Computational Materials Science 231 (2024) 112589

Contents lists available at ScienceDirect

Computational Materials Science

journal homepage: www.elsevier.com/locate/commatsci

Full length article

On the construction of an efficient finite-element solver for phase-field simulations of many-particle solid-state-sintering processes

Peter Munch [a,*], Vladimir Ivannikov [b], Christian Cyron [b,c], Martin Kronbichler [a,d]

[a] University of Augsburg, Universitätsstraße 12a, Augsburg, 86159, Germany
[b] Helmholtz-Zentrum Hereon, Max–Planck-Straße 1, Geesthacht, 21502, Germany
[c] Technical University of Hamburg, Eißendorfer Straße 42, Hamburg, 21073, Germany
[d] Ruhr University Bochum, Universitätsstraße 150, Bochum, 44801, Germany

ARTICLE INFO

Dataset link: https://github.com/hpsint/hpsint-data

Keywords:
Solid-state sintering
Finite-element computations
Matrix-free computations
Jacobian-free Newton–Krylov methods
Preconditioning
Grain tracking
Node-level performance analysis
Strong-scaling analysis

ABSTRACT

We present an efficient solver for the simulation of many-particle solid-state-sintering processes. The microstructure evolution is described by a system of equations consisting of one Cahn–Hilliard equation and a set of Allen-Cahn equations to distinguish neighboring particles. The particle packing is discretized in space via multicomponent linear adaptive finite elements and implicitly in time with variable time-step sizes, resulting in a large nonlinear system of equations with strong coupling between all components to be solved. Since on average 10k degrees of freedom per particle are necessary to accurately capture the interface dynamics in 3D, we propose strategies to solve the resulting large and challenging systems. This includes the efficient evaluation of the Jacobian matrix as well as the implementation of Jacobian-free methods by applying state-of-the-art matrix-free algorithms for high and dynamic numbers of components, advances regarding preconditioning, and a fully distributed grain-tracking algorithm. We validate the obtained results, examine in detail the node-level performance and demonstrate the scalability up to 10k particles on modern supercomputers. Such numbers of particles are sufficient to simulate the sintering process in (statistically meaningful) representative volume elements. Our framework thus forms a valuable tool for the virtual design of solid-state-sintering processes for pure metals and their alloys.

1. Introduction

Sintering is a physically complex process that includes various mechanisms interacting and competing with each other. The obtained densification and microstructure of the sintered packing are of key interest. The accurate prediction of the powder coalescence for a given material and heating profile is a challenging multiphysics problem, which couples mass transport and mechanics. It is convenient to split the entire sintering process into two stages, as visualized in Fig. 1: the *early stage* and the *later stage*. Initially, the microstructure mainly evolves due to intensive neck-growth and shrinkage, while the particles[1] become strongly non-spherical and grain growth starts to play an important role in the later stage. These rheological differences justify the application of specialized numerical models and methods with different computational costs for each of the stages. For instance, *molecular dynamics* [1,2] provides the most detailed insights into processes taking place during solid-state sintering, but can only be applied to domains spanning a few particles (or even less). Thus, it is typically not appropriate to predict the densification, which is a hallmark of sintering on the meso- and macroscale. On the contrary, approaches based on *continuum mechanics* [3] can operate on the macroscale but remain phenomenological, since they can predict changes of the geometry of a whole workpiece only with additional assumptions on local material densification and cannot resolve microscopic phenomena, such as grain growth. *Discrete element methods* (DEM) and *phase-field methods* are positioned between the two aforementioned approaches in terms of scale: they can handle packings of hundreds or thousands of particles. While both can capture shrinkage during analysis (provided the corresponding mechanisms are properly included in the model), DEM simulations typically rely on the assumption of nearly spherical particles and remain thus largely limited to early-stage sintering [4]. Capturing both densification and grain growth properly can be crucial for many applications,

* Corresponding author.
E-mail addresses: peter.muench@uni-a.de (P. Munch), vladimir.ivannikov@hereon.de (V. Ivannikov), christian.cyron@hereon.de (C. Cyron), martin.kronbichler@uni-a.de (M. Kronbichler).

[1] In practice, a powder particle may contain multiple grains. For reasons of simplicity, we use the terms *particle* and *grain* interchangeably. Such a simplification is admissible for the present work, since, in our studies, each particle always consists of a single grain.

https://doi.org/10.1016/j.commatsci.2023.112589
Received 22 June 2023; Received in revised form 15 October 2023; Accepted 16 October 2023
Available online 27 October 2023

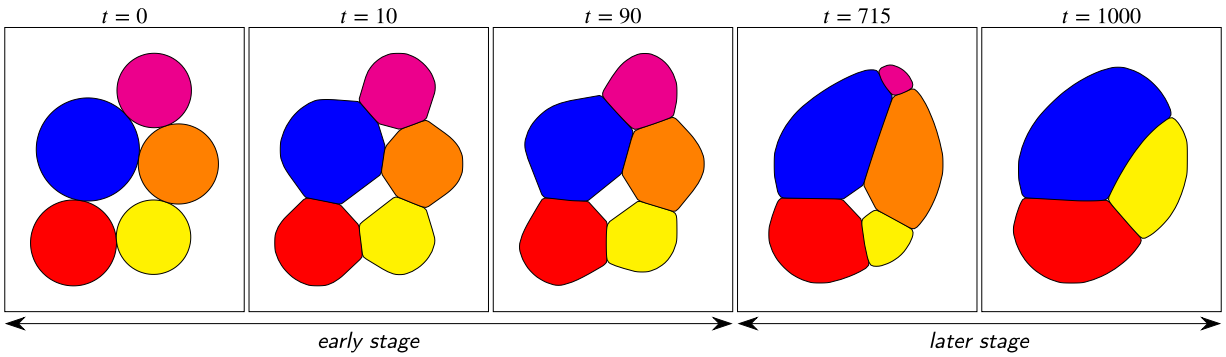


Fig. 1. Visualization of different phases of sintering for a 5-particle packing in 2D. The colors indicate the different grains. Up to $t = 90$ (*early stage*), a clear neck growth is visible and the shape of the particles remains, at least to some extent, spherical; after that (*later stage*), the grain-growth phenomenon plays an important role, with smaller grains disappearing.

for instance, the manufacturing of patient-specific biodegradable magnesium implants [5], where the mechanical properties as well as the biodegradation process may crucially depend on both the geometry and microstructure of an implant [6].

Recently, a number of large-scale simulations of sintering using phase-field methods has been reported. For example, [7] simulated early-stage sintering for packings containing 332, 333, 1172 and 2968 particles. The authors of [8] applied the phase-field framework `Pace3D` [9] to simulate sintering of 1.2 million particles, using discretization of 2400^3 grid cells. An even larger packing of 3.1 million particles on 2560^3 cells was analyzed in [10], however, for a simpler, ideal grain-growth problem. All these publications have in common that they use *finite difference methods* (FDM) on uniform meshes mostly with explicit time stepping. The *finite element method* (FEM) is successfully used, for instance, in the framework `MOOSE` [11] to demonstrate the sintering simulation of several hundreds of particles [12] and in the package `Tusas` [13] to perform various large-scale solidification simulations with up to 270 million unknowns; both using implicit time stepping and Jacobian-free Newton–Krylov (JFNK) methods. The package `PRISM-PF` [14], in contrast, applies the fast evaluation routines of `deal.II` to accelerate explicit time stepping in the context of a large set of phase-field simulation cases.

In our previous works, we have tackled modeling of solid-state sintering by multiple numerical methods in close interaction with experimental validation. In [15], we used a FEM-based phase-field approach to simulate shrinkage and neck growth between two particles. To simulate the early-stage sintering process for larger packings of particles, we applied DEM and proposed a novel approach [4] that couples the diffusive mass-transport processes described by an elementary 7-equation model [16] with mechanical interactions of particles arising because of changes of their geometry. Due to the low computational costs of the DEM approach, the developed code is able to simulate the sintering of relatively large packings consisting of 3000–5000 particles on a regular laptop within a few hours. However, this approach is limited to the early stage of sintering and cannot capture non-spherical grains and grain-growth phenomena. In the current work, we extend our phase-field-based code [15] to simulate packing sizes similar to those we considered in the case of DEM, without the limitation to the early sintering stages.

The development of a FEM-based code with implicit time stepping for such scales is a challenging task, since it involves algorithmic developments on many levels. In particular, the number of degrees of freedom (DoFs) N increases linearly with the number of particles (N_P): $N \sim \mathcal{O}(N_c N_P)$, with N_c being the number of order parameters, which are sets of non-neighboring particles. This number is, in practice, rather high (10–20) but independent of the number of particles. Nevertheless, the computational complexity is quadratic $\mathcal{O}(N_c^2 N_P)$, since the surface-coupling terms between particles need to be evaluated.

In this work, we develop efficient evaluation strategies of the Jacobian that fully exploit modern hardware features, reducing the effective complexity to $\mathcal{O}(N_P)$–$\mathcal{O}(N_c N_P)$. To this end, we adopt Jacobian-free Newton–Krylov approaches and fast matrix-free operator evaluation to solve systems of linear equations arising from implicit time stepping. We, furthermore, discuss efficient preconditioning and propose a fully distributed and improved version of the grain-tracking/remapping algorithm [17]. The latter is crucial to keep the number of order parameters low and, as a result, to make the operator evaluation more efficient. During the whole simulation, we maintain two representations of the particles: a 0D representation for postprocessing purposes and a phase field for each order parameter for computation purposes, as shown in Fig. 2. The 0D representation is used to detect situations when particles belonging to the same order parameter get too close and to resolve them such that the number of order parameters is minimized. As a consequence, the fast synchronization of both representations is crucial to reduce the computational time and enables then large-scale simulations.

Though the number of order parameters is minimized via grain tracking, it still remains significant. In the context of FEM, each order parameter would correspond, e.g., to a component of a vectorial element. Such a high number of components is not common for FEM applications in other areas, with *density functional theory* as an exception [18,19]. In our case, the number of components may also change between time steps, which poses an additional software challenge.

The remainder of this article is organized as follows. In Section 2, we provide an overview of the governing equations and their FEM discretization. Section 3 outlines the aims of our optimizations, and Section 4 proposes the performance-relevant solver components. Sections 5 and 6 present numerical results and discuss the overall performance of the solver in detail, respectively. Section 7 demonstrates the application of the proposed algorithms in alternative, more advanced, sintering formulations. Finally, Section 8 summarizes our conclusions and points out future research directions. Our novel simulation framework is freely available, as `hpsint`, on GitHub[2] and uses the open-source library `deal.II` as FEM backend [20,21].

2. Sintering model and its numerical solution

2.1. Governing equations

The classical formulation for modeling solid-state sintering of N_P particles proposed by Wang [22] and adopted in numerous works [23–26] is based on a system of Cahn–Hilliard and Allen–Cahn equations:

$$\frac{\partial c}{\partial t}(\mathbf{x}, t) = \nabla \cdot \left[M \nabla \frac{\delta F}{\delta c} \right], \tag{1a}$$

[2] https://github.com/hpsint/hpsint.

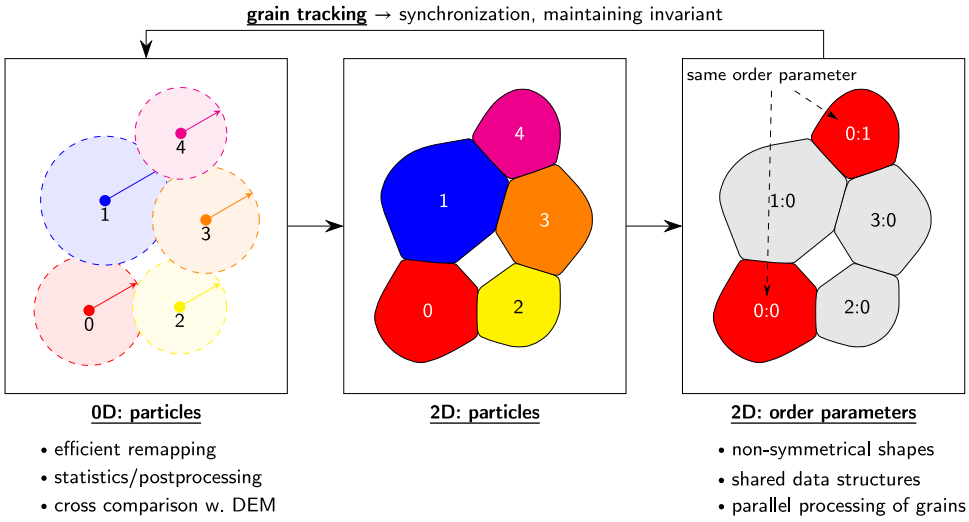


Fig. 2. Overview of types of particle representation considered in this publication and their interaction for a 5-particle packing in 2D. Phase-field simulations are run for order parameters, i.e., sets of particles that do not neighbor. For postprocessing purposes, we maintain a 0D representation with particles described by position, radius, and additional statistical quantities. This reduced model is also used to determine potential contacts of grains and to maintain the invariant that order parameters can only contain non-neighboring particles, which might involve the remapping of solution vectors.

$$\frac{\partial \eta_i}{\partial t}(\mathbf{x},t) = -L\frac{\delta F}{\delta \eta_i} \quad \text{for} \quad 1 \le i \le N_P, \tag{1b}$$

where $\mathbf{x}$ is the position vector in space. The microstructure evolution is described by a conserved variable c and a set of non-conserved unknowns η_i. Variable c can be interpreted as the molar fraction of the overall material and has a magnitude of 1 inside particles and 0 in voids. The unknown η_i describes the position of particle i within the domain such that $\eta_i = 1$ inside the ith particle and $\eta_i = 0$ elsewhere. Due to the local support of η_i, it is common in the literature [17,27,28] to collect non-neighboring particles in groups called *order parameters* and describe all particles in such a group by a single η_i. We have adopted this strategy in this publication. In the following, we implicitly assume that particles are treated in groups unless it is explicitly pointed out that particles are treated individually.

The free energy of system (1) is given by the integral

$$F = \int_\Omega \left[f\left(c,\eta_i\right) + \frac{1}{2}\kappa_c|\nabla c|^2 + \sum_i^{N_P}\frac{1}{2}\kappa_\eta|\nabla\eta_i|^2 \right] \mathrm{d}\Omega \tag{2}$$

based on the following Landau-type polynomial:

$$f = Ac^2(1-c)^2 + B\left[c^2 + 6(1-c)\sum_i^{N_P}\eta_i^2 - 4(2-c)\sum_i^{N_P}\eta_i^3 + 3\left(\sum_i^{N_P}\eta_i^2\right)^2\right], \tag{3}$$

where A and B are energy coefficients and κ_c and κ_η are gradient prefactors. These parameters can be extracted from the surface and grain-boundary energy properties of the material by using the relations obtained from the analysis of the behavior of the phase-field variables across the flat surface [24].

Even though it has been recently revealed [29] that the bulk free energy (3) may spontaneously generate the void phase on triple and higher-order junctions, we have still decided to use the original formulation from [22] due to its widespread use and for the sake of simpler validation and comparison with the existing literature. The implementation aspects discussed in the next sections are applicable to similar models based on other free-energy expressions, provided $F\left(c,\eta_i\right)$ is a continuous differentiable function.

Parameter L defines the mobility of the grain boundary and is typically set as a constant. The diffusion along different pathways is introduced by the scalar mobility [22]

$$M = M_{\mathrm{vo}}\phi + M_{\mathrm{va}}(1-\phi) + \underline{4M_{\mathrm{s}}c^2(1-c)^2} + M_{\mathrm{gb}}\sum_{i=1}^{N_P}\sum_{j\neq i}^{N_P}\eta_i\eta_j, \tag{4}$$

where $\phi = c^3\left(10 - 15c + 6c^2\right)$. Here, the subscripts *vo*, *va*, *s* and *gb* denote the mobility coefficients for the volumetric, vapor, surface, and grain-boundary paths, respectively. For the sake of simplicity, the scalar form of the mobility is chosen in the following, whereas a more complex tensorial form [23] is discussed as an extension separately in Section 7. The mobility coefficients can be introduced via the Arrhenius relationship by defining the corresponding prefactors and activation energies [30]. This allows the model to be calibrated with the available experimental data [15].

We note that the surface-mobility term in Eq. (4) (underlined) slightly differs from that used in [7,22,30,31] to enhance the convergence rate of the Newton solver. A more detailed discussion of this alteration can be found in Supplementary Material S1.

2.2. Discretization

We discretize Eq. (1) by means of multicomponent linear Lagrange C^0 finite elements. For this purpose, we reformulate the original system to:

$$\begin{aligned}
\frac{\partial c}{\partial t} &= \nabla\cdot\left[M\nabla\mu\right],\\
\mu &= \frac{\partial f}{\partial c} - \kappa_c\nabla^2 c,\\
\frac{\partial \eta_i}{\partial t} &= -L\left[\frac{\partial f}{\partial \eta_i} - \kappa_p\nabla^2\eta_i\right],
\end{aligned}$$

by introducing the chemical potential $\mu = \delta F/\delta c$ as auxiliary variable, expanding explicitly the variational derivatives $\delta F/\delta c$ and $\delta F/\delta\eta_i$, and exploiting the definition (2). This leads to the following weak form:

$$\left(v_c,\ \frac{\partial c}{\partial t}\right) = -\left(\nabla v_c,\ M\nabla\mu\right), \tag{5a}$$

$$\left(v_\mu,\ \mu\right) = \left(v_\mu,\ \frac{\partial f}{\partial c}\right) + \left(\nabla v_\mu,\ \kappa_c\nabla c\right), \tag{5b}$$

$$\left(v_{\eta_i},\ \frac{\partial \eta_i}{\partial t}\right) = -\left(v_{\eta_i},\ L\frac{\partial f}{\partial \eta_i}\right) - \left(\nabla v_{\eta_i},\ L\kappa_p\nabla\eta_i\right), \tag{5c}$$

where the usual boundary integrals arising after applying integration by parts vanish due to imposition of the no-flux boundary conditions.

