Modelling and Simulation in Materials Science and Engineering

PAPER

Parallelization of an efficient 2D-Lagrangian model for massive multi-domain simulations

To cite this article: Sebastian Florez *et al* 2021 *Modelling Simul. Mater. Sci. Eng.* **29** 065005

IOP Publishing Modelling and Simulation in Materials Science and Engineering

Modelling Simul. Mater. Sci. Eng. **29** (2021) 065005 (35pp) https://doi.org/10.1088/1361-651X/ac0ae7

Parallelization of an efficient 2D-Lagrangian model for massive multi-domain simulations

Sebastian Florez*, Julien Fausty, Karen Alvarado, Brayan Murgas and Marc Bernacki

Mines-ParisTech, PSL-Research University, CEMEF-Centre de mise en forme des matériaux, CNRS UMR 7635, CS 10207 rue Claude Daunesse, 06904 Sophia Antipolis Cedex, France

E-mail: sebastian.florez@mines-paristech.fr

Received 22 September 2020, revised 29 March 2021
Accepted for publication 11 May 2021
Published 1 July 2021

Abstract

The parallelization of algorithms is an essential step towards the optimization of large-scale computations. The modeling of evolving multi-domain problems is not an exception to this rule, specifically when it is applied to the context of microstructural evolutions. A new method for the simulation of evolving microstructures has been introduced in a previous work, consisting on a modified front-tracking approach where the main originality is that not only interfaces between domains are discretized but also their bulks. This new model has obtained promising results in terms of accuracy and numerical performance, however, it has been implemented in a sequential environment and is not readily usable in modern HPC units for parallel computations. This article proposes a parallel implementation for the new model using a distributed-memory approach developed with the standard protocol 'message passing interface'. The new parallel methodology has been tested in an HCP station with up to 140 cores for problems involving motion by curvature flow in polycrystals, i.e. by considering pure grain growth. Good results were obtained in terms of CPU-time and speed-up for large polycrystals (with up to 560 000 initial grains), showing that this model can lead to fast and/or large computations of microstructural evolutions in a full-field context.

Keywords: massive multidomain simulations, parallel implementation, front-tracking, moving-interfaces, grain-growth, body-fitted, unstructured meshes

(Some figures may appear in colour only in the online journal)

*Author to whom any correspondence should be addressed.

1. Introduction

The simulation of the dynamics of massive multidomain problems have been addressed by several numerical approaches [1–10], especially in the context of modeling the microstructural evolutions of metallic materials: grain growth (GG) [11–13], recrystallization (ReX) [14–17] or Zener pinning (ZP) [18–23]. Usually, these approaches aim to enhance their accuracy and numerical performance employing computational optimizations, especially for applications where the physical object, e.g. a representative volume element volume (RVE), is subdivided into thousands of domains (e.g. grains in a polycrystal). In the following, the term *domain* will reference an individual grain in a microstructure. However, the works presented here can be extended to other multidomain problems such as multiphase flows [24–26] or phase transformation [27, 28].

Many numerical approaches exist that are capable of handling thousands of domains:

The probabilistic **Monte Carlo** (MC) method (or Pott's model) [1, 2, 29–33] and the **cellular automata** (CA) method [3, 34–43] both relying in a computational domain discretized in voxels, making them very efficient in a parallel context [41, 43] although a greater effort must be done for applications where the deformation of the domain is expected [43].

Furthermore, the **level-set** (LS) approach [6, 7] coupled with a **finite element** (FE) method (FE-LS) [10, 11, 16, 44, 45] or with a **fast-Fourier transformation** (FFT) method (FFT-LS) [46–48] are also examples of robust and highly efficient methods in this context. Another method highly related to the LS method is the **phase-field** (PF) method [8, 9, 49–58] which uses, in general, an FE method for their resolution. The degree of difficulty during the parallelization of LS and PF methods depends on the used discretization approach. Methods using regular grids can be parallelized relatively easier than methods using unstructured meshes. However, the choice of the discretization approach cannot be determined by the difficulty to accomplish parallel computations. For instance, the FFT-LS method has proven to be able to simulate a large number of domains ($\approx 6 \times 10^5$ were used in [48] in a 2D GG context) but is currently limited to the use of regular grids, an aspect that restricts its domain of application to static or small deformations problems where remeshing is not necessary. This can be a major drawback for modeling dynamic recrystallization (DRX) in the context of large deformations (as involved in usual industrial thermomechanical treatments).

Additionally to the MC, CA, LS and PF, methods based on the Lagrangian displacement of interfaces can be used as in **vertex** [4, 59–63] or **front-tracking** (FT) approaches [5, 19–21, 64, 65], which are more complex to implement in parallel but are, in general, much faster than the aforementioned methods. This is why, the vertex and FT methods have been taken as inspiration for developing a new FT method: topological remeshing in Lagrangian framework for large interface motion (to real motion hereafter TRM) introduced in a previous publication [66]. The TRM method proposes the use of unstructured meshes to discretize grain interiors, contrary to conventional vertex and FT methods that only discretize grain boundaries (interfaces between domains). This choice was made for multiple reasons: (i) to allow the use of intragranular data such as stored energy (used in ReX), (ii) to model large deformation of the domain and DRX mechanism, and (iii) to make the TRM method more parallel-friendly than conventional FT models. The TRM model is thus aimed at being compatible with a wide range of phenomena (e.g. GG and DRX) as the LS-FE methods, to allow large polycrystals modeling as for the LS-FFT approach, and to do so with the best numerical performance for which a parallel implementation is needed. The latter aspect is precisely the main topic of this work.

The TRM model's performance was tested and compared against a classical front capturing LS-FE method [10–12] in an isotropic 2D-GG context. Multiple test cases showed an increase

in accuracy while the CPU-time was reduced by a factor of 14 (for sequential simulations) [45].

Regarding parallel systems, two categories of design can be proposed to exploit the capabilities of modern computational units. These two categories are differentiated by the management of the active memory of the running processes: *shared memory*, where all processes share and interact with the same location in memory, and *distributed memory*, where each process has its own independent memory location. Whether to use one or the other strongly relies on the hardware for which the model is aimed. Typically, applications implemented using *only* a shared memory framework cannot be used over a supercomputer cluster, as the memory in these systems is not connected to a single board, but it is distributed between several independent CPUs connected to the same network. In other words, shared memory can only be used within a single CPU, while distributed memory is intended to be used both within a single CPU and over a network of interconnected CPUs. A third option can be engineered through the use of a hybrid system, in which a combination of distributed and shared parallelism is made, offering, in general, the best computational performance. Of course, in a hybrid case, the parallel strategy's design can become complex, and it will be avoided in the present article. The parallel implementation of the TRM model presented in this work will only use a distributed memory approach.

Each process in a distributed memory system needs a way to communicate data to other processes. This communication can be established by the use of the standard protocol message passing interface (MPI) [67]. In this work, the MPI protocol will be used in a C++ environment. This choice enables a wide range of hardware compatibility (from small laptops to cluster stations) and low complexity in terms of code implementation. The immediate goal of this work is to be able to reproduce large-scale computations (e.g. modeling an RVE or small parts with hundreds of thousands of grains) using multiple CPU units (e.g. hundred of processors) with the most reasonable CPU-times.

Very few publications exist in the literature regarding the parallel implementation of FT models. Such examples can be found in [24, 25] in the context of two-phase flows and in [26] in a more general context of multiphase flows, however, tested for only just a few domains. Similarly, examples of vertex models using a parallel scheme can be found in [68, 69] using a *shared-memory* approach in the context of molecular-dynamics modeling. These examples are considerably different from our approach. Indeed, the TRM model avoids the use of two superposed discretization approaches (contrary to [24, 25]) and allows the discretization of domain boundaries by more than a vertex-vertex connection (contrary to [68, 69]).

In this paper, the TRM model's parallel implementation will be presented and tested in multiple hardware settings. The algorithms addressing the parallel implementation will be explained. Moreover, an additional tool is needed when performing parallel computations over a distributed memory approach in order to solve mesh-based problems: the initial partitioning of the numerical domain and the redistribution (when necessary) of charges (repartitioning) through the evolution of the simulation. We opted to use the open-source library Metis [70] to obtain the initial partitioning while a new redistribution algorithm has been developed and will also be presented in this paper. Finally, the model's performance and speed-up will be given and compared to other highly efficient parallel methods in the literature in the context of GG [46, 48].

2. The TRM model: sequential approach in a GG context

In [66], a numerical method for the TRM model was presented. This numerical method is built on a data structure defining the current state of the multidomain framework, defining

geometrical entities such as *points*, *lines*, and *surfaces*. Each *point* is composed of a *P*-Node (defining a mesh node with a topological degree equal to 0) and a set of connections to other *points* and *lines*. Each *line* is defined by an ordered set of *L*-nodes (nodes with a topological degree equal to 1), an initial *point*, and a final *point*. Finally, *surfaces* are defined by a set of *S*-Nodes (nodes with a topology degree equal to 2), a set of elements, and a set of delimiting *lines* and *points*.

Details regarding the construction of the data structure built upon an FE mesh can be found in [45, 66]. Natural parametric splines [71] are used to approximate the domain interfaces (*lines*) with third-degree piece-wise polynomials. These approximations are used to compute geometric properties such as curvature κ and normal $\vec{n}$.

Once a velocity has been computed on the mesh, depending on the physical model being simulated, the position of each node N_i of the mesh is updated using an explicit-forward Euler scheme. Mesh conformity (in an FE sense) is ensured by a local-iteratively movement-halving algorithm preventing element flipping (see [66]). Additionally, the TRM model involves a particular remeshing procedure. The TRM data structure needs to be maintained when the mesh evolves to ensure the definition of geometric entities. The TRM sequential remeshing algorithm can be found in appendix A.

Furthermore, the simulation of microstructural evolutions is given by the addition of complex and different phenomena such as GG, ReX, or ZP, all of which are good examples of multi-domain problems aimed to be modeled with the TRM method. In [66], isotropic GG was used to compare the TRM model to other approaches (LS-FE [11, 12, 15]). During GG, the energy minimization is driven by reducing the total amount and energy density of grain boundaries. It is well known that 'curved grain boundary migrates toward its center of curvature' [72], inducing an increase of the mean grain size by the disappearance of small grains and the growth of large grains. One of the first models involving the migration of grain boundaries, based on the observations given in [72–74], was proposed in [75, 76] in the form of a phenomenological law predicting the mean grain size. It was assumed that a driving force acting on the boundary was only produced by the surface tension of the boundaries, hence being directly proportional to its curvature. These conclusions were also based on the similarities encountered between the shapes of cells in foams and grains in polycrystals, pointed out by the authors in [74, 77]. In a full-field context, one of the formulations used to represent this phenomenon is commonly known as *curvature flow*, where the velocity $\vec{v}$ of every point of the interface is proportional to its local mean curvature:

$$\vec{v} = -M\gamma\kappa\vec{n}, \tag{1}$$

where M is the grain boundary mobility, γ the grain boundary energy, κ the local curvature in 2D and trace of the curvature tensor in 3D, and $\vec{n}$ the outward unit normal to the grain interface. Isotropic conditions will be considered hereafter: M is assumed only dependent on the temperature, which is constant over the space, and γ is constant.

Note that the velocity at multiple junctions cannot be deduced from equation (1) as the curvature and normal at these points cannot be mathematically computed. In the FT approach, the modeling of curvature flow is thus, ill-defined at multiple junctions, and equation (1) is not a good choice to define the velocity at these points. Here we used an alternative approach: model II of [4], where the product $\kappa\vec{n}$ is obtained from an approximation of a free energy functional in a vertex context.

The presented algorithm can be modified to simulate other different multi-domain problems by adapting the velocity equation defined in equation (1). Multiple topological changes of the polycrystal structure (grain disappearance and quadruple point dissociation) usually encountered in GG are also taken into account by the modeling algorithm. We refer the reader

to appendix B for details concerning the TRM algorithm for GG modeling in a sequential context.

3. Parallel strategy for the TRM model

A parallel TRM model, other than reducing the CPU-time of small/medium size computations (several tens of thousands of grains), enables the possibility of performing real scale full-field modeling of polycrystals (with millions of grains). Whether or not this could be interesting from a scientific point of view is still debatable, and it remains a perspective of this work. However, in order to test the parallel implementation introduced in this section, we aim to perform computations of GG with *at least* 5×10^5 number of grains and 100 time steps, which corresponds to the same scale encountered in publications such as [46, 48] using an FFT-LS approach.

As mentioned in the introduction, we will use here a *distributed memory* approach using the standard communication protocol 'MPI' [67] to communicate data between processes.

Moreover, two additional tools are needed to solve parallel mesh-based problems: (i) the *initial* partitioner, and (ii) the *repartitioner* of the numerical domain. These tools are used to balance the load (e.g. number of elements) each core has to handle at the beginning and during the simulation, respectively. In our context, the mesh is divided into several partitions equal to the number of running MPI processes (ideally, equals to the number of cores being used). Multiple libraries are available to divide a mesh made of simplices. Usually, these tools can work with topologies much more complex than an Eulerian mesh, as they are engineered to treat graphs[1] (see appendix C). Here we opted to use the free library Metis [70] to obtain the *initial* partitioning, but of course, other libraries can be used [78, 79]. Moreover, note that the initial partitioner tool (i.e. Metis) is intended to perform in sequential and cannot act as a *repartitioner* over a mesh distributed in the memory. Other libraries are available to accomplish this task (e.g. an extension of Metis called ParMetis [80]). However, we have opted to develop our repartitioner (see section 3.4), which enables us to optimize certain functionalities.

Finally, making changes to the mesh at the boundaries between partitions is difficult. In these regions, element patches are incomplete when seen from either of the adjacent partitions, as some part of the local patch of elements is not present in its memory. Typically, this is solved using halo regions, which repeat one or several layers of elements at the boundaries between partitions. We have not used this concept and a more straightforward methodology, known as zero-halo [81] approach, has been developed instead. At the same time, the remeshing strategy is handled using the properties of the repartitioning algorithm as explained in section 3.5.

Before explaining the different parts of our approach, the algorithm for a time step of the TRM model in parallel for the modeling of GG is summarized in algorithm 1.

3.1. Initial partitioning

The initial partitioning is performed on all of the k MPI running processes, asking for k number of partitions and using the function of Metis *METISPartMeshDual*, which makes a non-overlapping partition of elements (contrary to a not-overlapping partition of nodes as in [82, 83] see appendix C). The function *METISPartMeshDual* returns a list of tags with values in $0, 1, 2, \ldots, k$ which is fairly well distributed [84]. Finally, each MPI process i stores in memory the elements correspondingly tagged with the number i and discards the rest.

