Left	$\mathbf{u}_{i+1} = M(\mathbf{u}_i - N(\mathbf{F}, \mathbf{u}_i, \mathbf{b}), \mathbf{u}_i, \mathbf{b})$	$M -_L N$
Right	$\mathbf{u}_{i+1} = M(\mathbf{F}(N(\mathbf{F}, \mathbf{u}_i, \mathbf{b}), \mathbf{u}_i, \mathbf{b}))$	$M -_R N$

Algorithm 11 The algorithm implementing the *"Newton–Krylov methods"* for the approximate solution of Equation (195).

1: **procedure** NEWTONKRYLOVSOLVER($\boldsymbol{F}$, $\boldsymbol{b}$, $\boldsymbol{u}_0$, N, $\boldsymbol{u}_N$)
2: **Input:** $\boldsymbol{G}$, $\boldsymbol{b}$, $\boldsymbol{u}_0$, N
3: **Output:** $\boldsymbol{u}_N$
4: **for** $n = 0$ to $N - 1$ **do**
5: $\boldsymbol{r}(\boldsymbol{u}_n) \leftarrow$ **Compute** $\boldsymbol{F}(\boldsymbol{u}_n) - \boldsymbol{b}$
6: $\boldsymbol{J}_n \leftarrow$ **Compute** $\boldsymbol{J}_n$
7: $\Delta \boldsymbol{u}_n \leftarrow$ **Solve** $\boldsymbol{J}_n \Delta \boldsymbol{u}_n = \boldsymbol{r}(\boldsymbol{u}_n)$ ▷ The linear system is solved by a Krylov method
8: $\boldsymbol{u}_{n+1} \leftarrow$ **Compute** $\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \lambda \Delta \boldsymbol{u}_n$ ▷ λ is determined by line search
9: **end for**
10: **end procedure**

Definition 4. *The Nonlinear Additive Schwarz Method (NASM)*
Let us consider the solution of problem (195) where $\boldsymbol{u} \in \Re^n$*, and let* $\{\mathfrak{I}_d\}_{d=1,\ldots,N}$ *be a set of subsets of* $\mathfrak{I} = \{1, \ldots, n\}$ *whose overall union coincides with* $\mathfrak{I}$ *(i.e.,* $\bigcup_{d=1,\ldots,N} \mathfrak{I}_d = \mathfrak{I}$*). Each index subset* $\mathfrak{I}$ *defines a subdomain* $\mathfrak{D}_d \subseteq \Re^{n_d}$ *of dimension* $n_d = |\mathfrak{I}_d|$ *of* $\Re^n$*.*

Let us consider the two sets $\{\mathbf{R}_d\}_{d=1,\ldots,N}$ *and* $\{\mathbf{P}_d\}_{d=1,\ldots,N}$ *of* "restriction" *and* "prolongation" *operators, respectively, with the assumption that* $\mathbf{R}_d$ *and* $\mathbf{P}_d$ *satisfy* $\sum_{d=1,\ldots,N} \mathbf{P}_d \mathbf{R}_d = I$*.*

Examples for $\mathbf{R}_d$ *and* $\mathbf{P}_d$*, related with disjoint subsets* $\mathfrak{I}_j$ *(i.e.,* $\mathfrak{I}_j \cap \mathfrak{I}_i = \emptyset, \forall i \neq j$*), are, respectively,*

$$\mathbf{R}_d : \boldsymbol{u} \in \Re^n \mapsto \boldsymbol{u}^d \in \Re^{n_d} \qquad \text{where } \boldsymbol{u}^d = [\boldsymbol{u}_k]_{k \in \mathfrak{I}_d}^T$$

$$\mathbf{P}_d : \boldsymbol{u}^d \in \Re^{n_d} \mapsto \boldsymbol{u} \in \Re^n \qquad \text{where } \boldsymbol{u}_k = \begin{cases} \boldsymbol{u}_k^d & \text{if } k \in \Im_d \\ 0 & \text{if } k \notin \Im_d \end{cases}$$

Moreover, let $\boldsymbol{F}_d\left(\boldsymbol{u}^d\right)$ *denote the* "restriction" *of operator* $\boldsymbol{F}(\boldsymbol{u})$ *to the d-th subdomain* $\mathfrak{D}_d$ *defined as*

$$\boldsymbol{F}_d : \boldsymbol{u}^d \in \mathfrak{D}_d \mapsto \boldsymbol{F}_d\left(\boldsymbol{u}^d\right) = \mathbf{R}_d\left(\boldsymbol{F}\left(\mathbf{P}_d\left(\boldsymbol{u}^d\right)\right)\right) \in \mathfrak{D}_d.$$

If a set of solver method $\{M_d\}_{d=1,\ldots,N}$*, one for each subdomain* $\mathfrak{D}_d$*, is given, the Nonlinear Additive Schwarz solver Method (NASM) is based on the following statement:*

$$\boldsymbol{u}_{i+1} = \sum_{d=1}^{N} \mathbf{P}_d\left(M_d\left(\boldsymbol{F}_d, \boldsymbol{u}_i^d, \boldsymbol{b}^d\right)\right).$$

NASM can be used combined with a Newton–Krylov method: or (1) as its left preconditioner (i.e., see Dolean et al. [178]), or (2) by using *" NEWT-K"* as a local solver on each subdomain of an NASM.

7.2.3. Parallel Iterative and Direct Linear Solver

A sparse linear system solution is sometimes the most computationally demanding step in large-scale research and engineering simulations. Even though iterative approaches are increasingly being used to tackle many of these issues, direct solvers are still frequently used and will remain essential at the Exascale [156,179,180].

The fundamental advantage of direct solutions is their robustness. In fact, when solving extremely ill-conditioned linear systems, direct solvers are often the method of choice, that is, once the system matrix has been rearranged. When compared to iterative methods, where the complexity is related to the primary kernel in an iteration (which is typically a matrix-vector multiplication, the cost of which is typically proportionate to the number of nonzeros in the matrix) and the number of iterations required for convergence, the main drawbacks of direct approaches are their computational expense and memory usage [156].

Exascale computing will continue to require multilevel techniques because of their high concurrency properties. Some multi-level techniques exhibit a *"natural resilience to faults"* [156], which may serve as a useful foundation for the development of fault-tolerant algorithms. While multilevel approaches offer many advantageous characteristics, the applicability of any one algorithm is usually very limited. Exascale computing does, in fact, constrain algorithmic selections of multilevel approaches, conditioning selections for relaxation operators and the Multigrid schema structure (e.g., W-cycles ought to be disregarded due to their substantial communications costs) [156].

In recent years, the effort spent by some scientific communities around the world (for example, see the USA Exascale Computung Project (ECP) Multiprecision Effort Team [181]) has obtained great results in developing and deploying new mixed-precision technology. Some of the most promising and impactful achievements concern:

- Mixed-precision dense and sparse LU factorization based on Iterative Refinement (IR);
- Mixed-Precision GMRES based on IR;
- Compressed-Basis (CB) GMRES.

One of the most crucial extreme scale needs is the relaxation and lowering of synchronization, which may also help to relieve load balancing pressure. It is probably difficult to completely eliminate periodic synchronization in solvers, although many algorithms could be designed in a way that requires a lot less synchronization.

Within the category of standard iterative methods, examples of such a type of algorithm are the *"s-step"* CA-GMRES iterative method [21,182], which uses a redesign of the KSM algorithms based on BLAS 2 and 3 operations (products of a vector by a matrix, matrices products, etc.), which have a higher computational granularity (i.e., a higher ratio of computation time to communication time) than the original KSM formulation based on BLAS 1 operations.

Let $M_{j=0,\ldots,m-1}(\mathbf{A})$ be a suitable set of monomials of the matrix $\mathbf{A}$; then, the *"s-step"* version of Algorithm 5, which solves problem (126), is listed in Algorithm 12, where

$$\begin{aligned} v_{m+j} &= \prod_{i=0}^{j} M_i(\boldsymbol{A}) v_m, \;\; j = 1, \ldots, s \\ \mathfrak{Q}_0 &= [v_0, \ldots, v_{s-1}, v_s], \\ \mathfrak{Q}_{m+1} &= [q_0, \ldots, q_{ms-1}, q_{ms}, q_{ms+1}], \\ \bar{\mathfrak{Q}}_{m+1} &= [q_0, \ldots, q_{ms-1}, q_{ms}], \end{aligned}$$

$$V_{m+1} \in \mathbb{R}^{n \times s}, \mathfrak{Q}_{m+1} \in \mathbb{R}^{n \times (ms+1)}, Q_{m+1} \in \mathbb{R}^{n \times s}, R_{m+1} \in \mathbb{R}^{s \times s}, \mathfrak{H}_{m+1} \in \mathbb{R}^{(ms+1) \times ms}.$$

In the *"s-step"* version of GMRES (see line 5 of Algorithm 12), multiple column vectors $V_{m+1} = [v_{m+1}, \ldots, v_{m+s}]$ are calculated at the same step m, and the new orthonormal basis is computed by two normalizations steps:

1. Orthonormalize V_{m+1} against the orthonormal basis $\mathfrak{Q}_m$ and compute $\bar{V}_{m+1}$,
2. Orthonormalize columns of $\bar{V}_{m+1}$ by a QR factorization [147].

More details on Algorithm 12 can be found in Hoemmen et al. [182].

Algorithm 12 The m-th step of block-based GMRES Method.

