Parallel Computing with the Thick Level Set Method

Mororó, L. A. T.; van der Meer, F. P.

DOI
10.1137/21M1400742

Publication date
2021

Document Version
Final published version

Published in
SIAM Journal on Scientific Computing

SIAM J. SCI. COMPUT.
Vol. 43, No. 6, pp. C386–C410

PARALLEL COMPUTING WITH THE THICK LEVEL SET METHOD*

L. A. T. MORORÓ† AND F. P. VAN DER MEER‡

Abstract. The thick level set (TLS) method has been proposed as a nonlocal damage model for the modeling of failure in solids being able to deal with crack initiation, branching, and merging. The nonlocality of the TLS is achieved by introducing a characteristic length into the problem. This way, the TLS does not suffer from spurious localization in the strain field. This paper introduces a domain decomposition method to obtain a parallel implementation of the TLS method. It describes how to handle the numerical features specific to the TLS analysis steps involving level set update, equilibrium solution, and damage front advance. For each of these tasks an appropriate parallel strategy is proposed. The most demanding task in terms of computational cost, i.e., solving the linearized system of equations from the equilibrium problem, is performed with a parallel iterative method profiting from the domain decomposition method adopted. A communication strategy to deal with enriched nodes belonging to shared regions of subdomains is provided. Collective communication strategies are also proposed to deal with operations related to the level set update and damage front advance. Numerical experiments demonstrate the accuracy and efficiency of the proposed framework to handle parallel computing with the TLS method.

Key words. thick level set, parallel computing, domain decomposition, fracture mechanics

AMS subject classifications. 74S05, 74R99, 65Y99, 65M55

DOI. 10.1137/21M1400742

1. Introduction.

The thick level set (TLS) method, first introduced by Moës et al. [16], is a damage model that contains a nonlocal damage definition to prevent spurious strain localization. In this method, the location of the damage front that separates the damaged material from the undamaged material is tracked as the zero level set of an auxiliary field whose evolution is dictated by the nonlocal energy release rate of the material. The TLS damage variable depends on the distance to the damage front as evaluated with the signed distance level set field and varies over a thick band of material with a predefined width according to a user-defined damage function. Since its first advent, the TLS has been expanded in order to enhance its numerical implementation for quasi-static loading condition [3, 17], to deal with three-dimensional quasi-static problems [25] and dynamics [17], to couple with cohesive zone models [12], to treat fatigue crack growth [11], to improve the control of damage initiation and representation of free sliding in shear [30], and to couple damage with plasticity [18].

The TLS works with a staggered solution scheme in which displacements and damage are separately computed. From an existing level set field, i.e., a given damage distribution, an equilibrium problem is solved for the displacement field in a standard finite element analysis. After computing the displacements, the configurational force for front movement is evaluated with which the level set field is updated. As a result of

*Submitted to the journal's Software and High-Performance Computing section February 23, 2021; accepted for publication (in revised form) July 20, 2021; published electronically December 2, 2021.

https://doi.org/10.1137/21M1400742

Funding: The work of the first author was supported by the IFCE. The work of the second author was supported by the Dutch Research Council (NWO) under Vidi grant 16464.

†Federal Institute of Education, Science and Technology of Ceará (IFCE), Morada Nova, Brazil (l.a.taumaturgomororo@tudelft.nl, luiz.mororo@ifce.edu.br).

‡Faculty of Civil Engineering and Geosciences, Delft University of Technology, 2600 GA Delft, The Netherlands (f.p.vandermeer@tudelft.nl).

the staggered approach, the TLS provides a robust framework for handling topological events like merging and branching. From an implementation point of view, the global solution scheme of the TLS contains three modules, where each one encapsulates specific tasks: update of the level set field, equilibrium solution, and damage front evolution [11, 30, 18].

Despite its robustness, the TLS can be a time-demanding approach, mainly because of the equilibrium solution phase [22]. The high computational demand comes from the fact that the TLS requires element sizes smaller than the width of the damaged band in order to achieve a desirable accuracy, especially for the computation of nonlocal quantities [3, 11], giving rise to a system of equations with many degrees of freedom (DOFs). This issue can be amplified if the size of the width of the damage band is constrained to be small relative to the geometry of the problem being investigated like, for instance, interlaminar cusp formation in a polymer matrix of composite materials subjected to mixed mode loading condition [31], which takes place in a very narrow area and involves multiple cracks that eventually merge. Additionally, in order to guarantee numerical stability of the level set update, the damage front advance is constrained such that it does not move more than one element length per time step [3, 11, 30, 18]. Therefore, for simulations up to final failure of specimens with long cracks, many time steps are required [30, 18].

Parallel computing may be used to mitigate the computational effort associated with finite element simulations, where many operations (e.g., assembly of matrices and vectors) can be performed simultaneously for different parts of the domain on different cores. To take advantage of the parallel architecture of a machine for solving large systems of equations, numerical techniques for decomposing the original problem into collaborating subproblems are needed. In a finite element context, domain decomposition (DD) methods are used to build parallel frameworks running on different cores [24, 9]. The main idea behind a DD approach is to divide the whole domain into subdomains that can be solved almost independently on different cores. Since the solution on one subdomain is not completely independent from other subdomains, some exchange of data limited to the interface (or to a small overlapping region) between neighboring subdomains is necessary. The first mathematical studies on DD gave rise to a family of Schwarz algorithms based on overlapping domains (e.g., restricted additive Schwarz method [6]). Later, nonoverlapping methods whose interpretation is more mechanically oriented were proposed, such as finite element tearing and interconnecting method [7] and balanced DD [27].

The main premise of this paper is that such strategies can be advantageous to speed up the equilibrium solution stage that constitutes the main computational bottleneck related to the TLS method. However, when applying DD to one task of the TLS, care is required for the remaining analysis tasks (update of the level set field and damage front evolution).

This work seeks to describe how to apply the DD framework to obtain an efficient parallel version of the TLS method. It demonstrates how to handle the TLS-specific analysis phases that spawn multiple operations on different processors, where time-demanding parts of the solution scheme are performed in parallel, while other parts that require global solution strategies are kept sequential. Two slightly different TLS implementations are considered as starting points: firstly, the version by Mororó and van der Meer [18] for ductile fracture and secondly, the version by van der Meer and Sluys [30], which assumes a secant unloading behavior of material. Both of these build on original concepts of the first paper on the TLS method by Moës et al. [16] and the improvements on its implementation by Bernard, Moës, and Chevaugeon [3].

The parallel iterative solver combined with the Schwarz-type DD strategy by Lingen et al. [14] is used.

It is worth mentioning that apart from the two TLS implementations considered here, the proposed parallel framework can easily be extended to other TLS-based methods, such as the interfacial TLS method by Latifi, Vander Meer, and Sluys [11].

The paper is structured as follows. Section 2 is dedicated to present the main features that comprise the TLS method. The objective of section 2 is to be complete and to provide the context for the parallel framework introduced in section 3 without providing motivation for all details of the TLS as presented elsewhere in literature. Numerical examples considering linear elasticity and plasticity are presented in section 4 and used to assess the performance of the proposed framework. Numerical results are presented with focus on the scalability of the framework and accuracy of the parallel solution. Finally, conclusions are presented in section 5.

2. The TLS method. The TLS method makes use of the level set method [26] to track the location of a damage front Γ_0 that separates the undamaged material from a degraded region (see Figure 1). The damage front is defined as coinciding with the zero level set (or the "iso-0") of an auxiliary field $\phi(\mathbf{x})$, the level set field. The level set field ϕ is chosen to be the signed distance function to Γ_0 in which its gradient satisfies the eikonal equation:

$$\| \nabla\phi \| = 1 \quad \text{on} \quad \Omega, \tag{2.1}$$

where Ω is the domain on which ϕ is defined. This equation guarantees that the absolute value of ϕ at a given point $\mathbf{x}$ is the shortest distance between that point and Γ_0. On a discretized finite element domain, the definition of ϕ at $\mathbf{x}$ is determined by interpolating the values of ϕ from nodes to $\mathbf{x}$ using finite element shape functions.

In the TLS, a damage variable is introduced which is defined as a function of ϕ. The damage variable is constrained to follow a given profile in a transition zone with fixed length between undamaged and fully degraded zones as

$$d(\phi) = \begin{cases} 0, & \phi \le 0, \\ q(\phi), & 0 < \phi \le l_c \ , \\ 1, & \phi > l_c, \end{cases} \tag{2.2}$$

where $q(\phi)$ is a function that has the properties of $q(0) = 0$, $q(l_c) = 1$, and $q'(\phi) \ge 0$ on $\phi \in [0, l_c]$. Therefore, the damage variable changes from zero to one as ϕ goes from zero to the critical length l_c over a band bounded by $\Gamma_0 : \phi = 0$ and $\Gamma_c : \phi = l_c$. In this paper, the arctangent profile proposed by Bernard, Moës, and Chevaugeon [3] is used for all numerical simulations

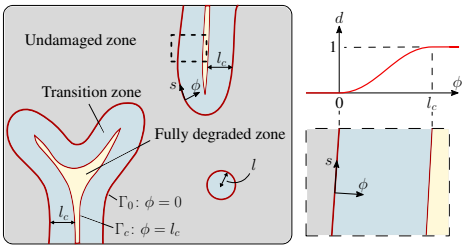

FIG. 1. *The TLS makes use of a single level set function to describe multiple damaged zones; the damage variable d is a function of the level set ϕ.*

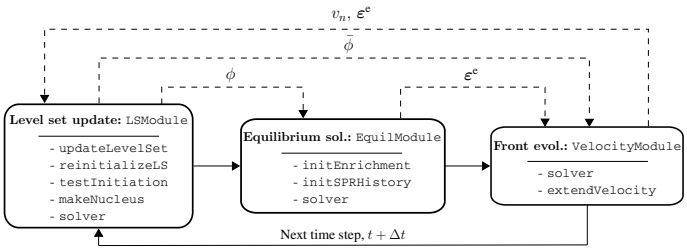

FIG. 2. *The global sequential staggered solution scheme for a single time step. Dashed arrows represent the data exchange among the three modules.*

$$q(\phi) = c_2 \arctan\left(c_1\left(\frac{\phi}{l_c} - c_3\right)\right) + c_4 \tag{2.3}$$

with $c_1 = 10$ and $c_3 = 0.5$ and the other coefficients given by $c_4 = -c_2 \arctan(-c_1 c_3)$ and $c_2 = (\arctan(c_1(1 - c_3)) - \arctan(-c_1 c_3))^{-1}$.

