HIGH PERFORMANCE SIMULATIONS OF PHASE-SEPARATING ELECTRODE PARTICLES IN CUSTOM GEOMETRIES*

SAMUEL DEGNAN-MORGENSTERN†

Abstract. By leveraging tools throughout the Julia scientific machine learning ecosystem, this work establishes a robust, high-perfomance simulation platform for the Cahn-Hilliard phase field model of phase separating electrode materials. Specifically, this work address common numerical challenges with simulating the stiff, nonlinear Cahn-Hilliard partial differential equation in custom geometries by exploiting Julia's high performance scientific computing suite. The paper outlines the theoretical background of battery dynamics in custom geometries, details of the simulation platform including support for GPU parallelization and applications to PDE-constrained optimization, a performance evaluation, and future improvements and applications. This work provides a foundation for future studies on learning constitutive relationships in batteries at the population scale.

Link to Github Repository: https://github.com/stmorgenstern/CahnHilliardSBM.jl

Key words. Nonlinear Dynamics, Cahn–Hilliard Model, Phase Separation, Smoothed Boundary Method

1. Introduction.

In recent literature [3, 25, 9, 22], nonequilibrium thermodynamics has emerged as the true first-principles foundation for modeling dynamics in many common lithium-ion battery materials. The dynamics of these systems are driven by the exploration of a non-convex free energy landscape that drives heterogeneous transport processes. Of these models, the Cahn-Hilliard continuum model [6] is the most general for describing the complex dynamics of lithium-ion battery materials that includes heterogeneous reaction kinetics, phase separation processes, and diffusion. Despite its generality, the Cahn-Hilliard model is a fourth-order, nonlinear partial differential equation (PDE), and it poses many challenges for numerical modeling including stiffness and sharp spatial gradients. For this reason, the field of battery simulation has been slow to incorporate the Cahn-Hilliard single particle model into existing battery simulation toolkits [4, 23].

Beyond battery simulation software, the Cahn-Hilliard model has potential applications in bridging the gap between theorists and experimentalists in the field of battery science. Past studies by Zhao et al. [26, 27] have leveraged similar phase field models to characterize the governing physics in battery systems directly from image data. These studies have relied on PDE-constrained optimization to learn functional and or constitutive relationships within PDE-based phase field models. These studies have primarily utilized the numerically simpler but less physically accurate Allen-Cahn model [2] from nonequilibrium thermodynamics. Furthermore, these studies used the smoothed boundary method [24] (discussed at length in subsection 2.2) to simulate the Allen-Cahn equation in custom geometries at a particle by particle level. To push the PDE inversion analysis further, higher fidelity simulations are needed to enable Cahn-Hilliard simulations at a particle population scale while still retaining the ability to estimate constitutive relationships in custom domains.

In this work, we will work towards achieving high performance and high fidelity simulations of the Cahn-Hilliard equation in custom domains using the smoothed

*Submitted to the 18.337 Teaching Team 5/15/2023

Funding: This work was funded by the Massachusetts Institute of Technology Presidential Fellowship

†Massachusetts Institute of Technology Department of Chemical Engineering, Cambridge, MA (stm16109@mit.edu.

boundary method. We propose to leverage tools throughout the Julia scientific machine learning (SciML) ecosystem to build a high performance simulation platform for the Cahn-Hilliard phase field PDE. In section 2, we will review the relevant theoretical background to the Cahn-Hilliard model and the smoothed boundary method for phase field models. In section 3, we will provide details on the simulation platform including the development of tools for creating custom smoothed boundary method domains, high performance Cahn-Hilliard PDE simulations, within method GPU parallelization, and the applications to parameter estimation. In section 4, we will discuss the performance of the platform, including a review of the best possible implementation and solvers. Lastly, in section 5, we will discuss future improvements and applications of this platform.

2. Background.

2.1. Physical Model. In a landmark 2013 paper [3], Bazant established the general theoretical framework for electrochemical transport based on nonequilibrium chemical thermodynamics. Using this framework, the governing equations for a model system of a depth-averaged 2D phase separating particle will be introduced.

For an infinitesimal control volume, the conservation of species can be expressed as follows [7]:

$$\frac{\partial c_i}{dt} = -\nabla \cdot F_i + R_i, \tag{2.1}$$

where c_i is the concentration of species i, F_i is the net flux of species i in the control volume, and R_i is the volumetric reaction term for species i in the control volume. For electrochemical systems, volumetric reactions can be neglected because electrochemical reactions predominantly occur at system interfaces. To formulate the species flux, we must take a detour into nonequilibrium thermodynamics. For a system that obeys linear irreversible thermodynamics, we can postulate a Gibbs free energy functional G and then state the flux as:

$$G[\{c_i\}] = \int_V g(c_i)\, dV + \oint_A \gamma(c_i)\, dA \tag{2.2}$$

$$F_i = -D(c_i) \cdot \nabla \left(\frac{\delta G}{\delta c_i} \right) = -D(c) \cdot \nabla \mu_i \tag{2.3}$$

$$\mu_i = \frac{\delta G}{\delta c_i} = \frac{\partial g}{\partial c_i} - \nabla \cdot \frac{\partial g}{\partial \nabla c_i} \tag{2.4}$$

where $D(c_i)$ is the concentration dependent diffusivity and μ_i is the chemical potential of the system. For this system, we will adopt the simple regular solution Cahn-Hilliard gradient expansion as a model for the homogenous free energy. In this framework we obtain the following chemical potential and flux:

$$\mu(\tilde{c}) = \log\left(\frac{\tilde{c}}{1-\tilde{c}} \right) + \Omega\,(1 - 2\tilde{c}) - \kappa \nabla^2 \tilde{c} \tag{2.5}$$

$$F = -D_0 \tilde{c}(1-\tilde{c}) \nabla \mu\,(\tilde{c}) \tag{2.6}$$

where $\tilde{c}$ is the normalized species concentration, Ω is the enthalpy of mixing parameter, κ is the gradient energy penalty coefficient, and the chemical diffusivity $D(\tilde{c}) = D_0 \tilde{c}(1-\tilde{c})$ is a thermodynamically consistent diffusivity. Substituting the

flux term in 2.6 into the species conservation equation in 2.1 we obtain the following governing equation for this system:

$$\frac{\partial \tilde{c}}{dt} = \nabla \cdot \left(D_0 \tilde{c}(1-\tilde{c}) \nabla \left(\log\left(\frac{\tilde{c}}{1-\tilde{c}}\right) + \Omega\,(1-2\tilde{c}) - \kappa \nabla^2 \tilde{c} \right) \right). \tag{2.7}$$

