Computer Physics Communications 215 (2017) 173–187

Contents lists available at ScienceDirect

Computer Physics Communications

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

Parallel multiphase field simulations with OpenPhase☆

Marvin Tegeler [a,*], Oleg Shchyglo [a], Reza Darvishi Kamachali [a], Alexander Monas [a], Ingo Steinbach [a], Godehard Sutmann [a,b]

[a] Interdisciplinary Centre for Advanced Materials Simulation, Ruhr-Universität Bochum, Universitätsstr. 150, 44801 Bochum, Germany
[b] Jülich Supercomputing Centre, Forschungszentrum Jülich, Wilhelm-Johnen-Straße, 52425 Jülich, Germany

ARTICLE INFO

Article history:
Received 10 May 2016
Received in revised form 20 January 2017
Accepted 27 January 2017
Available online 7 February 2017

Keywords:
Material science
Phase field
Parallel computing
Load-balancing

ABSTRACT

The open-source software project OpenPhase allows the three-dimensional simulation of microstructural evolution using the multiphase field method. The core modules of OpenPhase and their implementation as well as their parallelization for a distributed-memory setting are presented. Especially communication and load-balancing strategies are discussed. Synchronization points are avoided by an increased halo-size, i.e. additional layers of ghost cells, which allow multiple stencil operations without data exchange. Load-balancing is considered via graph-partitioning and sub-domain decomposition. Results are presented for performance benchmarks as well as for a variety of applications, e.g. grain growth in polycrystalline materials, including a large number of phase fields as well as Mg–Al alloy solidification.

Program summary
Program Title: OpenPhase
Program Files doi: http://dx.doi.org/10.17632/2mnv2fvkkk.1
Licensing provisions: GPLv3
Programming language: C++
Nature of problem: OpenPhase[1] allows the simulation of microstructure evolution during materials processing using the multiphase field method. In order to allow an arbitrary number of phase fields active parameter tracking is used, which can cause load imbalances in parallel computations.
Solution method: OpenPhase solves the phase field equations using an explicit finite difference scheme. The parallel version of OpenPhase provides load-balancing using over-decomposition of the computational domain and graph-partitioning. Adaptive sub-domain sizes are used to minimize the computational overhead of the over-decomposition, while allowing appropriate load-balance.
Additional comments including Restrictions and Unusual features: The distributed-memory parallelism in OpenPhase uses MPI. Shared-memory parallelism is implemented using OpenMP. The library uses C++11 features and therefore requires GCC version 4.7 or higher.

[1] www.openphase.de

1. Introduction

OpenPhase [1] is an open-source library that implements the multiphase field method. This method is commonly used to simulate evolution of microstructure during materials processing [2–5]. In multiphase field simulations grains of different phases and orientations are represented by auxiliary functions that are called phase fields. These functions form a partition of unity and provide a finite diffuse interface between grains. The first part of this work gives an introduction to the multiphase field method and presents its implementation in OpenPhase. OpenPhase uses sparse storages and active parameter tracking in order to reduce the memory requirements and computation time. This results in an inhomogeneous distribution of the workload, which requires load balancing techniques to avoid unnecessary waiting times in parallel applications.

A distributed-memory parallelization is needed as large scale simulations on large computational grids with thousands of different phase fields exceed the capabilities of single node computer architectures. Multiple attempts have been conducted to parallelize phase field simulations in order to treat large systems [6–8]. However, both [6] and [8] are restricted to a limited number of phase field and do not implement the multiphase field method. As [7] points out the parallel efficiency and the load-balancing

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: marvin.tegeler@rub.de (M. Tegeler).

http://dx.doi.org/10.1016/j.cpc.2017.01.023
0010-4655/

capabilities of the multiphase field software PACE3D [9] is limited by the lack of a three-dimensional domain decomposition. The well-known multiphase field software Micress® [10] does not support distributed-memory parallelization.

In [11] the authors presented a distributed-memory parallelization using a three dimensional domain decomposition with dynamic load-balancing, that has been implemented in OpenPhase to provide an efficient parallel open-source multiphase field library, which can be co-developed further on by the community. We recall and expand the concepts of the hybrid parallelization that combines MPI and OpenMP and show additional benchmarks and scaling tests. The presented parallelization techniques are applied in simulations of microstructure evolution for a large number of grains during normal grain growth and Mg–Al alloy solidification in full three-dimensional resolution.

2. Implementation of the multiphase field model in OpenPhase

In this section, we recall briefly the fundamental formalism for the phase field method, as it is implemented in OpenPhase. In the phase field method diffuse interfaces between grains of different phases and orientations are introduced. These diffuse interfaces are realized with auxiliary functions, i.e. the phase fields. In a domain Ω the functions $\phi_\alpha \in C^2(\Omega)$ take values between 0 and 1 with the condition $\sum_{\alpha=1}^{\infty}\phi_\alpha = 1$, $\phi_\alpha(x) = 1$ indicates the bulk of the phase α and $\phi_\alpha(x) = 0$ the absence of this phase. The evolution of these functions is determined by

$$\dot{\phi}_\alpha = \frac{\partial}{\partial t}\phi_\alpha(x,t) = -\sum_{\beta=1}^{N}\frac{\pi^2 M_{\alpha\beta}}{8\eta N}\left[\frac{\delta F}{\delta\phi_\alpha} - \frac{\delta F}{\delta\phi_\beta}\right], \tag{1}$$

with the interface mobility $M_{\alpha\beta}$ between phases α and β and the free energy

$$F = \int_\Omega \{f^{\mathrm{GB}} + \cdots\} dV, \tag{2}$$

and the number of phase fields that make a contribution

$$N := N(x) := \sum_{\alpha=1}^{\infty}\chi_{T(x)}(\phi_\alpha), \tag{3}$$

with

$$T(x) := \{\phi \in C^2(\Omega) | \phi(x) \neq 0 \vee \nabla\phi(x) \neq 0 \vee \nabla^2\phi(x) \neq 0\}, \tag{4}$$

and the characteristic function

$$\chi_T(\phi) = \begin{cases} 1 & \text{if } \phi \in T, \\ 0 & \text{else.} \end{cases} \tag{5}$$

The interfacial free energy density is given by

$$f^{\mathrm{GB}} = \sum_{\alpha\neq\beta}\frac{4\sigma_{\alpha\beta}}{\eta}\left[-\frac{\eta^2}{\pi^2}\nabla\phi_\alpha\cdot\nabla\phi_\beta + \phi_\alpha\phi_\beta\right] \tag{6}$$

with the width η and the energy $\sigma_{\alpha\beta}$ of the interface.

Following [5] and [3] by inserting Eq. (6) and Eq. (2) in Eq. (1) and calculating the variational derivative we get

$$\dot{\phi}_\alpha^{\mathrm{GB}} = \sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N}\frac{M_{\alpha\beta}}{N}\left[\sum_{\substack{\gamma=1\\\gamma\neq\beta}}^{N}\left(\sigma_{\beta\gamma} - \sigma_{\alpha\gamma}\right) I_\gamma\right] \tag{7}$$

$$= \frac{1}{N}\sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N} M_{\alpha\beta}\left[\sigma_{\alpha\beta}\left[I_\alpha - I_\beta\right] + \sum_{\substack{\gamma=1\\\alpha\neq\gamma\neq\beta}}^{N}\left(\sigma_{\beta\gamma} - \sigma_{\alpha\gamma}\right) I_\gamma\right], \tag{8}$$

$$= \frac{1}{N}\sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N}\dot{\psi}_{\alpha\beta}^{\mathrm{GB}}, \tag{9}$$

with $I_\alpha := \nabla^2\phi_\alpha + \frac{\pi^2}{\eta^2}\phi_\alpha$ and the grain boundary interface field

$$\dot{\psi}_{\alpha\beta}^{\mathrm{GB}} := M_{\alpha\beta}\left[\sigma_{\alpha\beta}\left[I_\alpha - I_\beta\right] + \sum_{\substack{\gamma=1\\\alpha\neq\gamma\neq\beta}}^{N}\left(\sigma_{\beta\gamma} - \sigma_{\alpha\gamma}\right) I_\gamma\right]. \tag{10}$$

2.1. Chemical driving-forces and diffusion

In Eq. (2) additional energy contributions can be considered. As proposed in [3] a chemical energy density can be considered by introducing

$$f^{\mathrm{CH}} = \sum_{\alpha=1}^{N}\phi_\alpha f_\alpha(c_\alpha) + \lambda\left[c - \sum_{\alpha=1}^{N}(\phi_\alpha c_\alpha)\right], \tag{11}$$

with bulk free energies $f_\alpha(c_\alpha)$ of the individual phases.

The vector c is the vector of the overall concentrations in a mixture and c_α is the vector of its concentrations in the phase α. λ is a generalized chemical potential, introduced as a Lagrange multiplier in order to conserve the mass balance between the phases

$$c = \sum_{\alpha=1}^{N}\phi_\alpha c_\alpha. \tag{12}$$

The evolution equation for the concentration is given by

$$\dot{c} = \nabla\cdot\left\{\sum_\alpha \phi_\alpha\left[D^\alpha\nabla c_\alpha\right] + \sum_{\alpha,\beta} J_{\alpha\beta}\right\} \tag{13}$$

with the diffusion matrices D^α in the individual phases α for the pairwise interaction of chemical components c_i and c_j. The last term on the right-hand side is the anti-trapping current $J_{\alpha\beta}$ which eliminates the numerical solute trapping due to the diffusiveness of the interface [3].

As in Eq. (7) we get

$$\dot{\phi}_\alpha = \sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N}\frac{M_{\alpha\beta}}{N}\left[\sum_{\substack{\gamma=1\\\gamma\neq\beta}}^{N}\left(\sigma_{\beta\gamma} - \sigma_{\alpha\gamma}\right) I_\gamma + \frac{\pi^2}{8\eta}\Delta g_{\alpha\beta}\right], \tag{14}$$

with driving force

$$\Delta g_{\alpha\beta} = -f_\alpha(c_\alpha) + f_\beta(c_\beta) + \lambda\left(c_\alpha - c_\beta\right), \tag{15}$$

and $\lambda = \frac{\partial f_\alpha}{\partial c_\alpha}$ according to [12].

The main advantage of the phase field model used in OpenPhase is the rigorous consideration of the interface properties. One of the key points of the model is the assumption that a steady phase field contour along the interface normal is maintained at all times during the simulation which also ensures a constant width of the interface. Such a steady phase field contour, also called traveling wave solution [3], is important for the correct consideration of the interface properties and the resulting interface kinetics.