1: **procedure** BB-GMRES(A, y, x_0, m, V_m, $\boldsymbol{x}_{m+1}$)
2: **Input:** A, y, x_0, m
3: **Output:** x_{m+1}
4: **OrthoBegin**▷ Build an orthonormal basis of column vectors $\mathfrak{Q}_{m+1} = [\mathfrak{Q}_m, Q_{m+1}]$ of $\mathcal{K}_{m+s}$
5: Compute $V_{m+1} = [v_{m+1}, \ldots, v_{m+s-1}, v_{m+s}]$
6: Compute $\mathfrak{R}_{m+1} = \mathfrak{Q}_m^T V_{m+1}$
7: Compute $\bar{V}_{m+1} = V_{m+1} - \mathfrak{Q}_m \mathfrak{R}_{m+1}$
8: Compute the QR factorization of $\bar{V}_{m+1} = Q_{m+1} R_{m+1}$
9: Compute the Hessenberg matrix $\underline{\mathfrak{H}}_{m+1}$ (from R_{m+1} and $\mathfrak{R}_{m+1}$) such that $A\bar{\mathfrak{Q}}_{m+1} = \mathfrak{Q}_{m+1}\underline{\mathfrak{H}}_{m+1}$
10: **OrthoEnd**
11: **SolBegin** ▷ Extract a suitable vector from a subspace $x_0 + \mathcal{K}_{m+s}$
12: compute $y_{m+1} = \text{argmin}_y \left\| \beta e_1 - \underline{\mathfrak{H}}_{m+1} y \right\|_2$
13: compute $x_{m+1} = x_0 + \mathfrak{Q}_{m+1} y_{m+1}$
14: **SolEnd**
15: **PrepBegin** ▷ Prepare for the next step
16: Assign $v_{m+1} \leftarrow q_{ms+1}$
17: **PrepEnd**
18: **end procedure**

In exact arithmetic, s steps of Algorithm 5 are equivalent to one step of Algorithm 12, but Algorithm 12 suffers from some instabilities when finite precision is used. A lot of work was spent in recent years to identify the origin of these phenomena and to mitigate their consequences (e.g., see Bai et al. [183] about the role of monomials $M_j(\mathcal{A})$ on the *"condition number"* of V_{m+1} and then on the convergence of Algorithm 12). Also, to temper instability effects, s is chosen to be $s << n$, and the execution of Algorithm 12 is restarted [152] just after a few steps $m << n$ [21].

7.2.4. Fast Multipole Methods and *"Hierarchical"* Matrices

Algorithms listed in the 20th century have included the Fast Multipole Methods (FMM), developed by Rokhlin Jr. and Greengard [184] ranked in the top ten. They are initially discussed in the context of particle simulations, where they lower the computational cost from $O(N^2)$ to $O(N)$ to $O(N \log(N))$ operations for every pairwise interaction in a

system of N particles. During its history [185–187], FFM was the basis for other useful applications such as the fast multiplication of vectors with fully populated special matrices. In the context of linear algebra, FFM takes the name of fast multiplication by *"$\mathcal{H}$-matrices"* which are a combination of the *"Panel Clustering Method"*, and the *"mosaic skeleton matrices"* approaches [188,189].

According to [188,189], let us introduce *"Hierarchical" matrices* (also called $\mathcal{H}$-matrices). First of all, it is necessary to introduce the concepts of the ***R**k*-matrix and the *"block cluster quad-tree"*.

Definition 5. *Definition of **R**k-matrix*
Let $R \in \Re^{M\times N}$ be a matrix of the form

$$R = AB^T,\ A \in \Re^{M\times k},\ B \in \Re^{k\times N}.$$

*Then, R is called an **R**k-matrix. Any matrix of a rank of at most k can be represented as an **R**k-matrix, and each **R**k-matrix has at most a rank of k.*

Definition 6. *Definition of "block cluster quad-tree" of a matrix A*
Let $\mathcal{T}_I(L)$ be a binary tree [190] with the $L+1$ levels $l = 0,\ldots,L$ and denote by T_I the set of its nodes. $\mathcal{T}_I$ is called a binary cluster tree corresponding to an index set I if the following conditions hold:

1. *Each node of T_I is a subset of the index set I;*
2. *I is the root of $\mathcal{T}_I(L)$ (i.e., the node at the 0-th level of $\mathcal{T}_I(L)$);*
3. *If $\tau \in T_I$ is a leaf (i.e, a node at the L-th level of $\mathcal{T}_I(L)$), then $|\tau| \leq C_{leaf}$;*
4. *If $\tau \in T_I$ is not a leaf whose set of sons is represented by $S(\tau) \subseteq T_I$, then $|S(\tau)| = 2$ and $\tau = \dot{\bigcup}_{\tau' \in S(\tau)} \tau'$*

Let I be an index and let $AC(\tau\times\sigma) = 0,1$ a logical value representing an "Admissibility Condition" *on $\tau\times\sigma$. Moreover, let $\mathcal{T}_I(L)$ be a binary cluster tree on the index set I. The block cluster quad-tree $\mathcal{T}_{I\times I}$ corresponding to $\mathcal{T}_I(L)$ and to the admissibility condition $AC(\tau\times\sigma)$ could be built by the procedure represented in Algorithm 13.*

Algorithm 13 Procedure for building the block quad-tree $\mathcal{T}_{I\times I}(L)$ corresponding to a cluster tree $\mathcal{T}_I(L)$ and an admissibility condition AC. The index set $\tau\times\sigma$ and the value l to be used in the first call to the recursive BLOCKCLUSTERQUADTREE procedure are such that $\tau = \sigma = I, l = 0$.

```
procedure BlockClusterQuadTree(T_{I×I}(L), L, l, AC(τ×σ), τ×σ)
Input: T_{I×I}(L), AC(τ×σ), L, l, τ×σ
    if (AC(τ×σ) = 0 and l < L) then
        S(τ×σ) = {τ'×σ' : τ' ∈ S(τ), σ' ∈ S(σ)}
        for τ'×σ' ∈ S(τ×σ) do
            BlockClusterQuadTree(T_{I×I}(L), L, l+1, AC(τ'×σ'), τ'×σ')
        end for
    else
        S(τ×σ) = ∅
    end if
end procedure
```

Definition 7. *Definition of $\mathcal{H}$-matrix*
Let $A \in \Re^{N\times N} = \left(a_{i,j}\right)_{i,j=1,\ldots,N}$ be a matrix, let $I = \{1,\ldots,N\}$ the index set of A, and let $k \in \mathcal{N}$. Let us assume that, for a vector v and a subset $\tau \subset I$, $v_{|\tau}$ represents the restriction vector $\left(v_j\right)_{j\in\tau}$ of v to τ, and that, for a matrix A and subsets $\tau,\sigma \subset I$, the notation $A_{|\tau\times\sigma}$ represents the block

$(A_{ij})_{i\in\tau, j\in\sigma}$. *And let* $\mathcal{T}_{I\times I}(L)$ *be the block cluster quad-tree on the index set* I *whose admissibility condition* $AC(\tau\times\sigma)$ *is defined as*

$$AC(\tau\times\sigma)=\begin{cases}1 & \text{if } A_{|\tau\times\sigma} \text{ can be approximated by an } \boldsymbol{R}k\text{-matrix in a specified norm } \|\cdot\|\\ 0 & \text{otherwise}\end{cases}$$

Then, the matrix A *is called the* $\mathcal{H}$*-matrix of blockwise rank* k *defined on block cluster quad-tree* $\mathcal{T}_{I\times I}(L)$.

Let us remember that, given a matrix A*, the matrix* $\tilde{A}$ *is said to be an approximation of* A *in a specified norm* $\|\cdot\|$ *if there exists* $\epsilon\in\Re$ *such that* $\left\|A-\tilde{A}\right\|<\epsilon$.

Different approaches can be used to compute an $\boldsymbol{R}k$-matrix approximation $\tilde{M}=\tilde{M}_1\tilde{M}_2$ of an arbitrary matrix M [188]; as an example, we cite the one based on the *"truncated singular value decomposition,"* which represents the best approximation of an arbitrary matrix in the spectral and Frobenius norm (see [189] for details).

As an example of the use of the $\mathcal{H}$-matrices to obtain algorithms of reduced complexity, we propose the algorithm for the calculation of the matrix-vector product (see Algorithm 14): the complexity for that matrix-vector multiplication algorithm is $O(kn\log(n))$. Other algorithms of interest from basic matrix algebra that use $\mathcal{H}$-matrices are described in Hackbusch et al. [189].

Algorithm 14 Matrix-vector multiplication $y=y+Ax$ of the $\mathcal{H}$-matrix $A\in\Re^{I\times I}$ (defined on block cluster quad-tree $\mathcal{T}_{I\times I}(L)$) with vector $x\in\Re^I$. The index sets $\tau\times\sigma$ to be used in the first call to the recursive HMATRIX-MVM procedure are such that $\tau=\sigma=I$.

```
1:  procedure HMATRIX-MVM(A, y, x, T_{I×I}(L), τ × σ)
2:  Input: A, y, x, T_{I×I}(L), τ × σ
3:  Output: y
4:      if S(τ × σ) ≠ ∅ then                          ▷ τ × σ is not a leaf of T_{I×I}(L)
5:          for each τ' × σ' ∈ S(τ × σ) do
6:              HMATRIX-MVM(A, y, x, T_{I×I}(L), τ' × σ')
7:          end for
8:      else                                          ▷ τ × σ is a leaf of T_{I×I}(L)
9:          y_{|τ} ← y_{|τ} + A_{|τ×σ} x_{|σ}         ▷ A_{|τ×σ} could be an unstructured or an Rk-matrix
10:     end if
11: end procedure
```

7.2.5. Parallel Monte Carlo Based Methods

Monte Carlo-like algorithms seem to have all the characteristics that make them suitable for Exascale computing systems. Taking inspiration from [191,192], we can observe that it is easy enough to implement strategies that make them algorithms with the following attributes.

Communication- and synchronization-avoiding: To implement parallelism in an MC-like algorithm, the named dynamic *bag-of-work model* can be used [191]. Using such a strategy, a large task, related to the computation of M states, is split into smaller independent P subtasks, where each task computes $\frac{M}{P}$ states and where all the P subtasks are executed independently of each other in an *"embarrassingly parallel"* approach.

