# Massively Parallel Stencil Code Solver with Autonomous Adaptive Block Distribution

Marco Berghoff, Ivan Kondov, Johannes Hötzer

**Abstract**—In the last decades, simulations have been established in several fields of science and industry to study various phenomena by solving, inter alia, partial differential equations. For an efficient use of current and future high performance computing systems, with many thousands of computation ranks, high node-level performance, scalable communication, and the omission of unnecessary calculations are of high priority in the development of new solvers. The challenge of contemporary simulation applications is to bridge the gap between the scales of the various physical processes. We introduce the NAStJA framework, a block-based MPI parallel solver for arbitrary algorithms, based on stencil code or other regular grid methods. NAStJA decomposes the domain of spatially complex structures into small cuboid blocks. A special feature of NAStJA is the dynamic block adaption which modifies the calculation domain around the region where the computation currently takes place, and hence avoids unnecessary calculations. This often occurs, inter alia, in phase-field simulations. Block creation and deletion is managed autonomously within local neighborhoods. A basic load balancing mechanism allows a re-distribution of newly created blocks to the involved computing ranks. The use of a multi-hop network, to distribute information to the entire domain, avoids collective all-gather communications. Thus, we can demonstrate excellent scaling. The present scaling tests substantiate the enormous advantage of this adaptive method. For certain simulation scenarios, we can show that the calculation effort and memory consumption can be reduced to only $3.5\,\%$, compared to the classical full-domain reference simulation. The overhead of $70 - 100\,\%$ for the dynamic adapting block creation is significantly lower than the gain. The approach is not restricted to phase-field simulations, and can be employed in other domains of computational science to exploit sparsity of computing regions.

**Index Terms**—stencil code, distributed memory, scalable parallel algorithms, massively parallel performance, multi-hop network, load balancing, partial differential equation, phase-field method

## 1 Introduction

Computer simulations help to understand the theory behind phenomena, and especially to investigate those that are difficult to access in experiments. With increasing computing power, simulations become larger and more complex. This leads to an increase of the resolution, the accuracy, and the gain in knowledge. In computational materials science, simulations with the phase-field method reach sizes that open up new ways for microstructure analysis. Especially for ternary eutectic directional solidification, large-scale simulations [1], [2] of realistic 3D microstructure evolution show spiral growth [3] and pattern formation without influences from the periodic boundary condition [4]. Similarly, large-scale simulations of dendritic growth [5] enabled new insights into the growth morphology [6], [7]. However, the required high resolution still remains one of the biggest challenges. A problem which is encountered with such a high resolution is the modeling of water droplets on structured surfaces. The phase-field method is able to simulate multiphase droplets [8] which are difficult to calculate with traditional tools, such as the Surface Evolver [9], [10].

Contrary to sharp interface models, finite differences on regular grids can be used to discretize the phase-field method. The individual material parameters are interpolated over the diffuse interface. This keeps the calculation rules the same throughout the domain. In contrast to sharp interface models, the interface does not need to be tracked with high computational effort. To enable accurate calculations, the mesh must be adapted according to the interfacial changes.

For certain phase-field applications, only the diffuse interface region between the various phases has to be calculated. The bulk, i.e., the part inside the phases, does not change in these calculations. For illustration purposes, we use an example application for the presented method, which is particularly suitable; for example, a water droplet on a structured surface. Bridging the scales between a few micrometers, for the structured surface, and a few millimeters, for the diameter of a water droplet, requires large domain sizes. If the water droplet is covered with a cuboidal simulation domain, most of the domain contains bulk, which does not require computation. The interface region that has to be calculated is very small. For a smooth discretization of the interface region, a width of about ten grid points is necessary. Water droplets on a chemically structured surface, with 700 lamellae, have been studied experimentally. However, this has not been achieved in simulations [11], [12]. A typical rain droplet of about 3 mm has the size to cover all 700 lamellae, each with size of

- *M. Berghoff and I. Kondov are with the Steinbuch Centre for Computing (SCC), Karlsruhe Institute of Technology (KIT), Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany. E-mail: {marco.berghoff,ivan.kondov}@kit.edu.*
- *J. Hötzer is with the Institute for Applied Materials (IAM), Karlsruhe Institute of Technology (KIT), Straße am Forum 7, 76131 Karlsruhe, Germany, Institute of Materials and Processes, Karlsruhe University of Applied Sciences, Moltkestrasse 30, D 76133 Karlsruhe, Germany. E-mail: johannes.hoetzer@kit.edu.*

*Manuscript received September 14, 2017; revised February 2, 2018.*

about 5 $\mu$m—the typical diameter of spider dragline silk. In a phase-field simulation, the lamellae should be significantly wider than the diffuse interface. For 15 grid points per lamella, this results in a resolution of around $6 \cdot 10^{11}$ grid points, in all three dimensions. To meet this challenge, we have developed new approaches, and have implemented them in the NAStJA[1] framework. The regions outside the interface do not have to be calculated, and thus are omitted in NAStJA, which responds to dynamic changes in the simulation domain. Details will be presented in the section "Methods".

Since NAStJA is designed to be both highly efficient and as general as possible, it supports regular grids with a computing stencil sweeping through the domain. Many problems in computational materials science and other fields can be described using phase-field methods or other methods based on regular grids, as cellular automata, which both can be tackled using NAStJA. Besides the phase-field method, the NAStJA framework supports different methods for droplets, as was done by Ben Said et al. [8]. Other methods that NAStJA can handle are phase-field crystal models like [13], [14] and the cellular Potts model, a cellular automaton for biological cell simulations, according to Graner et al. [15]. The framework can be simply extended with a wide range of algorithms that work on finite difference schemes or other regular grid methods.

The next section gives an overview of related work. Section 3 presents details of the methods used by NAStJA, beginning with a brief introduction of the phase-field method. Then, the data structure and the different communication layers are shown, followed by the dynamic methods. Measurements and a theoretical estimate of the parallel scaling are presented in Section 4. Finally, we discuss the results in Section 5, and draw the conclusions.

## 2 Related Work

A brief list of phase-field frameworks is given in Section 2.1. Section 2.2 presents certain methods to reduce the computational cost by adapting the mesh. This section closes with a brief overview of autonomously distributed systems in Section 2.3.

### 2.1 Frameworks

In the last years, various frameworks with different design goals have been established to conduct phase-field simulations. Despite the different programming languages, ranging from C, C++ to Python, the most frameworks are parallelized with Message Passing Interface (MPI) to exploit current high performance computing (HPC) systems. Open-source frameworks are DUNE [16], [17], FEniCS [18], [19], FiPy [20], [21], MOOSE [22], [23], OpenPhase [24], waLBerla [1], [25], and PRISMS [26]. Apart from these, proprietary codes are also developed, such as MICRESS [27], [28], Pace3D [29], [30], and COMSOL [31]. Most of these frameworks focus on the usability, a wide range of features, already implemented modules, and the flexibility to incorporate new models. Although waLBerla has a flexible block-based concept [32], and a high focus on performance, it does not allow to simulate arbitrary rectangular decomposed domains, where blocks are newly created and removed, allowing the decomposition to follow the structure during the simulation. This dynamic creation and deletion of blocks, to change the decomposition of arbitrary domains during the simulation, is one of the primary objectives of NAStJA.

1. Acronym: Neoteric Autonomous STencil code for Jolly Algorithm

### 2.2 Adaptive Mesh Refinement

Besides the ability to dynamically change the computational domain on a block level, as NAStJA does, adaptive mesh refinement (AMR) techniques are used in phase-field simulations of dendritic growth [33], [34], [35], [36], cell growth [37], [38], growth of lamellar structures [39], and wetting phenomena [40], [41]. For this purpose, the grid resolution is dynamically changed in local regions with high gradients. An overview of various dynamic AMR algorithms is summarized by [42].

### 2.3 Agent-based Modeling

Agent-based modeling is a bottom-up method to model a wide range of dynamic systems in various fields, such as artificial life, genetic programming, genetic evolution, or social studies. Individual elements of the system are computationally represented as agents with decision-making or action options. The system behavior results from the behavior of the individual agents, and is not predefined at system level. The distribution and action of adaptive autonomous agents is the subject of many studies [43], [44], [45]. In a certain way, the blocks of NAStJA can be mapped to agents.

## 3 Methods

In this section, we first present a brief overview of the phase-field method and its equations, and outline the general requirements for NAStJA. Then the general data structure, the program flow, and the different communication layers in NAStJA are introduced. The method for the dynamic block adaption is also addressed in more detail.

### 3.1 The Phase-field Method

The phase-field method is based on an entropy [29], a free energy [46], or a grand-potential functional [2], [46]. It is developed to ensure consistency with classical irreversible thermodynamics. An order parameter $\phi_\alpha$ describes the local fraction of phase $\alpha$. In the phase-field method, a phase indicates the aggregate state, such as solid or liquid, the different grain orientation in multigrain simulations, or several materials, e.g., water, oil, or air in droplet simulations. Each phase fulfills the relation $0 \leq \phi_\alpha \leq 1$, where $\phi_\alpha = 0$ denotes the absence of phase $\alpha$, and $\phi_\alpha = 1$ denotes the existence of the phase. All phases must satisfy $\sum_\alpha \phi_\alpha = 1$. The diffuse transition from 0 to 1 is called interface. In particular, we have two phases and two order parameters that effectively reduce to one order parameter $\phi$, because of the complementarity condition. In the present case, it is sufficient to use only one order parameter $\phi$. A detailed

derivation can be found in [47], [48]. The simplified phase-field equation reads as

$$\frac{\partial \phi}{\partial t} = \varepsilon \nabla^2 \phi - \frac{32}{\varepsilon \pi^2}(1 - 2\phi) - 6\phi(1 - \phi)\Delta f, \quad (1)$$

with $\varepsilon$ as a parameter related to the interface width $\Lambda = \varepsilon\pi^2/8$, and $\Delta f$ as a driving force which is responsible for the interface movement.

A common way to discretize the phase-field equation (1) is to use a finite difference scheme and an Euler time integration [49]. Therefore, the coupling of various data fields, via synchronized time steps, which is particularly important in multiphysics applications, is thus easily possible. According to the notation introduced by [1], an adequate discretization for the Laplacian is the D3C7 stencil, a three-dimensional stencil with seven input values, a center point, and six direct neighbor points. The size of the stencil determines the size of additional layers which are necessary for the discretization. These layers are wrapped around the data fields, and are called halo. In case of this phase-field model, the halo extends the data field by one grid point in each direction.