[1] Mathematical structures used to model pairwise relations (edges) between objects (vertices, nodes or points).

Algorithm 1. GG TRM algorithm in parallel.

Remeshing algorithm over non-blocked entities (algorithm 2 with the constraints explained in section 3.5)
Recompute load classification (section 3.4.1)
Mesh scattering (section 3.4.3)
Reconstruct geometries (appendix F)
Remeshing algorithm over non-blocked entities
Complete *lines* and *point* connections (temporary nodes) (section 3.6.1)
for all Points: P_i **do**
    while Number of connections > 3 **do**
        split multiple *point* P_i.
    end while
end for
for all Lines: L_i **do**
    Compute the spline approx. of L_i.
end for
for all L-nodes: LN_i **do**
    Compute curvature and normal ($\kappa\vec{n}$) over LN_i.
end for
for all P-nodes: PN_i **do**
    Compute the product $\kappa\vec{n}$ over PN_i using model II of [4].
end for
Delete temporary nodes
for all L-nodes and P-nodes: LPN_i **do**
    Compute velocity $\vec{v_i}$ of node LPN_i
end for
Iterative movement with flipping check in parallel (section 3.6.2)

Note that this procedure runs in sequential on each MPI process instead of being computed in only one process and then broadcasted to the others. This is because the broadcasting would add unnecessary overhead to the initial partitioning. Of course, this procedure is not optimized, and a better strategy would be to use ParMetis [80]. However, license constraints stop us from using this library.

3.2. Parallel identification of discrete geometric entities

In the present parallel context, it is possible that some geometric entities (*points*, *lines*, and *surfaces*) appear in several partitions at the same time. This creates issues as the algorithms used to reconstruct the geometric entities have been developed in a sequential context (see section 2.2 of [66]). As such, each partition will consider that the geometric entities stop at the boundaries between partitions. Problems as the one presented in figure 1 could arise: in this configuration with two *surfaces* and a *line*, the expected identification of each geometric entity should be the one illustrated in figure 1(a), instead, if the sequential numbering presented in [66] is used, the identification would be as illustrated in figure 1(c), multiple geometric entities of the same type may have the same identity and not correspond to the same entity (e.g. surf_1 from part_1 and surf_1 from part_2), or inversely, entities having different identities within the same partition but being the same entity in the whole domain (e.g. line_1 and line_2 from part_2 or surf_1 and surf_3 from part_2).

We have developed an algorithm to label all geometric entities such that they can be consistently tracked across the mesh boundaries. The details of this algorithm are explained in

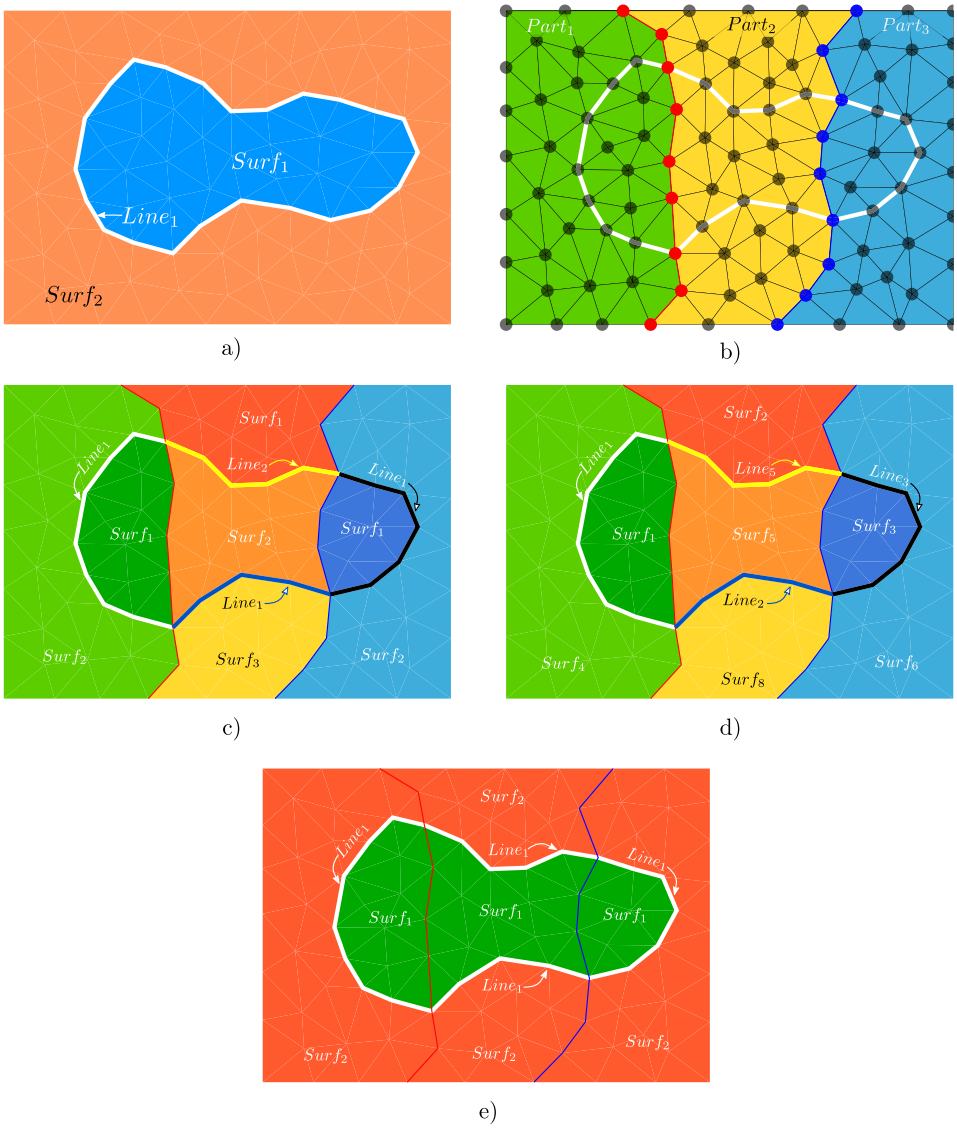

Figure 1. Example of the initial numbering of geometric entities of a domain crossed by the partition boundaries (red and blue edge trails). (a) Geometric entities sequentially numbered, (b) domain divided into three partitions: $part_1$ (green), $part_2$ (yellow) and $part_3$ (blue), (c) numbering of the geometric entities on each partition by default [66], (d) numbering of the geometric entities on each partition using the new numbering system: each geometric type (*point*, *line*, or *surface*) is numbered starting with its partition number and increasing by the number of total partitions, (e) configuration of the geometrical entities after regularization.

appendices D.1 and D.2. The numbering algorithm partially solves the issues presented in figure 1(c) and results in the numbering illustrated in figure 1(d).

3.3. Regularization

The regularization procedure solves inconsistencies in identifying entities that have been crossed by partition boundaries (e.g. turning the numbering of figure 1(d) into the numbering of figure 1(e)).

This procedure is implemented via a local identification of entities attached to *shared-nodes* (see section 3.4.2), these entities are renumbered using the lowest identity found for them across partitions; e.g. figure 1(d) shows that the internal *surface* has been identified as surf_1, surf_5 and surf_3 in partitions 1, 2 and 3 respectively. The algorithm chooses surf_1 and renumbers surf_5 and surf_3 with it. The entire regularization algorithm can be found in appendix D.2.

3.4. Rebalancing of loads, repartitioning: mesh scattering

Repartitioning in a distributed memory framework is much more complex than making an initial partitioning as all the data is scattered among all processes. Even though other tools solve these kinds of problems (e.g. ParMetis), we have opted to develop our repartitioner. This is mainly because we have to make sure that the data structure of the different geometric entities stay consistent during the repartitioning procedure, and controlling this procedure is key to avoid unnecessarily (or too frequent) mutations on the TRM data structure (see appendix F). Of course, our objective is not to develop a repartitioner with all the capabilities of the well established libraries of the community but to develop a robust and straightforward way of exchanging information between processes having parts of a scattered Eulerian mesh on its memory.

3.4.1. Process classification per load. The TRM model's repartitioning algorithm uses a classification system to determine which processes needs to increase or reduce its computational load by comparing them to their neighbours. All processes are classified following the number of elements, the number of nodes, or the total surface of the domain they manage. In this work, we will use the number of elements. Moreover, if two processes have the same load, a random attribution is held.

Each process sends its number of elements using the *MPI_alltoall* routine. Thus, each process computes and stores a list **PriorityOfRank** containing the *priority* of each partition to be load up, i.e. the lower the number of elements in a partition, the higher is its priority.

3.4.2. Shared-nodes. During the repartitioning, it will be necessary to know to which partitions each shared-node belongs. Consider, for example, figure C2(c) where some of the green nodes are shared with the cyan and yellow partitions. The cyan partition must know that these nodes are shared with the blue partition and vice versa. Note that one of the nodes is shared by the three partitions. As such, the blue partition must know that this node is shared with the yellow and red partitions. This information can be obtained easily: all partitions broadcast which nodes are at their boundary using *MPI_Allgatherv*, then the list of received nodes is matched against the local list of nodes, every match means that the node is shared with the sender partition. Every shared-node stores a list **SharedRanks** with the *id* of the partitions containing it.

3.4.3. Unidirectional element sending. One of the principal challenges during the repartitioning procedure, is to maintain consistency on the mesh. As explained in section 3.1, our parallel scheme maintains non-overlapping partitions of elements. As such, the unit of exchange during the repartitioning is also one element. If one partition exports one element, this element must have *only* one destination. This is achieved using algorithm 6 of appendix E, which employs the

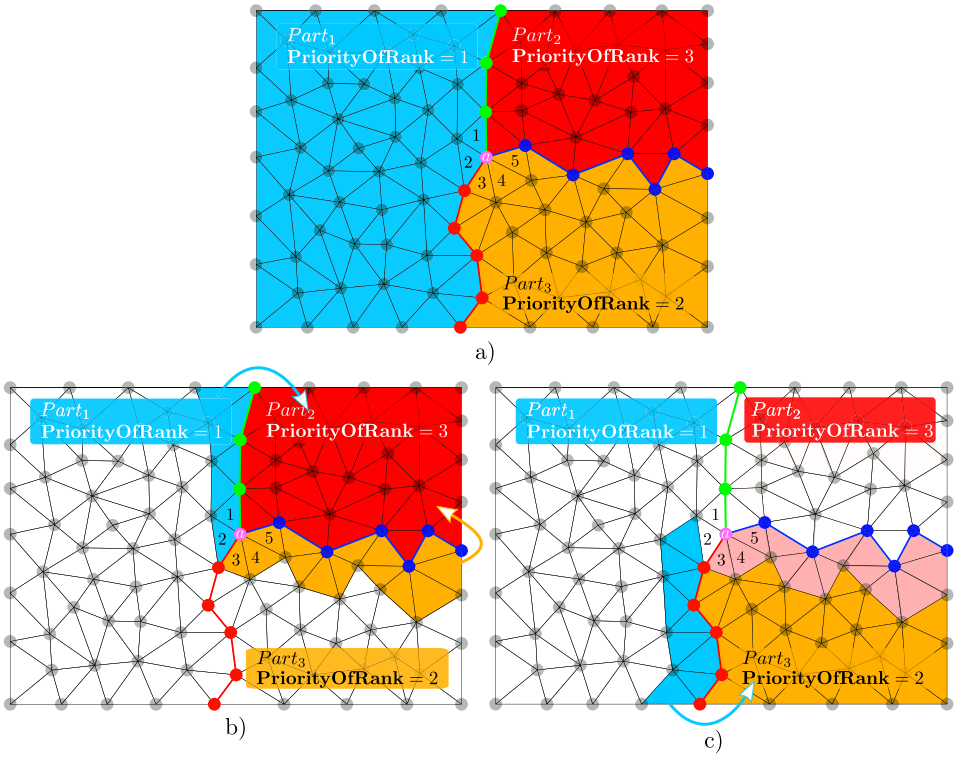

Figure 2. Example of the behavior of the unidirectional element selection algorithm. (a) Initial state with three parts, the name and the priority of each partition are displayed, (b) the elements to be sent to $part_2$ from $part_1$ and $part_3$ are cyan and yellow respectively, elements 3, 4 and 5 appear only to be sent to $part_2$, and not to $part_1$. Elements 1 and 2 appear initially on the list to be sent to $part_2$ and $part_3$, but they are filtered in the last part of the algorithm (line 15 of algorithm 6), (c) selected elements to be sent to $part_3$, the intersection of elements to be sent from $part_1$ to $part_2$ and $part_3$ is empty.

global list **PriorityOfRank** (section 3.4.1) and the **SharedRanks** lists of shares-nodes (section 3.4.2).

Figure 2 illustrates one example of the behavior of algorithm 6. The **PriorityOfRank** list is computed accordingly to the number of elements in decreasing order. In figure 2(b)) the elements to be sent to $part_2$ from $part_1$ and $part_3$ are cyan and yellow, respectively. Here, elements 3, 4, and 5 appear to be sent to $part_2$, and not to $part_1$, applying the filter of line 7 (**PriorityOfRank**[$Part_3$] $<$ **PriorityOfRank**[$Part_1$]), which returns false for these elements. Elements 1 and 2 appear initially on the list to be sent to $part_2$ and $part_3$ but are filtered in line 15 of the algorithm because $part_2$ has a higher priority. In figure 2(c) the elements to be sent to $part_3$ are cyan. Note that some of the elements of $part_3$ are going to be sent to $part_2$ hence they appear in a different color. Finally, the intersection of elements to be sent from $part_1$ to $part_2$ and $part_3$ is empty, and thus consistency is ensured in the scattering process. Figure 3 illustrates the initial and the final configuration after the scattering. All parts' boundaries have been displaced by the scattering, including the shared-node a (which is shared by the three partitions before the scattering). After the scattering, node b is shared by the three partitions, while node a is part of the bulk nodes of $part_2$.

