ARTICLE OPEN

PRISMS-PF: A general framework for phase-field modeling with a matrix-free finite element method

Stephen DeWitt¹ , Shiva Rudraraju^{2,3}, David Montiel¹, W. Beck Andrews¹ and Katsuyo Thornton¹

A new phase-field modeling framework with an emphasis on performance, flexibility, and ease of use is presented. Foremost among the strategies employed to fulfill these objectives are the use of a matrix-free finite element method and a modular, application-centric code structure. This approach is implemented in the new open-source PRISMS-PF framework. Its performance is enabled by the combination of a matrix-free variant of the finite element method with adaptive mesh refinement, explicit time integration, and multilevel parallelism. Benchmark testing with a particle growth problem shows PRISMS-PF with adaptive mesh refinement and higher-order elements to be up to 12 times faster than a finite difference code employing a second-order-accurate spatial discretization and first-order-accurate explicit time integration. Furthermore, for a two-dimensional solidification benchmark problem, the performance of PRISMS-PF meets or exceeds that of phase-field frameworks that focus on implicit/semi-implicit time stepping, even though the benchmark problem's small computational size reduces the scalability advantage of explicit time-integration schemes. PRISMS-PF supports an arbitrary number of coupled governing equations. The code structure simplifies the modification of these governing equations by separating their definition from the implementation of the numerical methods used to solve them. As part of its modular design, the framework includes functionality for nucleation and polycrystalline systems available in any application to further broaden the phenomena that can be used to study. The versatility of this approach is demonstrated with examples from several common types of phase-field simulations, including coarsening subsequent to spinodal decomposition, solidification, precipitation, grain growth, and corrosion.

npj Computational Materials (2020)6:29; <https://doi.org/10.1038/s41524-020-0298-5>

INTRODUCTION

Phase-field models are one of the foundational tools of computational materials science and are used to study microstructure evolution during a variety of processes, including solidification, grain growth, and solid-state phase transformations. A detailed review of phase-field models and their applications can be found in refs ^{1–8}. Phase-field models are almost exclusively solved numerically, yet developing software to perform phase-field simulations can be challenging for two reasons. First, phase-field simulations at scientifically relevant length and time scales are computationally intensive, often requiring millions of computing core hours on parallel computing platforms^{9,10}. Therefore, computational performance and parallel scalability are leading concerns when choosing a numerical approach. Second, a simulation code written for one application is often not transferable to another application without extensive modifications. Different applications require different numbers of phase-field variables, different forms of the free energy functional, and may require the solution of additional coupled equations (e.g., mechanical equilibrium). Even within a single application, such as grain growth in a polycrystalline metal, multiple approaches with very different governing equations are common⁷.

As a response to these challenges, a standard approach is to develop a different code for each application⁷. A single-purpose code has hard-coded governing equations, which reduces computational overhead and permits numerical approaches tailored for the problem at hand. However, this approach has its limitations. Creating and maintaining a number of separate codes, each with its own tests and documentation, can be difficult. Often,

only the numerical methods that are easiest to implement are used, namely finite difference methods and Fourier-spectral methods^{1,3}. While substantial efforts over the past twenty years have been focused on techniques that greatly improve performance such as adaptive mesh refinement and multilevel parallelism^{11–17}, these techniques are often neglected in user-developed single-purpose codes, as they are time-consuming to implement.

An alternative paradigm based on developing and utilizing open-source community frameworks is spreading through the phase-field community⁷. This type of framework contains building blocks for a variety of phase-field models. Therefore, developers' time can be spent extending the capability of the framework rather than implementing basic features in a new single-purpose code. Examples of such community frameworks are FiPy¹⁸, MOOSE^{19–21}, OpenPhase²², AMPE²³, and MMSP²⁴.

In this article, we introduce PRISMS-PF, a new open-source community framework for phase-field modeling, which is a key component of the open-source multiscale materials modeling framework developed by the PRISMS Center²⁵. PRISMS-PF was created upon four principles:

1. The computational performance, including parallel scalability, should meet or exceed that of typical phase-field codes.
2. The framework should support a variety of phase-field models to be useful to a large cross section of the phase-field community.
3. The interface for creating or modifying a set of governing equations should be straightforward and as separated as possible from the numerical methods used to solve them.

¹University of Michigan, Department of Materials Science and Engineering, 2300 Hayward St., Ann Arbor, MI 48109, USA. ²University of Michigan, Department of Mechanical Engineering, 2350 Hayward St., Ann Arbor, MI 48109, USA. ³Present address: University of Wisconsin, Madison, WI 53706, USA. ✉email: stdwtt@umich.edu; kthorn@umich.edu

4. The framework should be open source in order to enable widespread adoption, modification, and development by the members of the phase-field community.

Embodying these four principles, PRISMS-PF enables scientists and engineers in the phase-field community to rapidly develop and employ phase-field models to explore the frontiers of the field. The computational performance of PRISMS-PF is enabled through the use of a matrix-free variant of the finite element method, as opposed to the matrix-based finite element methods traditionally applied for phase-field modeling (e.g., in MOOSE). In combination with Gauss–Lobatto elements, this matrix-free approach permits efficient explicit time integration. PRISMS-PF also leverages adaptive mesh refinement and multilevel parallelism for further increases in performance. Furthermore, PRISMS-PF contains functionality for nucleation and for efficiently handling large polycrystalline systems, two common phenomena in physical systems studied by phase-field modeling, to broaden its applicability. Finally, PRISMS-PF is integrated with the Materials Commons²⁶ information repository and collaboration platform to collect, store, and share a detailed record of each simulation. A more detailed discussion of the methods mentioned above and their implementation in a code structure that delivers performance, ease of use, and flexibility and adaptability to a wide range of applications is given in the “Methods” section of this article.

RESULTS AND DISCUSSION

To demonstrate the performance and flexibility of PRISMS-PF, we compare its performance to that of typical approaches and present examples of its use to investigate several physical phenomena. The comparisons are to a finite difference code and to other open-source frameworks for phase-field modeling. Examples of the use of PRISMS-PF to study coarsening subsequent to spinodal decomposition, precipitate growth, grain boundary nucleation, different formulations of interfacial energy anisotropy, grain growth, and corrosion are presented to demonstrate its flexibility.

Parallel scaling: PRISMS-PF vs. finite difference code

To evaluate the parallel scaling efficiency of PRISMS-PF, a set of strong scaling tests were performed for PRISMS-PF and a custom-developed, optimized finite difference (FD) code written by the authors. The FD code is written in the Fortran language and employs MPI parallelization. The spatial discretization utilizes second-order, centered finite differences on a regular grid. Like PRISMS-PF, the FD code employs a forward Euler time

discretization. Although basic, this code is representative of a type of finite difference code commonly employed for phase-field modeling^{1,8}. The scaling tests were performed for a coupled Cahn–Hilliard/Allen–Cahn system of equations describing the growth of two particles in three dimensions. The initial conditions and final solution are shown in Fig. 1. This system of equations is a simplified version of the models commonly used for solid-state transformations and solidification. Full descriptions of this test problem, computing environment, and FD code are found in the Supplementary Information. The PRISMS-PF calculations were performed with linear elements, so that the theoretical order of accuracy and degrees of freedom (DOF) equal those for the FD code. The calculations on the regular and adaptive meshes have $\sim 3.4 \times 10^7$ DOF and 3.0×10^6 DOF, respectively.

The results of the strong scaling tests on 16–512 computing cores are given in Fig. 2. From Fig. 2a, the PRISMS-PF calculations on a regular mesh have the longest run times, followed by the FD calculations and then by the PRISMS-PF calculations with adaptive mesh refinement (except for 512 cores, where the finite difference calculation is slightly faster than the PRISMS-PF calculation with adaptive meshing). A detailed analysis of the computational cost of calculations using these two codes is presented in the next section.