A summary of the staggered algorithm used in this work is given in Figure 2. For every time step, there are three main modules (or *analysis phases*) that are named *level set update*, *equilibrium solution*, and *front evolution*. The level set update phase consists of the update of the level set field and its reinitialization, evaluation of damage initiation given an elastic strain field $\boldsymbol{\varepsilon}^e$, and computation of the size of damaged zones $\bar{\phi}$. In the equilibrium solution phase, for a given damage distribution (which could consist of zero damage throughout Ω), displacements and consequently strains and stresses are computed in a standard finite element analysis. In the front evolution phase, $\bar{\phi}$ and the elastic strain from the equilibrium solution, $\boldsymbol{\varepsilon}^e$, are used to compute the nonlocal configurational force $\bar{Y}$ and the averaged material resistance to damage growth $\bar{Y}_c$, which in turn are used to compute the front velocity v_n along the front that is subsequently extended throughout Ω. With the velocity at hand, the resulting new level set field can be determined and used for the next time step.

The remainder of this section is dedicated to outlining the main operations for each of the three modules. Modular pseudocodes are provided to clarify the implementation. Parts of these pseudocodes are highlighted in blue to indicate modifications related to the parallelism that will be addressed in section 3. In the present section, where the original sequential algorithm is presented, these parts can be ignored. For the sake of compactness, the framework for ductile fracture is firstly presented, and the other one for linear elastic fracture assuming secant unloading behavior is then addressed, inheriting the structure from the former.

2.1. Level set update. The first task in every time step is to define the level set field for that step. In the first time step, the level set field is initialized. In all other time steps it is updated with the nodal normal velocities, as shown in Algorithm 2.1. The level set field at every node belonging to $\mathcal{N}$ (the complete set of nodes in the mesh) is then updated as

$$\phi \leftarrow \phi + v_n \Delta t, \tag{2.4}$$

where Δt is the time increment size. In order to guarantee the numerical stability of the level set update, Δt is constrained as [18, 29]

$$\Delta t = \min\left\{\Delta t^0,\, \alpha_n \frac{h}{\max\{v_n\}}\right\}, \tag{2.5}$$

Algorithm 2.1 The `updateLevelSet` algorithm.

Input: the nodal normal velocities v_n; the default and maximum time increment size Δt^0; and the constant α_n and parameter h
Output: the updated level set field ϕ at nodes and time increment size Δt

$v_{\max} \leftarrow 0$ /* Compute the maximum velocity v_{max} */
for all node $i \in \mathcal{N}$ **do**
  if $v_{ni} > v_{\max}$ **then**
    $v_{\max} \leftarrow v_{ni}$
  end if
end for
Allreduce ($v_{\max}$) /* Get the maximum v_{max} from all processes */
$\Delta t \leftarrow \alpha_n \frac{h}{v_{\max}}$
if $\Delta t^0 < \Delta t$ **then** /* Update the time increment size, if required */
  $\Delta t \leftarrow \Delta t^0$
end if
for all node $i \in \mathcal{N}$ **do** /* Update the level set field */
  $\phi_i \leftarrow \phi_i + v_{ni}\,\Delta t$
end for

in which Δt^0 is the default and maximum time increment size, α_n is a constant defined as $0 < \alpha_n < 1$, h is the characteristic size of the smallest element,[1] and $\max\{v_n\}$ is the largest value of v_n over the entire domain.

In theory, when the level set field is updated with $v_n \Delta t$ and a properly extended velocity field, the updated level set field obtained by (2.4) remains a signed distance function. However, the discrete nature of the level set update deteriorates the property of (2.1). Thus, a reinitialization procedure is periodically performed with a fast marching algorithm [26, 29] in order to keep ϕ as an accurate representation of the signed distance function. Since it is a relatively cheap procedure [30, 18, 29], this reinitialization is performed every time step.

Next, an initiation check is performed at every integration point in a user-defined set of elements $\mathcal{E}_{\text{nucl}}$ where nucleation is allowed, according to Algorithm 2.2. A damage nucleus, i.e., a small region with positive level set values, is inserted when the local driving force for damage growth Y that depends on the elastic strain tensor $\boldsymbol{\varepsilon}^{\text{e}}$ is greater than or equal to the material resistance to damage initiation Y_c^0. If $Y \geq Y_c^0$ is met, a circle with radius ϕ_0 is inserted around the point $\mathbf{x}_{\text{nucl}}$ with the highest ratio Y/Y_c^0. The initiation check is only performed in those elements belonging to $\mathcal{E}_{\text{nucl}}$ that are at least a distance ϕ_{spacing} away from an existing damage front.

After the nucleation check, if a new damage front is added to the problem, the value of ϕ is checked at every node belonging to $\mathcal{N}$ and updated if the node is closer to the new front than to the existing front, as schematically shown in Algorithm 2.3.

The level set update phase is finished with the computation of a measure for the size of enclosed damaged zones. For this purpose, the averaged level set value $\bar{\phi}$ is computed in each time step in a similar way to $\bar{Y}$ in (2.13) by substituting level set values and unknowns $\bar{\phi}$ for Y and $\bar{Y}$, respectively, and by leaving out the weight factor d' in (2.14) and (2.16) [30]. Unlike the computation of $\bar{Y}$, in which only the variation in the normal direction of $\bar{Y}$ is eliminated by means of Lagrange multipliers, the variation of $\bar{\phi}$ along the front is also eliminated by considering a high value for κ [30]. By eliminating the variation of $\bar{\phi}$ in both directions, a single representative value for each individual damaged subdomain is obtained. By adopting the structure of (2.13), the computation of $\bar{\phi}$ does not add much complexity to the framework since its implementation inherits the basic structure from what is already needed for the computation of $\bar{Y}$ and $\bar{Y}_c$.

[1] h is defined as the length of the diagonal of the smallest possible rectangle around an individual element in the mesh.

Algorithm 2.2 The `testInitiation` algorithm.

Input: the parameter ϕ_{spacing}; the elastic strain field; and a function `getIPCoords` to retrieve the coordinates of a integration point j for a given element i
Output: the highest value of $\mathrm{Y}/\mathrm{Y}_{\mathrm{c}}^{0}$ and its corresponding location $\mathbf{x}_{\text{nucl}}$

for all element $i \in \mathcal{E}_{\text{nucl}}$ **do**
  if any node-ϕ-values of element $i > -\phi_{\text{spacing}}$ **then**
    continue
  end if
  $max \leftarrow 0$ /* initialize the maximum value of $\mathrm{Y}/\mathrm{Y}_{\mathrm{c}}^{0}$ */
  for all integration point j **do**
    $\mathbf{x}_j \leftarrow$ `getIPCoords`(i, j) /* Get coordinates of integ. point j */
    $Y_j \leftarrow$ (2.9)
    $Y_{cj}^{0} \leftarrow$ (2.17)
    $ratio \leftarrow \frac{Y_j}{Y_{cj}^{0}}$
    if $ratio > max$ **then** /* Update max and $\mathbf{x}_{\text{nucl}}$, if required */
      $max \leftarrow ratio$
      $\mathbf{x}_{\text{nucl}} \leftarrow \mathbf{x}_j$
    end if
  end for
end for
return max and $\mathbf{x}_{\text{nucl}}$

Algorithm 2.3 The `makeNucleus` algorithm.

Input: the nucleus coordinates $\mathbf{x}_{\text{nucl}}$ and size of new front ϕ_0
Output: the updated level set field ϕ at nodes after nucleation

for all node $i \in \mathcal{N}$ **do**
  $d_i \leftarrow \| \mathbf{x}_i - \mathbf{x}_{\text{nucl}} \|$
  $\phi_n \leftarrow \phi_0 - d_i$
  if $\phi_n > \phi_i$ **then**
    $\phi_i \leftarrow \phi_n$
  end if
end for

Algorithm 2.4 The `LSModule` algorithm.

Input: the nodal normal velocity v_n; the time increment size Δt^0; the constant h; the parameter α_n; the size of new front ϕ_0; the parameter ϕ_{spacing}; and elastic strain field
Output: the insertion of a new damage front and updated level set field ϕ

`updateLevelSet` $(v_n, \Delta t^0, \alpha_n, h)$
`Gather` (ϕ) /* Gather updated $\boldsymbol{\phi}$ to the root process */
`reinitializeLS` $()$
`Scatter` (ϕ) /* Scatter $\boldsymbol{\phi}$ all over the processes */
$Y/Y_c^0, \mathbf{x}_{\text{nucl}} \leftarrow$ `testInitiation` $(\phi_{\text{spacing}}, \boldsymbol{\varepsilon}^{\mathrm{e}})$
`Allreduce` $(Y/Y_c^0, \mathbf{x}_{\text{nucl}})$ /* Get the maximum $\mathrm{Y}/\mathrm{Y}_{\mathrm{c}}^{0}$ and $\mathbf{x}_{\text{nucl}}$ from all processes */
if $Y/Y_c^0 \geq 1$ **then**
  `makeNucleus` $(\mathbf{x}_{\text{nucl}}, \phi_0)$
end if
`Gather` $(\mathbf{K}, \mathbf{L}, \mathbf{f}^{\phi})$ /* Gather matrices and right-hand side vector for (2.13) */
$\bar{\phi} \leftarrow$ `solver` ((2.13))
`Scatter` $(\bar{\phi})$ /* Scatter $\bar{\boldsymbol{\Phi}}$ over the processes */

The level set update phase is summarized in Algorithm 2.4.

2.2. Equilibrium solution. Following [18], the equilibrium solution phase is executed in the framework of elasto-plastic finite element analysis for a given damage distribution assuming small displacements and additive decomposition of the total strain $\boldsymbol{\varepsilon}$ into an elastic part $\boldsymbol{\varepsilon}^{\mathrm{e}}$ and a plastic part $\boldsymbol{\varepsilon}^{\mathrm{p}}$, i.e., $\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{\mathrm{e}} + \boldsymbol{\varepsilon}^{\mathrm{p}}$. Moreover, plasticity is only allowed to evolve in the undamaged material and not in regions where $\phi > 0$.