Fault-Tolerant and Resilient: To make an MC-like algorithm resilient to hard failures, an *P-out-of-Q* strategy can be used [191]. Since MC-based applications *"accuracy"* depend only on the number M of computed states and not on which random sample set is estimated (provided that all the random samples are independent in a statistical sense), the actual size of the computation is obtained by increasing the number of subtasks from P to Q, where $Q>P$. The whole task is considered to be completed when P partial results, from each subtask of size $\frac{M}{P}$ states, are ready. In the *P-out-of-Q*

approach, more subtasks are needed than are actually scheduled. Therefore, none of these subtasks will become a "key" subtask and at most $Q - P$ faulted subtasks can be tolerated. The critical aspect is properly choosing the value Q since a "good" value for Q may prevent a few subtasks from delaying or halting the whole computation. On the other hand, an excessive value for Q could result in a much higher calculation workload with little gain to the computation outcomes. The proper choice of Q in the *P-out-of-Q* strategy can be determined by considering the average job-completion rate in the computing system. Suppose c is the completion probability of subtasks up to time t in the computing system. Clearly, $Q * c$ should be approximately P, i.e., the fraction of the Q subtasks finished should equal to P. Thus, a good choice is $Q = \left\lceil \frac{P}{c} \right\rceil$ [191].

To make an MC-like algorithm resilient to a soft fault causing an error in the computation, for example, due to a change in data value in memory, strategies such as the *"duplicate checking"* or the *"majority vote"* [191] could be considered. Subtasks are repeated and executed on separate nodes in these algorithms. Comparing the outcomes of the same subtask carried out on other nodes might help identify inaccurate computational results (e.g., by ensuring that the results fall within a suitable confidence interval). When performing duplicate checking, an incorrect result must be found by doubling computations. It takes at least three times as much calculation to find an incorrect computation result in a majority vote.

8. Conclusions and Future Work

This work aims at providing a comprehensive summary of the models and methods that can be employed for "in silico" experimentation of the physicochemical processes underlying the process of 3D bioprinting of cell-laden materials by using the computational resources available in the Exascale Era.

Until now, the only way to optimize the properties of cell-laden hydrogels and the 3D bioprinting process has been to replicate multiple expensive and time-consuming trials. The advent of computational models has allowed us to better understand the different phases of the process. However, there are still many challenges to address. Massive data sets, appropriate method selection, and the identification of relevant inputs and outputs are all necessary for the modeling. Large-scale data collection from bioprinting research involving expensive cells and supplies, as well as labor-intensive procedures, could not be feasible. The challenge relating to the selection of numerical methods concerns the development of fully coupled multiscale models, now more affordable in the Exascale Era, that (1) allow the chemical/physical processes of interest to be described more fully and precisely and (2) are solvable effectively and efficiently by new algorithms on the complex computing architectures available in the Exascale Era (for example, consider the definition of the most appropriate preconditioners for iterative methods that are built based on the nature of the reference chemical/physical problem).

This work is the result of an interdisciplinary collaboration since the two authors, respectively, carry out research activities in the fields of material sciences (with particular regard to bio-printing) and computational sciences (with particular regard to algorithms in the HPC context). Then, this work can be considered the synthesis of two inventory processes, the first relating to the numerical techniques already used for the simulation of bioprinting processes, the second relating to the availability of versions of these techniques capable of fully exploiting the new HPC resources in the Exascale Era.

Furthermore, this work intends to lay the foundations for the ambitious project of building an inventory of *"digital skills"* and *"mathematical knowledge"* both necessary for European *"twin green-digital" transitions* based on the case study of advanced materials for health. This case study is taken from the Materials Innovation Markets (MIMs) (defined by the Materials 2030 roadmap of the AMI2030 [193] initiative) which are the markets of primary interest for Europe in terms of impacts (on people, planet, and prosperity), in which advanced materials play a key enabling role. To achieve these goals, Europe intends

to maximize the sustainability characteristics of new advanced materials and their visibility using advanced digital technologies (the so-called *"Materials Digitalization"*) in the MIMs mentioned above.

Unfortunately, due to a mere question of space, in the drafting of this work, some of the most promising tools in the field of *"digitalization of materials"* were not included. Among these tools should be cited the techniques relating to (1) the construction of models starting from data (for example through the use of tools for *"learning from data"*, see [56,57,194,195]) or (2) the possibility of correcting/defining first-principles-based models using data (see for instance the data assimilation approach [196,197]). Combinations of both the techniques listed above should be investigated to implement hybrid approaches. For example, Bradley et al. [8] describe some hybrid approaches to combine first-principles models with data-driven methods, and Carlberg et al. [198] describe an approach to combine the data-driven method with the Parallel-In-Time method. Furthermore, another important topic to consider is that 3D bioprinting is evolving toward 4D Bioprinting, in which 3D bioprinted tissues can adapt in terms of shape, size, or pattern over time, according to external stimuli, adding another level of complexity to the process [199]. These important aspects will be the main objective of our future work.

Author Contributions: Conceptualization, L.C and U.D.; methodology, L.C and U.D.; investigation, L.C. and U.D.; writing—original draft preparation, L.C.; writing—review and editing, L.C. and U.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: Luisa Carracciuolo is member of the "Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM)".

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

References

1. D'Amora, U.; Ronca, A.; Scialla, S.; Soriente, A.; Manini, P.; Phua, J.W.; Ottenheim, C.; Pezzella, A.; Calabrese, G.; Raucci, M.G.; et al. Bioactive Composite Methacrylated Gellan Gum for 3D-Printed Bone Tissue-Engineered Scaffolds. *Nanomaterials* **2023**, *13*, 772.
2. D'Amora, U.; Soriente, A.; Ronca, A.; Scialla, S.; Perrella, M.; Manini, P.; Phua, J.W.; Ottenheim, C.; Di Girolamo, R.; Pezzella, A.; et al. Eumelanin from the Black Soldier Fly as Sustainable Biomaterial: Characterisation and Functional Benefits in Tissue-Engineered Composite Scaffolds. *Biomedicines* **2022**, *10*, 2945.
3. Ferroni, L.; Gardin, C.; D'Amora, U.; Calzà, L.; Ronca, A.; Tremoli, E.; Ambrosio, L.; Zavan, B. Exosomes of mesenchymal stem cells delivered from methacrylated hyaluronic acid patch improve the regenerative properties of endothelial and dermal cells. *Biomater. Adv.* **2022**, *139*, 213000.
4. Zhang, L.; D'Amora, U.; Ronca, A.; Li, Y.; Mo, X.; Zhou, F.; Yuan, M.; Ambrosio, L.; Wu, J.; Raucci, M.G. In vitro and in vivo biocompatibility and inflammation response of methacrylated and maleated hyaluronic acid for wound healing. *RSC Adv.* **2020**, *10*, 32183–32192.
5. Arjoca, S.; Robu, A.; Neagu, M.; Neagu, A. Mathematical and computational models in spheroid-based biofabrication. *Acta Biomater.* **2023**, *165*, 125–139. https://doi.org/10.1016/j.actbio.2022.07.024.
6. Szychlinska, M.A.; Bucchieri, F.; Fucarino, A.; Ronca, A.; D'Amora, U. Three-dimensional bioprinting for cartilage tissue engineering: insights into naturally-derived bioinks from land and marine sources. *J. Funct. Biomater.* **2022**, *13*, 118.
7. Lepowsky, E.; Muradoglu, M.; Tasoglu, S. Towards preserving post-printing cell viability and improving the resolution: Past, present, and future of 3D bioprinting theory. *Bioprinting* **2018**, *11*, e00034. https://doi.org/10.1016/j.bprint.2018.e00034.
8. Bradley, W.; Kim, J.; Kilwein, Z.; Blakely, L.; Eydenberg, M.; Jalvin, J.; Laird, C.; Boukouvala, F. Perspectives on the integration between first-principles and data-driven modeling. *Comput. Chem. Eng.* **2022**, *166*, 107898. https://doi.org/10.1016/j.compchemeng.2022.107898.
9. Kovalchuk, S.V.; de Mulatier, C.; Krzhizhanovskaya, V.V.; Mikyška, J.; Paszyński, M.; Dongarra, J.; Sloot, P.M. Computation at the Cutting Edge of Science. *J. Comput. Sci.* **2024**, 102379. https://doi.org/10.1016/j.jocs.2024.102379.
10. Naghieh, S.; Chen, X. Printability—A key issue in extrusion-based bioprinting. *J. Pharm. Anal.* **2021**, *11*, 564–579. https://doi.org/10.1016/j.jpha.2021.02.001.
11. Gómez-Blanco, J.C.; Mancha-Sànchez, E.; Marcos, A.C.; Matamoros, M.; Dìaz-Parralejo, A.; Pagador, J.B. Bioink Temperature Influence on Shear Stress, Pressure and Velocity Using Computational Simulation. *Processes* **2020**, *8*, 865. https://doi.org/10.3390/pr8070865.