The typical boundary conditions for this system are decided by the relevant reaction kinetics and surface wetting effects. However, to simplify the numerical implementation, we will only consider no-flux Neumann boundary conditions on species concentration and flux. The resulting equation is a fourth-order, nonlinear, and stiff PDE that will be simulated in a 2D Cartesian coordinate system.

2.2. Smoothed Boundary Method. In this section, we will develop the mathematical framework for the smoothed boundary method in the context of the Cahn-Hilliard phase field model, as introduced by Yu et al. [24]. A domain parameter ψ (also referred to as a mask) is introduced as followed:

$$\psi(x,y) = \begin{cases} 1 & \text{if } (x,y) \in \mathbf{S} \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

where $\mathbf{S}$ is the domain of interest. For systems with Neumann boundary conditions, we define the following identity for a function $F : \mathbb{R}^n \mapsto \mathbb{R}$:

$$\psi \nabla^2 F = \nabla \cdot (\psi \nabla F) - \nabla \psi \cdot \nabla F, \tag{2.9}$$

where, we can re-express $\nabla\psi \cdot \nabla$ in terms of the boundary conditions by recognizing $\vec{n} = \frac{\nabla \psi}{|\nabla \psi|}$ where $\vec{n}$ is the unit surface normal. Following this, we can state the following identity for the surface boundary condition:

$$\frac{\partial F}{\partial n} = \nabla F \cdot \vec{n} = \nabla F \cdot \frac{\nabla \psi}{|\nabla \psi|}, \tag{2.10}$$

allowing us to conclude, $|\nabla\psi| \frac{\partial F}{\partial n} = \nabla F \cdot \nabla \psi$. We will now apply this technique to the Cahn-Hilliard model. We begin by multiplying both sides of the Cahn-Hilliard equation with the domain parameter and applying a series of vector calculus identities:

$$\begin{aligned} \psi \frac{\partial \tilde{c}}{dt} &= \psi \nabla \cdot (D(\tilde{c}) \cdot \nabla \mu) \\ &= \nabla \cdot (\psi D(c) \nabla \mu) - \nabla \psi \cdot (D(\tilde{c}) \nabla \mu). \end{aligned} \tag{2.11}$$

Now we can revisit a simplified version of the free energy functional introduced in 2.3 to obtain an expression for the chemical potential μ.

$$\begin{aligned} \psi \cdot \mu &= \psi \cdot \left(\log\left(\frac{c}{1-c}\right) + \Omega\,(1-2c) \right) - \psi \kappa \nabla^2 c \\ &= \psi \cdot \left(\log\left(\frac{\tilde{c}}{1-\tilde{c}}\right) + \Omega\,(1-2\tilde{c}) \right) - \kappa \nabla \cdot (\psi \nabla \tilde{c}) - \nabla \psi \cdot \nabla \tilde{c}. \end{aligned} \tag{2.12}$$

Combining equations 2.11 and 2.12 and applying the assumption of a no wetting boundary condition($0 = \frac{\partial c}{\partial n}$) and no flux boundary condition ($0 = \frac{\partial \mu}{\partial n}$) we obtain the

final expression for the Cahn-Hilliard model using the smoothed boundary method:

$$\begin{aligned} \frac{\partial \tilde{c}}{dt} &= \frac{1}{\psi}\nabla \cdot (\psi D(\tilde{c})\nabla \mu) \\ &= \frac{D(\tilde{c})}{\psi}\nabla\psi \cdot \nabla\mu + \frac{\partial D}{\partial \tilde{c}}\nabla\tilde{c}\cdot\nabla\mu + D(\tilde{c})\nabla^2\mu \\ \mu &= \log\left(\frac{\tilde{c}}{1-\tilde{c}}\right) + \Omega\left(1-2\tilde{c}\right) - \kappa\nabla\cdot(\psi\nabla\tilde{c}) \\ &= \log\left(\frac{\tilde{c}}{1-\tilde{c}}\right) + \Omega\left(1-2\tilde{c}\right) - \kappa\left(\frac{\nabla\psi\cdot\nabla\tilde{c}}{\psi} + \nabla^2\tilde{c}\right), \end{aligned} \tag{2.13}$$

which we will develop the software implementation to simulate in subsection 3.2.

3. Methods.

3.1. Mask Generation. In this section, we will discuss the process of generating a custom mask that serves as the discrete set of the domain parameter, ψ, which we will use in the numerical simulations. The authors of the original smoothed boundary method paper recommend fitting a domain parameter function using the tanh function or level set methods [24]. However, we consider an alternative approach in this work based on image segmentation and image processing to identify the domain and create a diffuse interface. An overview of the workflow is provided in Figure 1.