The equilibrium equation without body force and the relation between the total strain $\boldsymbol{\varepsilon}$ and the displacement field $\mathbf{u}$ in Ω read, respectively, $\nabla \cdot \boldsymbol{\sigma} = \mathbf{0}$ and $\boldsymbol{\varepsilon} = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^{\mathrm{T}})$, in which $\boldsymbol{\sigma}$ is the stress tensor. The displacement field is subjected to Dirichlet boundary conditions $\mathbf{u} = \hat{\mathbf{u}}$ on Ω_u.

Under the hypothesis of decoupling between elasticity damage and plastic hardening, the specific free energy ψ is written as a function of the elastic strain $\boldsymbol{\varepsilon}^{\mathrm{e}}$, damage variable d, and equivalent plastic strain $\varepsilon^{\mathrm{p}}_{\mathrm{eq}}$ and is split as $\psi(\boldsymbol{\varepsilon}^{\mathrm{e}}, d, \varepsilon^{\mathrm{p}}_{\mathrm{eq}}) = \psi^{\mathrm{ed}}(\boldsymbol{\varepsilon}^{\mathrm{e}}, d) + \psi^{\mathrm{p}}(\varepsilon^{\mathrm{p}}_{\mathrm{eq}})$ into a sum of an elastic-damage contribution ψ^{ed} and a contribution due to hardening ψ^{p}.

The following elastic-damage energy density that takes into account stiffness recovery under compression [3] is used in all numerical simulations:

$$\psi^{\mathrm{ed}}(\boldsymbol{\varepsilon}^{\mathrm{e}}, d) = \mu(1 - \alpha_i d)(\varepsilon^{\mathrm{e}}_i)^2 + \frac{\lambda}{2}(1 - \alpha_v d)\mathrm{tr}(\boldsymbol{\varepsilon}^{\mathrm{e}})^2, \tag{2.6}$$

where λ and μ are Lamé's elastic constants, $\varepsilon^{\mathrm{e}}_i$ is the principal strain values, $\mathrm{tr}(\boldsymbol{\varepsilon}^{\mathrm{e}})$ is the trace of the elastic strain tensor,

$$\alpha_i = \begin{cases} 1, & \varepsilon^{\mathrm{e}}_i > 0, \\ 0, & \varepsilon^{\mathrm{e}}_i < 0, \end{cases} \quad \text{and} \quad \alpha_v = \begin{cases} 1, & \mathrm{tr}(\boldsymbol{\varepsilon}^{\mathrm{e}}) > 0, \\ 0, & \mathrm{tr}(\boldsymbol{\varepsilon}^{\mathrm{e}}) < 0. \end{cases} \tag{2.7}$$

With (2.6), the constitutive relation in principal stress space and the local driving force for damage growth can, respectively, be expressed as

$$\sigma_i = \frac{\partial \psi^{\mathrm{ed}}}{\partial \varepsilon^{\mathrm{e}}_i} = 2\mu(1 - \alpha_i d)\varepsilon^{\mathrm{e}}_i + \lambda(1 - \alpha_v d)\mathrm{tr}(\boldsymbol{\varepsilon}^{\mathrm{e}}) \tag{2.8}$$

and

$$Y = \frac{\partial \psi^{\mathrm{ed}}}{\partial d} = -\mu\alpha_i(\varepsilon^{\mathrm{e}}_i)^2 - \frac{\lambda}{2}\alpha_v \mathrm{tr}(\boldsymbol{\varepsilon}^{\mathrm{e}})^2. \tag{2.9}$$

In regions where $\phi \leq 0$, the basic form of the constitutive law for plasticity is

$$\boldsymbol{\sigma} = \mathbf{D}^{\mathrm{e}} : \boldsymbol{\varepsilon}^{\mathrm{e}} = \mathbf{D}^{\mathrm{e}} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^{\mathrm{p}}), \tag{2.10}$$

where $\mathbf{D}^{\mathrm{e}}$ is the elasticity tensor from Hooke's law. The plastic strain $\boldsymbol{\varepsilon}^{\mathrm{p}}$, whose evolution is defined by a plastic flow rule, is computed by using an elastic predictor/return mapping algorithm so that $\boldsymbol{\sigma}$ satisfies the yield criterion $f(\boldsymbol{\sigma}, \varepsilon^{\mathrm{p}}_{\mathrm{eq}}) \leq 0$ [20].

Inserting the constitutive relations in (2.8) and (2.10) into the equilibrium equation results in a nonlinear system of equations that is iteratively solved with the Newton–Raphson method. In order to capture sharp load drops during unstable damage growth, the update of prescribed displacements $\hat{u}$ is performed with the adaptive time step size from (2.5), $\hat{u} \leftarrow \hat{u} + \Delta\hat{u}^0 \frac{\Delta t}{\Delta t^0}$, with $\Delta\hat{u}^0$ being the initial displacement increment [18, 29].

The region with $\phi > l_c$ where $d = 1$ represents stress-free macrocracks in the TLS method. In simulations with regular finite elements, this region needs to be at least one row of elements wide to represent this stress-free state, causing mesh dependency [30]. In order to achieve a stress-free state in elements that are only partially in $\phi > l_c$, the enrichment strategy proposed by Bernard, Moës, and Chevaugeon [3] is employed, which allows for a strain discontinuity across the iso-l_c. In this strategy, extra DOFs are added to a node inside the fully degraded region if the elements in its node support

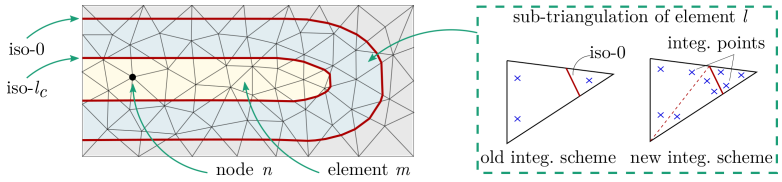

FIG. 3. *On the left, the nodes of element m and node n exemplify, respectively, typical cases of nodes without and with the necessity of being enriched. In the dashed box on the right, the change in integration scheme of the element l crossed by the iso-0 is illustrated (adapted from* [3]*).*

are cut twice by the iso-l_c, as shown in Figure 3. From an implementation point of view, nodes are dynamically enriched and unenriched as the damage front advances.

The enrichment function is constructed as a continuous ramp function across the fully degraded area where $\phi > l_c$. In practice, this scheme divides the element support for an enriched node into two lists of elements: positive and negative. This design will be important when building the parallel framework in section 3.

The numerical integration scheme necessary for accuracy and robustness of the TLS changes in elements that are crossed by the iso-0 and iso-l_c as the front advances from one load increment to another, e.g., the element l illustrated in Figure 3. In the case of crack propagation in elastic-plastic materials, extra care is required with this procedure since history variables that are stored at integration points have to be transferred from old to new integration schemes. In this study, the superconvergent patch recovery (SPR) technique [32, 33] is applied to transfer the plastic strain tensor $\boldsymbol{\varepsilon}^{\mathrm{p}}$ and equivalent plastic strain $\varepsilon^{\mathrm{p}}_{\mathrm{eq}}$, following [18].

In Algorithm 2.5, the equilibrium solution phase is summarized.

Algorithm 2.5 The `EquilModule` algorithm.

Input: the damage distribution $d(\phi)$ and level set field ϕ
Output: the nodal displacements; the new integration scheme; and transferred history variables

```
1: initEnrichment ()
2: commEnrich ()                          /* Enrichment update for receive nodes */
3: initSPRHistory ()
4: û ← û + Δû^0 Δt/Δt^0
5: u, ε^p, ε^p_eq ← solver ()
```

2.3. Front evolution. The last analysis phase concerns computation of the front velocity. It begins with evaluation of the averaged configurational force along the front. Due to the use of the signed distance function, all points sharing the same curvilinear coordinate s are affected when the front at $(0, s)$ (see Figure 1) experiences a front advance.[2] Thus, movement of the front at $(0, s)$ leads to an increase in damage and, consequently, to energy dissipation in all associated points. Therefore, the energy dissipation per unit length as the front moves a unit distance reads

$$g(s) = \int_0^l d'(\phi) Y(\phi, s) \left(1 - \frac{\phi}{\rho(s)}\right) \mathrm{d}\phi, \tag{2.11}$$

where $d'(\phi) = q'(\phi)$ is the spatial derivative of damage with respect to ϕ, l is the size of the damaged zone $l \in (0, l_c]$, and ρ is the curvature of the iso-0. To evaluate $g(s)$

[2] A curvilinear coordinate system (ϕ, s) is introduced here for the derivation, but in the final equations as implemented, the s-coordinates are not used or even constructed.

in a discretized setting, an averaged configurational force $\bar{Y}(s)$ is introduced which is related to $Y(\phi, s)$ through

$$\int_0^l d'(\phi) Y(\phi, s) \left(1 - \frac{\phi}{\rho(s)}\right) \mathrm{d}\phi = \int_0^l d'(\phi) \bar{Y}(s) \left(1 - \frac{\phi}{\rho(s)}\right) \mathrm{d}\phi. \tag{2.12}$$

Numerically, this averaged configurational force is discretized with DOFs on the nodes of those elements that are at least partially inside the damaged domain Ω^{d}. Lagrange multipliers are used to weakly enforce the constraint that $\bar{Y}$ is constant in ϕ direction [3]. The following system of equations is obtained [30]:

$$\begin{bmatrix} \mathbf{K} & \mathbf{L} \\ \mathbf{L} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \bar{\mathbf{Y}} \\ \mathbf{l} \end{Bmatrix} = \begin{Bmatrix} \mathbf{f}^{\mathrm{Y}} \\ \mathbf{0} \end{Bmatrix}, \tag{2.13}$$

in which $\bar{\mathbf{Y}}$ and $\mathbf{l}$ are vectors with $\bar{Y}$ and Lagrange multiplier DOFs, respectively. The matrices and the right-hand side vector are defined as

$$K_{ij} = \int_{\Omega^{\mathrm{d}}} d' N_i N_j + \frac{\kappa h^2}{l_c} \frac{\partial N_i}{\partial x_k} \frac{\partial N_j}{\partial x_k} \mathrm{d}\Omega, \tag{2.14}$$

$$L_{ij} = \int_{\Omega^{\mathrm{d}}} l_c \left(\frac{\partial N_i}{\partial x_k} \frac{\partial \phi}{\partial x_k}\right) \left(\frac{\partial N_j}{\partial x_k} \frac{\partial \phi}{\partial x_k}\right) \mathrm{d}\Omega, \quad \text{and} \tag{2.15}$$

$$f_i^{\mathrm{Y}} = \int_{\Omega^{\mathrm{d}}} N_i d' Y \mathrm{d}\Omega, \tag{2.16}$$

where N_i and N_j are the shape functions associated with nodes i and j, κ is a stabilization parameter, h is the same parameter used in (2.5), and Y is the local configurational force which depends on the current elastic strain field through (2.9).