12. Karvinen, J.; Kellomaki, M. Design aspects and characterization of hydrogel-based bioinks for extrusion-based bioprinting. *Bioprinting* **2023**, *32*, e00274. https://doi.org/10.1016/j.bprint.2023.e00274.
13. Carlier, A.; Skvortsov, G.A.; Hafezi, F.; Ferraris, E.; Patterson, J.; Koç, B.; Oosterwyck, H.V. Computational model-informed design and bioprinting of cell-patterned constructs for bone tissue engineering. *Biofabrication* **2016**, *8*, 025009. https://doi.org/10.1088/1758-5090/8/2/025009.
14. Hull, S.M.; Brunel, L.G.; Heilshorn, S.C. 3D Bioprinting of Cell-Laden Hydrogels for Improved Biological Functionality. *Adv. Mater.* **2021**, *34*, 2103691. https://doi.org/10.1002/adma.202103691.
15. Schwab, A.; Levato, R.; D'Este, M.; Piluso, S.; Eglin, D.; Malda, J. Printability and Shape Fidelity of Bioinks in 3D Bioprinting. *Chem. Rev.* **2020**, *120*, 11028–11055. https://doi.org/10.1021/acs.chemrev.0c00084.
16. D'Amora, U.; D'Este, M.; Eglin, D.; Safari, F.; Sprecher, C.M.; Gloria, A.; De Santis, R.; Alini, M.; Ambrosio, L. Collagen density gradient on three-dimensional printed poly(ϵ-caprolactone) scaffolds for interface tissue engineering. *J. Tissue Eng. Regen. Med.* **2018**, *12*, 321–329. https://doi.org/10.1002/term.2457.
17. Hölzl, K.; Lin, S.; Tytgat, L.; Vlierberghe, S.V.; Gu, L.; Ovsianikov, A. Bioink properties before, during and after 3D bioprinting. *Biofabrication* **2016**, *8*, 032002. https://doi.org/10.1088/1758-5090/8/3/032002.
18. USA National Institute of Standards and Technology (NIST). Computational Science. Available online: https://www.nist.gov/computational-science (accessed on 31 March 2024).
19. Carracciuolo, L.; Lapegna, M. Implementation of a non-linear solver on heterogeneous architectures. *Concurr. Comput. Pract. Exp.* **2018**, *30*, e4903. https://doi.org/10.1002/cpe.4903.
20. Mele, V.; Constantinescu, E.M.; Carracciuolo, L.; D'Amore, L. A PETSc parallel-in-time solver based on MGRIT algorithm. *Concurr. Comput. Pract. Exp.* **2018**, *30*, e4928. https://doi.org/10.1002/cpe.4928.
21. Carracciuolo, L.; Mele, V.; Szustak, L. About the granularity portability of block-based Krylov methods in heterogeneous computing environments. *Concurr. Comput. Pract. Exp.* **2021**, *33*, e6008. https://doi.org/10.1002/cpe.6008.
22. Carracciuolo, L.; Casaburi, D.; D'Amore, L.; D'Avino, G.; Maffettone, P.; Murli, A. Computational simulations of 3D large-scale time-dependent viscoelastic flows in high performance computing environment. *J. -Non-Newton. Fluid Mech.* **2011**, *166*, 1382–1395. https://doi.org/10.1016/j.jnnfm.2011.08.014.
23. Carracciuolo, L.; D'Amore, L.; Murli, A. Towards a parallel component for imaging in PETSc programming environment: A case study in 3D echocardiography. *Parallel Comput.* **2006**, *32*, 67–83. https://doi.org/10.1016/j.parco.2005.09.001.
24. Murli, A.; D'Amore, L.; Carracciuolo, L.; Ceccarelli, M.; Antonelli, L. High performance edge-preserving regularization in 3D SPECT imaging. *Parallel Comput.* **2008**, *34*, 115–132. https://doi.org/10.1016/j.parco.2007.12.004.
25. D'Amore, L.; Constantinescu, E.; Carracciuolo, L. A Scalable Space-Time Domain Decomposition Approach for Solving Large Scale Nonlinear Regularized Inverse Ill Posed Problems in 4D Variational Data Assimilation. *J. Sci. Comput.* **2022**, *91*, 59. https://doi.org/10.1007/s10915-022-01826-7.
26. Foster, I. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1995.
27. Balay, S.; Abhyankar, S.; Adams, M.F.; Benson, S.; Brown, J.; Brune, P.; Buschelman, K.; Constantinescu, E.M.; Dalcin, L.; Dener, A.; et al. PETSc Web Page. Available online: https://petsc.org/ (accessed on 31 March 2024).
28. The Trilinos Project Web Page. Available online: https://trilinos.org (accessed on 31 March 2024).
29. The German Priority Programme "Software for Exascale Computing" (SPPEXA) Web Page. Available online: http://www.sppexa.de/ (accessed on 31 March 2024).
30. Nasrollahzadeh, N.; Applegate, L.A.; Pioletti, D.P. Development of an Effective Cell Seeding Technique: Simulation, Implementation, and Analysis of Contributing Factors. *Tissue Eng. Part Methods* **2017**, *23*, 485–496. https://doi.org/10.1089/ten.tec.2017.0108.
31. Olivares, A.L.; Lacroix, D. Simulation of Cell Seeding within a Three-Dimensional Porous Scaffold: A Fluid-Particle Analysis. *Tissue Eng. Part Methods* **2012**, *18*, 624–631. https://doi.org/10.1089/ten.tec.2011.0660.
32. Dechriste, G.; Fehrenbach, J.; Griseti, E.; Lobjois, V.; Poignard, C. Viscoelastic modeling of the fusion of multicellular tumor spheroids in growth phase. *J. Theor. Biol.* **2018**, *454*, 102–109. https://doi.org/10.1016/j.jtbi.2018.05.005.
33. Reid, J.A.; Mollica, P.A.; Johnson, G.D.; Ogle, R.C.; Bruno, R.D.; Sachs, P.C. Accessible bioprinting: Adaptation of a low-cost 3D-printer for precise cell placement and stem cell differentiation. *Biofabrication* **2016**, *8*, 025017. https://doi.org/10.1088/1758-5090/8/2/025017.
34. Li, M.; Tian, X.; Schreyer, D.J.; Chen, X. Effect of needle geometry on flow rate and cell damage in the dispensing-based biofabrication process. *Biotechnol. Prog.* **2011**, *27*, 1777–1784. https://doi.org/10.1002/btpr.679.
35. Silva, C.; Cortres-Rodriguez, C.J.; Hazur, J.; Reakasame, S.; Boccaccini, A.R. Rational Design of a Triple-Layered Coaxial Extruder System: In silico and in vitro Evaluations Directed Toward Optimizing Cell Viability. *Int. J. Bioprinting* **2020**, *6*, 282. https://doi.org/10.18063/ijb.v6i4.282.
36. Sego, T.J.; Kasacheuski, U.; Hauersperger, D.; Tovar, A.; Moldovan, N.I.. A heuristic computational model of basic cellular processes and oxygenation during spheroid-dependent biofabrication. *Biofabrication* **2017**, *9*, 024104. https://doi.org/10.1088/1758-5090/aa6ed4.
37. Li, M.; Tian, X.; Kozinski, J.A.; Chen, X.; Hwang, D.K. Modeling Mechanical Cell Damage in the Bioprinting Process Employing a Conical Needle. *J. Mech. Med. Biol.* **2015**, *15*, 1550073. https://doi.org/10.1142/S0219519415500736.