In real simulations due to the variation of physical parameters in space and time the driving force $\Delta g_{\alpha\beta}$ also varies across the interface. This can significantly alter the phase field profile across the interface which significantly affects interface kinetics. However, it is important to note, that the correct phase field profile across the interface can only be obtained if the driving force $\Delta g_{\alpha\beta}$ is constant across the interface and varies only along the interface. Therefore we replace $\Delta g_{\alpha\beta}$ in Eq. (14) with $h'\Delta\bar{g}_{\alpha\beta}$, such that

$$\frac{1}{\eta}\int_{-\frac{\eta}{2}}^{\frac{\eta}{2}}\Delta g_{\alpha\beta} = h'\Delta\bar{g}_{\alpha\beta}, \tag{16}$$

with an average driving force $\Delta\bar{g}_{\alpha\beta}$, that is constant along the normal direction of the interface. In order to guarantee that the driving force contribution to the phase field evolution is independent of the chosen space discretization (the same is true for the interface curvature contribution), the prefactor $h' = \frac{8}{\pi}\sqrt{\phi_\alpha\phi_\beta}$ in front of the driving force $\Delta\bar{g}_{\alpha\beta}$ is obtained by integrating the traveling wave profile (for more details see [3]). By combining Eqs. (14), (8) and (9) we finally get

$$\dot{\phi}_\alpha = \sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N} \frac{M_{\alpha\beta}}{N}\left[\sum_{\substack{\gamma=1\\\gamma\neq\beta}}^{N}\left(\sigma_{\beta\gamma}-\sigma_{\alpha\gamma}\right)I_\gamma + \frac{\pi}{\eta}\sqrt{\phi_\alpha\phi_\beta}\,\Delta\bar{g}_{\alpha\beta}\right], \tag{17}$$

$$= \frac{1}{N}\sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N}\dot{\psi}^{\mathrm{GB}}_{\alpha\beta} + \frac{1}{N}\sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N} M_{\alpha\beta}\frac{\pi}{\eta}\sqrt{\phi_\alpha\phi_\beta}\,\Delta\bar{g}_{\alpha\beta}, \tag{18}$$

$$= \underbrace{\frac{1}{N}\sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N}\dot{\psi}^{\mathrm{GB}}_{\alpha\beta}}_{=:\dot{\phi}^{\mathrm{GB}}_\alpha} + \underbrace{\frac{1}{N}\sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N}\dot{\psi}^{\mathrm{CH}}_{\alpha\beta}}_{=:\dot{\phi}^{\mathrm{CH}}_\alpha}, \tag{19}$$

with the interface field for the chemical driving force

$$\dot{\psi}^{\mathrm{CH}}_{\alpha\beta} := M_{\alpha\beta}\frac{\pi}{\eta}\sqrt{\phi_\alpha\phi_\beta}\,\Delta\bar{g}_{\alpha\beta}. \tag{20}$$

Remark 1. We can see that if both $\phi_k(x) = 0$ and $\nabla^2\phi_k(x) = 0$ then phase k does not make a contribution to $\dot{\psi}^{\mathrm{GB}}_{ij}$.

Remark 2. $\dot{\phi}^n_i(x) > 0$ is possible even if both $\phi^n_i(x) = 0$ and $\nabla^2\phi^n_i(x) = 0$. This means a nucleation event is triggered that would create a new phase. By processing these nucleation events separately, we can modify Eq. (17) to

$$\dot{\phi}_\alpha = \chi_\alpha\frac{1}{N}\left[\sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N} M_{\alpha\beta}\left[\sigma_{\alpha\beta}\left[I_\alpha - I_\beta\right] + \sum_{\substack{\gamma=1\\\alpha\neq\gamma\neq\beta}}^{N}\left(\sigma_{\beta\gamma}-\sigma_{\alpha\gamma}\right)I_\gamma\right] + \sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N} M_{\alpha\beta}\frac{\pi}{\eta}\sqrt{\phi_\alpha\phi_\beta}\,\Delta\bar{g}_{\alpha\beta}\right], \tag{21}$$

by using the indicator function

$$\chi_i := \begin{cases} 1 & \text{if } \phi_i \neq 0 \vee \nabla^2\phi_i \neq 0,\\ 0 & \text{else,}\end{cases} \tag{22}$$

which allows us to skip certain computations. In Example 4.3 an explicit nucleation model is utilized.

2.2. Discretization

In OpenPhase the domain $\Omega \subset \mathbb{R}^3$ is discretized by a Cartesian grid – the set $\Omega_h \subset \Omega$ contains the grid points x_h in Ω with a grid spacing h. The phase fields are represented on grid functions ϕ^h_α. The evolution of these grid functions over time is discretized by an explicit first order scheme

$$\phi^h_\alpha(x_h, t_n + \Delta t_n) = \phi^h_\alpha(x_h, t_n) + \Delta t_n\dot{\phi}^{\mathrm{GB},h}_\alpha(x_h, t_n) + \Delta t_n\dot{\phi}^{\mathrm{CH},h}_\alpha(x_h, t_n)\ \forall x_h \in \Omega_h \tag{23}$$

with the separate updates $\dot{\phi}^{\mathrm{GB},h}_\alpha(x_h, t_n)$ and $\dot{\phi}^{\mathrm{CH},h}_\alpha(x_h, t_n)$ in each grid point $x_h \in \Omega_h$ at time t_n given by

$$\dot{\phi}^{\mathrm{GB},h}_\alpha(x_h, t_n) = \chi_\alpha(x_h, t_n)\frac{1}{N}\sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N}\dot{\psi}^{\mathrm{GB}}_{\alpha\beta}(x_h, t_n), \tag{24}$$

and

$$\dot{\phi}^{\mathrm{CH},h}_\alpha(x_h, t_n) = \chi_\alpha(x_h, t_n)\frac{1}{N}\sum_{\substack{\beta=1\\\beta\neq\alpha}}^{N} M_{\alpha\beta}(x_h, t_n) \times \frac{\pi}{\eta}\sqrt{\phi_\alpha(x_h, t_n)\phi_\beta(x_h, t_n)}\,\Delta\bar{g}_{\alpha\beta}(x_h, t_n). \tag{25}$$

The laplacians $\nabla^2\phi_i$ in I_α are approximated by the compact fourth order 27-point finite-difference stencil $\nabla^2_h\phi^h$ to get the updates

$$\dot{\psi}^{\mathrm{GB}}_{\alpha\beta}(x_h, t_n) = M_{\alpha\beta}\left[\sigma_{\alpha\beta}\left[I^h_\alpha - I^h_\beta\right] + \sum_{\substack{\gamma=1\\\alpha\neq\gamma\neq\beta}}^{N}\left(\sigma_{\beta\gamma}-\sigma_{\alpha\gamma}\right)I^h_\gamma\right] \tag{26}$$

with $I^h_\alpha = \nabla^2_h\phi^h_\alpha + \frac{\pi^2}{\eta^2}\phi^h_\alpha$. In order to quickly identify grid points that do not require any calculation, grid points are tagged as *interface* if they contain more than one phase fields. Any non-interface grid points neighboring an interface grid point are tagged as a *near-interface* grid points. In a single time step the interface between phase fields can move only to grid points that are tagged as near-interface and not to untagged grid points. This allows the reduction of unnecessary computations.

2.3. Driving-force

In order to ensure the correct phase field profile across the interface an averaging procedure for the driving force is implemented in the OpenPhase. During this procedure the driving force is averaged twice in each grid point over a set of points

$$S^R_{\alpha\beta}(x_h) = \{y_h \in \Omega_h | (R - \|y_h - x_h\|_2)\phi_\alpha(y_h)\phi_\beta(y_h) > 0\}, \tag{27}$$

that is the intersection of a sphere with radius $R = \frac{2}{3}\eta$ and the interface between phases α and β. In the first step the average

$$\Delta\bar{g}^0_{\alpha\beta}(x_h) = \frac{\sum_{y_h\in S^R_{\alpha\beta}(x_h)}\omega_1(y_h)\Delta g_{\alpha\beta}(y_h)}{\sum_{y_h\in S^R_{\alpha\beta}(x_h)}\omega_1(y_h)} \tag{28}$$

is weighted with $\omega_1(y_h) := \phi_\alpha(y_h)^2\phi_\beta(y_h)^2$, which emphasizes values near the center of the interface. The resulting average, however, is not sufficiently smooth and not constant along the normal of the interface. Therefore a second averaging step

$$\Delta\bar{g}_{\alpha\beta}(x_h) = \frac{\sum_{y_h\in S^R_{\alpha\beta}(x_h)}\omega_2(y_h)\Delta\bar{g}^0_{\alpha\beta}(y_h)}{\sum_{y_h\in S^R_{\alpha\beta}(x_h)}\omega_2(y_h)} \tag{29}$$

is needed that only takes values near the center of the interface by using a weight $\omega_2(y_h) := \Theta(\frac{\phi_\alpha}{\phi_\beta} - 0.5)\Theta(2 - \frac{\phi_\alpha}{\phi_\beta})$, in which $\Theta(.)$ is the Heaviside function. In practice this average is sufficiently smooth and constant along the normal of the interface. Other weights like $\omega^n_1(y_h) := \phi_\alpha(y_h)^2\phi_\beta(y_h)^2(R - \|y_h - x_h\|_2)^n$, which reduces smearing of the driving force along the interface, are possible.

2.4. Sparse storages

Considering Remarks 1 and 2 we can significantly reduce the computational load by skipping calculations that do not make a contribution to the phase field evolution. In a grid point $x_h \in \Omega_h$ with $\phi_\alpha(x_h) = 0$ and $\nabla^2\phi_\alpha(x_h) = 0$ we do not need to consider the contribution of the phase field ϕ_α to any other phase or changes to the phase ϕ_α itself. That means we only need to compute updates for Eq. (23) in grid points $X_h(t_n) \subset \Omega_h$, in which $\phi_\alpha(x) \neq 0$ for at least two phases or $\Delta\phi_\alpha(x) \neq 0$ for at least one phase, because we have

$$\dot{\phi}^{\mathrm{GB},h}_\alpha(x_h, t_n) = \dot{\phi}^{\mathrm{CH},h}_\alpha(x_h, t_n) = 0 \quad \forall x_h \in \Omega_h \setminus X_h(t_n), \tag{30}$$

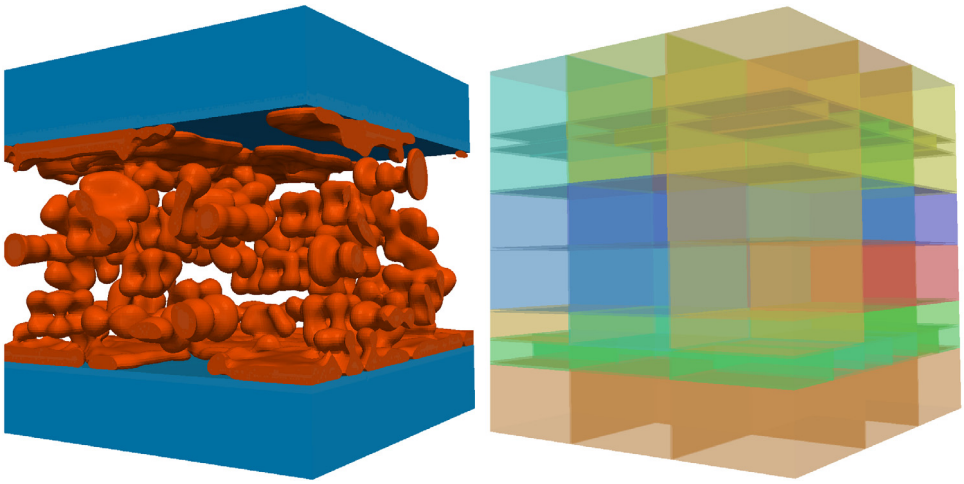


Fig. 1. Simulation of interdendritic solidification in a Mg–Al alloy. The domain decomposition shown on the right-hand side. Different colors indicate different process ranks. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and

$$\phi_\alpha^h(x_h, t_n + \Delta t_n) = \phi_\alpha^h(x_h, t_n) \quad \forall x_h \in \Omega_h \setminus X_h(t_n). \tag{31}$$

Also in this calculation we only need to regard the contribution of phases with $\phi_\alpha(x) \neq 0$ or $\nabla^2\phi_\alpha(x) \neq 0$. Therefore the utilization of sparse storages allows a significantly reduction of the memory requirement, when handling large numbers of phase fields. In order to store the phase fields ϕ_α and the interactions $\psi_{\alpha\beta}$ OpenPhase uses vectors of a container, that stores up to two indices and up to two values. In each grid point this vector only stores entries with non-zero values. Values of zero are implied if no entries for a phase field or interaction exist.

Generally, in the majority of grid points the vector only holds one entry, which means we are in the bulk of a phase. A vector with two entries, which corresponds with the interface of two phases, occurs quite frequently as well. However, grid points with five or more entries are very rare. Due to the small data sets a linear search algorithm is used as it is the most efficient in this case. Using these sparse storages OpenPhase can handle an arbitrarily large number of different phase fields. Depending on the specific data this storage can be interpreted as symmetric or antisymmetric, which further reduces the amount of data by half the number of possible combinations.