The material resistance to damage growth Y_c is made into a function of the size of the damaged zone in order to take into account independent input parameters for damage initiation and propagation [30]. In this approach, Y_c varies from an initial strength-based value Y_c^0 to an energy-based value Y_c^G as the size of the damaged zone $\bar{\phi}$ increases. For intermediate values of Y_c, the following interpolation is adopted [11]:

$$\log(Y_c) = \log(Y_c^0) + \frac{\bar{\phi} - \bar{\phi}_{\mathrm{init}}}{\bar{\phi}_{\max} - \bar{\phi}_{\mathrm{init}}} \left(\log(Y_c^G) - \log(Y_c^0)\right), \tag{2.17}$$

where $\bar{\phi}_{\mathrm{init}}$ and $l_c/3 \leq \bar{\phi}_{\max} \leq l_c/2$ are, respectively, the initial size of the damaged zone and the size for which the damaged zone is considered a crack. The quantity Y_c^0 is related to the strength of the material, where we follow earlier work in [18]. The parameter Y_c^G related to propagation is given by $Y_c^G = \frac{G_c}{2Al_c}$, with A being the area under the curve $q(\phi)$ given in (2.3) and G_c being the fracture energy [3].

As Y_c is not constant on Ω^{d} due to its dependence on $\bar{\phi}$ (cf. (2.17)) and in order to account for the general case of multiple materials being used in the same mesh, the resistance is also averaged over the width of the damaged band. Therefore, $\bar{Y}_c$ is computed by solving once more the system of equations in (2.13) but then with $\bar{Y}_c$ in the right-hand side vector instead of Y.

Next, the computation of nodal normal velocity v_n is carried out in two steps. Firstly, v_n is computed at the nodes belonging to $\mathcal{N}_0$ (the set of nodes of those elements that contain the front) based on $\bar{Y}$ and $\bar{Y}_c$ as

$$v_n = \frac{1}{\eta}\left\langle \frac{\bar{Y}}{\bar{Y}_c} - 1 \right\rangle_+, \tag{2.18}$$

where η is a parameter that can be interpreted as viscous resistance against crack growth. Brackets are used to denote the positivity condition, which reflects the irreversibility of crack growth. Secondly, the nodal velocity is propagated throughout the domain by solving $\nabla\phi \cdot \nabla v_n = 0$ with a fast marching method [26, 29].

Algorithm 2.6 summarizes the operations of this phase.

Algorithm 2.6 The `VelocityModule` algorithm.

Input: the nodal values of $\bar{\phi}$; the elastic strain field; and parameters l_c, h, κ, and η
Output: the nodal velocity v_n

```
1: Gather ( K, L, f^Y, f^Yc )    /* Gather matrices and right-hand side vector for (2.13) */
2: Ȳ, Ȳ_c ← solver ((2.13) )
3: Scatter ( Ȳ, Ȳ_c )                          /* Scatter Ȳ and Ȳ_c over the processes */
4: for all node i ∈ N_0 do
5: |  v_ni = (1/η) ⟨ Ȳ_i / Ȳ_ci − 1 ⟩_+
6: end for
7: Gather ( v_n )                                /* Gather v_n at nodes of the front */
8: extendVelocity ( )
9: Scatter ( v_n )                              /* Scatter v_n all over the processes */
```

2.4. Secant unloading scheme. If one wants to let the crack grow under the condition that $\bar{Y}$ cannot exceed $\bar{Y}_c$ instead of with (2.18), that is possible with an alternative loading scheme under the assumption of secant unloading [3, 16, 30]. This alternative involves solving a unit load analysis in the equilibrium solution and determination of a load scale factor γ after the $\bar{Y}$ corresponding to unit loading is evaluated.

With this approach, there is no adaptive time increment procedure. The velocity is defined such that the maximum is always equal to v_{lim}. Hence, the first loop in Algorithm 2.1 is skipped, and the second loop is passed with $\Delta t = 1\,\text{s}$ (cf. (2.4)). The nucleation check should be performed with the actual load level. The ratio Y/Y_c^0 is therefore scaled by γ^2 (see (2.19)) in the tenth and seventh lines of Algorithms 2.2 and 2.4, respectively. For simulation without plasticity, Y_c^0 is defined as $Y_c^0 = \frac{f_t^2}{2E}$, where f_t and E are, respectively, the tensile strength and Young's modulus.

Because unit load analysis only makes sense under secant unloading, this loading scheme cannot be used in combination with plasticity in the undamaged part of the material. Therefore, in Algorithm 2.5, the operations in lines three and four are skipped for this loading scheme.

The pseudocode in Algorithm 2.7 illustrates the front evolution phase with this loading scheme. The strain field and, consequently, the averaged configurational forces $\bar{Y}$ are evaluated with unit load boundary conditions. The actual load level for a time step is then determined by scaling the unit-load solution with a load scale factor γ such that the maximum scaled value for $\bar{Y}$ along the front is equal to $\bar{Y}_c$:

$$\gamma^2 \max_{i \in \mathcal{N}_0} \left\{ \frac{\bar{Y}_i}{\bar{Y}_{ci}} \right\} = 1. \tag{2.19}$$

Algorithm 2.7 The `VelocityModule` algorithm for secant unloading scheme.

Input: the nodal values of $\bar{\phi}$; the elastic strain field; and parameters l_c, h, c, k, and $v_{\max}$
Output: the nodal velocity v_n

```
Gather (K, L, f^Y, f^Yc)        /* Gather matrices and right-hand side vector for (2.13) */
Ȳ, Ȳ_c ← solver ((2.13) )
Scatter (Ȳ, Ȳ_c)                /* Scatter Ȳ and Ȳ_c over the processes */
max ← 0                          /* Initialize the highest ratio Ȳ/Ȳ_c */
for all node i ∈ N_0 do
    ratio ← Ȳ_i / Ȳ_ci
    if ratio > max then
        max ← ratio
    end if
end for
Allreduce (max)                  /* Get the highest ratio Ȳ/Ȳ_c from all the processes */
γ = sqrt(1/max)                  /* Compute the load scale factor */
for all node i ∈ N_0 do
    v_ni = k⟨c γ^2 Ȳ_i / Ȳ_ci − 1⟩_+
end for
Gather (v_n)                     /* Gather v_n at nodes of the front */
extendVelocity ()
Scatter (v_n)                    /* Scatter v_n all over the processes */
```

Finally, the front velocity v_n for every node in $\mathcal{N}_0$ is obtained through [30]

$$v_n = k\left\langle \frac{c\gamma^2 \bar{Y}}{\bar{Y}_c} - 1 \right\rangle_+ \quad \text{with} \quad k = \frac{v_{\lim}}{c-1}, \tag{2.20}$$

where $v_{\lim}$ is the maximum growth the front can experience for a time step. In order to guarantee the numerical stability of the staggered scheme, a value $v_{\lim} = \alpha_n h$ (cf. (2.5)) is used in this paper, following [30]. The parameter c influences the spread of the front movement to nodes with lower values for the ratio $\bar{Y}/\bar{Y}_c$. For $c \to 1$, only the node with the highest value $\bar{Y}/\bar{Y}_c$ undergoes a front advance. On the other hand, for higher values of c, nonzero front movement is found in more nodes.

3. Parallel version of the TLS method. This section deals with the modifications to the sequential algorithm presented above that are needed to parallelize it. This parallelization consists of concurrent *tasks* or *processes* being executed on distinct processors, in which each task encapsulates a copy of the sequential framework. Tasks are not completely independent; as a result, they need to interact by exchanging data (i.e., sending and receiving data) or *messages*. For the communication between processors, the message passing interface (MPI) [19] communication protocol is used.

3.1. DD. The DD strategy described by Lingen et al. [14] is used in this work. In this strategy, the problem is divided into subproblems that are solved almost independently. Therefore, the original finite element mesh is first partitioned into nonoverlapping groups of elements, each one corresponding to one subdomain as exemplified in Figure 4. A task is defined for each subdomain. Both tasks and subdomains are identified by an integer number called *rank* that ranges from 1 to n_p. The n_p subdomains are then optionally extended with overlapping regions by assigning extra nodes (and consequently elements) from neighboring subdomains.

The original nodes and elements on one subdomain are called *internal nodes* and *elements*, whereas the additional ones are called *overlapping nodes* and *elements* (see Figure 4). Observe that some nodes and elements on the overlapping region are also

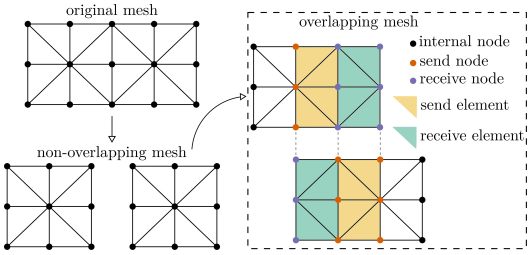

FIG. 4. *An original finite element mesh decomposed into two subdomains. The dashed box on the right indicates the mesh overlapping option.*

internal quantities; they are referred to as *send nodes* and *elements*. The counterparts of send nodes and elements are named *receive nodes* and *elements*. Although overlapping regions are not necessary in the DD strategy [21], it has been shown that this overlap can improve the convergence rate of parallel iterative algorithms [6].

Nodes and elements are indexed locally. However, the exchange of messages among subdomains is based on unique global indices (IDs), which are assigned during the partitioning of the original mesh [13, 21].

3.2. `Allreduce`, `Gather`, and `Scatter` routines. Modifications to achieve a desirable parallelism for the level set update and front evolution analysis phases require communication among tasks. Unlike the equilibrium solution phase, where the majority of communications are point-to-point (i.e., send and receive communication routines) between neighboring subdomains, collective communication routines in the sense that they involve participation of all subdomains or a group of them are proposed for the level set update and front evolution analysis phases. The main idea behind setting up these communication routines is to be minimally intrusive to the operations in Algorithms 2.4, 2.6, and 2.7.