38. Liravi, F.; Darleux, R.; Toyserkani, E. Additive manufacturing of 3D structures with non-Newtonian highly viscous fluids: Finite element modeling and experimental validation. *Addit. Manuf.* **2017**, *13*, 113–123. https://doi.org/10.1016/j.addma.2016.10.008.
39. Samanipour, R.; Wang, Z.; Ahmadi, A.; Kim, K. Experimental and computational study of microfluidic flow-focusing generation of gelatin methacrylate hydrogel droplets. *J. Appl. Polym. Sci.* **2016**, *133*, 43701. https://doi.org/10.1002/app.43701.
40. Prendergast, M.E.; Burdick, J.A. Computational Modeling and Experimental Characterization of Extrusion Printing into Suspension Baths. *Adv. Healthc. Mater.* **2021**, *11*, 2101679. https://doi.org/10.1002/adhm.202101679.
41. Carlier, A.; Geris, L.; Bentley, K.; Carmeliet, G.; Carmeliet, P.; Van Oosterwyck, H. Correction: MOSAIC: A Multiscale Model of Osteogenesis and Sprouting Angiogenesis with Lateral Inhibition of Endothelial Cells. *PLoS Comput. Biol.* **2013**, *9*, e1002724. https://doi.org/10.1371/annotation/38264a13-d4b5-49cd-b54e-47330bb19fe9.
42. Gironi, P.; Petraro, L.; Santoni, S.; Dede', L.; Colosimo, B.M. A computational model of cell viability and proliferation of extrusion-based 3D-bioprinted constructs during tissue maturation process. *Int. J. Bioprinting* **2023**, *9*, 741. https://doi.org/10.18063/ijb.741.
43. Göhl, J.; Markstedt, K.; Mark, A.; Håkansson, K.; Gatenholm, P.; Edelvik, F. Simulations of 3D bioprinting: Predicting bioprintability of nanofibrillar inks. *Biofabrication* **2018**, *10*, 034105. https://doi.org/10.1088/1758-5090/aac872.
44. Sun, Y.; Wang, Q. Modeling and simulations of multicellular aggregate self-assembly in biofabrication using kinetic Monte Carlo methods. *Soft Matter* **2013**, *9*, 2172–2186. https://doi.org/10.1039/C2SM27090K.
45. Jakab, K.; Norotte, C.; Damon, B.; Marga, F.; Neagu, A.; Besch-Williford, C.L.; Kachurin, A.; Church, H.K.; Park, H.; Mironov, V.; et al. Tissue Engineering by Self-Assembly of Cells Printed into Topologically Defined Structures. *Tissue Eng. Part A* **2008**, *14*, 413–421. https://doi.org/10.1089/tea.2007.0173.
46. Cristea, A.; Neagu, A. Shape changes of bioprinted tissue constructs simulated by the Lattice Boltzmann method. *Comput. Biol. Med.* **2016**, *70*, 80–87. https://doi.org/10.1016/j.compbiomed.2015.12.020.
47. Shafiee, A.; McCune, M.; Forgacs, G.; Kosztin, I. Post-deposition bioink self-assembly: A quantitative study. *Biofabrication* **2015**, *7*, 045005. https://doi.org/10.1088/1758-5090/7/4/045005.
48. McCune, M.; Shafiee, A.; Forgacs, G.; Kosztin, I. Predictive modeling of post bioprinting structure formation. *Soft Matter* **2014**, *10*, 1790–1800. https://doi.org/10.1039/C3SM52806E.
49. Flenner, E.; Marga, F.; Neagu, A.; Kosztin, I.; Forgacs, G. Relating Biophysical Properties Across Scales. In *Multiscale Modeling of Developmental Systems*; Current Topics in Developmental Biology Series; Schnell, S., Maini, P.K., Newman, S.A., Newman, T.J., Ed.; Academic Press: Cambridge, MA, USA, 2008; Volume 81, pp. 461–483. https://doi.org/10.1016/S0070-2153(07)81016-7.
50. Mohammadrezaei, D.; Moghimi, N.; Vandvajdi, S.; Powathil, G.; Hamis, S.; Kohandel, M. Predicting and elucidating the post-printing behavior of 3D printed cancer cells in hydrogel structures by integrating in-vitro and in-silico experiments. *Sci. Rep.* **2023**, *13*, 1211. https://doi.org/10.1038/s41598-023-28286-9.
51. Kuan, H.S.; Pönisch, W.; Jülicher, F.; Zaburdaev, V. Continuum Theory of Active Phase Separation in Cellular Aggregates. *Phys. Rev. Lett.* **2021**, *126*, 018102. https://doi.org/10.1103/PhysRevLett.126.018102.
52. Beaune, G.; Sinkkonen, L.; Gonzalez-Rodriguez, D.; Timonen, J.V.; Brochard-Wyart, F. Fusion Dynamics of Hybrid Cell-Microparticle Aggregates: A Jelly Pearl Model. *Langmuir* **2022**, *38*, 5296–5306. https://doi.org/10.1021/acs.langmuir.1c02949.
53. Aguilar, I.N.; Smith, L.J.; Olivos, D. J., III; Chu, T.M.G.; Kacena, M.A.; Wagner, D.R. Scaffold-free bioprinting of mesenchymal stem cells with the regenova printer: Optimization of printing parameters. *Bioprinting* **2019**, *15*, e00048. https://doi.org/10.1016/j.bprint.2019.e00048.
54. Semple, J.L.; Woolridge, N.; Lumsden, C.J. Review: In Vitro, In Vivo, In Silico: Computational Systems in Tissue Engineering and Regenerative Medicine. *Tissue Eng.* **2005**, *11*, 341–356. https://doi.org/10.1089/ten.2005.11.341.
55. Bardini, R.; Di Carlo, S. Computational methods for biofabrication in tissue engineering and regenerative medicine—A literature review. *Comput. Struct. Biotechnol. J.* **2024**, *23*, 601–616. https://doi.org/10.1016/j.csbj.2023.12.035.
56. Xu, H.; Liu, Q.; Casillas, J.; Mcanally, M.; Mubtasim, N.; Gollahon, L.S.; Wu, D.; Xu, C. Prediction of cell viability in dynamic optical projection stereolithography-based bioprinting using machine learning. *J. Intell. Manuf.* **2022**, *33*, 995–1005. https://doi.org/10.1007/s10845-020-01708-.
57. Lee, J.; Oh, S.J.; An, S.H.; Kim, W.D.; Kim, S.H. Machine learning-based design strategy for 3D printable bioink: elastic modulus and yield stress determine printability. *Biofabrication* **2020**, *12*, 035018. https://doi.org/10.1088/1758-5090/ab8707.
58. Peskin, C.S. The immersed boundary method. *Acta Numer.* **2002**, *11*, 479–517. https://doi.org/10.1017/S0962492902000077.
59. Osher, S.; Fedkiw, R.P. Level Set Methods: An Overview and Some Recent Results. *J. Comput. Phys.* **2001**, *169*, 463–502. https://doi.org/10.1006/jcph.2000.6636.
60. Gibou, F.; Fedkiw, R.; Osher, S. A review of level-set methods and some recent applications. *J. Comput. Phys.* **2018**, *353*, 82–109. https://doi.org/10.1016/j.jcp.2017.10.006.
61. Brackbill, J.; Kothe, D.; Zemach, C. A continuum method for modeling surface tension. *J. Comput. Phys.* **1992**, *100*, 335–354. https://doi.org/10.1016/0021-9991(92)90240-Y.
62. Cherizol, R.; Sain, M.; Tjong, J. Review of Non-Newtonian Mathematical Models for Rheological Characteristics of Viscoelastic Composites. *Green Sustain. Chem.* **2015**, *5*, 6–14. https://doi.org/10.4236/gsc.2015.51002.
63. Nair, K.; Gandhi, M.; Khalil, S.; Yan, K.C.; Marcolongo, M.; Barbee, K.; Sun, W. Characterization of cell viability during bioprinting processes. *Biotechnol. J.* **2009**, *4*, 1168–1177. https://doi.org/10.1002/biot.200900004.
64. Burova, I.; Wall, I.; Shipley, R.J. Mathematical and computational models for bone tissue engineering in bioreactor systems. *J. Tissue Eng.* **2019**, *10*, 2041731419827922. https://doi.org/10.1177/2041731419827922.

65. Flenner, E.; Janosi, L.; Barz, B.; Neagu, A.; Forgacs, G.; Kosztin, I. Kinetic Monte Carlo and cellular particle dynamics simulations of multicellular systems. *Phys. Rev. E* **2012**, *85*, 031907. https://doi.org/10.1103/PhysRevE.85.031907.
66. Kosztin, I.; Vunjak-Novakovic, G.; Forgacs, G. Colloquium: Modeling the dynamics of multicellular systems: Application to tissue engineering. *Rev. Mod. Phys.* **2012**, *84*, 1791–1805. https://doi.org/10.1103/RevModPhys.84.1791.
67. Yang, X.; Mironov, V.; Wang, Q. Modeling fusion of cellular aggregates in biofabrication using phase field theories. *J. Theor. Biol.* **2012**, *303*, 110–118. https://doi.org/10.1016/j.jtbi.2012.03.003.
68. Plan, E.L.C.M., VI; Yeomans, J.M.; Doostmohammadi, A. Active matter in a viscoelastic environment. *Phys. Rev. Fluids* **2020**, *5*, 023102. https://doi.org/10.1103/PhysRevFluids.5.023102.
69. Bonchev, D.; Thomas, S.; Apte, A.; Kier, L.B. Cellular automata modelling of biomolecular networks dynamics. *SAR QSAR Environ. Res.* **2010**, *21*, 77–102. https://doi.org/10.1080/10629360903568580.
70. A Supercomputing Journey Inspired by Curiosity—History of Cray Supercomputers. Available online: https://www.hpe.com/uk/en/compute/hpc/cray.html (accessed on 31 March 2024).
71. Reed, D.A.; Dongarra, J. Exascale Computing and Big Data. *Commun. ACM* **2015**, *58*, 56–68. https://doi.org/10.1145/2699414.
72. Edelman, A.. Chapter 6 of the selected Lecture Notes from Applied Parallel Computing (SMA 5505) course. Massachusetts Institute of Technology. Parallel Machines. Available online: https://dspace.mit.edu/bitstream/handle/1721.1/77902/18-337j-spring-2005/contents/lecture-notes/chapter_6.pdf (accessed on 31 March 2024).
73. Becker, D.J.; Sterling, T.; Savarese, D.; Dorband, J.E.; Ranawak, U.A.; Packer, C.V. BEOWULF: A Parallel Workstation For Scientific Computation. In Proceedings of the 24th International Conference on Parallel Processing, Urbana-Champain, IL, USA, 14–18 August 1995; CRC Press: Boca Raton, FL, USA, 1995; pp. 11–14.
74. Gara, A.; Blumrich, M.A.; Chen, D.; Chiu, G.L.; Coteus, P.; Giampapa, M.; Haring, R.A.; Heidelberger, P.; Hoenicke, D.; Kopcsay, G.V.; et al. Overview of the Blue Gene/L system architecture. *IBM J. Res. Dev.* **2005**, *49*, 195–212. https://doi.org/10.1147/rd.492.0195.
75. Tuomi, I. The Lives and Death of Moore's Law. *First Monday* **2002**, *7*, 11. https://doi.org/10.5210/fm.v7i11.1000.
76. Petitet, A.; Whaley, R.C.; Dongarra, J.J.; Cleary, A. *A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*; Innovative Computing Laboratory: Knoxville, TN, USA, 2000 Available online: https://icl.utk.edu/hpl/index.html (accessed on 31 March 2024).
77. Top 500—The List. Available online: https://www.top500.org/ (accessed on 31 March 2024).
78. Top 500 List—June 2022. Available online: https://www.top500.org/lists/top500/2022/06/ (accessed on 31 March 2024).
79. Geist, A.; Lucas, R. Major Computer Science Challenges At Exascale. *Int. J. High Perform. Comput. Appl.* **2009**, *23*, 427–436. https://doi.org/10.1177/1094342009347445.
80. Chen, W. The demands and challenges of Exascale computing: An interview with Zuoning Chen. *Natl. Sci. Rev.* **2016**, *3*, 64–67. https://doi.org/10.1093/nsr/nww012.
81. Kumar, V.; Gupta, A. Analyzing Scalability of Parallel Algorithms and Architectures. *J. Parallel Distrib. Comput.* **1994**, *22*, 379–391. https://doi.org/10.1006/jpdc.1994.1099.
82. Kwiatkowski, J. Evaluation of Parallel Programs by Measurement of Its Granularity. In Proceedings of the Parallel Processing and Applied Mathematics, Naleczow, Poland, 9–12 September 2001; Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; pp. 145–153. https://doi.org/10.1007/3-540-48086-2_16.
83. European Union. The EU Enters the Exascale Era with the Announcement of New Supercomputing Hosting Sites. Available online: https://digital-strategy.ec.europa.eu/en/news/eu-enters-Exascale-era-announcement-new-supercomputing-hosting-sites (accessed on 31 March 2024).
84. Matsuoka, S. To Exascale and Beyond. Japan RIKEN Center for Computational Science. Available online: https://www.riken.jp/en/news_pubs/research_news/rr/2019spring/ (accessed on 31 March 2024).
85. US Exascale Computing Project. The Office of Science (SC) and the National Nuclear Security Administration (NNSA) organizations of U.S. Department of Energy (DOE). Available online: https://www.Exascaleproject.org/ (accessed on 31 March 2024).
86. Barone, G.B.; Boccia, V.; Bottalico, D.; Carracciuolo, L. SCoPE@Scuola: (In)-formative Paths on Topics Related with High Performance, Parallel and Distributed Computing. In Proceedings of the Euro-Par 2017: Parallel Processing Workshops, Santiago de Compostela, Spain, 28–29 August 2017; Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 191–202. https://doi.org/10.1007/978-3-319-75178-8_16.
87. Yelick, K. Beyond Exascale Computing. Available online: https://www.isc-hpc.com/conference-keynote-2024.html (accessed on 13 May 2024).
88. Yelick, K. Beyond Exascale Computing. Available online: https://people.eecs.berkeley.edu/~yelick/talks/exascale/PostExascale-ICPP2023.pdf (accessed on 20 May 2024).
89. National Academies of Sciences, Engineering, and Medicine. *Charting a Path in a Shifting Technical and Geopolitical Landscape: Post-Exascale Computing for the National Nuclear Security Administration*; The National Academies Press: Washington, DC, USA, 2023. https://doi.org/10.17226/26916.