Figure 2b shows the parallel efficiency (the ratio of the time assuming ideal scaling to the actual time) for the strong scaling tests. For PRISMS-PF with a regular mesh, the parallel efficiency is above 90% for 5.3×10^5 DOF/core or more (64 cores or less), and decreases to 68% at 6.6×10^4 DOF/core (512 cores). For the FD calculation, the parallel efficiency is above 90% for 1.1×10^6 DOF/core or more (32 cores or less), and decreases to 43% at 6.6×10^4 DOF/core (512 cores). While the PRISMS-PF calculations exhibit improved parallel efficiency, the improvement is not driven by a reduction in the absolute deviation from the ideal time, which is actually larger for PRISMS-PF than for FD (see Fig. 2c) due to the more complex data structures involved. Instead, the improved parallel efficiency is driven by the longer baseline wall time of the 16-core calculation for PRISMS-PF. The longer baseline time leads to longer ideal times for PRISMS-PF, meaning that it can have a larger absolute deviation from ideality but still have a lower relative deviation from ideality, as measured by the parallel efficiency.

For the adaptive mesh calculations with PRISMS-PF, Fig. 2a shows that additional cores decrease the wall time up to 256 cores (1.2×10^4 DOF/core), after which, the wall time starts to increase. Unlike the calculations on regular meshes, the parallel efficiency in Fig. 2b for the adaptive mesh calculations does not asymptotically approach unity as the DOF per core increases. This finding

Fig. 1 Two-particle benchmark problem. The (a) initial and (b) final solution for the two-particle test problem used for performance comparisons to the finite difference code.

Fig. 2 Strong scaling behavior. The strong scaling behavior on 16–512 cores for PRISMS-PF with either a regular or adaptive mesh compared with the finite difference code is described in the text. **a** The wall time for the calculations, with dashed lines that give the ideal scaling behavior, where product of the time and the number of cores is fixed at the 16-core value. **b** The parallel efficiency for the calculations. **c** The estimated communication time, defined as the difference between the observed time and the ideal time.

Fig. 3 Performance comparison with a finite difference code. Performance comparison results between PRISMS-PF and the finite difference code for the two-particle test problem. These plots show the wall time on 16 cores at a range of error levels. Dashed gray lines mark error levels corresponding to approximately three, five, and seven points in the interface for the finite difference calculations. **a** Comparison between PRISMS-PF with a regular mesh and linear, quadratic, and cubic elements and the finite difference code. **b** Comparison between PRISMS-PF with cubic elements and a regular or adaptive mesh and the finite difference code.

suggests that the adaptive meshing calculation is out of the ideal scaling regime even for 16 cores (1.9×10^5 DOF/core). At the same DOF per core as the 16-core adaptive meshing calculation, the parallel efficiency for a PRISMS-PF calculation on a regular mesh is ~81% (i.e., it is out of the ideal scaling regime). To correct for the deviation from ideality in the 16-core baseline adaptive meshing calculation, Fig. 2b also shows the parallel efficiency results with the adaptive meshing calculations shifted downward to be equal to the regular mesh results at 1.9×10^5 DOF/core. The corrected curve for the adaptive meshing calculations overlaps with the regular mesh curve and then continues on to lower DOF/core values. This behavior indicates that the adaptive meshing calculations exhibit similar scaling performance to the calculations on regular meshes.

In summary, PRISMS-PF maintains near-ideal strong scaling up to 5.3×10^5 DOF/core on regular meshes. It shows improved parallel efficiency over a finite difference code, although the wall time is about one order of magnitude larger. The PRISMS-PF calculations on adaptive meshes exhibited similar scaling performance as the calculations on regular meshes when corrected for being outside the ideal scaling regime due to their fewer DOF.

Computational cost at fixed error: PRISMS-PF vs. finite difference code

When performing a phase-field calculation, one must balance the objectives of reducing the error and the required computational

resources. With this in mind, we performed a second comparison between PRISMS-PF and the custom FD code, in which we examined the error and the wall time for simulations using the same test problem as the previous section. A detailed description of the test conditions is found in the Supplementary Information. The error for each simulation is defined as the L_2 norm of the difference between its solution and the solution from a simulation that is highly resolved in time and space.

Figure 3 shows the relationship between the time required for PRISMS-PF and FD simulations and their error. Table 1 uses these results to determine the speedup factor for PRISMS-PF compared with the finite difference code for levels of error resulting from three, five, and seven points across the particle/matrix interface in the FD simulations. On a regular mesh with linear elements (which have the same spatial order of accuracy as the FD discretization), PRISMS-PF requires an order of magnitude more run time than a FD calculation with the same error. This finding confirms the common wisdom that FD codes are more computationally efficient than finite element codes in terms of raw throughput.

Despite this disadvantage, Fig. 3 and Table 1 show that PRISMS-PF can leverage higher-order elements and adaptive meshing to become substantially faster than the FD code for this test case. From Fig. 3a, increasing the element degree reduces the run time for the PRISMS-PF calculations across all error levels examined. With a regular mesh, PRISMS-PF with quadratic or cubic elements is slower than the FD code when the allowed error is set to that of the FD code with three points in the interface. As the allowed error

Table 1. The ratio of the wall time for the finite difference calculations to the wall time for the PRISMS-PF calculations for simulations with the same observed error (E_{L2}) for three values of E_{L2} that correspond to different numbers of grid points in the interface for the finite difference calculations ($\lambda_{FD} \approx 3, 5, 7$).

PRISMS-PF element and mesh type	$E_{L2} = 88.18$ ($\lambda_{FD} \approx 7$ pts.)	$E_{L2} = 142.0$ ($\lambda_{FD} \approx 5$ pts.)	$E_{L2} = 508.2$ ($\lambda_{FD} \approx 3$ pts.)
Linear elements, regular mesh	0.0471 ^a	0.0279	0.0390 ^a
Quadratic elements, regular mesh	3.08	0.900	0.377
Cubic elements, regular mesh	5.79	1.90	0.541 ^a
Cubic elements, adaptive mesh	41.3	12.0	2.67 ^a

^aExtrapolated value.

Fig. 4 Solidification benchmark problem. **a** The order parameter (left, and reflected with respect to the simulation domain) and the temperature field (right) at $t = 1500$ for a PRISMS-PF simulation of the solidification of a pure material. Due to fourfold symmetry, the simulation considers only one quarter of the dendrite. **b** The dimensionless velocity of the dendrite tip for the PRISMS-PF simulation and the analytical sharp-interface steady-state solution from ref. ⁷⁰. The simulation parameters are the same as those for PRISMS-PF (#2) in Table 2 (additional details can be found in the Supplementary Information).

is reduced, the PRISMS-PF calculations with quadratic or cubic elements become faster than the FD calculations due to their increased spatial order of accuracy. When the error corresponds to a typical level of resolution in the FD simulation (five points in the interface), PRISMS-PF with cubic elements is 1.9 times faster than the FD code. At that error level, the run times for the PRISMS-PF with quadratic elements and the FD code are approximately the same. However, in general this speedup factor will depend on the number of grid points required to resolve the interface, which will vary depending on the particular phase-field model and coupled physics that are utilized as well as the degree of accuracy intended for the simulation. Unfortunately, to the authors' knowledge there is no systematic review of the minimum number of grid points required for different types/applications of phase-field models, and the number has to be determined on a case-by-case basis with convergence analysis. We can expect the use of higher-order elements to be more advantageous for problems that require higher resolution at the interface. As can be seen in Fig. 3b, the speed of a PRISMS-PF calculation can be further increased with adaptive meshing, with a negligible increase in error. With the typical five-point interface resolution, PRISMS-PF with adaptive meshing is 12 times faster than the FD code for this test case. For more accurate calculations corresponding to seven points in the interface for the FD code, the PRISMS-PF calculations are up to 41.3 times faster.