Three general collective routines are used: `Allreduce`, `Gather`, and `Scatter`.[3] `Allreduce` is an all-to-all collective routine that is used to compute the maximum value of a quantity from all processes and distributes the result back, as schematically illustrated in Figure 5. `Gather` and `Scatter` are, respectively, designed to collect and spread messages involving a single receiving or originating process named the *root*. `Gather` is an all-to-one (or a "some-to-one") collective function in which each process (root process included) sends the contents of their send buffer; the root process receives the messages and stores them in rank order. `Scatter` , on the other hand, is a one-to-all (or a "one-to-some") collective routine used by the root to send a message, possibly with different sizes, to all processes or a group of them.

Note that `Gather` and `Scatter` are well suited for having operations executed by the root that are mostly sequential in nature and/or have the necessity of using global solution strategies. Depending on the nature of the operation executed by the root, all processes or a subset of processes participate in the `Gather` and `Scatter` routines, as illustrated in Figure 6. In all algorithms, operations that are positioned in between a `Gather-Scatter` pair are only executed by the root, and the inputs and outputs necessary for this operation are exchanged between the root and other processes by the `Gather` and `Scatter` routines.

[3]They have similar functionality as `MPI_Allreduce`, `MPI_Gather`, and `MPI_Scatter` functions found in MPI. However, `Allreduce`, `Gather` , and `Scatter` might operate on nondefault MPI data types. Moreover, `Gather` and `Scatter` might involve only a small group of processes.

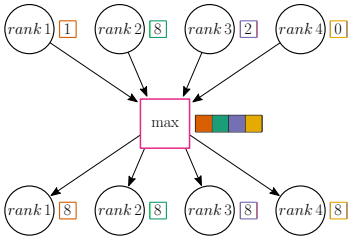

FIG. 5. *The collective routine* `Allreduce` *is used to compute the global maximum value of a quantity from all the local values* $\{1, 8, 2, 0\}$ *allocated on different tasks and distribute the result back to all the tasks. Arrows and colored blocks indicate, respectively, the data flow and content.*

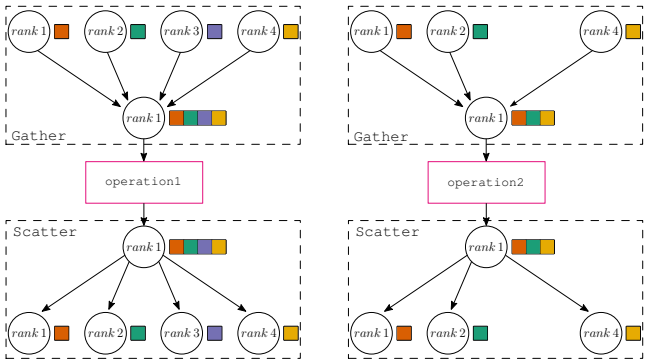

FIG. 6. *The root executes functions* `operation1` *and* `operation2` *along with a* `Gather-Scatter` *execution block. Rank* 1 *is considered as the root process. In this illustration, all processes contribute to both communications for* `operation1`*, whereas rank* 3 *is the only process that does not participate in the communication routines for* `operation2`*. Arrows and colored blocks indicate, respectively, the data flow and content.*

The two fast marching algorithms used for the reinitialization of the level set field ϕ (i.e., the `reinitializeLS` function in Algorithm 2.4) and for the extension of front velocity v_n (i.e., the `extendVelocity` functions in Algorithms 2.6 and 2.7) are performed in combination with the `Gather` and `Scatter` routines, mainly because they use a global sorting of the nodes based on their level set values for determining the order in which the ϕ and v_n are updated [29].

Furthermore, `Gather` and `Scatter` routines are used for the solution of (2.13) for the averaged quantities $\bar{Y}$, $\bar{Y}_c$, and $\bar{\phi}$ (i.e., the `solver` functions in Algorithms 2.4, 2.6, and 2.7). When solving (2.13) concurrently, the imposed constraints (i.e., $\bar{Y}$, $\bar{Y}_c$, and $\bar{\phi}$ are constant in ϕ direction and $\bar{\phi}$ also in s direction) for these averaged quantities will not be satisfied in a global sense. Note that these quantities are only computed on the damaged domain Ω^{d}. In many cases, not all subdomains will contain damage, which is why some-to-one and one-to-some versions of the `Gather` and `Scatter` routines are relevant.

For the execution of these general `Gather` and `Scatter` communication routines, a data structure management, slightly different from what is already used for the parallel iterative solver, is adopted in which the root has information on the original mesh and each subdomain keeps an operator $\mathbf{R}$ to extract elements from a root vector as $\mathbf{a}_i = \mathbf{R}_i\mathbf{a}$, where $\mathbf{a}_i$ is a vector containing the elements from a root vector $\mathbf{a}$ associated with the ith subdomain. The root also keeps a collection of operators

$\mathbf{Q}_i$ associated with each subdomain for assembly of root vectors as $\mathbf{a} = \sum_{i=1}^{n_p} \mathbf{Q}_i^{\mathrm{T}} \mathbf{a}_i$. Note that n_p does not necessarily include all processes. The operators $\mathbf{R}$ and $\mathbf{Q}$ are nonsquare Boolean matrices and have similar structures as the right and left restriction operators defined in the parallel iterative solver proposed in [14].

3.3. Level set update. The additional modifications related to the level set update phase are addressed following the order of the operations shown in Algorithm 2.4. Firstly, `updateLevelSet` (see Algorithm 2.1) is executed. Evaluation of the time increment size with (2.5) needs the largest value of velocity v_n over the whole mesh. After executing the first loop in parallel, the function `Allreduce` is therefore called, as shown in Algorithm 2.1. `Allreduce` computes the maximum value of $\max\{v_n\}$ from all tasks and distributes the result back to all of them. For those subdomains without damage front, their contributions to this operation are null.

Next, the fast marching algorithm for reinitialization of ϕ is executed by the root only. The `reinitializeLS` function is therefore sandwiched between a `Gather-Scatter` pair, where the root first receives the ϕ updated by `updateLevelSet` from each process and sends back the reinitialized values. All processes are involved, similar to `operation1` in Figure 6.

After the reinitialization, `testInitiation` is called in order to concurrently perform the nucleation check using `Allreduce` to compute the global maximum ratio Y/Y_c^0. For this particular operation, `Allreduce` is designed such that it also returns the corresponding coordinates $\mathbf{x}_{\mathrm{nucl}}$ related to the maximum value because `makeNucleus` needs $\mathbf{x}_{\mathrm{nucl}}$ in all processes to update ϕ concurrently.

Finally, the averaged quantity $\bar{\phi}$ is computed by the root. Again, the function `solver` is positioned in between the `Gather-Scatter` pair. The matrices and right-hand side vector necessary for solving (2.13) for $\bar{\phi}$ are concurrently partially assembled but only by processes belonging to $\mathcal{P}_{\mathrm{d}}$ (the set of processes that possess any damage front). The root process gathers these partial quantities from the processes in $\mathcal{P}_{\mathrm{d}}$ by means of a `Gather` routine in order to assemble the global system of equations. Once the solution for $\bar{\phi}$ is obtained, the root sends it back to the same set of processes through a `Scatter` routine.

Unlike the `Gather` routine used in the reinitialization operation in which a one-way communication is performed, i.e., the data flows from senders to the root with nothing going back in return, the `Gather` routine considered in the assembly procedure of (2.13) encompasses three stages of data communication. First, the root queries processes that have a damage front (i.e., processes that form the set $\mathcal{P}_{\mathrm{d}}$) for global IDs of the nodes belonging to the set of elements completely or partially inside the damaged domain Ω^{d}. Once the root receives these nodal IDs, the root numbers DOFs at these nodes and sends them back along with the size of the final system of equations to the processes belonging to $\mathcal{P}_{\mathrm{d}}$ in rank order. Then, each process makes use of this information to assemble its own matrix and right-hand side vector and sends them back to the root. The root then assembles the final system of equations by summing these contributions, as depicted in Figure 7.

For the sake of consistency and efficiency in terms of message communication, two points of extra attention exist in the aforementioned assembly procedure. First, to avoid duplicated contributions from the elements in the overlapping region (see Figure 8), receive elements are excluded from the assembly procedure. Second, due to the sparse structure of the system of equations, there may be many zero entries which can unnecessarily overload these `Gather` and `Scatter` calls. To make matters worse, the expanded system of equations assembled on a single subdomain may have

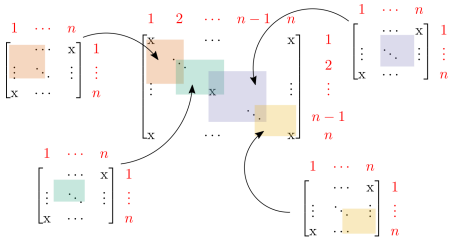

FIG. 7. *Assembly of the global matrix, belonging to $\mathcal{P}_d$, used by the root to solve the final system of equations. All the small matrices, each belonging to a single process, have the same indexing as the global $n \times n$ matrix.*

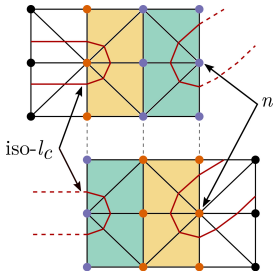

FIG. 8. *The node n is shared by two subdomains and can automatically be identified when the iso-l_c crosses both subdomains. Dashed lines represent the part of the iso-l_c that belongs to the neighboring subdomain.*

just a few nonzero elements. Therefore, a sparse data storage structure is necessary. For this purpose, a compressed row storage format is adopted [2, 10], which stores $2nnz+n+1$ elements instead of n^2 for an $n \times n$ matrix with nnz nonzero entries. Note that each quantity assembled concurrently by a single process belonging to $\mathcal{P}_{\mathrm{d}}$ uses the same global indexing from the root. This design allows for efficient summation of the global sparse matrix in the root process from a number of sparse matrices from the other processes.

3.4. Equilibrium solution. For solving the linear system of equations from the equilibrium problem, the parallel iterative generalized minimum residual (GMRES) solver proposed by Lingen et al. [14] is used. This solver is equipped with a two-level preconditioner that consists of a restricted additive Schwarz preconditioner that acts on the level of subdomains and an algebraic coarse grid preconditioner that operates on the global level. The restricted addictive Schwartz preconditioner is based on an incomplete Cholesky decomposition, while the coarse grid preconditioner is constructed in terms of the rigid body modes of the subdomains.