90. Dongarra, J.; Deelman, E.; Hey, T.; Matsuoka, S.; Sarakar, V.; Bell, G.; Foster, I.; Keyes, D.; Kranzlmueller, D.; Lucas, B.; et al. Can the United States Maintain Its Leadership in High-Performance Computing? A Report from the ASCAC Subcommittee on American Competitiveness and Innovation to the ASCR Office. Report, The Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee on American Competitiveness and Innovation: Washington, DC, USA, 2023. Available online: https://science.osti.gov/-/media/ascr/ascac/pdf/meetings/202306/ASCAC_Subcommittee_on_American_Competitiveness_202306.pdf (accessed on 20 May 2024).
91. Burini, D.; Chouhad, N. A multiscale view of nonlinear diffusion in biology: From cells to tissues. *Math. Model. Methods Appl. Sci.* **2019**, *29*, 791–823. https://doi.org/10.1142/S0218202519400062.
92. Fish, J.; Wagner, G.J.; Keten, S. Mesoscopic and multiscale modelling in materials. *Nat. Mater.* **2021**, *20*, 774–786. https://doi.org/10.1038/s41563-020-00913-0.
93. Amani, A.; Balcàzar, N.; Naseri, A.; Rigola, J. A numerical approach for non-Newtonian two-phase flows using a conservative level-set method. *Chem. Eng. J.* **2020**, *385*, 123896. https://doi.org/10.1016/j.cej.2019.123896.
94. Ahmadi, M.; Farsani, A. CFD simulation of non-Newtonian two-phase fluid flow through a channel with a cavity. *Therm. Sci.* **2018**, *2018*, 151. https://doi.org/10.2298/TSCI180102151A.
95. Irgens, F. *Rheology and Non-Newtonian Fluids*; Springer: Berlin/Heidelberg, Germany, 2014. https://doi.org/10.1007/978-3-319-01053-3.
96. Bush, J. Surface tension. In *New Trends in the Physics and Mechanics of Biological Systems: Lecture Notes of the Les Houches Summer School: July 2009*; Oxford University Press: Oxford, UK, 2011; Volume 92, pp. 27–64. https://doi.org/10.1093/acprof:oso/9780199605835.003.0002.
97. López-Herrera, J.; Popinet, S.; Castrejón-Pita, A. An Adaptive Solver for Viscoelastic Incompressible Two-phase Problems Applied to the Study of the Splashing of Weakly viscoelastic droplets. *J. -Non-Newton. Fluid Mech.* **2019**, *264*, 144–158. https://doi.org/10.1016/j.jnnfm.2018.10.012.
98. Wielage, K. Analysis of Non-Newtonian Two-Phase Flows. Ph.D. Thesis, University of Paderborn, Paderborn, Germany, 2005. Available online: https://d-nb.info/978191463/34 (accessed on 31 March 2024).
99. Keslerova, R.; Reznicek, H.; Padelek, T. Numerical modelling of generalized Newtonian fluids in bypass tube. *Adv. Comput. Math.* **2019**, *45*, 2047–2063. https://doi.org/10.1007/s10444-019-09684-y.
100. Lagrangian and Eulerian Specification of the Flow Field—Wikipedia Page. Available online: https://en.wikipedia.org/wiki/Lagrangian_and_Eulerian_specification_of_the_flow_field#cite_note-Batchelor-1 (accessed on 31 March 2024).
101. Bhalla, A.P.S.; Bale, R.; Griffith, B.E.; Patankar, N.A. A unified mathematical framework and an adaptive numerical method for fluid–structure interaction with rigid, deforming, and elastic bodies. *J. Comput. Phys.* **2013**, *250*, 446–476. https://doi.org/10.1016/j.jcp.2013.04.033.
102. Mittal, R.; Iaccarino, G. Immersed Boundary Methods. *Annu. Rev. Fluid Mech.* **2005**, *37*, 239–261. https://doi.org/10.1146/annurev.fluid.37.061903.175743.
103. Zhang, L.; Gerstenberger, A.; Wang, X.; Liu, W.K. Immersed Finite Element Method. *Comput. Methods Appl. Mech. Eng.* **2004**, *193*, 2051–2067. https://doi.org/10.1016/j.cma.2003.12.044.
104. Wang, X.; Liu, W.K. Extended immersed boundary method using FEM and RKPM. *Comput. Methods Appl. Mech. Eng.* **2004**, *193*, 1305–1321. https://doi.org/10.1016/j.cma.2003.12.024.
105. Balcazar, N.; Jofre, L.; Lehmkuhl, O.; Castro, J.; Rigola, J. A finite-volume/level-set method for simulating two-phase flows on unstructured grids. *Int. J. Multiph. Flow* **2014**, *64*, 55–72. https://doi.org/10.1016/j.ijmultiphaseflow.2014.04.008.
106. Olsson, E.; Kreiss, G. A conservative level set method for two phase flow. *J. Comput. Phys.* **2005**, *210*, 225–246. https://doi.org/10.1016/j.jcp.2005.04.007.
107. Olsson, E.; Kreiss, G.; Zahedi, S. A conservative level set method for two phase flow II. *J. Comput. Phys.* **2007**, *225*, 785–807. https://doi.org/10.1016/j.jcp.2006.12.027.
108. O'Dea, R.; Byrne, H.; Waters, S. Continuum Modelling of In Vitro Tissue Engineering: A Review. In *Computational Modeling in Tissue Engineering*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 229–266. https://doi.org/10.1007/8415_2012_140.
109. Extracellular Matrix—Wikipedia Page. Available online: https://en.wikipedia.org/wiki/Extracellular_matrix (accessed on 31 March 2024).
110. Dongarra, J.; Sullivan, F. Guest Editors' Introduction: The Top 10 Algorithms. *Comput. Sci. Eng.* **2000**, *2*, 22–23. https://doi.org/10.1109/MCISE.2000.814652.
111. Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H.; Teller, E. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.* **2004**, *21*, 1087–1092. https://doi.org/10.1063/1.1699114.
112. Richey, M. The Evolution of Markov Chain Monte Carlo Methods. *Am. Math. Mon.* **2010**, *117*, 383–413. https://doi.org/10.4169/000298910X485923.
113. Larget, B. Introduction to Markov Chain Monte Carlo Methods in Molecular Evolution. In *Statistical Methods in Molecular Evolution*; Springer: New York, NY, USA, 2005; pp. 45–62. https://doi.org/10.1007/0-387-27733-1_3.
114. Norris, J.R. *Markov Chains*; Cambridge Series in Statistical and Probabilistic Mathematics Series; Cambridge University Press: Cambridge, UK, 1997. https://doi.org/10.1017/CBO9780511810633.
115. Hastings, W.K. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika* **1970**, *57*, 97–109. https://doi.org/10.2307/2334940.