### 3.2 Architecture of NAStJA

The NAStJA solver divides the domain into *blocks* of uniform, pairwise disjoint cuboids. These blocks contain the data fields, for which the equations are solved. On the left, Figure 1 shows blocks distributed to parallel MPI ranks. A block is an object that builds a skeleton for data fields. The block specifies its geometry and global position in the domain, and stores the data fields with regular calculation grid, which can be seen in the middle and on the right of Figure 1. In a multiphysics application, the blocks contain several data fields, e.g., one scalar field for temperature, one vector field for velocity, and so on. *Data fields* are used according to the application. They can be arbitrary data types like scalar fields, vector fields, and structures in three-dimensional arrays in memory. For consistency, the data fields are extended by a *halo*, where one or more layers surround the data field. The halo holds a local copy of the boundary grid layers shared with the neighboring blocks. The size of the halo depends on the size of the stencil, which is illustrated on the right of Figure 1. The numerical scheme is described by a stencil which calculates a central grid point value by using several neighboring grid point values around the central point. With every time step, a kernel of this stencil sweeps over the whole data field.

The sweeping of the kernel, over the data field, is an *action*. For multiphysics applications, there may be several kernels which sweep over the different data fields. Other actions can be the writing of output data. Among others, NAStJA offers the output of VTI files (ParaView VTK Image Data) and mesh data. Also the execution of the boundary conditions and halo exchanges are actions. Figure 2 illustrates these actions in the program flow.

The phase-field application presented in the introduction only has a small interface region which needs to be calculated, whereas the other regions are uninteresting. In addition to the interface region from the example, other simulations with these properties, where only small parts are interesting, are conceivable. In this work, we use the term *interface* to describe the region where computations are needed.

All blocks are located on a regular grid, the *block-structured grid*. The *virtual domain* defines the maximum extension of the block-structured grid that is available, i.e., it determines the last available block in x-, y- and z-direction. This is required for domains of a fixed size to set the correct global boundary condition for each side of the virtual domain. Furthermore, the blocks are identified by a unique ID. This is the linearized index of the position in the block-structured grid, whose calculation requires the maximum number of blocks per dimension. Figure 3(a) shows a schematic of the phase-field interface within the virtual domain of the block-structured grid. The interface is covered by blocks depicted in Figure 3(b) that build the *computational domain*, and blocks outside the computational domain do not have to be allocated. The computational domain is distributed over the MPI ranks, see Figure 3(c). This is an advantage over ordinary simulation tools which use a costly remeshing or allocate the whole domain, and use a detection to determine the interface region, while the calculations of the other regions are omitted, as is done in [50]. NAStJA can cover complex geometries by blocks without wasting much memory for regions, where no calculations are needed. The kernel sweeps over the whole block, even if it has non-interface regions. This is often faster, because the blocks are small enough to fit into the cache. The block size can be chosen in such a way that the processor cache is used optimally, i.e., the whole block or the stencil input area—usually three layers—fits into the cache. Uniform calculation of one block, without the need of grid point index calculation, is a prerequisite to achieve a high node-level performance on HPC systems.

Usually, the interfaces are moving during the simulation, as shown in Figure 3(c). For correct results, the interface must always be covered by the blocks of the computational domain. If the computational domain does not adapt dynamically, the initial computational domain must be large enough to completely cover the interface at any time. For this purpose, the computational domain can be extended with spare blocks, which will contain parts of the interface, at a later simulation time. The spare blocks are initialized uniformly with a default value. This value depends on the side of the interface, on which the spare block is located. In this phase-field application, $\phi = 1$, inside the structure, and hence the block is initialized with 1. Outside the structure, $\phi = 0$, and so the block is initialized with 0. In advance, it is principally impossible to know the exact way in which the interface will evolve. Therefore, the extension of the computational domain must be estimated generously.

Hence, the approach of NAStJA is to dynamically create and delete blocks during the simulation. While the interface is moving, the computational domain is adapted to the interface, see Figure 3(d). The neighboring block is created before the structure leaves the inside of one block. On the other hand, it is deleted, if the interface region moves out of a block.

In the following, we describe how some prior knowledge about the model and the simulation domain geometry helps to preserve consistency. *Consistency* means that the result of a simulation with omitted blocks is the same as the result

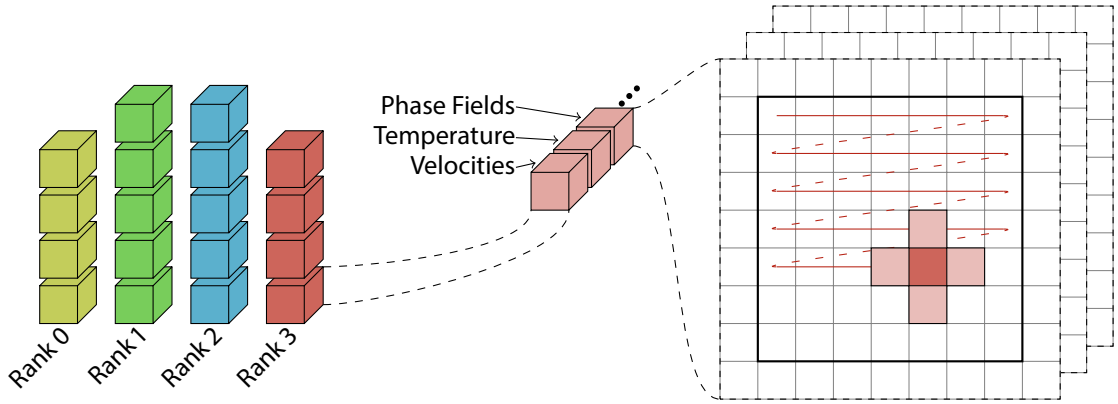



Fig. 1: Data structure for simulations in NAStJA. Blocks with the computing load are distributed to ranks, here rank 0 to rank 3. Each block can contain several data fields, e.g., velocities, phase fields. The data fields are an array of three-dimensional values, including the halo. A stencil, the red cross on the right, sweeps throughout the field. It loads input values from one or more fields, calculates the finite difference scheme, and stores the result on the center point.

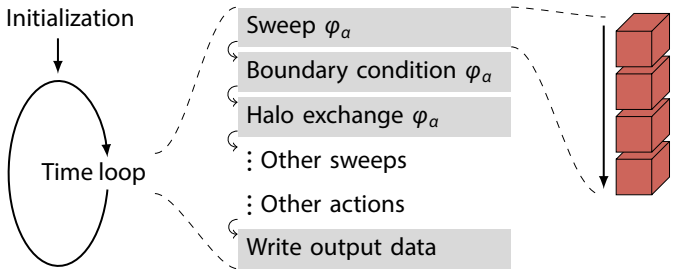



Fig. 2: After the initialization, the program runs a time loop. All actions are called one after another, whereby each action acts on all blocks. Then the next time step is processed.

from a full-domain reference simulation. Figure 4 illustrates different situations, and demonstrates the procedures that are necessary to enable the interface to remain consistent. At the boundary, the stencil always requires values from the halo. Sides with neighboring blocks receive these values from the neighbor via the halo exchange. In Figure 4, the stencil in position 2 accesses values from this halo. This case is consistent with the full-domain reference simulation. If there is no neighboring block, the global boundary condition sets these values. Inside the domain, a special boundary condition is introduced, the *inner boundary condition*, which is a Neumann boundary condition without flux. The stencils in positions 1 and 3 load values from this inner boundary layer, where the inner boundary condition is applied, whereas the stencil in position 1 is outside the interface region, and thus does not change the value during the calculation. As long as the interface does not touch the inner boundary, its values do not change, and therefore it has no influence on the result. The stencil in position 3 loads values from the interface, and changes the calculated value. The inner boundary layer would change, and thus distorts the result. For this reason, the last inner layer of the data field, the *test layer*, is used to check whether the values have changed. If the values have changed, a new block must be created on the right side of block 2. If the interface moves out of the block, the entire block, including the halo, is constant, in the case of the phase-field 0 or 1. Then, the block can be deleted, and the application of the inner boundary condition is replaced by the halo exchange.

In the stencil computation, the movement of the interface per time step is at most by the number of grid points that the stencil has in the backward direction. For the stencil of the phase-field method, this is a maximum of one grid point per time step. The fastest movement of the interface requires at least $n$ time steps to grow through one block with size $n$, from one side to the opposite side. This time can be used to rebuild and communicate the neighborhood of blocks before the consistency is violated. Concurrently, this is a restricting factor for the adaptive actions. In Section 3.4, we will later see some conditions why knowledge about the blocks in the neighborhood is indispensable.

### 3.3 The Different Communication Networks

Before the details of the dynamic block adaption are presented, we first introduce the different layers of communication that are used in NAStJA to achieve an autonomous adaptive and scalable code. Besides the halo exchange, two further communication networks are needed to accomplish the knowledge exchange required for the consistency of the dynamic block adaption. One of these networks provides the exchange within local groups of directly connected ranks. If necessary, the other network connects disjunct groups of ranks, e.g., when two interfaces grow towards each other. This network bypasses all-gather communications, and distributes the information efficiently throughout the whole domain, after several time steps.

*Halo Exchange*. After each calculation step, neighboring blocks, which have at least one common corner, exchange their halos. In three dimensions, a block has up to 26 neighbors, including six side neighbors, twelve edge neighbors, and eight corner neighbors. For full stencils, as is required for the mesh output, the halo exchange is done for all 26 neighbors. For smaller stencils, the corners, or even the edges, can be omitted. The present phase-field stencil only needs an exchange of the six sides. The communication effort for the halo exchange remains the same for each block, when the number of used ranks is scaled up.

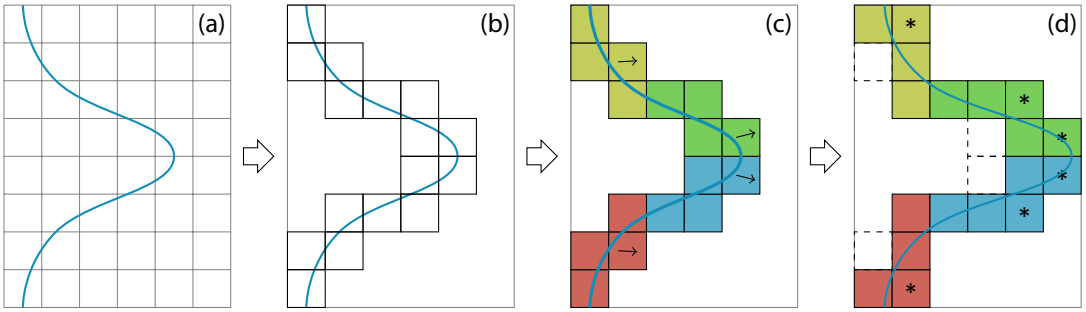



Fig. 3: Two-dimensional representation of a phase-field simulation. The blue line denotes the interface region where the phase changes from 0 to 1. (a) A block-structured grid is laid over the virtual domain. (b) From this grid, only the blocks holding a part of the interface region are allocated. These blocks form the computational domain. (c) All blocks of the computational domain are distributed to the available ranks, here marked by color. During the simulation, the interface changes and moves in space. (d) The blocks in the computational domain are adapted correspondingly. The stars denote newly created blocks, and the dashed blocks are deleted.