The interaction communication associated to this solver is mostly one-to-one, i.e., involving only the shared region between adjacent subdomains, except when global reduction operations such as a global sum necessary for computing a scalar product. The messages involved in this interaction consist of a vector with a given length and a data type (e.g., integer or double). The length of this vector is usually proportional to the number of DOFs attached to the nodes (i.e., send and receive nodes) on the shared regions.

3.4.1. Parallel enrichment scheme. It is important to recall that as the iso-l_c evolves, DOFs are added to and/or removed from the problem, which changes the

dimensions of stiffness matrices and force vectors. If one overlapping node is enriched or unenriched on a subdomain, its counterparts, i.e., the nodes with the same global ID on different subdomains, also have to be enriched or unenriched accordingly. This is necessary since the messages exchanged by the routines of the solver along the boundaries of neighboring subdomains need to be equal in size.

Hence, after performing its own enrichment procedure, each process has to exchange data with its neighbors about the enrichment status of its own overlapping nodes. Note that this communication is only necessary when a crack is propagating across subdomain boundaries. In this communication, the message is a vector with entries of a special data type, which encodes three types of information (or a *triplet*): the global ID of an overlapping node, its enrichment flag (whether or not it is enriched), and element support (positive and negative lists of global element IDs).

The send nodes control this process because they are interior nodes and, hence, always have sufficient information to decide whether the node should be enriched and because every node in the overlapping region is always a send node in no more and no less than one subdomain. Thus, when DOFs attached to one send node are updated, receive nodes on the neighboring subdomains will follow, as shown in Figure 8. Note that this is only possible as long as the continuity of ϕ is guaranteed across the boundaries of subdomains, which ensures that all subdomains involved in this communication process share the same geometric location of the iso-l_c on their overlapping regions. Consequently, these subdomains are able to determine the exact length of the message for both send and receive packets of data.

The parallel enrichment scheme involves two stages. Firstly, each process p executes the function `initEnrichment` (see Algorithms 2.5 and 3.1) for all its nodes with level set value $\phi > l_c$, except for those that are receive nodes. During the execution of `initEnrichment`, the send and receive nodes are collected, and their triplets (i.e., global ID, enrichment flag, and element support) are stored in two special data structures sendBuffer and recvBuffer, respectively. Both sendBuffer and recvBuffer store the exact amount of data for each neighbor of process p. At this point, the triplet of each node that is stored in recvBuffer only contains the global node IDs, whereas the triplet of each node that is stored in sendBuffer may be complete. If a send node is unenriched, the element support associated with this node is not stored, and hence, the enrichment flag in the triplet related to this node becomes false.

Secondly, the exchange of information between processes is handled. The procedure for sending and receiving sendBuffer and recvBuffer on one process p is executed only if send and/or receive nodes have been collected. First, once data are stored in recvBuffer, process p loops over all its neighbors i calling the function `InitReceive`,[4] which initiates a nonblocking receive communication [19]. This function returns a handle (or request recvreq) that can be used at a later time to check whether the message has been received. Note that if p does not have messages to be received from one specific neighbor i, the size of recvBuffer (i.e., recvBuffer[i] .size( ) ≤ 0) is checked and the call of `InitReceive` is then skipped. Because `InitReceive` does not block the calling process, messages can be called in any order without risking "deadlock" issues. Next, as long as sendBuffer contains any data, the send procedure is executed in a similar way to the receive one. After these two receive and send loops, the func-

[4] `InitReceive` and `InitSend` make use of nonblocking functions from MPI, such as `MPI_Irecv` and `MPI_Isend`, as well as `MPI_Waitall` (which is represented here as `WaitAll`) for the completion of communication. A special MPI data type is also designed in order to deal with recvBuffer and sendBuffer data structures.

Algorithm 3.1 The `commEnrich` algorithm for communication of enrichment.

```
Input: the enrichment status of send nodes
Output: the enrichment update of receive nodes
if processor p has collected receive nodes then                    /* Receive data */
    for all neighbor i of process p do
        if recvBuffer[i] .size( ) ≤ 0 then
            continue                          /* Skip neighbors without message to be sent */
        end if
        InitReceive ( recvBuffer[i], recvreq[i], i )
    end for
end if
if processor p has collected send nodes then                       /* Send data */
    for all neighbor i of process p do
        if sendBuffer[i] .size( ) ≤ 0 then
            continue                      /* Skip neighbors without message to be received */
        end if
        InitSend ( sendBuffer[i], sendreq[i], i )
    end for
end if
if process p has collected receive nodes then                  /* End receive procedure */
    WaitAll ( recvreq )
    setLEnrich( recvBuffer )                  /* Update enrichment for receive nodes */
end if
if process p has collected send nodes then                     /* End send procedure */
    WaitAll ( sendreq )
end if
```

tions `WaitAll` are called in order to complete the multiple receive and send requests. Finally, once p has received messages from all its neighbors, it updates the enrichment status for receive nodes by means of the function `setLEnrich`.

Note that `InitReceive` is called as early as possible to increase the chance that a matching call `InitSend` by another process can be completed immediately. This strategy helps to lower communication overhead because each message that cannot be moved directly to a receiver buffer must be temporarily stored in a pending queue [19]. This communication pattern is similar to what was proposed in [21] to deal with enriched nodes in an extended finite element context for hydraulic fracturing in elastic materials.

3.4.2. Mapping operators. Regarding the SPR technique for transferring history, it is chosen to let each subdomain deal with its own execution of routine `initSPRHistory`, even though this means that incomplete patches are used for nodes at boundaries of subdomains. In order to be more consistent, an extra communication strategy would be necessary for those patches to be completely assembled. However, this will not have a significant effect on the global response.

Another option would be the inverse distance weighted interpolation scheme in which all the transferring of history takes place locally on the element level without iteration with its vicinity. However, it was shown in [18] that the SPR technique is more suitable, especially for coarse meshes.

3.5. Front evolution. The modifications of the front evolution analysis phase follow those introduced for the parallel version of level set update phase. Two `Gather-Scatter` execution blocks are introduced, as shown in Algorithm 2.6. For the solution of (2.13) for the averaged forces $\bar{Y}$ and $\bar{Y}_c$, the same strategy is adopted as already discussed for $\bar{\phi}$. For the fast marching algorithm, `extendVelocity`, the strategy is the same as for the reinitialization of the level set field.

3.6. Secant unloading scheme. The main difference between the two loading schemes considered in this work is found in the front evolution phase, as outlined in subsection 2.4. To parallelize the scheme with secant unloading (see Algorithm 2.7),

one additional `Allreduce` call is introduced in order to compute the maximum value of the load scale factor, which is needed for the nucleation check by all subdomains, as shown in Algorithm 2.4.

4. Results and discussion. The performance of the parallel versions of the TLS model is investigated with numerical examples in this section. The models have been developed within the parallel open source Jem/Jive toolkit [10], which provides graph-based partitioning algorithms for decomposition of the original mesh [21] and the parallel GMRES solver [13]. The simulations have been run on the numerical mechanics cluster HPC27, which is a regular high performance computing master-slave system at Delft University of Technology. Each node is equipped with two Intel Xeon E5-2630 version 4 processors, having 10 cores each, and 128 GB memory.

For all numerical examples, unstructured meshes of linear triangles generated with Gmsh [8] are considered. For nucleation, the size of a new damage nucleus ϕ_0 is about the effective element size h. Regarding the stabilization parameter, two values of κ are used: $\kappa = 1$ for $\bar{Y}$ ((2.13)) and $\kappa = 1 \cdot 10^4$ for $\bar{\phi}$. All presented results are obtained under plane strain assumptions.

4.1. Doubly notched square plate (DNSP). As a first example, the response of a DNSP [16] is simulated. The material is modeled as elastic, allowing the use of the secant unloading scheme. To investigate the scalability of the parallel approach, the same analysis is performed with different numbers of subdomains, each time using as many cores as there are subdomains. For each number of subdomains, the problem is run three times, and the average runtime is computed.

Boundary conditions and the geometry of the specimen are shown in Figure 9. Young's modulus, Poisson's ratio, fracture energy, and tensile strength are, respectively, $E = 7000\,\text{MPa}$, $\nu = 0.3$, $G_c = 40\,\text{N/mm}$, and $f_t = 79\,\text{MPa}$. The critical length l_c is equal to 0.8 mm. This example is performed with $c = 2$, $\alpha_n = 0.5$, $\bar{\phi}_{\text{init}} = 0$, and $\bar{\phi}_{\max} = l_c/3$. A region around the notches with refined mesh is defined, where the effective element size $h = 0.1\,\text{mm}$. Away from this region, the element size is 0.5 mm, leading to a mesh of 151370 elements and 76136 nodes, i.e., 152272 DOFs. The nucleation check is only performed on elements in the fine mesh region.

Before the number of subdomains is varied, the influence of using an overlapping region on the runtime is investigated. The analysis with 20 subdomains is performed with overlap region ranging from zero to six elements wide. Table 1 shows the total runtime for different layers of elements in the overlapping region. It is found that an overlapping region with one layer of elements is optimal for this example.

The load-displacement curve for a reference solution, obtained without parallelism, is compared with the result of using 20 subdomains in Figure 10. Even in the

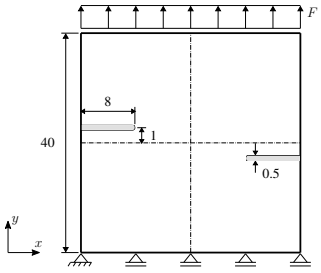

FIG. 9. *DNSP: boundary condition and geometry (dimensions in mm).*

TABLE 1
Total runtime for different sizes of the overlap region.