116. Barker, A.A. Monte Carlo Calculations of the Radial Distribution Functions for a Proton-electron Plasma. *Aust. J. Phys.* **1965**, *18*, 119–134. https://doi.org/10.1071/PH650119.
117. Landau, D.P.; Binder, K. *A Guide to Monte Carlo Simulations in Statistical Physics*, 4th ed.; Cambridge University Press: Cambridge, UK, 2015.
118. Newman, T.J.. Modeling Multicellular Structures Using the Subcellular Element Model. In *Single-Cell-Based Models in Biology and Medicine*; Birkhäuser: Basel, Switzerland, 2007; pp. 221–239. https://doi.org/10.1007/978-3-7643-8123-3_10.
119. van Kampen, N. Chapter IX—The Langevin Approach. In *Stochastic Processes in Physics and Chemistry*, 3rd ed.; North-Holland Personal Library; Elsevier: Amsterdam, The Netherlands, 2007; pp. 219–243. https://doi.org/10.1016/B978-044452965-7/50012-X.
120. Kier, L.B.; Seybold, P.G.; Cheng, C.-K. *Modeling Chemical Systems Using Cellular Automata*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2005. https://doi.org/10.1007/1-4020-3690-6.
121. von Neumann, J. *Theory of Self-Reproducing Automata*; University of Illionois Press: Champain, IL, USA, 1966.
122. Ulam, S. Random processes and transformations. In Proceedings of the International Congress of Mathematicians, Cambridge, MA, USA, 30 August–6 September 1950; American Mathematical Society: Providence, RI, USA, 1952; Volume 2, pp. 264–275.
123. Wolfram, S. Cellular Automata. *Los Alamos Sci.* **1983**, *9*, 2–27.
124. Menshutina, N.V.; Kolnoochenko, A.V.; Lebedev, E.A. Cellular Automata in Chemistry and Chemical Engineering. *Annu. Rev. Chem. Biomol. Eng.* **2020**, *11*, 87–108. https://doi.org/10.1146/annurev-chembioeng-093019-075250.
125. Reddy, J.N.; Gartling, D.K. *The Finite Element Method in Heat Transfer and Fluid Dynamics: Third Edition*; CRC Press: Boca Raton, FL, USA, 2010. https://doi.org/10.1201/9781439882573.
126. Alves, M.A.; Oliveira, P.J.; Pinho, F.T.. Numerical Methods for Viscoelastic Fluid Flows. *Annu. Rev. Fluid Mech.* **2021**, *53*, 509–541. https://doi.org/10.1146/annurev-fluid-010719-060107.
127. Chung, T. *Computational Fluid Dynamics for Engineers*; Cambridge University Press: Cambridge, UK, 2002. https://doi.org/10.1017/CBO9780511606205.
128. Fontes, E. FEM vs. FVM. Available online: https://www.comsol.com/blogs/fem-vs-fvm (accessed on 31 March 2024).
129. Sjodin, B. What's The Difference between FEM, FDM and FVM? Available online: https://www.machinedesign.com/3d-printing-cad/fea-and-simulation/article/21832072/whats-the-difference-between-fem-fdm-and-fvm (accessed on 31 March 2024).
130. Hatami, M. Chapter 2—Weighted Residual Methods Principles and Modifications. In *Weighted Residual Methods*; Hatami, M., Ed.; Academic Press: Cambridge, MA, USA, 2018; pp. 83–152. https://doi.org/10.1016/B978-0-12-813218-0.00002-9.
131. Baaijens, P.T.F. Mixed Finite Element Methods for viscoelastic flow analysis: A review. *J. -Non-Newton. Fluid Mech.* **1998**, *79*, 361–385. https://doi.org/10.1016/S0377-0257(98)00122-0.
132. Marchandise, E.; Geuzaine, P.; Chevaugeon, N.; Remacle, J.F. A stabilized Finite Element Method using a discontinuous level set approach for the computation of bubble dynamics. *J. Comput. Phys.* **2007**, *225*, 949–974. https://doi.org/10.1016/j.jcp.2007.01.005.
133. Chen, T.; Minev, P.D.; Nandakumar, K. A projection scheme for incompressible multiphase flow using adaptive Eulerian grid. *Int. J. Numer. Methods Fluids* **2004**, *45*, 1–19. https://doi.org/10.1002/fld.591.
134. Pillapakkam, S.; Singh, P. A Level-Set Method for Computing Solutions to Viscoelastic Two-Phase Flow. *J. Comput. Phys.* **2001**, *174*, 552–578. https://doi.org/10.1006/jcph.2001.6927.
135. Larson, M.G.; Bengzon, F. *The Finite Element Method: Theory, Implementation, and Applications*; Springer: Berlin/Heidelberg, Germany, 2013. https://doi.org/10.1007/978-3-642-33287-6.
136. Reddy, J.N. *An Introduction to the Finite Element Method*, 2nd ed.; McGraw-Hill Higher Education: Boca Raton, FL, USA, 1993. ISBN: 9780070513556.
137. Gerald, C.; Wheatley, P. *Applied Numerical Analysis*; Featured Titles for Numerical Analysis; Pearson/Addison-Wesley: London, UK, 2004.
138. Eymard, R.; Gallouët, T.; Herbin, R. Finite Volume Methods. In *Handbook of Numerical Analysis*; Lions, J.L., Ciarlet, P., Eds.; Elsevier: Amsterdam, The Netherlands, 2000; Volume 7, pp. 713–1020. https://doi.org/10.1016/S1570-8659(00)07005-8.
139. Chen, L. Finite Volume Methods. In *Lecture Notes for the Course Math 226: Computational PDEs*; University of California: Irvine, CA, USA. Available online: https://www.math.uci.edu/~chenlong/lectures.html (accessed on 31 March 2024).
140. Cai, Z. On the finite Volume element methods. *Numer. Math.* **1990**, *58*, 713–735. https://doi.org/10.1007/BF01385651.
141. Mazumder, S. Chapter 6—The Finite Volume Method (FVM). In *Numerical Methods for Partial Differential Equations*; Mazumder, S., Ed.; Academic Press: Cambridge, MA, USA, 2016; pp. 277–338. https://doi.org/10.1016/B978-0-12-849894-1.00006-8.
142. Kumar, S. A Time Integration Scheme for Dynamic Problems. Ph.D. Thesis, Department of Mechanical Engineering, Indian Institute of Technology Guwahati, Guwahati, India, 2015. https://doi.org/10.13140/RG.2.2.10053.93924.
143. Geradin, M.; Rixen, D. *Mechanical Vibrations: Theory and Application to Structural Dynamics*, 3rd ed.; John Wiley & Sons: Hoboken, NJ, USA, 2015.
144. Higham, N.J. *Accuracy and Stability of Numerical Algorithms*; Society of Industrial and Applied Mathematics: Philadelphia, PA, USA, 1996.
145. Wriggers, P. *Nonlinear Finite Element Methods*; Springer: Berlin/Heidelberg, Germany, 2008. https://doi.org/10.1007/978-3-540-71001-1.

146. Duff, I.S.; Erisman, A.M.; Reid, J.K. *Direct Methods for Sparse Matrices*, 2nd ed.; Oxford University Press: Oxford, UK, 2017. https://doi.org/10.1093/acprof:oso/9780198508380.001.0001.
147. Golub, G.H.; Van Loan, C.F. *Matrix Computations*, 3rd ed.; The Johns Hopkins University Press: Baltimore, MD, USA, 1996.
148. Duff, I.S.; Reid, J.K. The Multifrontal Solution of Indefinite Sparse Symmetric Linear. *ACM Trans. Math. Softw.* **1983**, *9*, 302–325. https://doi.org/10.1145/356044.356047.
149. Amestoy, P.; Buttari, A.; Duff, I.; Guermouche, A.; L'Excellent, J.Y.; Uçar, B., Multifrontal Method. In *Encyclopedia of Parallel Computing*; Springer: Boston, MA, USA, 2011; pp. 1209–1216. https://doi.org/10.1007/978-0-387-09766-4_86.
150. Lawson, C.L.; Hanson, R.J.; Kincaid, D.R.; Krogh, F.T. Basic Linear Algebra Subprograms for Fortran Usage. *ACM Trans. Math. Softw.* **1979**, *5*, 308–323. https://doi.org/10.1145/355841.355847.
151. van der Vorst, H.A.. Krylov subspace iteration. *Comput. Sci. Eng.* **2000**, *2*, 32–37. https://doi.org/10.1109/5992.814655.
152. Saad, Y. *Iterative Methods for Sparse Linear Systems*, 2nd ed.; Other Titles in Applied Mathematics, Society of Industrial and Applied Mathematics: Philadelphia, PA, USA, 2003. https://doi.org/10.1137/1.9780898718003.
153. Dubrulle, A.A.; Weisstein, E.W. Hessenberg Matrix. From MathWorld—A Wolfram Web Resource. Available online: https://mathworld.wolfram.com/HessenbergMatrix.html (accessed on 31 March 2024).
154. Stüben, K. A review of algebraic Multigrid. *J. Comput. Appl. Math.* **2001**, *128*, 281–309. https://doi.org/10.1016/S0377-0427(00)00516-1.
155. Benzi, M.; Golub, G.H.; Liesen, J. Numerical Solution of Saddle Point Problems. *Acta Numer.* **2005**, *14*, 1–137. https://doi.org/10.1017/S0962492904000212.
156. Dongarra, J.; Hittinger, J.; Bell, J.; Chacon, L.; Falgout, R.; Heroux, M.; Hovland, P.; Ng, E.; Webster, C.; Wild, S. *Applied Mathematics Research for Exascale Computing*; Technical Report LLNL-TR-651000; Lawrence Livermore National Laboratory (LLNL): Livermore, CA, USA; 2014. https://doi.org/10.2172/1149042.
157. Ang, J.; Evans, K.; Geist, A.; Heroux, M.; Hovland, P.D.; Marques, O.; Curfman McInnes, L.; Ng, E.G.; Wild, S.M. *Report on the Workshop on Extreme-Scale Solvers: Transition to Future Architectures*; Department of Energy, ASCR: Washington, DC, USA, 2012. Available online: https://science.osti.gov/-/media/ascr/pdf/program-documents/docs/reportExtremeScaleSolvers2012.pdf (accessed on 31 March 2024).
158. Khaleel, M.A. *Scientific Grand Challenges: Crosscutting Technologies for Computing at the Exascale—February 2–4, 2010, Washington, D.C.*; Technical Report; Pacific Northwest National Laboratory (PNNL): Richland, WA, USA, 2011. https://doi.org/10.2172/1008243.
159. Liu, W.K.; Park, H.S.; Karpov, E.G.; Farrell, D. Bridging Scale Method and Its Applications. In *Meshfree Methods for Partial Differential Equations III*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 185–205. https://doi.org/10.1007/978-3-540-46222-4_11.
160. Park, H.S.; Liu, W.K. An introduction and tutorial on multiple-scale analysis in solids. *Comput. Methods Appl. Mech. Eng.* **2004**, *193*, 1733–1772. https://doi.org/10.1016/j.cma.2003.12.054.
161. Chacon, L.; Chen, G.; Knoll, D.A.; Newman, C.; Park, H.; Taitano, W.; Willert, J.A.; Womeldorff, G. Multiscale high-order/low-order (HOLO) algorithms and applications. *J. Comput. Phys.* **2017**, *330*, 21–45. https://doi.org/10.1016/j.jcp.2016.10.069.
162. Keyes, D.E.; McInnes, L.C.; Woodward, C.; Gropp, W.; Myra, E.; Pernice, M.; Bell, J.; Brown, J.; Clo, A.; Connors, J.; et al. Multiphysics simulations: Challenges and opportunities. *Int. J. High Perform. Comput. Appl.* **2013**, *27*, 4–83. https://doi.org/10.1177/1094342012468181.
163. HPC Wiki—Scaling. Available online: https://hpc-wiki.info/hpc/Scaling#Strong_or_Weak_Scaling (accessed on 1 March 2024).
164. Keyes, D.E. Exaflop/s: The why and the how. *Comptes Rendus MéCanique* **2011**, *339*, 70–77. https://doi.org/10.1016/j.crme.2010.11.002.
165. Brown, P.N.; Hindmarsh, A.C. Matrix-Free Methods for Stiff Systems of ODE's. *SIAM J. Numer. Anal.* **1986**, *23*, 610–638. https://doi.org/10.1137/0723039.
166. Davies, T.; Karlsson, C.; Liu, H.; Ding, C.; Chen, Z. High performance linpack benchmark: a fault tolerant implementation without checkpointing. In Proceedings of the International Conference on Supercomputing, Tucson, AZ, USA, 31 May–4 June 2011; pp. 162–171. https://doi.org/10.1145/1995896.1995923.
167. Brezinski, C.; Redivo Zaglia, M. Studies in Computational Mathematics 2. In *Extrapolation Methods Theory and Practice*; Studies in Computational Mathematics Series; Elsevier: Amsterdam, The Netherlands, 1991; Volume 2. https://doi.org/10.1016/B978-0-444-88814-3.50001-5.
168. Hairer, E.; Norsett, S.P.; Wanner, G. *Solving ordinary differential equations I: Nonstiff Problems*; Springer: Berlin/Heidelberg, Germany, 1993. https://doi.org/10.1007/978-3-540-78862-1.
169. Althaus, K. Theory and Implementation of the Adaptive Explicit Midpoint Rule Including Order and Stepsize Control. Bachelor's Thesis, Department of Mathematics, Technische Universitat Munchen, Munchen, Germany, 2018. Available online: https://github.com/AlthausKonstantin/Extrapolation/blob/master/Bachelor%20Theseis.pdf (accessed on 1 March 2024).
170. Constantinescu, E.M.; Sandu, A. Extrapolated Implicit-Explicit Time Stepping. *SIAM J. Sci. Comput.* **2010**, *31*, 4452–4477. https://doi.org/10.1137/080732833.
171. Constantinescu, E.M.; Sandu, A. *Achieving Very High Order for Implicit Explicit Time Stepping: Extrapolation Methods*; Report Preprint ANL/ANL/MCS-TM-306; Argonne National Laboratory: Lemont, IL, USA, 2009. Available online: https://www.mcs.anl.gov/uploads/cels/papers/TM-306.pdf (accessed on 1 March 2024).