In summary, PRISMS-PF with cubic elements outperforms a representative FD code for a two-particle test problem at an error level corresponding to a typical choice of five points across the interface for the FD calculation. Without adaptive meshing, the PRISMS-PF calculation is nearly twice as fast and, with adaptivity enabled, it is 12 times faster. At higher error levels, the PRISMS-PF calculation with a regular mesh is slower than the FD calculation, although the PRISMS-PF calculation with adaptive mesh

refinement remains faster. While the governing equations and initial conditions for the test problem are similar to those used in precipitation and solidification simulations, note that the full diversity of phase-field models cannot be represented by a single test case. Changes to the governing equations could increase or decrease the advantages of higher-order elements in PRISMS-PF. The benefit of adaptive meshing is also strongly problem-dependent. For problems that are not very amenable to adaptive meshing (e.g., the initial stages of spinodal decomposition) the performance, relative to the FD code, would be similar to that of simulations on regular meshes. (See applications below for an example of a large-scale simulation of spinodal decomposition and subsequent coarsening that leads to similar run time using PRISMS-PF and FD.) For problems especially amenable to adaptive meshing (e.g., the initial stages of particle nucleation), even larger speedups due to adaptivity may be observed than those presented here. It should be noted that FD codes can also utilize adaptive meshing, albeit with a large increase in their complexity¹³. Even with these caveats, the tests presented here demonstrate that PRISMS-PF does not sacrifice performance for generality, and instead can yield improved performance over a code employing a basic, yet common finite difference approach.

Computational cost: PRISMS-PF vs. other open-source frameworks
As discussed in the introduction, open-source frameworks for phase-field modeling are increasing in popularity. To compare the performance of PRISMS-PF to other such frameworks, we reference results that have been uploaded to the PFHub phase-field benchmarking website for 2D dendritic solidification in a pure material^{27–29}. Note that this benchmark problem can be solved to reasonable accuracy using less than 20,000 DOF, a small enough problem size that the advantage of the explicit time stepping scheme in PRISMS-PF over implicit/semi-implicit

Table 2. Solidification benchmark problem results.

Phase-field framework	Tip velocity	Absolute error	Relative error	Normalized computational cost ^a (core-h)
PRISMS-PF (#1)	7.9×10^{-4}	8.6×10^{-5}	10%	0.016
PRISMS-PF (#2)	8.6×10^{-4}	1.7×10^{-5}	2%	0.35
MOOSE (#1)	7.8×10^{-4}	9.1×10^{-5}	10%	0.029
MOOSE (#2)	7.8×10^{-4}	9.1×10^{-5}	10%	0.10
AMPE	8.3×10^{-4}	4.5×10^{-5}	5%	14
FiPy	1.1×10^{-3}	2.6×10^{-4}	30%	79

A summary of the benchmark results uploaded to the PFHub phase-field benchmarking website for the dendritic solidification problem.

^aDue to differences in the processor clock speeds, the computational cost is scaled to approximate a calculation on a processor with a 3.0 GHz clock speed.

schemes for large problems is minimal. Figure 4 shows the PRISMS-PF solution to this benchmark problem. The tip velocity is steadily approaching the sharp-interface solution, but does not fully converge before nearing the boundary of the computational domain. An extension of the domain size in the existing benchmark that allowed the dendrite tips to reach steady-state velocity would permit a direct comparison with the sharp-interface solution. Unfortunately, given the setup of the benchmark problem, the analytical solution does not provide the accurate solution for the problem. Therefore, a highly resolved simulation is employed to benchmark the accuracy. Convergence tests in time and space indicate that the nondimensional tip velocity is $\sim 8.8 \times 10^{-4}$ at the stopping time designated in the problem definition ($t = 1500$).

Table 2 shows results for this benchmark problem for PRISMS-PF and selected results uploaded to the PFHub website using three other open-source frameworks that focus on implicit or semi-implicit time stepping (MOOSE¹⁹, AMPE²³, and FiPy¹⁸). The PRISMS-PF calculations required three orders of magnitude fewer normalized core hours to complete than the calculations using AMPE and FiPy, while having similar or lower error. The fastest calculations using PRISMS-PF and MOOSE have similar computational cost and tip velocity error.

Each calculation represented in Table 2 was performed by a primary developer of the listed open-source framework. Uploads by users who are not primary developers of the framework used were not included in this analysis as they are more likely to make substantially suboptimal decisions while performing the calculation. The two different PRISMS-PF results represent different parameter sets used to obtain different balances of computational cost and numerical error. The two MOOSE results are from the same developer, but on different computers and with a different remeshing frequency. Details regarding the PRISMS-PF calculations and the analysis of the data from the PFHub website can be found in the Supplementary Information. The computational cost given in Table 2 for each result is normalized for reported processor clock speed in an attempt to make fair comparisons between calculations performed on different hardware systems. While other hardware characteristics likely impact the computational cost of these simulations, the clock speed is the only hardware information gathered on the PFHub website.

In summary, for tests using a dendritic solidification benchmark problem, PRISMS-PF achieves more accurate results with similar or (often much) lower computational cost than other leading phase-field frameworks. To reiterate the caveat from the previous section, no single test problem can contain the full diversity exhibited by phase-field models. The relative performance of codes using explicit versus implicit/semi-implicit approaches depends on many factors (e.g., computational problem size, deviation from equilibrium, the presence of short time-scale phenomena such as nucleation, and the order of the partial differential equations). However, the dendritic solidification

benchmark problem used here provides a neutral, community-determined reference point that is directly relevant to an important class of phase-field simulations.

Applications

The flexibility of PRISMS-PF enables it to be utilized across a range of applications. PRISMS-PF contains 25 example applications that simulate a variety of physical phenomena. In this section, we highlight seven areas in which PRISMS-PF has been applied to showcase various aspects of the framework. Simulation results from these applications are shown in Fig. 5. Table 3 includes standardized comparisons of the simulations for these applications and one of the two-particle simulations from the FD performance comparison above. The table shows the wide variation of calculation sizes in the simulations, ranging from under 400,000 DOF to $\sim 1,000,000,000$ DOF. The table also includes a normalized performance metric, the number of core-seconds per DOF required for each time step of the simulation. The normalized performance metric varies by approximately an order of magnitude. This variation originates from several factors including the number of arithmetic operations in the governing equations, required iterative solutions to linear/nonlinear equations per time step, additional overhead due to nucleation or grain remapping, efficiency losses due to non-ideal parallel scaling, and differences in computing architecture.

Spinodal decomposition followed by coarsening of the microstructure is a classic example of a phenomenon studied with phase-field modeling. To demonstrate the capabilities of PRISMS-PF for large simulations, we show in Fig. 5a the results of a simulation of this phenomenon using the *cahnHilliard* application with 9.6×10^8 DOF. This example is based on a previous calculation performed using a finite difference code to study coarsening behavior subsequent to spinodal decomposition³⁰. The calculations were performed at the National Energy Research Scientific Computing Center (NERSC) on the Intel Xeon Phi Knights Landing architecture. With its multilevel parallelism approach, PRISMS-PF can leverage the strengths of this modern architecture, especially its high level of vectorization. Quadratic elements were used in this calculation because the octree structure of the mesh did not permit an efficient mesh with cubic elements (see the Supplementary Information for a more detailed discussion regarding this choice). The high density of interfaces eliminates the benefits of adaptive mesh refinement, and thus the simulation was performed on a regular mesh. The simulation was performed on 1024 computing cores (16 nodes) for 5.6 days. A corresponding calculation using the finite difference code from ref. ³⁰ requires 3.5 days on 1024 cores, assuming that the relationship between the element size and error from the two-particle benchmark tests holds. (The finite difference estimate was determined by doubling the wall time of a finite difference simulation with half the simulated time of the PRISMS-PF simulation). This example demonstrates the application of PRISMS-PF to a system with

Fig. 5 | Example applications. Examples of five applications of PRISMS-PF. **a** The composition during a simulation of spinodal decomposition and subsequent coarsening at $t = 100,000$ and $t = 250,000$. **b** The evolution of the particles in the scaled-up version of the two-particle growth simulation. **c** A cross section of a 3D simulation of two precipitates in an Mg–Nd alloy (left) and a corresponding transmission electron microscopy image (right). (Reproduced with permission from ref. ³¹. Copyright 2017 Acta Materialia Inc.) **d** The initial (left) and final (right) grain structure for a 2D grain growth simulation. The color represents the grain identification number that remains associated with a grain, as it is transferred between order parameters. **e** A 2D simulation of pitting corrosion. Snapshots are at times 0, 2 s, and 4 s. For each of these times, the left frame (A) represents the metal surface in half of the system, and the right frame (B) represents the metal cation concentration in the other half. **f** A 2D simulation of grain boundary nucleation showing the composition (upper) and the average of the order parameter parallel to the grain boundary (lower). The grain boundary region with a decreased nucleation barrier is denoted by the dashed lines. **g** Initially spherical particles evolved using two different functions to describe strong interfacial anisotropy: a conventional form (left), which enables simulation of shapes with sharp corners and edges⁴¹, and a more sophisticated form (right), which enables flatter facets and non-centrosymmetric anisotropies⁴².

nearly one billion degrees of freedom with performance of only slightly worse than a custom FD code under some unfavorable conditions (no adaptivity, quadratic rather than cubic elements).