3. Parallelization

In the following section a hybrid parallelization of OpenPhase is presented that uses both MPI and OpenMP. As commonly utilized in parallel computing, the computational domain is divided into sub-domains that are distributed among the processors to share the workload. For reasons of simplicity we only use rectangular sub-domains, as these allow for a simple description by their position and size in each direction. Communication of data between these sub-domains is implemented using MPI. Henceforth these sub-domains will be referred to as blocks; the computation on these blocks is further parallelized using the thread-based OpenMP. While for a homogeneous load distribution a simple domain decomposition in which each process works on one equally sized sub-domain remains feasible, inhomogeneous load distributions have to be considered due to the sparse storages (see Section 2.4). This leads load imbalances in a parallel setting, if no redistribution of work among the processors is considered. Since the computational work is mainly concentrated in the interfaces between the phases, the computational load strongly depends on the particular problem and may change during the course of a simulation. The scalability of a program is dominantly limited by its slowest processors, thus a load imbalance can severely restrict the scalability and overall performance of a code as a negative side effect. Therefore an over-decomposition of the domain is utilized as depicted in Fig. 1, i.e. more blocks than processes, which allows to dynamically assign blocks to processes and achieve a sufficient load balance.

Performing a stencil operation invalidates grid points at the boundary, for which the stencil could not access grid points with valid data. The canonical approach would be to update the ghost data after every sweep with a stencil operation. This, however, is problematic as it introduces a synchronization point and for a good runtime behavior, it is necessary to have a balanced load between synchronization points. If a time step is split into different sections of differing load distributions by the synchronization as depicted in the schedule at the top of Fig. 2, no sufficient load-balance can be achieved without changing the domain decomposition for each section, which involves a significant communication overhead. In order to obtain a load balance between all synchronization points multi-constrained load-balancing can be employed, but depending on the problem and the number of synchronization points it can result in rather poor load distributions. However, using a wide halo pattern as presented in [13] allows us to remove unnecessary synchronization points as seen in Fig. 2. In Fig. 3 the concept of the wide halo is depicted. The size of the halo is increased such that information is still valid when all modules hit a common synchronization point; as such the size of the halo depends on the particular problem.

The load balancing strategies, that were discussed in [11], are used to assign the blocks to processes in a way such that the maximum load among processes is minimized. For completion we repeat the key points as depicted in Fig. 5 of the load balancing, which are a combination of the greedy graph-partitioner in Section 3.1 and the bisection of blocks in Section 3.2, which improves the expected quality of the graph-partitioning results. A dynamic scheduling in the OpenMP parallelization on each block easily allows for load-balancing among the threads for each MPI-process.

3.1. Graph-partitioning

The blocks are distributed among the processors by an approximate graph-partitioning method in order to achieve a balanced distribution of work while also minimizing the amount of communicated data. If two adjacent blocks are handled by the same process, obviously no communication is needed.

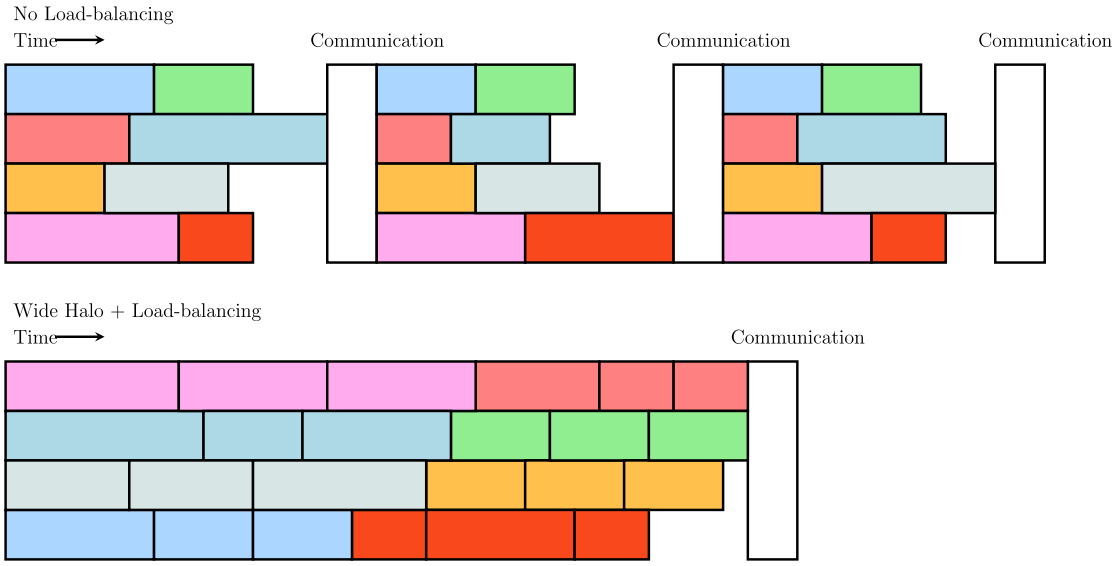


Fig. 2. Example for a scheduling with four processes and two sub-domains on each process. Each work segment on the same sub-domain as indicated by the same color has to be assigned to the same process. The schedule at the top shows a parallelization with multiple synchronization points per time step, while the bottom shows a parallelization with load balancing and removed synchronization points. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In the following each block B_i is represented by a vertex $v_i \in V$ in the graph $G = (V, E)$. The edges $e_{ij} = v_i v_j \in E$ in the graph G indicate whether two blocks B_i and B_j need to communicate data with each other, i.e. ghost cells of one block are interior cells of the other block. The vertex weight $\omega_i = \omega(v_i)$ of a vertex v_i is the computational load of its corresponding block B_i, it is determined by timing the computation on a blocks using ***omp_get_wtime***().

The edge weights $c_{ij} = c(e_{ij})$ are chosen according to the number of common grid points between blocks B_i and B_j. The computational load on process k is denoted by $W_k = \sum_{i \in I_k} \omega_i$, with the index set I_k indicating the blocks on process k. The average load per process for p process is defined as

$$\tilde{W} = \frac{W_{Total}}{p}, \tag{32}$$

with $W_{Total} = \sum \omega_i$ being the total workload.

The goal is to divide the graph into p partitions $P_i \subset V$, such that the edge-cut is minimized and the maximum load

$$W_{max} := \max_{1 \le i \le p} \sum_{v \in P_i} \omega(v) < (1 + \gamma)\, \tilde{W} \tag{33}$$

is bounded. The graph-partitioning problem is NP-complete (see [14]). In order to work independently from graph-partitioning tools such as Zoltan [15] or METIS/ParMETIS [16] OpenPhase provides a simple graph-partitioner, that computes an approximate solution using the greedy algorithm shown in Algorithm 1. Here $N(v)$ denotes the set of vertices adjacent to the vertex v. As computing the partitions and exchanging the blocks can be quite costly and the load should only change slowly over time, it is neither necessary nor beneficial to compute new block distributions too frequently. It is more efficient to measure the load and its imbalance in every time step and to compute a new block distribution if

$$W_k > (1 + \gamma)\, \tilde{W} \tag{34}$$

for any block k and a threshold parameter $\gamma > 0$.

Proposition 1. *Let W_{opt} be the maximum load among partitions achieved by an optimal graph partitioning.*

Algorithm 1 Graph-based load balancing algorithm.

Let $W_{Total} = \sum \omega_i$ be the total load and $\tilde{W} = W_{Total}/p$ the average load on the partitions.
Set cost $c_i = c(v_i) = \sum_{v_j \in N(v_j)} c_{ij}$ of each vertex $v \in V$ and set $Q = V$
Set empty sets $M_i = \varnothing\ \forall\ 0 \le i \le p - 1$ and start with $k = 0$.
while $Q \neq \varnothing$ **do**
  Set $D = \{v \in Q \mid c(v) \le c(w) \forall w \in Q\}$.
  Take $v_* \in D$ with $\omega(v_*)$ maximal, if $\omega(v_*) + W_k \ge \tilde{W}$, take $v_* \in D$ so that $\omega(v_*) + W_k \ge \tilde{W}$ and $\omega(v_*)$ is minimal with that property.
  Add $v*$ to M_k and set $Q = Q \setminus \{v_*\}$.
  Compute the total weight of this set $W_k = \sum_{v_i \in M_k} \omega_i$.
  if $W_k \ge \tilde{W}$ **then**
    Increase k by one and reset all costs $c_i = c(v_i) = \sum_{v_j \in N(v_i)} c_{ij}$ of each vertex $v \in Q$.
  else
    Set the cost $c_j = c_j - 2c_e$ if $e = v_* v_j \in E$.
  end if
end while

Algorithm 1 *fulfills*

$$W_i \le 2W_{opt}\ \forall\ i\ \in I. \tag{35}$$

Proof. See [11].

Remark 3. In fact, we can give an upper bound of the maximum load as

$$W_{max} < \tilde{W} + \omega_{max}. \tag{36}$$

Thus in the worst case the relative overhead due to load imbalances is bounded by $\omega_{max}/\tilde{W}$ as

$$\frac{W_{max}}{\tilde{W}} < 1 + \frac{\omega_{max}}{\tilde{W}}. \tag{37}$$

In addition to graph-partitioner presented in [11] two different post-optimization steps are used to improve the results of the

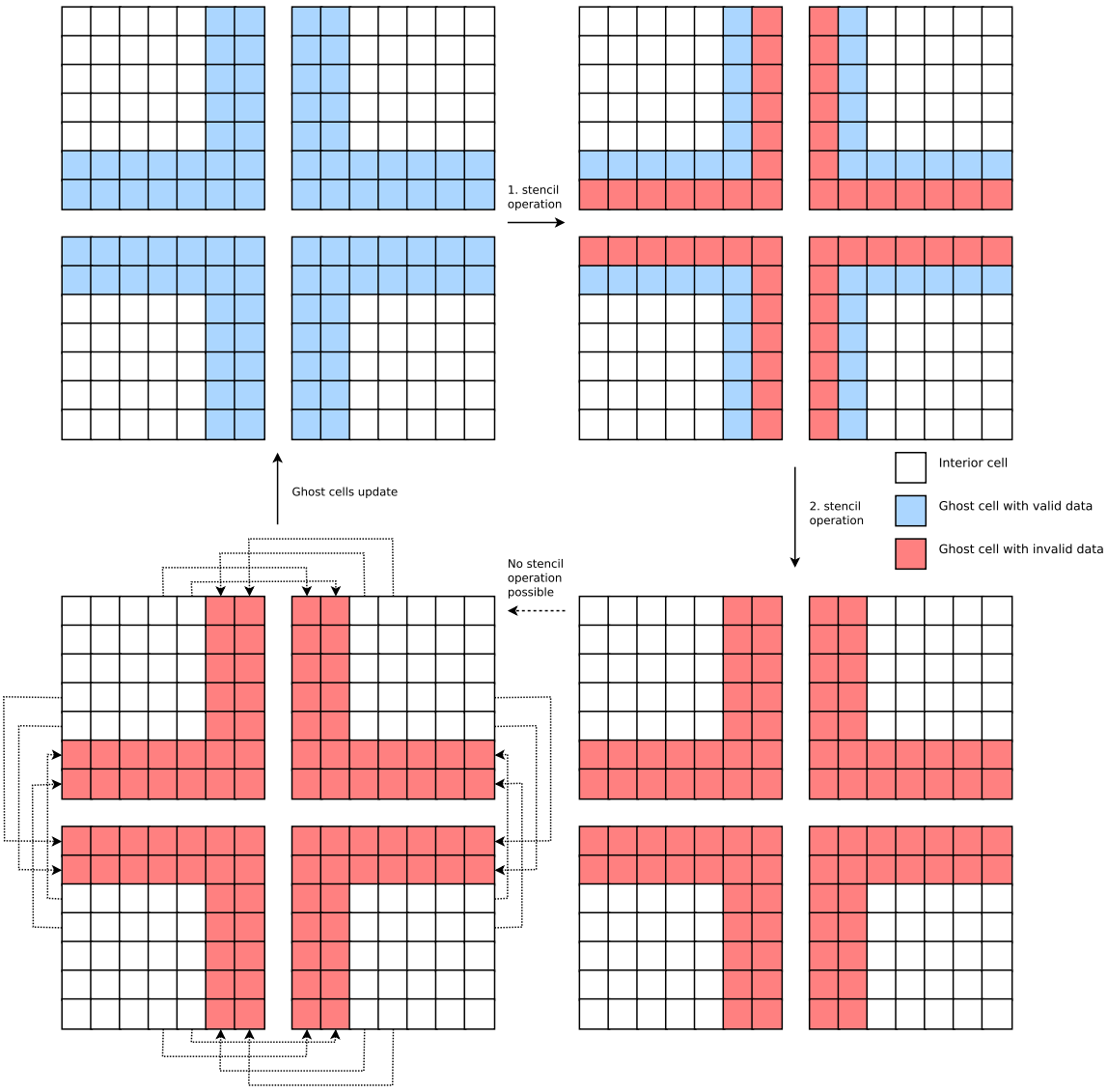


Fig. 3. With two layers of ghost cells two stencil operations are possible before the ghost data needs to be synchronized. More layers of ghost cells, allow more stencil operation without interruption by communication.