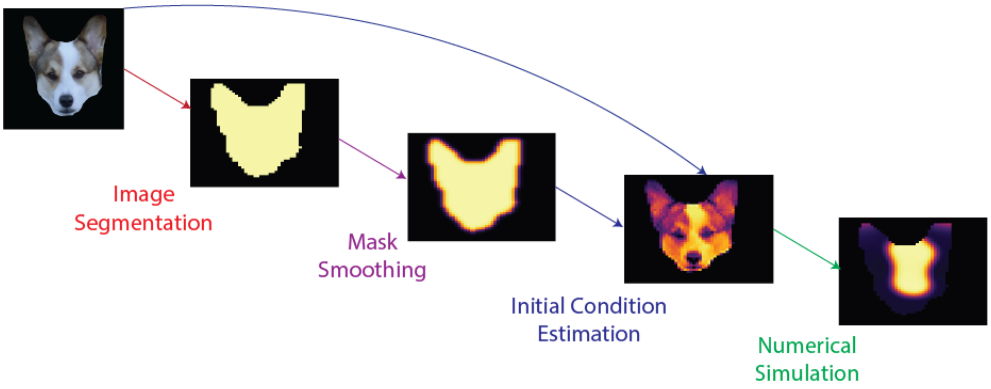

FIG. 1. *An example workflow is provided here for processing an image to generate a mask, estimate the initial condition, and simulate the Cahn-Hilliard equation on the custom domain. The image is first converted to grayscale, and the domain is determined by binarizing all the pixels above a predetermined threshold. The smoothing process is then run on the binarized mask to create a diffuse interface. The binary mask is then multiplied elementwise by the grayscale image to create the initial condition. Then initial condition is used for simulation according to the model described in equation 2.7.*

Using the Images.jl package in Julia [5], a PNG image is loaded and converted into grayscale. Each pixel is then determined to be on the domain by evaluating whether it is above a predetermined threshold. For more complex images, it is possible to use more advanced image segmentation methods to determine the working domain. To generate a diffuse interface on the mask, a short smoothing procedure is used by solving the reaction–diffusion equation on the entire domain with no flux boundary

conditions. Specifically, the following equation is solved:

$$
\begin{aligned}
\frac{\partial \psi}{\partial t} &= \zeta^2 \nabla^2 \psi + R(\psi) \\
R(\psi) &= 4(2\psi - 1)(\psi - 1)\psi \\
\left.\frac{\partial \psi(x,y,t)}{\partial x}\right|_{x=0} &= 0 \\
\left.\frac{\partial \psi(x,y,t)}{\partial x}\right|_{x=1} &= 0 \\
\left.\frac{\partial \psi(x,y,t)}{\partial x}\right|_{y=0} &= 0 \\
\left.\frac{\partial \psi(x,y,t)}{\partial x}\right|_{y=1} &= 0 \\
\psi(x,y,t=0) &= \psi_0(x,y),
\end{aligned}
\tag{3.1}
$$

where ζ is the dimensionless group controlling the ratio between the rate of "diffusion" to rate of "reaction" (set to ζ=0.04 for all simulations), and $R(\psi)$ is the "reaction" term. The simulation is run on the dimensionless domain of $(x, y) \in [0, 1] \times [0, 1]$ between, $t \in [0, t_f]$ where t_f is adjusted to control the width of the boundary layer. This mask can then be used in the simulations of equation 2.7, as described in 3.2. For some applications, it may be necessary to estimate the initial condition for the Cahn-Hilliard simulations directly from the image. One approach to do this is to take the grayscale values of the processed image. For more complex RGB image data, it is possible to estimate the concentration from a colorimetric calibration curve. For further discussion on this method, interested readers should consult Maire et al. [17].

3.2. Numerical Simulation. In this section, we will discuss the development of the high performance Cahn-Hilliard simulation platform. The development of the simulation platform required developing an optimized function to compute the right-hand side (RHS) of equation 2.7, as well as a procedure for exploiting the sparse structure of the function to use in a well-selected ODE solver.

The Cahn-Hilliard equation is a fourth order parabolic PDE [18]; for this reason, we use the method of lines scheme with second order accuracy centered finite differencing to handle the spatial discretization. The potential advantages of the alternative finite volume discretization scheme are discussed in section 5. The RHS function was developed in three implementations: an elementwise direct computation method, an allocating matrix multiplication stencil, and a cached non-allocating matrix multiplication stencil. An overview of the directed acyclic graph (DAG) underlying all three of the computations is presented in figure 2.

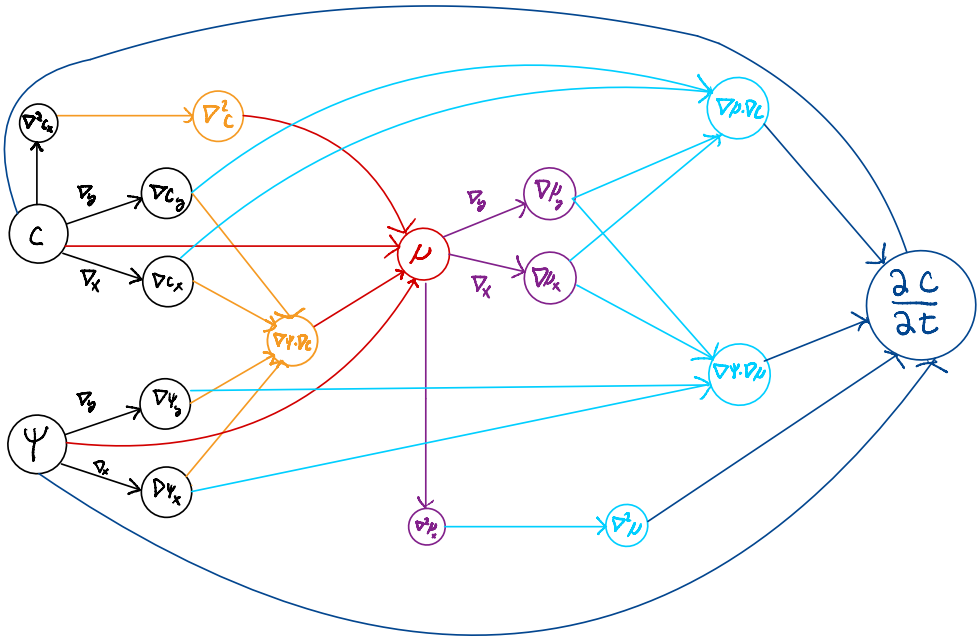

FIG. 2. *The DAG for calculating the RHS of equation 2.7 is presented here. The figure highlights the six levels of computation, as well as the independence of many operations. Each grid point can undergo elementwise calculations by propagating through the DAG elementwise. Alternatively, using finite difference matrix stencils and broadcasting operations, we can treat these calculations as vectorized operations.*