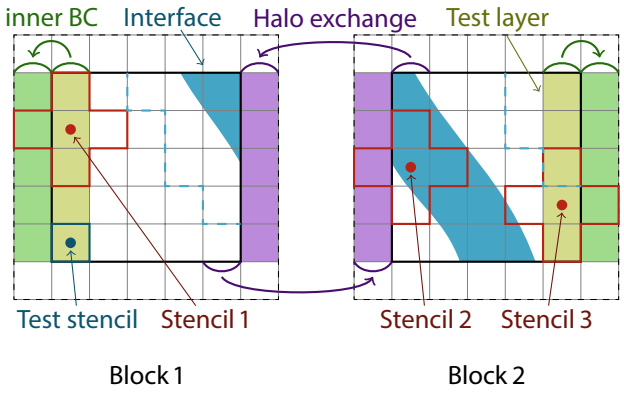



Fig. 4: Representation of two neighboring blocks. For the sake of illustration, we only consider the x-direction. The data fields (thick black rectangles) are extended by a halo (dashed rectangles). The green area denotes the inner boundary layer (BL). Here, the inner boundary condition (BC) is applied. The BL holds a copy of the layer next to it. The purple area denotes the halo exchange between block 1 and block 2. The interface is illustrated by a blue tube, and the interface region is restricted by the dashed blue line. The three stencils (red) demonstrate different situations. Stencil 1 loads from the inner BL, where all input values are the same, because it is not touching the interface. The finite difference scheme does not change the center value. Stencil 2 loads values from the interface, and the exchanged halo data from block 1, and it changes the center value. Stencil 3 has input values in the interface. It changes the center value on the test layer. The test layer (yellow) is a layer next to the inner BL, where a test stencil detects a change of values. The change of stencil 3 implies the creation of a new block to the right of block 2.

*Neighborhood Communication*. The knowledge about the existence of a block is essential for the dynamic block adaption. A trivial approach is to hold a list of all blocks on a master rank, or an up-to-date list of each rank. For both approaches, collective communications are required. With an increasing number of ranks, collective communications become a bottleneck, and limit the scalability. Therefore, interacting ranks build a local group of ranks with neighboring blocks, as described in detail later, and act autonomously within this group. This locality limits the number of connections per rank, and thus the communication overhead, and so results in a high scalability.

*Global Announcement Network*. The third communication network is a multi-hop broadcast network. The topology of the global announcement communication network is arbitrary, with the restriction that the diameter $k$ must be small enough to ensure global consistency after $k$ hops. In order to reduce the communication overhead, the network should have a small degree $d$, the number of MPI communication partners per rank. Each rank has a message manager, which sends messages to all its neighbors in this topology, and also receives messages from its neighbors. So, a message is spread to all ranks, after at most $k$ hops. This sending and receiving is coupled with the time steps, such that one hop is done in each time step. Each message is extended by a time to live (TTL) counter. This counter is initialized with $k$, and is decremented with each hop. When the TTL becomes zero, the message is globally known and can be deleted. For the topology, we use a multidimensional Manhattan Street Network (MSN) [51], [52], [53]. The MSN dimension is equal to the degree $d$, as $d$ increases with decreasing $k$.

### 3.4 Methods for Dynamic Block Adaption

For the dynamic block adaption, several actions are performed between the calculation action and the halo exchange action. The first action detects whether the interface is entering or leaving a block, as described in Figure 4. From this, a message is built, and is shared. The ranks decide autonomously where the new block is created. After the block creation and deletion, the halo exchange has to be reconfigured for new and deleted blocks. A global announcement of new blocks is initialized. Finally, an optional load balancing action can be performed.

For an elaborated description, we firstly define some terms. Let $\mathbb{V} = [0, X-1] \times [0, Y-1] \times [0, Z-1]$ be the set of all block representatives. This is called the *virtual domain*, with $X, Y$, and $Z$ as the number of blocks per dimension. The coordinates represent a position in the block-structured grid. Obviously, $\mathbb{V} \subset \mathbb{Z}^3$. For two elements, $a, b \in \mathbb{V}$ is $a - b \in \mathbb{Z}^3$.

Let $\|\cdot\|_\infty$ denote the maximum pseudometric on $\mathbb{Z}^3$. We define the *local neighborhood* of a block, which we identify with its representant $b$ as $L(b) := \left\{x \in \mathbb{V} \subset \mathbb{Z}^3 \,\middle|\, \|b - x\|_\infty \leq 2\right\}$. Note that depending on the global boundary conditions, also periodic blocks are located in $L(b)$. Therefore, $\mathbb{V}$ becomes a multidimensional torus, which is omitted for clarity. $L(b)$ contains all directly neighboring blocks, and the neighbors of these neighbors, in total 125 blocks.

$\mathbb{D} \subset \mathbb{V}$ denotes the set of all blocks in the *computational domain*. Let $\mathbb{H}$ be the set of all ranks, which is denoted by the MPI rank. Then we have a disjunct union $\mathbb{D} = \dot{\bigcup}_{h\in\mathbb{H}} \mathbb{B}_h$, where $\mathbb{B}_h$ is the set of the blocks hosted by rank $h$.

Let $H(b)$ be the *host process* for a given block $b$. Since $\mathbb{D}$ is not distributed to all ranks, each rank $h$ only knows a subset $\mathbb{K}_h \subset \mathbb{D}$, the *known blocks*. In a consistent state, the rank knows all blocks in the local neighborhood that are in the computational domain, i.e., $L(b) \cap \mathbb{K}_h = L(b) \cap \mathbb{D}$ for all $b$ on all $h$.

For each $h$, we define the *local group* $\mathbb{G}_h := \left\{H(n) \,\middle|\, n \in \bigcup_{b\in\mathbb{B}_h}(L(b) \cap \mathbb{K}_h)\right\}$, i.e., the set of all ranks that host a block in the local neighborhood $L(b)$ of any block $b \in \mathbb{B}_h$ of $h$. Note that it is not mandatory that all blocks of one rank, or the blocks inside the local group, are topologically contiguous.

For each *partner rank*, $p \in \mathbb{G}_h$, a *communicator* connects $p$ with $h$. This is done by a *communication manager* which manages the meta communications for the handling of the adaptive blocks. The communication manager keeps the data for the connection, including partner ranks, connection status, blocks on each partner rank, and the known blocks of each partner rank. They are stored in associative containers, i.e., maps from the C++ Standard Template Library. An initial state of these data is generated at the beginning of the simulation.

*Block creation*. If one rank detects that the interface reaches the boundary of a block (Figure 4), a message is generated for the blocks that need to be created. Depending on the stencil, each detection can produce one (on a side) or up to seven (on a corner) new blocks. An arbitrary function determines the host rank for the newly created block. If two blocks, $a$ and $c$, on different ranks, $A$ and $C$, simultaneously detect the same new block $b^*$, then both ranks must calculate the same host rank for $b^*$. This is possible when both ranks know each other, and use only consistent information for the determination of the host rank for $b^*$. For the local neighborhoods, it is $L(a) \supset N(b^*) \subset L(c)$, where $N(b^*)$ is the set of all neighbors of $b^*$, as demonstrated in Figure 5.

The function implemented in NAStJA to determine the host rank can be described as follows: Let $b^*$ be the new block. Let $\mathcal{N} := \left\{H(n) \,\middle|\, n \in N_w(b^*)\right\}$ be the set of all host ranks of all neighbors of $b^*$. Here, $w$ indicates the different number of neighbors depending on the stencil. Then the host rank of the new block is given by the rank $r = \min\left\{\arg\min_{n\in\mathcal{N}} \mathrm{load}(n)\right\}$, i.e., the partner rank with the least load. If two or more ranks have the same least load, the host rank with the lowest rank is chosen. Here, the function *load* gives the number of blocks hosted by this rank.

The message about new blocks is sent to each partner rank, where the message is processed. On the new host rank, a block is created and initialized with the correct values. In case of the example application with the simple phase-field function of one phase, these values are 1 when the new block is detected on the inner side of the interface / inside the structure, e.g., inside the droplet, and 0 when it is detected on the outer side. The receiver rank also knows the other ranks that are connected to the sender. Hence, the receiver knows which partners have received the message about the new blocks from the sender. If the receiver rank is connected to partners that have not received the message from the original sender, the new block is added to the list to announce the block to all these partners. In the next time step, the list for the announcing blocks is appended to the message which is sent to its partners by this rank. For each partner, there is a separate list for announcing blocks.

For these ranks, the halo exchange is retained. All other partners configure the halo exchange. Figure 6 demonstrates the creation and the announcement. In Figure 7, the retained halo exchange is illustrated. After the ranks receive the message with the announcement, they know the blocks hosted by the other ranks, and the retained halo exchange is configured and performed. Since the interface needs some time to go through a block, the missing halo exchange has no influence on the simulation result.

Before we explain the deletion of blocks, we describe the communication and connection between the ranks.

*Global announcement*. In the two cases above, we have

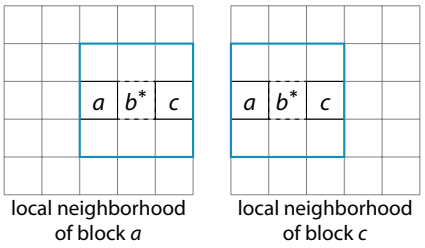



Fig. 5: The local neighborhoods $L(a), L(c)$ of block $a$ and $c$, respectively. The block $b^*$ is newly created. The direct neighbors $N(b^*)$ (blue rectangle) of $b^*$ are included in both neighborhoods. A 2D representation is used to simplify the illustration.

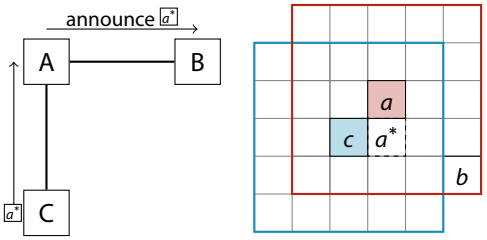



Fig. 6: Left: Connection between the ranks $A, B$ and $C$. $C$ detects a new block $a^*$, which will be created on $A$. $C$ sends this message to $A$. Since $C$ and $B$ are not connected, $B$ does not receive this message, so that $A$ has to announce $a^*$ to $B$ in a subsequent step to provide consistency. Right: Example of a representation of blocks in the virtual domain. Block $a$ on $A$ and block $c$ on $C$ are shown with their local neighborhoods depicted in the red and blue rectangle, respectively. Block $b$ is included in the local neighborhood of $a$, but is excluded from the local neighborhood of $c$, while $B$ and $C$ are not connected.