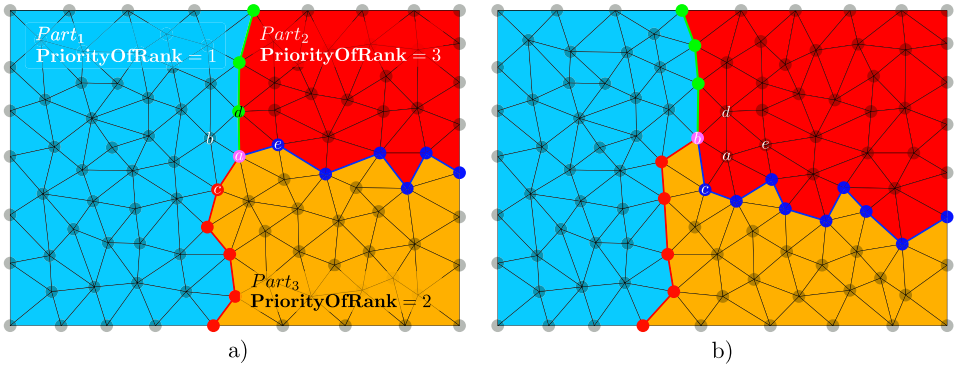

Figure 3. Example of the configuration of figure 2 after scattering the elements, the boundaries of all partitions are displaced, (a) initial configuration, (b) configuration after scattering.

3.5. Blocking remeshing at partition boundaries

Applying remeshing operators at the partition boundaries is very difficult because part of the local patch of elements might not be accessible. While this is typically solved using halo regions, we have solved this more simply: (i) blocking operations producing a change (change of connectivity or position) on the edges of partition boundaries, and (ii) remeshing once, then repartitioning (partition boundaries change), then remeshing the initially blocked regions.

For example, in figure 3(a), *node collapse*, *edge swapping*, and *edge splitting* are blocked for the edge $\overline{ac}$ (the edge delimited by nodes a and c). Note, however, that *edge swapping* and *edge splitting* are allowed on edge $\overline{de}$, as this would not change any of the edges defining the partition boundaries. Of course, the repositioning of any of these nodes is not allowed.

During the repartitioning procedure presented in section 3.4, a complete layer of elements between the sender and the receiver partitions is exchanged, hence completely changing the partition boundaries every time it is performed. Figure 3(b) illustrates how all the blocked operations mentioned above between edges $\overline{ac}$ and $\overline{de}$ are unblocked since these edges no longer belong to the boundary. A new remeshing procedure can be executed for all elements and nodes previously blocked, finishing the remeshing procedure of the whole domain.

However, note that our solution for the remeshing procedure in parallel can produce different results from the ones obtained with a sequential remeshing as the different remeshing operations will not be performed in the same order. Nonetheless, this is expected to be of minimal impact on our model's precision (see section 4).

3.6. Other parallel treatments

The computation of properties at shared-nodes and the Lagrangian movement of these nodes cannot be computed using the default sequential way as in [66], this is because shared-nodes cannot access some information of their surroundings (halo regions are not used). The solutions to these difficulties are detailed in the following sections.

3.6.1. Computation of properties at shared-nodes. To obtain a velocity using equation (1) (GG phenomena), the mean curvature κ and the normal $\vec{n}$ of the interfaces are needed. As mentioned in section 2, these properties are computed for L-nodes using a spline approximation [71]. Thus, the position of nodes adjacent to shared L-nodes is needed. This information is

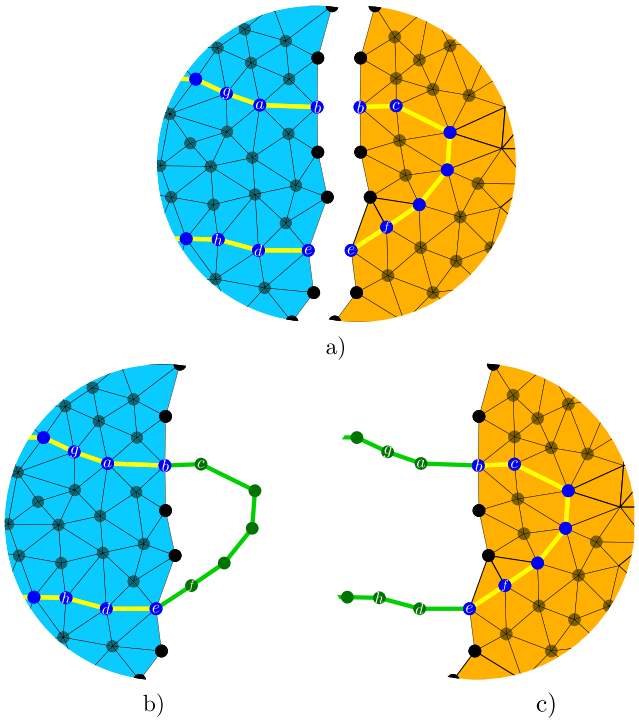

Figure 4. Example of *line* patch with temporal nodes (nodes in green), (a) configuration for two partitions, (b) part_1 after adding the required temporary nodes (c) part_2 after adding the needed temporal nodes.

transmitted by the neighboring partitions and stored in *temporal* nodes. Figure 4 illustrates this situation: blue nodes correspond to *L*-nodes belonging to the local partition while green nodes correspond to temporal nodes. Geometrical properties can now be computed for the *L*-nodes *b* and *e* on each partition, obtaining the same result in both of them. These temporal nodes only exist during the computation of the geometrical properties. A similar operation is done for *P*-Nodes. However, only one adjacent node is necessary for the computation of model II of [4] (used for the calculation of the points velocity field in our model).

3.6.2. Lagrangian movement in parallel. In [66], every time one node moves, the algorithm checks for invalid configurations (flipped elements, see figure 5). In parallel, this has been implemented as follows: each partition makes its respective check over the elements known to it, if a flipped element is found, the ID of the node responsible for the event is sent to the partitions where it is shared, then the movement is executed with half the displacement in a further iteration.

4. Numerical results

In [66], three academic test cases corresponding to the circle-shrinkage, T-junction, and square-shrinkage tests, were performed in order to validate the accuracy of the TRM model. A final test was performed to validate the model in a more realistic environment, where a 2D-RVE material composed of 10 000 initial grains was simulated using curvature flow and properties

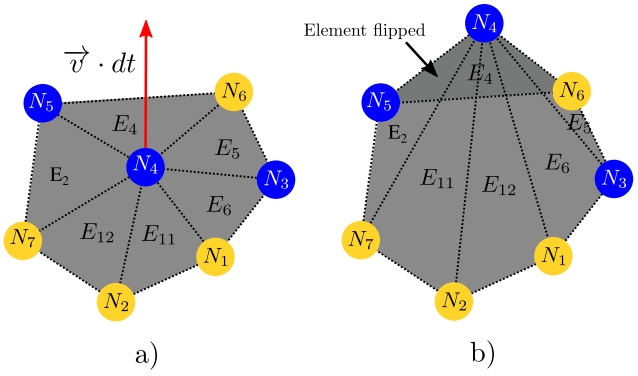

Figure 5. Example of element flipping, (a) initial state, the displacement vector of N_4 $\vec{v} \cdot dt$ lies outside the element patch. (b) State after updating the position of N_4, element E_4 has been flipped [66].

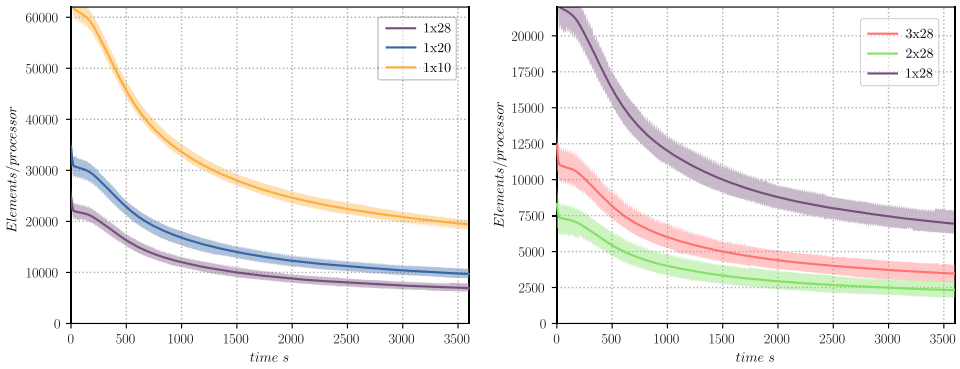

Figure 6. Mean number of elements per core for the test with a surface of 50 mm^2 performed in 10, 20, 28, 56 (2 × 28), and 84 (3 × 28) cores. The evolution of the number of elements for the simulations contained in one node (left) and in multiple nodes (right) is shown. The range for the number of elements of all cores in the same simulation is shown in the same color with an alpha component.

of a 304L stainless steel. In this article, two sets of 2D simulations were performed: for the first set, domains of fixed size using an increasing number of cores (strong scaling) were simulated. For the second set, the average computational load per core remained approximately the same. This was realized by upscaling the number of cores N_p and the simulation domain's size (weak scaling). Finally, each simulation was performed four times, and the times obtained were averaged.

All simulations were performed on a cluster facility composed of *nodes*[2] Bullx R424 equipped with two processors Intel Xeon E5-2680v4 (at 2.4 GHz) with 14 cores each, for a total of 28 cores per node. Each node has 128 Gb of RAM memory. The nodes are connected using an Infiniband FDR at a speed of 56 Gb s^{-1}. All libraries were compiled using gcc 7.3.0

[2] Here the meaning of the word node represent a physical CPU connected in the network of the HPC.

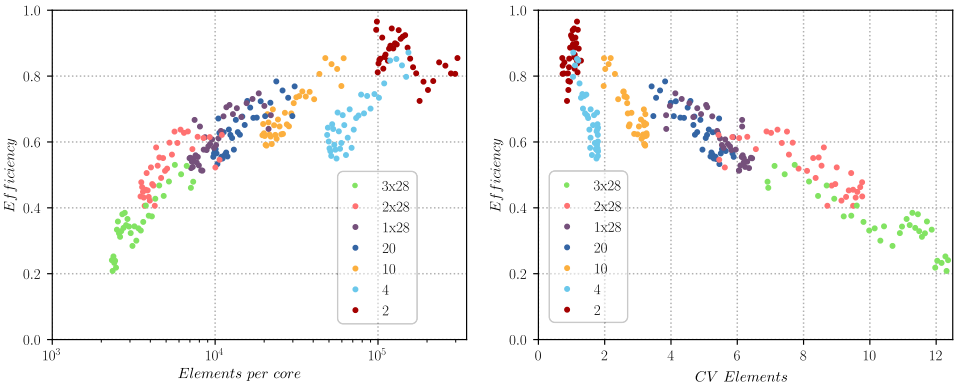

Figure 7. Efficiency of the simulation for the case with a surface of 50 mm^2 against the coefficient of variation CV of the number of elements across partitions (right). The data has been averaged over 10 increments.

in a Centos 6.7 environment. The version of MPI used was OpenMPI 1.10.7 with default settings, and each core managed one MPI process. Hereafter we will use the syntax $N_n \times N_p$ to describe the configuration used in our computations, where the term N_n describes the number of nodes used and N_p the number of cores used per node.

4.1. Strong scaling benchmark

Here, two similar tests to the one presented in [66] with 10 000 grains have been performed, modifying the surface of the RVE to fit first 50 000 and then 560 000 grains, but maintaining the same initial grain size distribution, the same thermal treatment (1 h at 1050 °C) and identical physical properties: the generation of the initial tessellation has been performed with a Laguerre–Voronoi cell generation procedure [85–87] over a squared domain and the values for M and γ are chosen as representative of a 304L stainless steel at 1050 °C (with $M = M_0 * \mathrm{e}^{-Q/RT}$ where M_0 is a constant $M_0 = 1.56 \times 10^{11}$ mm^4 J^{-1} s^{-1}, Q is the thermal activation energy $Q = 2.8 \times 10^5$ J mol^{-1}, R is the ideal gas constant, T is the absolute temperature $T = 1323$ K and $\gamma = 6 \times 10^{-7}$ J mm^{-2}) [16, 17]. Moreover, the initial grain radius distribution is imposed as a log-normal distribution curve with a median value of 0.017 mm and a standard deviation of 0.006 mm. Additionally, the maximal and minimal limits for the grains radius are defined as 0.04 mm and 0.011 mm, respectively. With these values, approximately 1000 grains fit in a surface of 1 mm^2 of the surface. As such, the simulation with 50 000 grains fits in a domain with 50 mm^2 and the one with 560 000 grains in a domain of 560 mm^2 of the surface. After the generation of the Laguerre–Voronoi tessellation, the real number of grains were 52 983 and 544 913 grains for the domains of 50 and 560 mm^2, respectively. Finally, the mesh size parameter and the time step have been held constant and equal to $h_{\mathrm{trm}} = 0.004$ mm and d$t = 10$ s, respectively, accordingly to [66].