The first implementation of the RHS function relied on computing each operation in figure 2 elementwise using inline functions to avoid allocation. The second approach transformed this calculation to use finite difference matrix stencil operators to compute all discretized differentiation operators, and it used broadcasting operations to calculate the chemical potential μ and final output. This approach was then further optimized by using the cached *mul*! operator from LinearAlgebra.jl that calls accelerated BLAS routines under the hood. The cached, non-allocating matrix multiplication approach for calculating the RHS function achieve a 15x speedup relative to the equivalent elementwise computation. For further discussion of the relative performance, please consult section 4.

Prior to any simulation, it is strongly advantageous to leverage the performance optimization tools throughout the scientific machine learning (SciML) ecosystem to accelerate the code and enhance functionality. In this work, PreallocationTools.jl was used to enable automatic differentiation compatibility for our cached RHS function implementation, which is critical for enabling the required implicit solvers. Furthermore, the sparsity detection and matrix coloring tools in Symbolics.jl [11, 12] and SparseDiffTools.jl are deployed to greatly accelerate the computation of the Jacobian for our function by exploiting its sparse, banded structure.

Lastly, the simulation platform is finalized by leveraging the functionality of the DifferentialEquations.jl library [20] for solving systems of ODEs for the Cahn-Hilliard model. This model exhibits a variety of time scales corresponding to different physical processes in the simulation. Specifically, random perturbations evolve on a fast timescale according to nucleation, while coarsening evolves on a longer timescale [16].

This separation of timescales means the system will be stiff and potentially unstable. To remediate this stiffness, we will utilize a variety of specialized solvers for stiff problems in the simulation platform. Furthermore, this model is composed of a system of $N \times N$ differential equations from the discretized Cahn-Hilliard equation. For typical grid sizes of around 25×25 to 300×300, this system can consist of between $10^3 - 10^5$ elements, necessitating specialized solvers for large, stiff systems. Given these constraints, we tested the following solvers: the explicit stabilized second order Runga-Kutta methods ROCK2(), ROCK4(), RKC(), SERK2(), ESERK5() [1], the second order implicit TRBDF2() solver [14], the implicit, stiff KenCarp4() solver [15], the implicit, stiff Rosenbrock23() solver, and the Sundials CVODE_BDF() solver [10, 13]. Simulations were benchmarked on a model 40×40 system described in figure 3 for a time horizon of $t \in [0, 5]$ to capture both short time nucleation processes and long time coarsening dynamics. The best performing solvers from the smaller system were tested on a 256×256 grid. The results of these tests are further described in detail in section 4.

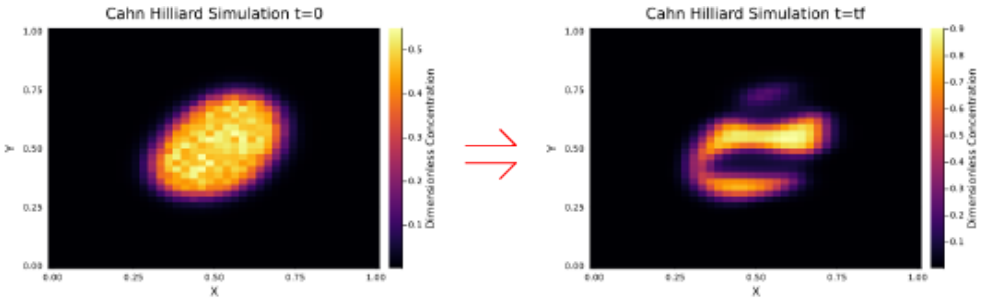

FIG. 3. *The initial and final conditions of the model system used for benchmarking the Cahn-Hilliard simulations is shown here. The custom domain is a hand drawn tilted ellipse with rough boundaries, meant to mimic a prototypical electrode particle. The initial condition is generated by sampling from a uniform distribution* $C_0 \sim \mathcal{U}(0, 1)$. *The system is simulated for a time horizon* $t \in [0, 5]$ *to encapsulate both short and long term dynamics. The final state of the system shows a prototypical phase separated state with different sections of the particle existing in a high concentration state and other sections existing in a low concentration state with sharp boundaries between the two regions. The confirmation of phase separation provides high-level validation for the success of the underlying numerical methods.*

3.3. Applications and Extensions. In this section, we will discuss the extensions and applications of the Cahn-Hilliard simulation platform developed in subsection 3.2. Specifically, we will discuss how within-method parallelization can be leveraged for massively parallel PDE simulation on the GPU, and how this method can be embedded within optimization programs to perform parameter estimation.

Due to Julia's core design principle of multiple dispatch, we can naturally override the previous numerical implementation to use GPU arrays (CuArrays) and GPU operations. This override is enabled by the formulation of the RHS function in terms of matrix multiplications and broadcasting operations. However, it is important to consider when it is worth it to switch to the GPU implementation. Specifically, the GPU operations have a large overhead due to data transfer and managing the parallel computations, so it is important to consider when we will get a speedup from running computations on the GPU of the CPU. As shown in figure 4, this question can be partially answered by looking at the time to perform matrix multiplication on the

GPU versus the CPU.