graph-partitioner. In the first one a random vertex $v_1 \in P_1$ and a random partition P_2 are chosen. The vertex is moved to this partition either if it improves the load balance between partitions P_1 and P_2, i.e.

$$W_{P_2} + \omega(v_1) < W_{P_1}, \tag{38}$$

and the edge cut is not increase, i.e.

$$\sum_{v_j \in N(v_1) \setminus P_2} c_{1j} - 2 \sum_{v_j \in N(v_1) \cap P_2} c_{1j} \leq \sum_{v_j \in N(v_1) \setminus P_1} c_{1j} - 2 \sum_{v_j \in N(v_1) \cap P_1} c_{1j}, \tag{39}$$

or if the edge cut is decreased and maximum load W_{max} is not surpassed, i.e.

$$W_{P_2} + \omega(v_1) \leq W_{max}. \tag{40}$$

The second post-optimization option is switching the partitions of two random vertices $v_1 \in P_1$ and $v_2 \in P_2$ with $P_1 \neq P_2$ either if the switch improves the balance between these two partitions

$$\max(W_{P_1} - \omega(v_1) + \omega(v_2), W_{P_2} - \omega(v_2) + \omega(v_1)) < \max(W_{P_1}, W_{P_2}), \tag{41}$$

without increasing the edge-cut

$$\sum_{v_j \in N(v_1) \setminus P_2} c_{1j} - 2 \sum_{v_j \in N(v_1) \cap P_2} c_{1j} + 2\chi_{N(v_1)}(v_2) c_{12} \tag{42}$$

$$+ \sum_{v_j \in N(v_2) \setminus P_1} c_{2j} - 2 \sum_{v_j \in N(v_2) \cap P_1} c_{2j} + 2\chi_{N(v_2)}(v_1) c_{21} \tag{43}$$

$$\leq \sum_{v_j \in N(v_1) \setminus P_1} c_{1j} + \sum_{v_j \in N(v_2) \setminus P_2} c_{2j}, \tag{44}$$

or if the switching decreases the edge-cut and the new maximum load does not exceed the old maximum load W_{max}

$$\max(W_{P_1} - \omega(v_1) + \omega(v_2), W_{P_2} - \omega(v_2) + \omega(v_1)) \leq W_{max}. \tag{45}$$

These two post-optimization steps are alternated in a loop that terminates after the time the greedy graph-partitioner used to create the initial partition.

Remark 4. For a small number of vertices, i.e. less than 50 000, the performance of the greedy graph-partitioner is sufficient. For larger graphs multilevel graph-partitioners like METIS are much more efficient.

3.2. Bisection

Remark 3 implies that if we reduce the cost ω_{max} of the most costly block, we can reduce the upper bound of the load imbalance. A reduction of the maximum load ω_{max} can be achieved by a bisection of this blocks into two smaller blocks. This, however, will introduce additional communications and duplicated computations of the overlap between these two blocks. This makes the bisection of the block with the maximum load ω_{max} not feasible if ω_{max} is already comparatively small, so that the expected performance benefit of a better load balance is nullified by the additional costs.

We calculate an optimal maximum load ω_{opt}, such that the blocks with a load higher than ω_{opt} are split into two blocks. As described in [11], we have to compute a minimum of $g(\omega)$, defined by

$$g(\omega) = \sum_{i \in I} \frac{\omega_{cut}(B_i)}{p} \max\left[0, \left\lceil \log_2\left(\frac{\omega(B_i)}{\omega}\right)\right\rceil\right] + \omega, \tag{46}$$

in order to find the optimal maximum load ω_{opt} of blocks.

The function g is obtained in the following way: Let B be a block with computational load $\omega(B)$. If block B is split in two blocks B_1 and B_2, these will have loads $\omega(B_1)$ and $\omega(B_2)$. Generally the amount of work will increase as

$$\omega_{cut}(B) = \omega(B_1) + \omega(B_2) - \omega(B), \tag{47}$$

which is the overhead due to duplicated computations in the wide halo of the blocks. If we cut the block B_{max} with the highest load, we get

$$W^{(1)}_{max} < \tilde{W}^{(1)} + \omega^{(1)}_{max}, \tag{48}$$

with the new highest load $\omega^{(1)}_{max}$ among the blocks after the cut and the new average load

$$\tilde{W}^{(1)} = \tilde{W} + \frac{\omega_{cut}(B_{max})}{p}, \tag{49}$$

which includes the old average $\tilde{W}$ and newly introduced work $\omega_{cut}(B_{max})$. If we repeatedly cut different blocks B_i until $\omega(B_i) < \omega^{(n)}_{max}$ for all blocks B_i, we get

$$W^{(n)}_{max} < \tilde{W} + \sum_{i \in I^{(n)}_{cut}} \frac{\omega_{cut}(B_i)}{p} + \omega^{(n)}_{max}. \tag{50}$$

We have to consider blocks, that are split in multiple stages. If a block B is split in blocks B_1 and B_2, we make the assumption $\omega_{cut}(B) \approx \omega_{cut}(B_1) + \omega_{cut}(B_2)$. The number of times a block is recursively cut is given by $\lceil \log_2\left(\frac{\omega(B_i)}{\omega^{(n)}_{max}}\right)\rceil$. In order to find the optimal maximum size ω_{opt} of blocks we have to find a minimum of $g(\omega)$, defined by

$$g(\omega) = \sum_{i \in I} \frac{\omega_{cut}(B_i)}{p} \max\left[0, \left\lceil \log_2\left(\frac{\omega(B_i)}{\omega}\right)\right\rceil\right] + \omega. \tag{51}$$

This function is monotonically increasing in each interval divided by

$$\frac{\omega(B_1)}{1}, \frac{\omega(B_1)}{2}, \frac{\omega(B_1)}{4}, \ldots, \frac{\omega(B_1)}{2^n}, \frac{\omega(B_2)}{1}, \frac{\omega(B_2)}{2}, \frac{\omega(B_2)}{4}, \ldots, \tag{52}$$

and for practical purposes it is usually enough to subdivide blocks up to three or four times. If blocks are divided further they will get too small compared to the size of the wide halo.

We find the optimal maximum size ω_{opt} by testing at these points for the reasonable choice of $n = 4$. This only sets a limit on how many times a block is split in one load balancing step. If it is possible and necessary to split a block further, splitting will occur in additional load balancing steps.

Conversely, in order to save redundant computations on the halo as well as communications between blocks, two blocks B_i and B_j that are located on the same process and are able to form a cuboid are combined if

$$\omega(B_i) + \omega(B_j) < \omega_{opt}. \tag{53}$$

3.3. Estimation of computational load

In Section 3.2 and especially in Eq. (47) we make use of the computational loads on blocks, while $\omega(B)$ for an existing blocks B is known, $\omega_{cut}(B)$, $\omega(B_1)$ and $\omega(B_2)$ are not. The load on a blocks B can be assumed to be sum of the loads $\omega(x_h)$ of individual grid points $x_h \in B$

$$\omega(B) = \sum_{x_h \in B} \omega(x_h). \tag{54}$$

While the determination of the load $\omega(x_h)$ directly is possible it is not feasible as timing each individual grid point would increase the computational load severely. Considering the implementation details given in Section 2 and especially 2.4 we notice that the amount of computational work in a grid point x_h depends on the number $n(x_h)$ of locally active phase fields. On the one hand we have operations such as the settings of tags as described in Section 2.2, that are independent of the number $n(x_h)$ of active phase fields and therefore scale with $n(x_h)^0$. On the other hand we have operations that are heavily dependent on the number $n(x_h)$ of locally active phase fields and scale with $n(x_h)^3$, such as the computation of the interface energy $\dot{\psi}^{GB}_{\alpha\beta}$ in Eq. (19). Therefore we use the approximation

$$\omega(x_h, \kappa) \approx \tilde{\omega}(x_h) := \sum_{i=0}^{m} \kappa_i n(x_h)^i, \tag{55}$$

with the number of non-zero phase fields $n(x_h)$ at grid point (x_h) to determine the load of each grid point indirectly at run time. With the constrained Broyden–Fletcher–Goldfarb–Shanno algorithm as implemented in dlib [17] we find the minimum of

$$g(\kappa) := \sum_{B \in S} \left[\omega(B) - \sum_{x_h \in B} \tilde{\omega}(x_h, \kappa)\right]^2 \tag{56}$$

with $\kappa_i \geq 0$ for $0 \leq i \leq m$ and a set S of finished computations on blocks. Fig. 4 shows the correlation of the estimate

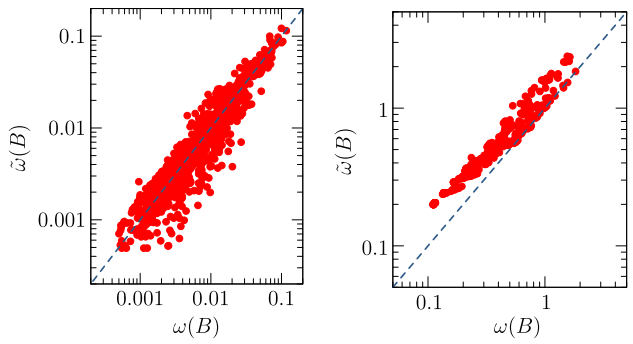
$$\tilde{\omega}(B) = \sum_{x_h \in B} \tilde{\omega}(x_h) \tag{57}$$

to the load $\omega(B)$ on block B for the grain growth scaling test with a gaussian particle distribution in Section 4.2.2 as well as the Mg–Al alloy solidification scaling test with a gaussian particle distribution in Section 4.4.4.

Remark 5. The computational load of stencil operations can also depend on the number of locally active phase fields within the stencil region, therefore the spacial arrangement of phase fields can influence the computational load as well. For simplicity this is not considered in estimation (55).

3.4. Determination of the cutting plane

Using estimate (55) cutting planes for the bisection are selected that are aligned to the grid. For each of the three normal directions



(a) Shows the correlation between $\omega(B)$ and the estimate $\tilde{\omega}(B)$ for the grain growth scaling test in Section 4.2.2 using $m = 3$. The correlation coefficient is $r = 0.95888$.

(b) Shows the correlation between $\omega(B)$ and the estimate $\tilde{\omega}(B)$ for the Mg–Al alloy solidification scaling test in Section 4.4.4 using $m = 3$. The correlation coefficient is $r = 0.95156$.

Fig. 4. Correlation of the estimate $\tilde{\omega}(B)$ obtained by Eq. (55) to the load $\omega(B)$ on block B per time step using a gaussian particle distribution in each test.