172. Gander, M.J. 50 Years of Time Parallel Time Integration. In Proceedings of the Multiple Shooting and Time Domain Decomposition Methods, Heidelberg, Germany, 6–8 May 2013; Carraro, T., Geiger, M., Körkel, S., Rannacher, R., Eds.; Springer: Cham, Switzerland, 2015; pp. 69–113.
173. Gander, M.J.; Lunet, T.; Ruprecht, D.; Speck, R. A Unified Analysis Framework for Iterative Parallel-in-Time Algorithms. *SIAM J. Sci. Comput.* **2023**, *45*, A2275–A2303. https://doi.org/10.1137/22M1487163.
174. Brune, P.R.; Knepley, M.G.; Smith, B.F.; Tu, X. Composing Scalable Nonlinear Algebraic Solvers. *SIAM Rev.* **2015**, *57*, 535–565. https://doi.org/10.1137/130936725.
175. Balay, S.; Gropp, W.D.; McInnes, L.C.; Smith, B.F. Efficient Management of Parallelism in Object Oriented Numerical Software Libraries. In *Modern Software Tools in Scientific Computing*; Arge, E., Bruaset, A.M., Langtangen, H.P., Eds.; Birkhäuser Press: Basel, Switzerland, 1997, pp. 163–202.
176. Smith, B.; McInnes, L.C.; Constantinescu, E.; Adams, M.; Balay, S.; Brown, J.; Knepley, M.G.; .; Zhang, H. PETSc's Software Strategy for the Design Space of Composable Extreme-Scale Solvers. In Proceedings of the DOE Exascale Research Conference, Portland, OR, USA, 16–18 April 2012; Report Preprint ANL/MCS-P2059-0312; Argonne National Laboratory: Lemont, IL, USA, 2012 Available online: http://www.mcs.anl.gov/uploads/cels/papers/P2059-0312.pdf (accessed on 31 March 2024).
177. Pawlowski, R.P.; Shadid, J.N.; Simonis, J.P.; Walker, H.F. Globalization Techniques for Newton–Krylov Methods and Applications to the Fully Coupled Solution of the Navier–Stokes Equations. *SIAM Rev.* **2006**, *48*, 700–721. https://doi.org/10.1137/S0036144504443511.
178. Dolean, V.; Gander, M.J.; Kheriji, W.; Kwok, F.; Masson, R. Nonlinear Preconditioning: How to Use a Nonlinear Schwarz Method to Precondition Newton's Method. *SIAM J. Sci. Comput.* **2016**, *38*, A3357–A3380. https://doi.org/10.1137/15M102887X.
179. Carson, E. Communication-Avoiding Krylov Subspace Methods in Theory and Practice. Technical Report No. UCB/EECS-2015-179. Ph.D. Thesis, Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA, USA, 2015. Available online: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-179.pdf (accessed on 31 March 2024).
180. Anzt, H.; Boman, E.; Falgout, R.; Ghysels, P.; Heroux, M.; Li, X.; Curfman McInnes, L.; Tran Mills, R.; Rajamanickam, S.; Rupp, K.; et al. Preparing sparse solvers for Exascale computing. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* **2020**, *378*, 20190053. https://doi.org/10.1098/rsta.2019.0053.
181. Abdelfattah, A.; Anzt, H.; Ayala, A.; Boman, E; Carson, E.; Cayrols, S.; Cojean, T.; Dongarra, J.; Falgout, R.; Gates, M.; et al. *Advances in Mixed Precision Algorithms: 2021 Edition*; Technical Report LLNL-TR-825909; Lawrence Livermore National Lab. (LLNL): Livermore, CA, USA, 2021. https://doi.org/10.2172/1814677.
182. Hoemmen, M. Communication-avoiding Krylov Subspace Methods. Technical Report No. UCB/EECS-2010-37. Ph.D. Thesis, Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA, USA, 2010. Available online: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-37.pdf (accessed on 31 March 2024).
183. Bai, Z.; Hu, D.; Reichel, L. A Newton basis GMRES implementation. *IMA J. Numer. Anal.* **1994**, *14*, 563–581. https://doi.org/10.1093/imanum/14.4.563.
184. Greengard, L.; Rokhlin, V. A fast algorithm for particle simulations. *J. Comput. Phys.* **1987**, *73*, 325–348. https://doi.org/10.1016/0021-9991(87)90140-9.
185. Cipra, B.A. The Best of the 20th Century: Editors Name Top 10 Algorithms. *SIAM News* **2000**, *33*, 1–2.
186. Beatson, R.; Greengard, L. A Short Course on Fast Multipole Methods. Available online: http://math.nyu.edu/faculty/greengar/shortcourse_fmm.pdf (accessed on 31 March 2024).
187. Martinsson, P.G. Fast Multipole Methods. In *Encyclopedia of Applied and Computational Mathematics*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 498–508. https://doi.org/10.1007/978-3-540-70529-1_448.
188. Fenn, M.; Steidl, G. FMM and H-matrices: A Short Introduction to the Basic Idea. 2002. Available online: https://madoc.bib.uni-mannheim.de/744/ (accessed on 31 March 2024).
189. Hackbusch, W.; Grasedyck, L.; Börm, S. An introduction to hierarchical matrices. *Math. Bohem.* **2002**, *127*, 229–241. https://doi.org/10.21136/MB.2002.134156.
190. Weisstein, E.W. Binary Tree. From MathWorld—A Wolfram Web Resource. Available online: https://mathworld.wolfram.com/BinaryTree.html (accessed on 31 March 2024).
191. Li, Y.; Mascagni, M.. Grid-Based Monte Carlo Application. In Proceedings of the Grid Computing—GRID 2002, Baltimore, MD, USA, 18 November 2002; Springer: Berlin/Heidelberg, Germany, 2002; pp. 13–24. https://doi.org/10.1007/3-540-36133-2_2.
192. Rosenthal, Jeffrey S. Parallel computing and Monte Carlo algorithms. *Far East J. Theor. Stat.* **2000**, *4*, 207–236. Available: http://probability.ca/jeff/ftpdir/para.pdf.
193. Advanced Materials 2030 Initiative. *Materials 2030 Manifesto—A Systemic Approach of Advanced Materials for Prosperity—A 2030 Perspective*; EU Publications Office: Brussels, Belgium, 2022. Available online: https://research-and-innovation.ec.europa.eu/system/files/2022-02/advanced-materials-2030-manifesto.pdf (accessed on 31 March 2024).
194. Abbadessa, A.; Landín, M.; Oude Blenke, E.; Hennink, W.E.; Vermonden, T. Two-component thermosensitive hydrogels: Phase separation affecting rheological behavior. *Eur. Polym. J.* **2017**, *92*, 13–26. https://doi.org/10.1016/j.eurpolymj.2017.04.029.
195. Mobarak, M.H.; Mimona, M.A.; Islam, M.A.; Hossain, N.; Zohura, F.T.; Imtiaz, I.; Rimon, M.I.H. Scope of machine learning in materials research—A review. *Appl. Surf. Sci. Adv.* **2023**, *18*, 100523. https://doi.org/10.1016/j.apsadv.2023.100523.

196. Yamanaka, A.; Maeda, Y.; Sasaki, K. Ensemble Kalman filter-based data assimilation for three-dimensional multi-phase-field model: Estimation of anisotropic grain boundary properties. *Mater. Des.* **2019**, *165*, 107577. https://doi.org/10.1016/j.matdes.2018.107577.
197. Evensen, G.; Vossepoel, F.C.; Van Leeuwen, P.J. *Data Assimilation Fundamentals: A Unified Formulation of the State and Parameter Estimation Problem*; Springer: Cham, Switzerland, 2022. https://doi.org/10.1007/978-3-030-96709-3.
198. Carlberg, K.; Brencher, L.; Haasdonk, B.; Barth, A. Data-Driven Time Parallelism via Forecasting. *SIAM J. Sci. Comput.* **2019**, *41*, B466–B496. https://doi.org/10.1137/18M1174362.
199. An, J.; Chua, C.K.; Mironov, V. A perspective on 4D bioprinting. *Int. J. Bioprinting* **2016**, *2*, 3–5. https://doi.org/10.18063/IJB.2016.01.003.