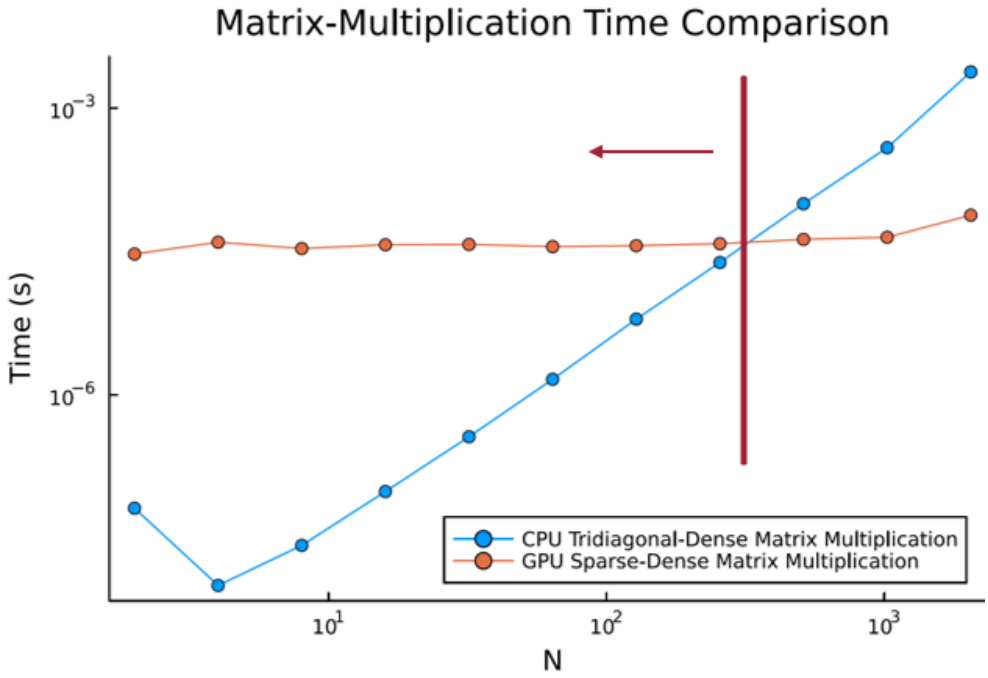

FIG. 4. *For the simulations, the most efficient matrix multiplication type specializations are tridiagonal-dense matrix multiplication on the CPU and sparse-dense matrix multiplication on the GPU. These two operations were compared for randomly generated $N \times N$ matrices of the appropriate type and structure, corresponding to those encountered in this problem. It is found that until around 300×300 sized matrices, it is cheaper to evaluate the expressions on the CPU. To enable more efficient parallelization, it is critical to shift this intersection towards smaller system sizes by utilization more efficient GPU specific operations or designing custom GPU kernels.*

Within this simulation platform, support was built for initializing the system on the GPU and simulating it using the within method parallelization methods in DiffEqGPU.jl. The viability of this extension will be qualified further in section 4.

The final application of this platform was to solve a parameter estimation optimization problem of the form:

$$\min_{\theta \in \mathbb{R}^2} \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \sum_{k=1}^{N_t} \left(\hat{c}(x_i, y_j, t_k; \theta) - c_{data}(x_i, y_j, t_k)\right)^2 \tag{3.2}$$

where $\theta = [D, \Omega]$ is the vector of model parameters to estimate, $N_x = N_y = 40$, $N_t = 100$ are the number of x spatial nodes, y spatial nodes, t temporal nodes respectively, $\hat{c}$ is the concentration profile predicted by the model developed in subsection 3.2, and c_{data} is a synthetic dataset generated using $D = 0.1$ and $\Omega = 3.0$. Support for gradient calculations was enabled using the ForwardDiff.jl [21] and SciMLSensitivity.jl [19] packages. Using these tools for computing gradients, the parameter estimation problem was solved using the NewtonTrustRegion() algorithm within Optimization.jl [8]. The results of the parameter estimation are discussed in section 4.

4. Results and Discussion. In this section, we will analyze the results and performance of the Cahn-Hilliard simulation platform including the different RHS function implementations, the simulations themselves, the performance of the GPU adaptation, and the application to parameter estimation.

The first analysis will focus on the performance of each implementation of the RHS functions; the benchmarks for each implementation can be visualized in figure 5.

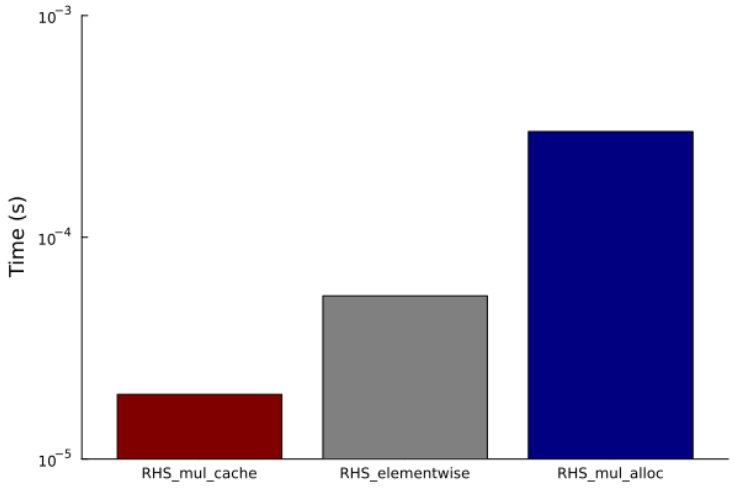

FIG. 5. *This figure depicts the runtime benchmark for the three implementations of the RHS function described in subsection 3.2. The cached matrix stencil function exhibits the best performance on the order of approximately 10 µs. The elementwise implementation performs about an order of magnitude worse, and the allocating matrix stencil implementation is about two orders of magnitudes slower due to the high overhead of allocating memory at each call. From this analysis, we can determine that the best implementation should be done such that it can exploit performance optimizations in Julia's underlying linear algebra routines.*