Overlap layers	**Time** [s]
0	595.21
1	526.68
2	555.31
4	584.70
6	628.79

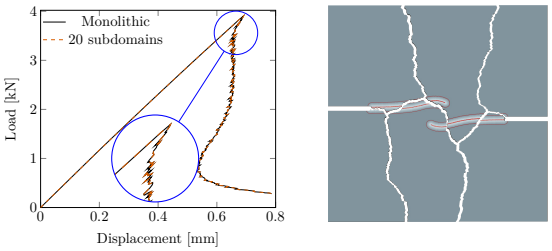

FIG. 10. *DNSP: load-displacement curve (right) and final crack distribution in the mesh partitioned into six subdomains (left).*

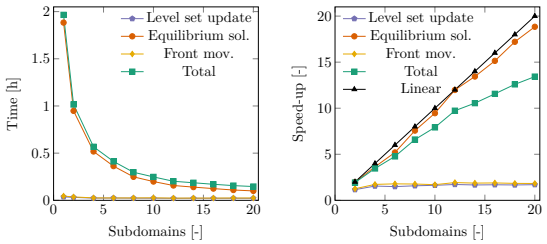

FIG. 11. *DNSP: wall clock time (left) and speedup (right) graphs.*

oscillatory postpeak part, the results with 20 cores are exactly the same as those from a single core. By using 20 cores, the parallel framework accelerates the sequential approach by a factor of 13.4 without loss of accuracy. Figure 10 shows for the case of six subdomains that the continuity of the level set field is ensured even when the crack crosses the subdomain boundaries.

Figure 11 shows the total runtime as well as the total time spent for each analysis phase (level set update, equilibrium solution, and front evolution) as a function of the number of subdomains. As expected, the equilibrium solution phase is the most time consuming. The total time spent on this phase scales very well as the number of subdomains increases with close to the optimum of linear scaling.

Unlike the equilibrium solution phase, the level set update and front evolution phases are barely accelerated since their main operations rely on collective communication patterns, yielding a low level of parallelism. Going from one to four subdomains, a speedup of about a factor of two is obtained, but further increasing the number of subdomains does not lead to significant changes in the total time needed for these phases. However, the total time needed for these phases without parallelism is much less than for the equilibrium solution phase. Therefore, the overall scaling is still very favorable. Nevertheless, the fact that part of the framework is not scaling optimally means that the scaling in total runtime deviates increasingly from optimal linear scaling.

One possible option to achieve better scaling for the level-set-related operations in the level set update and front evolution phases would be to use the weak form, and its discretization, to solve the reinitialization and velocity extension problems following Adams, Giani, and Coombs [1] and Dekker et al. [5]. In this approach, we have systems of equations for both problems which can also be solved by the parallel iterative solver. Matrices and right-hand side vectors have assembly procedures similar to standard finite elements. Thus, we could have a better parallelism level, i.e., a more one-to-one communication strategy, instead of a global-like solution strategy, which involves global reduction communication patterns as presented.

4.2. Single-notched shear test (SNST). In the second example, the response of a single-edge notched plate considering plasticity is simulated. Once again, the scalability and accuracy of the parallel framework are assessed by means of load-displacement and speedup curves considering different numbers of subdomains and cores for the same analysis.

Figure 12 shows the boundary conditions and geometry of the example. A horizontal displacement is applied to the top half of the left edge. The material is modeled with the pressure-dependent plasticity model for polymers by Melro et al. [15] as revised by van der Meer [28]. This plasticity model makes use of a paraboloidal yield surface that takes into account different compressive and tensile yield stresses. Young's modulus, Poisson's ratio, and fracture energy are, respectively, $E = 3760\,\mathrm{MPa}$, $\nu = 0.3$, and $G_c = 0.9\,\mathrm{N/mm}$ [15, 28]. The other properties of the material, such as hardening curves, plastic Poisson ratio, and the ultimate yield stress values for the nucleation check, are the same as in [28]. The critical length l_c is equal to 2 mm. This example is performed with $\alpha_n = 0.5$, $\bar{\phi}_{\mathrm{init}} = 0$, and $\bar{\phi}_{\mathrm{max}} = l_c/3$. The whole geometry is meshed with $h = 0.4\,\mathrm{mm}$, leading to 157600 elements and 79419 nodes or 158838 DOFs. The nucleation check is only performed on a region near the notch tip. The viscous parameter against crack growth and displacement rate are, respectively, $\eta = 5\,\mathrm{s\,mm^{-1}}$ and $\dot{u} = \Delta u^0/\Delta t^0 = 0.05\,\mathrm{mm\,s^{-1}}$.

Figure 13 depicts the comparison between the reference solution obtained with the sequential framework and the result of using 20 subdomains. Again, there is no loss of accuracy in the sense that both responses are in excellent agreement. The level set continuity across the subdomain boundaries is also preserved, as illustrated in Figure 13 for the case of 20 subdomains. With 20 subdomains, the parallel framework accelerates the sequential approach by a factor of 13.2, as shown in Figure 14.

The total runtime, as well as the total time spent for each analysis phase, and their corresponding speedups are given in Figure 14. The same trends presented for the previous example with the secant unloading scheme are also observed for this example

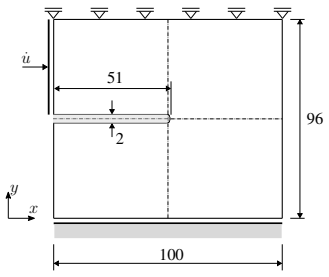

FIG. 12. *SNST: boundary condition and geometry (dimensions in mm).*

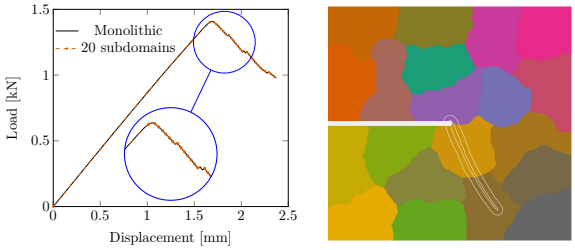

FIG. 13. *SNST: load-displacement curve (right) and final crack distribution in the mesh partitioned into its* 20 *subdomains (left). Shading indicates the subdivision into its* 20 *subdomains.*

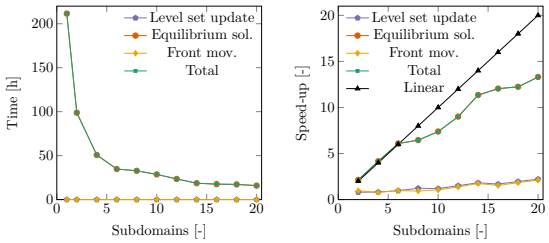

FIG. 14. *SNST: wall clock time (left) and speedup (right) graphs.*

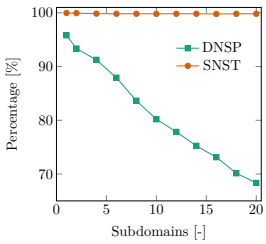

FIG. 15. *Percentage of the equilibrium solution phase to the total runtime for the first two examples.*

with plasticity. The equilibrium solution phase is the most time consuming and presents the best scalability among the three analysis phases, although the scalability of the equilibrium solution phase is not as good as in the previous example. The level set update and front evolution phases, again, do not scale as well as the equilibrium solution phase, but for this example they have an even smaller contribution to the total runtime.

Figure 15 shows the percentage of the equilibrium solution phase to the total runtime for the two cases discussed so far. Despite having slightly different mesh sizes, it is noteworthy that the equilibrium solution phase is even more dominant for the plasticity case, reinforcing the importance of a good parallel strategy for that particular phase. However, when comparing speedup of the equilibrium solution phase to that of the simulation without plasticity (Figure 14 versus Figure 11), it can be observed that the additional nonlinearity reduces the scalability of the parallel iterative solver. As a result, overall speedups are similar for the two cases.

4.3. Three-point bend end-notched flexure (3ENF) test configuration. The final example is chosen as a case where computation time without parallel

approach would become prohibitively long. The case is inspired by experimental observations of cusp crack patterns taking place in resin-rich regions of composite materials in mode II loading conditions [31, 4]. This process begins with an array of inclined cracks perpendicular to the maximum principal stress, which eventually merge to form a single crack on a higher level of observation. Therefore, this example requires the proposed parallel framework to deal with several damage nuclei arising on various subdomains that grow and join up in merging and branching events. Moreover, in realistic simulation of this process, nucleation and growth of the crack should take place in a medium with hardening plasticity. The 3ENF setup is adopted (see Figure 16). Because the cusp formation takes place in a narrow area close to the notch tip, a very fine discretization is needed near in this area, which gives rise to a large system of equations that would be prohibitive to be solved with a direct solver or with a single computer core.

The geometry consists of two stiff arms and one weak core as schematically illustrated in Figure 16. The round cross-section is inspired by the rail shear test in [23]. Note that the top arm is also supported in y direction in order to avoid interpenetration without having to model contact between the arms. The two faces are considered as linear elastic materials whose properties are $E = 200\,\mathrm{GPa}$, $\nu = 0.33$, $G_c = 9\,\mathrm{N/mm}$, and $f_t = 960\,\mathrm{MPa}$. The core is modeled with the same plasticity model and material properties mentioned in the second example.

When the crack reaches the interface between the core and faces, the crack cannot grow in pure mode I, and the constitutive law in (2.6) leads to stress transfer across the crack. As a result, an artificial hardening is found, as reported by van der Meer and Sluys [30]. In order to circumvent this undesirable behavior, the interphase constitutive law introduced in [30] is adopted in a band next to the material interface as indicated in Figure 16. This constitutive law only takes into account stiffness recovery on the strain component normal to the plane that defines the interface, which, in this case, is the strain component in y direction.

The region where the nucleation check is performed is indicated in Figure 16. The smallest element size h is 0.1 mm for the nucleation check region. For the other region of the core and faces, the mesh has an element size of 0.15 mm and 0.45 mm, respectively. The mesh therefore has 229919 elements, 115527 nodes, and, initially, 231054 DOFs. The critical length l_c is equal to 0.6 mm. The distance between a new damage nucleus and existing damage front is set to be $\phi_{\mathrm{spacing}} = 4\,\mathrm{mm}$. The viscous parameter against crack growth and displacement rate are, respectively, $\eta = 5\,\mathrm{s\,mm^{-1}}$ and $\dot{u} = \Delta u^0/\Delta t^0 = 0.01\,\mathrm{mm\,s^{-1}}$. This simulation is performed with $\alpha_n = 0.2$, $\bar{\phi}_{\mathrm{init}} = 0.1$, and $\bar{\phi}_{\mathrm{max}} = 0.36$.

Figure 17 shows the load-displacement curves for different numbers of subdomains. The original mesh is divided into 20, 30, and 40 subdomains. The whole analysis takes 74.35 h by using 40 subdomains, whereas the models subdivided into 30 and 20 take 86.54 h and 118.94 h, respectively. Observe that all solutions are in excellent agreement.