The performance tests presented in Figs 2 and 3 involved the simulated growth of two particles. However, matters of scientific or engineering interest often involve many interacting particles. Thus, Fig. 5b shows a PRISMS-PF simulation using the same

governing equations as the two-particle simulations, but scaled up to 128 particles, each with a random initial location and size. This calculation required 5.5 days on 512 computing cores to simulate 400 time units and shows the transition from particle growth to coarsening. From Table 3, this simulation requires $\sim 2 \times 10^{-7}$ core-s/DOF-time step. This normalized performance is approximately two times worse than the two-particle simulation with an adaptive

Table 3. A summary of the performance (measured in core-seconds per DOF for a single time step) of selected simulations presented in this article including the two-particle simulation (cubic elements, adaptive mesh with $\Delta x = 100/64$) and simulations from seven applications highlighted in this section.

Application	DOF	Cores	No. of time steps	Wall time (s)	Core-s/DOF-time step
Two particles	1,341,692 ^a	16 ^b	50,000	400	9.54×10^{-8}
Spinodal decomposition	909,513,218	1024 ^c	3,125,000	483,100	1.74×10^{-7}
128 particles	90,925,928 ^d 372,928,960 ^e	512 ^b	4,000,000	656,640	2.48×10^{-7} 2.36×10^{-7}
Mg–Nd precipitate	965,000 ^a	64 ^b	15,000,000	54,000	2.39×10^{-7}
Precipitate-free zone	285,762 ^d 2,219,878 ^e	64 ^b	1,500,000	6710	1.86×10^{-7} 1.72×10^{-7}
Particle w/strong anisotropy	4,019,691 ^a	16 ^b	400,000	34,900	3.47×10^{-7}
Grain growth	6,774,756 ^a	6 ^f	40,000	8290	1.8×10^{-7}
Corrosion	396,294	64 ^b	4,300,000	44,700	1.78×10^{-6}

DOF degrees of freedom.

^aCalculated using the average DOF.^b2.6 GHz Intel Xeon E5-2670 (Sandy Bridge).^c1.4 GHz Intel Xeon Phi 7250 (Knights Landing).^dAt 5% of the full simulated time.^eAt the full simulated time.^f3.5 GHz Intel Xeon E5-1650 (Ivy Bridge).

mesh on 16 computing cores (9.5×10^{-8} core-s/DOF-time step). This discrepancy indicates non-ideal weak scaling, potentially due to imperfect load balancing. A comparable FD calculation is expected to take 17.2 days on 512 cores, more than twice as long, assuming the same relationship between element size and error from the two-particle tests holds (and determined by scaling the wall time of a simulation with 3% of the total simulated time in the PRISMS-PF simulation).

PRISMS-PF has already been applied toward advancing the understanding of lightweight structural components, the primary testbed for the PRISMS integrated software framework²⁵, to simulate the evolution of precipitates in magnesium-rare earth alloys³¹. Simulations of isolated precipitates were shown to be largely consistent with experimental observations. A two-precipitate simulation (see Fig. 5c) demonstrated a precipitate interaction mechanism to explain outliers. These simulations used the Kim–Kim–Suzuki model³² and linear elasticity to account for misfit strain between the precipitate and the matrix. PRISMS-PF contains an application named *MgRE_precipitate_single_Bppp* for performing these and similar simulations.

Another common application of phase-field modeling is grain growth in polycrystals. The *grainGrowth_dream3D* application uses the grain-remapping algorithm (largely based on ref. ³³), described in the “Methods” section, to simulate grain growth in an isotropic system using the Fan and Chen model³⁴. The initial microstructure containing 739 grains is imported from Dream3D³⁵. Figure 5f shows this initial microstructure as well as an evolved structure that contains 186 grains. The grain remapping led to a 62-fold reduction in the number of order parameters to just 12 shared order parameters to track the grains; no artificial coalescence was observed during the simulation. Less than 9% of the simulation time was spent on the grain-remapping algorithm, a small cost for the nearly two orders of magnitude reduction in the number of order parameters and thus equations to be solved.

The use of phase-field models to describe microstructure evolution in electrochemical systems has been an area of increasing interest. In this area, an application for corrosion is under development using a phase-field model developed by Chadwick et al.³⁶. The electric potential and the transport of ions in the electrolyte are calculated using the smoothed boundary method³⁷ to apply boundary conditions along the metal surface. A Cahn–Hilliard equation with a source term proportional to the

reaction rate tracks the dissolution of the metal. A continuity equation for the electric potential is solved using the nonlinear solver described in the “Methods” section³⁸. Figure 5e shows the evolution of the metal surface and the dissolving metal cation concentration in the electrolyte as a pit grows.

The nucleation capabilities of PRISMS-PF have also been used to study precipitate nucleation in grain boundaries and the associated creation of precipitate-free zones³⁹. Figure 5f shows the results from the *nucleationModel_preferential* application. This application uses a coupled Cahn–Hilliard/Allen–Cahn model similar to the one used in the finite difference performance comparisons and leverages the flexibility of PRISMS-PF in handling nucleation. Nuclei are added explicitly via a nucleation probability that depends on the local nucleation rate, which is calculated dynamically for each element. This method is described with more detail in the “Methods” section below and in the “Nucleation Algorithm Description” section of the Supplementary Information. The nucleation rate function in the nucleation application file includes a spatial dependence for the nucleation barrier (in addition to a dependence on the supersaturation) to mimic the effect of preferential nucleation at a grain boundary. The use of adaptive meshing is particularly effective here because the composition and order parameter fields are nearly uniform across the majority of the domain at the beginning of the simulation.

The user interface for PRISMS-PF permits straightforward modification of governing equations to add more physical phenomena by those who are familiar with C++ programming. PRISMS-PF contains three similar applications with different approaches for handling interfacial energy anisotropy. Each of these is an extension of a single-particle version of the Cahn–Hilliard/Allen–Cahn application used in the finite difference performance comparisons. The *CHAC_anisotropyRegularized* and *CHAC_anisotropy* applications use a standard form of the anisotropy⁴⁰, with and without a fourth-order regularization term that prevents a lack of convergence of the numerical solution with increasing resolution, which can occur with strong anisotropies (i.e., leading to edges or corners)⁴¹. The *facetAnisotropy* application uses the same regularization term, but has a more sophisticated and versatile anisotropy function⁴². Figure 5g shows examples of simulations using these applications.

Future work

PRISMS-PF remains under active development, with initiatives to improve its accessibility, performance, and scope. To reduce the barrier to entry, particularly in educational contexts, PRISMS-PF applications are under development for the nanoHUB website⁴³ with a graphical user interface (GUI) developed using the Raproute Toolkit⁴⁴. The incorporation of implicit time stepping into PRISMS-PF is being investigated for systems that are near equilibrium, in which explicit time stepping is inefficient. The existing nonlinear solver can be applied to compute the implicit update, but improved parallel preconditioning options are required. An extension to the grain-remapping algorithm that can more accurately handle irregularly shaped grains is also planned. Another focus of near-term development is improved integration with other computational tools. Tight coupling with the PRISMS-Plasticity⁴⁵ framework for crystal and continuum plasticity will enable the simulation of dynamic recrystallization. Finally, integration is planned with ThermoCalc, a CALPHAD software package, and with CASM⁴⁶, a first-principles statistical mechanics software package, to provide thermodynamic and kinetic information directly to PRISMS-PF. This integration provides an alternative to the current approach of manually inserting thermodynamic and kinetic parameters for a given material system into the input parameters file.