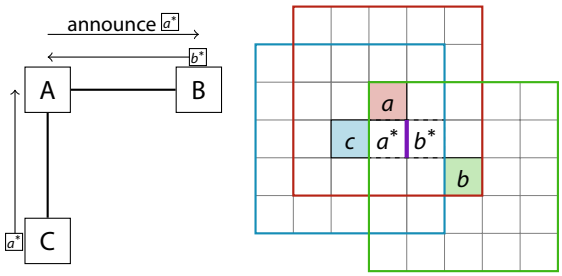



Fig. 7: The same situation as in Figure 6. The supplementary block $b$, with its neighborhood marked in green, is hosted by $B$, which detects a new block $b^*$, and will host the new block. This message is received by $A$, but not by $C$. The thick purple line shows the retained halo exchange. $A$ knows that $B$ has not received the message $a^*$ from $C$. The subsequent step of announcing $a*$ to $B$ eliminates the restraint, and the halo exchange is configured.

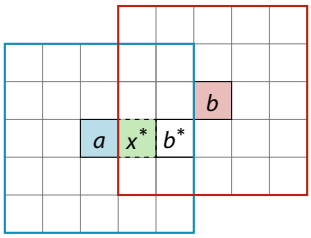



Fig. 8: Initially, block $a$ on rank $A$ and $b$ on $B$ have no common blocks in their local neighborhoods, i.e., $A$ and $B$ are not connected. When $B$ creates $b^*$, $A$ does not recognize this. Block $x^*$ is a neighbor of blocks on $A$ and $B$. Before the interface moves through $b^*$, and $B$ can create $x^*$, the global announcement has to announce $b^*$ to $A$, such that $A$ knows $b^*$, which is in the local neighborhood of $A$. From then on, $A$ and $B$ will determine the same host rank for $x^*$.

illustrated that either all existing blocks located in the local neighborhood are already known, or that they are made known through blocks on indirectly connected ranks, and thus the consistency is kept. Apart from that, there are also cases where a block $b^*$ is created, which is in a local neighborhood of $a$. However, the two host ranks $A$ and $B$ have no connection, which is shown in Figure 8. Then, a new block $x^*$ can be created from $A$ or $B$, since they do not know each other. This must be absolutely prevented. This means the ranks $A$ and $B$, and thus the blocks $a$ and $b$, need to know each other in the local neighborhood before the block $x^*$ will be created. As described at the end of Sec. 3.2, this knowledge transfer must be finished within the next $n$ time steps, where $n$ is the side length of the block. For this purpose, the global announcement network is used by sending messages with block ID and rank. The diameter of the network topology is limited in such a way that the knowledge transfer can be finished in sufficient time. Therefore, the connection setup and detection must be considered. In the worst case, the connection setup requires 5 time steps. The detection needs more than one time step to ensure that the connection is already established at the beginning of the time step. This limits the diameter.

*Connection setup process*. If a rank $h$ receives a message

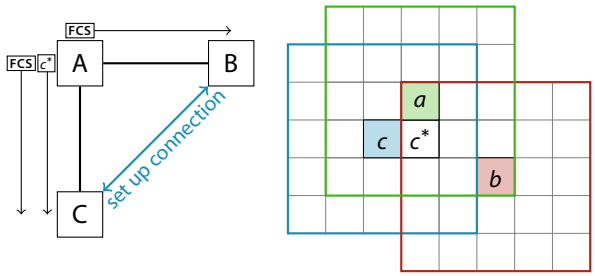



Fig. 9: $A$ detects a new block $c^*$, which will be created on $C$. $A$ knows $B$, and knows that $c^*$ is located inside the local neighborhood of $b$, so $B$ and $C$ have to communicate. $A$ additionally sends an FCS message to $B$ and $C$, which then establish a connection.

announcing an unknown block $u \notin B_h$ that is in the local neighborhood of any block $u \in \bigcup_{b \in B_h} L(b)$, then a connection between $h$ and $r = H(u)$, the host rank of $u$, must be created. A handshake protocol is used to establish the connection within three to five time steps, which is similar to the TCP (Transmission Control Protocol) three-way handshake with the SYN, SYN-ACK, ACK packages [54]. The communication between the communication managers is based on one message to/from each partner per time step. The communication manager also probes messages from unknown ranks. If there is a first handshake message from an unknown rank, the rank is added to the list of communicators, such that the handshake messages that follow are sent and received once per time step. When the handshake finishes, both partners are synchronized, and establish the connection in the same time step. The communicator is not visible outside the communication manager, until the handshake is completed. This is important because the halo exchange is only configured and executed from both sides simultaneously.

*Fast connection setup*. (FCS) In many cases, the rank detecting a new block knows about new connections. Rank $A$, for example, is connected with ranks $B$ and $C$, while $B$ and $C$ are not connected. One block on $A$ reaches the border, and a new block must be created on $C$. $A$ knows that the new block has a neighbor on $B$. Then it sends an additional message to both, $B$ and $C$, so that $B$ and $C$ can establish a connection, as depicted in Figure 9.

*Connection establishment*. After a handshake reaches the state 'established', or a fast connection setup message is received, an additional communication message is exchanged between the newly connected partners. This message contains all knowledge of the communicator that the partner has to know, particularly the list of the blocks and the connections. If this knowledge is transferred, the connection state is changed to 'connected', and all halo exchanges between blocks on these partner ranks can also be configured.

*Block deletion*. When a block without interface is detected, it can be removed, because further calculation will not change any value. Calculating a block without an interface takes computing time, but does not influence the result. Therefore, blocks are only allowed to be deleted if their existence is known to the whole domain after a global announcement process. This ensures that the block is known to all local neighborhoods to which it belongs. All partners

in the local neighborhood receive the deletion message and remove the block from their lists. The partners supersede the halo exchange with the application of the inner boundary condition. This is a much simpler process than the creation of blocks. To achieve this, each block is assigned a decreasing counter of minimum time steps to live, depending on the diameter of the global announcement network. The deletion is deferred as long as this counter is positive.

### 3.5 Load Balancing

Even if new blocks are created on the rank with the least number of blocks, it happens that the ranks have a very different number of blocks, which results in different calculation times per time step. Some blocks can be moved from one rank to another rank, based on a diffusion algorithm. This load balancing action can be performed independently on the dynamic block adaption action. The frequency of the execution of the load balancing action, i.e., each time step or all n time steps, can be chosen depending on the load per block and the change rate of blocks per rank. The load balancing is an autonomous process which uses the same neighborhood communication as the adapting action. To avoid an overestimation after the load balancing action, in which one target rank has more blocks than the source rank, each rank calculates and communicates an *offer* of the number of accepted blocks.

The average number of blocks is calculated for each local group. The rank then knows the number of spare blocks it can offer to other ranks. It offers the equally divided spare blocks to all ranks in its group, which have the maximum number of blocks. This offer is made in the time step before the load balancing step, and corrected by the number of created and deleted blocks in the dynamic block adaption. Then, every rank selects excessive blocks over the average, in its local group, and assigns them to the ranks in its group, depending on their offers. The assignment is chosen to minimize the halo communication to remote hosts, i.e., to keep as many MPI exchanges as possible on the same rank. The data of the blocks are packed and moved to their targets. As before, the blocks that are to be deleted are only allowed to move after they have been announced throughout the whole domain. In this case, all blocks in the neighborhood are known, and for each moving block, the new communication partners are known as well, such that they can perform a fast communication setup.

In the case where two neighboring blocks are moved to disconnected hosts, the new host ranks have to perform another fast communication setup.

## 4 Performance and Scalability

In this section, we show that the chosen algorithms, based on a performance model, have a good scalability. Subsequently, we present measurement results of the individual modules in the NAStJA framework.

We consider the times of the individual actions, $t_{\text{sweep}}$, for the computation stencil sweeping over the data field, $t_{\text{halo}}$, for the halo exchange, and $t_{\text{dynamic}}$, by gathering all individual times of the dynamic adaptive block actions, which include

$$t_{\text{dynamic}} := t_{\text{detection}} + t_{\text{adaption}} + t_{\text{balance}} + t_{\text{global}}, \tag{2}$$

where $t_{\text{detection}}$ is the time of the detection action. The time of the dynamic block adaption $t_{\text{adaption}}$ and load balance $t_{\text{balance}}$ actions depends on the neighborhood communication. The global announcement network communication is measured by $t_{\text{global}}$. For the performance analysis, all disk I/O actions are neglected.

Within the NAStJA framework, three different parallelization variants are conceivable, with the times

$$t_{\text{serial}} = t_{\text{sweep}}, \tag{3a}$$

$$t_{\text{parallel}} = t_{\text{sweep}} + t_{\text{halo}}, \tag{3b}$$

$$t_{\text{autonomous}} = t_{\text{sweep}} + t_{\text{halo}} + t_{\text{dynamic}}. \tag{3c}$$

The serial variant ($t_{\text{serial}}$) only consists of the calculation sweep, while $t_{\text{sweep}}$ scales as $\mathcal{O}(n^3)$, with the size of the block $n$.

In the parallel variant ($t_{\text{parallel}}$), the time of the halo exchange ($t_{\text{halo}}$) is added. In case of weak scaling, i.e., the workload for each rank is the same, $t_{\text{sweep}}$ stays constant, and $t_{\text{halo}}$ is an additional constant effort depending on $n$ and the stencil. A constant number of exchanges are performed. Up to 26 connections are possible with one or more halo layer per block. The size of the largest message in the halo exchange is proportional to $\mathcal{O}(n^2)$. In a perfectly communicating network, $t_{\text{halo}}$ is independent of the number of ranks.

For the autonomous variant ($t_{\text{autonomous}}$), the dynamic actions $t_{\text{dynamic}}$ are added. The time for the detection of reaching the border $t_{\text{detection}}$ is a pure local function, and hence is of the same order as $t_{\text{sweep}}$. The global announcement action $t_{\text{global}}$ has a low number of connections per rank. For $N = 185\,088$ ranks (the number of cores of the Hazel Hen, Germany's highest ranked HPC system in the TOP500 list [55]), and a block size of only 32 grid points per dimension, i.e., $32^3$ grid points in total, a six-dimensional Manhattan Street Network is sufficient, and hence only six connections are required. The diameter $d$ of the Manhattan Street Network [51] is given by $d \leq \sum_i \lceil N_i/2 \rceil + 2$, where $N_i$ is the number of ranks in each dimension. The number two is added for irregular networks, and for networks where each $N_i \mod 4 = 0$. As an estimate, this gives $d \approx (n \sqrt[n]{N})/2$. Using the numbers above, $N_i = \{8, 8, 8, 8, 7, 7\}$, and thus $d = 25$.