4.1.1. 2D grain growth 50 000 initial grains. Multiple simulations for the case with 50 mm^2 of surface were performed using up to 84 (3×28) cores. In appendix G, figure G1 gives the evolution of the mean grain size pondered by surface and the L2-error (measured with respect to the sequential case). Here, all parallel tests behave almost exactly like the sequential one (1 core). Similarly, figure G2 shows the evolution of the mean grain size distributions. Once again, the results are almost identical for all the tests. Deviations from the sequential computation are

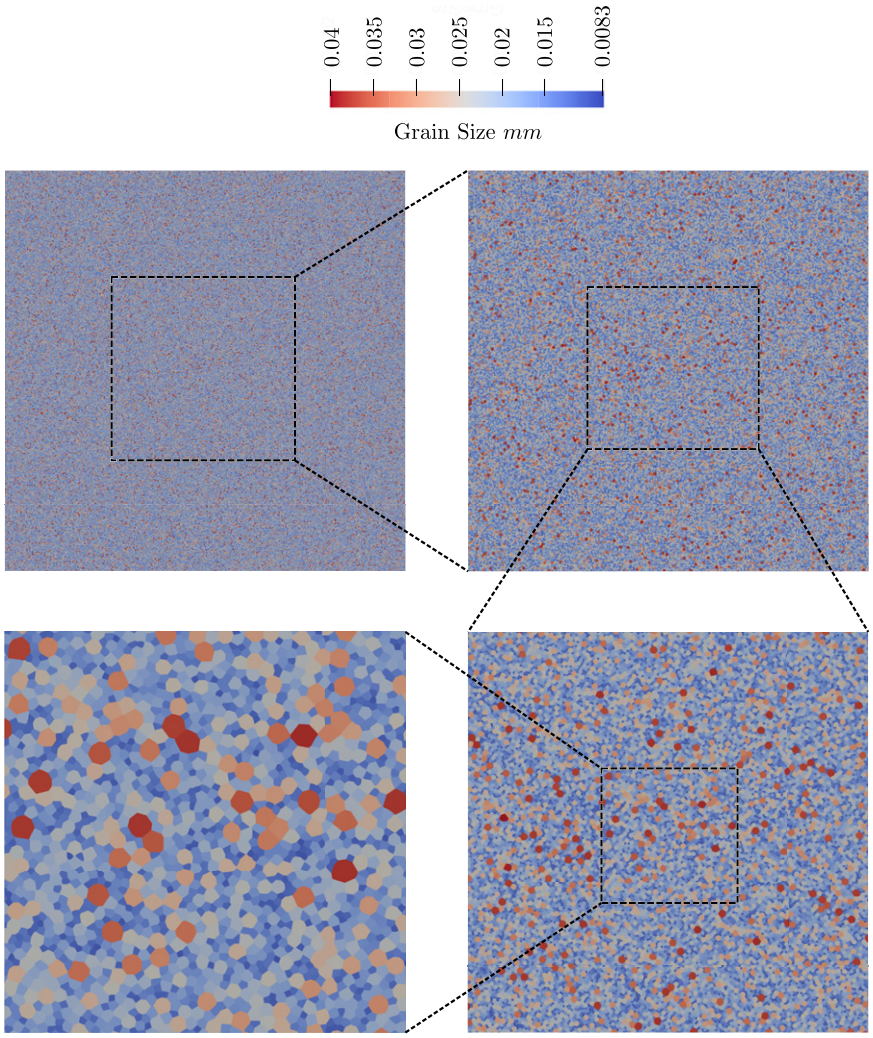

Figure 8. Initial state of the case with a surface of 560 mm^2, 544 913 grains are shown in the bigger image. Subsequent zoom views are given.

attributed to the fact that the remeshing procedure is not performed in the same order in parallel than in sequential (see section 3.5)

Figure 6 describes the mean number of elements and its range for the simulations performed in 10, 20, 28, 56 and 84 cores. Here, two plots are given: for the simulations performed in one CPU (one node) and in multiple CPUs (multiple nodes). Note that the range increases when the number of cores increases as the partition boundaries present more changes. However, the range is well contained around the mean number of elements per core. The balancing of loads appears to have good behavior.

One interesting index to follow is the coefficient of variation CV (or relative standard deviation) measured over the number of elements. This index reflects the spread of the imbalance across partitions, hence one can use it to study the global efficiency of the numerical procedure.

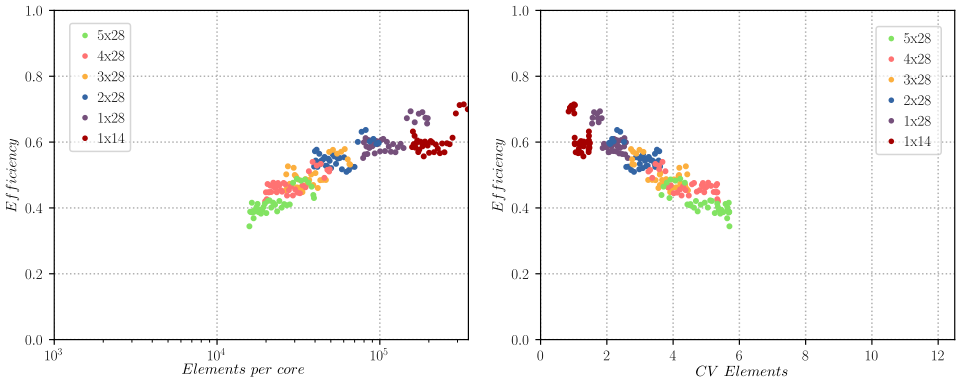

Figure 9. Efficiency of the simulation for the case with a surface of 560 mm^2 against the coefficient of variation CV of the number of elements across partitions (right). The data has been averaged over 10 increments.

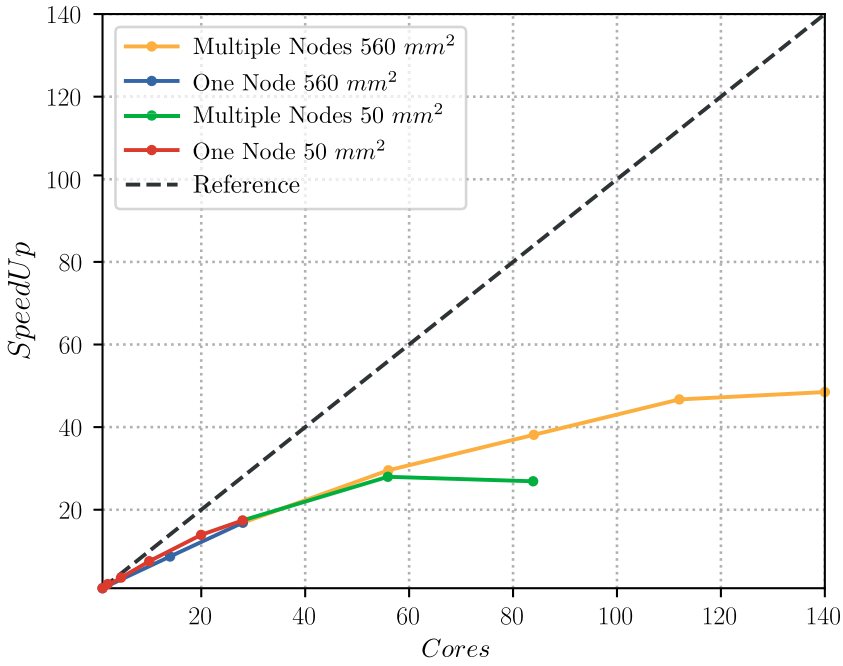

Figure 10. Performance of the TRM model in a strong scaling benchmark: the domain surface is maintained constant while the number of cores increases, two test were performed, one with a surface of 50 mm^2 and the second with a surface of 560 mm^2. The data is compared to the optimal speed up (reference).

During strong scaling, we define the efficiency of one increment i for a simulation with N_p number of cores as follows:

$$(\text{Efficiency})_i = \frac{t_1^i}{t_{N_p}^i \cdot N_p}, \tag{2}$$

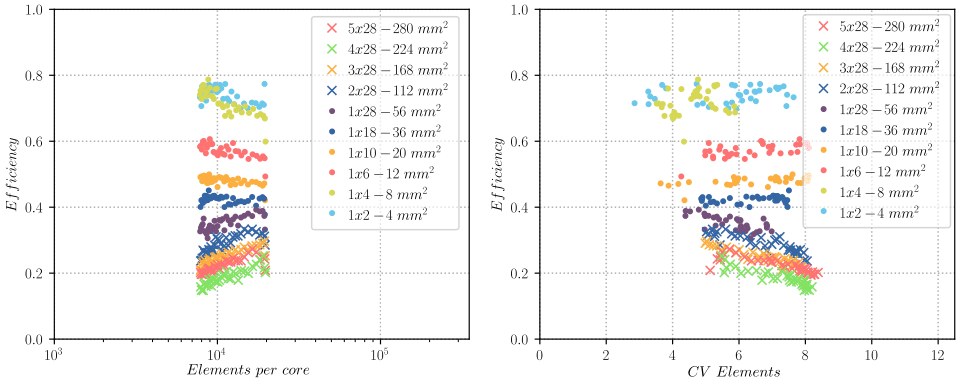

Figure 11. Mean number of elements per core for the test with a DSPC of 2 mm^2 performed in 2, 28, 84 (3 × 28), and 140 (5 × 28) cores. The range for the number of elements of all cores in the same simulation is shown in the same color with an alpha component.

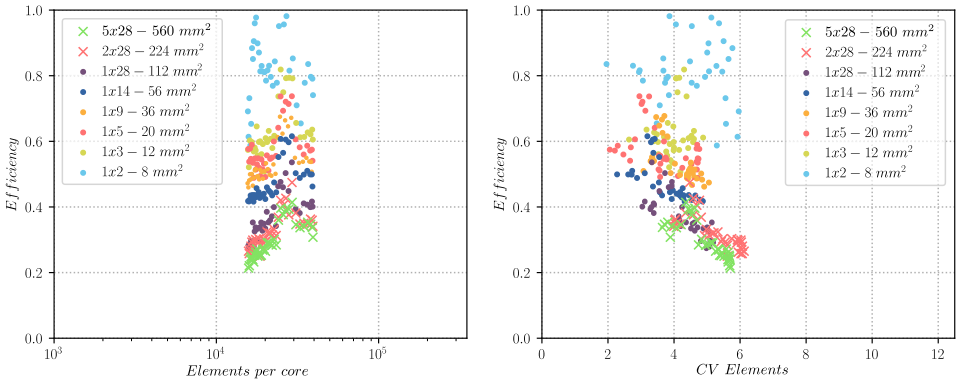

Figure 12. Mean number of elements per core for the test with a DSPC of 4 mm^2 performed in 2, 28, 84 (3 × 28), and 140 (5 × 28) cores. The range for the number of elements of all cores in the same simulation is shown in the same color with an alpha component.

where the term $t^i_{N_p}$ determines the CPU-time needed to perform the ith in a simulation with N_p the number of cores. This equation computes the amount of resources needed for one increment of a parallel simulation compared to a sequential one. Figure 7 plots the simulation's efficiency for the case with 50 000 grains against the mean number of elements per core (left) and against the CV index (right). Of course, the lower the efficiency per increment for a given simulation, the lower we expect to be its speed-up. The mean efficiency of the 84 cores test (around 0.29) is much lower than the mean efficiency of the 56 cores test (around 0.48), thus a reduction of the efficiency of 65% for an increase in the number of cores of 50% (from 56 to 84 cores). These results suggest that the simulation performed with 84 cores is over partitioned and that one should aim to obtain a number of elements per core higher than 10 000 or a CV index lower than 8%.

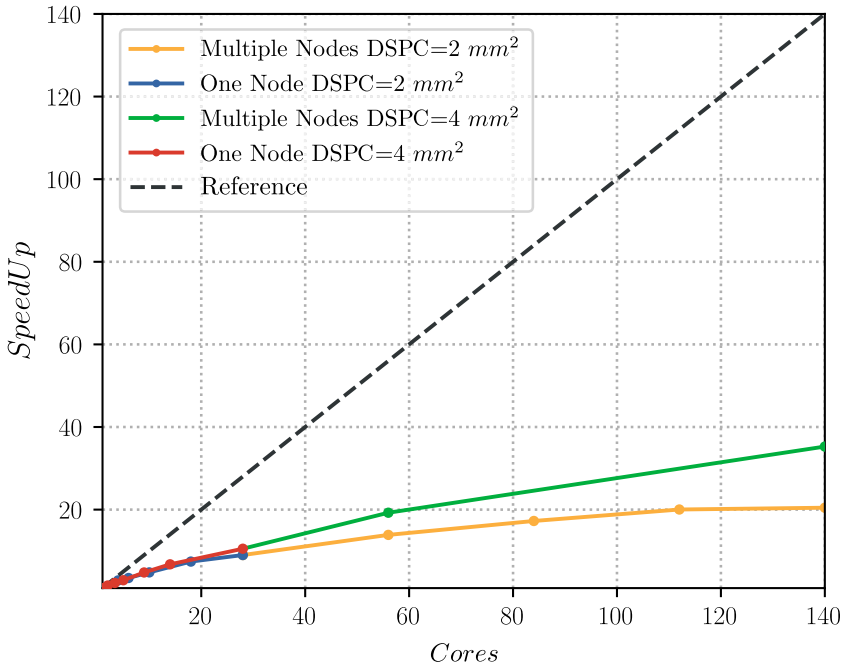

Figure 13. Performance of the TRM model in a weak scaling benchmark: the domain surface increases proportionally to the number of cores, two tests were performed, the first with a DSPC of 2 mm^2 (approximately 2000 grains per core) and the second with a DSPC of 4 mm^2 (around 4000 grains per core). The data are compared to the optimal speed up (reference).

Finally, figure 10 plots the speed-up of the parallel implementation of the TRM model against the number of cores. Here, the reference for the optimal speed-up is also shown. Of course, the optimal speed-up cannot be reached as communications create additional overhead.

4.1.2. 2D grain growth 560 000 initial grains. Let us extend the RVE surface to 560 mm^2 and the number of cores to 140 (5 $\times$ 28). Figure 8 illustrates the initial microstructure for this case. To the knowledge of the author, this is the maximum amount of grains that have been modeled using 2D unstructured finite-element meshes, and the second-largest simulation in GG context. The first is the one presented in [48] for a microstructure with 671 000 initial grains, using a FFT and regular grids.

Figure H1 gives the results for the evolution of the mean grain size and its L2-error for the tests with 560 mm^2 of surface. The range of the L2-errors has been reduced by at least 1/5 compared to the previous tests (50 mm^2). At this level, we consider that the parallel TRM model's influence over the precision of the simulation is quasi-non-existent.

The index CV is reduced by increasing the RVE surface as the mean number of elements per core is increased. Figure 9 plots the simulation's efficiency for the case with 560 000 grains against the mean number of elements per core (left) and against the CV index (right); efficiency is maintained over 0.4.

Figure 10 illustrates the evolution of the speed up for both strong scaling cases. The maximum speed-up obtained was for the case with a surface of 560 mm^2 performed over 140 (5 $\times$ 28) cores for which a speed-up of 48.5 was obtained. This simulation took 38 min and 45 s,

Algorithm 2. Remeshing algorithm [66].

for all Nodes: N_i **do**
  for all Neighbours of N_i: N_j **do**
    if $\delta_c(N_i, N_j) < |\overline{N_i N_j}|$ **then**
      selective node collapse: N_j collapses N_i
    end if
  end for
end for
for all S-nodes: SN_i **do**
  selective vertex smoothing: SN_i
end for
for all L-nodes: LN_i **do**
  selective vertex gliding: LN_i
end for
for all Edges: $\{N_i, N_j\}_{j>i}$ **do**
  if $\delta_s(N_i, N_j) > |\overline{N_i N_j}|$ **then**
    selective edge splitting: N_i, N_j
  end if
end for
for all Elements with $Q_s < q_s$: E_i **do**
  for all Edges of E_i: $\{N_j, N_k\}_{k>j}$ **do**
    if quality $Q_{mean}(N_j, N_k)$ will improve by swapping **then**
      selective edge swapping: $\{N_j, N_k\}$
    end if
  end for
end for

while the one achieved in 1 core took 31 h and 20 min. The total number of grains at the end of this simulation was 117 157, hence 21.5% of the total initial number of grains, while for the simulation with 50 mm^2 of surface, it was 10 399, hence 19.6% of the total initial number of grains.