We use BDF2 with adaptive time steps for time discretization. The resulting nonlinear system $\mathcal{F}(\boldsymbol{u}) = 0$, with the vector of unknowns $\mathbf{u} = \begin{bmatrix} c & \mu & \eta_i \end{bmatrix}^T$, is solved by means of a Newton solver. The action of the Jacobian on a factor is either evaluated exactly or approximated

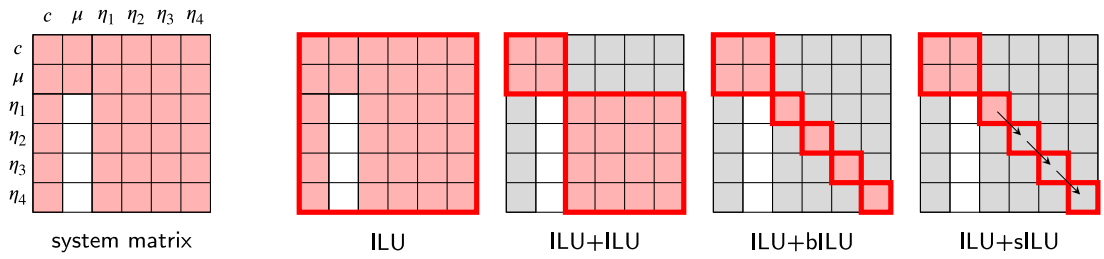


Fig. 3. Visualization of the block sparsity pattern of (1) the system matrix and (2) the considered preconditioners for four order parameters.

by finite differences around the linearization point $\mathbf{u}_{\text{lin}}$ (Jacobian-free approach):

$$J(\mathbf{u}_{\text{lin}})\mathbf{p} \approx J'(\mathbf{u}_{\text{lin}})\mathbf{p} = \frac{\mathcal{F}(\mathbf{u}_{\text{lin}} + \beta\mathbf{p}) - \mathcal{F}(\mathbf{u}_{\text{lin}})}{\beta}, \tag{6}$$

where the parameter β is chosen as described in [32–34]. The linearization of the weak form (5) is derived with respect to variations of the state vector $\delta\mathbf{u}$ as

$$\left(v_c,\ \frac{\partial \dot{c}}{\partial c}\delta c\right) = -\Big(\nabla v_c,\ \Big[\frac{\partial M}{\partial c}\delta c + \frac{\partial M}{\partial \nabla c}\nabla\delta c + \frac{\partial M}{\partial \eta_j}\delta\eta_j + \frac{\partial M}{\partial \nabla \eta_j}\nabla\delta\eta_j\Big]\nabla\mu\Big), \tag{7a}$$

$$\left(v_\mu,\ \delta\mu\right) = \left(v_\mu,\ \frac{\partial^2 f}{\partial c^2}\delta c + \frac{\partial^2 f}{\partial c \partial \eta_j}\delta\eta_j\right) + \left(\nabla v_\mu,\ \kappa_c \nabla\delta c\right), \tag{7b}$$

$$\left(v_{\eta_i},\ \frac{\partial \dot{\eta}_i}{\partial \eta_i}\delta\eta_i\right) = -\left(v_{\eta_i},\ L\frac{\partial^2 f}{\partial \eta_i \partial \eta_j}\delta\eta_j\right) - \left(\nabla v_{\eta_i},\ L\kappa_p\nabla\delta\eta_i\right). \tag{7c}$$

The notation $\dot{\psi} = \partial\psi/\partial t$ is used here to denote the first time derivative of an arbitrary variable ψ for convenience. Note that (7a) contains derivatives of M with respect to gradients of concentration c and order parameters η_j in order to account for tensorial mobility described in Section 7. The linearized forms of the free-energy function and the mobility are listed in Supplementary Material S2. Fig. 3 shows the sparsity pattern of the resulting Jacobian matrix. The coupling terms introduce a quadratic complexity $\mathcal{O}(N_c^2 N_P)$ in both storage and computational effort, making the assembly of the Jacobian unfeasible.

2.3. Algorithmic overview

Fig. 4 shows an overview of the algorithm used to solve the solid-state-sintering problem. Before solving the nonlinear system with a Newton solver, we optionally run adaptive mesh refinement (AMR) and the grain tracker to detect potential new contacts and to minimize the number of order parameters. After the solution or when the linear/nonlinear solver fails to converge in the prescribed number of iterations, we optionally increase or decrease the time-step size τ. In the following, we investigate this algorithm regarding performance.

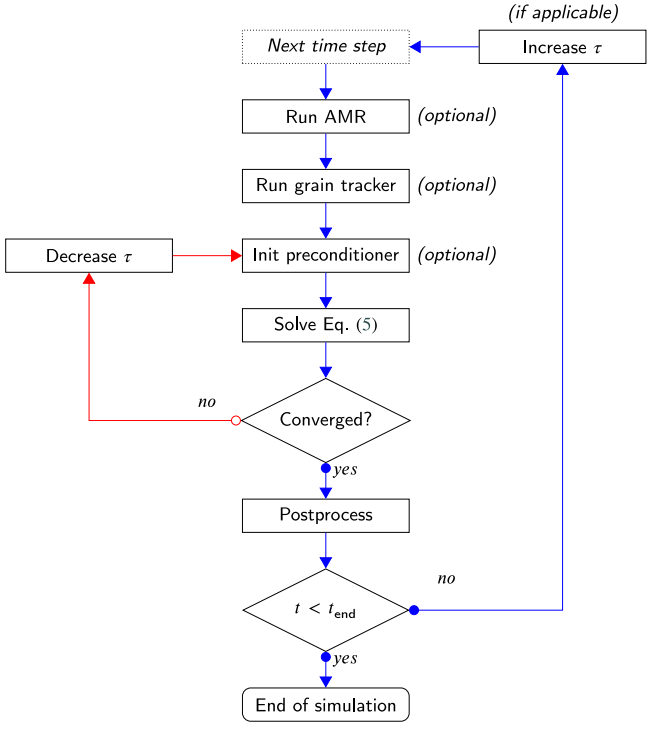


Fig. 4. Simplified flow chart of the solution procedure of the sintering problem with adaptive time stepping and adaptive meshing.

3. Performance metric

Our aim is to minimize the computational time for running a solid-state-sintering simulation up to the required physical end time, allowing the simulation of larger problem sizes. For the described solution approach, the runtime can be estimated by the sum of the costs of the nonlinear solution process and other costs, like AMR, grain tracking, and postprocessing:

$$T = T_{\text{sol}} + T_{\text{AMR}} + T_{\text{grain tracking}} + T_{\text{post}} + \cdots$$

The cost of the nonlinear solution process [35] is the sum of the costs of each time step:

$$T_{\text{sol}} = \sum T_{\text{sol},i} = N_T \overline{T}_{\text{sol}}.$$

In the following, we consider averaged times, which are indicated by overbars. Under the assumption that we use a Newton solver and solve the Jacobian by means of iterations of the preconditioned generalized minimal residual method (GMRES), we can refine the estimates of the costs $\overline{T}_{\text{sol}} = T_{\text{sol}}/N_T$:

$$\begin{aligned}\overline{T}_{\text{sol}} = {} & \overline{T}_{P,\text{setup}} && \triangleright \text{ setup preconditioner}\\ & + \overline{N}_N \overline{T}_{J,\text{setup}} && \triangleright \text{ setup Jacobian}\\ & + \overline{N}_N \overline{N}_R \overline{T}_{\text{residual}} && \triangleright \text{ residual evaluation}\\ & + \overline{N}_N \overline{N}_L \underbrace{(\overline{T}_{J,\text{apply}} + \overline{T}_{P,\text{apply}} + \overline{T}_{\text{updates}})}_{\triangleright \text{single linear iteration}},\end{aligned}$$

with the preconditioner only updated once per nonlinear solve. Here, $\overline{N}_N$ is the number of nonlinear iterations, each of which requires $\overline{N}_R$ residual evaluations and $\overline{N}_L$ iterations of the linear solver. To accelerate the solution process, one needs, on the one hand, to minimize the accumulated number of Jacobian and preconditioner evaluations (each $N_T \overline{N}_N \overline{N}_L$) and of their setup ($N_T$ and $N_T \overline{N}_N$ times, respectively) and, on the other hand, to minimize the cost of their application and construction ($\overline{T}_{J,\text{apply}}$, $\overline{T}_{P,\text{apply}}$, $\overline{T}_{J,\text{setup}}$, $\overline{T}_{P,\text{setup}}$). The costs $\overline{T}_{\text{updates}}$ related to the updates of the solution vector are fixed and can hardly be optimized.

We note that these different costs need to be balanced against each other. For example, minimizing only N_T by increasing the time-step sizes can lead to increased $\overline{N}_N$ and $\overline{N}_L$ due to increased nonlinearity

and nonsymmetry of the system of linear equations to be solved. In the present work, we increase the size of the time steps as long as a user-provided threshold regarding the numbers of linear and nonlinear iterations is not violated (see Section 4.5.2). This implies that the main factors we can tune are the costs of the evaluation of the Jacobian ($\overline{T}_{J,\text{apply}}$) and the setup and application costs of the preconditioner ($\overline{T}_{P,\text{setup}}$, $\overline{T}_{P,\text{apply}}$). The choice of the preconditioner has an effect not only on $\overline{T}_{P,\text{setup}}$, $\overline{T}_{P,\text{apply}}$ but also on $\overline{N}_P$, $\overline{N}_N$, and $\overline{N}_L$. Hence, an effective preconditioner is characterized by a tradeoff between the setup and application times as well as the resulting iteration counts. Generally $\overline{N}_L \gg \overline{N}_R$, which implies that the costs of the residual evaluation are not crucial. At the same time, this function constitutes the core algorithm of the Jacobian-free implementation. For computational efficiency, this work uses a matrix-free evaluation of the Jacobian, which is conceptually similar to the residual evaluation. Here, the cost of setting up the factors of the Jacobian matrix is small and only involves storing the linearization point and, potentially, precomputing its values at the quadrature points.

Minimizing the solution time by splitting the spatial computation domain into partitions for parallel computation – so that each process only works on a part of the mesh cells, called *locally owned cells* – is a key aspect to enable large systems, since most of the computational time is spent here. However, extensive memory usage and poor parallel scalability of other parts of the code might become a bottleneck as well. For instance, gathering data from all processes to be able to run the grain-tracking algorithm, as done in [17,36], is only feasible for small number of processes but is not an option at a larger scale, limiting the maximum problem sizes per process that can be solved. This implies that we need a grain-tracking algorithm that returns the minimal number of order parameters but is also computationally and memory-wise cheap to apply even if it is not executed at each time step.

Hardware

All experiments are performed, unless stated otherwise, on a dual-socket 20-core *Intel Cascade Lake Xeon Gold 6230* system (2.6 TFLOP/s, 180 GB/s, AVX-512) with up to 8 compute nodes. To give a broader performance perspective, we also report experiments from a dual-socket *AMD EPYC 7713* processor with 64 cores per socket. It has an aggregated bandwidth of 320–340 GB/s and only supports AVX2, but the clock frequency and core count are higher (4.6 TFLOP/s). The parallel-scaling experiments are executed on a dual-socket 24-core *Intel Xeon Platinum 8174* (Skylake) system of the supercomputer SuperMUC-NG,[3] (achieved STREAM triad memory throughput of 205 GB/s) with up to 1024 compute nodes (49k processes).

4. Solver components

Section 3 identifies the solver components needed for scalable large-scale simulations of solid-state-sintering processes: fast operator evaluation of the Jacobian and the residual as well as a balance between the costs and effectiveness of the preconditioner. Both ingredients need to able to deal with a large and dynamic number of order parameters, i.e., components. A fully parallel implementation of the grain-tracking algorithm enables a low number of order parameters also for large simulations, where replicating information is unfeasible. In the following, we present the algorithmic realization of these steps.

Algorithm 1: Cell loop considering *all* vector blocks. The conversion of the number of components into a constant expression is done at a central place before looping over all cells.

```
if n_blocks = n_blocks^static then
    for cell ∈ cells do
        for b = 1 to n_blocks do
            read from block b of source vector
        perform cell integral → action of A_e in (8)
        for b = 1 to n_blocks do
            write to block b of destination vector
else
    not shown
```

4.1. Dealing with large and dynamic number of components

During sintering, grains build necks with neighboring particles, increase and/or decrease in size, and might disappear. The grain-tracking algorithm assigns and reassigns the grains to order parameters. Typical values of the number of order parameters are between 8 and 14. Depending on the topology of grains, the number of order parameters can be dynamically reduced or has to be increased. A natural choice is to assign each order parameter to a component of a vectorial finite element. For solid-state sintering, this implies vectorial elements with two components for the Cahn–Hilliard system and one component for each order parameter. Such a dynamic behavior is in contrast to the fixed number of components in most vectorial problems solved with finite-element models.

In order to simplify the workflow in the context of a general-purpose FEM library, we do not actually work with vectorial elements but with multivectors defined upon scalar finite elements, which we manually combine on the cell[4]/quadrature-point level. Each component corresponds to a vector in the multivector, which simplifies adding or removing blocks during remapping. In the following, we use the term *block* to denote an individual vector related to a scalar function space within the multivector and denote the vector of all unknowns as block vector. To accelerate computations of integrals on the mesh cells, we use the C++ template mechanism for generating separate code for different numbers of components to provide the compiler with optimization opportunities via known loop bounds and data-structure sizes. From compute kernels precompiled up to a known maximal possible number of components, the right kernel is chosen at runtime. The resulting procedure is summarized in Algorithm 1. The memory consumption is $\mathcal{O}(N_c N_P)$ and the computational complexity is $\mathcal{O}(N_c^2 N_P)$, since coupling between order parameters is considered on the cell level also when a cell is not cut by the boundary of a grain.

In the context of large-scale finite-difference implementations [8,10], it is common to exploit the local support of grains [37], i.e., $\eta_i(\mathbf{x}) > 0$ only for limited $\mathbf{x}$. For this purpose, the indices of relevant grains are stored for each cell. In practice, the number of possible relevant grains is fixed and limited, e.g., to $c_{\max} = 6$. Since $c_{\max} \ll c$ is generally much lower than the number of order parameters and – per definition – constant, the memory consumption is $\mathcal{O}(N_P)$ and the computational complexity is $\mathcal{O}(N_P)$ with a larger constant of proportionality.

It is also possible to adopt this approach in the context of FEM if one realizes that the data structure used for existing finite-difference codes is – in a nutshell – a compressed-row-storage-format object with rows being the cell/vertex/DoF indices, storing a fixed number of the relevant column indices (grain indices) and associated values (η_i). In the context of computational solution of the density functional theory based on FEM [38,39], such a data structure was successfully used in the form of *sparse block vectors*.

[3] https://top500.org/system/179566/ received on December 11, 2022.

[4] We use the terms *cell* and *element* interchangeably.

Algorithm 2: Cell loop considering only vector blocks *relevant for the current cell*. The conversion of the number of components into a constant expression has to be performed for each cell.

```
for cell ∈ cells do
    blocks ← relevant blocks of cell
    for b ∈ blocks do
        read from block b of source vector
    if |blocks| = n_blocks^static then
        perform cell integral → action of A_e in (8)
    else
        not shown
    for b ∈ blocks do
        write to block b of destination vector
```

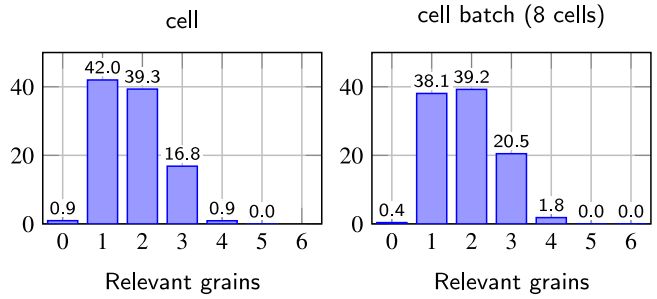


Fig. 5. 51 particles: distribution [%] of cells and cell batches with n relevant grains at $T = 500$ (see Section 6).

Despite of the appeal of sparse block vectors with N_P blocks regarding memory consumption, their usage within an implicit (non)linear solver is challenging. For instance, sparse block vectors imply different and frequently changing sparsity patterns of the matrix of each block, which makes the setup of preconditioners more expensive and parallelization more challenging if established linear-algebra libraries should be used. However, processing all particles of one order parameter implies a natural parallelization across particles under the assumption that the grain-tracking algorithm is able to reduce the number of order parameters to a reasonable value. Therefore, we defer the investigation of such data structures to future work and, instead, focus on a novel simplified approach: we allocate (dense) block vectors in memory but limit the arithmetic work on cells to the relevant blocks in the block vector, including the coupling in cell integrals. This is achieved by storing the relevant order parameters for each cell. Obviously, this approach still implies a memory consumption of $\mathcal{O}(N_c N_P)$ and an associated overhead throughout the remainder of the solver (e.g., non-relevant entries of the vectors have to be zeroed and communicated), but the computational effort can be significantly lowered. In the current work, we use the—heuristic—criterion $\eta_i > 10^{-5}$ to determine whether a grain is relevant within a cell. Fig. 5 shows the distribution of the resulting number of components on cells. The value ranges from 0 to 6 with 81% of the cells only containing 1 or 2 grains, which indicates a significant increase of efficacy. We present the corresponding performance comparison in Section 6. The resulting algorithm is shown in Algorithm 2 indicating that the number of components needs to be translated into a constant expression for each cell.

4.2. Fast Jacobian and residual evaluation

A widespread optimization of iterative linear solvers for modern hardware is to not assemble the final global matrix but implement the action of the linear(ized) operator through a loop over cells and compute the FEM integrals on the fly in a matrix-free way. This approach has been established in the high-order spectral-element community. By now, it is commonly used in the computational fluid dynamics formulations [40–42] and has been also applied, e.g., in the context of solid mechanics [43,44], material science [14] and computational plasma physics [45]. Depending on the polynomial degree and the underlying quadrature formula, the matrix-free operator evaluation is beneficial for accelerating the actual matrix–vector product or only for reducing the setup costs of the Jacobian. The latter might provide, depending on iteration counts, a better overall runtime also when the actual evaluation is more expensive.