The dynamic block adaption $t_{\text{adaption}}$ and the optional load balancing $t_{\text{balance}}$ times are of the same order. In these steps, every rank has to communicate with all ranks that host one block in the local neighborhood of any hosted block. With an increasing number of ranks, the number of communications increases depending on the distribution of blocks. Since the number of neighbors is limited to 125 neighbors per block, this results in an upper limit of 125 connections per hosted block. With the assumption of an ideal communication network, where each connection is performed in a constant time, we get

$$t_{\text{autonomous}}(p) \leq b \cdot \hat{t}_{\text{parallel}} + \max\{p, 125 \cdot b\} \cdot \hat{t}_{\text{dynamic}}, \tag{4}$$

where $p$ is the number of ranks, $b$ represents the maximum number of blocks per rank, and $\hat{t}_{\text{parallel}}$ is the sum of all times of the constant scaling actions for one block, including the calculation and the halo exchange, as described above. $\hat{t}_{\text{dynamic}}$ is the time for one communication, and scales linearly with $b$. For $p > 125 \cdot b$, the timing becomes independent

from the number of ranks, which results in a constant scaling behavior.

Large block sizes have a significant influence on the constant part, and thus improve the scaling. The only communication time that is influenced by the block size is $t_{\text{halo}}$, which increases as the power of two, when the block size increases as the power of three. More expensive calculations, i.e., longer $t_{\text{sweep}}$, improve the scaling additionally by increasing the constant part. However, the cluster interconnects, and implementations of the message passing interface (MPI) deviate from a perfect communication network. In the following, we will investigate to what extent our theoretical estimation of the parallel scaling holds for NAStJA.

### 4.1 Measurements

To perform the scaling test, we use the systems ForHLR II, located at Karlsruhe Institute of Technology (fh2), and JURECA, located at Forschungszentrum Jülich (jureca).

ForHLR II has 1152 20-way Intel Xeon compute nodes. Each of these nodes contains two deca-core Intel Xeon processors E5-2660 v3, with Haswell architecture, which run at a basic clock rate of 2.6 GHz, and have $10 \times 256$ KB of level 2 cache, and 25 MB of shared level 3 cache. Each node has 64 GB of main memory, and an FDR adapter to connect to the InfiniBand 4X EDR interconnect. In total, 512 nodes can be used, which are connected by a quasi fat tree topology, with a bandwidth ratio of 10:11 between the switches and leaf switches. The implementation of Open MPI is used.

JURECA consists of 1872 24-way Intel Xeon compute nodes. Each of these nodes contains two dodeca-core Intel Xeon E5-2680 v3 Haswell CPUs which run at a base clock rate of 2.5 GHz and have $12 \times 256$ KB of level 2 cache, and 30 MB of shared level 3 cache. Each node has at least 128 GB of main memory, and a Mellanox EDR InfiniBand adapter with a non-blocking fat tree topology. The test runs with Intel MPI.

All scaling tests were performed in the setting of a weak scaling. As the basis value ($t_1$), we use the runtime on 20 cores of one full node on fh2, and 24 cores on jureca. The side length of a cubic block is varied, as well as the number of blocks per rank. During testing, we omit the disk I/O routines. The parallel efficiency $\eta$, used in the following, is defined by

$$\eta = \frac{t_1}{t_p}, \tag{5}$$

where $t_p$ is the parallel runtime with $p$ nodes. We have explicitly decided not to start with one rank as the basis value, to avoid effects that occur for a small number of ranks. These effects are influenced by the shared L3 cache of the processors in both machines, so that the usable cache per rank is increased for fewer ranks. For the calculation and communication times, we present absolute timings to see how the massively parallel simulation will behave, compared to a serial simulation. The test size increases by a power of two, for up to 256 nodes, corresponding to 5 120 cores on fh2 and 6 144 cores on jureca. We were able to run several simulations on 512 nodes, and 10 240 cores on fh2. Nevertheless, fh2 is composed of two islands, so that simulations of this size use 2/3 of the larger island. The switches are selected from the batch system which cannot be changed in the productive environment. In the following results, gaps occur at 128 or 256 nodes, which are caused by the topology of the switches. In cooperation with the operators, this has been further investigated, but cannot be fully explored on a productive machine. Usually, we present the best of several runs. Strongly deviating runs were repeated to eliminate most of the effects of productive HPC environments.

To consider the different parts of NAStJA, five scenarios are designed to test the different communication components separately. The scenarios are designed based on the previous scenarios, and test an additional module. First, the halo exchange is tested with an artificial workload, and then with a phase-field calculation, with a six-side exchange and a 26-side full exchange, followed by a synthetic test of the global announcement network. Finally, all modules are tested together to get the impact of the neighborhood communication, which cannot be tested separately.

*Scenario 1 (artificial workload).* An ideal calculation is simulated by a testing sleep function to test the communication. The chosen duration of 65 ms is based on the experience with highly optimized code for expensive calculations [1]. The halo is exchanged with the six direct neighbors, and is tested on fh2, using OpenMPI, and on jureca, using Intel MPI, with up to 32 blocks per rank. The block size is chosen as 100 grid points per dimension.

The timing is normalized to one block, and the efficiency is shown in Figure 10. In the case of a single block per rank, the measured times on fh2 and jureca are nearly the same (difference < 1 ms) on 1 up to 128 nodes. This gives rise to a parallel efficiency close to unity, for up to 128 nodes. On 256 nodes, the time measured on jureca increases significantly, and the efficiency drops to 50 %. Running with two blocks per rank on jureca changes the efficiency: it is nearly constant around 50 %, on 2 up to 128 nodes, followed by a small decrease by a further 20 %. In contrast, the code on fh2 performs and scales well for up to 256 nodes, with 1, 8, and 32 blocks per rank. This difference between fh2 and jureca may be attributed to different ratios of inter- and intra-node communication latencies for the two systems. On fh2, the parallelization overhead, which is defined as the ratio between the pure calculation time (65.1 ms) and the measured average time (67.2 ms), varies from about 3 %, on one node, to only 4 %, on 256 nodes, with a measured average total time of 67.9 ms. A small gap is seen for two nodes, where the communication changes from intra-node communication to inter-node communication. The efficiency for 256 nodes is 99.0 %, 99.3 %, 99.6 % for 1, 8, and 32 blocks, respectively. The halo exchange can be performed in a nearly theoretically optimal time. The efficiency increases by increasing the workload per rank, even if the number of communications is increased. For this measurement, we intentionally choose a short computing time per step. Because this time will be longer in real-life use cases, the efficiency is expected to be even better, according to Eqs. (3c), (5), so that this measurement provides a lower bound estimate of the efficiency for any workload under the conditions of this scenario.

*Scenario 2 (static simulation domain).* In this scenario, the previous workload is replaced by a real calculating function.

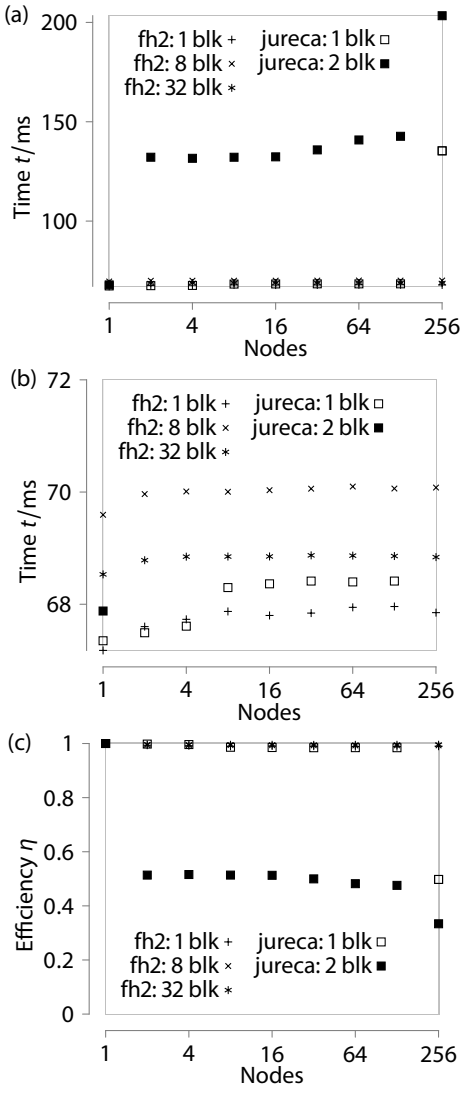



Fig. 10: Scaling at a constant workload per rank, with a halo exchange of six sides, normalized to one block, scenario 1. (a) Average time per time step, (b) details for times of 66 – 72 ms, and (c) efficiency.

The calculation is performed by the phase-field method from Sec. 3.1. A planar crystal front is set in an undercooled melt. As the crystal grows, the solid–liquid interface moves in one direction. For this scenario, a halo exchange with six neighbors is sufficient. Tests are performed on fh2 and jureca, with up to four blocks per rank. The block edge size is varied over 80, 100, and 120 grid points. Additionally, for size 100, several runs with 1, 2, and 4 blocks per rank are performed.

Figure 11 compares the runtimes and the efficiency. The scaling behavior on fh2, and on jureca, with a different number of blocks, is similar to that found in scenario 1: with one block per rank, the times on the two systems are very similar. Again, with more than one block per rank, the times on jureca increase significantly on two and more nodes (see Figure 11(b)). This may be attributed to the different ratios of intra- and inter-node communication latencies, for the two HPC systems. A small drop, and more than one block, is seen for 256 nodes. The same drop is present

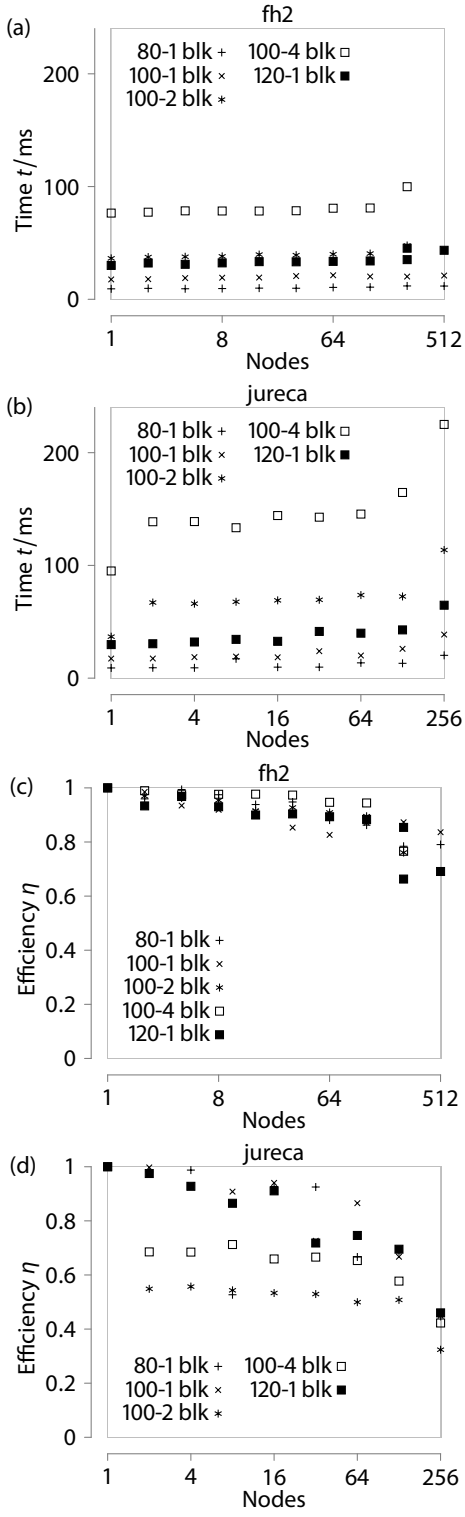



Fig. 11: Scaling of phase-field calculations, and a halo exchange of six neighbors, scenario 2. (a), (b) Average time per time step, and (c), (d) efficiency. The results are from runs on (a), (c) fh2, and (b), (d) jureca. Runs with a block edge size of 80, 100, and 120 are shown, as well as 1, 2, or 4 blocks per rank.