Conclusions

This article describes a new phase-field modeling framework, PRISMS-PF, with four guiding principles: high performance, flexibility, ease of use, and open access. One characteristic aspect of PRISMS-PF is its use of a matrix-free variant of the finite element method, which enables efficient explicit time stepping by eliminating the need to diagonalize the mass matrix as in traditional finite element methods. This method, combined with advanced adaptive meshing and parallelization strategies, is key to the framework's competitive performance. A benchmark test of two precipitates, in which mesh adaptivity provides a significant speedup, demonstrated that PRISMS-PF was up to 12 times faster than a basic, custom-developed finite difference code at the same level of error. Even for a test case involving spinodal decomposition and subsequent coarsening with no mesh adaptivity and quadratic, rather than cubic elements, the performance of PRISMS-PF is close to that of the finite difference code. In addition, comparison of the results for a dendritic solidification benchmark problem demonstrate that PRISMS-PF yields a similar or higher accuracy with similar or lower computational cost than three other open-source frameworks for phase-field modeling. A second characteristic aspect of PRISMS-PF is its modular, application-centric structure. It is structured such that users primarily interact with applications that contain governing equations, initial/boundary conditions, and parameters. In an application, an arbitrary number of coupled governing equations are input into simple C++ functions, giving users substantial flexibility in their construction. The core library contains shared functionality for the applications and shields users from most of the numerical complexity, allowing them to focus on their system of interest. The core library also includes functionalities for nucleation and for polycrystalline systems that can be activated in any application, thereby broadening the scope of phenomena that can be investigated. The versatility of this approach is demonstrated by its use in a range of applications, including precipitate nucleation, dendritic solidification, grain growth, and corrosion. In sum, this new phase-field modeling framework provides the performance, flexibility, ease of use, and open availability to drive breakthroughs across the field of materials science.

METHODS

PRISMS-PF utilizes advanced numerical methods to enable computational performance at the frontier of the field while also supporting common applications of phase-field modeling. While variants of all of the methods described below have been previously applied in the phase-field community or the wider numerical partial differential equation community, to the authors' knowledge PRISMS-PF is currently the only open-source framework to combine all of these methods within the context of phase-field modeling. PRISMS-PF is structured so that users can leverage these advanced methods without detailed knowledge of their implementation. This structure allows users to focus on the unique governing equations and parameters for their particular application. This article describes version 2.1 of PRISMS-PF⁴⁷.

Code structure

PRISMS-PF is written in the C++ programming language and built upon the deal.II finite element library⁴⁸. The structure of PRISMS-PF reflects the principles set forth in the "Introduction", particularly that it should accommodate a wide variety of governing equations and that those governing equations should be straightforward to modify and be as separated as possible from the numerical methods used to solve them. Therefore, PRISMS-PF is broken into two main components: the core library and the PRISMS-PF applications. The core library contains shared functionality for use in any PRISMS-PF calculation, including the methods described later in this section. An application describes a particular type of simulation, defining an arbitrary number of coupled governing equations and the associated boundary/initial conditions and parameters. The structure of PRISMS-PF allows three levels of engagement: (1) using a preset application and changing the parameters, (2) creating a custom application, and (3) modifying the core library. Thus, PRISMS-PF accommodates a range of users, from those with no programming expertise to expert programmers who want full control over their calculations.

The core library performs the actual finite element calculations, parses input, generates and adapts the mesh, initializes variables, outputs results, handles nucleation, and performs grain remapping. At the heart of the core library is the solver loop, which is structured to maximize flexibility and performance. A flow chart of the solver loop is given in Fig. 6. One cycle of this loop is performed in each time step in the solution of the governing equations (or once for a purely time-independent calculation). To control what operations are necessary for a particular governing equation in the solver loop, governing equations are classified as "explicit time-dependent", "time-independent", or "auxiliary." Explicit

Fig. 6 Flowchart of the solver loop. A flowchart showing the structure of the solver loop in PRISMS-PF. The solver loop consists of four stages. The first stage is for updating the mesh, the list of nuclei, and the list of grains and their assignment to order parameters. In the second stage, all of the governing equations that are to be updated explicitly are updated. The third stage is the loop implementing the nonlinear solver discussed below. The final stage of the solver loop is to output variable values to file (including postprocessed variables) and to save checkpoints at user-specified time increments.

time-dependent equations solve an initial boundary value problem (e.g., the Allen–Cahn equation) via explicit time stepping. Time-independent governing equations solve boundary value problems for sets of linear or nonlinear equations that are time-independent (e.g., Poisson's equation). Auxiliary equations are relational expressions that can be directly calculated from known values at each time step, but do not contain any derivatives with respect to time (e.g., the chemical potential in the split form of the Cahn–Hilliard equation).

A PRISMS-PF application defines a set of governing equations, initial conditions, boundary conditions, and parameters. Each application is derived from the `MatrixFreePDE` class in the core library. This class structure allows any application to selectively override behavior in the core library for increased customization. An application requires a set of input files, each for the purpose of (1) setting constant parameters, (2) setting the governing equations, (3) setting the initial and boundary conditions, (4) defining the custom class for the application, and (5) containing the main C++ function. Additional files to define postprocessing expressions and the nucleation rate are optional.

The contents of the five mandatory app files are briefly discussed below. For a more detailed description of the application files, please refer to the PRISMS-PF user manual⁴⁹. The parameters file is a parsed text file that contains numerical parameters (e.g., mesh parameters) and model parameters (e.g., diffusivity). The model parameters differ between applications, and are allowed to have various data types (e.g., floating-point number, tensor of floating-point numbers, boolean). The type of boundary condition for each governing equation is also set in the parameters file. The equations source file contains the attributes of the variables in the governing equations as well as expressions for the terms in the governing equations themselves. Another mandatory source file contains expressions for the initial conditions for each variable and expressions for Dirichlet boundary conditions that vary in time or space (simpler boundary conditions can be fully specified in the parameters file). Users have flexibility in customizing the governing equations, initial conditions, and boundary conditions because they are entered into C++ functions that can use loops and conditional statements alongside standard arithmetic operations. The class definition file is a C++ header file that defines all of the members of the class, including member variables representing each of the model parameters in the parameters file. The final mandatory application file is a source file containing the C++ main function. This file can be modified to change the overall flow of a simulation, but is identical for the majority of applications.

Beyond the core library and the applications, the third primary component of PRISMS-PF is a test suite containing a battery of unit tests and regression tests. Following best practices, continuous integration testing is used (i.e., the test suite is run after every change to the code in the public PRISMS-PF repository⁴⁷).

Matrix-free finite element calculations

For improved computational performance over typical finite element codes, PRISMS-PF takes advantage of the matrix-free finite element capabilities in the `deal.II` finite element library⁴⁸. For a detailed discussion of this approach, please refer to ref. ⁵⁰; a brief summary is provided below. A central ingredient of any finite element code is the evaluation of a discrete differential operator that approximates a term in the governing equation(s). The discrete operator can typically be evaluated as a matrix-vector product (or a set of matrix-vector products). The standard approach is to assemble a global sparse matrix representing the discrete differential operator for the entire mesh and then multiply it with a vector. However, storing even a sparse matrix is memory intensive for problems with many degrees of freedom, and calculations are often limited by memory bandwidth rather than the speed of the floating-point operations⁵⁰.