Due to the implementation similarity between the matrix-free evaluation of the Jacobian and the evaluation of the residual, we introduce the concepts of high-performance evaluation of an arbitrary operator $\mathcal{A}(x)$. The overall structure is [46–48]:

$$\boldsymbol{v} = \mathcal{A}(\boldsymbol{x}) = \sum_e \mathcal{G}_e^T \circ \underbrace{\tilde{\mathcal{S}}_e^T \circ \mathcal{Q}_e \circ \mathcal{S}_e}_{\boldsymbol{v}_e = \mathcal{A}_e(\boldsymbol{x}_e)} \circ \mathcal{G}_e \boldsymbol{x}. \tag{8}$$

For each cell e, the operator $\mathcal{G}_e$ gathers the values pertaining to the local FEM solution expansion from the source vector $\boldsymbol{x}$ and applies constraints (like hanging-node constraints [49] related to AMR). The operator $\mathcal{S}_e$ computes values or gradients associated to the vector $\boldsymbol{x}$ at the quadrature points. These quantities are processed by a quadrature-point operation $\mathcal{Q}_e$; the result is integrated and summed into the vector $\boldsymbol{v}$ by applying $\tilde{\mathcal{S}}_e^T$ and $\mathcal{G}_e^T$. In the literature, specialized implementations for GPUs [48,50–52] and CPUs [45–47,50,53] for operations as expressed in (8) have been presented, including the use of the structure in interpolation matrices. For CPUs, it is an option to vectorize across elements [46,47], i.e., evaluate $\mathcal{A}_e$ for multiple cells in different lanes of vector execution units by the same instructions. This necessitates the data to be laid out in a struct-of-arrays fashion. The necessary permutations to support unstructured meshes can be done, e.g., by $\mathcal{G}_e$, while looping through all elements [46]. On modern CPUs, the most common number of lanes N_{SIMD} is either 4 (AVX) or 8 (AVX512) for double-precision floating-point numbers, implying that batches of 4 or 8 cells are processed at once.

For multicomponent FEM, the indices within $\mathcal{G}_e$ are the same for each block, allowing to process blocks one by one with the same index data. Furthermore, $\mathcal{S}_e$ and $\tilde{\mathcal{S}}_e^T$ can be executed for each component individually. The operator $\mathcal{Q}_e$ determines the quantities to be provided at quadrature points, encoding the actual physics via the weak forms (5) and (7). In the case of the residual (5) and the Jacobian (7), we need the values and the gradients of c, μ, η_i and have to multiply the results by both the value and the gradient of the respective test function. When evaluating the operators in (8) sequentially, the amount of temporary data is $N_{\text{SIMD}}(N_c + 2)(d + 1)(p + 1)^d$ data fields, with p the polynomial degree and d the spatial dimension. The performance crucially depends on the ability to keep the temporary results between the steps of Eq. (8) on a cell accessible quickly. Hence, the higher numbers of components require larger cache capacities to remain fast.

At each quadrature point, the values and gradients of c, μ, η_i are coupled via $\mathcal{Q}_e$. From a performance point of view, the two limiting factors are possible register spills for high number of components and $\mathcal{O}(N_c^2)$ arithmetic operations. Apart from constants, this is the same complexity as for a matrix-based implementation, since coupling between all components arises in both cases. The crucial difference is that a matrix-based code involves the coupling over the whole stencil of a point. Furthermore, matrices are large objects in memory with low data reuse, implying that data needs to be loaded via the slow main memory, whereas the matrix-free evaluation only experiences the complexity on a single point and on cached data. In the ideal case, the time complexity is thus $\mathcal{O}(N_P N_c)$ if the computation can be hidden behind the global memory access. In the following, we concentrate on linear elements ($p = 1$) in 3D, as common in the literature [12,26,30] for simulating sintering problems.

Fig. 6 shows the time and throughput of the computation of the residual and the application of the Jacobian. Here, the throughput is

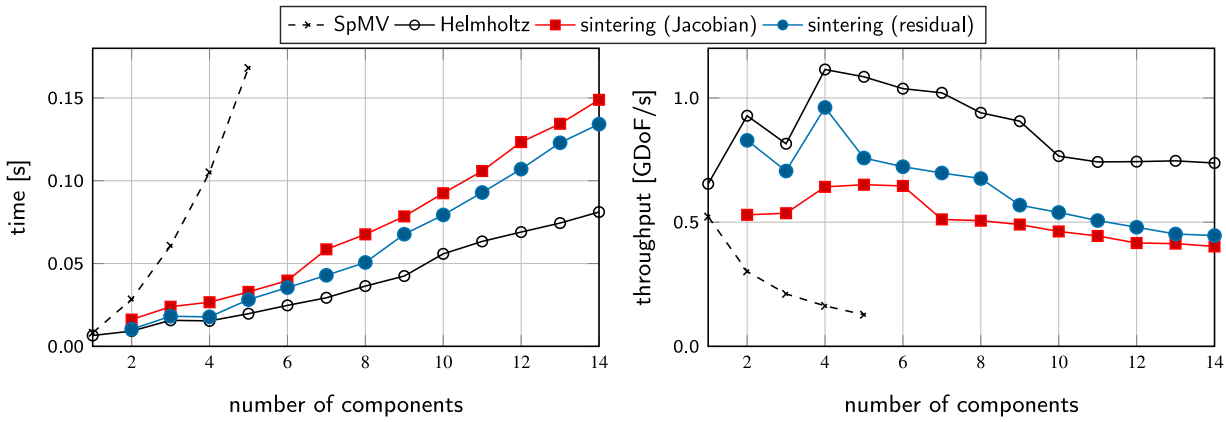


Fig. 6. Comparison of the time and throughput (given in GDoF/s=10^9DoF/s) of the application of sparse matrix (SpMV), of the vector Helmholtz operator, and of the generic sintering operator. 4.3 million DoFs per vector component for different number of components. Run on 1 compute node with Intel Xeon 6230 (40 cores).

computed as the ratio between the number of degrees of freedom for all vector components and the runtime/wall time of the experiment in seconds, in short, the number of DoFs processed per second. We use Algorithm 1 on a uniformly refined Cartesian mesh with 4.3 million DoFs per vector component. As a reference, we also list the timings for the evaluation of the vector Helmholtz operator $(v_i, u_i)+(\nabla v_i, \nabla u_i)$. Since this operator involves the same operations $\mathcal{S}_e$ and $\tilde{\mathcal{S}}_e^T$, the difference illustrates the cost of the physics-based $\mathcal{Q}_e$. The residual evaluation of the sintering operator involves around 50% more floating-point operations on average, with the execution being 30% slower than the one of the vector Helmholtz operator. Evaluating the Jacobian gives a 19% lower throughput than the residual because of additional floating-point operations from linearization and additional data access for loading a given linearization point, as we precompute the values of c, μ, η_i and the gradients of c, μ at the quadrature points. We also show the times of the multiplication with a sparse matrix in which all components are coupled. It is clear that the time of the sparse matrix–vector multiplication increases quadratically with the number of components, as does the memory consumption, making this approach unfeasible for large number of components. Note that sparse matrices involve additional costs for the assembly and the creation of the pattern of non-zero entries.

The experiments also show a decrease in throughput with increasing the number of components for the matrix-free case, which indicates a quadratic complexity with low constant of proportionality. These results motivate the development of algorithmic variants that reduce the number of components a particular cell needs to work on, e.g., by using Algorithm 2. The central ingredient for this algorithm is to track the relevant grains *per cell batch*. Since the batching of cells is done a priori, changes in the simulation lead to a slight increase in the number of relevant grains per cell batch, compared to the one of the relevant grains per cell, as illustrated in Fig. 5.

Table 1 illustrates the performance metrics of the underlying experiments. One can see that the write access to main memory, normalized per unknown, increases only marginally with increasing number of components, indicating that all relevant data needed for $\mathcal{S}_e/\mathcal{Q}_e$ fit into cache. The data read per DoF decreases with increasing number of components, reflecting the fact that other data (e.g., indices and metric terms) can be shared between components and the relative amount of data related to the linearization point, where the Cahn–Hilliard part dominates, decreases. The number of floating-point operations increases as 73→78 and 86→99 FLOP per quadrature point as the number of components increases in the cases of residual and Jacobian evaluation, respectively. As a summary, the results are inserted into a graphical representation of the roofline performance model [55] for the residual evaluation in Fig. 7. With increasing the number of components, the arithmetic intensity increases (11.5→13.1 FLOP/byte)

Table 1
Comparison of measured throughput, read/write memory access and arithmetic work for evaluation of the residual and the Jacobian of generic sintering operator for different number of components (see Fig. 6). As a reference, the vector Helmholtz operator involves 416 FLOP/DoF. The hardware performance counters were accessed with the LIKWID tool [54].

N_c+2	Sintering (residual)				Sintering (Jacobian)			
	D/s	r/D	w/D	F/D	D/s	r/D	w/D	F/D
2	0.83	5.3	1.3	608	0.53	37.7	1.6	690
3	0.71	4.9	1.5	591	0.54	29.4	1.7	688
4	0.96	4.7	1.5	579	0.64	25.1	1.7	704
5	0.76	4.6	1.6	579	0.65	22.6	1.8	711
6	0.72	4.6	1.6	579	0.65	20.9	1.8	717
7	0.70	4.5	1.6	582	0.51	19.7	1.9	723
8	0.68	4.4	1.7	585	0.51	18.7	1.9	730
9	0.57	4.4	1.7	590	0.49	18.1	1.9	756
10	0.54	4.4	1.7	595	0.46	17.5	2.0	763
11	0.51	4.3	1.7	601	0.44	17.0	2.0	771
12	0.48	4.2	1.7	607	0.42	16.5	2.1	779
13	0.45	4.2	1.7	614	0.41	16.2	2.1	786
14	0.45	4.2	1.8	620	0.40	15.9	2.1	794

D/s: throughput in [GDoFs/s]; r/D: read data per DoF in [Double/DoF]; w/D: written data per DoF in [Double/DoF], F/D: work [FLOP/DoF]. Due to the usage of Gauss–Legendre quadrature implying approx. 8 quadrature points per vertex, F/D needs to be divided by 8 to obtain the number of FLOP per quadrature point and component. The numbers represent the average of 100 repetitions of the experiment.

and the obtained performance decreases (556→277 GFPLOP/s). As a result, 27%–12% of the obtainable hardware bandwidth limit is reached. Albeit the gap appears to be significant, the result can be explained by limits not included in this simplistic roofline model, most prominently by the relatively high share of non-floating point operations for linear shape functions, such as unstructured gather/scatter access and related integer operations. For comparison, a sparse matrix–vector product (SpMV) would reach a much lower performance of 45 GFLOP/s due to an arithmetic intensity ≈ 0.25 FLOP/byte.

In order to reduce the impact of a higher number of components, the following optimization strategies are developed:

- We do not explicitly compute individual terms of the free energy (3) or mobility (4) but instead apply them directly to vectors. This allows us to transform equations like

$$\sum_{j,i\neq j} \frac{\partial^2 f}{\partial \eta_i \partial \eta_j} u_j = \sum_{j,i\neq j} (\eta_i \eta_j) u_j = \eta_i \sum_{j,i\neq j} \eta_j u_j = \eta_i(\alpha - \eta_i u_i),$$

  with the precomputed factor $\alpha = \sum_i \eta_i u_i$. Note that some scaling factors are dropped for the sake of simplicity.
- Other factors, e.g., $\sum_i \eta_i^2$ and $\sum_i \eta_i^3$, can be precomputed and reused for each grain.
- Off-diagonal block entries are processed in pairs, and symmetry is exploited, e.g., $\sum_{i=1}^{g} \sum_{j=1,i\neq j}^{g} \eta_i \eta_j = 2 \sum_{i=1}^{g} \sum_{j=1}^{i-1} \eta_i \eta_j$.

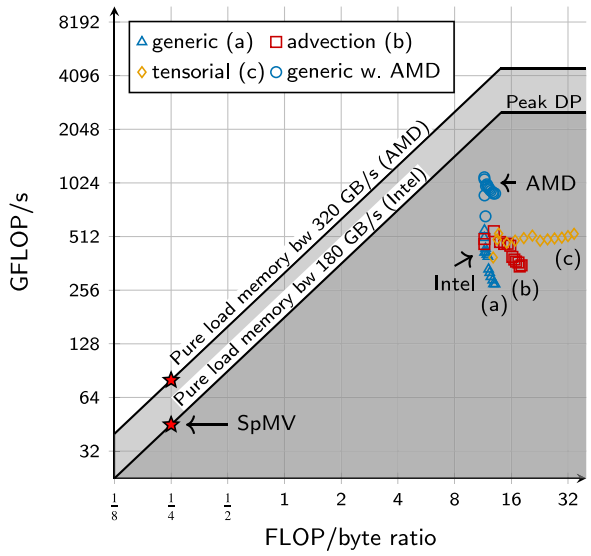


Fig. 7. Roofline performance model of the evaluation of the residual of the generic operator (Table 1) as well as the advection and tensorial operator (Table 8) for different number of components. On the Intel system, the bandwidth and the number of floating-point operations are extracted from hardware-performance counters. On the AMD system, where direct measurements have not been possible, the memory access of the Intel system is assumed as a model.

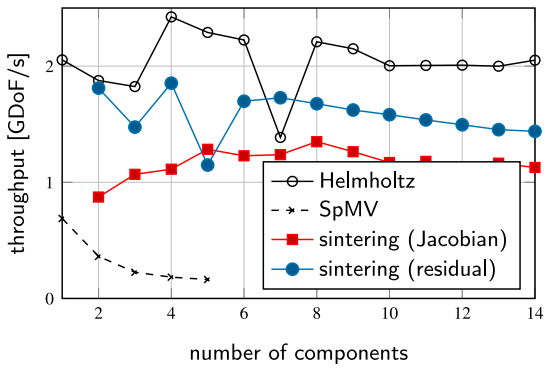


Fig. 8. Comparison of the throughput of application of the sparse matrix, of the vector Helmholtz operator, and of the generic sintering operator on the AMD hardware.

- Those coupling terms that cannot be reformulated and, as a consequence, introduce a quadratic complexity are explicitly optimized.

Further examples are discussed in Section 7 for tensorial mobility, which requires more transformations due to a higher arithmetic load. In particular, that section contains an example for the last bullet point.

Remark. We have experimented with advanced techniques to increase the throughput of the cell integrals, namely (1) interleaving the evaluation/integration with the loop over the quadrature points by performing quadrature in two-dimensional layers at a time [47] and (2) implementing the arithmetic work for 2^d cells at once. These techniques allow, e.g., to decrease the size of the working set by additional cache blocking and to reduce the cost of indirect addressing when accessing the DoFs on cells. The preliminary results are promising with speedups between 50% and 100%, allowing for a reduction of the gap to the hardware bandwidth limit. Detailed investigations on these options and alternative vectorization strategies, e.g., vectorization over components [53], are deferred to future work.

Behavior on AMD and GPU hardware

The above-mentioned performance bottlenecks in terms of the number of components are partly due to the specific microarchitecture of the chosen Intel hardware. In order to better illustrate the performance capabilities on the expected trajectory of computer hardware evolution, we performed a node-level performance experiment on a more recent AMD Epyc 7713 processor. According to Fig. 8, the throughput is on average 2.3× higher than the Intel results from Fig. 6, which is related to the higher bandwidth and compute performance (1.8× each) as well as to larger-bandwidth caches. Our code utilizes around 50% of the available memory bandwidth for a low number of components and around 40% for a higher number of components, as indicated by the roofline performance model (Fig. 7). This shows that the proposed algorithms also work, as expected, on alternative CPU-based hardware, however, the dependency on the number of components is less prominent due to a more balanced cache system of the AMD hardware.

The present work focuses on CPU implementations. However, fast (matrix-free) operator evaluation is also attractive on GPUs. Refs. [44, 51] discuss optimizations for some standard operators. For the sintering operator, the need to compute significantly larger amount of data at the quadrature points limits the applicability of current algorithms, since many implementations rely on keeping most intermediate data in registers rather than in caches used for CPUs. However, we believe that the novel algorithm design to organize work along 2D layers, as described above, combined with moving the remaining data into shared memory, as proposed for high orders in [56], could be beneficial.

Application in Jacobian-free methods

We conclude this subsection by discussing an efficient implementation of a Jacobian-free method (6) in the case of the fast operator evaluation (8) for the residual. In order to evaluate a Jacobian-free operator n times, we need to evaluate the residual $n+1$ times. During the first iteration, the residual at the linearization point $F(\mathbf{u})$ has to be evaluated once, which can be reused in the subsequent operations. Within the actual Jacobian-free evaluation, the linearization point has to be perturbed, the residual is evaluated, and the finite difference needs to be taken. We propose to merge the two vector-update steps into the loop over cells and perform them on ranges of the vectors only when the value is needed for the cell integral (`pre`) and once contributions from all cells have been added to an index (`post`), as also has been done to accelerate conjugate-gradient solvers [57] and additive Schwarz solvers [58]. In order to minimize the memory footprint, we perturb the vector containing the linearization point and revert the modifications afterwards. The resulting `pre` und `post` operations are as follows:

$$\underline{\textbf{pre}}\colon\ u_i^{\text{lin}} \leftarrow u_i^{\text{lin}} + \beta p_i,\quad \underline{\textbf{post}}\colon\ \begin{cases} v_i \leftarrow (v_i + r_i^{\text{lin}})/\beta \\ u_i^{\text{lin}} \leftarrow u_i^{\text{lin}} - \beta p_i, \end{cases}$$

which are run interleaved with $v \leftarrow F(u^{\text{lin}})$. This implies 6 additional floating-point operations and – under the assumption that the vector entries are still in cache – one extra write operation per DoF besides those in Table 1. In order to obtain the parameter β, our current implementation computes the l_2-norm of the source vector, implying, in addition, one global vector reduction step with the associated data access before the cell loop.