As shown in figure 5, the fastest performing solver is the RHS that uses broadcasting and cached matrix multiplication for the finite differencing. The elementwise RHS performs about an order of magnitude worse, and the allocating matrix multiplication RHS function performs the worst overall. The allocating matrix multiplication RHS function should be the slowest because of the time it takes to allocate memory at each function call. Furthermore, the cached matrix multiplication approach is much faster than the elementwise computation despite neither allocating memory because it is able to take advantage of optimized BLAS routines for the underlying linear algebra. Despite this analysis, it is important to also consider how each method fares using a variety of implicit and stabilized explicit solvers during the actual numerical simulation, as ultimately the total simulation runtime is the most critical performance metric.

To analyze the overall performance, we will benchmark the cached matrix multiplication and elementwise implementations with the following solvers: ROCK2(), ROCK4(), RKC(), SERK2(), ESERK5(),TRBDF2(), KenCarp4(), Rosenbrock23(), CVODE_BDF() (note all implicit solvers tested with the default implementation and with a Krylov subspace GMRES linear solver). The results of this test are displayed in figure 6.

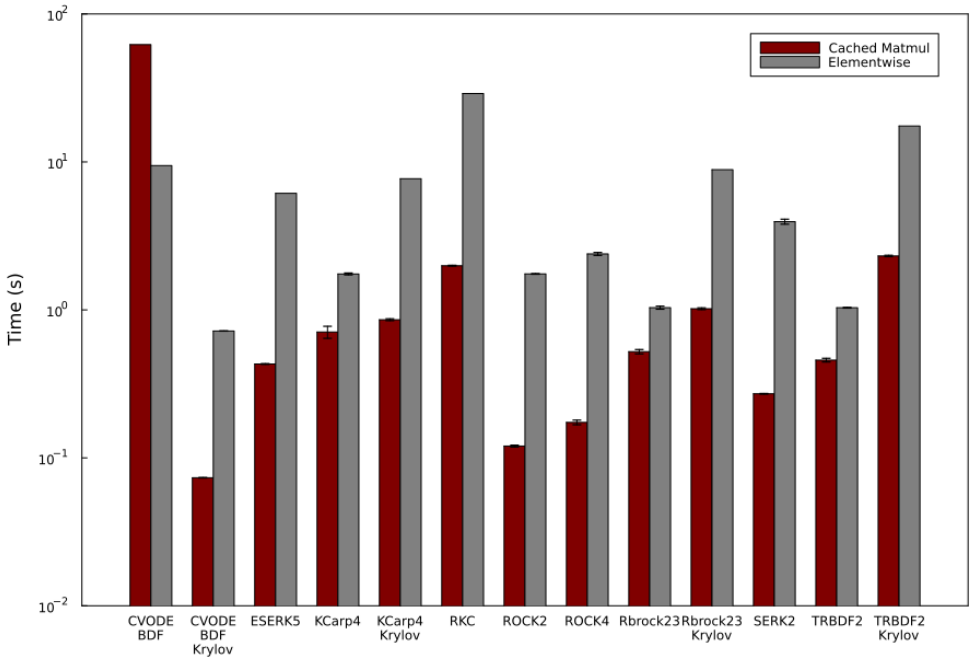

FIG. 6. *In this figure, we plot the performance of the Cahn-Hilliard simulation with a variety of specialized stiff native and non-native Julia solvers in the DifferentialEquations.jl [20] package. From this analysis, we can confidently determine the cached matrix stencil RHS implementation is faster than the equivalent elementwise RHS version by up to several orders of magnitude in most cases. We also determine the fastest solver is the Sundials CVODE_BDF solver with a Newton-Krylov method GMRES Jacobian free linear solver. The closest native Julia solver is ROCK2(), but this solver is explicit, so it will exhibit limited stability with respect to more noisy initial conditions. Furthermore, the best native Julia implicit solver is the TRBDF2 method without a GMRES linear solver. The performance of this might be accelerated by tuning the preconditioner of the GMRES linear solver method.*

It was generally found that the Sundials CVODE_BDF() solver with a GMRES linear solver performed the best in terms of stability and computational time, while the native Julia solvers ROCK2() performed fast but lacked stability, and the TRBDF2() solver had much more favorable stability, but it was an order of magnitude slower than CVOD_BDF() and ROCK2(). For a large stiff system such as this, these results make sense, as any direct implicit solver will require numerous operations to compute the Jacobian at each step during the solution process. The reason Sundials performs so well with the GMRES linear solver is because the method is Jacobian free and does not require these expensive operations that must be computed with automatic differentiation at each step in the solver. The equivalent Julia solvers lack the automatic tuning of the GMRES linear solver that you get with Sundials, and as a result the native Julia solvers suffer in performance relative to Sundials.

The within method GPU parallelization was successfully deployed for this system; however, for the 40×40 system, there was no improvement in the performance on the GPU due to the overhead of GPU operations. Furthermore, when testing the simulation platform on a larger 256×256 system, there was poor stability that hampered the feasibility of the simulation. At this system size, the CPU simulation with Sundials had erroneous divergent behavior, likely due to excessive noise in the

initial condition. Specifically, it was found that the second derivative terms quickly diverged, leading to poor behavior in the simulations. Further research is required to ensure stability in the larger simulations and enable more favorable testing of the GPU platform.