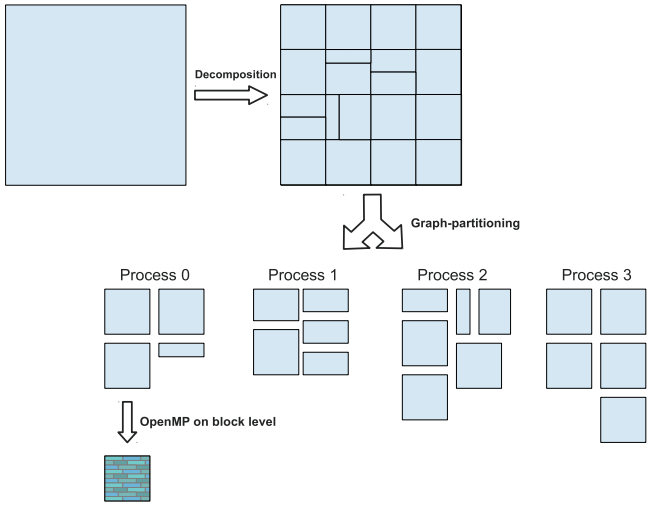


Fig. 5. The computational domain is sub-divided into smaller blocks, that are assigned to MPI-processes by a graph-partitioner. OpenMP provides further parallelism on each block.

$d = x, y$ or z a cutting plane is determined for which the difference $C_d := \|\tilde{\omega}(B_1)_d - \tilde{\omega}(B_2)_d\|$ of the loads of the two resulting blocks minimal is minimal. The cutting plane with normal d that has the minimal difference C_d is chosen and the block is split along this plane.

Remark 6. The number of ghost cell layers poses a constraint on the minimum size of the blocks as we do not allow communication with blocks that are not direct neighbors.

Remark 7. Using the above estimation the overhead by bisection only considers additional computations, but not an added cost for communication. Therefore it makes sense to modify Eq. (51) to

$$g(\omega) = \theta \sum_{i \in I} \frac{\omega_{cut}(B_i)}{p} \max\left[0, \left\lceil \log_2\left(\frac{\omega(B_i)}{\omega}\right)\right\rceil\right] + \omega \tag{58}$$

with a parameter $\theta \geq 1$.

Remark 8. If we set $(1+\gamma)\tilde{W} = \tilde{W} + \omega_{opt}$ the threshold parameter γ in inequality (34) becomes

$$\gamma := \frac{\omega_{opt}}{\tilde{W}}, \tag{59}$$

which means that we compute a new block distribution if the load W_k on a process k is larger than the previous W_{max} due to

$$W_k \geq (1+\gamma)\tilde{W} = \tilde{W} + \omega_{opt} > W_{max}. \tag{60}$$

3.5. The full balancing algorithm

In the beginning of the simulation the user has control over the initial domain decomposition. He can set the size of blocks and the number of blocks in each direction. At the start all blocks have the same size and are automatically assigned to processes so that the number of blocks per process differs at most by one, computational load is not considered at this stage. By calling the balancing method the following steps are executed as also depicted in Fig. 5:

1. Determination of the parameters κ_i as defined in Section 3.3.
2. Determination of the cutting planes for all blocks B as described in Section 3.4 and computing the cost of cutting $\tilde{\omega}_{cut}(B)$.
3. Minimization of function (51) and obtaining the optimal maximum size ω_{opt}.
4. If $W_k > \tilde{W} + \omega_{opt}$, splitting of all blocks B with load $\omega(B) > \omega_{opt}$ along the precomputed cutting plane.
5. Merging of blocks B_i and B_j if they are located on the same process, forms a cuboid and fulfills $\omega(B_i) + \omega(B_j) < \omega_{opt}$.
6. If $W_k > \tilde{W} + \omega_{opt}$, graph-partitioning in order to reassign blocks to processes.

Remark 9. In order to reduce the overhead created by the wide halo it is preferred to design blocks as large as possible. Therefore it can be beneficial to use hybrid-parallelism as depicted in Fig. 5, combining MPI and OpenMP. This way the number of processes p in Eq. (58) gets smaller and blocks remain larger, while still maintaining balance between the processes. Using dynamic scheduling within OpenMP allows easy load-balancing between threads associated to one MPI-process.

3.6. Dynamic load-balancing

During the course of a simulation the distribution of the computational load can change significantly. In order to keep the computational load on the processes balanced it is required to dynamically compute new domain decompositions during the simulation in order to account for the changing load conditions. As computing a new domain-decomposition can be quite costly itself as it contains the bisectioning and merging of blocks as well as moving blocks between processes, we do not want to compute new domain-decompositions unless we assume it is beneficial.

If we have computed a new domain decomposition in time step k it makes sense to compute the next domain decomposition in step $k + n$ for which the average wall time per time step

$$\tilde{T}_k(n) := \frac{1}{n}\left(T^{LB}_{k+n} + \sum_{i=k}^{k+n} T^k_i\right), \tag{61}$$

between load-balancing steps is minimal; T^k_i denotes the wall time for time step i after load balancing in time step k and T^{LB}_{k+n} denotes the wall time for load balancing in time step $k + n$. Thus we have to find $n \in \mathbb{N}$ that minimizes the function $\tilde{T}_k(n)$. As the number of block bisections and merging steps as well as the amount of communication can change, the wall times of the balancing steps

are generally not constant and we do not know the wall time T^{LB}_{k+n} for the balancing algorithm in time step $k+n$ a priori. Therefore we use the exponential moving average $\tilde{T}^{LB}_{k+n}$

$$\tilde{T}^{LB}_{k+n} := \lambda T^{LB}_{k} + (1-\lambda)\,\tilde{T}^{LB}_{k}, \tag{62}$$

with the last wall time for load balancing T^{LB}_k in time step k and the prior average $\tilde{T}^{LB}_k$. Now we search for the first time step $k+n$ that is a local minimum of

$$\tilde{T}_k(n) := \frac{1}{n}\left(\tilde{T}^{LB}_{k+n} + \sum_{i=k}^{k+n} T^k_i\right), \tag{63}$$

that is the first time step with

$$\frac{1}{n}\left(\tilde{T}^{LB}_{k+n} + \sum_{i=k}^{k+n} T^k_i\right) < \frac{1}{n+1}\left(\tilde{T}^{LB}_{k+n} + \sum_{i=k}^{k+n+1} T^k_i\right). \tag{64}$$

Under the assumption that load balancing creates a proper domain decomposition, that is at least as good as the domain decomposition obtained the last time the load balancing was carried out, it would have made sense to re-balance the system after time step $k+n$. However, as the time step $k+n+1$ has already finished in order to obtain the time T^k_{k+n+1}, re-balancing is carried out after time step $k+n+1$. If the wall time per time step is not increasing no $n \in \mathbb{N}$ exists that fulfills inequality (64). However, there can be situations in which the wall time per time step is not increasing, but the load distribution changes so that a better load balance becomes possible. Therefore it is useful to use a secondary criterion for load balancing, such as periodically calling the balancing algorithm of Section 3.5 every m steps.

4. Example applications

In this section we will discuss a series of examples to demonstrate the capabilities of the parallelized OpenPhase. In Section 4.1 we will take a look at normal grain growth for which tens of thousands of different phase fields are required.

In Section 4.3 results for Mg–Al alloy solidification simulations are presented, which compared to normal grain growth have a higher computational complexity and also require a large number of phase fields.

4.1. Normal grain growth

Structural materials are mainly formed from a very large number of grains (cluster of atoms) which are separated from each other by some crystalline properties such as orientation or composition. Polycrystalline bodies are subject to thermally activated change with respect to size and distribution of the grains known as 'grain growth' during which larger grains grow at the expense of smaller ones. Grain growth is a many-body problem for which multiphase field models [18,2,3] with large number of fields have been very promising. In practice, however, there are difficulties to deal with large number of grains. Early phase field simulations of grain growth suffered from poor statistics due to limited box size [19,20]. Even though the simulation capacity gradually grew, the number of possible independent phase fields was still a barrier. A solution for this problem was to introduce a finite number of grains repeatedly which improve the statistical accuracy of the results [21,22]. But yet another problem appeared as grains with the same indexes may coalesce somewhere along the simulation. These are perhaps some of reasons why it took so long till recently, to investigate the correlation between self-similarity and topological characteristics of normal grain growth [23].

In a previous study [24], the authors applied an efficient storage structure combined with OpenMP parallelization to investigate normal grain growth. Despite significant improvements in statistical and topological analysis, the outlook of the results points yet towards larger and more efficient simulations [23]. Thus a careful implementation of massive parallelization became evident.

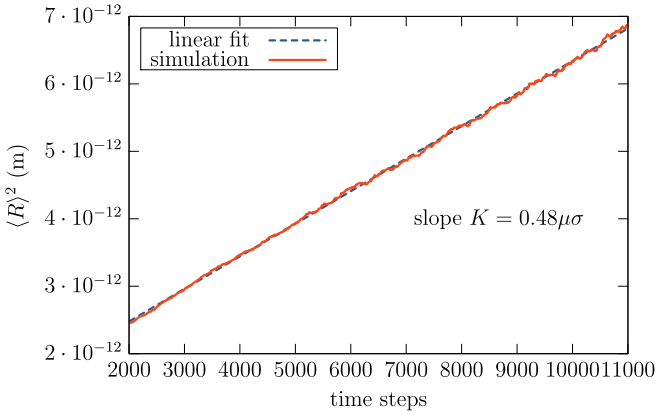


Fig. 6. Squared average grain sizes over time.

A simulation box of 512^3 is considered with 30 000 grains with a narrow distribution initialized in a Voronoi tessellation. Initial grains are small compared to the box size, but sufficiently large with respect to the grid spacing Δx and interfacial width $\eta = 6\Delta x$. Nevertheless, the early stages of simulation are omitted in the analysis to stay securely away from numerical artifacts. We have chosen an interfacial energy and mobility of 1 J m^{-2} and 10^{-14} m^4 J^{-1} s^{-1} respectively, within the accepted range for metallic systems. The time step and grid spacing are $dt = 10^{-1}$ s and $\Delta x = 10^{-7}$ m. In the following we present a series of benchmarks both for performance and accuracy of the results. The results of our study are compared against previous studies.

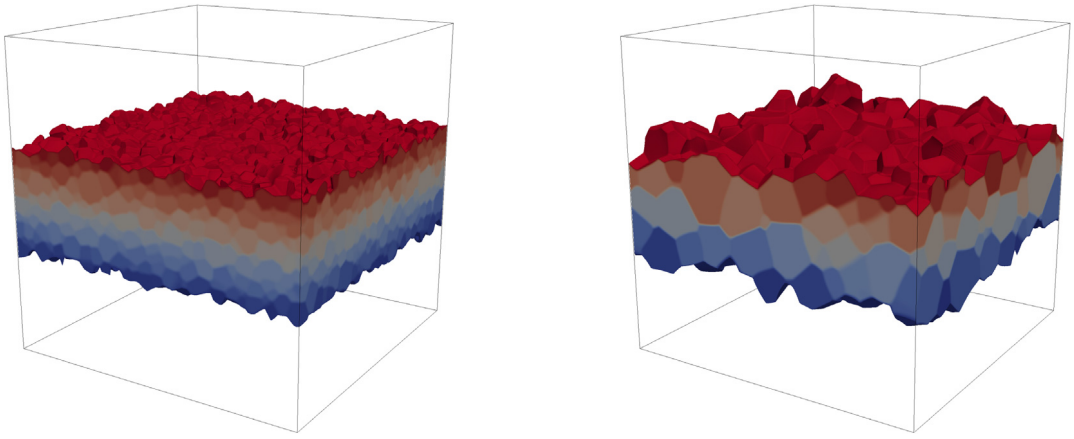
4.1.1. Results and discussions

A characteristic property of any polycrystalline is its averaged grain size that is subject to the changes during grain growth. For an ideal material, like in our case, a parabolic relation

$$\langle R\rangle^2 = Kt + R_0^2 \tag{65}$$

is proposed theoretically, where $K = AM\sigma$ and R_0 is the initial grain size. Here M and σ are interface mobility and energy, respectively, and parameter A reflects the overall geometry of grains. The results for the growth kinetics are shown in Fig. 6. The linear trend of the graph can be well described by Eq. (65) revealing the parabolic kinetics of the growth. In the simulations, the value for parameter A is obtained as 0.48 which is well comparable to the value 0.5 found in the previous study using serial computations [24,25]. Note that we present our results after 2000 initial time steps where the steady state growth is achieved. Fig. 7 shows the snapshots of the simulation after 2000 and 10 000 time steps.