for the 120-edged block in one of the two runs. With a higher amount of nodes, the probability increases to obtain assigned switches that have more hops between them. With

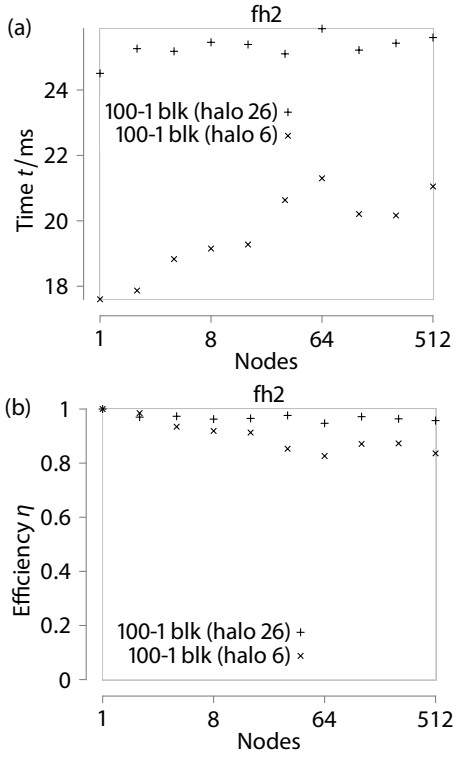



Fig. 12: Scaling of phase-field calculations, with a full halo exchange to 26 neighbors, compared to 6 neighbors, scenario 3. (a) Average time per time step, and (b) efficiency.

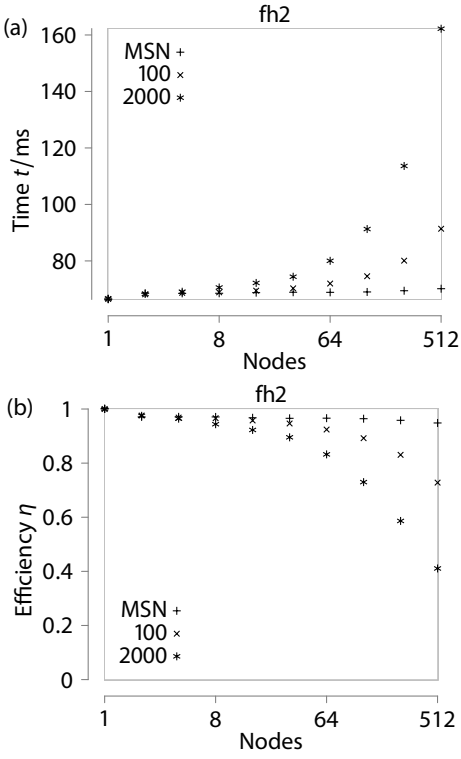



Fig. 13: Scaling of the global announcement network, scenario 4, for sizes of 100 and 2000 values, and the Manhattan Street Network (MSN) topology. (a) Average time of one time step, and (b) efficiency.

up to 128 nodes, the four-block simulation shows a better efficiency, 94 %, than the other one, which can be argued by the larger increase of computational time, compared to the communication times. The run with a block size of 120 needs 30.0 ms of average time per step, and 27.8 ms of pure calculation time, such that the overall overhead is 8 % for one node. This scales with an efficiency of 86 %, on 256 nodes, and 69 %, on 512 nodes. Runs with a block size of 100 show an efficiency of 87 %, on up to 256 nodes, and a smaller efficiency decrease on 512 nodes, with 84 %. As expected, a longer computation time results in a better efficiency for the one block calculations, with a block size of 80 and 100. In the following scenarios, communication is examined in more detail. Due to unexpected performance variations on jureca, the studies are only continued on fh2.

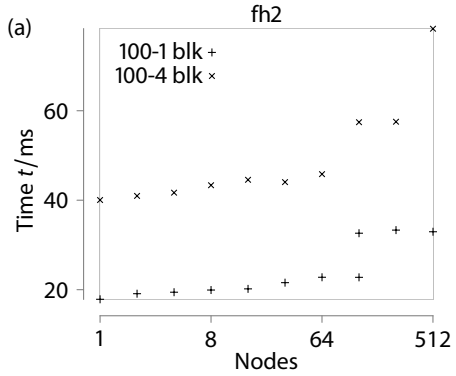
*Scenario 3 (full stencil).* In addition to the previous scenario 2, the calculation uses a full stencil. The halo exchange is extended to 26 sides to meet the requirements of full stencils. It is tested on fh2, with the calculation from scenario 2, and with one block per rank. The block edge size is 100 grid points.

The results compared with scenario 2 are shown in Figure 12. Compared to scenario 2, the measured pure calculation time 16.1 ms is not changed. The average time per step is increased up to 24.5 ms, and the overall overhead is increased to 52 %. The scalability is characterized by an efficiency of 96 %, for 512 nodes and 10 240 cores, respectively. With 26 sides, the ranks are entangled more strongly, and therefore a larger but equable overhead occurs. This larger overhead results in a better scalability than the six-side halo exchange in scenario 2, with an efficiency of 84 %.

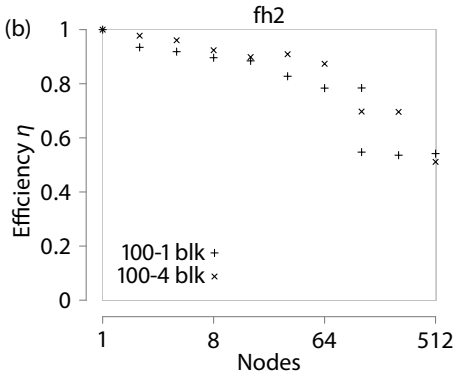
*Scenario 4 (global announcement).* A synthetic test of the global announcement network, with a halo exchange of 26 sides, and a testing sleep function of 65 ms, is performed on fh2, with one block per rank, and with a size of 100 grid points.

Figure 13 shows the Manhattan Street Network, which is compared to all-gather-communication, with 100 and 2000 values per rank. As expected, the runtime increases exponentially for the collective all-gather-communications, even for small messages with only 100 values per rank. This is a significant drawback of the collective communication. In contrast, the Manhattan Street Network approach scales very well, with an efficiency of 95 %, for 512 nodes.

*Scenario 5 (all communications).* The first four scenarios demonstrate a good communication behavior of NAStJA, for the halo exchange and the global announcement. In this scenario, the neighborhood communication is activated. In addition to the previous tests, the dynamic creation and deletion of blocks, as well as the load balancing, are activated. This scenario uses the full halo exchange of 26 sides, with a block edge size of 100 grid points. The same phase-field calculation of a growing crystal front, as in scenario 2, is used on fh2. A virtual domain, with one block in crystal growth direction, is used to measure the overhead and scaling of the communication for the dynamic block adaption, without the creation or deletion of new blocks. The behavior of deleting and creating blocks is investigated in a second test series, with a virtual domain of four blocks




(a) Average time per step




(b) Efficiency

Fig. 14: Scaling of the dynamic rank in scenario 5, for one and four initial blocks in growth direction. (a) Average time of one time step, and (b) efficiency.

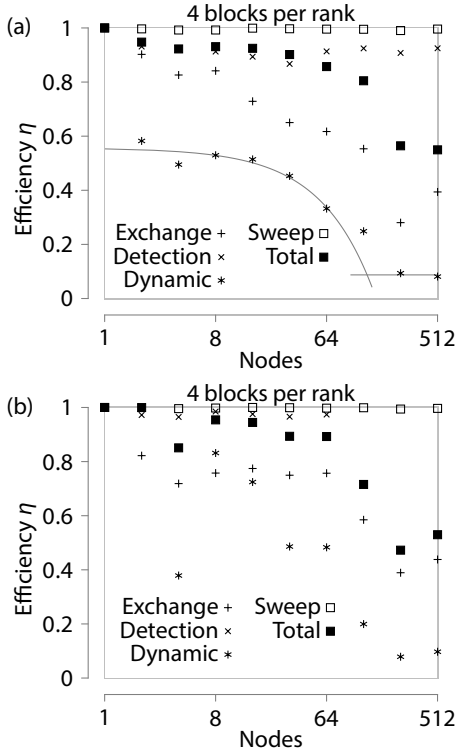



Fig. 15: Efficiency of the individual actions for (a) one and (b) four blocks per rank. For one block, the fit to the performance model is shown by the gray lines.

in crystal growth direction. The runs are initialized with all four blocks in crystal growth direction. The interface is only in one block, so that the three blocks without interface disappear as expected. During growth, the interface is moving, and a second block is created. For this test run, the average count of blocks per rank is 2.0495.

The results are presented in Figure 14. The simulation with four blocks requires around twice the time as the simulation with one block, which matches with the average count of blocks. On 128 nodes, the measured times strongly increase by about 10 ms, for both the one- and four-block cases. The overall overhead is 11 %. On one node with one block, a total average runtime of 17.85 ms is measured, compared to a calculation time of 16.14 ms. For one block, the efficiency decreases to 78 %, for 128 nodes (best run), and to 55 % (worst run). For the worst run, the efficiency stays at this level, 54 %, for up to 512 nodes. The four-block run shows an efficiency of 87 % before the gap on 64 nodes, and an efficiency of 70 % for 256 nodes. After the new gap, the efficiency on 512 nodes is still 51 %. The 128 node runs with one block show a large difference in runtime. For the dynamic module, a few more small messages are sent, compared to the scenarios that only use static distribution.

The different number of used switches, and therefore the different number of hops for the two 128 node runs, serves as an indicator. The worst run uses eleven switches, while the best one is only distributed over three switches.