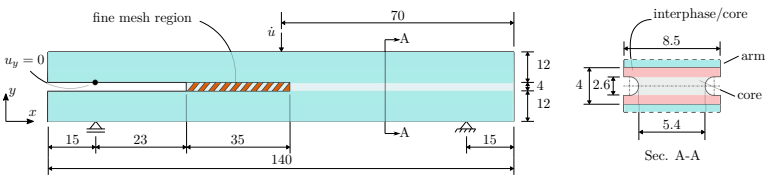

FIG. 16. *3ENF: boundary condition and geometry (dimensions in mm).*

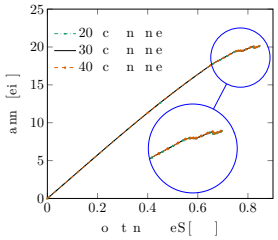

FIG. 17. 3*ENF: load-displacement curve.*

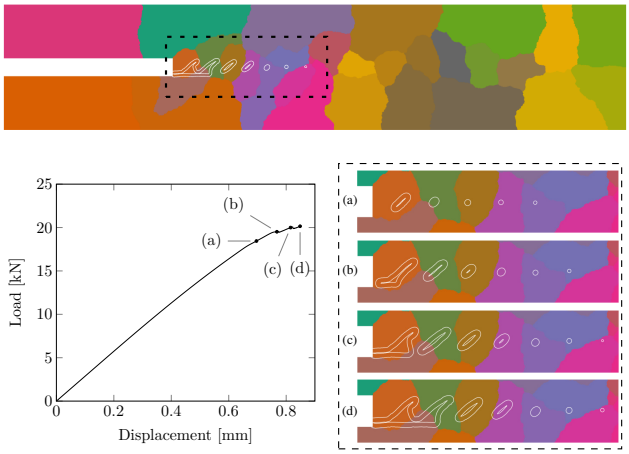

FIG. 18. 3*ENF: final crack distribution, load-displacement graph, and crack evolution (close-up) located ahead of the notch tip considering the mesh partitioned into* 30 *subdomains. The colored set of elements represents the* 30 *subdomains.*

Figure 18 presents the damage distribution for the model subdivided into 30 subdomains. Observe that damage initiation takes place in different subdomains and some initial fronts evolve crossing multiple subdomains where the compatibility of ϕ is guaranteed on shared regions.

5. Conclusions. In this paper, a parallel framework is proposed for the TLS method. Two TLS models have been adopted: one considering the secant unloading loading scheme [30] and the other one for ductile fractures [18]. The parallel iterative solver by Lingen et al. [14] equipped with a DD scheme is used for the equilibrium solution stage. Profiting from the adopted DD scheme, collective communication strategies have been introduced in order to deal with the level set information and the computation of averaged quantities. Moreover, a special data type and communication pattern have been developed to handle enriched nodes belonging to shared regions in the mesh.

In three examples, a successful parallel implementation has been shown. The quality of the results is not influenced by the number of subdomains. The results show that it is possible to apply the parallel framework to different variations of the TLS to benefit from parallel computing power.

Near-ideal speedups are obtained for the equilibrium solution phase, which is the most time demanding part of the TLS in terms of computational cost. For the level set update and front evolution phases, speedups remained limited. However,

because these phases are less demanding, they did not become a real bottleneck for the investigated number of cores. Adding plasticity makes the equilibrium solution phase more dominant, but the added nonlinearity also reduces the speedup of that particular phase such that overall speedups with and without plasticity are comparable. Altogether, substantial speedups have been achieved using a moderate amount of cores which decreased the total computational time significantly. This is a necessary improvement in order to use the TLS models for fracture analysis in large-scale problems, as exemplified with the simulation of shear cusps at the notch tip in mode II loading conditions.

Data availability. The data used to generate the graphs and the mesh partitions of the first two examples in this article are available at the 4TU.ResearchData repository through https://doi.org/10.4121/14869314.

REFERENCES

[1] T. Adams, S. Giani, and W. M. Coombs, *A high-order elliptic PDE based level set reinitialisation method using a discontinuous Galerkin discretisation*, J. Comput. Phys., 379 (2019), pp. 373–391.

[2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Society for Industrial and Applied Mathematics, Philadelphia, 1994.

[3] P. E. Bernard, N. Moës, and N. Chevaugeon, *Damage growth modeling using the thick level set (TLS) approach: Efficient discretization for quasi-static loadings*, Comput. Methods Appl. Mech. Engrg., 233-236 (2012), pp. 11–27.

[4] A. Biel and U. Stigh, *Strength and toughness in shear of constrained layers*, Int. J. Solids Struct., 138 (2018), pp. 50–63.

[5] R. Dekker, F. P. van der Meer, J. Maljaars, and L. J. Sluys, *A level set model for stress-dependent corrosion pit propagation*, Int. J. Numer. Methods Eng., 122 (2021), pp. 2057–2074.

[6] V. Dolean, P. Jolivet, and F. Nataf, *An Introduction to Domain Decomposition Methods*, Society for Industrial and Applied Mathematics, Philadelphia, 2015.

[7] C. Farhat and F.-X. Roux, *A method of finite element tearing and interconnecting and its parallel solution algorithm*, Int. J. Numer. Methods Eng., 32 (1991), pp. 1205–1227.

[8] C. Geuzaine and J.-F. Remacle, *Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities*, Int. J. Numer. Methods Eng., 79 (2009), pp. 1309–1331.

[9] P. Gosselet and C. Rey, *Non-overlapping domain decomposition methods in structural mechanics*, Arch. Comput. Methods Eng., 13 (2006), pp. 515–572.

[10] Jem-Jive, *Software Development Kit for Advanced Numerical Simulations*, https://software.dynaflow.com/jive/. Accessed November 16, 2021.

[11] M. Latifi, F. P. van der Meer, and L. J. Sluys, *An interface thick level set model for simulating delamination in composites*, Int. J. Numer. Methods Eng., 111 (2017), pp. 303–324.

[12] B. Lé, N. Moës, and G. Legrain, *Coupling damage and cohesive zone models with the thick level set approach to fracture*, Eng. Fract. Mech., 193 (2018), pp. 214–247.

[13] F. J. Lingen, *Design of an Object Oriented Finite Element Package for Parallel Computers*, Ph.D. thesis, Delft University of Technology, 2000.

[14] F. J. Lingen, P. G. Bonnier, R. B. J. Brinkgreve, M. B. V. Gijzen, and C. Vuik, *A parallel linear solver exploiting the physical properties of the underlying mechanical problem*, Comput. Geosci., 18 (2014), pp. 913–926.

[15] A. R. Melro, P. P. Camanho, F. M. A. Pires, and S. T. Pinho, *Micromechanical analysis of polymer composites reinforced by unidirectional fibres: Part* I - *constitutive modelling*, Int. J. Solids Struct., 50 (2013), pp. 1897–1905.

[16] N. Moës, C. Stolz, P.-E. Bernard, and N. Chevaugeon, *A level set based model for damage growth: The thick level set approach*, Int. J. Numer. Methods Eng., 86 (2011), pp. 358–380.

[17] K. Moreau, N. Moës, D. Picart, and L. Stainier, *Explicit dynamics with a non-local damage model using the thick level set approach*, Int. J. Numer. Methods Eng., 102 (2015), pp. 808–838.

[18] L. A. T. Mororó and F. P. van der Meer, *Combining the thick level set method with plasticity*, Eur. J. Mech. A Solids, 79 (2020), 103857.
[19] MPI Forum, *MPI, Version* 3.1, http://www.mpi-forum.org.
[20] E. A. S. Neto, D. Perić, and D. R. J. Owen, *Computational Methods for Plasticity: Theory and Applications*, 1st ed., John Wiley and Sons, West Sussex, United Kingdom, 2008.
[21] E. W. Remij, *Fluid Driven and Mechanically Induced Fracture Propagation: Theory and Numerical Simulations*, Ph.D. thesis, Eindhoven University of Technology, 2017.
[22] I. B. C. M. Rocha, F. P. van der Meer, L. A. T. Mororó, and L. J. Sluys, *Accelerating crack growth simulations through adaptive model order reduction*, Int. J. Numer. Methods Eng., 121 (2020), pp. 2147–2173.
[23] C. E. Rogers, *Investigating the Micromechanisms of Mode* II *Delamination in Composite Laminates*, Ph.D. thesis, Imperial College London, 2009.
[24] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia, 2003.
[25] A. Salzman, N. Moës, and N. Chevaugeon, *On use of the thick level set method in* 3*D quasi-static crack simulation of quasi-brittle material*, Int. J. Fract., 202 (2016), pp. 21–49.
[26] J. A. Sethian, *Level Set Methods and Fast Marching Methods: Envolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Material Science*, 2nd ed., Cambridge University Press, Cambridge, United Kingdom, 1999.
[27] P. L. Tallec, Y. H. D. Roeck, and M. Vidrascu, *Domain decomposition methods for large linearly elliptic three-dimensional problems*, J. Comput. Appl. Math., 34 (1991), pp. 93–117.
[28] F. P. van der Meer, *Micromechanical validation of a mesomodel for plasticity in composites*, Eur. J. Mech. A Solids, 60 (2016), pp. 58–69.
[29] F. P. van der Meer, N. Moës, and L. J. Sluys, *A level set model for delamination – Modeling crack growth without cohesive zone or stress singularity*, Eng. Fract. Mech., 79 (2012), pp. 191–212.
[30] F. P. van der Meer and L. J. Sluys, *The thick level set method: Sliding defomations and damage initiation*, Comput. Methods Appl. Mech. Engrg., 285 (2015), pp. 64–82.
[31] H. Wafai, A. Yudhanto, G. Lubineau, R. Yaldiz, and N. Verghese, *An experimental approach that assesses in-situ micro-scale damage mechanisms and fracture toughness in thermoplastic laminates under out-of-plane loading*, Compos. Struct., 207 (2019), pp. 546–559.
[32] O. C. Zienkiewicz and J. Z. Zhu, *The superconvergent patch recovery and a posteriori error estimates. Part* I: *The recovery technique*, Int. J. Numer. Methods Eng., 33 (1992), pp. 1331–1364.
[33] O. C. Zienkiewicz and J. Z. Zhu, *The superconvergent patch recovery (SPR) and adaptive finite element refinement*, Comput. Methods Appl. Mech. Engrg., 101 (1992), pp. 207–224.