4.2. Weak scaling benchmark

When performing a benchmark on a strong scaling context, the amount of memory that a core has to maintain decreases when the number of cores increase. Of course, when performing a sequential simulation, all the memory has to be maintained by only one core, making it longer to read or to add pieces of information to the data set of the running process. This artifact makes that the optimization made by the parallel implementation in a strong scaling context, be both, in memory management (to access and to write in a given memory location) and in number of operations (such as remeshing or moving nodes) to perform by a core. Instead, a weak scaling benchmark aims to measure the speed-up[3] generated by a parallel implementation only on the number of operations performed by a core. While it is impossible to maintain a perfectly balanced and constant load on all cores, it is possible to approach weak scaling by increasing the size RVE proportionally to the number of cores used. In this section, we give the results of two sets of simulations: the first with a domain surface per core (DSPC) of 2 mm^2 (approximately

[3] Contrary to the speed up in strong scaling, the speed up for a weak scaling benchmark is obtained by multiplying the CPU-time of the sequential case with the number of cores of the parallel case, then dividing by the CPU-time of the parallel case.

Algorithm 3. Isotropic GG TRM algorithm.

1: Perform **remeshing algorithm** (algorithm 2)
2: **for all** Points: P_i **do**
3: **while** Number of connections > 3 **do**
4: split multiple point P_i.
5: **end while**
6: **end for**
7: **for all** Lines: L_i **do**
8: Compute the natural spline approximation of L_i.
9: **end for**
10: **for all** L-nodes: LN_i **do**
11: Compute curvature and normal ($\kappa\vec{n}$) over LN_i.
12: **end for**
13: **for all** P-nodes: PN_i **do**
14: Compute the product $\kappa\vec{n}$ over PN_i using model II of [4].
15: **end for**
16: **for all** L-nodes and P-nodes: LPN_i **do**
17: Compute velocity $\vec{v_i}$ of node LPN_i
18: Iterative movement with flipping check over LPN_i
19: **end for**

2000 grains per core) and the second with a DSPC of 4 mm^2 (approximately 4000 grains per core). Moreover, even though the same grain size distribution is used in the Laguerre–Voronoi tessellation generation, it is not possible to obtain a perfectly equal statistical distribution for different sizes of domains. As such, at the beginning of the simulations, minor variations on the mean grain size or the grain size distributions may appear.

Figure I1 summarizes the results for the evolution of the mean grain size of the cases with a DSPC of 2 mm^2 along with its L2-error using the largest simulation as a reference (5 $\times$ 28-280 mm^2). The curves appear to be very similar to their references, with an error inferior to 3%. Note that the L2-error has a tendency to decrease when the simulation domain increases (as expected) and that the error after a total surface of 56 mm^2 (approximately 56 000 initial grains) is inferior to 1%.

In the appendix, figure I2 illustrates the evolution of the mean size distribution in surface for one set of simulations (DSPC of 2 mm^2), at the beginning and after 3600 s of simulated time. Figure I3 gives the evolution of the L2-error over the grain size distributions for both sets of simulations. Similarly to the evolution of the mean grain size, the L2-error is obtained to be lower than 3% with a domain size of at least 56 mm^2, suggesting that our microstructure can be statistically well represented by a simulation with over 50 000 initial grains. On the contrary, simulations performed with a domain of 20 mm^2 and below show an error higher than 5%, suggesting that they contain too few grains (at the end of the simulation) or that boundary conditions may highly influence statistical results. Simulations with 20 000 grains and below should be avoided in our GG context.

The evolution of the efficiency, the number of elements per core, and the CV index are presented in figures 11 and 12 for both sets of simulations. One can note that the number of elements per core is contained in a given range for both sets (weak scaling), while the efficiency drops when the number of cores increases. Moreover, the sets using a DSPC of 4 mm^2 have a lower overall CV value and a higher overall efficiency.

Algorithm 4. Identity Regularization algorithm performed on part_i.

1: **for all** Shared-nodes: N_i **do**
2: **for all SharedRanks** of N_i: part_j **do**
3: $I_\text{L} \leftarrow$ local identity of the coupled entity of N_i
4: send to part_j: pair(N_i, I_L)[a]
5: **end for**
6: **end for**
7: **for all** Parts: $\text{part}_j \neq \text{part}_i$ **do**
8: **for all** Received pairs from part_j: pair_k **do**
9: $N_i \leftarrow$ the node first(pair_k)
10: $I_\text{L} \leftarrow$ local identity of the coupled entity of N_i
11: $I_\text{R} \leftarrow$ second(pair_k)
12: send to all parts: triplet(I_L, I_R, part_j)[b]
13: **end for**
14: **end for**
15: **while** Triplets to treat **do**
16: $TT \leftarrow$ take first non-treated triplet
17: Create empty list of pairs: *SameEntity* ▷ [identity, partition]
18: Call **RecursiveTripletTreatment**(TT, *SameEntity*)
19: $I_\text{Lowest} \leftarrow$ the lowest value of the first item in all pairs of *SameEntity*
20: **for all** Pairs in *SameEntity*: pair_k **do**
21: **if** Second(pair_k) == part_i **then**
22: $I_\text{Old} \leftarrow$ first(pair_k)
23: change identity of entity with number I_Old to I_Lowest
24: **end if**
25: **end for**
26: **end while**

[a]A pair is a structure of data with two objects, the function first(pair_i) extracts the first object in pair_i and the function second(pair_i) extracts the second object.
[b]A triplet is a structure of data with three objects, the function first(pair_i) extracts the first object in pair_i, the function second(pair_i) extracts its second object and the function third(pair_i) extracts its third object.

Algorithm 5. Recursive function for the identification of the same entity given a triplet Tx and a list of pairs to fill P_L.

1: **function** RecursiveTripletTreatment(Tx, P_L)
2: $I_\text{L} \leftarrow$ first(Tx) (local identity)
3: $I_\text{R} \leftarrow$ second(Tx) (remote identity)
4: $\text{Part}_r \leftarrow$ third(Tx) (remote partition)
5: **for all** Received triplets from part Rpart: triplet_i **do**
6: **if** Triplet_i is still not treated and first(triplet_i) == I_R **then**
7: Set triplet_i as treated
8: Add to P_L: pair(second(triplet_i), third(triplet_i))
9: Call **RecursiveTripletTreatment**(triplet_i, P_L)
10: **end if**
11: **end for**
12: **end function**

Algorithm 6. Unidirectional element selection executed in part part_i.

```
Store a list of lists: SharedNodesPerPart
for all Shared-nodes: N_i do
    for all SharedRanks of N_i: part_j do
        Add item N_i to SharedNodesPerPart[part_j]
    end for
end for
Store a list of lists: ElementsToSendToPart
for all Parts: part_j do
    if PriorityOfRank[part_i] <PriorityOfRank[part_j] then
        for all SharedNodesPerPart[part_j]: N_j do
            Add elements(N_j) to ElementsToSendToPart[part_j]
        end for
        Take out repeated elements in ElementsToSendToPart[part_j]
    end if
end for
for all Parts: part_j do
    for all ElementsToSendToPart[part_j]: E_j do
        for all Nodes(E_j)[a]: N_k do
            for all SharedRanks of N_k: part_k do
                if PriorityOfRank[part_j] <PriorityOfRank[part_k] then
                    Erase E_j from ElementsToSendToPart[part_j]
                end if
            end for
        end for
    end for
end for
```

[a]Function nodes(entity) extracts the nodes present in entity, where entity can be a point, a line, a surface or an element.

Figure 13 describes the speed-up of both sets of simulations. For the simulations performed in one node (28 cores max.) the speed-up is very similar, while above 28 cores (multiple nodes), the speed-up favors the simulations with a higher DSPC.

5. Discussion, conclusion, and perspectives

The parallel implementation of the TRM method employed several sub-algorithms addressing different tasks: the partitioning of the domain, the re-numbering scheme developed to track geometric entities across partitions, the *mesh scattering* algorithm allowing to balance the load between partitions, a simplified parallel remeshing procedure and the Lagrangian movement in parallel. When performed in sequential and parallel, the accuracy of our model was studied, obtaining the same response with negligible errors in every test.

The maximum speed-up observed was for a case with 544 913 initial grains using 140 (5×28) cores for which a speed-up of 48.5 was obtained. This simulation took 39 min to end (360 increments). Moreover, it was observed that for simulations performed with a high number of partitions the speed-up might not be satisfactory if the number of elements is too low (lower than 10 000 elements per partition).

Weak scaling and strong scaling benchmarks were performed. The results of these simulations confirmed that the higher the number of initial grains considered, the higher the

simulation's efficiency (and speed-up). The TRM model shows a good behavior for both kinds of benchmarks although with conservative efficiency and speed-up.

Another observation of the weak scaling benchmark was that the L2-error drops below 3% for an initial number of grains larger than 50 000. On the contrary, simulations performed with 20 000 initial grains and below show an error of more than 5%, which suggest that they might contain too few grains at the end of the simulation or that boundary conditions may highly influence statistical results (in our case, orthogonality of grain boundaries with the domain limits). Globally, it seems that simulations with lower than 4000 grains at the end of simulations should be used with care. This also enforces the argument behind the studies aiming to increase the performance of massive multidomain simulations as the main obstacle to increase the number of domains is given by their high computational cost.

In the context of massive multidomain simulations, to the knowledge of the authors, only two methodologies in the literature have attempted to perform simulations with hundreds of thousands of grains: the one in [46] with a maximum number of 100 000 grains and the one in [47, 48] with 671 000. Results in [46] showed a better speed-up of their parallel implementation than the one presented in this article for our TRM model. Concerning the CPU-time, the parallel strategy presented in [46] was able to perform 20 increments of a simulation with 100 000 initial grains in 32 s when performed in 128 cores. In [48], no data concerning the speed-up was provided. However, it was mentioned that a simulation with 671 000 initial grains was performed until less than 4000 grains remained, with a total CPU-time in the range of 9 to 12 days, when running on 18 intel Nahalem cores. Moreover, the main originality of the proposed model here is, for the first time, to exhibit very efficient simulations in the context of unstructured FE meshes (regular grids in FFT context are considered in [46, 48]). Indeed, our strategy will enable us to consider large deformation of the calculation domain, paving the way to more complex mechanisms such as DRX. Our meshing/remeshing strategy, by conserving a description of the bulk of the grains, will also make it possible to investigate intragranular fields.

This parallel scheme's implementation corresponds to the first perspective fulfilled for the general TRM approach. Other perspectives concern: the development of the DRX and post-DRX TRM model, the possibility to perform simulation taking into account anisotropic grain boundary properties, and the extension of the algorithms to a 3D context. These questions will be discussed in future works.

Acknowledgments

The authors thank the ArcelorMittal, ASCOMETAL, AUBERT & DUVAL, CEA, FRAMATOME, SAFRAN, TIMET, Constellium and the ANR for their financial support through the DIGIMU consortium and ANR industrial Chair (Grant No. ANR-16-CHIN-0001). The authors also would like to thank Dr. Ing. Markus Kühbach, from the Fritz-Haber-Institut of the Max Plank Haber Society, for the interesting exchanges on the theme of this article.

Data availability statement

The data generated and/or analysed during the current study are not publicly available for legal/ethical reasons but are available from the corresponding author on reasonable request.

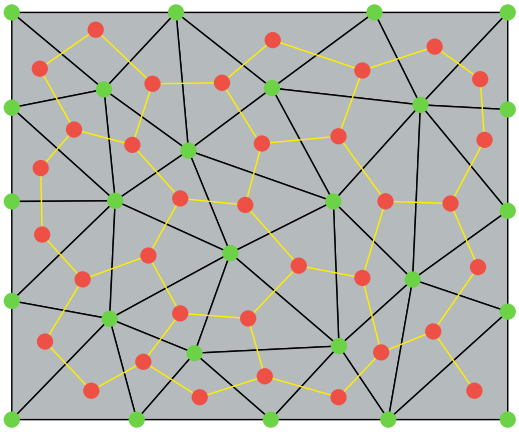

Figure C1. Example of a mesh and its dual graph in 2D, the green points are nodes of the mesh, and the red points are nodes of the dual graph of the mesh.

Appendix A. Sequential remeshing algorithm

When a remeshing procedure is performed, the mesh evolves, and the sets defining each geometric entity have to be adapted. Therefore, the remeshing procedure must take into account the local data structure of the nodes and elements involved in each remeshing operation.

The remeshing strategy of the TRM model uses the separate definition of locally *selective*[4] remeshing operators: selective vertex smoothing, selective node collapsing, selective edge splitting, selective edge swapping, and selective vertex gliding (see [66] for a complete definition of each operator). As a general rule, the remeshing procedure is performed to increase the general quality Q of the mesh (or a patch of elements of the mesh). Here, the notion of mesh quality Q is computed as a factor of the shape and the size of the elements using the same approach as in [88]. However, the selective remeshing procedure is not only driven by the local mesh quality Q but also by the local topological degree of the nodes involved in the operation. In [66], a global remeshing procedure was introduced, driven by two nodal fields δ_c and δ_s corresponding to the collapsing and splitting fields, and a minimum quality shape coefficient q_s. The complete remeshing procedure is summarized in algorithm 2.

Appendix B. TRM sequential algorithm for grain growth

The complete algorithm for an increment of the TRM model using one core is presented in algorithm 3. Note that line 4 states 'split multiple point P_i', this procedure is explained in [66] and corresponds to the decomposition of unstable multiple junctions that may appear during the simulation.

[4] The word selective denotes a variation of the original remeshing operations when performed over the data structure of the TRM model. Each remeshing operation is performed differently over nodes with different topologies (*P*-node, *L*-node and *S*-node).