A more detailed view of these scaling results is shown in Figure 15. The results for one and four blocks are split into contributions of the individual actions. The non-communicating parts—the sweep calculation and the detection of the reaching of the border—show a perfect efficiency. The measured efficiencies for the border detection of four blocks, on more than 64 nodes, are higher than 100 %, due to a slightly faster detection than on one node, and therefore are not shown in Figure 15(b). The dynamic parts in Figure 15(a) show a higher decrease, where especially a drop from one to two nodes is recognizable. Here, the communication changes from intra- to inter-node communications. For up to 64 nodes, the efficiency follows a linear regime, then it drops until 256 nodes. For 256 and 512 nodes, the efficiency is constant again. In Figure 15(a), the gray lines fit the time to a linear function for 2 to 64 nodes, and to a constant function for 256 to 512 nodes. This corresponds to the estimated behavior of the performance model (3c), which first has a linear time increase, and finally has a constant time. The measured value for 128 nodes is in the transition from linear to constant. With four blocks, the efficiency increases slightly from 256 and 512 nodes, as seen in Figure 15(b). The halo exchange time decreases, although the number of communications per block is constant. However, the scaling in scenario 1 to 3 shows the expected behavior, as presented in Figures 10 – 12. The individual communication parts cannot be completely separated from each other, so that the efficiency of the halo exchange is adapted to the efficiency of the dynamic part. The average times of the individual actions are listed in Table 1. It should be noted that only one block fits the cache, such that the

| Nodes/Blocks | 1/1 | 512/1 | 1/4 | 512/4 |
|---|---|---|---|---|
| Halo Exchange | 945.90 | 2 401.46 | 3 824.86 | 8 729.36 |
| Detection | 0.76 | 0.82 | 517.68 | 504.76 |
| DynBlock | 887.48 | 7 250.14 | 2 737.88 | 18 988.70 |
| Load Balance | 304.28 | 7 436.39 | 688.96 | 20 034.40 |
| Sweep | 16 173.50 | 16 240.40 | 33 717.20 | 33 831.50 |

TABLE 1: Average times of the individual actions for one and four blocks, on one and 512 nodes, in $\mu$s.

detection produces cache misses in the case of more than one block per rank, which increase the time.

## 5 Discussion

The measurements in the previous section confirm that the design of NAStJA is well chosen. The individual communication levels show a good to excellent scalability. In cases with small stencils, with a six-side exchange, the halo exchange scales very well, as well as with full stencils that additionally require communication to diagonal neighbors, and have 26 exchanges in total. The global exchange also scales excellently, because of the usage of the multi-hop process, instead of a collective communication which does not scale, as shown in Figure 13. The communication for the adaptive action first appears to influence the efficiency strongly. However, the used algorithm can completely dispense with collective communications, which scale worse than the local neighborhood communications. The results of the measurements show that the efficiency of many nodes (from 128 onwards) is approximately constant, which would not be the case when using collective communications. However, the saved part of the computational domain, and thus the saved work load, can be enormous. As an illustration, blocks with an edge size of 80 grid points are used to cover a quarter of a spherical segment with the radius 5 000 (center height 1 000). As a result, only 10 436 blocks are required, when considering an interface of 16 grid points. Compared to this, 297 675 blocks are required to cover the spherical segment with a cuboid of blocks. Here, the expansion of the domain is not yet taken into account in an advanced simulation. This means that the adaptive approach only requires 3.5 % of the usually required computational domain and computing effort. This puts the overhead of factor 2 of the communications into perspective. Theoretically, this is a speedup of over 14, for this very advantageous case. Less advantageous cases are also expected to benefit from this method. In summary, a higher computing load, such as more difficult calculations or larger blocks, leads to a better scalability, because the communication time stays constant or only increases quadratically, in the case of the halo-exchange, while the block size increases cubically.

## 6 Conclusion

We have shown that for the given example, the saving of computational effort is enormous, as the required effort just corresponds to 3.5 % of the effort required with traditional solutions. The methods of the NAStJA framework are working very well, and are highly recommended. The expensive communication results in an overhead of around 100 % for one block, and 70 % for four blocks per rank. This is an indicator that an increase of the workload increases the scalability.

Although the sample problem is very specific, the used properties of the phase-field method, i.e., calculations only performed in the interface region, and the limited growth rate, can also be found in other fields of application. First of all, the multiphase-field method should be mentioned, which is used for multigrain simulations or geological simulations. Cellular automata, such as the cellular Potts model, are also conceivable, which can describe biological cells for, e.g., tissue growth. The NAStJA framework supports many of these methods, such as a phase-field method, a phase-field crystal model, and the cellular Potts model. It is flexibly designed, such that it can be simply extended to a wide range of algorithms that act on finite difference schemes. In addition, other methods on regular grids, such as cellular automata, are also supported.

Furthermore, the usage of blocks in a block-structured grid has some additional, great advantages for the distribution concept, and is very flexible and readily extensible. The user can choose the block size such that the whole block or the layers required for the stencil calculations fit into the processor cache, e.g., three layers, as for the example stencil presented here. Load balancing is much simpler when the geometry of the halo exchange does not change, even in the case of off-loading the operations of whole blocks to accelerator devices, such as GPUs, Xeon Phi, and vector cards. The dynamic block adaption can be extended to different data fields, such that only the required fields are calculated. An adaption to the resolution is also conceivable, where the resolution of the calculation grid is based on a block level. Since the neighborhood communication is the most expensive one, and hence mostly limits the scalability factor, it is conceivable not to perform the adaptive action during every time step.

In particular, it can be assumed that the interface does not cross the boundary until the next adaptive action. If it comes into contact with the boundary, the interface is influenced by this. Small artifacts are formed in the interface. In later time steps, when the interface can move freely, and without boundary influences, the smoothing behavior of the phase-field interface compensates these artifacts. This would lead to an even better scalability.

Further possible studies include the influence of the block size on the overall efficiency. A smaller block size reduces the part of the areas that have to be calculated in the blocks. However, more blocks are needed, that are deleted or created more often, which results in a higher communication effort. Where is the sweet spot? In the current state of the NAStJA framework, users have to write their own sweeps to perform calculations on their data fields. In addition, they can provide actions to perform special tasks for their simulation besides the calculations, e.g., counting, communicating, or modifying the data fields. It is planned to provide a mechanism to assemble actions and sweeps to an application, via entries in the configuration file. It is also imaginable to provide API access to common utilities in a later version, so that the actions and sweeps only have to be built by the users themselves. There are also plans to

release NAStJA under an open source license soon. Until then, academic preview licenses are available.

## ACKNOWLEDGMENTS

This work was performed on the computational resource ForHLR II, funded by the Ministry of Science, Research and the Arts Baden-Württemberg and the DFG ("Deutsche Forschungsgemeinschaft"). We thank M. Soysal and R. Walter for the explanation of the topology and the fruitful discussions. The authors gratefully acknowledge the computing time granted by the JARA-HPC Vergabegremium on the supercomputer JURECA [56], at Forschungszentrum Jülich.

## REFERENCES

[1] M. Bauer, J. Hötzer, M. Jainta, P. Steinmetz, M. Berghoff, F. Schornbaum, C. Godenschwager, H. Köstler, B. Nestler, and U. Rüde, "Massively parallel phase-field simulations for ternary eutectic directional solidification," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 8.

[2] J. Hötzer, M. Jainta, P. Steinmetz, B. Nestler, A. Dennstedt, A. Genau, M. Bauer, H. Köstler, and U. Rüde, "Large scale phase-field simulations of directional ternary eutectic solidification," *Acta Materialia*, vol. 93, no. 0, pp. 194 – 204, 2015.

[3] J. Hötzer, P. Steinmetz, M. Jainta, S. Schulz, M. Kellner, B. Nestler, A. Genau, A. Dennstedt, M. Bauer, H. Köstler *et al.*, "Phase-field simulations of spiral growth during directional ternary eutectic solidification," *Acta Materialia*, vol. 106, pp. 249–259, 2016.

[4] P. Steinmetz, Y. C. Yabansu, J. Hötzer, M. Jainta, B. Nestler, and S. R. Kalidindi, "Analytics for microstructure datasets produced by phase-field simulations," *Acta Materialia*, vol. 103, pp. 192–203, 2016.

[5] T. Shimokawabe, T. Aoki, T. Takaki, A. Yamanaka, A. Nukada, T. Endo, N. Maruyama, and S. Matsuoka, "Peta-scale phase-field simulation for dendritic solidification on the TSUBAME 2.0 supercomputer," *2011 International Conference for, High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–11, 2011.

[6] T. Takaki, T. Shimokawabe, M. Ohno, A. Yamanaka, and T. Aoki, "Unexpected selection of growing dendrites by very-large-scale phase-field simulation," *Journal of Crystal Growth*, vol. 382, pp. 21–25, 2013.

[7] T. Takaki, M. Ohno, Y. Shibuta, S. Sakane, T. Shimokawabe, and T. Aoki, "Two-dimensional phase-field study of competitive grain growth during directional solidification of polycrystalline binary alloy," *Journal of Crystal Growth*, p. –, 2016.

[8] M. Ben Said, M. Selzer, B. Nestler, D. Braun, C. Greiner, and H. Garcke, "A phase-field approach for wetting phenomena of multiphase droplets on solid surfaces," *Langmuir*, vol. 30, no. 14, pp. 4033–4039, 2014.

[9] K. A. Brakke, "The surface evolver," *Experimental mathematics*, vol. 1, no. 2, pp. 141–165, 1992.

[10] W. C. Carter, "Surface evolver as a tool for materials science research," *Mathematics of Microstructure Evolution*, pp. 1–14, 1995.

[11] S. Brandon, N. Haimovich, E. Yeger, and A. Marmur, "Partial wetting of chemically patterned surfaces: The effect of drop size," *Journal of colloid and interface science*, vol. 263, no. 1, pp. 237–243, 2003.

[12] H. P. Jansen, O. Bliznyuk, E. S. Kooij, B. Poelsema, and H. J. Zandvliet, "Simulating anisotropic droplet shapes on chemically striped patterned surfaces," *Langmuir*, vol. 28, no. 1, pp. 499–505, 2011.

[13] K. Elder, N. Provatas, J. Berry, P. Stefanovic, and M. Grant, "Phase-field crystal modeling and classical density functional theory of freezing," *Physical Review B*, vol. 75, no. 6, p. 064107, 2007.

[14] M. Berghoff and B. Nestler, "Phase field crystal modeling of ternary solidification microstructures," *Computational Condensed Matter*, vol. 4, pp. 46–58, 2015.

[15] F. Graner and J. A. Glazier, "Simulation of biological cell sorting using a two-dimensional extended potts model," *Physical Review Letters*, vol. 69, no. 13, p. 2013, 1992.