Another characteristic measure of the system is the distribution of grains with respect to their size (GSD). Here also the parallel computation of our system finely recovers the GSD and the trend of its change over the entire simulation period. After about 2000 time steps, the size distribution matches the well-known Hillert's analytical solution [26] as seen in Fig. 8. The predicted deviation from this solution after 10 000 time steps is also well recovered. In fact the current parallel version of our program allows further investigation of the evolution of size distribution with significantly improved statistics. The physical interpretation of those simulations will be discussed elsewhere.



(a) Grains after 2000 time steps.

(b) Grains after 10 000 time steps.

Fig. 7. Normal grain growth simulation of Section 4.1.

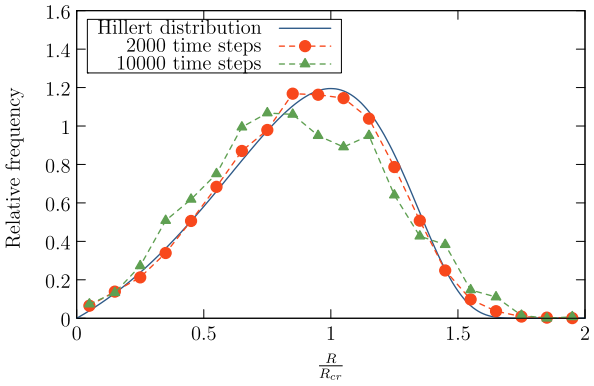


Fig. 8. Comparison of the simulation results to the Hillert distribution [26] with $R_{cr} = \frac{9}{8}\langle R\rangle$.

4.2. Scaling benchmark for grain growth simulations

In the following section we look at the performance in a series of benchmark tests that reflect the computational demands of grain growth simulations as discussed in Section 4.1. The benchmarks use exactly the same algorithms as the normal grain growth application in Section 4.1, however, for simplicity the initial data consists of random distribution of particles in a matrix. This saves the work of generating a Voronoi tessellation of the domain and allows the usage of the same particle distribution in the alloy solidification benchmarks in Section 4.3. Physically this simulation corresponds to precipitate coarsening in a solid matrix. Precipitates and matrix are of the same thermodynamic phase, but have different crystallographic orientations.

For the benchmark we setup the simulation using N blocks with

$$N = 2N_{\text{MPI}}, \tag{66}$$

where N_{MPI} is the number of MPI processes. Each block has a size of $64 \times 64 \times 64$ grid points in single threaded benchmarks, with two threads per process the size of each blocks is $128 \times 64 \times 64$ and with four threads it is $128 \times 128 \times 64$. The blocks are assembled in a way so that the computational domain Ω forms a cuboid with the minimal surface area. Therefore we have 12 582 912 grid points for 24 cores and go up to 402 653 184 grid points for 768 cores.

In actual simulations the initial conditions are generated by a Dirichlet decomposition acquired from a random distribution of center points, using Voro++ [27]. However, for testing purposes we will generate spherical particles at random positions with a uniform and a Gaussian distribution of the particle positions as depicted in Fig. 9. A halo size of two layers of grid points is used, one layer is needed for the stencil operation in Eq. (26) and another layer is needed to tag grid points as interface or near-interface. In this test the load distribution does not change rapidly and once a domain decomposition with a suitable load balance is found no further load balancing steps need to be performed. For the first 300 time steps the load balancing algorithm as presented in Section 3.5 will be called every 10 time steps. This is done to give the algorithm enough steps to adapt the block sizes as in on balancing step each block is only allowed to be split once or merged once. The timing results are obtained from the average of the next 100 time steps, during which no further load balancing occurs.

4.2.1. Uniform distribution of particles

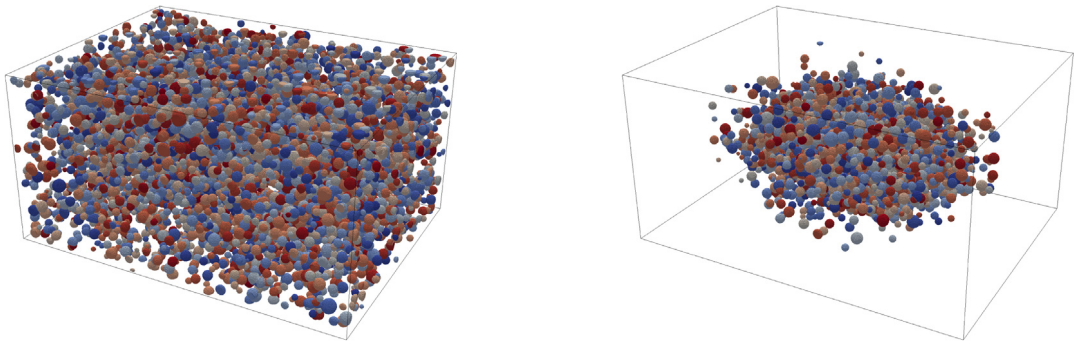
In the domain Ω we embed $N_p = 10 \times N_{\text{MPI}} \times N_{\text{OpenMP}}$ spherical particles at random coordinates. The particles are distributed uniformly in the domain and the particle sizes follow a normal distribution with a mean radius of $\mu_r = 10$ and a standard deviation $\sigma_r = 2$. This test represents a simulation without significant load imbalances.

In this test only very minor load-imbalances occur, however, as seen in Fig. 10 load balancing using bisection and either METIS or the greedy graph-partitioner mentioned in Section 3.1 is beneficial to the performance.

4.2.2. Gaussian distribution of particles

In the domain Ω a number $N_p = 10 \times N_{\text{MPI}} \times N_{\text{OpenMP}}$ of spherical particles are embedded at random coordinates. The particle positions follow a normal distribution with a mean of $\mu_d = \frac{N_d}{2}$ and a standard deviation $\sigma_d = \frac{N_d}{8}$ for the dimensions $d \in \{x, y, z\}$ and the particle sizes again follow a normal distribution with a mean radius of $\mu_r = 10$ and a standard deviation $\sigma_r = 2$. That means the particles are clustered near the center of the domain Ω and the phase fields form many multiple junctions, which leads to severe load imbalances when using a simple domain decomposition (see Fig. 9b).

The benchmark results in Fig. 11 show that especially for a large number of cores the performance deteriorates without load-balancing. The utilization of load-balancing consisting of the bisection of blocks and either METIS or the greedy graph-partitioner to assign blocks to process provides a high weak-scaling efficiency. In Fig. 12 the performance using dynamic resizing of blocks is compared to static block sizes using METIS as the graph-partitioner.



(a) Uniform particle distribution.

(b) Gaussian particle distribution.

Fig. 9. Initial conditions for the benchmarks in Section 4.2. Both start with the same number of particles, however on the right hand side, the particles in the center of the domain form more multiple junction which increases the computational load severely, while the amount of work near the boundary of the domain is very low.

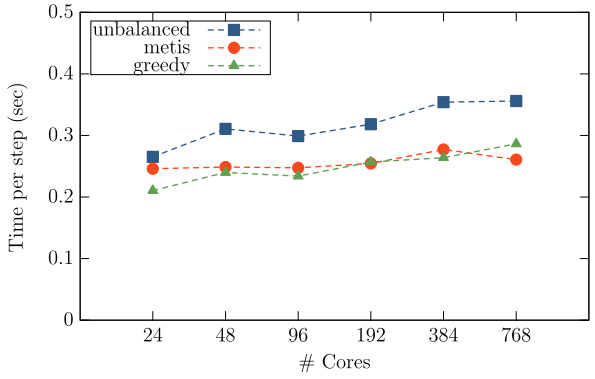


Fig. 10. Grain growth scaling benchmark for a uniform particle distribution, see Section 4.2.1. Although in this case there are no significant load-imbalances, load-balancing using either METIS or the greedy graph-partitioner provided with OpenPhase are beneficial to the performance.

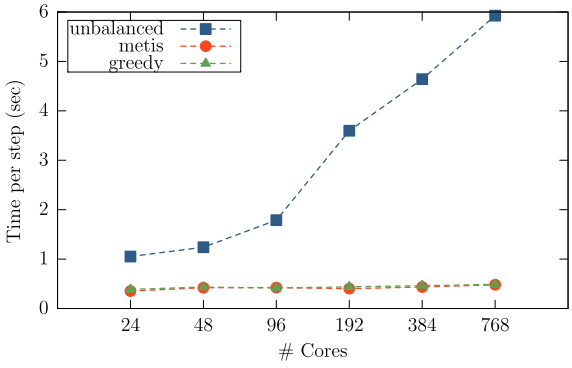


Fig. 11. Grain growth scaling benchmark for a Gaussian particle distribution, see Section 4.2.2. In this case the load-imbalances are very significant and with load-balancing the parallelization does not scale. With load-balancing using either METIS or the greedy graph-partitioner provided with OpenPhase we observe a proper weak-scaling performance.

Bisection of blocks results is the best performance and further refinement of blocks will not deliver better results for the static block sizes as Fig. 13 shows that the overhead created by small blocks already raised the average load $\tilde{W}$ per partition over load of the maximum partition using dynamic resizing of blocks. Fig. 14 shows the performance using different numbers of OpenMP threads, while still using the same number of total cores. For a small problem size with a total of 24 cores the hybrid parallelization with 6 threads per MPI process shows the best performance. In this case the 4 processes are assigned to 8 blocks with similar computational load because the particles are clustered near the center of the domain. Therefore a good load balance is achieved without using the load balancing algorithm in Section 3.5, which means that less overhead is created by the bisection of blocks. The situation is the same for up 48 cores, when using 12 threads per process. For larger system sizes the results show a similar performance for 4 and 6 threads per process. A pure MPI-parallelization shows a slightly worse performance, except for 384 cores. For large system sizes the performance deteriorates significantly when using 24 threads per MPI-process. The current implementation only allows one thread to use MPI calls, which decreases the rate of parallelism. Hybrid-parallelism is further investigated for the simulation of Mg–Al alloy solidification in Section 4.4.

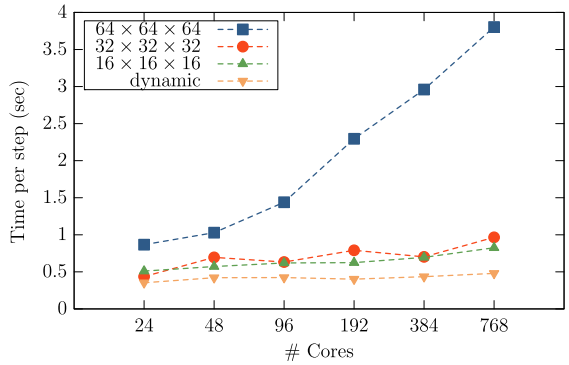


Fig. 12. Grain growth scaling benchmark for a Gaussian particle distribution, see Section 4.2.2. Comparing the dynamic splitting and merging of blocks to static block sizes using METIS for graph-partitioning in all cases. Using dynamic blocksizes results in the best performance in this test.