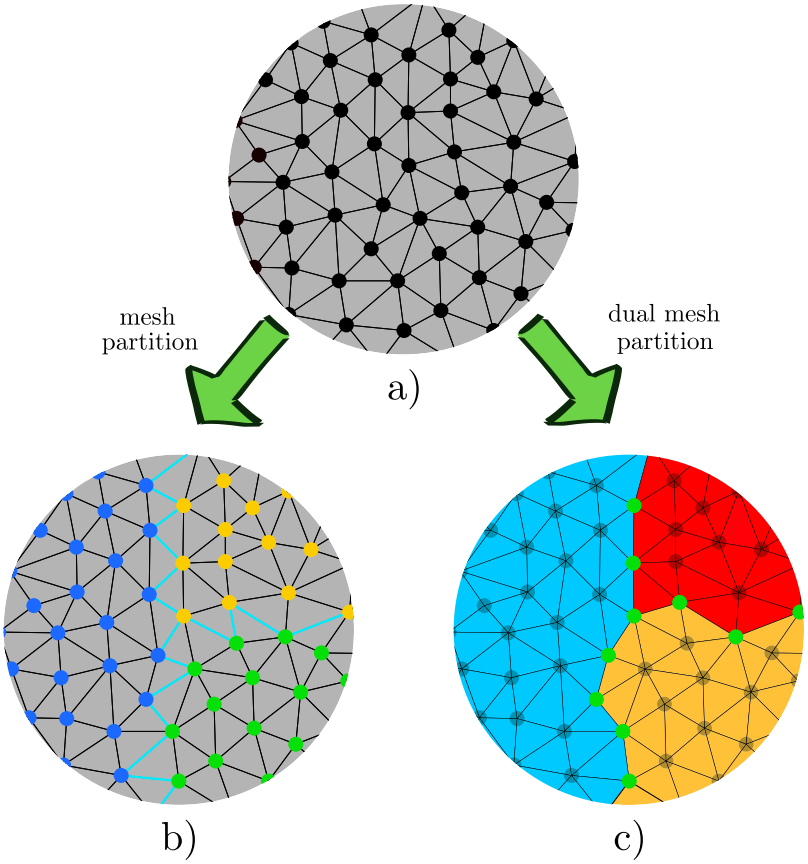

Figure C2. Example of partitioning using the graph and the dual graph. (a) Mesh to partition. (b) Mesh partitioned using the graph, subsets of nodes in color (yellow, blue, and green) represent each node subset, cyan vertices are crossed by the partitions' boundaries. (c) Mesh partitioned using the dual mesh, subsets of elements (red, yellow, cyan) represent each element subset, nodes in green define the boundaries between partitions, these are *shared-nodes*.

Appendix C. Graphs, and graph partitioning

Two ways of partitioning a mesh are classically used: the first is to partition the mesh graph (each node is a vertex in the graph and each edge of the elements is an edge on the graph) to obtain multiple subsets of nodes. Each sub-graph will contain nodes that are not present in any other sub-graph. The second is to make the partitions using the mesh's dual graph (or dual graph of a planar graph[5]) of the initial mesh. In the dual graph, each vertex represents an element of the initial planar graph, and each edge represents an edge of the initial planar graph that is shared by two elements. Coherent Eulerian meshes are planar graphs: in a 2D, the computation of the dual mesh uses each two-simplex (elements) as a vertex and each one-simplex (edges) as an edge, while in 3D each three-simplex (elements) is treated as a vertex

[5] A graph that can be projected into a plane and that its edges intersect only at their nodes.

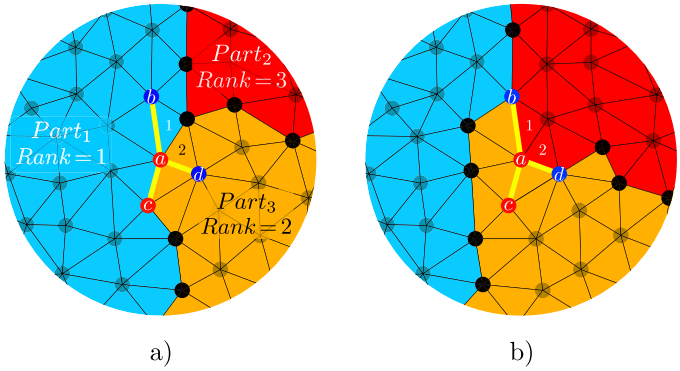

Figure F1. Example of the scattering near a *P*-node *a* the corresponding connections of its coupled *point* are displayed in yellow to *L*-nodes *b* and *d* and to *P*-node *c*. *L*-nodes are blue while *P*-nodes are red. (a) Initial configuration, connections to the *point* of *P*-node *a* are stored in part_1 and part_3 and (b) configuration after the scattering, connections to the *point* of *P*-node *a* are distributed within all parts.

and each two-simplex (facets) as an edge. Partitioning the dual graph of the mesh produces subsets of elements of the initial mesh, each sub-graph will contain a subset of elements not being present in any other sub-graph. Figure C1 shows examples of the dual graph of a mesh in 2D.

Figure C2 illustrates examples of the two ways of partitioning a mesh (figure C2(a)), using the graph (figures C2(b) and (d)) and the dual graph (figures C2(c) and (e)).

Appendix D. Parallel numbering and regularization across partitions

D.1. Non-repeating numbering

The non-repeating numbering is a straightforward process: each geometric type (point, line or surface) will be numbered according to the partition where it is present, starting with the number of the local partition (i.e. if numbering in part_3, the first Line will be numbered as line_3, the first point as point_3 and the first surface as surf_3), subsequently the numbering increases by the number of total partitions (i.e. if numbering in part_3 and the total number of partitions is 4 the second Line will be numbered as line_7).

D.2. Regularization algorithm

Algorithm 4 contextualizes this procedure with the help of a recursive function presented in algorithm 5. Note that the lines 4 and 10: of algorithm 4 send information to other partitions. These are performed using predefined functions of MPI (MPI_Alltoallv and MPI_AllGatherv). The purpose of the call of algorithm 5 in line 14: is to fill the set *SameEntity*. This list of sets stores the identities given to each geometric entity crossed by partition boundaries. In the configuration given in figure 1(d), when algorithm 4 is performed over the surfaces, *SameEntity* would store two lists, one for each iteration of the while loop of line 11: in the first iteration the list would be composed of three pairs: $\{\{\text{surf}_1, \text{part}_1\}, \{\text{surf}_5, \text{part}_2\}, \{\text{surf}_3, \text{part}_3\}\}$ and the second iteration of four pairs: $\{\{\text{surf}_4, \text{part}_1\}, \{\text{surf}_8, \text{part}_2\}, \{\text{surf}_2, \text{part}_2\}, \{\text{surf}_6, \text{part}_3\}\}$. On each iteration, the entity will be named on all partitions with the lowest identity found for it. In

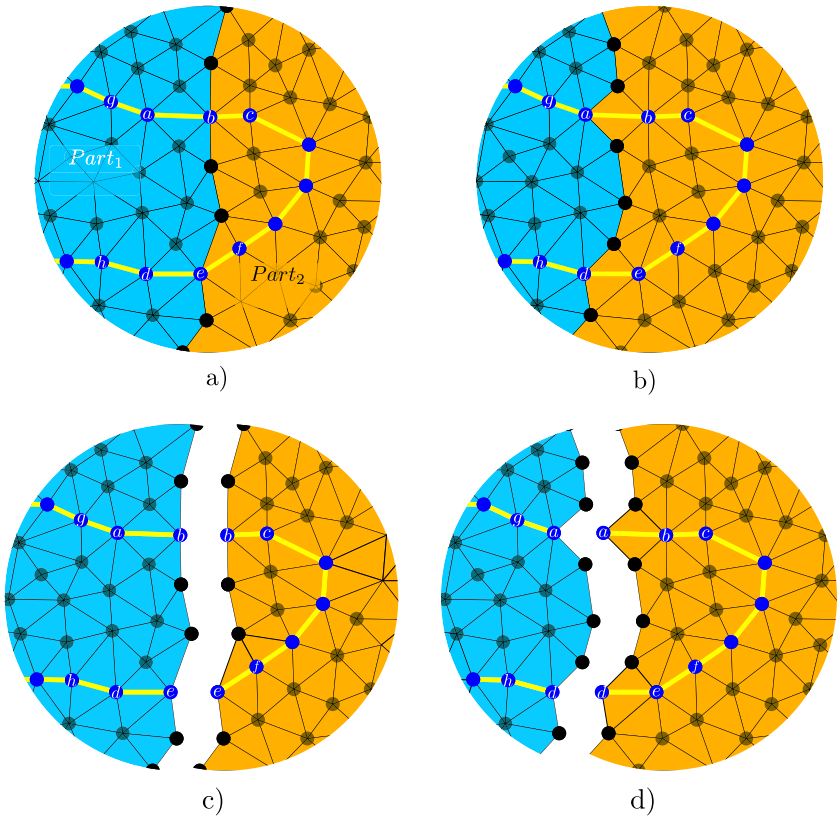

Figure F2. Example of the scattering with a line (yellow) crossed by the boundary of two partitions $part_1$ and $part_2$, the assembled and separated views are displayed, the assembled view shows the domain being simulated while the separated view shows the actual memory of each partition. (a) and (c) Initial configuration, assembled and separated views respectively, (b) and (d) configuration after scattering, assembled and separated views, respectively.

our example, $surf_5$ and $surf_3$ will be named $surf_1$ in their respective partitions, and $surf_4$, $surf_8$, and $surf_6$ will be named $surf_2$ in their respective partitions.

Appendix E. Unidirectional element selection algorithm

In the first section of algorithm 6 (lines 1 to 10), two lists are created and filled. These lists contain the share-nodes arranged by partitions and the elements to send to each partition. A first element filter is applied via the inequality of line 7. The list of elements to export only accepts elements that share at least one node with a partition of superior priority (a partition with a lower number of elements). In the last section of the algorithm (lines 11 to 16), a final filter is applied to subtract the elements that appear to be sent to multiple destinations (i.e. elements with nodes shared by more than two partitions). The final destination for these elements is defined, once again, by choosing the partition with the highest priority (line 15). Once this algorithm is executed by all partitions, a *scattering* process begins, sending each element to its destination along with some associated information: node positions, node fields, element

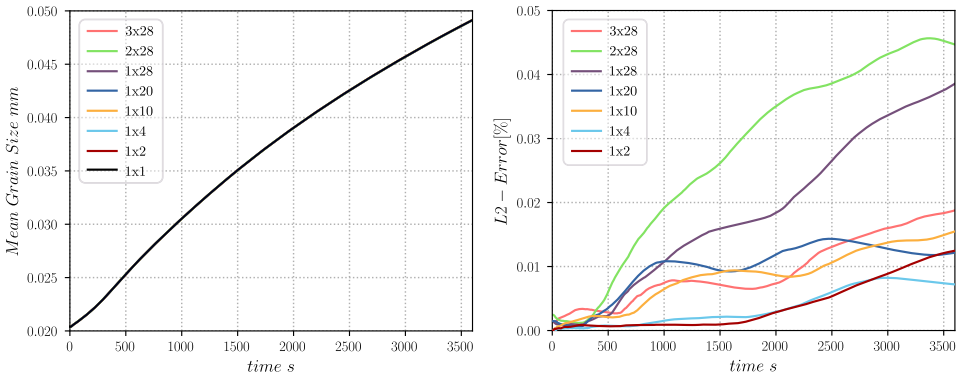

Figure G1. Results of the test with a surface of 50 mm^2 performed with 1 to 84 (3×28) cores. (Left) Mean grain size evolution, (right) L2-error of the mean grain size evolution with the test performed in sequential as a reference.

fields, and some data necessary to fill the data structure of the geometric entities involved in the scattering (see section appendix F).

Appendix F. Reconstruction of geometric entities

When exchanging elements and nodes between partitions, a reconstruction of geometrical entities involving those elements and nodes must be considered. These reconstructions must be addressed in function of the type of entity to reconstruct (*point*, *line*, or *surface*). While reconstructing *surfaces* is trivial (they move along with elements, [66]), the reconstruction of *lines* and *points* is more complex.

F.1. Point reconstruction

Received *P*-nodes need to build *points*. As such, information regarding the connections of *points* to other *points* and *lines* is required (see the data structure of *points* of section 2.1 of [66]). In some cases, this information needs to be gathered from multiple partitions as illustrated in figure F1. Here the scattering procedure is performed near a *point* attached to *P*-node *a*. This *point* exists in the memory of part_1 and part_3 (hence *P*-node *a* is a *shared-node*). The connections of *L*-nodes *a* and *b* are only known by part_1 and part_3 while both partitions know the connection to *P*-node *c*. Note that part_2 does not have any information regarding the connections of *P*-node *a*. During the scattering, elements 1 and 2 are sent to part_2 by Part_1 and Part_3 respectively, also, both partitions need to send the nodes unknown by part_2. Part_1 sends the node *a* and *b* while part_3 sends the node *a* and *d* (both partitions sent information of node *a* because they have no way to know that the other partition has already sent it). The information available on each partition regarding the connections of the *point* of *P*-node *a* is attached to the communication. Part_1 sends the *P*-node *a* along with the identities *c* and *b*, and part_3 sends the *P*-node *a* along with the identities *c* and *d*. Part_2 receives two times the *P*-node *a*, however, it is only created once. Then, the information received about its connections is used: part_2 searches on its memory for the existence of nodes *b*, *c*, and *d*. If they exist (each separately), the connection is stored in the data structure. Note that *P*-node *c* is not found by part_2; hence that connection is not created.