4.3. Block preconditioner

In the following, we propose a preconditioner of the Jacobian (7). The preconditioner is needed when we apply the Jacobian directly or via a finite-difference approach. In the current work, we consider incomplete LU factorizations that are of block-Jacobi type across MPI processes, in short, ILU-based block preconditioners. In order to compute an ILU, we need access to the entries of the underlying sparse matrix. To avoid the cost for memory access, we propose to set up ILU for the Cahn–Hilliard block and a single Allen–Cahn block, applying the

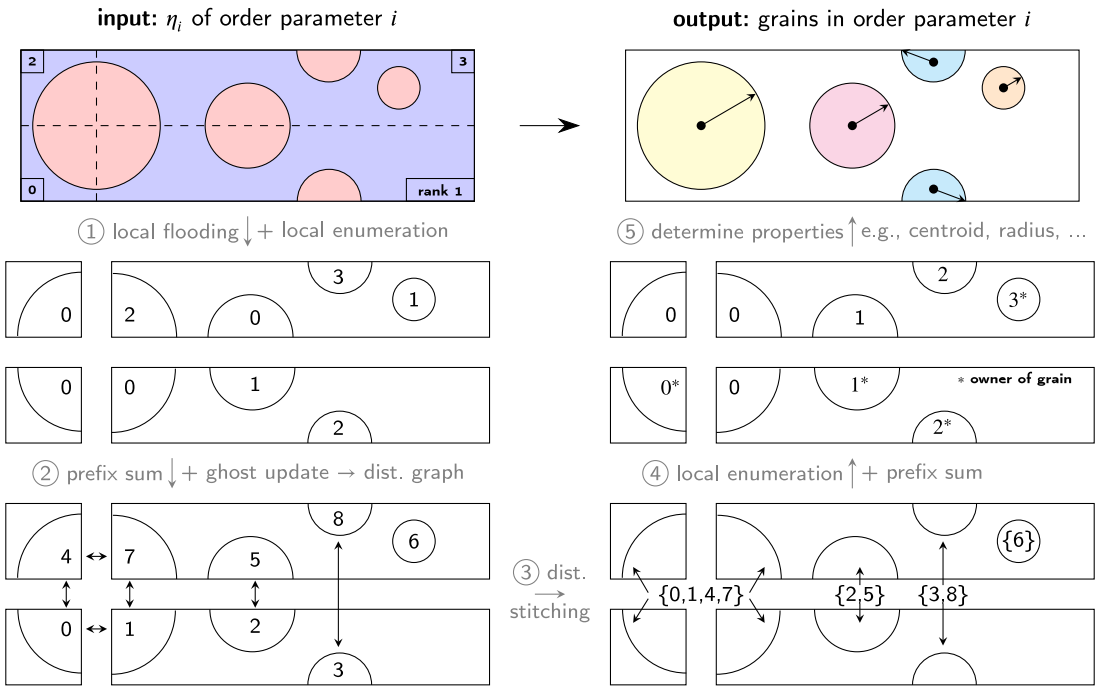


Fig. 9. Visualization of the fully distributed determination of grains within the same order parameter, starting with a discrete concentration field η_i. The results are (1) the grain index of each locally owned cell and (2) grain properties, e.g., centroid and bounding radius. In the example, the mesh is partitioned among four processes. For the sake of simplicity, the mesh is not shown and isocontours in η_i are perfectly circular. Grain indices are constant within cells. Steps ③ and ④ can be interpreted as a distributed connected-component algorithm that returns, for each node, a component number.

latter to each block, as indicated in the rightmost panel of Fig. 3. The setup is of constant complexity $\mathcal{O}(N_P)$, and the application naturally has a linear complexity $\mathcal{O}(N_c N_P)$. The runtime cost of the preconditioner applications can be reduced if the application is *batched* in a matrix-multivector form, thus only loading the entries of the factorized matrix once. To be able to reuse the ILU instance for each Allen–Cahn block, we take the maximum of the term $\left(v_i, L\partial^2 f/\partial\eta_i^2 u_i\right)$ in (7c) over all order parameters.

Remark. We use an ILU for the two-by-two Cahn–Hilliard block. For better performance with larger time steps or higher diffusivity, physics-based preconditioners, like the one in [59], might reduce the iteration counts. We defer the investigation of such preconditioners to future work, since the Cahn–Hilliard block is independent of the number of order parameters and, in our experiments, is not the bottleneck ($2 \ll N_c$).

4.4. Fully distributed grain tracking and remapping

In the following, we propose a fully distributed version of the grain-tracking algorithm, which we need to guarantee a non-conflicting assignment of the order parameters according to the invariant described in Fig. 2. The algorithm is inspired by reference [17], which presents the grain-tracking implementation in MOOSE. The procedure (1) detects grains in each order parameter, based on the discrete values of η_i, (2) checks for conflicts within each order parameter, and (3) reassigns the grains with conflicts to existing or new order parameters, which involves also the copying of data in the solution vectors.

Our implementation is heavily graph-based and works only on locally owned cells, allowing the algorithm to scale to large problem sizes.

4.4.1. Grain detection

For each order parameter η_i, each process runs a flooding algorithm [60] on locally owned cells to identify all agglomerations of cells with $\eta_i > \eta_{\text{lim}}$ ($\eta_{\text{lim}} = 0.01$ in our simulations). An agglomeration either forms a whole grain itself or is a part of a grain if it is located at internal boundaries between the neighboring processes. The challenging task is to match agglomerations related to grains stretched over multiple processes. For this purpose, we propose the following algorithm, which is inspired by the unique enumeration of DoFs in the context of FEM [61] and is visualized in 5 steps in Fig. 9.

Initially, we locally enumerate all identified agglomerations and give them globally unique indices via a parallel prefix sum (steps 1 and 2 in Fig. 9). Once each agglomeration has a unique index, we communicate that information by a ghost-value update so that processes know the agglomeration index of each ghost cell. This information is enough to build a distributed graph with nodes being agglomerations and edges being connections determined based on the information from the ghost cells. By determining *all connected components*, we get all agglomerations (and cells) that make up a grain (distributed stitching, step 3 in Fig. 9). Finally, we give each grain a unique index and determine properties, like centroid or bounding radius and—optionally—force or torque (see Section 7.2), via parallel reduction operations (steps 4 and 5 in Fig. 9).

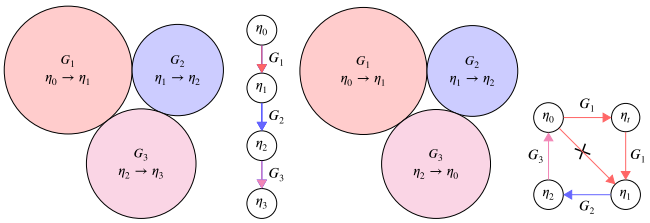
Note: The described algorithm also works for systems with periodic boundary conditions if it is possible to access ghost values across periodic boundaries (see the blue grain in Fig. 9). In addition, special care has to be taken during determination of properties, e.g., the grain centroids.

4.4.2. Grain tracking

After the grains have been detected for the current configuration, they need to be matched to those from the previous one. This is required to ensure no new grains have appeared, which would be an abnormal, not physically admissible behavior in the case of sintering. In order to determine, for a grain, the closest grain from the previous configuration, we build an R-tree, which allows a fast query.

4.4.3. Grain reassignment

Once all the grains have been identified and matched between the current and previous configurations, they are checked for collisions in a safety region $r_i' = r_i + 0.05 r_i$. If the safety regions of any two grains $i \neq j$ in the same order parameter overlap, we need to reassign one of



(a) Dependent remappings. (b) Cyclic remappings.

Fig. 10. Special remapping cases. Grain packings and remapping graphs.

them, since they might come into conflict in the next time steps. For reassigning, we use two algorithms.

A *greedy algorithm* checks, for a conflicting grain pair from order parameter i, whether any grain from all other order parameters $i \neq j$ would have a conflict with one of the two grains being currently under investigation. If not, one of the conflicting grains can be moved there, resolving the conflict. If no such order parameter could be found, the conflicting grain has to be assigned to a newly created order parameter or a more involved algorithm has to be used. In order to check whether a grain is in conflict with any grain in an order parameter, we use an R-tree data structure to accelerate the query.

Alternatively to the greedy algorithm, we use an algorithm based on *graph coloring*. In this case, grains are the nodes and conflicts are the edges. The graph coloring gives the minimum number of colors, which we interpret as the new order parameters. This approach, however, does not guarantee that grains keep their current order parameter so that remapping might become overly expensive even in the case of slight topological changes.

4.4.4. Grain remapping

The previous step has assigned grain G_i to an order parameter j: $\eta_j \to \eta_{j'}$ with possibly $j \neq j'$ so that grains in the same order parameter do not have any conflicts. Now, values corresponding to grain i in the current and previous solution vectors have to be transferred from the block corresponding to order parameter j to the one of j', without overwriting the values in the destination vector that still need to be moved.

Let us consider the two configurations in Fig. 10, each aiming to reassign three grains. While the remapping can be done straightforwardly with the sequence $(G_3\colon \eta_2 \to \eta_3)$, $(G_2\colon \eta_1 \to \eta_2)$, $(G_1\colon \eta_0 \to \eta_1)$ in the first case, this is not possible in the second case due to cyclic dependencies. Here, we need to introduce a temporary vector and can run the remapping, e.g., with the following sequence: $(G_1\colon\ \eta_0 \to \eta_t)$, $(G_3\colon\ \eta_2 \to \eta_0)$, $(G_2\colon\ \eta_1 \to \eta_2)$, $(G_1\colon\ \eta_t \to \eta_1)$, where η_t corresponds to the temporary vector.

In order to automatically detect and resolve possible remapping issues, we construct a directed graph containing all necessary remappings. We run a depth-first search algorithm to determine all connected nodes, i.e., grain-remapping operations that need to be serialized. If a cluster of connected nodes does not contain cycles, we can schedule the reassignment in topological order. However, if a cycle is detected, a dummy node corresponding to the temporal vector is introduced to break the cyclic dependency. For performance purposes, the described remapping procedure is applied within a single cell loop, based on cached resolved graphs.

Note: In our implementation, we are performing the graph/tree algorithms in the cases of grain stitching, matching, reassignment and remapping redundantly on each process, using either `boost.graph/boost.geometry` *[62] or* `Zoltan` *from the* `Trilinos` *library [63]. This implies that 0D grain information has to be replicated on each process. An extension to distributed graphs is possible but is deferred to future work to enable packings with more than 10k grains.*

4.5. Solver configuration

We conclude this section by summarizing the ingredients of our solver. We refer to the configuration of the solver as *default configuration* if the Jacobian is evaluated in a matrix-free way and all order parameters are considered on the cell level (Algorithm 1). When different variants are investigated, these are labeled accordingly.

4.5.1. Nonlinear and linear solver

In order to solve the nonlinear system (5), we use a Newton solver with cubic line search from the `NOX` package from the Trilinos library [63]. This nonlinear solver is run until the l_2-norm of the residual has been decreased by $\epsilon_{\text{nonlin}} = 10^{-5}$ and the Jacobian is solved with GMRES with a rather coarse relative tolerance of $\epsilon_{\text{lin}} = 10^{-2}$, in line with Brown [35] (see also Supplementary Material S3.2). The choice $\epsilon_{\text{nonlin}} = 10^{-5}$ as default tolerance for our nonlinear solver is the result of experiments over a wide range of tolerances and benchmark applications, ensuring sufficient accuracy in the mass conservation and the free energy of the discretized solution. A more detailed discussion on this matter with the elementary assessment of the conservation error in view of [64] can be found in Supplementary Materials S3.1 and S3.3. The preconditioner for GMRES is set up at the beginning of each time step once and is reinitialized between Newton iterations only if the number of linear iterations at a particular single Newton step exceeds the value of 50. We use the ILU implementation from the `Ifpack` package from the Trilinos library [63].

4.5.2. Time stepping

We use BDF2 with adaptive time steps. We increase the time-step size by 20% if the number of nonlinear iterations is less than 5 and the accumulated number of linear iterations is less than 100, i.e., less than 20 linear iterations per nonlinear iteration on average. We consider a time step as failed if the nonlinear solver needs more than 10 iterations or the linear solver more than 100 iterations in total. If the nonlinear solver has failed, we decrease the time-step size by 50% and rerun the time step, based on the old converged solution. The described time-stepping heuristic is conservative, since time-step sizes are not increased for challenging configurations, reducing the number of time steps that need to be repeated. Nonetheless, the obtained time-step sizes are quite large overall, leading to significant pressure on the linear solver and its preconditioner.

4.5.3. Initial mesh generation

The initial geometry is defined as a list of spherical particles. The definition of potentially overlapping particles can be supplied for the phase-field simulations from the preliminary DEM early-stage-sintering analysis [4]. The computational domain is then constructed based on a bounding box over all particles with a boundary padding $0.5r_{\text{max}}$, where r_{max} is the radius of the largest particle in the packing.

We choose the diffuse interface thickness a priori as $w = 0.1r_{\text{avg}} \dots 0.15r_{\text{avg}}$, where r_{avg} is the average radius of particles in the packing. The choice of w then defines the free-energy properties $A = (12\gamma_s - 7\gamma_{gb})/w$, $B = \gamma_{gb}/w$, $\kappa_c = 3w\left(2\gamma_s - \gamma_{gb}\right)/4$, $\kappa_\eta = 3w\gamma_{gb}/4$, based on a limit case analysis [24,30,31] with the physical surface γ_s and the grain boundary energies γ_{gb}, both usually given by the physics of the material of interest. Depending on the problem size and the desirable accuracy in capturing the interface motion, the thickness of the latter is discretized by 1–4 cells, giving the finest mesh size h_e desired only at the interface itself. The mesh is obtained by locally refining a coarse quad-/hex-only mesh. This mesh is constructed in such a way that cells have a good aspect ratio and the value h_e is approximately obtained in each direction by local refinement. For the latter, we use a forest-of-trees approach where cells are recursively replaced by 2^d child cells and rely on `p4est` [61,65]. The proposed strategy generates meshes with on average less than 10k and 100k scalar DoFs per particle for 2D and 3D simulations, respectively.

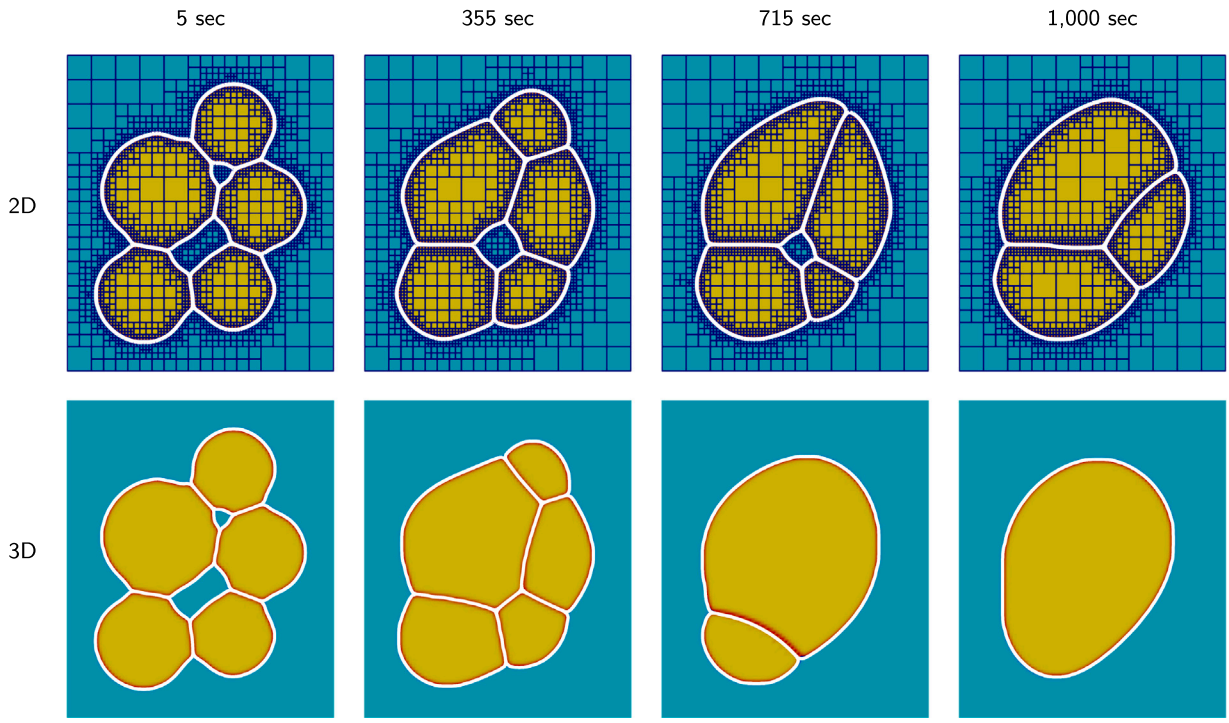


Fig. 11. 5 particles: microstructures obtained with MOOSE and with our code during simulations of the 5-particle sintering. The plots show $\sum_i \eta_i^2$ computed with MOOSE, and the white lines display the isosurfaces constructed for each $\eta_i = 0.5$ with `hpsint`.

4.5.4. Adaptive mesh refinement

The AMR algorithm is triggered every 10th time step or when the mesh quality has deteriorated in relation to the values of η_i; we only allow $|\max(\eta_i) - \min(\eta_i)| < 0.5$ for all cells. We coarsen cells if they are not close to the boundary of a particle, which is identified by $0.05 < \eta_i < 0.95$. At boundaries themselves, we keep the mesh fine throughout the simulation. The number of maximum/minimum refinements is a runtime parameter and is chosen depending on the material properties.