[16] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, and O. Sander, "A generic grid interface for parallel and adaptive scientific computing. part i: abstract framework," *Computing*, vol. 82, no. 2, pp. 103–119, 2008.

[17] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger, and O. Sander, "A generic grid interface for parallel and adaptive scientific computing. part ii: Implementation and tests in dune," *Computing*, vol. 82, no. 2-3, pp. 121–138, 2008.

[18] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells, "The fenics project version 1.5," *Archive of Numerical Software*, vol. 3, no. 100, pp. 9–23, 2015.

[19] M. J. Welland, D. Karpeyev, D. T. O'Connor, and O. Heinonen, "Miscibility gap closure, interface morphology, and phase microstructure of 3d li x fepo4 nanoparticles from surface wetting and coherency strain," *ACS nano*, vol. 9, no. 10, pp. 9757–9771, 2015.

[20] J. E. Guyer, D. Wheeler, and J. A. Warren, "Fipy: partial differential equations with python," *Computing in Science & Engineering*, vol. 11, no. 3, 2009.

[21] D. Wheeler, J. A. Warren, and W. J. Boettinger, "Modeling the early stages of reactive wetting," *Physical Review E*, vol. 82, no. 5, p. 051601, 2010.

[22] M. R. Tonks, D. Gaston, P. C. Millett, D. Andrs, and P. Talbot, "An object-oriented finite element framework for multiphysics phase field simulations," *Computational Materials Science*, vol. 51, no. 1, pp. 20–29, 2012.

[23] P. C. Millett, M. R. Tonks, K. Chockalingam, Y. Zhang, and S. Biner, "Three dimensional calculations of the effective kapitza resistance of uo 2 grain boundaries containing intergranular bubbles," *Journal of Nuclear Materials*, vol. 439, no. 1, pp. 117–122, 2013.

[24] I. Steinbach, "Phase-field models in materials science," *Modelling and simulation in materials science and engineering*, vol. 17, no. 7, p. 073001, 2009.

[25] C. Godenschwager, F. Schornbaum, M. Bauer, H. Köstler, and U. Rüde, "A framework for hybrid parallel flow simulations with a trillion cells in complex geometries," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 35.

[26] K. Thornton, S. Rudraraju, and S. DeWitt, "PRISMS-PF," 2017. [Online]. Available: https://github.com/prisms-center/phaseField

[27] I. Steinbach, F. Pezzolla, B. Nestler, M. Seeßelberg, R. Prieler, G. J. Schmitz, and J. L. Rezende, "A phase field concept for multiphase systems," *Physica D: Nonlinear Phenomena*, vol. 94, no. 3, pp. 135–147, 1996.

[28] M. Mecozzi, J. Eiken, M. Santofimia, and J. Sietsma, "Phase field modelling of microstructural evolution during the quenching and partitioning treatment in low-alloy steels," *Computational Materials Science*, vol. 112, pp. 245–256, 2016.

[29] B. Nestler, H. Garcke, and B. Stinner, "Multicomponent alloy solidification: phase-field modeling and simulations," *Physical Review E*, vol. 71, no. 4, p. 041609, 2005.

[30] A. Vondrous, M. Selzer, J. Hötzer, and B. Nestler, "Parallel computing for phase-field models," *The International Journal of High Performance Computing Applications*, vol. 28, no. 1, pp. 61–72, 2014.

[31] COMSOL Inc., "COMSOL," 2017. [Online]. Available: https://comsol.de

[32] F. Schornbaum and U. Rüde, "Massively parallel algorithms for the lattice boltzmann method on nonuniform grids," *SIAM Journal on Scientific Computing*, vol. 38, no. 2, pp. C96–C126, 2016.

[33] N. Ofori-Opoku and N. Provatas, "A quantitative multi-phase field model of polycrystalline alloy solidification," *Acta Materialia*, vol. 58, no. 6, pp. 2155–2164, 2010.

[34] G. Amberg, "Semisharp phase field method for quantitative phase change simulations," *Physical review letters*, vol. 91, no. 26, p. 265505, 2003.

[35] T. Takaki, T. Fukuoka, and Y. Tomita, "Phase-field simulation during directional solidification of a binary alloy using adaptive finite element method," *Journal of crystal growth*, vol. 283, no. 1, pp. 263–278, 2005.

[36] M. Greenwood, K. Shampur, N. Ofori-Opoku, T. Pinomaa, L. Wang, S. Gurevich, and N. Provatas, "Quantitative 3d phase field modelling of solidification using next-generation adaptive mesh refinement," *Computational Materials Science*, vol. 142, pp. 153–171, 2018.

[37] C. Lan and Y. Chang, "Efficient adaptive phase field simulation of directional solidification of a binary alloy," *Journal of Crystal Growth*, vol. 250, no. 3, pp. 525–537, 2003.

[38] Y. Li and J. Kim, "Phase-field simulations of crystal growth with adaptive mesh refinement," *International Journal of Heat and Mass Transfer*, vol. 55, no. 25, pp. 7926–7932, 2012.

[39] R. Folch and M. Plapp, "Quantitative phase-field modeling of two-phase growth," *Physical Review E*, vol. 72, no. 1, p. 011602, 2005.

[40] X. Cai, H. Marschall, M. Wörner, and O. Deutschmann, "A phase field method with adaptive mesh refinement for numerical simulation of 3d wetting processes with openfoam®," in *2nd International Symposium on Multiscale Multiphase Process Engineering (MMPE), Hamburg, Germany*, 2014.

[41] X. Cai, H. Marschall, M. Wörner, and O. Deutschmann, "Numerical simulation of wetting phenomena with a phase-field method using openfoam®," *Chemical Engineering & Technology*, vol. 38, no. 11, pp. 1985–1992, 2015.

[42] N. Provatas, M. Greenwood, B. Athreya, N. Goldenfeld, and J. Dantzig, "Multiscale modeling of solidification: phase-field methods to adaptive mesh refinement," *International Journal of Modern Physics B*, vol. 19, no. 31, pp. 4525–4565, 2005.

[43] P. Maes, "Modeling adaptive autonomous agents," *Artificial life*, vol. 1, no. 1_2, pp. 135–162, 1993.

[44] M. Scheutz and P. Schermerhorn, "Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models," *Journal of Parallel and Distributed Computing*, vol. 66, no. 8, pp. 1037–1051, 2006.

[45] N. Fachada, V. V. Lopes, R. C. Martins, and A. C. Rosa, "Parallelization strategies for spatial agent-based models," *International Journal of Parallel Programming*, vol. 45, no. 3, pp. 449–481, 2017.

[46] A. Choudhury and B. Nestler, "Grand-potential formulation for multicomponent phase transformations combined with thin-interface asymptotics of the double-obstacle potential," *Physical Review E*, vol. 85, no. 2, p. 021602, 2012.

[47] M. Berghoff, M. Selzer, and B. Nestler, "Phase-field simulations at the atomic scale in comparison to molecular dynamics," *The Scientific World Journal*, vol. 2013, 2013.

[48] M. Berghoff, *Skalenübergreifende Modellierung und Optimierung vom atomistischen kristallinen Phasenfeldmodell bis zur mesoskopischen Phasenfeldmethode*. KIT Scientific Publishing, 2015, vol. 49.

[49] J. Hötzer, O. Tschukin, M. B. Said, M. Berghoff, M. Jainta, G. Barthelemy, N. Smorchkov, D. Schneider, M. Selzer, and B. Nestler, "Calibration of a multi-phase field model with quantitative angle measurement," *Journal of materials science*, vol. 51, no. 4, pp. 1788–1797, 2016.

[50] M. Berghoff, M. Selzer, A. Choudhury, and B. Nestler, "Efficient techniques for bridging from atomic to mesoscopic scale in phase-field simulations," *Journal of Computational Methods in Sciences and Engineering*, vol. 13, no. 5, 6, pp. 441–454, 2013.

[51] B. Khasnabish, "Topological properties of manhattan street networks," *Electronics Letters*, vol. 25, no. 20, pp. 1388–1389, 1989.

[52] T.-Y. Chung and D. P. Agrawal, "Design and analysis of multidimensional manhattan street networks," *IEEE transactions on communications*, vol. 41, no. 2, pp. 295–298, 1993.

[53] F. Comellas, C. Dalfó, and M. A. Fiol, "Multidimensional manhattan street networks," *SIAM Journal on Discrete Mathematics*, vol. 22, no. 4, pp. 1428–1447, 2008.

[54] J. Postel, "Transmission control protocol," Internet Requests for Comments, RFC Editor, STD 7, 1981, http://www.rfc-editor.org/rfc/rfc793.txt.

[55] TOP500.org, "Top500 List - November 2017," 2017. [Online]. Available: https://www.top500.org/list/2017/11/

[56] Jülich Supercomputing Centre, "JURECA: General-purpose supercomputer at Jülich Supercomputing Centre," *Journal of large-scale research facilities*, vol. 2, no. A62, 2016.

**Marco Berghoff** received his diploma in mathematics from the University of Paderborn, Germany, with a focus on microlocal analysis, numerics, and physics. He has been a member of the Karlsruhe Institute of Technology, at the Institute for Applied Materials, where he received his PhD in Computational Materials Science. He has years of experience in multiscale modeling and high performance optimization, with the atomistic phase-field crystal model and the mesoscopic phase-field method. As a postdoctoral researcher in the Simulation Laboratory "NanoMicro", he has introduced the framework NAStJA, and currently leads the developments. He is involved in several activities within this project, in particular in the development of large-scale simulations for biological or material science research topics.

**Ivan Kondov** has received his PhD degree in theoretical physics at the Chemnitz University of Technology, with a thesis dealing with efficient and scalable numerical schemes for solving quantum master equations. As a postdoc at the Technical University of Munich, he worked on the simulation of interfacial electron transfer processes. Since 2010, he has been the leader of the Simulation Laboratory "NanoMicro" at Steinbuch Centre for Computing (SCC), at Karlsruhe Institute of Technology, and since 2015, he has been deputy head of the department "Scientific Computing and Simulation". His current research includes multiscale modeling and simulations using high performance computing, workflows for model and data integration, hierarchical modeling of tightly coupled multiscale systems, employing the concepts of model-driven architecture and service-oriented architecture in computational materials science, and preparing HPC applications for the exascale.

**Johannes Hötzer** started work on the phase-field model during his master studies of computer science, and composed his master thesis on the optimization and parallelization of PACE3D. In his PhD, he focused on large-scale simulations of sintering processes, and on directional solidification of ternary eutectics. He has several years of experience in phase-field modeling, as well as high performance computing, and is the group leader of activities in "High Performance Materials Computing and Data Science". The focus of his work is on the solidification of ternary eutectics, and on sintering processes. For several semesters, he has given a lecture on "High Performance Computing", with an integrated computer lab for students of computer science and mechanical engineering.