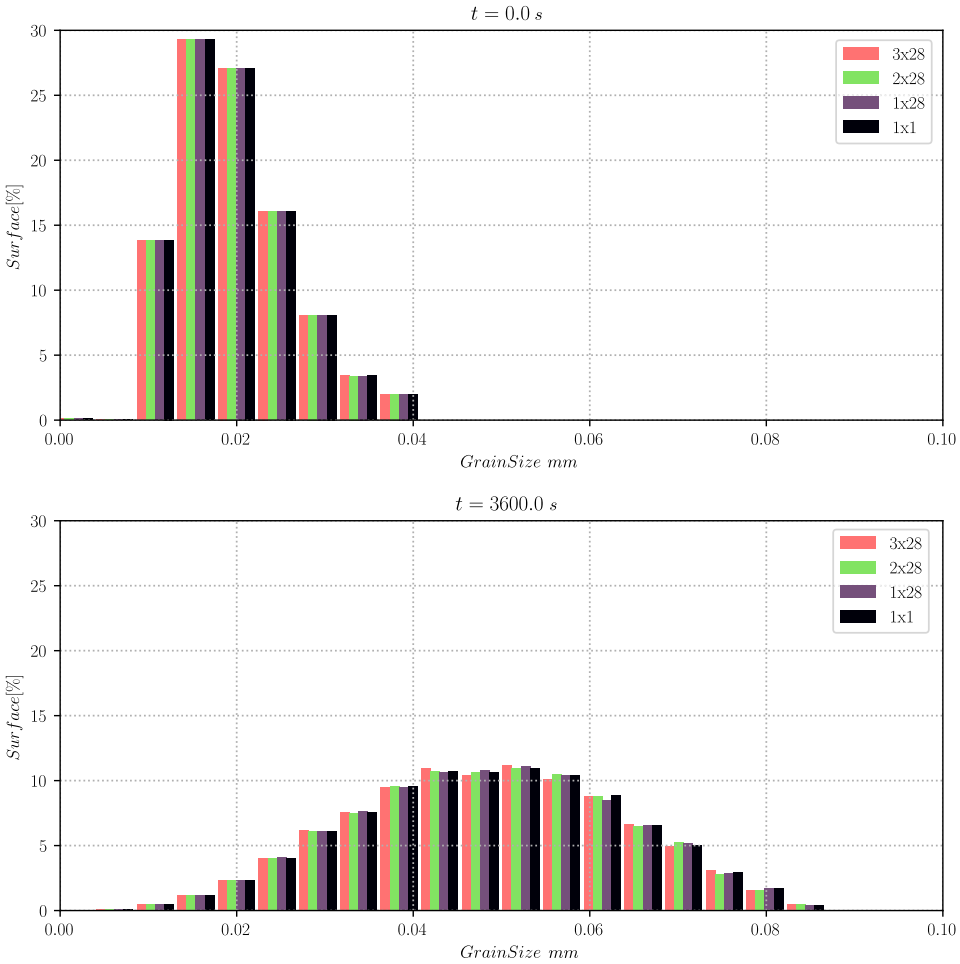

Figure G2. Grain size distributions for the test with a surface of 50 mm^2 performed in 1, 28, 56 (2 $\times$ 28), and 84 (3 $\times$ 28) cores. Initial state (top), state at 3600 s (bottom).

F.2. Line reconstruction

Line reconstructions are performed similarly to *point* reconstructions. The data structure of *lines* is particular as it is composed of an ordered sequence of *L*-nodes and optional initial and final *points* (section 2.1 of [66]). Here, an additional property is included in the data structure of *L*-nodes: the *previous* and *next* nodes within the same *line*. This additional property helps to reconstruct *lines* similarly to point connections help to reconstruct *points*. The *L*-node position within a *line* is obtained by its relative position to its next and previous nodes. If one of these nodes does not appear in memory, it means that the *line* is segmented at that *L*-node.

Figure F2 shows the initial and final states of a scattering between two partitions where a *line* has been crossed by the partition boundary. During the scattering, two *L*-nodes are sent from part$_1$ to part$_2$. Node *a* carries the identity of its previous (*L*-node *g*) and next (*L*-node *b*) nodes, and node *d* carries the identity of its previous (*L*-node *e*) and next (*L*-node *h*) nodes. Nodes *a* and *d* are then placed in the line of part$_2$ before *L*-node *b* and after *L*-node *e*. Note that even though the information about the previous node of *L*-node *a* was sent, it is not used

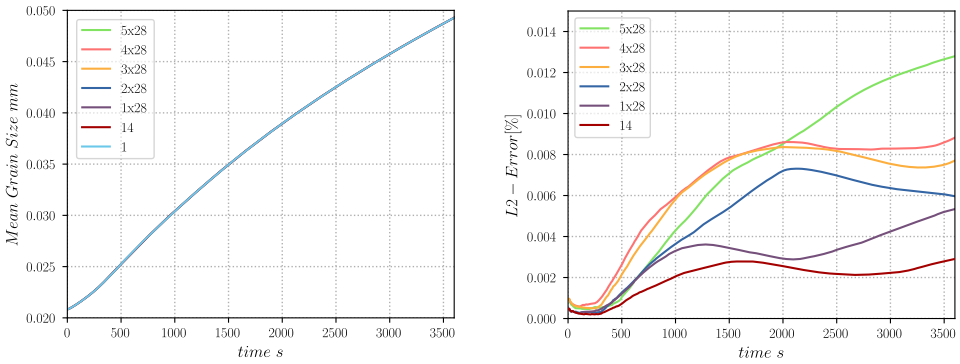

Figure H1. Results of the case with a surface of 560 mm^2 performed with 1 to 140 (5×28) cores. (Left) Mean grain size evolution, (right) L2-error of the mean grain size evolution (reference is the sequential computation).

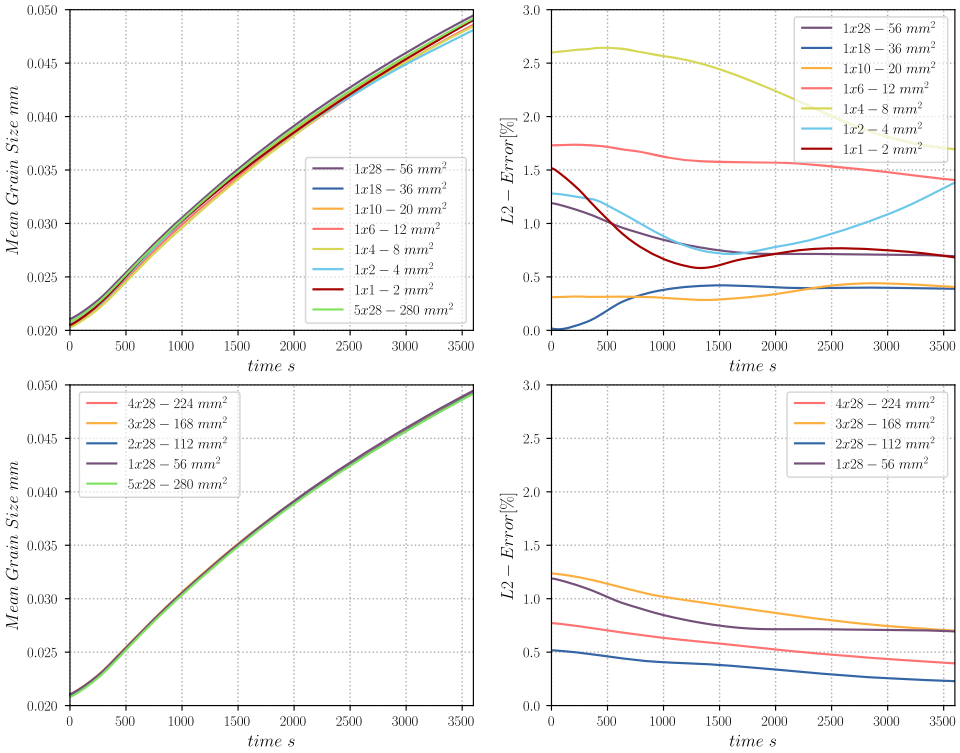

Figure I1. Mean size of the case with a DSPC of 2 mm^2. Simulations using, (top) one node (28 cores max.), (bottom) multiple nodes. The L2-error reference is the largest simulation (140 (5×28) cores).

because node g does not appear in the memory of part$_2$ (see figutr F2(d)). Consequently, the previous node of L-node a is set to null in part$_2$.

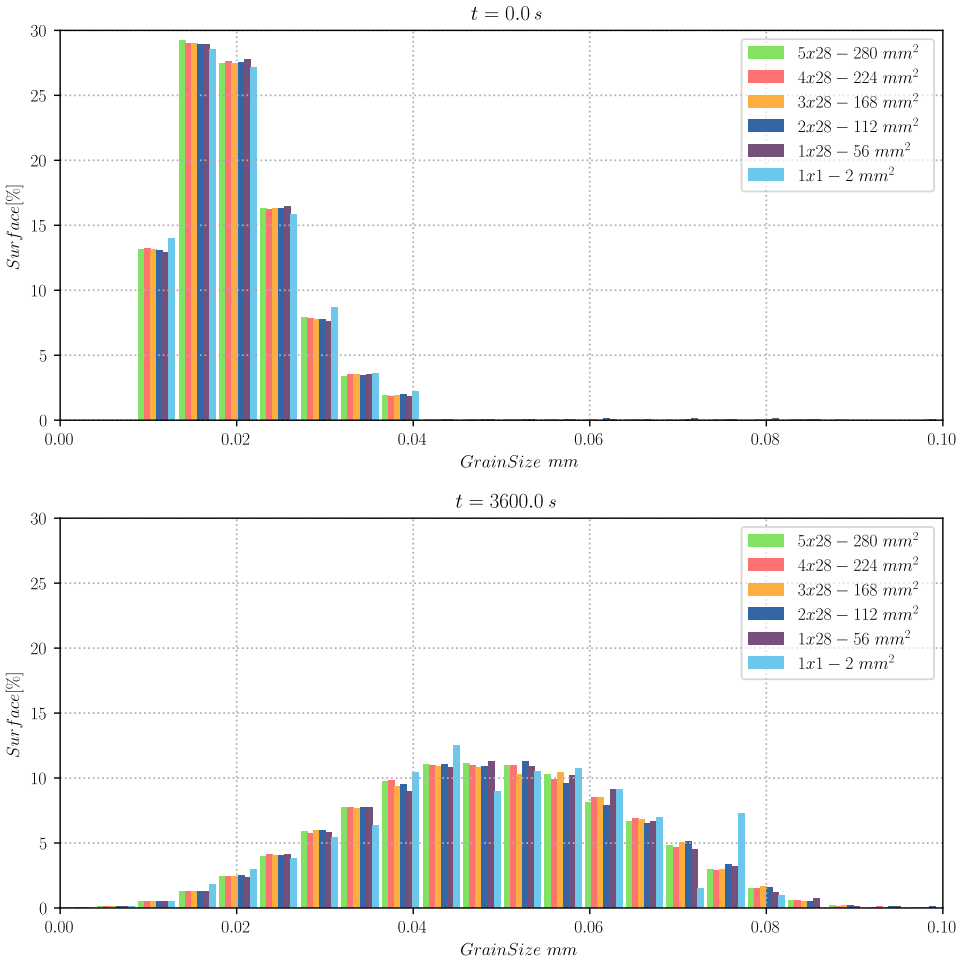

Figure I2. Grain size distributions for the test with a DSPC of 2 mm^2 performed with 1 to 140 (5 × 28) cores. Initial state (top), distributions after 3600 s (bottom).

Appendix G. Strong scaling, results in a 50 mm^2-RVE

(See figures G1 and G2).

Appendix H. Strong scaling, results in a 560 mm^2-RVE

(See figure H1).

Appendix I. Weak scaling results

(See figures I1–I3).

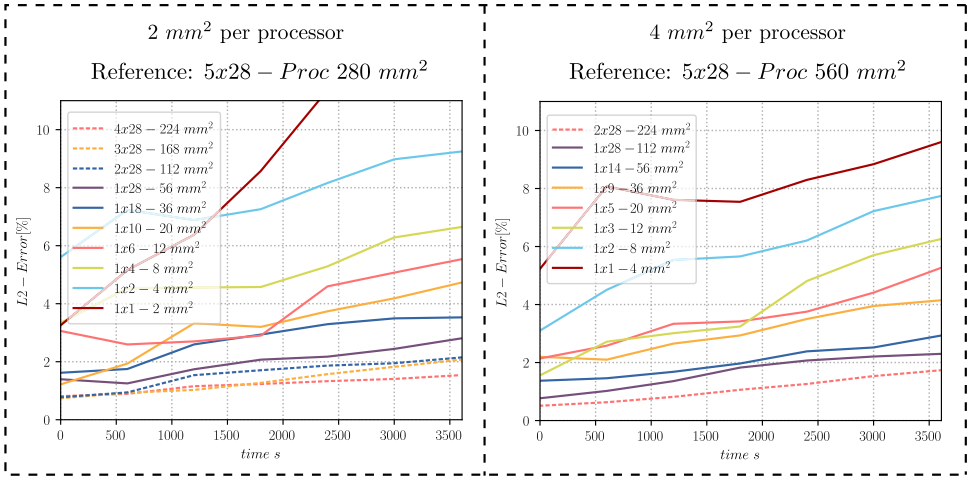

Figure I3. L2-error of the grain size distribution for the test with a DSPC of 2 mm^2 (left) and a DSPC of 4 mm^2 (right) compared to the largest simulation of each context (140 (5×28) cores) (280 and 560 mm^2 respectively).

ORCID iDs

Sebastian Florez https://orcid.org/0000-0002-5962-7700
Julien Fausty https://orcid.org/0000-0001-9911-1859
Karen Alvarado https://orcid.org/0000-0003-3081-7292
Brayan Murgas https://orcid.org/0000-0002-6513-7505
Marc Bernacki https://orcid.org/0000-0002-6677-2850

References

[1] Anderson M P, Srolovitz D J, Grest G S and Sahni P S 1984 Computer simulation of grain growth-I. Kinetics *Acta Metall.* **32** 783–91
[2] Srolovitz D J, Anderson M P, Grest G S and Sahni P S 1984 Computer simulation of grain growth-III. Influence of a particle dispersion *Acta Metall.* **32** 1429–38
[3] Hesselbarth H W and Göbel I R 1991 Simulation of recrystallization by cellular automata *Acta Metall. Mater.* **39** 2135–43
[4] Kawasaki K, Nagai T and Nakashima K 1989 Vertex models for two-dimensional grain growth *Phil. Mag.* B **60** 399–421
[5] Brakke K A 1992 The surface evolver *Exp. Math.* **1** 141–65
[6] Osher S and Sethian J A 1988 Fronts propagating with curvature-dependent speed: algorithms based on Hamilton–Jacobi formulations *J. Comput. Phys.* **79** 12–49
[7] Merriman B, Bence J K and Osher S J 1994 Motion of multiple junctions: a level set approach *J. Comput. Phys.* **112** 334–63
[8] Chen L-Q and Yang W 1994 Computer simulation of the domain dynamics of a quenched system with a large number of nonconserved order parameters: the grain-growth kinetics *Phys. Rev.* B **50** 15752–6
[9] Steinbach I, Pezzolla F, Nestler B, Seeßelberg M, Prieler R, Schmitz G J and Rezende J L L 1996 A phase field concept for multiphase systems *Physica* D **94** 135–47
[10] Bernacki M, Chastel Y, Coupez T and Logé R E 2008 Level set framework for the numerical modelling of primary recrystallization in polycrystalline materials *Scr. Mater.* **58** 1129–32