To circumvent the performance issues related to storing sparse matrices, PRISMS-PF uses a matrix-free approach (also referred to as a cell-based approach) as implemented in `deal.II` library⁴⁸. The contributions to the global matrix-vector product are calculated element-by-element, and then summed together. Furthermore, for improved performance, the underlying `deal.II` implementation enables sum factorization for the matrix-free approach. Sum factorization refers to a restructuring of the calculation of function values and derivatives on the quadrature points of the unit cell into a product of sums along each direction using 1D basis functions. This strategy for computing and assembling operators significantly reduces the number of arithmetic operations and, more importantly, lends itself for efficient implementation using multilevel parallelism. Furthermore,

PRISMS-PF uses Gauss–Lobatto finite elements instead of traditional Lagrangian finite elements. The quadrature points for Gauss–Lobatto elements coincide with the element nodes, and this feature yields improved conditioning for higher-order elements, and efficient explicit time stepping due to a trivially invertible diagonal “mass matrix” (a matrix where each element is given by $M_{ij} = \phi_i \phi_j$, where ϕ_n is the n th shape function). This combination of the matrix-free finite element approach with sum-factorization and Gauss–Lobatto elements has been previously shown to significantly outperform matrix-based calculations for wave and fluid dynamics calculations⁵⁰.

Explicit time stepping

The default scheme of time discretization in PRISMS-PF is the forward Euler method. The advantages of this explicit scheme are that it is simple to implement^{1,5,51–53}, the calculation for each time step is fast^{1,5,51–53}, and it scales well for parallel calculations^{54–56}. The disadvantages are that it is only first-order accurate, and the time step is limited by the Courant–Friedrichs–Lewy (CFL) condition. Implicit and semi-implicit schemes are alternatives that permit stable calculations with larger time steps, but require the solution of a linear or nonlinear system of equations at each time step. This process is time intensive, and parallel scaling often suffers when the number of degrees of freedom (DOF) exceeds a few million due to the lack of effective parallel preconditioners, except for a small class of problems for which physics-based or geometric multigrid preconditioners are available^{57–59}. Efficient implementations of implicit/semi-implicit schemes also require significant user effort to select/implement the solution strategy (monolithic/staggered), appropriate preconditioner, time-step size, and convergence tolerances⁵⁹. As a result of the advantages of explicit time integration, a recent review of phase-field modeling noted that a majority of the papers it highlighted as making high-impact contributions to materials design employed this approach⁸.

In the context of PRISMS-PF, explicit time stepping is attractive for four reasons. First, the time step for phase-field simulations is often limited to values near the CFL condition by the physics of interest (e.g., sharp gradients in the interface, topological changes), decreasing the advantage of taking larger time steps while using implicit/semi-implicit methods. Second, the improved parallel scaling for explicit methods enables the types of simulations for which PRISMS-PF is designed, with hundreds of millions (or more) of DOF. Third, the diagonal mass matrix provided by the Gauss–Lobatto elements permits efficient explicit time stepping without any ad hoc mass lumping to diagonalize the mass matrix as is required in traditional finite element approaches with Lagrange elements⁵⁰. Fourth, the simplicity of explicit methods reduces the possible ways novice users can make ill-informed choices that substantially reduce code performance.

Adaptive mesh refinement

Adaptive mesh refinement can greatly improve the speed of a simulation with a negligible decrease in accuracy^{14,60}. Phase-field calculations are particularly well suited for adaptive meshing, since order parameters are nearly uniform outside the interfacial regions¹⁴. Despite these benefits, the use of a regular mesh for computationally intensive phase-field simulations is still a common practice due to the complexity of implementing adaptive meshing (e.g., refs ^{9,61}).

PRISMS-PF utilizes the adaptive mesh refinement capabilities from the `deal.II`⁴⁸ and `p4est`⁶² libraries. The mesh is a set of connected quadtree/octrees (a “forest of quadtree/octrees”⁶²), a structured approach that offers improved efficiency over unstructured approaches, while maintaining the flexibility to represent arbitrary geometries⁶². In PRISMS-PF, the user selects an upper and lower bound for one or more variables to define the interfacial regions. The mesh is maximally refined in this region of interest, and is allowed to gradually coarsen to a user-defined minimum level outside of it. Over the course of a simulation, the mesh is periodically regenerated using updated values of the model variables. Thus, as the solution evolves, so does the mesh.

Multilevel parallelism

Parallel computation is crucial for phase-field codes to reduce the run times of large simulations. Using features of the `deal.II`⁴⁸, `p4est`⁶², and `Threading Building Blocks`⁶³ libraries, PRISMS-PF employs three levels of parallelism: distributed memory parallelization, task-based threading, and vectorization. On the distributed memory level, the domain is decomposed

such that each core stores only the refined mesh and corresponding degrees of freedom for its subdomain. This allows for calculations across multiple computing nodes, and permits calculations that require more memory than is available on a single compute node⁶⁴. Each core independently performs calculations to update its portion of the mesh, and communicates the variable values along the subdomain boundary to cores storing adjacent subdomains employing the Message Passing Interface (MPI) protocol. Task-based parallelism is utilized to divide the computational load within a node/core across all available threads. Finally, vectorization up to the AVX-512 standard is permitted, corresponding to operations on eight double-precision floating-point numbers per clock cycle. The sum-factorization approach used in PRISMS-PF is particularly well suited for efficient vectorization⁵⁰. From the perspective of a PRISMS-PF user, this multilevel parallelism scheme is nearly invisible; the user sets the number of MPI processes at run time, and parallelization is handled in the background.

Nonlinear Newton/Picard solver

Many of the partial differential equations involved in phase-field modeling are nonlinear. The nonlinear terms for evolution equations are straightforwardly handled with explicit time-integration methods. However, a nonlinear solver is necessary for nonlinear time-independent partial differential equations, such as the continuity equation for the electric potential in the corrosion model described in the “Results and Discussion” section. In PRISMS-PF, a hybrid Newton/Picard approach is taken to leverage the performance of Newton’s method and the simple implementation of Picard’s method³⁸. Each governing equation is solved using Newton’s method, and if there are multiple nonlinear equations, they are coupled using a Picard (fixed-point) iteration until convergence is reached³⁸. For the Newton iterations, backtracking line search is used to determine the step size to ensure stability without compromising the quadratic convergence rate⁵¹. Linear equations with no coupling to other non-explicit-time-dependent equations are only solved during the first iteration of the nonlinear solver to avoid unnecessary calculation overhead.

Nucleation

Nucleation is an important phenomenon in a number of systems of interest for phase-field modeling, such as solid-state transformations, solidification, and recrystallization¹. To aid in the treatment of these systems, PRISMS-PF contains functionality to assist in the placement of nuclei during a simulation. PRISMS-PF uses the explicit nucleation approach⁶⁵, where supercritical nuclei are stochastically introduced at a time and spatial location determined by a nucleation probability function. The probability function is typically determined from classical nucleation theory^{65,66}, although any probability model can be utilized. In PRISMS-PF, users specify the desired probability function, which can depend on the value of any model variable, spatial location, and time. Different phases can have different nucleation probabilities and different nucleus properties (e.g., size and shape). The initial nuclei are ellipsoids with an arbitrary rotation with respect to simulation axes. The details of the algorithm used to place the nuclei on a distributed mesh (where each processor stores only a portion of the mesh, as in PRISMS-PF) are described in the Supplementary Information.

Grain remapping

For simulations of polycrystalline systems, the naive approach of assigning one-order parameter per grain is intractable even for small systems with hundreds of grains. To handle polycrystalline systems, PRISMS-PF uses the grain remapping approach⁶⁷. Each order parameter stores multiple grains, and grains are transferred between order parameters as needed to prevent direct contact that would lead to artificial coalescence between neighboring grains with the same order parameter. The advantage of this approach is that the evolution equations and data structures are unchanged from those of a typical phase-field simulation. The process used to transfer grains is largely based on the approach from ref. ³³. A recursive flood-fill algorithm is used to identify the elements in each grain. A simplified representation of each grain is created, and these simplified representations are used to identify grains to be transferred. Finally, these grains are transferred to new order parameters. The details of this process, which includes communication over a distributed mesh, are discussed in the Supplementary Information. A similar process is employed when importing

a polycrystalline microstructure (e.g., from electron backscatter diffraction or Dream3D³⁵) as the initial condition for a simulation.