4.5.5. Grain tracking

We run the grain-tracking algorithm every 25 s of the simulated time or according to the same quality criterion as in the case of AMR described in Section 4.5.4. The initial distribution of grains over the order parameters is performed by running the graph-colorization algorithm that is also used at the reassignment step (see Section 4.4.3).

5. Numerical results

In the following, we present numerical results obtained with the developed solver, mainly to verify it from the sintering-physics point of view. We keep the discussion short and regard this section as an introduction into the setups of the performance analyses in Section 6.

5.1. 5-Particle case

We first analyze the 5-particle geometry depicted in Fig. 1; the centers and radii of the particles are listed in Table 2. The cubic computational domain of size $42 \times 50 \times 27$ in 3D is generated as described in Section 4.5.3. The free energy constants are defined as $A = 16$, $B = 1$ and the energy barriers are set to $\kappa_c = 1.0$, $\kappa_\eta = 0.5$. These values render the diffuse interface thickness $w \approx 1.0$. The following diffusion mobilities are chosen: $M_{\text{vo}} = 10^{-2}$, $M_{\text{va}} = 10^{-10}$, $M_{\text{s}} = 4.0$, $M_{\text{gb}} = 0.4$. The grain-boundary mobility is set to $L = 1.0$. This is a common set of parameters used in multiple works [22,25,30]. The packing is analyzed in 2D and 3D for the period up to $t_{\text{end}} = 1{,}000$ seconds. For this benchmark, we use a tighter tolerance $\epsilon_{\text{lin}} = 10^{-5}$ for the linear solver, matching the default value in MOOSE, in order to make sure that the coarser value of 10^{-2}, which we normally use as stated in Section 4.5.1, does not degrade the performance of MOOSE and, that way, ensures the comparison of the two codes. Note that no units are given except for time, since the geometry, energy and mobility properties for the current problem are defined as dimensionless.

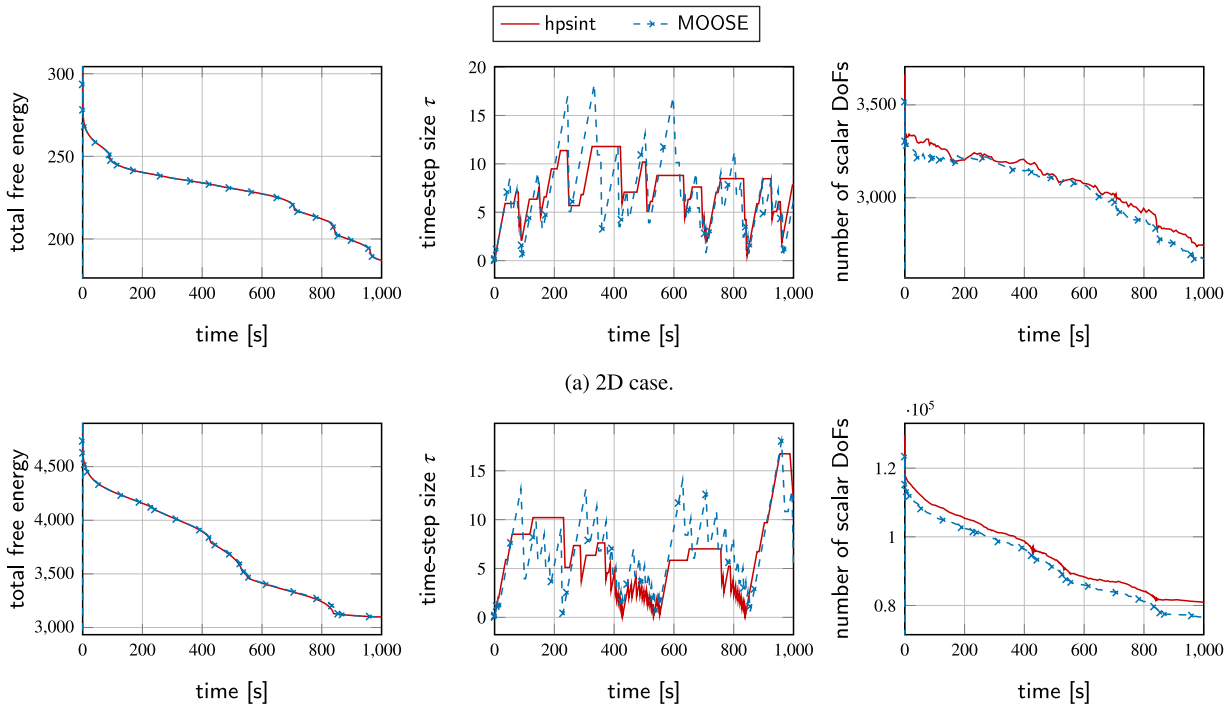
Table 2
5 particles: initial locations and radii of the particles.

ID	x	y	z	r
1	7.5	7.5	0	7.5
2	10	23.81	0	9.0
3	21.464	8.5	0	6.5
4	25.8	21.285	0	7.0
5	21.583	34.109	0	6.5

The aim of these simulations is to compare the results with an alternative, well-established implementation [26,30,66] of the same phase-field model available in project `Crow`,[5] which is based on the MOOSE framework [11]. `Crow` reuses most of the parts of the phase-field module of MOOSE, including the kernels that provide the mobility and free-energy terms as materials required by this particular sintering model. We carefully tuned the AMR settings, absolute and relative tolerances, iteration thresholds, implicit time-integration properties of both solvers such that the numbers of DoFs and of linear and nonlinear iterations are comparable in both codes. Fig. 11 shows meshes at different times in the 2D case. Other settings were set to be optimal for the MOOSE solver according to its documentation.[6] For instance, the preconditioned JFNK solution strategy provided by SNES from PETSc [34] was used and the parallel ILU implementation provided by the `Euclid` library from `hypre` [67] was chosen as preconditioner for the GMRES linear solver.

[5] https://github.com/SudiptaBiswas/Crow.

[6] https://mooseframework.inl.gov/source/systems/NonlinearSystem.html, https://mooseframework.inl.gov/modules/phase_field/Solving.html.



(a) 2D case.

(b) 3D case.

Fig. 12. 5 particles: comparison of total free energy, time-step size, and number of scalar DoFs over the simulation time.

Table 3
5 particles: performance metrics of the 2D and 3D simulations in `hpsint` and `MOOSE`. Scalar DoFs are given.

	2D		3D	
	`hpsint`	`MOOSE`	`hpsint`	`MOOSE`
Wall time [s]	164.8	704.4	1072	69,898
max τ	12.22	18.22	16.74	18.05
# time steps	190	223	303	241
# DoFs initial	3604	3504	128,700	123,209
# DoFs final	2751	2673	80,909	76,697

As can be seen in Fig. 11, the microstructures obtained for both solvers are identical for the 2D and 3D cases. For this purpose, the quantity $\sum_i \eta_i^2$ obtained in `MOOSE` is compared with the isolines constructed for each $\eta_i = 0.5$. For the 3D case, the corresponding plots are shown at the cross-section plane $z = 0$. As expected and previously demonstrated in [12], the 3D case exhibits faster microstructure evolution than the 2D simulations for the same material parameters. This is due to the added driving force from a second curvature term in 3D and also due to qualitative topological differences: for comparable geometries, those pores that are closed in 2D are usually open in 3D.

Fig. 12 shows the plots of the total free energy (2). The curves obtained in both codes reveal a gradual reduction of the total free energy. The comparison of its components (bulk and interfacial) along with some other model metrics is shown in Supplementary Material S3.1. Additionally, Fig. 12 presents the evolution of the total number of DoFs and the time-step size during the numerical simulations. In distinct contrast to `MOOSE`, our code does not always attempt to increase τ, since we target a predefined number of linear and nonlinear iterations; once that limit is exceeded, the time-step size is not enlarged. Confirming the design of the experiment, the numbers of DoFs in both codes are found to vary similarly.

The solver-related metrics[7] are summarized in Table 3. In particular, we show timings for the `MOOSE`-based and our implementation. The wall times for both codes were obtained in separate runs with output disabled. A speedup of 4.3/65.3 is visible for 2D and 3D. The numbers are primarily intended to show the significance of the proposed optimizations, as the comparison to the `MOOSE` code has to remain qualitative: even though its settings have been tuned to provide the maximum performance, we believe that better performance might be possible for developers with deeper insight into `MOOSE`.[8]

5.2. 332-Particle case

As a second validation benchmark, we compare the results of our simulations with those presented in [7]. The authors of that paper kindly provided us the initial packings used for their numerical analysis. The original focus of the discussion in [7] was on the influence of the rigid body motions on the sintered microstructures. Even though our code also implements the advection terms (see Section 7.2), we intentionally perform the comparison of results without advection and, thus, rigid body motions. Due to this reason, we run the sintering simulation of the packing containing 332 particles, for which a number

[7] The simulations were executed on a single node of the cluster at Helmholtz-Zentrum Hereon (dual-socket 24-core 2.1 GHz Intel Xeon Scalable Platinum 8160 processor; Skylake), using 24 processes for 2D and 48 for 3D.

[8] A small benchmark code based on `deal.II`, which evaluates (8) for the residual without exploiting the structure of the shape functions (tensor-product, same for each component, ...) and does not vectorize over cells, results in a $\approx 30\times$ lower throughput. The code path is not specialized for the evaluation of the residual but for easy-to-read and generic assembly of a sparse matrix, which normally has to be computed less frequently so that performance optimizations are not as crucial as if it would be done in each linear iteration. Since `libMESH` [68], the FEM backend of `MOOSE`, evaluates the residual similarly to the implementation of the slow path in `deal.II`, we are confident that the presented performance comparison with `MOOSE` is qualitatively reasonable.

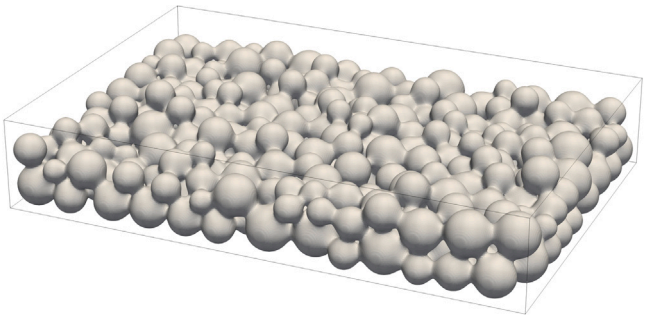


Fig. 13. 332 particles: the final configuration after sintering for 3.47 h.

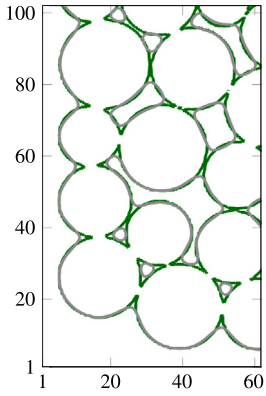


Fig. 14. 332 particles: contour lines on the $z = 10$ µm plane taken at $t = 0$ (green) and 3.47 h (gray).

of metrics is available in [7], referred to as "Case 3" in that publication. The numerical implementation in [7] is based on finite differences on a uniform grid and uses explicit time integration.

We use the same material parameters: the free-energy constants are set to $A = 32$, $B = 8$, $\kappa_c = 0.4$, $\kappa_\eta = 0.2$, and the diffusion mobilities are defined as $M_{\text{vo}} = 10^{-2}$, $M_{\text{va}} = 10^{-3}$, $M_{\text{s}} = 10.0$, $M_{\text{gb}} = 1.0$. The energy properties lead to the diffuse interface thickness $w \approx 0.18$. The same grain-boundary mobility $L = 100$ as in [7] is used, ensuring that the grain-growth effects do not dominate in the regime examined. Similarly to the original publication, we also solve the dimensionless form of the phase-field equations, using the same scaling parameters: $l = 5$ µm, $M_0 = 10^{-12}$ cm^2/s, and $t_{\text{ref}} = l^2/M_0 = 2.5 \times 10^5$ s. Given the physical sintering time $t_{\text{physical}} = 3.47$ h, the simulation time is $t_{\text{end}} = 0.05$. The initial time-step size is chosen as $\tau_{\text{initial}} = 10^{-3}$.

The computational domain is defined by the bounding box with dimensions of $-1.5 < x < 41.5$, $-1.5 < y < 61.5$, $-1.5 < z < 13$ such that the distance between each particle and the domain boundary is at least 1. The coarse mesh consists of $86 \times 126 \times 29$ cubic cells, and we perform two local-refinement steps to obtain the minimal cell size $h_e = 0.125$, which is comparable to the grid spacing $\Delta x = 0.1$ from [7]. This results in $\approx 6-7$ million scalar DoFs in our case in contrast to the ≈ 34 million points ($420 \times 620 \times 130$ cells) of the uniform mesh in [7]. A typical simulation of this test case with our code needs only 14 time steps when using the time-stepping strategy described in Section 4.5.2 and, on 16 nodes of SuperMUC-NG, runs in about 5 min (not counting the time required for generating the output).

In this particular simulation, we employ the conventional surface-mobility term [22] instead of the one from (4) in order to achieve a better agreement with the original results from [7]. To alleviate the arising convergence issue due to the non-smooth coefficient, we apply the Jacobian-free formulation (6).

In order to postprocess and visualize the results, the physical domain is used. Fig. 13 presents the final configuration of the packing,

Table 4
Numbers of order parameters, cells and scalar DoFs for 51–10,245-particle packings.

N_{grains}		51	102	212	316	603	1370	3760	6140	10,245
N_{op}		9	10	10	11	10	11	11	11	11
N_{cells}	[×1e6]	3.0	5.7	10.4	18.8	34.0	53.5	153.2	325.5	589.1
N_{DoFs}	[×1e6]	3.4	6.4	11.7	20.9	37.9	59.5	170.2	360.8	651.7

and Fig. 14 shows the microstructure view at the cross-section plane $z = 10$ µm. Visually, both images are in close agreement with the analogous Figs. 7(a) and (c) presented in [7], without noticable differences.

To perform a quantitative comparison, we also compute the microstructure metrics

$$\text{solid-volume fraction} \quad \int_\Omega c \, \mathrm{d}\Omega, \tag{9a}$$

$$\text{surface area} \quad \sum_i \int_{\Gamma_i} 1 \, \mathrm{d}\Gamma_i \text{ w. } \Gamma_i = \{p \in \Omega | \eta_i(p) = 0.5\}, \tag{9b}$$

$$\text{grain-boundary area} \quad \sum_i \int_{\hat{\Gamma}_i} 0.5 \, \mathrm{d}\hat{\Gamma}_i \text{ w. } \Gamma_i = \{p \in \Gamma_i | \eta_i \eta_j > 0.14 \; \exists j \neq i\} \tag{9c}$$

in the control volume $5 < x < 195$ µm, $5 < y < 295$ µm, $2.5 < z < 35$ µm as in [7]; the corresponding graphs are shown in Fig. 15. The curves do not match perfectly. For the solid-volume fraction, the mismatch is, in fact, negligible given the scale of the y-axis. For the remaining two quantities, the difference is more tangible but still does not exceed 3% for the surface and 10% for the grain-boundary areas, respectively, and, is most probably related to the details in the implementation of the postprocessors, which extract the isosurfaces. Despite of the differences in the values, the slopes of the curves are in a very good agreement, meaning that the dynamics of the sintering processes in both simulations are the same.

6. Performance

In the following, we analyze the performance of the implementation of our solver in detail from a holistic point of view. This is needed because of the multi-faceted challenges of the solution of sintering processes, in which it is not enough to optimize and analyze the (matrix-free) linear operator evaluation, as we did in Section 4.2. For example, the preconditioner choices implied by the matrix-free solver design need to be assessed in terms of the total solution time.

This section is divided into two parts. We start by studying the performance of the solver for a moderate number of grains (51; far left in Fig. 16), where the focus is on the influence of different variations of the solver. Next, we analyze the parallel scalability of the code by increasing the number of particles up to 10k and using up to 50k processes on a supercomputer.

All packings considered in the current section are cubic and shown in Fig. 16. Bounding boxes of different sizes are used to extract smaller packings containing 51 … 6,140 particles from the largest one having 10,245 grains within a control volume of size $1399 \times 1400 \times 1318$ (µm^3). The latter has been obtained by the preliminary DEM simulations performed with the package `Yade` [69], following the procedure proposed in [4], which was designed to deliver an isotropic initial configuration with irregular, realistic distributions of particles having relatively low porosity. The particle-size distribution of the largest packing is shown in Fig. 16.

In contrast to Section 5.1, larger powder particles are considered in this section. For this reason, the energy parameters are defined as $A = 4.35$, $B = 0.15$, $\kappa_c = 9.0$, $\kappa_\eta = 1.79$. These values yield a thicker diffuse interface with $w = 4$ for the surface and grain-boundary free-energy values $\gamma_s = 1.8$ and $\gamma_{gb} = 0.6$. The diffusion mobilities are defined as $M_{\text{vo}} = 10^{-2}$, $M_{\text{va}} = 10^{-3}$, $M_{\text{s}} = 4.0$, $M_{\text{gb}} = 0.4$, and the grain-boundary mobility is set to $L = 1.0$. The initial number of divisions per interface thickness is chosen as $\zeta = 3$. The initial numbers of scalar

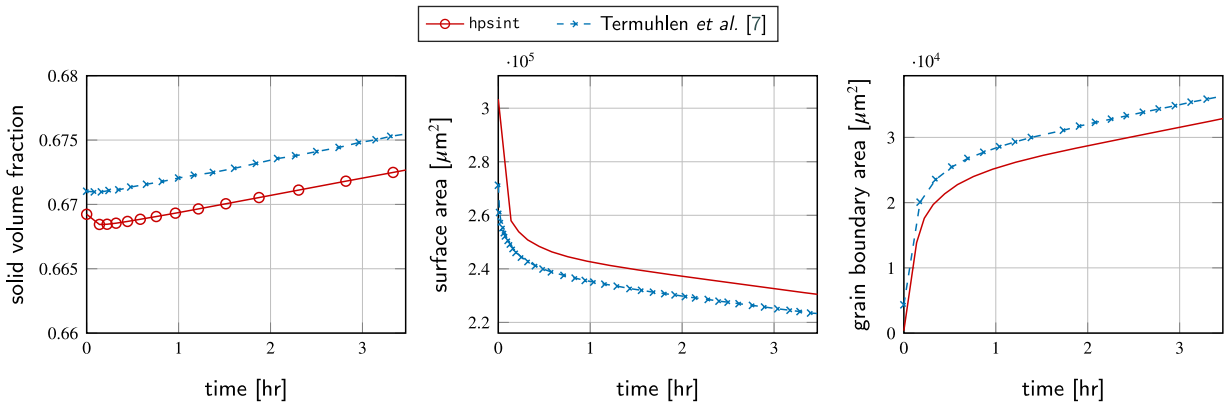


Fig. 15. 332 particles: comparison of relevant microstructure metrics according to (9).