The final performance analysis will focus on the results of the parameter estimation tools in the simulation platform. The simulation was run as an unconstrained PDE-constrained optimization problem with the initial guess of $D = 0.3$ and $\Omega = 2.5$. The algorithm converged in 24 iterations with a loss landscape displayed in figure 7. It was found that despite the costly Hessian construction, NewtonTrustRegion() performed the best due to its favorable convergence relative to other optimization algorithms such as gradient descent, BFGS, or ADAM. The parameter estimation problem was solved in a reasonable 350 – 400 seconds to a tolerance of 1e-4.

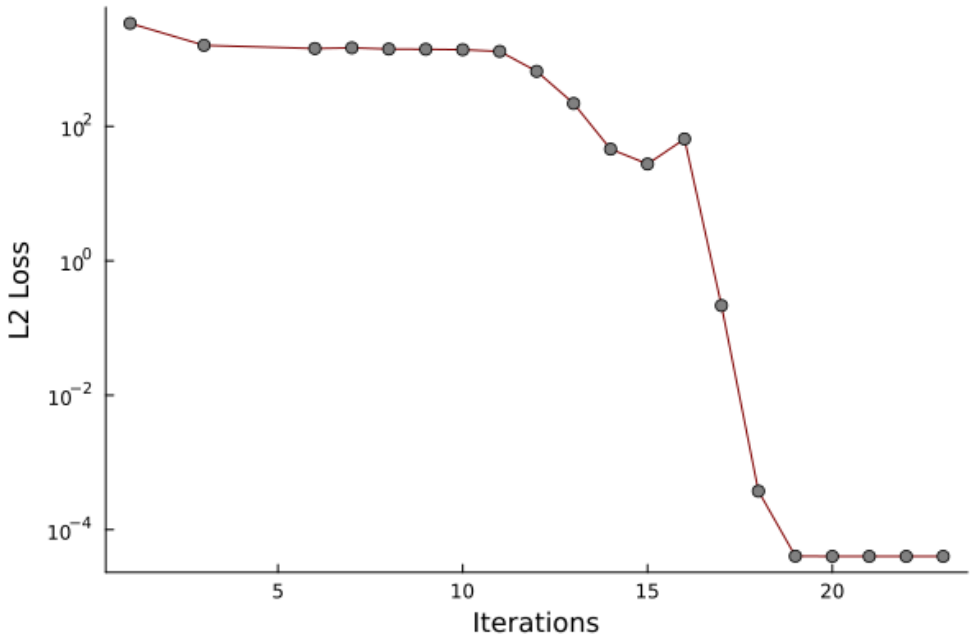

FIG. 7. *A convergence plot for the parameter estimation, showing the L2 error versus the number of iterations. The optimization problem exhibits steady linear convergence followed by rapid convergence close to the minimum, as it is characteristic of the Newton trust region optimization algorithm.*

5. Conclusions. In this work, we established a high performance simulation platform for the 2D Cahn-Hilliard reaction model in custom geometries with the smoothed boundary method. This work utilizes state-of-the-art tools from across the SciML ecosystem to solve the stiff, nonlinear PDE with extremely fast performance. A thorough performance analysis was conducted to test several potential implementations of the Cahn-Hilliard smoothed boundary method model with a variety of candidate solvers from the DifferentialEquations.jl [20] library. This study determines the best simulation performance is exhibited by the Sundials.jl[10, 13] CVODE_BDF() solver with a GMRES linear solver. The simulation platform is further extended to support within method GPU parallelization for large simulations. The simulations are

successfully embedded within a PDE-constrained parameter estimation optimization problem using tools from SciMLSensitivity.jl, Optimization.jl, and ForwardDiff.jl.

Future work on this project will focus primarily on improving numerical stability while maintaining fast performance, improving the viability of within method GPU performance, and conducting more advanced PDE-constrained inversion problems. Specifically, alternative schemes for the spatial discretization will be considered. The finite volume method will be adapted for the 2D SBM model, as it has favorable numerical properties in representing a conservation equation such as the model presented here. This improved stability will likely aid the performance of GPU algorithm by enabling more stable large-scale simulations. Further work will be done to create specialized GPU routines and kernels for the Cahn-Hilliard solver to provide additional GPU-specific performance optimizations. Lastly, the platform developed in this project will be utilized for more advanced parameter estimation and model discovery tasks. We have experimental image data available from collaborators depicting phase transformation in graphite anodes during charging and discharging that will be used for training universal differential equations following the paradigm of Rackauckas et al. [19]. The development of the high performance simulation platform for the Cahn-Hilliard reaction model with the smoothed boundary method is a strong foundation for future studies in advanced modeling and optimization in the field of lithium-ion battery science.

Acknowledgments. The author acknowledges Alex Cohen for his smoothed boundary method mask generation starter code and thoughtful discussion on the numerical methods used in this project. Additionally, the author thanks Chris Rackauckas for his in depth blog posts on simulating stiff PDEs in Julia and his rapid response to questions on the Julia slack. Funding support was provided by the MIT Robert T. Haslam Presidential Fellowship award.