Automatic capture of simulation data and metadata

PRISMS-PF is integrated with the Materials Commons²⁶ information repository and collaboration platform that is specifically designed for materials scientists. Using the Materials Commons command line tool^{68,69}, information from the PRISMS-PF input files are automatically parsed and uploaded to Materials Commons. The same tool can be used to upload the results of the simulation, creating a comprehensive record that can be published for broader dissemination. This capability was utilized for the simulations presented in the “Results and Discussion” section of this article, which can be viewed at the hyperlink given in the “Data Availability” section below.

DATA AVAILABILITY

All of the input files, analysis scripts, and data described in this article are available on Materials Commons: <https://materialscommons.org/mcapp/#/data/dataset/e4cf967e-88a9-4dcb-900b-dd304baa4a0b>.

CODE AVAILABILITY

PRISMS-PF is an open-source computer code under the GNU Lesser General Public License version 2.1. The source code for PRISMS-PF is available at the following hyperlink: <https://github.com/prisms-center/phaseField>. The finite difference codes used in the performance comparisons are available upon request.

Received: 10 September 2019; Accepted: 25 February 2020;

Published online: 26 March 2020

REFERENCES

- Moelans, N., Blanpain, B. & Wollants, P. An introduction to phase-field modeling of microstructure evolution. *Calphad Comput. Coupling Phase Diagr. Thermochem.* **32**, 268–294 (2008).
- Shen, C. & Wang, Y. Phase-field microstructure modeling. in *ASM Handbook*. Vol. 22A (eds Furrer, D. U. & Semiatin, S. L.) 297–308 (ASM International, 2009).
- Chen, L. Q. Phase-field models for microstructure evolution. *Annu. Rev. Mater. Sci.* **32**, 113–140 (2002).
- Emmerich, H. Advances of and by phase-field modelling in condensed-matter physics. *Adv. Phys.* **57**, 1–87 (2008).
- Provatas, N. & Elder, K. *Phase-Field Methods in Materials Science and Engineering* (Wiley-VCH, 2010).
- Steinbach, I. Phase-field models in materials science. *Model. Simul. Mater. Sci. Eng.* **17**, 073001 (2009).
- DeWitt, S. & Thornton, K. Phase field modeling of microstructural evolution. in *Computational Materials System Design* (eds Shin, D. & Saal, J.) 67–87 (Springer, Cham, 2018).
- Tonks, M. R. & Aagesen, L. K. The phase field method: mesoscale simulation aiding material discovery. *Annu. Rev. Mater. Res.* **49**, 79–102 (2019).
- Poulsen, S. O. & Voorhees, P. W. Early stage phase separation in ternary alloys: a test of continuum simulations. *Acta Mater.* **113**, 98–108 (2016).
- Takaki, T. et al. Primary arm array during directional solidification of a single-crystal binary alloy: large-scale phase-field study. *Acta Mater.* **118**, 230–243 (2016).
- Plapp, M. & Karma, A. Multiscale finite-difference-diffusion-Monte-Carlo method for simulating dendritic solidification. *J. Comput. Phys.* **165**, 592–619 (2000).
- Plapp, M. & Karma, A. Multiscale random-walk algorithm for simulating interfacial pattern formation. *Phys. Rev. Lett.* **84**, 1740–1743 (2000).
- Greenwood, M. et al. Quantitative 3D phase field modelling of solidification using next-generation adaptive mesh refinement. *Comput. Mater. Sci.* **142**, 153–171 (2018).
- Provatas, N., Goldenfeld, N. & Dantzig, J. Efficient computation of dendritic microstructures using adaptive mesh refinement. *Phys. Rev. Lett.* **80**, 3308–3311 (1998).
- Hötzer, J., Kellner, M., Steinmetz, P., Dietze, J. & Nestler, B. Large-scale phase-field simulations of directional solidified ternary eutectics using high-performance computing. In *High Performance Computing in Science and Engineering '16: Transactions of the High Performance Computing Center Stuttgart (HLRS) 2016* (eds Nagel, W., Kröner, D., & Resch, M.) 635–646 (Springer, Cham, 2017).