4.2.3. Moving particles

In the tests above grains are stationary and therefore the distribution of the computational load does not change significantly over time. In this test we use a similar setup as in Section 4.2.2, but the grains move one grid point along the x-direction every 5 time steps, in order to create a dynamic load distribution. The computational domain is a perfect cube consisting of N_b blocks in each direction, that have an edge length of N_e. N_b and N_e are determined by

$$N_b = \max(\lceil N_{\text{MPI}}^{\frac{1}{3}} \rceil, 3), \tag{67}$$

$$N_e = \left\lfloor \frac{\lfloor \left(2 \cdot 64^3 N_{\text{MPI}} N_{\text{OpenMP}}\right)^{\frac{1}{3}} \rfloor}{N_b} \right\rfloor \tag{68}$$

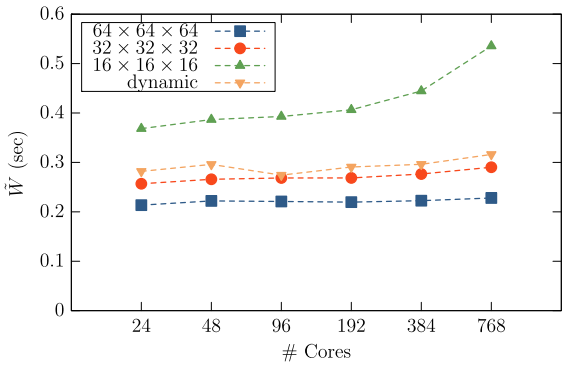


Fig. 13. Grain growth scaling benchmark for a Gaussian particle distribution, see Section 4.2.2. This shows the average load $\tilde{W}$ per time step using different blocksizes. For $16 \times 16 \times 16$ blocks the average load $\tilde{W}$ already exceeds the time per step of the dynamic blocksizes in Fig. 12, which means that using even smaller blocks will not perform better than using the dynamic blocksizes.

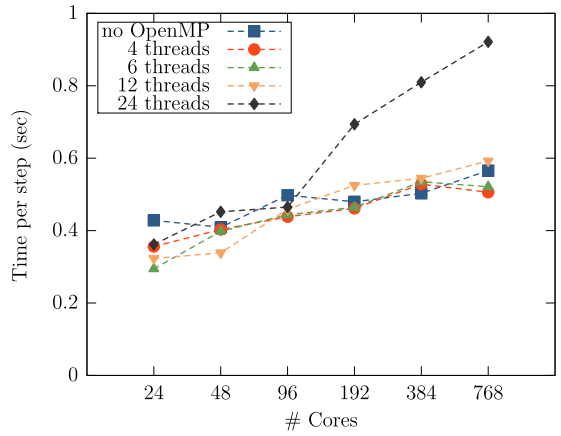


Fig. 14. Grain growth scaling benchmark for a Gaussian particle distribution, see Section 4.2.2. Comparison of the performance using a varying number of OpenMP threads per MPI-process.

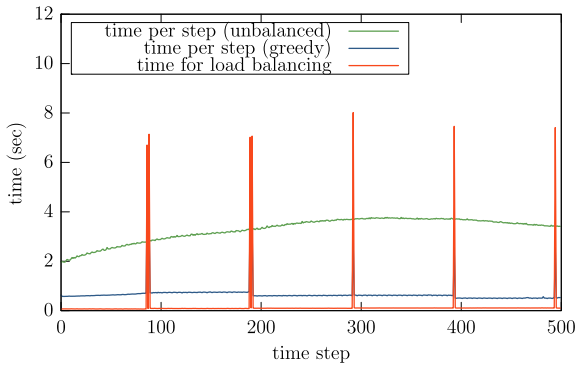


Fig. 15. Comparison of the wall time per time step for the simulation of moving particles in Section 4.2.3 with load balancing and without load-balancing for 192 processes.

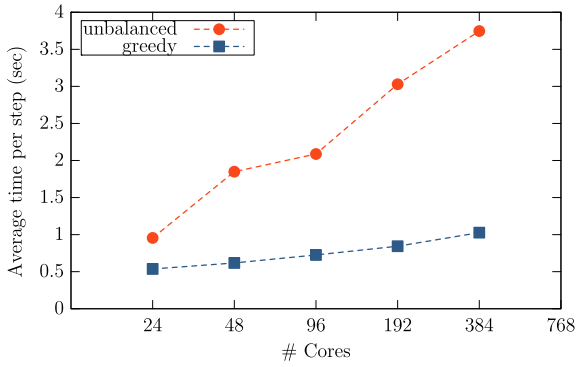


Fig. 16. Scaling behavior for Example 4.2.3 using the greedy graph partitioner and a simple domain decomposition using the average wall time per step $\tilde{T}$, which includes the time used for load balancing.

with the number of MPI processes N_{MPI} and the number of OpenMP threads per process N_{OpenMP}. In the simulation with load balancing an initial ten load balancing steps are used before the start of the benchmark in order to let the load balancing algorithm adapt to the initial configuration. In Fig. 15 we see the wall times per time step on 192 processes with dynamic load balancing as well as for a simple static domain decomposition. We notice that the load balancing steps can take as long as about ten time steps in this example. The simulation uses $\lambda = 0.5$ for Eq. (62) and balances every $m = 100$ time steps if inequality (64) is not fulfilled or if a block B exists with

$$W_B \geq 2(1+\gamma)\tilde{W}, \tag{69}$$

which triggers the secondary load balancing steps in steps 89 and 192. Fig. 16 shows the scaling behavior for the average wall time per step $\tilde{T}$, which includes the time used for load balancing.

4.3. Mg–Al alloy solidification

Because of their low density, Mg–Al alloys are most commonly used for structural lightweight application in automotive and aerospace engineering and in high-end consumer electronics. However, the low nobility leads to corrosion sensitivity in Mg–Al alloys, especially in contact with other metals. Mg–Al cast alloys usually consist of a primary HCP-Mg-α phase which solidifies in a dendritic fashion. These dendrites are then enclosed during the solidification of the interdendritic eutectic, of which the majority is the inter-metallic $Mg_{17}Al_{12}$-β-phase. One of the many strategies to inhibit the galvanic corrosion in Mg–Al alloys is to prevent networks of primary α-phase grains from forming. Ensuring a closed shell of β-phase around the individual primary dendrites, significantly increased corrosion resistance [28]. To understand the microstructure evolution of as-cast Mg–Al alloys during solidification and to be able to predict the connectivity of the β-phase depending on process parameters as composition and cooling rate, phase field studies have been carried out and will be briefly explained below. Fig. 17 shows a simulation of Mg–Al alloy solidification on a grid of 300^3 grid points. For further information see our publications [29,30,11].

The parallelization techniques, explained in Section 3 were used to enable a large scale simulation of a complete solidification process of a Mg-5at.%Al alloy. This simulation shows the sequential nucleation and growth of primary α- and in a later stage secondary β-phase. To recover the physics behind the process, multiple solvers have to be used. The phase field evolution with the driving-force evaluation has to be calculated. This is of course dependent on the local concentration of Al and the temperature. While the diffusion equation is solved to model the evolution of the composition field, the heat is assumed to diffuse orders of magnitude faster than the diffusion and is modeled using a heat-balance equation. The inclusion of the latent heat term creates the necessity to know at every time step how much phase was transforming in the whole domain. The nucleation events were modeled such that a list of virtual particles was generated pseudo-randomly with a position and size each. Then at every time step it is checked if a nucleation is taking place on that particle by evaluation

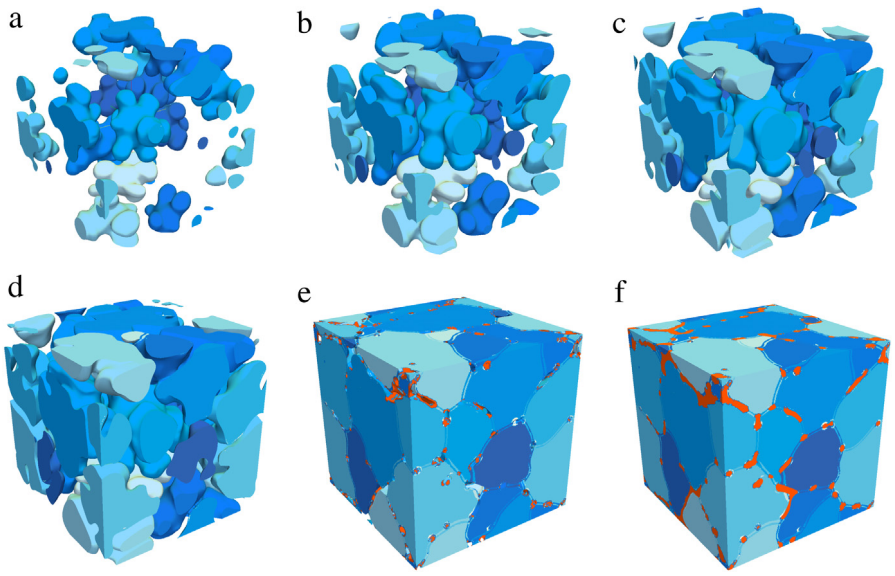


Fig. 17. Simulation of Mg–Al alloy solidification.

of the local composition and temperature. In case of a high enough under-cooling, a new grain is planted at the exact position of that particle.

4.4. Scaling benchmark for Mg–Al alloy solidification

Compared to the grain growth simulation in Section 4.1 this simulation requires more work. It includes the calculation and averaging of the driving force based on the local composition of magnesium and aluminum and the temperature, as described in Section 2.3. In addition to that the diffusion of aluminum in magnesium needs to be calculated as well. The calculation of the laplacian of the phase fields and the concentrations require stencils that access the next neighbors. In this case we need $\lfloor \frac{2}{3}\eta \rfloor$ boundary layers for each of the driving force averaging steps in Eqs. (28) and (29), one for the tagging of grid points as interface and near-interface points as well as one for the diffusion in Eq. (13) and one additional layer is needed to compute the anti-trapping current $J_{\alpha\beta}$. The stencil operation in Eq. (26) can be computed simultaneously and does not increase the halo size. With an interface width $\eta = 5$ grid points a total of six grid points is needed for the driving force averaging, which makes a halo size of nine grid points necessary.

For the scaling tests we use a domain Ω consisting of N blocks with

$$N = 2N_{\text{MPI}}, \tag{70}$$

where N_{MPI} is the number of MPI processes. Each block has a size of 48 ×48 ×48 grid points plus nine boundary points in each direction in single threaded benchmarks, with two threads per process the size of each blocks is 96 ×48 ×48 and with four threads it is 96 ×96 ×48. The blocks are again assembled in a way so that the computational domain Ω forms a cuboid with the minimal surface area. Therefore we have 5 308 416 grid points for 24 cores and go up to 169 869 312 grid points for 768 cores.

4.4.1. Uniform distribution of particles

As the computational complexity in this test is significantly higher compared to the normal grain growth example in Section 4.1 we only embed $N_p = 2 \times N_{\text{MPI}} \times N_{\text{OpenMP}}$ spherical particles in the domain Ω. The particles are distributed uniformly in the domain and the particle sizes follow a normal distribution with a mean radius of $\mu_r = 10$ and a standard deviation $\sigma_r = 2$. This test represents a simulation without significant load imbalances.

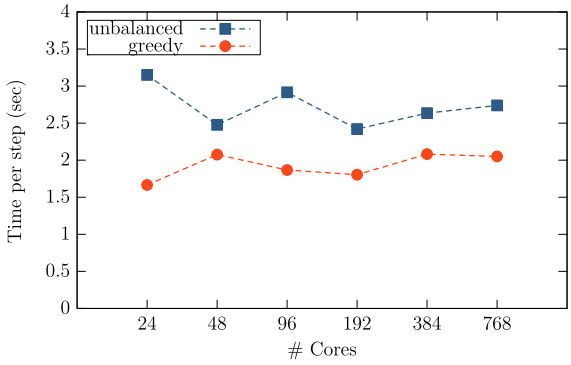


Fig. 18. Mg–Al alloy solidification benchmark for a uniform particle distribution, see Section 4.4.1. Although the particles are distributed uniformly in the domain some small load-imbalances occur and load-balancing improves the performance.

4.4.2. Gaussian distribution of particles

In the domain Ω a number $N_p = 2 \times N_{\text{MPI}} \times N_{\text{OpenMP}}$ of spherical particles are embedded at random coordinates. The particle positions follow a normal distribution with a mean of $\mu_d = \frac{N_d}{2}$ and a standard deviation $\sigma_d = \frac{N_d}{8}$ for the dimensions $d \in \{x, y, z\}$ and the particle sizes again follow a normal distribution with a mean radius of $\mu_r = 10$ and a standard deviation $\sigma_r = 2$. That means the particles are clustered near the center of the domain Ω and the phase fields form many multiple junctions, which leads to severe load imbalances when using a simple domain decomposition (see Fig. 9b).