[11] Cruz-Fabiano A L, Logé R and Bernacki M 2014 Assessment of simplified 2D grain growth models from numerical experiments based on a level set framework *Comput. Mater. Sci.* **92** 305–12
[12] Maire L, Scholtes B, Moussa C, Bozzolo N, Pino Muñoz D and Bernacki M 2016 Improvement of 3D mean field models for capillarity-driven grain growth based on full field simulations *J. Mater. Sci.* **51** 10970–81
[13] Furstoss J, Bernacki M, Ganino C, Petit C and Pino-Muñoz D 2018 2D and 3D simulation of grain growth in olivine aggregates using a full field model based on the level set method *Phys. Earth Planet. Inter.* **283** 98–109
[14] Bernacki M, Resk H, Coupez T and Logé R E 2009 Finite element model of primary recrystallization in polycrystalline aggregates using a level set framework *Modelling Simul. Mater. Sci. Eng.* **17** 64006
[15] Bernacki M, Logé R E and Coupez T 2011 Level set framework for the finite-element modelling of recrystallization and grain growth in polycrystalline materials *Scr. Mater.* **64** 525–8
[16] Scholtes B, Boulais-Sinou R, Settefrati A, Pino Muñoz D, Poitrault I, Montouchet A, Bozzolo N and Bernacki M 2016 3D level set modeling of static recrystallization considering stored energy fields *Comput. Mater. Sci.* **122** 57–71
[17] Maire L, Scholtes B, Moussa C, Bozzolo N, Muñoz D P, Settefrati A and Bernacki M 2017 Modeling of dynamic and post-dynamic recrystallization by coupling a full field approach to phenomenological laws *Mater. Des.* **133** 498–519
[18] Weygand D, Bréchet Y and Lépinoux J 1999 Zener pinning and grain growth: a two-dimensional vertex computer simulation *Acta Mater.* **47** 961–70
[19] Couturier G, Maurice C and Fortunier R 2003 Three-dimensional finite-element simulation of Zener pinning dynamics *Phil. Mag.* **83** 3387–405
[20] Couturier G, Maurice C, Fortunier R, Doherty R and Driver J H 2004 Finite element simulations of 3D Zener pinning *Mater. Sci. Forum* **467–470** 1009–18
[21] Couturier G, Doherty R, Maurice C and Fortunier R 2005 3D finite element simulation of the inhibition of normal grain growth by particles *Acta Mater.* **53** 977–89
[22] Agnoli A, Bernacki M, Logé R, Franchet J-M, Laigo J and Bozzolo N 2015 Selective growth of low stored energy grains during δ sub-solvus annealing in the Inconel 718 nickel-based superalloy *Metall. Mater. Trans.* A **46** 4405–21
[23] Alvarado K, Florez S, Flipon B, Bozzolo N and Bernacki M 2021 A level set approach to simulate grain growth with an evolving population of second phase particles *Modelling Simul. Mater. Sci. Eng.* **29** 035009
[24] Laucoin É and Calvin C 1996 A parallel front-tracking method for two-phase flow simulations *Parallel Computational Fluid Dynamics* vol 2004 (Amsterdam: Elsevier) pp 289–96
[25] da Silveira Neto A, Villar M, Roma A, Serfaty R, van Wachem B and Pivello M 2016 A parallel front-tracking algorithm for the simulation of rising bubbles *9th Int. Conf. on Multiphase Flow (ICMF-2016)* (February1–6 2016)
[26] Pan K-L and Yin G-C 2012 Parallel strategies of front-tracking method for simulation of multiphase flows *Comput. Fluids* **67** 123–9
[27] Collins J B and Levine H 1985 Diffuse interface model of diffusion-limited crystal growth *Phys. Rev.* B **31** 6119–22
[28] Langer J S 1986 Models of pattern formation in first-order phase transitions 165–86
[29] Srolovitz D J, Grest G S, Anderson M P and Rollett A D 1988 Computer simulation of recrystallization-II. Heterogeneous nucleation and growth *Acta Metall.* **36** 2115–28
[30] Rollett A D, Luton M J and Srolovitz D J 1992 Microstructural simulation of dynamic recrystallization *Acta Metall. Mater.* **40** 43–55
[31] Peczak P and Luton M J 1993 A Monte Carlo study of the influence of dynamic recovery on dynamic recrystallization *Acta Metall. Mater.* **41** 59–71
[32] Peczak P 1995 A Monte Carlo study of influence of deformation temperature on dynamic recrystallization *Acta Metall. Mater.* **43** 1279–91
[33] Radhakrishnan B and Zacharia T 1995 Simulation of curvature-driven grain growth by using a modified Monte Carlo algorithm *Metall. Mater. Trans.* A **26** 167–80
[34] Pezzee C F and Dunand D C 1994 The impingement effect of an inert, immobile second phase on the recrystallization of a matrix *Acta Metall. Mater.* **42** 1509–24
[35] Liu Y, Baudin T and Penelle R 1996 Simulation of normal grain growth by cellular automata *Scr. Mater.* **34** 1679–83

[36] Raabe D 1998 Disrete mesoscale simulation of recrystallization microstructure and texture using a stochastic cellular automation approach *Materials Science Forum* **273-275** 169–74
[37] Raabe D 1999 Introduction of a scalable three-dimensional cellular automaton with a probabilistic switching rule for the discrete mesoscale simulation of recrystallization phenomena *Phil. Mag.* A **79** 2339–58
[38] Rollett A D and Raabe D 2001 A hybrid model for mesoscopic simulation of recrystallization *Comput. Mater. Sci.* **21** 69–78
[39] Lin Y C, Liu Y-X, Chen M-S, Huang M-H, Ma X and Long Z-L 2016 Study of static recrystallization behavior in hot deformed Ni-based superalloy using cellular automaton model *Mater. Des.* **99** 107–14
[40] Hallberg H, Wallin M and Ristinmaa M 2010 Simulation of discontinuous dynamic recrystallization in pure Cu using a probabilistic cellular automaton *Comput. Mater. Sci.* **49** 25–34
[41] Rauch L, Madej L, Spytkowski P and Golab R 2015 Development of the cellular automata framework dedicated for metallic materials microstructure evolution models *Arch. Civ. Mech. Eng.* **15** 48–61
[42] Li H, Sun X and Yang H 2016 A three-dimensional cellular automata-crystal plasticity finite element model for predicting the multiscale interaction among heterogeneous deformation, DRX microstructural evolution and mechanical responses in titanium alloys *Int. J. Plast.* **87** 154–80
[43] Madej L, Sitko M, Legwand A, Perzynski K and Michalik K 2018 Development and evaluation of data transfer protocols in the fully coupled random cellular automata finite element model of dynamic recrystallization *J. Comput. Sci.* **26** 66–77
[44] Logé R, Bernacki M, Resk H, Delannay L, Digonnet H, Chastel Y and Coupez T 2008 Linking plastic deformation to recrystallization in metals using digital microstructures *Phil. Mag.* **88** 3691–712
[45] Florez S, Shakoor M, Toulorge T and Bernacki M 2020 A new finite element strategy to simulate microstructural evolutions *Comput. Mater. Sci.* **172** 109335
[46] Mießen C, Velinov N, Gottstein G and Barrales-Mora L A 2017 A highly efficient 3D level-set grain growth algorithm tailored for ccNUMA architecture *Modelling Simul. Mater. Sci. Eng.* **25** 084002
[47] Elsey M, Esedoğlu S and Smereka P 2009 Diffusion generated motion for grain growth in two and three dimensions *J. Comput. Phys.* **228** 8015–33
[48] Elsey M, Esedoğlu S and Smereka P 2013 Simulations of anisotropic grain growth: efficient algorithms and misorientation distributions *Acta Mater.* **61** 2033–43
[49] Fan D and Chen L-Q 1997 Diffuse-interface description of grain boundary motion *Phil. Mag. Lett.* **75** 187–96
[50] Krill C E and Chen L Q 2002 Computer simulation of 3D grain growth using a phase-field model *Acta Mater.* **50** 3057–73
[51] Kazaryan A, Wang Y, Dregia S A and Patton B R 2001 Grain growth in systems with anisotropic boundary mobility: analytical model and computer simulation *Phys. Rev.* B **63** 184102
[52] Moelans N, Blanpain B and Wollants P 2008 Quantitative analysis of grain boundary properties in a generalized phase field model for grain growth in anisotropic systems *Phys. Rev.* B **78** 024113
[53] Moelans N, Godfrey A, Zhang Y and Juul Jensen D 2013 Phase-field simulation study of the migration of recrystallization boundaries *Phys. Rev.* B **88** 1–10
[54] Chen L *et al* 2015 An integrated fast Fourier transform-based phase-field and crystal plasticity approach to model recrystallization of three dimensional polycrystals *Comput. Methods Appl. Mech. Eng.* **285** 829–48
[55] Chang K, Chen L-Q, Krill C E and Moelans N 2017 Effect of strong nonuniformity in grain boundary energy on 3D grain growth behavior: a phase-field simulation study *Comput. Mater. Sci.* **127** 67–77
[56] Garcke H, Nestler B and Stoth B 1999 A multiphase field concept: numerical simulations of moving phase boundaries and multiple junctions *SIAM J. Appl. Math.* **60** 295–315
[57] Miyoshi E and Takaki T 2017 Multi-phase-field study of the effects of anisotropic grain-boundary properties on polycrystalline grain growth *J. Cryst. Growth* **474** 160–5
[58] Takaki T, Yoshimoto C, Yamanaka A and Tomita Y 2014 Multiscale modeling of hot-working with dynamic recrystallization by coupling microstructure evolution and macroscopic mechanical behavior *Int. J. Plast.* **52** 105–16

[59] Soares A, Ferro A C and Fortes M A 1985 Computer simulation of grain growth in a bidimensional polycrystal *Scr. Metall.* **19** 1491–6
[60] Weygand D, Bréchet Y and Lépinoux J 1998 A vertex dynamics simulation of grain growth in two dimensions *Phil. Mag.* B **78** 329–52
[61] Lépinoux J, Weygand D and Verdier M 2010 Modeling grain growth and related phenomena with vertex dynamics *C. R. Phys.* **11** 265–73
[62] Barrales Mora L A 2010 2D vertex modeling for the simulation of grain growth and related phenomena *Math. Comput. Simul.* **80** 1411–27
[63] Mellbin Y, Hallberg H and Ristinmaa M 2015 A combined crystal plasticity and graph-based vertex model of dynamic recrystallization at large deformations *Modelling Simul. Mater. Sci. Eng.* **23** 045011
[64] Frost H J, Thompson C V, Howe C L and Whang J 1988 A two-dimensional computer simulation of capillarity-driven grain growth: preliminary results *Scr. Metall.* **22** 65–70
[65] Becker J K, Bons P D and Jessell M W 2008 A new front-tracking method to model anisotropic grain and phase boundary motion in rocks *Comput. Geosci.* **34** 201–12
[66] Florez S, Alvarado K, Muñoz D P and Bernacki M 2020 A novel highly efficient Lagrangian model for massively multidomain simulation applied to microstructural evolutions *Comput. Methods Appl. Mech. Eng.* **367** 113107
[67] Walker D W 1992 Standards for message-passing in a distributed memory environment *Technical Report* Center for Research on Parallel Computing (CRPC), Oak Ridge National Lab. TN (United States)
[68] Sussman D M 2017 cellGPU: massively parallel simulations of dynamic vertex models *Comput. Phys. Commun.* **219** 400–6
[69] Madhikar P, Åström J, Westerholm J and Karttunen M 2018 CellSim3D: GPU accelerated software for simulations of cellular growth and division in three dimensions *Comput. Phys. Commun.* **232** 206–13
[70] Karypis G and Kumar V 1998 A fast and high quality multilevel scheme for partitioning irregular graphs *SIAM J. Sci. Comput.* **20** 359–92
[71] de Boor C 1978 *A Practical Guide to Splines* (*Applied Mathematical Sciences* vol 27) (New York: Springer)
[72] Sutoki T 1928 On the mechanism of crystal growth by annealing *Scientific Reports of Tohoku* Imperial University vol 17 pp 857–76
[73] Carpenter H and Elam C F 1920 Crystal growth and recrystallization in metals *J. Inst. Met.* **24** 83–131
[74] Harker D and Parker E R 1945 Grain shape and grain growth *Trans. Am. Soc. Met.* **34** 156–201
[75] Burke J E 1949 Some factors affecting the rate of grain growth in metals *AIMS Trans.* **180** 73–91
[76] Burke J E and Turnbull D 1952 Recrystallization and grain growth *Prog. Met. Phys.* **3** 220–92
[77] Smith C S 1948 Grains, phases, and interfaces: an introduction of microstructure *Trans. Metall. Soc. AIME* **175** 15–51
[78] Pellegrini F and Roman J 1996 Scotch: a software package for static mapping by dual recursive bipartitioning of process and architecture graphs *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* vol 1067 pp 493–8
[79] Devine K, Hendrickson B, Boman E, John M S and Vaughan C 2000 Design of dynamic load-balancing tools for parallel applications *Proc. of the Int. Conf. Supercomputing* pp 110–8
[80] Karypis G and Kumar V 1996 Parallel multilevel graph partitioning *Proc. Int. Conf. Parallel Processing* (IEEE Comput. Soc. Press) pp 314–9
[81] Mohanamuraly P, Hascoët L and Müller J-D 2020 Seeding and adjoining zero-halo partitioned parallel scientific codes *Optim. Methods Software* **35** 618–37
[82] Coupez T, Digonnet H and Ducloux R 2000 Parallel meshing and remeshing *Appl. Math. Modelling* **25** 153–75
[83] Mesri Y, Digonnet H and Coupez T 2009 Advanced parallel computing in material forming with CIMLib *Eur. J. Comput. Mech.* **18** 669–94
[84] Karypis G and Kumar V 1998 Multilevel k-way partitioning scheme for irregular graphs *J. Parallel Distrib. Comput.* **48** 96–129
[85] Imai H, Iri M and Murota K 1985 Voronoi diagram in the Laguerre geometry and its applications *SIAM J. Comput.* **14** 93–105

[86] Hitti K, Laure P, Coupez T, Silva L and Bernacki M 2012 Precise generation of complex statistical representative volume elements (RVEs) in a finite element context *Comput. Mater. Sci.* **61** 224–38
[87] Ilin D N and Bernacki M 2016 Advancing layer algorithm of dense ellipse packing for generating statistically equivalent polygonal structures *Granul. Matter* **18** 43
[88] Shakoor M, Bernacki M and Bouchard P-O 2015 A new body-fitted immersed volume method for the modeling of ductile fracture at the microscale: analysis of void clusters and stress state effects on coalescence *Eng. Fract. Mech.* **147** 398–417