16. Hötzer, J. et al. Application of large-scale phase-field simulations in the context of high-performance computing. In *High Performance Computing in Science and Engineering '15: Transactions of the High Performance Computing Center, Stuttgart (HLRS) 2015* (eds Nagel, W., Kröner, D., & Resch M.) 659–674 (Springer, Cham, 2016).
17. Hötzer, J. et al. The parallel multi-physics phase-field framework PACE3D. *J. Comput. Sci.* **26**, 1–12 (2018).
18. Guyer, J. E., Wheeler, D. & Warren, J. A. FiPy: partial differential equations with python. *Comput. Sci. Eng.* **11**, 6–15 (2009).
19. Gaston, D., Newman, C., Hansen, G. & Lebrun-Grandié, D. MOOSE: a parallel computational framework for coupled systems of nonlinear equations. *Nucl. Eng. Des.* **239**, 1768–1778 (2009).
20. Tonks, M. R., Gaston, D., Millett, P. C., Andrs, D. & Talbot, P. An object-oriented finite element framework for multiphysics phase field simulations. *Comput. Mater. Sci.* **51**, 20–29 (2012).
21. Schwen, D., Aagesen, L. K., Peterson, J. W. & Tonks, M. R. Rapid multiphase-field model development using a modular free energy based approach with automatic differentiation in MOOSE/MARMOT. *Comput. Mater. Sci.* **132**, 36–45 (2017).
22. Tegeler, M. et al. Parallel multiphase field simulations with OpenPhase. *Comput. Phys. Commun.* **215**, 173–187 (2017).
23. Dorr, M. R., Fattebert, J.-L., Wickett, M. E., Belak, J. F. & Turchi, P. E. A. A numerical algorithm for the solution of a phase-field model of polycrystalline materials. *J. Comput. Phys.* **229**, 626–641 (2010).
24. MMSP GitHub Repository. <https://github.com/mesoscale/mmSP> (2019).
25. Aagesen, L. et al. PRISMS—an integrated, open source framework for accelerating predictive structural materials science. *JOM* **70**, 2298–2314 (2018).
26. Puchala, B. et al. The materials commons: a collaboration platform and information repository for the global materials community. *JOM* **68**, 1–10 (2016).
27. PFHub Benchmark Problem 3. <https://pages.nist.gov/pfhub/benchmarks/benchmark3.ipynb/> (2019).
28. Jokisaari, A. M., Voorhees, P. W., Guyer, J. E., Warren, J. A. & Heinonen, O. G. Phase field benchmark problems for dendritic growth and linear elasticity. *Comput. Mater. Sci.* **149**, 336–347 (2018).
29. Boettinger, W. J., Warren, J. A., Beckermann, C. & Karma, A. Phase-field simulation of solidification I. *Annu. Rev. Mater. Res.* **32**, 163–194 (2002).
30. Andrews, W. B., Elder, K. L. M., Voorhees, P. W. & Thornton, K. Coarsening of bicontinuous microstructures via surface diffusion. Preprint at <http://arxiv.org/abs/2002.09428> (2020).
31. DeWitt, S. et al. Misfit-driven β'' precipitate composition and morphology in Mg–Nd alloys. *Acta Mater.* **136**, 378–389 (2017).
32. Kim, S. G., Kim, W. T. & Suzuki, T. Phase-field model for binary alloys. *Phys. Rev. E* **60**, 7186–7197 (1999).
33. Permann, C. J., Tonks, M. R., Fromm, B. & Gaston, D. R. Order parameter remapping algorithm for 3D phase field model of grain growth using FEM. *Comput. Mater. Sci.* **115**, 18–25 (2016).
34. Fan, D., Chen, S. P., Chen, L.-Q. & Voorhees, P. W. Phase-field simulation of {2-D} Ostwald ripening in the high volume fraction regime. *Acta Mater.* **50**, 1895–1907 (2002).
35. Groeber, M. A. & Jackson, M. A. DREAM.3D: a digital representation environment for the analysis of microstructure in 3D. *Integr. Mater. Manuf. Innov.* **3**, 56–72 (2014).
36. Chadwick, A. F., Stewart, J. A., Enrique, R. A., Du, S. & Thornton, K. Numerical modeling of localized corrosion using phase-field and smoothed boundary methods. *J. Electrochem. Soc.* **165**, C633–C646 (2018).
37. Yu, H.-C., Chen, H.-Y. & Thornton, K. Extended smoothed boundary method for solving partial differential equations with general boundary conditions on complex boundaries. *Model. Simul. Mater. Sci. Eng.* **20**, 075008 (2012).
38. Putti, M. & Paniconi, C. Picard and Newton linearization for the coupled model for saltwater intrusion in aquifers. *Adv. Water Resour.* **18**, 159–170 (1995).
39. Porter, D. A. & Easterling, K. E. *Phase Transformations in Metals and Alloys* (Van Nostrand Reinhold Company, 1981).
40. Kobayashi, R. Modeling and numerical simulations of dendritic crystal growth. *Phys. D. Nonlinear Phenom.* **63**, 410–423 (1993).
41. Wise, S. M. et al. Quantum dot formation on a strain-patterned epitaxial thin film. *Appl. Phys. Lett.* **87**, 1–3 (2005).
42. Salvaggio, M., Backofen, R., Bergamaschini, R., Montalenti, F. & Voigt, A. Faceting of equilibrium and metastable nanostructures: a phase-field model of surface diffusion tackling realistic shapes. *Equil. Growth Des.* **15**, 2787–2794 (2015).
43. DeWitt, S. & Gentry, S. PRISMS-PF: Equilibrium Shape for a Misfitting Precipitate. <https://nanohub.org/resources/prismspfmisfit> (2019).
44. Rappture Homepage. <https://nanohub.org/infrastructure/rappture> (2019).
45. Yaghoobi, M. et al. PRISMS-plasticity: an open-source crystal plasticity finite element software. *Comput. Mater. Sci.* **169**, 109078 (2019).
46. CASM GitHub Repository, v0.1.0. <https://github.com/prisms-center/CASMcode> (2015).
47. PRISMS-PF GitHub Repository. <https://github.com/prisms-center/phaseField> (2019).
48. Bangerth, W., Hartmann, R. & Kanschat, G. deal.II—a general-purpose object-oriented finite element library. *ACM Trans. Math. Softw.* **33**, 24-es (2007).
49. DeWitt, S. PRISMS-PF User Manual v2.1. https://prisms-center.github.io/phaseField/doxygen_files/manual.html (2018).
50. Kronbichler, M. & Kormann, K. A generic interface for parallel cell-based finite element operator application. *Comput. Fluids* **63**, 135–147 (2012).
51. Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. *Numerical Recipes* (Cambridge University Press, 2007).
52. Hirsch, C. *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics* (Butterworth-Heinemann, 2007).
53. Tóth, G., De Zeeuw, D. L., Gombosi, T. I. & Powell, K. G. A parallel explicit/implicit time stepping scheme on block-adaptive grids. *J. Comput. Phys.* <https://doi.org/10.1016/j.jcp.2006.01.029> (2006).
54. Gruber, R., Ahusborde, E., Azaiez, M., Keller, V. & Latt, J. High performance computing for partial differential equations. *Comput. Fluids*. <https://doi.org/10.1016/j.compfluid.2010.07.001> (2011).
55. Zhang, J. et al. Extreme-scale phase field simulations of coarsening dynamics on the sunway Taihulight supercomputer. SC '16: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 34–45 (2016).
56. Tennyson, P. G., Karthik, G. M. & Phanikumar, G. MPI + OpenCL implementation of a phase-field method incorporating CALPHAD description of Gibbs energies on heterogeneous computing platforms. *Comput. Phys. Commun.* <https://doi.org/10.1016/j.cpc.2014.09.014> (2015).
57. Kelley, C. T. *Iterative Methods for Linear and Nonlinear Equations* (Society for Industrial and Applied Mathematics, 1995).
58. Pyzara, A., Bylina, B. & Bylina, J. The influence of a matrix condition number on iterative methods' convergence. *Proc. Fed. Conf. Comput. Sci. Inf. Syst.* 459–464 (2011).
59. Keyes, D. E. et al. Multiphysics simulations. *Int. J. High. Perform. Comput. Appl.* **27**, 4–83 (2013).
60. Rosam, J., Jimack, P. K. & Mullis, A. A fully implicit, fully adaptive time and space discretisation method for phase-field simulation of binary alloy solidification. *J. Comp. Phys.* **225**, 1271–1287 (2007).
61. Shimokawabe, T. et al. Peta-scale phase-field simulation for dendritic solidification on the TSUBAME 2.0 supercomputer. SC '11: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11 (2011).
62. Burstedde, C., Wilcox, L. C. & Ghattas, O. p4est: scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.* **33**, 1103–1133 (2011).
63. Reinders, J. Intel threading building blocks: outfitting C++ for multi-core processor parallelism. *J. Comput. Sci. Coll.* <https://doi.org/10.1145/1559764.1559771> (2007).
64. Bangerth, W., Burstedde, C., Heister, T. & Kronbichler, M. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Trans. Math. Softw.* **38**, 14 (2011).
65. Simmons, J. P., Shen, C. & Wang, Y. Phase field modeling of simultaneous nucleation and growth by explicitly incorporating nucleation events. *Scr. Mater.* **43**, 935–942 (2000).
66. Jokisaari, A. M., Permann, C. & Thornton, K. A nucleation algorithm for the coupled conserved-nonconserved phase field model. *Comput. Mater. Sci.* **112**, 128–138 (2016).
67. Krill, C. E. & Chen, L.-Q. Computer simulation of {3-D} grain growth using a phase-field model. *Acta Mater.* **50**, 3059–3075 (2002).
68. Materials Commons API GitHub Repository. <https://github.com/materials-commons/mcapi/> (2019).
69. PRISMS-PF Materials Commons CLI Plugin GitHub Repository. https://github.com/prisms-center/prismspf_mcapi (2019).
70. Karma, A. & Rappel, W. J. Quantitative phase-field modeling of dendritic growth in two and three dimensions. *Phys. Rev. E - Stat. Phys., Plasmas, Fluids, Relat. Interdiscip. Top.* <https://doi.org/10.1103/PhysRevE.57.4323> (1998).

ACKNOWLEDGEMENTS

This work was supported by the U.S. Department of Energy, Office of Basic Energy Sciences, Division of Materials Sciences and Engineering under Award #DE-SC0008637 as part of the Center for Predictive Integrated Structural Materials Science (PRISMS Center) at University of Michigan. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. This research was also supported through computational resources and services provided by Advanced Research Computing at the University

of Michigan, Ann Arbor. The authors thank other contributors to the PRISMS-PF codebase, including Dr. Larry Aagesen, Mr. Jason Luce, and Mr. Xin Bo Qi.

AUTHOR CONTRIBUTIONS

S.D., S.R., and K.T. conceived this work. S.D. and S.R. were the primary developers of the code, with additional contributions from D.M. and W.B.A., S.D. performed the scaling and performance tests and associated analysis. S.D., D.M., and W.B.A. performed the calculations in the applications section. S.D. provided the initial draft of the paper, with contributions from the remaining authors. All of the authors discussed the contents of the paper, edited the paper, and approved its publication.

COMPETING INTERESTS

The authors declare no competing interests.

ADDITIONAL INFORMATION

Supplementary information is available for this paper at <https://doi.org/10.1038/s41524-020-0298-5>.

Correspondence and requests for materials should be addressed to S.D. or K.T.

Reprints and permission information is available at <http://www.nature.com/reprints>

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

© The Author(s) 2020