4.4.3. Uniform particle distribution

At first we take a look at the performance for a uniform distribution of particles as mentioned in Section 4.4.1. The results are depicted in Fig. 18. As for the normal grain growth we again observe that load-balancing improves the performance even for a uniform particle distribution without significant load-imbalances.

4.4.4. Gaussian particle distribution

For the Gaussian particle distribution as in Section 4.4.2 the scaling results are presented in Fig. 19. As in the grain growth benchmark in Section 4.2.2 load-balancing drastically improves the performance. Without load-balancing a very poor weak scaling

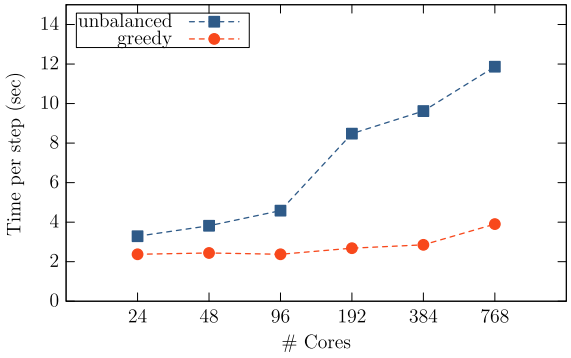


Fig. 19. Mg–Al alloy solidification benchmark for a Gaussian particle distribution, see Section 4.2.2. In this case the load-imbalances are very significant and the performance deteriorates without load-balancing.

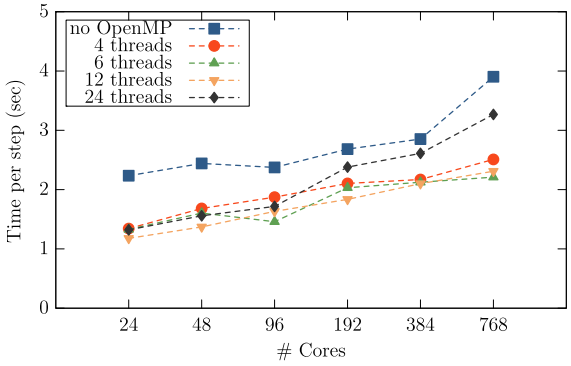


Fig. 20. Mg–Al alloy solidification benchmark for a Gaussian particle distribution, see Section 4.2.2. Comparison of the performance with load-balancing using a varying number of OpenMP threads per MPI-process. 6 or 12 threads per MPI process provide the best performance in this test.

behavior is observed. With load-balancing, however, the time per step remains close to 5 s. Using the hybrid approach is beneficial in this case as seen in Fig. 20. Using 6 or 12 threads yields the best results across all problem sizes. In contrast to the grain growth benchmark in Section 4.1 the workload per time step in the example is high compared to the amount of communication. This makes a hybrid-parallelization viable even when only one thread is allowed to make MPI calls. Also we can see that without OpenMP the time per step shoots up at 768 cores. This is caused by a single block that cannot be further split but remains so costly as to cause a severe load-imbalance. Also the hybrid parallelization results in larger block sizes as mentioned in Remark 9, which causes less duplicated computations on the wide halo and thus is more efficient. For 24 threads per MPI process the results for 96 or less cores, i.e. 1 to 4 MPI-processes or 2–8 blocks, are on par with 6 or 12 threads per process. In these cases every block has about the same computational load and thus a good load-balance is achieved without resizing or reassignment of blocks — the same is true for up 48 cores when using 12 threads per MPI-process or for 24 cores when using 6 threads per MPI-process. Using more cores, however, load-imbalances between blocks occur and blocks are split and reassigned. Some blocks become too small for an efficient OpenMP parallelization using 24 threads, which causes the deterioration of the performance seen in Fig. 20.

5. Outlook and conclusion

We have presented a parallel implementation of OpenPhase that allows 3d-simulations on large domains and shown its capabilities on example problems, namely grain growth and Mg–Al alloy solidification. The distributed-memory parallelism enables simulations that were previously impossible due to memory constraints or unreasonable long wall times. These benchmarks have shown that load balancing is critical in achieving a good performance when the computational load is distributed inhomogeneously, while also showing a slight performance benefit for homogeneous load distributions. The adaptive splitting and merging of sub-domain has a significant impact on the performance, while the choice of the graph-partitioner is less important in order to achieve a proper load-balance. The wide halo approach contributes a substantial amount of computational overhead for complex problem, therefore the splitting of time steps into multiple segments, that need to be balanced individually by multi-constraint load-balancing, will be examined in the future. This is particularly important for problems in which a single synchronization point within one time step is not feasible, e.g. systems that include fluid dynamics, heat transport and similar that use different time scales.

Acknowledgments

This work was financially supported by ThyssenKrupp Steel Europe AG and the Fundamental Research Program of the Korea Institute of Materials Science (KIMS). The authors gratefully acknowledge the computing time granted on the supercomputer JURECA at Jülich Supercomputing Centre (JSC).

References

[1] OpenPhase. URL http://www.openphase.de.
[2] L.-Q. Chen, Ann. Rev. Mater. Res. 32 (1) (2002) 113–140. http://dx.doi.org/10.1146/annurev.matsci.32.112001.132041.
[3] I. Steinbach, Model. Simul. Mater. Sci. Eng. 17 (7) (2009) 073001. URL http://stacks.iop.org/0965-0393/17/i=7/a=073001.
[4] I. Steinbach, J. Miner. Met. Mater. Soc. (ISSN: 1047-4838) 65 (9) (2013) 1096–1102. http://dx.doi.org/10.1007/s11837-013-0681-5.
[5] I. Steinbach, Ann. Rev. Mater. Res. 43 (1) (2013) 89–107. http://dx.doi.org/10.1146/annurev-matsci-071312-121703.
[6] W.L. George, J.A. Warren, J. Comput. Phys. (ISSN: 0021-9991) 177 (2) (2002) 264–283. http://dx.doi.org/10.1006/jcph.2002.7005.
[7] A. Vondrous, M. Selzer, J. Hötzer, B. Nestler, Int. J. High Perform. Comput. Appl. (ISSN: 1094-3420) 28 (1) (2014) 61–72. http://dx.doi.org/10.1177/1094342013490972.
[8] T. Shimokawabe, T. Aoki, T. Takaki, T. Endo, A. Yamanaka, N. Maruyama, A. Nukada, S. Matsuoka, Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, in: SC'11, ACM, New York, NY, USA, ISBN: 978-1-4503-0771-0, 2011, pp. 3:1–3:11. http://dx.doi.org/10.1145/2063384.2063388. URL http://doi.acm.org/10.1145/2063384.2063388.
[9] PACE3D. URL http://www.iaf.hs-karlsruhe.de/ice/ice/nestler/index.php?id=113.
[10] Micress®. URL http://web.micress.de/.
[11] M. Tegeler, A. Monas, G. Sutmann, Proceedings of Fourth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, http://dx.doi.org/10.4203/ccp.107.5.
[12] I. Steinbach, M. Apel, Physica D 217 (2006). http://dx.doi.org/10.1016/j.physd.2006.04.001.
[13] F.B. Kjolstad, M. Snir, Proceedings of the 2010 Workshop on Parallel Programming Patterns, in: ParaPLoP'10, ACM, New York, NY, USA, ISBN: 978-1-4503-0127-5, 2010, pp. 4:1–4:9. http://dx.doi.org/10.1145/1953611.1953615. URL http://doi.acm.org/10.1145/1953611.1953615.
[14] K. Andreev, H. Räcke, Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, in: SPAA'04, ACM, New York, NY, USA, ISBN: 1-58113-840-7, 2004, pp. 120–124. http://dx.doi.org/10.1145/1007912.1007931. URL http://doi.acm.org/10.1145/1007912.1007931.
[15] E.G. Boman, U.V. Catalyurek, C. Chevalier, K.D. Devine, Sci. Program. 20 (2) (2012) 129–150. http://dx.doi.org/10.1155/2012/713587.
[16] G. Karypis, V. Kumar, SIAM J. Sci. Comput. 20 (1) (1998) 359–392. http://dx.doi.org/10.1137/S1064827595287997.
[17] D.E. King, J. Mach. Learn. Res. 10 (2009) 1755–1758. URL http://jmlr.csail.mit.edu/papers/volume10/king09a/king09a.pdf.
[18] M.T. Lusk, Proc. R. Soc. Lond. A: Math. Phys. Eng. Sci. (ISSN: 1364-5021) 455 (1982) (1999) 677–700. http://dx.doi.org/10.1098/rspa.1999.0329.

[19] L.-Q. Chen, Scr. Metall. Mater. (ISSN: 0956-716X) 32 (1) (1995) 115–120. http://dx.doi.org/10.1016/S0956-716X(99)80022-3. URL http://www.sciencedirect.com/science/article/pii/S0956716X99800223.
[20] D. Fan, L.-Q. Chen, Acta Mater. (ISSN: 1359-6454) 45 (2) (1997) 611–622. http://dx.doi.org/10.1016/S1359-6454(96)00200-5. URL http://www.sciencedirect.com/science/article/pii/S1359645496002005.
[21] C.K. III, L.-Q. Chen, Acta Mater. (ISSN: 1359-6454) 50 (12) (2002) 3059–3075. http://dx.doi.org/10.1016/S1359-6454(02)00084-8. URL http://www.sciencedirect.com/science/article/pii/S1359645402000848.
[22] Y. Suwa, Y. Saito, H. Onodera, Scr. Mater. (ISSN: 1359-6462) 55 (4) (2006) 407–410. http://dx.doi.org/10.1016/j.scriptamat.2006.03.034. URL http://www.sciencedirect.com/science/article/pii/S1359646206002405.
[23] R. Darvishi Kamachali, A. Abbondandolo, K. Siburg, I. Steinbach, Acta Mater. (ISSN: 1359-6454) 90 (2015) 252–258. http://dx.doi.org/10.1016/j.actamat.2015.02.025. URL http://www.sciencedirect.com/science/article/pii/S1359645415001251.
[24] R. Darvishi Kamachali, I. Steinbach, Acta Mater. 60 (2012) 2719–2728. http://dx.doi.org/10.1016/j.actamat.2012.01.037.
[25] R. Darvishi Kamachali, Grain Boundary Motion in Polycrystalline Materials (Ph.D. thesis), RuhrUniversity Bochum, Interdisciplinary Centre for Advanced Materials Simulation, STKS, 2013. URL http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/DarvishiKamachaliReza/diss.pdf.
[26] M. Hillert, Acta Metall. (ISSN: 0001-6160) 13 (3) (1965) 227–238. http://dx.doi.org/10.1016/0001-6160(65)90200-2. URL http://www.sciencedirect.com/science/article/pii/0001616065902002.
[27] C.H. Rycroft, Chaos 19 (4) (2009). http://dx.doi.org/10.1063/1.3215722. URL http://scitation.aip.org/content/aip/journal/chaos/19/4/10.1063/1.3215722.
[28] M.-C. Zhao, M. Liu, G. Song, A. Atrens, Corros. Sci. (ISSN: 0010-938X) 50 (7) (2008) 1939–1953. http://dx.doi.org/10.1016/j.corsci.2008.04.010.
[29] A. Monas, O. Shchyglo, D. Höche, M. Tegeler, I. Steinbach, IOP Conf. Ser.: Mater. Sci. Eng. 84 (1) (2015) 012069. URL http://stacks.iop.org/1757-899X/84/i=1/a=012069.
[30] A. Monas, O. Shchyglo, S.-J. Kim, C.D. Yim, D. Höche, I. Steinbach, J. Miner. Met. Mater. Soc. (ISSN: 1047-4838) 67 (8) (2015) 1805–1811. http://dx.doi.org/10.1007/s11837-015-1418-4. URL http://dx.doi.org/10.1007/s11837-015-1418-4.