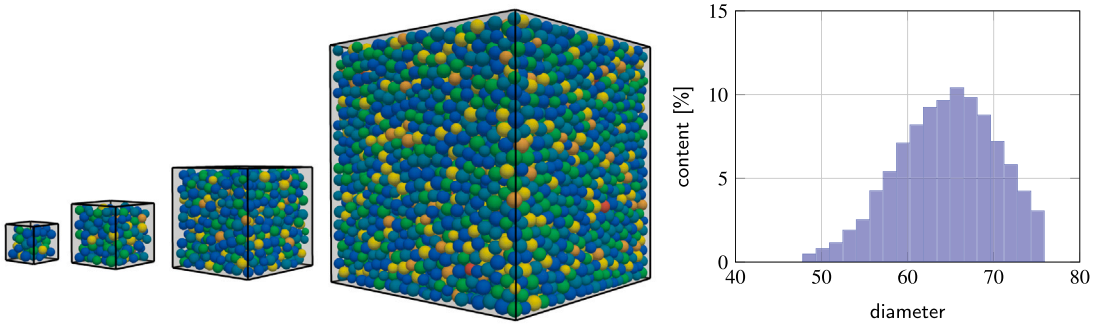


Fig. 16. Left: Particle packings considered for the performance study in Section 6: 51, 212, 1037, 10,245 particles (not shown: 102, 316, 603, 3076, 6140 particles), with colors indicating the order parameters. Right: Distribution of particle diameters for the packing containing 10,245 particles.

Table 5
51 particles (up to $t = 500$): Comparison of different solver configurations.

Configuration	$\tau_{\max}$	N_T	$\overline{N}_N$	$\overline{N}_L$	$\overline{N}_R$	N_T^F	N_L^F	T	$\overline{T}_J$	$\overline{T}_{P,setup}$	$\overline{T}_{P,apply}$	$\overline{T}_R$
Default	**19.78**	**51**	**4.6**	**25.3**	**1.2**	**0.0%**	**0.0%**	**463.2**	**0.027**	**1.279**	**0.013**	**0.028**
JF (Jacobian-free)	19.78	51	4.6	25.7	1.2	0.0%	0.0%	444.7	0.023	1.277	0.013	0.028
cut-off	19.78	51	4.6	25.4	1.2	0.0%	0.0%	353.9	0.009	1.269	0.013	0.009
JF + cut-off	19.78	51	4.6	25.3	1.2	0.0%	0.0%	356.1	0.008	1.272	0.013	0.009
tensorial	16.48	73	4.4	16.4	1.2	4.1%	5.7%	551.4	0.040	1.253	0.014	0.036
tensorial + JF	19.78	59	4.5	19.5	1.2	1.7%	2.2%	450.8	0.028	1.239	0.014	0.032
tensorial + cut-off	16.48	60	4.6	19.8	1.2	0.0%	0.0%	355.3	0.010	1.192	0.013	0.010
advection + JF	19.78	56	4.4	24.4	1.2	1.8%	2.7%	523.0	0.025	1.262	0.014	0.065
advection + JF + cut-off	19.78	56	4.5	24.4	1.2	1.8%	2.6%	434.1	0.010	1.238	0.014	0.025

$\tau_{\max}$: maximal time-step size, N_T: number of time steps, $\overline{N}_N$: average number of nonlinear iterations per time step, $\overline{N}_L$: number of linear iterations per nonlinear iteration, $\overline{N}_R$: number of residual evaluations per nonlinear iteration, N_T^F: fraction of repeated time steps, N_L^F: fraction of linear iterations of repeated time steps, T: total runtime in seconds, $\overline{T}_J$: average time for application of Jacobian, $\overline{T}_{P,setup}$: average setup time of the preconditioner, $\overline{T}_{P,apply}$: average time for application of the preconditioner, $\overline{T}_R$: average time for evaluation of the residual

degrees of freedom generated for each of the packings by using such settings are shown in Table 4.

For time marching, the BDF2 scheme is used with the initial time-step size $\tau = 0.1$ and its growth is limited by the maximum value $\tau_{\max} = 100$.

6.1. 51 Particles

We start with the investigation of a 51-particle packing. All experiments are run on 4 Intel Cascade Lake Xeon Gold 6230 nodes with a total of 160 processes. In this section, each experiment has been run once and we average the results over at least 50 time steps, 230 nonlinear iterations, and 5200 linear iterations.

6.1.1. Default configuration

The line "default" in Table 5 shows the solver statistics obtained over a complete simulation up to $t = 500$ ($\tau_{\max}$, N_T, $\overline{N}_N$, $\overline{N}_L$, $\overline{N}_R$, T, $\overline{T}_J$, $\overline{T}_{P,\text{setup}}$, $\overline{T}_{P,\text{apply}}$, $\overline{T}_R$; see Section 3). The maximum time-step size achieved is 19.78. The average numbers of nonlinear and linear iterations are about 4.6 and 25.3, respectively. These numbers lie in the expected range, given the control parameters of the adaptive time stepping according to Section 4.5.2.

Similarly, Table 7 shows the data for $t = 15{,}000$. One can see that the nonlinear solver fails to converge for certain time steps, requiring a second attempt with decreased time-step size. However, their number and the resulting wasted (non)linear iterations are rather small and do not exceed 6%.

For the long simulation, Fig. 17 shows the time share of different parts of the code. 80% of the time are spent on solving the Jacobian. From this linear solver process, 46% are spent on applying the

Table 6
51 particles (up to $t = 500$): Comparison of different preconditioners. The default configuration uses `ILU + sILU (max)`.

Preconditioner name	$\tau_{\max}$	N_T	$\overline{N}_N$	$\overline{N}_L$	$\overline{N}_R$	N_T^F	N_L^F	T	$\overline{T}_J$	$\overline{T}_{P,setup}$	$\overline{T}_{P,apply}$	$\overline{T}_R$
Default	**19.78**	**51**	**4.6**	**25.3**	**1.2**	**0.0%**	**0.0%**	**449.6**	**0.024**	**1.265**	**0.013**	**0.026**
ILU[a]	23.74	56	4.4	22.3	1.2	1.8%	6.3%	2495.0	0.022	22.446	0.140	0.024
ILU + ILU	23.74	54	4.5	21.9	1.2	1.9%	6.3%	2141.0	0.026	19.037	0.128	0.028
ILU + bILU	19.78	50	4.4	24.7	1.2	0.0%	0.0%	530.8	0.025	2.890	0.019	0.027
ILU + sILU (none)	5.52	112	4.3	29.9	1.2	0.0%	0.0%	1009.0	0.024	1.047	0.013	0.026
ILU + sILU (avg)	19.78	53	4.5	24.8	1.2	1.9%	2.8%	478.9	0.027	1.315	0.014	0.028

[a] 5 compute nodes used due to increased memory consumption of the sparse matrix needed to set up the preconditioner.

Table 7
51 particles (up to $t = 15{,}000$): Comparison of different preconditioners. The default configuration uses `ILU + sILU (max)`.

Preconditioner name	$\tau_{\max}$	N_T	$\overline{N}_N$	$\overline{N}_L$	$\overline{N}_R$	N_T^F	N_L^F	T	$\overline{T}_J$	$\overline{T}_{P,setup}$	$\overline{T}_{P,apply}$	$\overline{T}_R$
Default	**25.52**	**1043**	**4.8**	**26.0**	**1.2**	**3.5%**	**4.6%**	**10 240.0**	**0.027**	**1.117**	**0.014**	**0.027**
ILU + bILU	27.43	990	5.3	27.4	1.5	2.5%	5.3%	12 710.0	0.025	2.785	0.019	0.029
ILU + sILU (avg)	24.57	1113	4.7	25.8	1.2	3.7%	4.7%	10 790.0	0.028	1.133	0.014	0.029

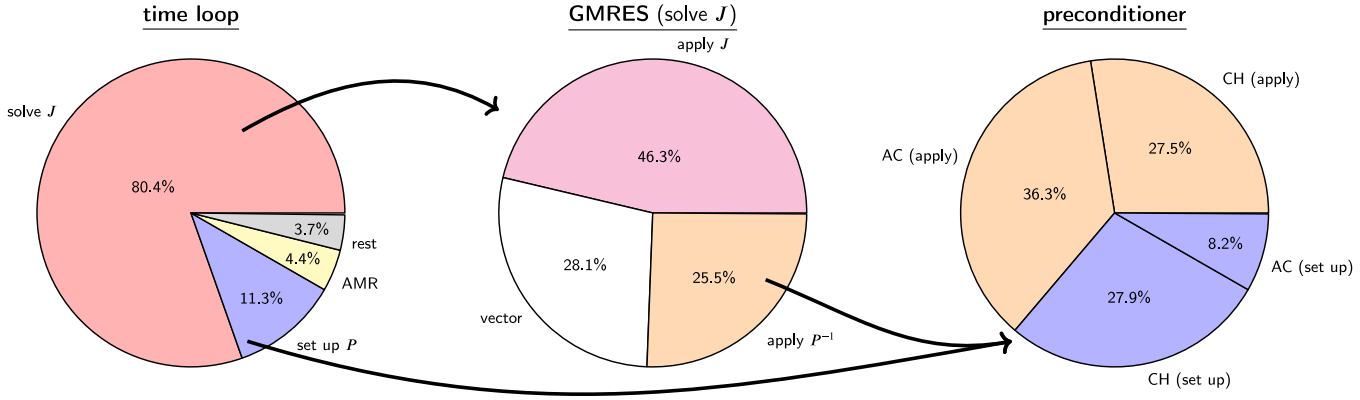


Fig. 17. 51 particles (up to $t = 15{,}000$): time share of different parts of the code.

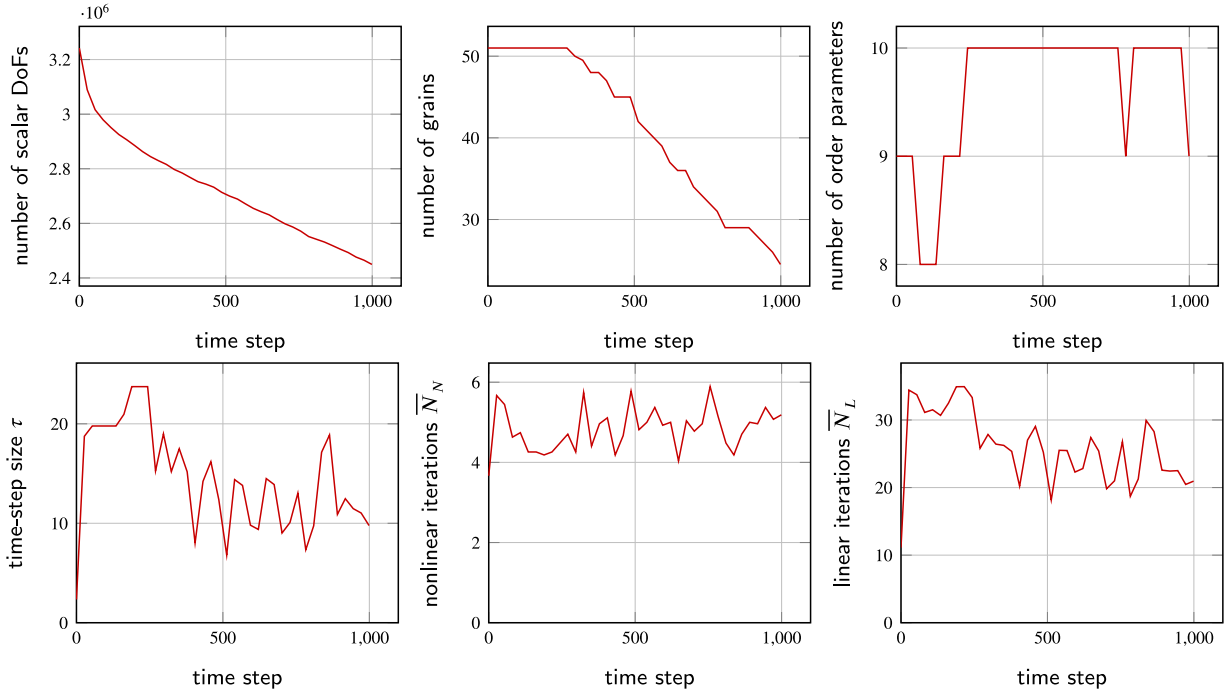


Fig. 18. 51 particles (up to $t = 15{,}000$): number of scalar DoFs, number of grains, number of order parameters, time-step size τ, nonlinear iterations $\overline{N}_N$, and linear iterations $\overline{N}_L$ over time. The values are averaged over ranges of 30 time steps.

Jacobian, 26% on the application of the preconditioner and the rest on vector operations within GMRES. About 11% of the time is spent on setting up the preconditioner, which is dominated by the setup of the preconditioner for the 2 × 2 Cahn–Hilliard block. Fig. 18 shows relevant quantities (number of DoFs, number of grains, number of order parameters, time-step size, number of nonlinear and linear iterations) over time. The number of DoFs is decreasing during the simulation, which is correlated with the disappearance of grains. The number of order parameters does not exceed 10 due to the developed grain-tracking algorithm. On average, the number of order parameters is 9.5. The number of nonlinear and linear iterations is kept at a constant value by our strategy to increase/decrease time-step sizes and resetting the preconditioner. The time-step size shows a strong variation over time, primarily driven by the high dynamics when new necks are forming between particles.

6.1.2. Alternative preconditioners

In the following, we compare the proposed preconditioner (`ILU + sILU (max)`) with the following alternatives, motivated by the block sparsity pattern in Fig. 3. (1) `ILU`: set up ILU for the complete Jacobian; (2) `ILU + ILU`: set up ILU for the Cahn–Hilliard block and the Allen–Cahn block; (3) `ILU + bILU`: set up ILU for the Cahn–Hilliard block and for each scalar Allen–Cahn block, ignoring the coupling between the Allen–Cahn blocks. (4) In addition to taking the maximum of $\left(v_i, L\partial^2 f/\partial\eta_i^2 u_i\right)$ in `ILU + sILU (max)`, we also consider ignoring this problematic term (`none`) and taking the average value over all order parameters (`avg`).

Table 6 evaluates the solver efficiency – for a simulation up to time $t = 500$ – for the different preconditioners. We note that the choice of the preconditioner does not influence the accuracy of the solution, which is primarily controlled by the nonlinear tolerance. It is clear that `ILU` and `ILU+ILU` are inherently expensive due to quadratic complexities in both setup costs and memory consumption. Considering only the blocks on the diagonal of the Allen–Cahn block (`ILU+bILU`) improves the situation without significantly affecting the convergence of the overall solver. In order to reduce the setup costs, the reuse of the preconditioner of one Allen–Cahn block helps. Our observation is that `ILU+sILU(avg)` and `ILU+sILU(max)` are able to cut down the setup costs of the preconditioner by a factor of two and the costs of applying by 30%, since the ILU instance has to be loaded only once due to the batched application.[9] Overall, the performance can be improved by 20%. `ILU+sILU(none)`, however, results in a significant drop in applicable time-step sizes, leading to an increase in the required overall solution time. For the best three preconditioners, Table 7 shows the results obtained for simulations up to $t = 15{,}000$, indicating that the scalar preconditioners are also applicable to later stages of sintering simulations and that the proposed variant `ILU + sILU (max)` is indeed the fastest overall.

6.1.3. Matrix-free vs. Jacobian-free evaluation

Table 5 (row "JF") also evaluates the efficiency of the Jacobian-free approach. The number of linear iterations is growing moderately. In total, a matrix-free approach that evaluates the Jacobian exactly and a Jacobian-free approach result in similar solution times, with slight advantages in favor of the Jacobian-free approach due to lower costs of residual evaluation without loss of accuracy. This observation allows users to skip explicit linearization of the weak form as well as to avoid the implementation and its cumbersome performance optimization. Nevertheless, a fast parameter-free approach (we still need to identify β for the Jacobian-free case, see Section 4.2) might be crucial and useful in practice.

[9] Please note that the costs of setting up the Cahn–Hilliard 2 × 2 block are now dominating.

6.1.4. Relevant grains on cells

Table 5 (rows "cut-off" and "JF + cut-off") shows the impact of considering only relevant grains on a cell level. In the case of matrix-free and Jacobian-free operator evaluations, the costs can be cut down by a factor of 3.0 and 2.9, respectively. Considering that the operator evaluation only takes half of the overall runtime, the solver speedup is rather modest at 30% or 25%, respectively. These results are a strong motivation for adopting the cut-off strategy to other parts of the code in future work. For example, the design of a suitable variation of the chosen preconditioner is not straightforward without manually rewriting sparse matrix kernels. We refer interested readers to Supplementary Material S3.4 for the results of experiments regarding the influence of the cut-off tolerance on the model accuracy.

6.2. Scaling up to 10k particles

We conclude this section by presenting scaling results. For this purpose, we run packings with 51, 102, 212, 316, 603, 1037, 3076, 6140, and 10,245 particles on 8, 16, 32, 64, 128, 256, 512, and 1024 compute nodes on SuperMUC-NG. In each case, we perform the simulation for a fixed number of 10 time steps: the grain tracking is triggered once, at the first step. We run all experiments three times and report the best timings. The statistical distribution is close to the minimal time within a few percent.