REFERENCES

[1] A. Abdulle and A. A. Medovikov, *Second order chebyshev methods based on orthogonal polynomials*, Numerische Mathematik, 90 (2001), pp. 1–18.
[2] R. W. Balluffi, S. M. Allen, and W. C. Carter, *Kinetics of materials*, John Wiley & Sons, 2005.
[3] M. Z. Bazant, *Theory of Chemical Kinetics and Charge Transfer based on Nonequilibrium Thermodynamics*, Accounts of Chemical Research, 46 (2013), pp. 1144–1160, https://doi.org/10.1021/ar300145c, https://doi.org/10.1021/ar300145c (accessed 2023-04-18). Publisher: American Chemical Society.
[4] M. D. Berliner, D. A. Cogswell, M. Z. Bazant, and R. D. Braatz, *Methods—petlion: Open-source software for millisecond-scale porous electrode theory-based lithium-ion battery simulations*, Journal of The Electrochemical Society, 168 (2021), p. 090504.
[5] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, *Julia: A fresh approach to numerical computing*, SIAM Review, 59 (2017), pp. 65–98, https://doi.org/10.1137/141000671, https://epubs.siam.org/doi/10.1137/141000671.
[6] J. W. Cahn and J. E. Hilliard, *Free Energy of a Nonuniform System. I. Interfacial Free Energy*, jcp, 28 (1958), pp. 258–267, https://doi.org/10.1063/1.1744102.
[7] W. M. Deen, *Analysis of Transport Phenomena*, Topics in Chemical Engineering, Oxford University Press, New York, NY, 2 ed., Nov. 2011.
[8] V. K. Dixit and C. Rackauckas, *Optimization.jl: A unified optimization package*, 2023, https://doi.org/10.5281/ZENODO.7738524, https://zenodo.org/record/7738524.
[9] T. R. Ferguson and M. Z. Bazant, *Nonequilibrium thermodynamics of porous electrodes*, Journal of The Electrochemical Society, 159 (2012), p. A1967.
[10] D. J. Gardner, D. R. Reynolds, C. S. Woodward, and C. J. Balos, *Enabling new flexibility in the SUNDIALS suite of nonlinear and differential/algebraic equation solvers*, ACM Transactions on Mathematical Software (TOMS), (2022), https://doi.org/10.1145/

3539801.
[11] S. Gowda, Y. Ma, A. Cheli, M. Gwozdz, V. B. Shah, A. Edelman, and C. Rackauckas, *High-performance symbolic-numerics via multiple dispatch*, arXiv preprint arXiv:2105.03949, (2021).
[12] S. Gowda, Y. Ma, V. Churavy, A. Edelman, and C. Rackauckas, *Sparsity programming: Automated sparsity-aware optimizations in differentiable programming*, (2019).
[13] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, *SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers*, ACM Transactions on Mathematical Software (TOMS), 31 (2005), pp. 363–396, https://doi.org/10.1145/1089014.1089020.
[14] M. Hosea and L. Shampine, *Analysis and implementation of tr-bdf2*, Applied Numerical Mathematics, 20 (1996), p. 21–37.
[15] C. A. Kennedy, *Additive Runge-Kutta schemes for convection-diffusion-reaction equations*, National Aeronautics and Space Administration, Langley Research Center, 2001.
[16] Y. Li, Y. Choi, and J. Kim, *Computationally efficient adaptive time step method for the cahn–hilliard equation*, Computers & Mathematics with Applications, 73 (2017), pp. 1855–1864.
[17] P. Maire, A. Evans, H. Kaiser, W. Scheifele, and P. Novák, *Colorimetric determination of lithium content in electrodes of lithium-ion batteries*, Journal of The Electrochemical Society, 155 (2008), p. A862.
[18] A. Miranville, *The Cahn–Hilliard equation: recent advances and applications*, SIAM, 2019.
[19] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, and A. Ramadhan, *Universal differential equations for scientific machine learning*, arXiv preprint arXiv:2001.04385, (2020).
[20] C. Rackauckas and Q. Nie, *Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia*, Journal of Open Research Software, 5 (2017), p. 15.
[21] J. Revels, M. Lubin, and T. Papamarkou, *Forward-mode automatic differentiation in Julia*, arXiv:1607.07892 [cs.MS], (2016), https://arxiv.org/abs/1607.07892.
[22] R. B. Smith and M. Z. Bazant, *Multiphase Porous Electrode Theory*, Journal of The Electrochemical Society, 164 (2017), p. E3291, https://doi.org/10.1149/2.0171711jes, https://iopscience.iop.org/article/10.1149/2.0171711jes/meta (accessed 2023-04-18). Publisher: IOP Publishing.
[23] M. Torchio, L. Magni, R. B. Gopaluni, R. D. Braatz, and D. M. Raimondo, *Lionsimba: a matlab framework based on a finite volume model suitable for li-ion battery design, simulation, and control*, Journal of The Electrochemical Society, 163 (2016), p. A1192.
[24] H.-C. Yu, H.-Y. Chen, and K. Thornton, *Extended smoothed boundary method for solving partial differential equations with general boundary conditions on complex boundaries*, Modelling and Simulation in Materials Science and Engineering, 20 (2012), p. 075008, https://doi.org/10.1088/0965-0393/20/7/075008, https://dx.doi.org/10.1088/0965-0393/20/7/075008 (accessed 2023-04-27). Publisher: IOP Publishing.
[25] Y. Zeng, P. Albertus, R. Klein, N. Chaturvedi, A. Kojic, M. Z. Bazant, and J. Christensen, *Efficient Conservative Numerical Schemes for 1D Nonlinear Spherical Diffusion Equations with Applications in Battery Modeling*, Journal of The Electrochemical Society, 160 (2013), p. A1565, https://doi.org/10.1149/2.102309jes, https://iopscience.iop.org/article/10.1149/2.102309jes/meta (accessed 2023-04-18). Publisher: IOP Publishing.
[26] H. Zhao, H. Deng, A. Cohen, J. Lim, Y. Li, D. Fraggedakis, B. Jiang, B. Storey, W. Chueh, R. Braatz, and M. Bazant, *Learning heterogeneous reaction kinetics from x-ray movies pixel-by-pixel*, (2022), https://doi.org/10.21203/rs.3.rs-2320040/v1, https://doi.org/10.21203/rs.3.rs-2320040/v1.
[27] H. Zhao, B. D. Storey, R. D. Braatz, and M. Z. Bazant, *Learning the physics of pattern formation from images*, Phys. Rev. Lett., 124 (2020), p. 060201, https://doi.org/10.1103/PhysRevLett.124.060201, https://link.aps.org/doi/10.1103/PhysRevLett.124.060201.