Fig. 19 shows the scaling behavior of the most important ingredients for our solver. Overall, it is clear that the solver (linear solver and setup of preconditioner) scales well up to 512 compute nodes. The performance drops significantly when going to 1k nodes, which is related to an unexpected spike of setup costs of ILU in `Trilinos`.

Alternative plotting of the data in Fig. 20 indicates excellent parallel efficiency over large ranges in the case of the linear solver: for sufficiently large problem sizes per process, one can increase the number of processes by a factor of 16–32 with only a loss of 25% in parallel efficiency. Furthermore, the lowest times to solutions are reached for about 10k DoFs per process.

The cost of the grain-tracking algorithm increases linearly with the number of processes. This is related to the fact that the graphs are gathered during stitching. Since the costs are rather small compared to other parts of the code, we defer the investigation of the grain-tracking algorithm to future work, for which we plan to adopt algorithms based on distributed graphs.

7. Extensions

So far, we have considered a basic phase-field approach for simulating solid-state-sintering processes. More involved physical models involve additional computations (see Section 2.1). Since we do not assemble matrices, where one would only pay the cost of additional computations during assembly and the cost of the application of the matrix is independent of the depth of the physical model as long as the sparsity pattern is not changing, additional computations could limit the throughput in a matrix-free implementation. Also the choice of the physical model might influence the preconditioner selection. In the following, we investigate these points for two common physical extensions: (1) tensorial mobility and (2) advection terms (rigid body motions).

7.1. Extension 1: tensorial mobility

In the basic model (1), the mass fluxes are defined by the scalar mobility term (4). Despite its simplicity, this term is not the best choice if a rigorous treatment of the surface and the grain-boundary fluxes is of primary interest, as shown in [23]. In this case, the tensorial form

$$\mathbf{M} = M_{\mathrm{vo}}\phi\mathbf{I} + M_{\mathrm{va}}(1-\phi)\mathbf{I} + 4M_{\mathrm{s}}c^2(1-c)^2\mathbf{T}_s + M_{\mathrm{gb}}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\eta_i\eta_j\mathbf{T}_{ij} \quad (10)$$

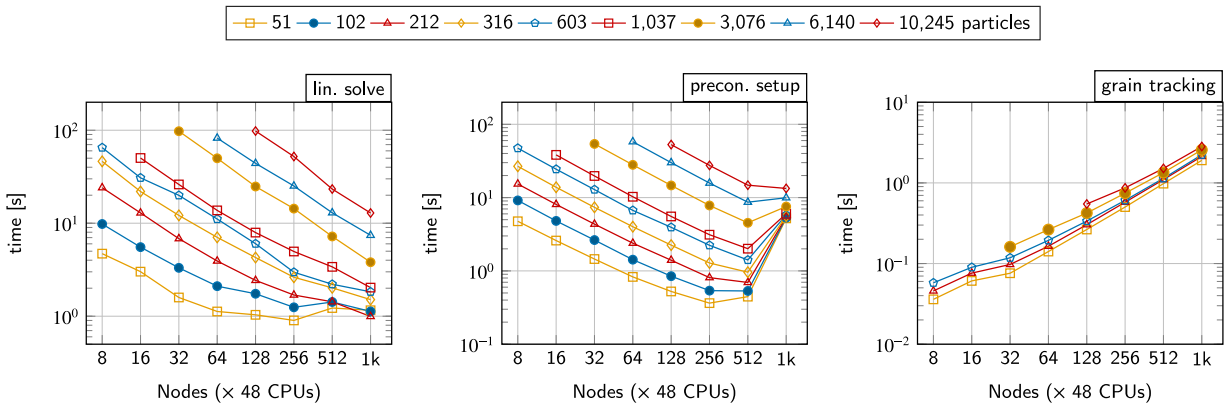


Fig. 19. Scaling of different and computationally most expensive parts of the solver for 51–10,245 particles and 10 time steps. Times are accumulated over all time steps and include both computations and communications. The data for "grain tracking" is only shown for a selected set of particles (51, 212, 603, 3076, 10,245 particles).

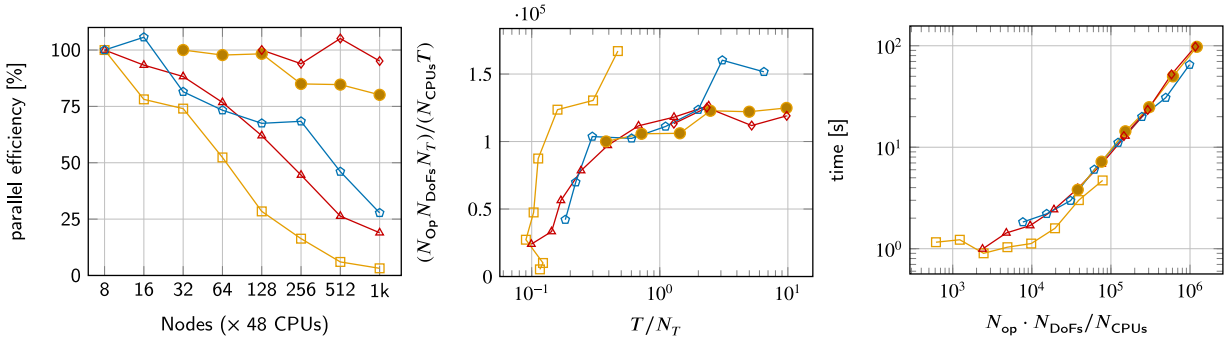


Fig. 20. Detailed scalability analysis of linear solver (see Fig. 19). The data is only shown for a selected set of particles (51, 212, 603, 3076, 10,245 particles).

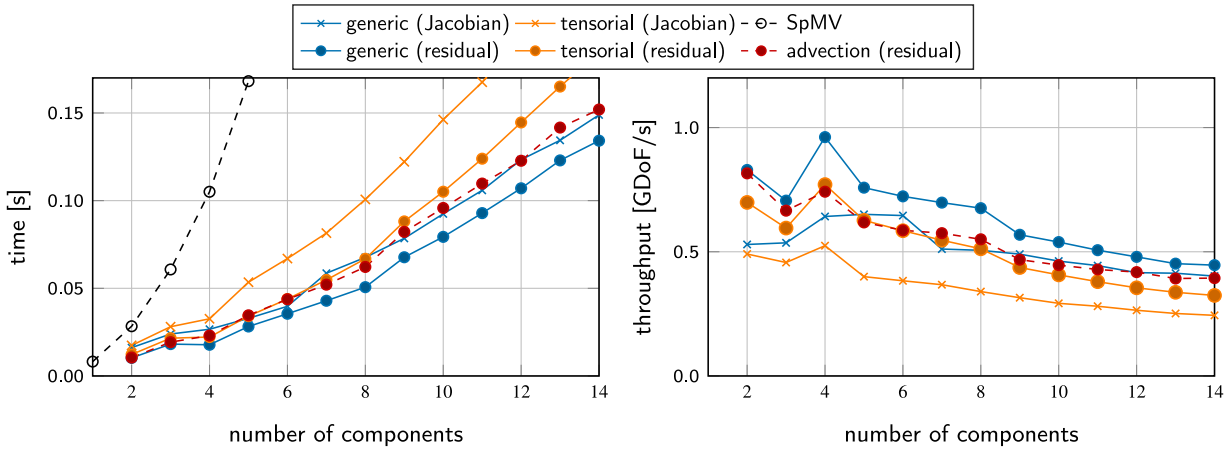


Fig. 21. Comparison of different sintering operator variants for different number of components: generic (Section 2.2), tensorial (Section 7.1), and advection operator (Section 7.2).

can be considered. This definition restrains the surface and the grain-boundary atomic fluxes to be strictly tangent to the corresponding diffusion paths, by introducing the following tensors:

$$\mathbf{T}_s = \mathbf{I} - \mathbf{n}_s \otimes \mathbf{n}_s,$$

$$\mathbf{T}_{ij} = \mathbf{I} - \mathbf{n}_{ij} \otimes \mathbf{n}_{ij},$$

where the unit normal vector to the grain surface is computed as

$$\mathbf{n}_s = \frac{\nabla c}{|\nabla c|}, \tag{11}$$

and the unit normal vector between the phases i and j is computed from the gradients of the corresponding order parameters as

$$\mathbf{n}_{ij} = \frac{\nabla \eta_i - \nabla \eta_j}{\left|\nabla \eta_i - \nabla \eta_j\right|}. \tag{12}$$

Here, symbol $\otimes$ denotes a tensor product.

The definition of the Jacobian given by (7) also holds for the present case. Of course, the derivatives of the mobility differ, as detailed in Supplementary Material S2. Since the modified mobility only influences the rows of the Jacobian related to the Cahn–Hilliard block and does not influence the Allen–Cahn blocks, we can adopt the preconditioner from Section 4.3 also in the current context.

Table 8
Comparison of different sintering operator variants for different number of components (see Fig. 21). Table 1 contains the data for the generic implementation.

N_c+2	tensorial (residual)				tensorial (Jacobian)				advection (residual)			
	D/s	r/D	w/D	F/D	D/s	r/D	w/D	F/D	D/s	r/D	w/D	F/D
2	0.70	5.3	1.3	706	0.49	14.0	1.5	1055	0.82	5.3	1.3	608
3	0.60	4.9	1.5	656	0.46	13.6	1.6	941	0.67	6.1	1.5	698
4	0.77	4.8	1.5	690	0.52	13.3	1.7	1024	0.74	5.6	1.6	739
5	0.63	4.6	1.6	753	0.40	13.2	1.7	1136	0.62	5.3	1.6	772
6	0.58	4.6	1.6	830	0.38	13.2	1.8	1278	0.59	5.1	1.6	794
7	0.55	4.5	1.6	918	0.37	13.1	1.8	1437	0.57	5.0	1.6	811
8	0.51	4.5	1.7	1012	0.34	13.0	1.8	1607	0.55	4.9	1.7	827
9	0.44	4.4	1.7	1111	0.32	13.0	1.9	1805	0.47	4.8	1.7	840
10	0.41	4.4	1.7	1212	0.29	13.0	1.9	1988	0.45	4.7	1.7	853
11	0.38	4.3	1.7	1316	0.28	13.0	2.0	2175	0.43	4.6	1.7	864
12	0.35	4.3	1.7	1421	0.26	12.9	2.0	2366	0.42	4.5	1.7	875
13	0.34	4.2	1.7	1528	0.25	12.9	2.1	2558	0.39	4.5	1.8	886
14	0.32	4.2	1.8	1636	0.24	12.9	2.1	2752	0.39	4.4	1.8	896

Additional challenges in comparison to the scalar-mobility case are: (1) in the application of the Jacobian, one needs the gradient of η_i at the linearization point in addition to those of c and μ, as required in the scalar case, and (2) more complex coupling terms need to be evaluated. We address the first point by precomputing the values of the linearization point at the quadrature points and by computing the gradients on the fly. In order to reduce the costs of the coupling, we use similar strategies as discussed in Section 4.2. In particular, we exploit the facts (1) that a tensor of the form $\mathbf{n}_a \otimes \mathbf{n}_b$ applied to a vector $\mathbf{v}$ can be efficiently replaced by a dot product and a scalar scaling, $(\mathbf{n}_a \otimes \mathbf{n}_b)\mathbf{v} = \mathbf{n}_a(\mathbf{n}_b \cdot \mathbf{v})$, without the need to actually set up the tensor and (2) that in (11) and (12) no square root has to be computed, since it cancels out later on during subsequent multiplications.

Fig. 21 and Table 8 present timings, data volume, and work compared to the scalar-mobility case for the evaluation of the residual and of the Jacobian. One can observe a clear slowdown of up to 29 and 38%, respectively. This is caused by an increase of arithmetic operations by 7–127 and 35–244 FLOP per quadrature point for different numbers of components in the two cases, showing more pronouncedly the quadratic complexity. According to the roofline performance model of Fig. 7, high FLOP/s can be maintained with increasing number of components. In the case of the Jacobian, the measured read data volume decreases, as the gradients are computed on the fly. In total, the read and write data volumes are similar to the ones in Table 1, indicating that the compute-intensive operations are performed on cached data.

Table 5 gives statistics of a complete 51-particle simulation, showing an increase in the simulation time by 19% with an increase in the number of time steps and a decrease in the number of linear iterations. We note that the matrix-free and the Jacobian-free evaluation as well as the cut-off approach are applicable here without the need for any modifications.

7.2. Extension 2: advection terms

The system (1) is capable to capture the evolution of the microstructure during sintering, however, completely omits shrinkage mechanisms. In order to resolve this issue, Wang [22] added advection terms to system (1):

$$\frac{\partial c}{\partial t}(\mathbf{x},t) = \nabla \cdot \left[M \nabla \frac{\delta F}{\delta c} \right] - \nabla \cdot (c\mathbf{v}), \tag{13a}$$

$$\frac{\partial \eta_i}{\partial t}(\mathbf{x},t) = -L \frac{\delta F}{\delta \eta_i} - \nabla \cdot \left(\eta_i \mathbf{v}_i\right). \tag{13b}$$

Here, $\mathbf{v} = \sum_i \mathbf{v}_i$ and the advection velocity within individual particles $\mathbf{v}_i$ consists of translational and rotational components:

$$\mathbf{v}_i(\mathbf{x}) = \begin{cases} \mathbf{v}_{t\,i}(\mathbf{x}) + \mathbf{v}_{r\,i}(\mathbf{x}), & \text{if inside (e.g., } \eta_i \geq 0.1) \\ 0, & \text{otherwise} \end{cases}$$

$$\mathbf{v}_{t\,i}(\mathbf{x}) = \frac{m_t}{V_i}\mathbf{F}_i,$$

$$\mathbf{v}_{r\,i}(\mathbf{x}) = \frac{m_r}{V_i}\mathbf{T}_i \times \left(\mathbf{x} - \mathbf{x}_{c\,i}\right),$$

where m_t and m_r are constants characterizing the particle translation and rotation and where

$$V_i = \int_\Omega \eta_i \, \mathrm{d}\Omega$$

is the particle volume. Vector $\mathbf{x}_{c\,i}$ denotes the mass center of the ith particle. The velocity components $\mathbf{v}_{t\,i}$ and $\mathbf{v}_{r\,i}$ are proportional to the force and torque [22], which are given by

$$\mathbf{F}_i = \int_\Omega \mathrm{d}\mathbf{F}_i, \tag{14a}$$

$$\mathbf{T}_i = \int_\Omega \left(\mathbf{x} - \mathbf{x}_{c\,i}\right) \times \mathrm{d}\mathbf{F}_i. \tag{14b}$$

The effective local force density $\mathrm{d}\mathbf{F}_i$ is the key component of the entire shrinkage mechanism and is related to the annihilation of the over-saturated vacancies at the grain boundaries:

$$\mathrm{d}\mathbf{F}_i = k \sum_{i \neq j} \left(c - c_0\right) \langle \eta_i \eta_j \rangle \left[\nabla \eta_i - \nabla \eta_j\right] \mathrm{d}\Omega.$$

The quantities V_i, $\mathbf{F}_i$, $\mathbf{T}_i$, $\mathbf{x}_{c\,i}$ can be considered as additional properties of the 0D representation, as discussed in Section 4.4. These quantities have to be gathered for each cell (batch) during the cell loop when multiple particles are assigned to the same order parameter.

The definition of advective velocities as proposed in [22] is far from optimal and introduces several drawbacks: the lack of rigorous physically based foundation, complicated calibration of the advection model parameters or severe size effects [70] if only the original Wang's forces are employed. In fact, the discussion of the rigid body motions in the phase-field sintering models [15,25,70] is a big topic by itself. Keeping the above-mentioned limitations in mind, we still implemented this particular extension due to its wide use and the fact that it can be applied for other non-local advection mechanisms. For instance, the staggered coupling of the phase-field and DEM approaches [71] also requires the reduced 0D modeling for its implementation and, thus, could benefit from the techniques described in the present publication.

The velocity terms in (13b) result in different Allen–Cahn blocks, potentially requiring different (ILU) preconditioners. Our experiments indicate that these terms can be dropped during preconditioning, allowing to work with a single ILU instance for all Allen–Cahn blocks, as proposed in Section 4.3.

Due to the non-local terms (14), the evaluation of the exact Jacobian would be both memory-intensive and computationally demanding. Therefore, we only consider the Jacobian-free implementation. Figs. 7, 21, Tables 5, and 8 describe the properties of the residual evaluation as stand-alone and in the context of the 51-particle sintering benchmark with and without cut-off. The increase in the simulation time can be

mainly contributed to the fact that the grain-tracking algorithm has to be performed at each time step. Also the number of operations during the residual evaluation increases by a fixed value of ∼33 FLOP per quadrature point.

8. Conclusions and outlook

We have presented an efficient, adaptive, implicit finite-element solver for modeling solid-state-sintering processes by means of a well-established phase-field approach that is able to capture diffusive mass-transport, shrinkage, and grain-growth phenomena. Our implementation, which is freely available as `hpsint`, has been verified with reference data from the literature and successfully simulates packings with ten thousands of particles in high-performance-computing environments.

To enable such large-scale simulations, we have performed a holistic optimization of the solver on many levels by an interdisciplinary effort in order to minimize the time to solution. The proposed optimizations include a tailored block preconditioner, a distributed graph-based version of the grain-tracking algorithm, and the usage of fast matrix-free evaluation kernels. For the latter, the presented solver relies on the open-source library `deal.II` [20,21], particularly, on its state-of-the-art matrix-free framework [46,47]. Even though it is most efficient for higher-degree shape functions, the underlying algorithmic choices following the current trends of exascale algorithms ensure a high node-level performance also for the linear shape functions in the present case, with increasing advantage for larger numbers of components. We have extended the matrix-free algorithm to deal with varying number of vector components related to changing number of order parameters and to work only with locally-relevant components related to the local support of the phase field as well as presented low-level strategies to reduce the computational effort at quadrature points.

In addition to these fundamental advances, we have discussed possible optimizations that build upon the current developments and might allow an additional speedup of 2× in the near future. This includes the usage of sparse block vectors [39], interleaving of evaluation and quadrature-point loops [47], and physics-based preconditioning [59].

CRediT authorship contribution statement

Peter Munch: Conceptualization, Investigation, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Vladimir Ivannikov:** Formal analysis, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Christian Cyron:** Conceptualization, Funding acquisition, Supervision, Writing – review & editing. **Martin Kronbichler:** Investigation, Methodology, Software, Supervision, Writing – review & editing.

Declaration of competing interest

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

Data availability

The raw data required to reproduce these findings and the processed data required to reproduce these findings are available to download from https://github.com/hpsint/hpsint-data.

Acknowledgments

The authors acknowledge collaboration with Thomas Ebel, Daniel Paukner, Magdalena Schreter-Fleischhacker, and Regine Willumeit-Römer as well as the `deal.II` community. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (LRZ, www.lrz.de) through project id pn36li. This work was also funded by Helmholtz-Zentrum Hereon through I^2B MgSinter project. We would like to credit Hui-Chia Yu, Robert Termuhlen, Xanthippi Chatzistavrou and Jason D. Nicholas for providing us the input data of their simulations.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.commatsci.2023.112589.

References

[1] Huilong Zhu, Robert S. Averback, Molecular dynamics simulations of densification processes in nanocrystalline materials, Mater. Sci. Eng. A 204 (1) (1995) 96–100, Proceedings of the Symposium on Engineering of Nanostructured Materials.
[2] Lifeng Ding, Ruslan L. Davidchack, Jingzhe Pan, A molecular dynamics study of sintering between nanoparticles, Comput. Mater. Sci. 45 (2) (2009) 247–256.
[3] Ken-ichiro Mori, Finite element simulation of powder forming and sintering, Comput. Methods Appl. Mech. Engrg. 195 (48) (2006) 6737–6749.
[4] Vladimir Ivannikov, Fritz Thomsen, Thomas Ebel, Regine Willumeit-Römer, Coupling the discrete element method and solid state diffusion equations for modeling of metallic powders sintering, Comput. Part. Mech. 10 (2) (2023) 185–207.
[5] Christopher Gloeckle, Thomas Konkol, Olaf Jacobs, Wolfgang Limberg, Thomas Ebel, Ulrich A. Handge, Processing of highly filled polymer-metal feedstocks for fused filament fabrication and the production of metallic implants, Materials 13 (19) (2020).
[6] Eshwara Phani Shubhakar Nidadavolu, Diana Krüger, Berit Zeller-Plumhoff, Domonkos Tolnai, Björn Wiese, Frank Feyerabend, Thomas Ebel, Regine Willumeit-Römer, Pore characterization of pm mg–0.6ca alloy and its degradation behavior under physiological conditions, J. Magnes. Alloys 9 (2) (2021) 686–703.
[7] Robert Termuhlen, Xanthippi Chatzistavrou, Jason D. Nicholas, Hui-Chia Yu, Three-dimensional phase field sintering simulations accounting for the rigid-body motion of individual grains, Comput. Mater. Sci. 186 (2021) 109963.
[8] Henrik Hierl, Johannes Hötzer, Marco Seiz, Andreas Reiter, Britta Nestler, Extreme scale phase-field simulation of sintering processes, in: 2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA), IEEE, 2019, pp. 25–32.
[9] Johannes Hötzer, Andreas Reiter, Henrik Hierl, Philipp Steinmetz, Michael Selzer, Britta Nestler, The parallel multi-physics phase-field framework Pace3D, J. Comput. Sci. 26 (2018) 1–12.
[10] Eisuke Miyoshi, Tomohiro Takaki, Munekazu Ohno, Yasushi Shibuta, Shinji Sakane, Takashi Shimokawabe, Takayuki Aoki, Ultra-large-scale phase-field simulation study of ideal grain growth, NPJ Comput. Mater. 3 (1) (2017) 25.
[11] Derek Gaston, Chris Newman, Glen Hansen, Damien Lebrun-Grandié, Moose: A parallel computational framework for coupled systems of nonlinear equations, Nucl. Eng. Des. 239 (10) (2009) 1768–1778.
[12] Ian Greenquist, Michael R. Tonks, Larry K. Aagesen, Yongfeng Zhang, Development of a microstructural grand potential-based sintering model, Comput. Mater. Sci. 172 (2020) 109288.
[13] Supriyo Ghosh, Christopher K. Newman, Marianne M. Francois, Tusas: A fully implicit parallel approach for coupled phase-field equations, J. Comput. Phys. 448 (2022) 110734.
[14] Stephen DeWitt, Shiva Rudraraju, David Montiel, W. Beck Andrews, Katsuyo Thornton, PRISMS-PF: A general framework for phase-field modeling with a matrix-free finite element method, npj Comput. Mater. 6 (1) (2020) 1–12.
[15] Vladimir Ivannikov, Fritz Thomsen, Thomas Ebel, Regine Willumeit-Römer, Capturing shrinkage and neck growth with phase field simulations of the solid state sintering, Modelling Simul. Mater. Sci. Eng. 29 (7) (2021) 075008.
[16] Fritz Thomsen, Götz Hofmann, Thomas Ebel, Regine Willumeit-Römer, An elementary simulation model for neck growth and shrinkage during solid phase sintering, Materialia 3 (2018) 338–346.

[17] Cody J. Permann, Michael R. Tonks, Bradley Fromm, Derek R. Gaston, Order parameter re-mapping algorithm for 3D phase field model of grain growth using FEM, Comput. Mater. Sci. 115 (2016) 18–25.

[18] Phani Motamarri, Sambit Das, Shiva Rudraraju, Krishnendu Ghosh, Denis Davydov, Vikram Gavini, DFT-FE – a massively parallel adaptive finite-element code for large-scale density functional theory calculations, Comput. Phys. Comm. 246 (2020) 106853.

[19] Nikhil Kodali, Gourab Panigrahi, Debashis Panda, Phani Motamarri, Fast hardware-aware matrix-free computations of higher-order finite-element discretized matrix multi-vector products, 2022, arXiv preprint arXiv:2208.07129.

[20] Daniel Arndt, Wolfgang Bangerth, Marco Feder, Marc Fehling, Rene Gassmöller, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Peter Munch, Jean-Paul Pelteret, Simon Sticko, Bruno Turcksin, David Wells, The `deal.II` library, version 9.4, J. Numer. Math. 30 (3) (2022) 231–246.

[21] Daniel Arndt, Wolfgang Bangerth, Denis Davydov, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Jean-Paul Pelteret, Bruno Turcksin, David Wells, The deal.II finite element library: Design, features, and insights, Comput. Math. Appl. 81 (2021) 407–422.

[22] Yu U. Wang, Computer modeling and simulation of solid-state sintering: A phase field approach, Acta Mater. 54 (4) (2006) 953–961.

[23] Jie Deng, A phase field model of sintering with direction-dependent diffusion, Mater. Trans. 53 (2) (2012) 385–389.

[24] Karim Ahmed, Clarissa A. Yablinsky, A. Schulte, Todd Allen, Anter El-Azab, Phase field modeling of the effect of porosity on grain growth kinetics in polycrystalline ceramics, Modelling Simul. Mater. Sci. Eng. 21 (6) (2013) 065005.

[25] Marco Seiz, Effect of rigid body motion in phase-field models of solid-state sintering, Comput. Mater. Sci. 215 (2022) 111756.

[26] Sudipta Biswas, Daniel Schwen, Hao Wang, Maria A. Okuniewski, Vikas Tomar, Phase field modeling of sintering: Role of grain orientation and anisotropic properties, Comput. Mater. Sci. 148 (2018) 307–319.

[27] Carl E. Krill III, Long-Qing Chen, Computer simulation of 3-D grain growth using a phase-field model, Acta Mater. 50 (12) (2002) 3059–3075.

[28] Srikanth Vedantam, Prasad Patnaik, Efficient numerical algorithm for multiphase field simulations, Phys. Rev. E 73 (2006) 016703.

[29] Qingcheng Yang, Yongxin Gao, Arkadz Kirshtein, Qiang Zhen, Chun Liu, A free-energy-based and interfacially consistent phase-field model for solid-state sintering without artificial void generation, Comput. Mater. Sci. 229 (2023) 112387.

[30] Sudipta Biswas, Daniel Schwen, Vikas Tomar, Implementation of a phase field model for simulating evolution of two powder particles representing microstructural changes during sintering, J. Mater. Sci. 53 (8) (2018) 5799–5825.

[31] K. Chockalingam, Varvara. G. Kouznetsova, Olaf van der Sluis, Marc G.D. Geers, 2D phase field modeling of sintering of silver nanoparticles, Comput. Methods Appl. Mech. Engrg. 312 (2016) 492–508.

[32] Michael Pernice, Homer F. Walker, NITSOL: A Newton iterative solver for nonlinear systems, SIAM J. Sci. Comput. 19 (1) (1998) 302–318.

[33] Peter N. Brown, Youcef Saad, Hybrid Krylov methods for nonlinear systems of equations, SIAM J. Sci. Stat. Comput. 11 (3) (1990) 450–481.

[34] Satish Balay, Shrirang Abhyankar, Mark Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William Gropp, et al., PETSc Users Manual, Argonne National Laboratory, 2019.

[35] Jed Brown, Efficient nonlinear solvers for nodal high-order finite elements in 3D, J. Sci. Comput. 45 (1) (2010) 48–63.

[36] Stephen DeWitt, Shiva Rudraraju, David Montiel, W. Beck Andrews, Katsuyo Thornton, PRISMS-PF: A general framework for phase-field modeling with a matrix-free finite element method, npj Comput. Mater. 6 (1) (2020) 29.

[37] Seong Gyoon Kim, Dong Ik Kim, Won Tae Kim, Yong Bum Park, Computer simulations of two-dimensional and three-dimensional ideal grain growth, Phys. Rev. E 74 (6) (2006) 061605.

[38] Denis Davydov, Timo Heister, Martin Kronbichler, Paul Steinmann, Matrix-free locally adaptive finite element solution of density-functional theory with nonorthogonal orbitals and multigrid preconditioning, Phys. Status Solidi (b) 255 (9) (2018) 1800069.

[39] Denis Davydov, Martin Kronbichler, Algorithms and data structures for matrix-free finite element operators with MPI-parallel sparse multi-vectors, ACM Trans. Parallel Comput. (TOPC) 7 (3) (2020) 1–30.

[40] Daniel Arndt, Niklas Fehn, Guido Kanschat, Katharina Kormann, Martin Kronbichler, Peter Munch, Wolfgang A. Wall, Julius Witte, ExaDG: High-order discontinuous Galerkin for the exa-scale, in: Software for Exascale Computing-SPPEXA 2016-2019, Springer, 2020, pp. 189–224.

[41] Michel O. Deville, Paul F. Fischer, Ernest H. Mund, High-Order Methods for Incompressible Fluid Flow, Cambridge University Press, 2002.

[42] Martin Kronbichler, Ababacar Diagne, Hanna Holmgren, A fast massively parallel two-phase flow solver for microfluidic chip simulation, Int. J. High Perform. Comput. Appl. 32 (2) (2018) 266–287.

[43] Denis Davydov, Jean-Paul Pelteret, Daniel Arndt, Martin Kronbichler, Paul Steinmann, A matrix-free approach for finite-strain hyperelastic problems using geometric multigrid, Internat. J. Numer. Methods Engrg. 121 (13) (2020) 2874–2895.

[44] Jed Brown, Valeria Barra, Natalie Beams, Leila Ghaffari, Matthew Knepley, William Moses, Rezgar Shakeri, Karen Stengel, Jeremy L. Thompson, Junchao Zhang, Performance portable solid mechanics via matrix-free p-multigrid, 2022, arXiv preprint arXiv:2204.01722.

[45] Peter Munch, Katharina Kormann, Martin Kronbichler, hyper.deal: An efficient, matrix-free finite-element library for high-dimensional partial differential equations, ACM Trans. Math. Software 47 (4) (2021) 33/1–34.

[46] Martin Kronbichler, Katharina Kormann, A generic interface for parallel cell-based finite element operator application, Comput. & Fluids 63 (2012) 135–147.

[47] Martin Kronbichler, Katharina Kormann, Fast matrix-free evaluation of discontinuous Galerkin finite element operators, ACM Trans. Math. Softw. 45 (3) (2019) 1–40.

[48] Tzanio Kolev, Paul Fischer, Misun Min, Jack Dongarra, Jed Brown, Veselin Dobrev, Tim Warburton, Stanimire Tomov, Mark S. Shephard, Ahmad Abdelfattah, Valeria Barra, Natalie Beams, Jean-Sylvain Camier, Noel Chalmers, Yohann Dudouit, Ali Karakus, Ian Karlin, Stefan Kerkemeier, Yu-Hsiang Lan, David Medina, Elia Merzari, Aleksandr Obabko, Will Pazner, Thilina Rathnayake, Cameron W. Smith, Lukas Spies, Kasia Swirydowicz, Jeremy Thompson, Ananias Tomboulides, Vladimir Tomov, Efficient exascale discretizations: High-order finite element methods, Int. J. High Perform. Comput. Appl. 35 (6) (2021) 527–552.

[49] Peter Munch, Karl Ljungkvist, Martin Kronbichler, Efficient application of hanging-node constraints for matrix-free high-order FEM computations on CPU and GPU, in: International Conference on High Performance Computing, Springer, 2022, pp. 133–152.

[50] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, Stefano Zampini, MFEM: A modular finite element methods library, Comput. Math. Appl. 81 (2021) 42–74.

[51] Martin Kronbichler, Karl Ljungkvist, Multigrid for matrix-free high-order finite element computations on graphics processors, ACM Trans. Parallel Comput. 6 (1) (2019) 2:1–32.

[52] Karl Ljungkvist, Matrix-free finite-element computations on graphics processors with adaptively refined unstructured meshes, in: SpringSim (HPC), 2017.

[53] Steffen Müthing, Marian Piatkowski, Peter Bastian, High-performance implementation of matrix-free high-order discontinuous Galerkin methods, 2017, arXiv preprint arXiv:1711.10885.

[54] Jan Treibig, Georg Hager, Gerhard Wellein, LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments, in: Proceedings of PSTI2010, San Diego CA, 2010, pp. 207–216.

[55] Samuel Williams, Andrew Waterman, David Patterson, Roofline: an insightful visual performance model for multicore architectures, Commun. ACM 52 (4) (2009) 65–76.

[56] Katarzyna Świrydowicz, Noel Chalmers, Ali Karakus, Timothy Warburton, Acceleration of tensor-product operations for high-order finite element methods, Int. J. High Perform. Comput. Appl. 33 (4) (2019) 735–757.

[57] Martin Kronbichler, Dmytro Sashko, Peter Munch, Enhancing data locality of the conjugate gradient method for high-order matrix-free finite-element implementations, Int. J. High Perform. Comput. Appl. (2022) 10943420221107880.

[58] Peter Munch, Martin Kronbichler, Cache-optimized and low-overhead implementations of additive Schwarz methods for high-order FEM multigrid computations, 2023, Manuscript.

[59] Owe Axelsson, Petia Boyanova, Martin Kronbichler, Maya Neytcheva, Xunxun Wu, Numerical and computational efficiency of solvers for two-phase problems, Comput. Math. Appl. 65 (3) (2013) 301–314.

[60] James D. Foley, Andries Van Dam, John F. Hughes, Steven K. Feiner, Computer Graphics: Principles and Practice, second ed., Addison-Wesley, 1990.

[61] Wolfgang Bangerth, Carsten Burstedde, Timo Heister, Martin Kronbichler, Algorithms and data structures for massively parallel generic adaptive finite element codes, ACM Trans. Math. Softw. 38 (2) (2012) 1–28.

[62] Jeremy G. Siek, Lie-Quan Lee, Andrew Lumsdaine, The Boost Graph Library: User Guide and Reference Manual, The, Pearson Education, 2001.

[63] Michael Heroux, Roscoe Bartlett, Vicki Howle Robert Hoekstra, Jonathan Hu, Tamara Kolda, Richard Lehoucq, Kevin Long, Roger Pawlowski, Eric Phipps, Andrew Salinger, et al., An Overview of Trilinos, Citeseer, 2003.

[64] Xiaobing Feng, Steven Wise, Analysis of a Darcy–Cahn–Hilliard diffuse interface model for the Hele–Shaw flow and its fully discrete finite element approximation, SIAM J. Numer. Anal. 50 (3) (2012) 1320–1343.

[65] Carsten Burstedde, Lucas C. Wilcox, Omar Ghattas, p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, SIAM J. Sci. Comput. 33 (3) (2011) 1103–1133.

[66] Sudipta Biswas, Daniel Schwen, Jogender Singh, Vikas Tomar, A study of the evolution of microstructure and consolidation kinetics during sintering using a phase field modeling based approach, Extreme Mech. Lett. 7 (2016) 78–89.

[67] Robert D. Falgout, Ulrike Meier Yang, hypre: A library of high performance preconditioners, in: Computational Science–ICCS 2002: International Conference Amsterdam, The Netherlands, April 21–24, 2002 Proceedings, Part III, Springer, 2002, pp. 632–641.

[68] Benjamin S. Kirk, John W. Peterson, Roy H. Stogner, Graham F. Carey, `libMesh`: A C++ library for parallel adaptive mesh refinement/coarsening simulations, Eng. Comput. 22 (3–4) (2006) 237–254.

[69] Vaclav Smilauer, et al., Yade Documentation, second ed., The Yade Project, 2015, http://yade-dem.org/doc/.

[70] Marco Seiz, Henrik Hierl, Britta Nestler, An improved grand-potential phase-field model of solid-state sintering for many particles, Modelling Simul. Mater. Sci. Eng. 31 (5) (2023) 055006.

[71] Kazunari Shinagawa, Simulation of grain growth and sintering process by combined phase-field/discrete-element method, Acta Mater. 66 (2014) 360–369.